1. HDK Help . . . 29
1.1 Recent Changes in the HDK Help Wiki . . . 29
1.2 Welcome . . . 30
1.2.1 What's New in this Release . . . 31
1.2.2 What's New in Lua . . . 33
1.2.3 What's New in the HDK API . . . 34
1.2.4 Known Issues . . . 34
1.2.5 Bug Fixes . . . 36
1.3 Brand Guidelines . . . 36
1.4 Getting Started . . . 37
1.4.1 HDK Overview . . . 38
1.4.1.1 What is the Home Development Kit? . . . 39
1.4.1.2 Developing with the HDK . . . 40
1.4.1.3 Art and Sound Tools . . . 41
1.4.1.4 Animation Overview . . . 42
1.4.1.5 Scripted Assets Overview . . . 42
1.4.1.6 The Editors . . . 45
1.4.1.7 Screens Overview . . . 46
1.4.1.8 Character Components and Furniture Creation . . . 47
1.4.1.9 Game Launching and Home Rewards . . . 48
1.4.1.10 Profiling Overview . . . 49
1.4.2 Installing the HDK . . . 49
1.4.2.1 Validating the Installation . . . 57
1.4.2.2 Content Folders . . . 57
1.4.2.2.1 Creating Content Folders . . . 58
1.4.2.2.2 Switching Content Folders . . . 60
1.4.2.2.3 Upgrading Content Folders . . . 61
1.4.2.3 Installing the Home Developer Package . . . 64
1.4.2.4 Configuring the Network . . . 64
1.4.2.5 Launching the Tools . . . 65
1.4.3 The Home Client . . . 66
1.4.3.1 Launching the Home Developer SELF . . . 66
1.4.3.2 Testing Content in Online Mode . . . 70
1.4.3.3 Reference Tool Dip Switches . . . 72
1.4.3.4 The DEV DEBUG Menu . . . 73
1.4.3.5 The Debug Console . . . 74
1.4.3.6 Taking Screenshots . . . 77
1.4.3.7 Debug Console Command Reference . . . 78
1.4.3.8 PlayStation®Home Error Codes . . . 82
1.5 Objects . . . 84
1.5.1 Active Items . . . 84
1.5.1.1 Adding Mini-game Components . . . 86
1.5.1.2 Adding Script Components . . . 89
1.5.2 Portable Objects . . . 94
1.5.3 Animation Packs . . . 94
1.5.4 Locomotion Objects . . . 94
1.5.4.1 Creating Locomotion Objects . . . 95
1.5.4.2 Configuring Locomotion Objects . . . 98
1.5.5 Resource Packs . . . 108
1.5.5.1 Creating a Resource Pack . . . 110
1.5.5.2 Managing Resource Packs in Script . . . 110
1.5.5.3 Resource Pack Memory Management . . . 111
1.5.5.4 Resource Pack Validation Guidelines . . . 111
1.5.6 Embed Objects into Scenes . . . 111
1.5.6.1 Creating a Scene Object . . . 113
1.5.6.2 Defining Instance Parameters . . . 115
1.5.7 Furniture and Decorations . . . 120
1.5.7.1 Picture Frames and Wall Decorations . . . 120
1.5.7.1.1 Picture Frames . . . 121
1.5.7.1.2 Creating Picture Frames in Maya . . . 123
1.5.7.1.3 Exporting Picture Frames . . . 128
1.5.7.1.4 Adding Picture Hooks . . . 131
1.5.7.1.5 Picture Frame Commerce and Regional Restrictions . . . 131
1.5.7.2 Creating a Seat . . . 132
1.5.7.3 Selecting Default Shaders for Furniture . . . 135
1.5.7.4 Menu Categories for Furniture . . . 136
1.5.7.5 Validating and Exporting Furniture . . . 137
1.5.7.6 Textures for Furniture . . . 141
1.5.7.7 The Furniture Shelf and Furniture Menu . . . 143
1.5.7.8 Rules and Conventions for Modeling Furniture . . . 144
1.5.7.9 Creating a Lamp . . . 147
1.5.8 Furniture Block System . . . 147
1.5.8.1 Determining Active Item Resource Usage . . . 148
1.5.8.2 Optimizing Actives Memory Usage . . . 152
1.5.9 Object Editor . . . 153
1.5.9.1 Object Creation Workflow . . . 153
1.5.9.2 Object Editor Interface . . . 155
1.5.9.2.1 Object Search Panel . . . 157
1.5.9.2.2 Object Summary Panel . . . 158

1.5.9.2.3 Object View Panel . . . 159
1.5.9.2.4 HDK Project Panel . . . 160
1.5.9.2.5 Output Panel . . . 161
1.5.9.2.6 Properties Panel . . . 161
1.5.9.2.7 Scripting Panel . . . 162
1.5.9.2.8 Viewer Panel . . . 162
1.5.9.3 Setting Object Editor Preferences . . . 164
1.5.9.4 Object Structure and Files . . . 166
1.5.9.5 Working with HDK Project Files . . . 168
1.5.9.5.1 Organizing Objects Using HDK Projects . . . 170
1.5.9.5.2 Working with Templates in Projects . . . 172
1.5.9.6 Creating a New Object . . . 172
1.5.9.7 Overwriting and Copying Objects . . . 175
1.5.9.8 Viewing and Searching for Objects . . . 176
1.5.9.9 Editing Multiple Objects . . . 180
1.5.9.10 Completing the Object Header . . . 181
1.5.9.11 Working with Object Components . . . 183
1.5.9.12 Object Component Properties . . . 185
1.5.9.12.1 Active Item Component . . . 188
1.5.9.12.2 Arcade Game Component . . . 189
1.5.9.12.3 Camera Component . . . 189
1.5.9.12.4 Clothing Component . . . 190
1.5.9.12.5 Entity Component . . . 190
1.5.9.12.6 Event Timer Component . . . 190
1.5.9.12.7 Furniture Component . . . 191
1.5.9.12.8 Game Launch Component . . . 192
1.5.9.12.9 Game Spawner Component . . . 193
1.5.9.12.10 Header Component . . . 194
1.5.9.12.11 Light Component . . . 195
1.5.9.12.12 Lua Environment Component . . . 196
1.5.9.12.13 MiniGame Component . . . 197
1.5.9.12.14 Network Component . . . 197
1.5.9.12.15 OSD Component . . . 198
1.5.9.12.16 Pad Component . . . 199
1.5.9.12.17 Particles Component . . . 199
1.5.9.12.18 RealTime Game Component . . . 199
1.5.9.12.19 Renderer Component . . . 203
1.5.9.12.20 Repertoire Component . . . 203
1.5.9.12.21 Resource Pack Component . . . 208
1.5.9.12.22 Scene Object Component . . . 209
1.5.9.12.23 Screen Component . . . 210
1.5.9.12.24 Targetable Component . . . 210
1.5.9.13 Managing Object Resources . . . 211
1.5.9.13.1 Adding and Deleting Object Resources . . . 211
1.5.9.13.2 Overriding Object Resources . . . 215
1.5.9.13.3 Recovering Object Resource Files . . . 216
1.5.9.14 Localizing Objects . . . 217
1.5.9.15 Object Metadata . . . 221
1.5.9.16 Compiling Scripts in the Object Editor or Scene Editor . . . 223
1.5.9.17 Managing the User Inventory . . . 223
1.5.9.18 Defining Animation Registers . . . 224
1.5.9.19 Rewards and Commercial Items . . . 225
1.5.9.20 Preparing Objects for Packaging . . . 227
1.5.9.21 Packaging Objects . . . 232
1.5.10 Game Components . . . 234
1.5.10.1 Arcade Games . . . 234
1.5.10.1.1 Rendering a Sprite . . . 236
1.5.10.1.2 Animating a Sprite . . . 238
1.5.10.1.3 Adding Audio . . . 240
1.5.10.1.4 Deploying an Arcade Game in a Scene . . . 241
1.5.10.1.5 Deploying an Arcade Game in Furniture . . . 244
1.5.10.2 Mini-games . . . 246
1.5.10.2.1 Mini-game Design . . . 249
1.5.10.2.2 Maximum Number of Players in Mini-Games . . . 253
1.5.10.2.3 Example Mini-games . . . 255
1.5.10.2.4 Queueing Best Practice . . . 257
1.5.10.3 Realtime Games . . . 258
1.5.10.3.1 Creating a Realtime Game . . . 259
1.5.10.3.2 Joining a Realtime Game . . . 259
1.5.10.3.3 Realtime Game Tokens . . . 260
1.5.10.3.4 Realtime Network XML . . . 261
1.5.10.3.5 Realtime Game Network Guidelines . . . 264
1.5.10.3.6 Realtime Game Network XML Message Tips . . . 266
1.5.10.3.7 Realtime Design Tips . . . 267
1.5.11 Companion Objects . . . 268
1.5.11.1 Creating Companion Objects . . . 270
1.5.11.2 Adding Companion Names . . . 276
1.5.11.3 Recreating Companion Objects . . . 276
1.5.11.4 Companion Object Configuration File . . . 276

1.5.11.4.1 Local and Remote Models . . . 277
1.5.11.4.2 Defining Particle Effects for Companions . . . 278
1.5.11.4.3 Air Element . . . 279
1.5.11.4.4 Ground Element . . . 282
1.5.11.4.5 Companion Animations . . . 285
1.5.11.4.6 Defining Sound Effects for Companions . . . 286
1.5.11.4.7 Proportion Attributes . . . 287
1.5.11.4.8 Specifying a Twist Layer . . . 288
1.5.11.4.9 Actions and Behaviours . . . 289
1.5.11.5 Example Companion Configurations . . . 290
1.5.11.5.1 Configuration 1 - Walk and Wait . . . 290
1.5.11.5.2 Configuration 2 - Hop and Look . . . 291
1.5.11.5.3 Configuration 3 - Roaming Behavior . . . 292
1.5.11.5.4 Configuration 4 - Basic Air State . . . 293
1.5.11.5.5 Configuration 5 - Multiple Air States . . . 294
1.5.11.5.6 Configuration 6 - All Behavior States . . . 295
1.5.11.6 Allocating Memory for Companions . . . 296
1.5.12 Character Components . . . 298
1.5.12.1 Character Component Creation Workflow . . . 299
1.5.12.2 Character Component Types . . . 299
1.5.12.2.1 Avatar Slots . . . 300
1.5.12.2.2 Composite Character Components . . . 305
1.5.12.2.3 Fitting Components Together . . . 311
1.5.12.2.4 Menu Categories for Clothing . . . 312
1.5.12.3 Creating a Character Component . . . 313
1.5.12.4 Creating Headgear Components . . . 314
1.5.12.4.1 Hair Creation Guidelines . . . 315
1.5.12.4.2 Creating a Hat . . . 332
1.5.12.4.3 Creating Headphones . . . 336
1.5.12.4.4 Creating Jewelry . . . 339
1.5.12.4.5 Creating Spectacles . . . 340
1.5.12.4.6 Creating Facial Components . . . 348
1.5.12.5 Building LODs . . . 349
1.5.12.5.1 Building LOD 1 . . . 349
1.5.12.5.2 Building LODs 2 and 3 . . . 356
1.5.12.6 Creating Fat and Thin Targets . . . 360
1.5.12.7 Applying Skinning and Weighting . . . 362
1.5.12.8 Saving a Character Component . . . 368
1.5.12.9 Exporting a Character Component . . . 368
1.5.12.10 Previewing Character Components in PS Home . . . 373
1.5.12.11 Character Component Creation Rules and Conventions . . . 377
1.5.12.12 General Character Component Quality . . . 380
1.5.12.13 Character Component Modeling Best Practice . . . 388
1.5.12.14 Creating Object Variations with the Variation Editor . . . 389
1.5.13 Game Launch Objects . . . 393
1.5.13.1 Title IDs and Communication IDs . . . 394
1.5.13.2 Developing a Game Launch Object . . . 395
1.5.13.3 PS Home Game Launching Menu . . . 399
1.5.13.4 The Game Launch Object Component . . . 400
1.5.13.5 Static XML Game Launching . . . 403
1.5.13.6 Custom Lua Scripted Game Launching . . . 408
1.5.13.7 Supporting Game Launching in your Title . . . 410
1.5.13.8 Launching From Mini-games . . . 413
1.5.13.9 Game Launching Samples . . . 413
1.5.13.10 Game Launching FAQs . . . 418
1.5.14 Portable Items . . . 419
1.5.14.1 Creating Portable Items . . . 419
1.5.14.2 Configuring Portable Items . . . 421
1.5.15 Avatar Interaction Packs . . . 423
1.5.15.1 Creating Avatar Interaction Packs . . . 424
1.5.15.2 Configuring Avatar Interaction Packs . . . 424
1.5.16 Group Animation Packs . . . 427
1.5.16.1 Creating Group Animation Packs . . . 428
1.5.16.2 Configuring Group Animation Packs . . . 429
1.5.17 Sound Packs . . . 431
1.5.17.1 Creating Sound Packs . . . 432
1.5.17.2 Configuring Sound Packs . . . 433
1.6 Scenes . . . 435
1.6.1 PS Home Spaces . . . 436
1.6.1.1 Designing a Home Space . . . 439
1.6.1.2 Clubhouses . . . 444
1.6.1.2.1 Creating Clubhouses . . . 445
1.6.1.2.2 Adding a Club Bulletin Screen . . . 446
1.6.1.2.3 Testing Clubhouses . . . 447
1.6.1.2.4 Launching a Clubhouse . . . 448
1.6.1.3 Regional and Global Instances . . . 449
1.6.1.4 Scene Flags and Settings for Spaces . . . 450
1.6.2 Environments . . . 451
1.6.2.1 Environment Rules and Conventions . . . 452

1.6.2.2 Environments and Scale . . . 453
1.6.2.3 Environment Lighting . . . 454
1.6.2.4 Environment Lightmaps . . . 466
1.6.2.5 Environment Textures . . . 484
1.6.2.6 Shaders . . . 485
1.6.2.7 Blend Shaders . . . 486
1.6.2.7.1 Creating Textures for the Blend Shader . . . 486
1.6.2.7.2 Setting up the Blend Shader . . . 487
1.6.2.7.3 Assigning a Color Set to a Shader . . . 489
1.6.2.7.4 Painting Vertex Color onto the Model . . . 490
1.6.2.7.5 Blend Shader Settings . . . 493
1.6.2.8 Validating and Exporting Environments . . . 495
1.6.2.9 Imogen - PS Home Distributed Lighting Exporter . . . 500
1.6.2.9.1 Imogen Requirements . . . 500
1.6.2.9.2 Imogen Overview . . . 501
1.6.2.9.3 Installing the Imogen Client . . . 503
1.6.2.9.4 Installing the Imogen Slaves . . . 503
1.6.2.9.5 Installing the Imogen Coordinator . . . 504
1.6.2.9.6 Displaying the Imogen Control Panel . . . 504
1.6.2.9.7 Displaying the Imogen Peers Dialog . . . 505
1.6.2.9.8 Validating and Exporting for Imogen Distributed Lighting Exporter . . . 506
1.6.2.9.9 Notes and Troubleshooting . . . 509
1.6.3 Scene Editor . . . 509
1.6.3.1 Scene Editor Interface . . . 510
1.6.3.1.1 Scene Editor Design View . . . 510
1.6.3.1.2 Scene Editor Navigation . . . 511
1.6.3.1.3 Scene Editor Panels . . . 511
1.6.3.1.4 Scene Editor Toolbar Buttons . . . 516
1.6.3.2 Setting Scene Editor Preferences . . . 520
1.6.3.3 Opening Scenes in the Scene Editor . . . 523
1.6.3.4 Adding Objects to a Scene . . . 525
1.6.3.5 Adding Assets to a Scene . . . 526
1.6.3.6 Working With Objects in a Scene . . . 528
1.6.3.7 Game Object Types . . . 529
1.6.3.8 Scene, Object and Asset Properties . . . 540
1.6.3.9 Downloading Objects with Scenes . . . 552
1.6.3.10 Dynamic Spawning . . . 555
1.6.3.11 Targeting System . . . 559
1.6.3.12 Launching a Scene in PS Home . . . 566
1.6.3.13 Preparing Scenes for Packaging . . . 569
1.6.3.14 Packaging Scenes . . . 576
1.7 Scripted Content . . . 577
1.7.1 Lua Scripting . . . 578
1.7.1.1 Scene Scripting . . . 578
1.7.1.2 Object Scripting . . . 580
1.7.1.3 Lua Resources . . . 582
1.7.1.4 Compiling Scripts When Packaging . . . 588
1.7.1.5 Profiling Lua Scripts . . . 590
1.7.1.6 Scripting Tips . . . 594
1.7.1.7 Debugging Lua Script using SLED . . . 595
1.7.1.7.1 Debug Library and Public Library . . . 595
1.7.1.7.2 Launching the .self in Debug Mode . . . 599
1.7.1.7.3 Creating and Managing Projects . . . 601
1.7.1.7.4 Creating and Managing Lua Scripts . . . 603
1.7.1.7.5 Preparing Scripts for Debugging . . . 605
1.7.1.7.6 Setting Breakpoints . . . 606
1.7.1.7.7 Controlling Script Execution . . . 607
1.7.1.7.8 Monitoring Variables . . . 608
1.7.1.7.9 Debugging using SLED - Example . . . 609
1.7.1.8 Lua Garbage Collection . . . 612
1.7.1.8.1 Managing Garbage Collection . . . 615
1.7.2 Lua Features . . . 621
1.7.2.1 Object Information . . . 622
1.7.2.2 Scene Information . . . 623
1.7.2.3 System Information . . . 624
1.7.2.4 Person Information . . . 625
1.7.2.5 Retrieving Object Metadata . . . 627
1.7.2.6 Lobby Instances . . . 628
1.7.2.7 Relocating to Instances . . . 630
1.7.2.8 Managing Groups . . . 632
1.7.2.8.1 Using the GroupDoor Library . . . 633
1.7.2.9 Using the Clubs Library . . . 639
1.7.2.10 Adding Dynamic Lighting . . . 647
1.7.2.11 2D Blending . . . 651
1.7.2.12 Rendering 3D Graphics . . . 655
1.7.2.13 Customizing Displays . . . 656
1.7.2.14 Setting the Rendered Text Font . . . 657
1.7.2.15 Using the OSK Library . . . 658
1.7.2.16 Using the XMB™ Media Lua Library . . . 658

1.7.2.17 PS Home Camera . . . 661
1.7.2.18 Adding Network Communication . . . 664
1.7.2.19 DME Limits . . . 672
1.7.2.20 Additional Guides on Scripting . . . 673
1.7.2.21 Intersection Testing . . . 673
1.7.3 HDK API Scripting . . . 676
1.7.3.1 Running Scripts . . . 676
1.7.3.2 Importing Python Scripts . . . 677
1.7.3.3 Custom UI Elements . . . 678
1.7.3.4 Custom Menu Example . . . 680
1.7.3.5 HDK API Guidelines . . . 682
1.7.3.6 HDK API Executable . . . 682
1.8 Animation and Effects . . . 683
1.8.1 Sky Design . . . 684
1.8.1.1 Sky Design Workflow . . . 686
1.8.1.2 Example Sky Scene Files . . . 687
1.8.1.3 Home Cloud Modeler UI . . . 687
1.8.1.4 Sky Tool UI . . . 696
1.8.1.5 Creating a Home Sky . . . 708
1.8.1.6 View Your Sky . . . 723
1.8.1.7 Adding Sky to Your Scene . . . 727
1.8.1.8 Sky Profiling . . . 730
1.8.1.9 Sky Design UI Nodes . . . 734
1.8.1.10 Atmosphere Lua Reference . . . 744
1.8.1.11 Sky Design Quick Start Tutorial . . . 748
1.8.1.11.1 Initializing a Home Sky Scene . . . 749
1.8.1.11.2 Creating Background Sky Color . . . 751
1.8.1.11.3 Creating High Wispy Clouds . . . 752
1.8.1.11.4 Creating Particle Cloud Dome . . . 755
1.8.1.11.5 Creating Sky Fog . . . 757
1.8.1.11.6 Creating Scene Fog . . . 758
1.8.1.11.7 Exporting and Viewing the Sky on the PlayStation®3 . . . 759
1.8.2 ATG Particle Effects Tool . . . 759
1.8.2.1 ATG Particle Effects Tool Interface . . . 760
1.8.2.2 Particle Effect Nodes . . . 761
1.8.2.3 Working with Particle Effects . . . 767
1.8.2.4 Creating a Particle Effect . . . 768
1.8.2.5 Using Shapers in Particle Effects . . . 773
1.8.2.6 Particle Effect Console Commands . . . 775
1.8.2.7 Particle Effects Profiling . . . 775
1.8.2.8 Exporting Particle Effects . . . 776
1.8.3 Avatar Animation . . . 777
1.8.3.1 Avatar Animation Repertoires . . . 777
1.8.3.1.1 Repertoire - Relationships . . . 778
1.8.3.1.2 Repertoires - Actions & Behaviors . . . 779
1.8.3.1.3 Repertoires - Animation Resources . . . 784
1.8.3.1.4 Repertoires - Blend Tree Animation . . . 785
1.8.3.1.5 Repertoires - Conditions . . . 787
1.8.3.1.6 Repertoires - Output Nodes . . . 792
1.8.3.2 Repertoire Editor . . . 794
1.8.3.2.1 Repertoire Editor Interface . . . 796
1.8.3.2.2 Adding a Repertoire . . . 801
1.8.3.2.3 Adding an Animation Resource . . . 803
1.8.3.2.4 Repertoire Actions . . . 804
1.8.3.2.5 Repertoire Behaviours . . . 805
1.8.3.2.6 Repertoire Editor's Output Node . . . 807
1.8.3.2.7 Repertoire Conditions and Logic . . . 809
1.8.3.2.8 Repertoire Condition Rules . . . 816
1.8.3.2.9 Repertoire Locomotions . . . 818
1.8.3.2.10 Repertoire Emotes . . . 822
1.8.3.2.11 Repertoire Emote Localization . . . 827
1.8.3.2.12 Repertoire Transitions . . . 828
1.8.3.2.13 Repertoire States . . . 829
1.8.3.2.14 Repertoire Animation Blending Workflow . . . 832
1.8.3.2.15 Repertoire Animation Blending . . . 832
1.8.3.2.16 Validating Repertoires . . . 834
1.8.3.2.17 Viewing Repertoires in PS Home . . . 835
1.8.3.2.18 Accessing Repertoires in Script . . . 836
1.8.3.2.19 Repertoire Editor Node Reference . . . 837
1.8.3.3 Avatar Animation Packs . . . 848
1.8.4 Other Animation and Animation Effects . . . 850
1.8.4.1 Creating a Simple Animated Object . . . 851
1.8.4.2 Exporting a Simple Animated Object . . . 853
1.8.4.3 Creating an Entity with a Simple Animated Object . . . 854
1.8.4.4 Scripted Entity Animation . . . 857
1.8.4.5 Scripted Material Animations . . . 861
1.8.4.5.1 Scripted Material Animation Example . . . 863
1.9 Media . . . 866
1.9.1 Media Memory Usage . . . 867

1.9.2 Media Formats for PS Home . . . 872
1.9.2.1 Image Formats . . . 872
1.9.2.2 Video Formats . . . 874
1.9.2.3 Audio Formats . . . 882
1.9.2.4 Media Format Reference . . . 883
1.9.3 Introduction to Audio . . . 883
1.9.3.1 Creating an Audio Asset . . . 883
1.9.3.2 Adding Sounds to a Scene . . . 884
1.9.3.3 Adding Audio Assets to a Scene . . . 886
1.9.3.4 Working with Sound Objects . . . 889
1.9.4 Introduction to Screens . . . 891
1.9.4.1 Screen Types . . . 891
1.9.4.2 Creating Screens . . . 893
1.9.4.3 Streaming Content on Screens . . . 898
1.9.4.4 Troubleshooting Screens . . . 901
1.9.4.5 HSML File Format . . . 901
1.9.4.6 Media RSS File Specification . . . 902
1.9.4.7 Screen Format Quick Reference . . . 905
1.9.5 Video Recorder . . . 906
1.9.5.1 Upload Videos to the Web . . . 910
1.9.5.2 Video Recorder Guidelines and Tips . . . 913
1.9.5.3 Video Recorder Reference . . . 913
1.9.6 VCR . . . 916
1.10 Art Tools . . . 917
1.10.1 Maya and the HDK . . . 917
1.10.1.1 Maya Workflows . . . 919
1.10.1.1.1 General Maya Operations . . . 919
1.10.1.1.2 Smooth Edge Optimization . . . 920
1.10.1.1.3 Environment Optimization . . . 923
1.10.1.2 The Maya Interface . . . 925
1.10.1.2.1 Home_Env Shelf . . . 925
1.10.1.2.2 Home_Char Shelf . . . 927
1.10.1.2.3 Home_Furn Shelf . . . 928
1.10.1.2.4 Home_AvatarAnim Shelf . . . 929
1.10.1.2.5 Home Drop-down Menu . . . 929
1.10.1.3 Geometry Guidelines . . . 934
1.10.1.4 Maya Textures . . . 934
1.10.1.5 Maya File Structure . . . 939
1.10.1.6 Validating and Exporting in Maya . . . 940
1.10.1.7 Export Profiles . . . 946
1.10.1.7.1 Maya Avatar Animation Scene . . . 946
1.10.1.7.2 Maya Character Component Scene . . . 947
1.10.1.7.3 Maya Environment Scene . . . 947
1.10.1.7.4 Maya Furniture Scene . . . 948
1.10.1.7.5 Maya Joint Animation Scene . . . 949
1.10.1.7.6 Maya Model Scene . . . 950
1.10.1.7.7 Maya Prelit Model Scene . . . 950
1.10.1.7.8 Maya Rigid Body Animation . . . 951
1.10.2 Animation Tools . . . 952
1.10.2.1 Animation Compression Tools . . . 952
1.10.2.2 Avatar Animation Tools . . . 954
1.10.2.2.1 Avatar Animation - Creating and Exporting . . . 955
1.10.2.2.2 Avatar Animation Puppet Poser . . . 960
1.10.2.2.3 Avatar Animation Puppet Rig . . . 961
1.10.2.2.4 Optimizing Animation to Reduce Memory Usage . . . 964
1.10.2.3 Joint Animation and Rigid Body Animation . . . 970
1.10.3 Character Tools . . . 971
1.10.3.1 Vertex Normal Tool . . . 972
1.10.3.2 Weighting Tool . . . 974
1.10.3.3 Replace Fat and Thin Blend Shapes . . . 975
1.10.3.4 Character Component Reference . . . 976
1.10.3.4.1 Character Component Facial Hair . . . 977
1.10.3.4.2 Character Component Hair . . . 978
1.10.3.4.3 Character Component Headgear . . . 979
1.10.3.4.4 Character Component Headdress . . . 980
1.10.3.4.5 Character Component Full Headdress . . . 981
1.10.3.4.6 Character Component Jewellery (Left Ear) (Right Ear) . . . 982
1.10.3.4.7 Character Component Spectacles . . . 983
1.10.3.4.8 Character Component HeadPhones . . . 984
1.10.3.4.9 Character Component Hands . . . 985
1.10.3.4.10 Character Component Torso . . . 986
1.10.3.4.11 Character Component Legs . . . 987
1.10.3.4.12 Character Component Feet . . . 988
1.10.3.4.13 Character Component Torso & Legs . . . 989
1.10.3.4.14 Character Component Legs & Feet . . . 990
1.10.3.4.15 Character Component Torso & Legs & Feet . . . 991
1.10.3.4.16 Character Component FullBodySuit . . . 992
1.10.4 Furniture Tools . . . 993
1.10.4.1 Furniture Actives and Arcades . . . 994

1.10.4.2 Furniture Chairs and Seating . . . 995
1.10.4.3 Furniture Lamps and Lights . . . 996
1.10.4.3.1 Maya Light Mask Baking . . . 998
1.10.4.4 Furniture Picture Frames and Wall Decorations . . . 100
1.10.5 Environment Tools . . . 100
1.10.5.1 Maya Lights . . . 100
1.10.5.2 Lighting Set Editor . . . 100
1.10.5.3 Dynamic Reflection Editor . . . 100
1.10.5.4 LOD Groups and LODs . . . 100
1.10.5.5 Atgi Locators . . . 100
1.10.5.6 Creating Particle Locators . . . 101
1.10.5.7 Cloth Simulation for Meshes . . . 101
1.10.6 Static Lighitng (Prelit Meshes) . . . 101
1.10.6.1 Lighting Optimization . . . 102
1.10.6.2 Light Probe Dynamic Lighting . . . 102
1.10.6.3 Static Lighting Export Settings . . . 102
1.10.6.4 Static Lighting RNM and LOM . . . 103
1.10.7 Shader Reference . . . 103
1.10.7.1 Prelit Default . . . 103
1.10.7.2 Prelit Glass . . . 103
1.10.7.3 Prelit Blend . . . 103
1.10.7.4 Default . . . 104
1.10.7.5 Glass . . . 104
1.10.7.6 Blend . . . 104
1.10.7.7 Hair . . . 104
1.10.7.8 Water . . . 104
1.10.7.9 Skin . . . 105
1.10.7.10 Emissive . . . 105
1.10.7.11 Render State . . . 105
1.10.7.12 Texture Scrolling and Flipbook Animation . . . 105
1.10.7.13 Shader Parameters - Tinting . . . 105
1.10.7.14 Lighting Tab . . . 105
1.10.7.15 Real Time Reflections . . . 105
1.10.7.16 Blend Shader Blending . . . 106
1.10.7.17 Environment Maps . . . 106
1.10.7.18 Lightmap UV set 2 . . . 106
1.10.7.19 Vertex Alpha . . . 106
1.10.8 Collision Tools . . . 106
1.10.8.1 Primitive Collision & Mesh Collision . . . 107
1.10.8.2 Collision For Active Items . . . 107
1.10.8.3 Collision For Furniture and Objects . . . 107
1.10.8.4 Collision For Companions . . . 107
1.10.8.5 Collision For Environments . . . 107
1.10.8.6 Collision For Avatars . . . 107
1.10.8.7 Customize Collision Filter . . . 107
1.10.8.8 Scripting and Debugging Collision . . . 108
1.10.8.8.1 Lua Collision Functions . . . 108
1.10.8.8.2 Debugging Collision Items . . . 108
1.10.8.9 Collision Requirements and Recommendations . . . 108
1.10.9 Optional Home Tools . . . 108
1.10.9.1 Source Animation Reference Tool . . . 108
1.11 Testing, Validating and Submitting Content . . . 108
1.11.1 Content Requirements . . . 108
1.11.1.1 Memory Limits . . . 108
1.11.1.1.1 Memory Limits for Spaces . . . 108
1.11.1.1.2 Memory Limits for Scripted Objects . . . 109
1.11.1.1.3 Memory Limits for Non-Scripted Objects . . . 109
1.11.1.2 Models and Meshes . . . 109
1.11.1.3 Dimensions . . . 109
1.11.1.4 Thumbnails . . . 110
1.11.1.4.1 Scene Thumbnails . . . 110
1.11.1.4.2 Navigator Thumbnails . . . 110
1.11.1.4.3 Furniture Thumbnails . . . 110
1.11.1.4.4 Clothing Thumbnails . . . 111
1.11.1.4.5 Companion Thumbnails . . . 112
1.11.1.4.6 Active Item Thumbnails . . . 112
1.11.1.4.7 Unusual Thumbnails . . . 112
1.11.1.5 Age Restrictions . . . 112
1.11.1.6 General Design Guidelines . . . 112
1.11.1.7 Save Data Guidelines . . . 112
1.11.1.8 Non-Player Character Guidelines . . . 112
1.11.2 Profiling in the Client . . . 112
1.11.2.1 Profile GUI . . . 113
1.11.2.2 Character Viewer . . . 113
1.11.2.3 HDK Browser . . . 114
1.11.2.4 Profiling Scenes . . . 114
1.11.2.4.1 PPU Statistics . . . 114
1.11.2.4.2 Dev Debug Console Commands . . . 114
1.11.2.4.3 Frame Rate . . . 114

1.11.2.4.4 Profiling Particles . . . 115
1.11.2.5 Profiling Objects . . . 115
1.11.2.5.1 Individual Non-Scripted Objects . . . 115
1.11.2.5.2 Profiling Character Components . . . 115
1.11.2.5.3 Profiling Companion Objects . . . 115
1.11.2.5.4 Objects with Network Components . . . 115
1.11.2.6 Command Line Batch Packaging . . . 115
1.11.2.7 Batch Validator . . . 115
1.11.2.7.1 Launching and Running the Batch Validator . . . 116
1.11.2.7.2 Batch Validator Metrics . . . 116
1.11.2.7.3 Batch Validator Results . . . 116
1.11.2.7.4 Using the Batch Validator for Furniture and Clothing . . . 116
1.11.2.7.5 Batch Validator Clothing Limits . . . 116
1.11.3 HDK Tools Validations . . . 117
1.11.3.1 Global Validations . . . 117
1.11.3.2 Model Validations . . . 117
1.11.3.3 Space Validations . . . 117
1.11.3.4 Character Components (Clothing) Validations . . . 118
1.11.3.5 Custom Avatar Animation Validations . . . 118
1.11.3.6 Furniture Validations . . . 118
1.11.3.7 Active Item Validations . . . 119
1.11.3.8 Object Editor Validations . . . 119
1.11.3.9 Scene Editor Validations . . . 119
1.11.4 Submission Testing and the CDS . . . 120
1.11.4.1 Submission FAQs . . . 120
1.11.5 Using TTY for Debugging . . . 120
1.11.6 Launching and Testing Content . . . 120
1.12 Systems and Services . . . 120
1.12.1 Collision in PS Home . . . 120
1.12.1.1 Seating and Collision . . . 120
1.12.2 PS Home Physics . . . 120
1.12.2.1 Client Update Rates . . . 120
1.12.2.2 Pulse Forces . . . 121
1.12.2.3 Out of Date Physics Updates . . . 121
1.12.2.4 Physics Functions . . . 121
1.12.3 Chat System in PS Home . . . 121
1.12.3.1 Channels . . . 121
1.12.4 Network Platform . . . 121
1.12.4.1 NP Lua API . . . 121
1.12.4.2 Using the Async Library . . . 121
1.12.4.3 NP Request Queue . . . 122
1.12.4.4 NP Entitlements . . . 122
1.12.4.5 NP Ranking . . . 122
1.12.4.6 NP Ticketing . . . 122
1.12.4.7 NP Ticketing Test Server . . . 122
1.12.4.8 NP Title User Storage . . . 122
1.12.4.9 NP Trophy . . . 122
1.12.4.10 NP Best Practices . . . 122
1.12.4.11 NP Restrictions . . . 123
1.12.4.12 Example - NP Ticketing Mini-game . . . 123
1.12.5 Launch into PS Home from XMB™ or PS3 Title . . . 123
1.12.5.1 Returning to PS Home from PS3 Title - Game Return . . . 123
1.12.5.2 Launch into PS Home from XMB™ . . . 123
1.12.5.3 Testing Game Return . . . 123
1.12.6 The DUALSHOCK®3 Controller . . . 123
1.12.6.1 Handling User Input . . . 123
1.12.6.2 Pad Icons in Localization . . . 124
1.12.7 OSD (On Screen Display) Lua Library . . . 124
1.12.7.1 OSD Elements . . . 124
1.12.7.2 Creating OSD Elements . . . 124
1.12.7.3 Blocked OSD Functions . . . 125
1.12.7.4 Create Custom OSD Elements with XML . . . 125
1.12.7.5 Troubleshooting OSD Elements . . . 125
1.12.7.6 OSD Reference . . . 125
1.12.8 PS Home Rewards . . . 125
1.12.8.1 Rewards Entitlement ID . . . 125
1.12.8.2 Awarding Reward Tickets . . . 125
1.12.8.3 Collecting Reward Tickets . . . 126
1.12.9 Commerce in PS Home . . . 126
1.12.9.1 Developing and Publishing Commerce Content . . . 126
1.12.9.2 Creating a Commerce Point . . . 126
1.12.9.3 Types of Commerce . . . 127
1.12.9.4 Quick Purchase . . . 127
1.12.9.5 Troubleshooting Commerce Content . . . 127
1.12.10 Save Data Service . . . 127
1.13 Lua API Reference . . . 127
1.13.1 Misc . . . 127
1.13.1.1 Const . . . 127
1.13.1.2 CreateTable . . . 127

1.13.1.3 DebugGetProfileTime . . . 127
1.13.1.4 GetDeltaTime . . . 127
1.13.1.5 GetInstanceTime . . . 127
1.13.1.6 GetThreadId . . . 127
1.13.1.7 IsPaused . . . 127
1.13.1.8 KillThread . . . 127
1.13.1.9 LoadLibrary . . . 128
1.13.1.10 Pause . . . 128
1.13.1.11 Resume . . . 128
1.13.1.12 Sleep . . . 128
1.13.1.13 SpawnFunction . . . 128
1.13.1.14 Type . . . 128
1.13.2 Active . . . 128
1.13.2.1 Active.GetEntity . . . 128
1.13.2.2 Active.GetMaxSaveDataSize . . . 128
1.13.2.3 Active.IsMoving . . . 128
1.13.2.4 Active.IsSafeVolumeIntersected . . . 128
1.13.2.5 Active.IsSaveDataAccess . . . 128
1.13.2.6 Active.IsUpright . . . 128
1.13.2.7 Active.LoadData . . . 128
1.13.2.8 Active.SaveData . . . 128
1.13.3 ActivityChannel . . . 128
1.13.3.1 ActivityChannel.GetConnectionState . . . 128
1.13.3.2 ActivityChannel.Initialize . . . 128
1.13.3.3 ActivityChannel.SetCommunicationPermissions . . . 128
1.13.3.4 ActivityChannel.SetTitle . . . 129
1.13.4 Animation . . . 129
1.13.4.1 Animation.Create . . . 129
1.13.4.2 Animation.GetBlendState . . . 129
1.13.4.3 Animation.GetBlendTime . . . 129
1.13.4.4 Animation.GetBlendWeight . . . 129
1.13.4.5 Animation.GetDuration . . . 129
1.13.4.6 Animation.GetName . . . 129
1.13.4.7 Animation.GetPlaybackRate . . . 129
1.13.4.8 Animation.GetTime . . . 129
1.13.4.9 Animation.IsValid . . . 129
1.13.4.10 Animation.SetBlendWeight . . . 129
1.13.4.11 Animation.SetLoopCallback . . . 129
1.13.4.12 Animation.SetPlaybackRate . . . 129
1.13.4.13 Animation.SetTime . . . 129
1.13.5 ArcadeGame . . . 129
1.13.5.1 ArcadeGame.EnableMlaa . . . 129
1.13.5.2 ArcadeGame.EnableSceneRender . . . 129
1.13.5.3 ArcadeGame.EnableVsync . . . 129
1.13.5.4 ArcadeGame.GetScreen . . . 130
1.13.5.5 ArcadeGame.LeaveGame . . . 130
1.13.6 AsyncCommand . . . 130
1.13.6.1 AsyncCommand.Cancel . . . 130
1.13.6.2 AsyncCommand.Create . . . 130
1.13.6.3 AsyncCommand.Execute . . . 130
1.13.6.4 AsyncCommand.GetErrorCode . . . 130
1.13.6.5 AsyncCommand.GetNpErrorCode . . . 130
1.13.6.6 AsyncCommand.GetResultCount . . . 130
1.13.6.7 AsyncCommand.GetResults . . . 130
1.13.6.8 AsyncCommand.IsFinished . . . 130
1.13.6.9 AsyncCommand.IsInProgress . . . 130
1.13.6.10 AsyncCommand.Succeeded . . . 130
1.13.7 AsyncContext . . . 130
1.13.7.1 AsyncContext.Create . . . 130
1.13.8 Atmos . . . 130
1.13.8.1 Atmos.GetParam . . . 130
1.13.8.2 Atmos.SetParam . . . 130
1.13.9 Base64 . . . 130
1.13.9.1 Base64.Decode . . . 130
1.13.9.2 Base64.DecodeFromString . . . 131
1.13.9.3 Base64.Encode . . . 131
1.13.10 BasicGenx . . . 131
1.13.10.1 BasicGenx.AddAttribute . . . 131
1.13.10.2 BasicGenx.AddAttributeInt . . . 131
1.13.10.3 BasicGenx.AddText . . . 131
1.13.10.4 BasicGenx.AddTextAsInt . . . 131
1.13.10.5 BasicGenx.CheckText . . . 131
1.13.10.6 BasicGenx.Create . . . 131
1.13.10.7 BasicGenx.EndDocument . . . 131
1.13.10.8 BasicGenx.EndElement . . . 131
1.13.10.9 BasicGenx.StartDocument . . . 131
1.13.10.10 BasicGenx.StartElement . . . 131
1.13.11 BigInt . . . 131
1.13.11.1 BigInt.Add . . . 131

1.13.11.2 BigInt.And . . . 131
1.13.11.3 BigInt.AsHexString . . . 131
1.13.11.4 BigInt.Create . . . 132
1.13.11.5 BigInt.Divide . . . 132
1.13.11.6 BigInt.GetBit . . . 132
1.13.11.7 BigInt.Multiply . . . 132
1.13.11.8 BigInt.Negate . . . 132
1.13.11.9 BigInt.Not . . . 132
1.13.11.10 BigInt.Or . . . 132
1.13.11.11 BigInt.Set . . . 132
1.13.11.12 BigInt.SetBit . . . 132
1.13.11.13 BigInt.SetZero . . . 132
1.13.11.14 BigInt.ShiftLeft . . . 132
1.13.11.15 BigInt.ShiftRight . . . 132
1.13.11.16 BigInt.Subtract . . . 132
1.13.11.17 BigInt.ToNumber . . . 132
1.13.11.18 BigInt.Xor . . . 132
1.13.12 Camera . . . 132
1.13.12.1 Camera.Activate . . . 132
1.13.12.2 Camera.AttachToParent . . . 132
1.13.12.3 Camera.Create . . . 132
1.13.12.4 Camera.Deactivate . . . 133
1.13.12.5 Camera.EnableAvatarCameraSync . . . 133
1.13.12.6 Camera.GetInverseViewMatrix . . . 133
1.13.12.7 Camera.GetLookAtPosition . . . 133
1.13.12.8 Camera.GetPosition . . . 133
1.13.12.9 Camera.GetProjectionMatrix . . . 133
1.13.12.10 Camera.GetViewMatrix . . . 133
1.13.12.11 Camera.GetViewPortFOV . . . 133
1.13.12.12 Camera.GetViewPortZFar . . . 133
1.13.12.13 Camera.GetViewPortZNear . . . 133
1.13.12.14 Camera.IsActivated . . . 133
1.13.12.15 Camera.SetLookAtPosition . . . 133
1.13.12.16 Camera.SetPosition . . . 133
1.13.12.17 Camera.SetViewPortAspect . . . 133
1.13.12.18 Camera.SetViewPortFOV . . . 133
1.13.12.19 Camera.SetViewPortZFar . . . 133
1.13.12.20 Camera.SetViewPortZNear . . . 133
1.13.13 Cloth . . . 133
1.13.13.1 Cloth.SetWindAngle . . . 133
1.13.14 Clubs . . . 134
1.13.14.1 Clubs.GetClubCount . . . 134
1.13.14.2 Clubs.GetClubInfo . . . 134
1.13.14.3 Clubs.GetClubList . . . 134
1.13.14.4 Clubs.GetInstanceMemberList . . . 134
1.13.14.5 Clubs.GetMemberCount . . . 134
1.13.14.6 Clubs.GetMemberInfo . . . 134
1.13.14.7 Clubs.GetMemberList . . . 134
1.13.14.8 Clubs.IsRefreshed . . . 134
1.13.14.9 Clubs.IsRefreshing . . . 134
1.13.14.10 Clubs.RequestRefresh . . . 134
1.13.15 Collision . . . 134
1.13.15.1 Collision.Create . . . 134
1.13.15.2 Collision.CreateLinearCast . . . 134
1.13.15.3 Collision.CreateRaycast . . . 135
1.13.16 Debug . . . 135
1.13.16.1 Debug.ArcadeGameGetActive . . . 135
1.13.16.2 Debug.ArcadeGameLoad . . . 135
1.13.16.3 Debug.ArcadeGameReset . . . 135
1.13.16.4 Debug.ArcadeGameSetBorderColor . . . 135
1.13.16.5 Debug.DebuggerRegister . . . 135
1.13.16.6 Debug.DebuggerSetupProject . . . 135
1.13.16.7 Debug.DebuggerUnRegister . . . 135
1.13.16.8 Debug.DrawBox3d . . . 135
1.13.16.9 Debug.DrawCone3d . . . 135
1.13.16.10 Debug.DrawCylinder3d . . . 135
1.13.16.11 Debug.DrawLine3d . . . 135
1.13.16.12 Debug.DrawQuad3d . . . 135
1.13.16.13 Debug.DrawRect . . . 135
1.13.16.14 Debug.DrawSphere3d . . . 135
1.13.16.15 Debug.DrawText . . . 136
1.13.16.16 Debug.DrawTriangle3d . . . 136
1.13.16.17 Debug.GetDevMode . . . 136
1.13.16.18 Debug.GetStack . . . 136
1.13.16.19 Debug.GetThreadIds . . . 136
1.13.16.20 Debug.GetWorldToScreen . . . 136
1.13.16.21 Debug.KeyIsHeld . . . 136
1.13.16.22 Debug.KeyIsPressed . . . 136
1.13.16.23 Debug.KeyIsReleased . . . 136

1.13.16.24 Debug.MemGetLargestBlock . . . 136
1.13.16.25 Debug.MemGetTotal . . . 136
1.13.16.26 Debug.MemGetUsed . . . 136
1.13.16.27 Debug.MiniGameGetActive . . . 136
1.13.16.28 Debug.MiniGameGetClosest . . . 136
1.13.16.29 Debug.MiniGameReset . . . 136
1.13.16.30 Debug.MouseGetVelocityX . . . 136
1.13.16.31 Debug.MouseGetVelocityY . . . 137
1.13.16.32 Debug.MouseIsButtonHeld . . . 137
1.13.16.33 Debug.MouseIsButtonPressed . . . 137
1.13.16.34 Debug.MouseIsButtonReleased . . . 137
1.13.16.35 Debug.RunScript . . . 137
1.13.16.36 Debug.SceneIsReady . . . 137
1.13.16.37 Debug.SceneReset . . . 137
1.13.16.38 Debug.ScreenSetContent . . . 137
1.13.16.39 Debug.SetErrorCallback . . . 137
1.13.16.40 Debug.System . . . 137
1.13.17 Entity . . . 137
1.13.17.1 Entity.ApplyAngularImpulse . . . 137
1.13.17.2 Entity.ApplyImpulse . . . 137
1.13.17.3 Entity.AttachPersonLabel . . . 137
1.13.17.4 Entity.AttachToBone . . . 137
1.13.17.5 Entity.AttachToParent . . . 137
1.13.17.6 Entity.BlendAnimIn . . . 137
1.13.17.7 Entity.BlendAnimOut . . . 138
1.13.17.8 Entity.Copy . . . 138
1.13.17.9 Entity.Create . . . 138
1.13.17.10 Entity.CreateForceAction . . . 138
1.13.17.11 Entity.CreatePointForceAction . . . 138
1.13.17.12 Entity.CreateTorqueAction . . . 138
1.13.17.13 Entity.DebugSetAxesVisible . . . 138
1.13.17.14 Entity.DebugSetCollisionVisible . . . 138
1.13.17.15 Entity.DebugSetHierarchyVisible . . . 138
1.13.17.16 Entity.DestroyAction . . . 138
1.13.17.17 Entity.DetachPersonLabel . . . 138
1.13.17.18 Entity.EnableAction . . . 138
1.13.17.19 Entity.EnableCollision . . . 138
1.13.17.20 Entity.EnableCollisionCallbacks . . . 138
1.13.17.21 Entity.EnableDynamics . . . 138
1.13.17.22 Entity.EnableDynamicsAutoDeactivation . . . 138
1.13.17.23 Entity.EnableInterpolatedMode . . . 139
1.13.17.24 Entity.EnableKeepUpright . . . 139
1.13.17.25 Entity.EnableLayerCollision . . . 139
1.13.17.26 Entity.EvaluateAnims . . . 139
1.13.17.27 Entity.FindMaterials . . . 139
1.13.17.28 Entity.GetActiveAnim . . . 139
1.13.17.29 Entity.GetAngularDamping . . . 139
1.13.17.30 Entity.GetAngularVelocity . . . 139
1.13.17.31 Entity.GetBoneCount . . . 139
1.13.17.32 Entity.GetBoneIndex . . . 139
1.13.17.33 Entity.GetBoneLocalMatrix . . . 139
1.13.17.34 Entity.GetBoneWorldMatrix . . . 139
1.13.17.35 Entity.GetCenterOfMassLocal . . . 139
1.13.17.36 Entity.GetCenterOfMassWorld . . . 139
1.13.17.37 Entity.GetChildren . . . 139
1.13.17.38 Entity.GetCloth . . . 139
1.13.17.39 Entity.GetCollisionLayer . . . 140
1.13.17.40 Entity.GetContactCount . . . 140
1.13.17.41 Entity.GetContactEntity . . . 140
1.13.17.42 Entity.GetContactId . . . 140
1.13.17.43 Entity.GetContactPoint . . . 140
1.13.17.44 Entity.GetContactRelVelocity . . . 140
1.13.17.45 Entity.GetContactUserId . . . 140
1.13.17.46 Entity.GetCullDistance . . . 140
1.13.17.47 Entity.GetGravity . . . 140
1.13.17.48 Entity.GetIndexOfAnim . . . 140
1.13.17.49 Entity.GetLinearDamping . . . 140
1.13.17.50 Entity.GetLinearVelocity . . . 140
1.13.17.51 Entity.GetLocalBounds . . . 140
1.13.17.52 Entity.GetLocalMatrix . . . 140
1.13.17.53 Entity.GetMass . . . 140
1.13.17.54 Entity.GetNumActiveAnims . . . 140
1.13.17.55 Entity.GetParent . . . 141
1.13.17.56 Entity.GetPointVelocity . . . 141
1.13.17.57 Entity.GetPosition . . . 141
1.13.17.58 Entity.GetRotationQuat . . . 141
1.13.17.59 Entity.GetRotationXyz . . . 141
1.13.17.60 Entity.GetScale . . . 141
1.13.17.61 Entity.GetSiblings . . . 141

1.13.17.62 Entity.GetUniqueId . . . 141
1.13.17.63 Entity.GetWorldBounds . . . 141
1.13.17.64 Entity.GetWorldMatrix . . . 141
1.13.17.65 Entity.HasActiveAnims . . . 141
1.13.17.66 Entity.IsActionEnabled . . . 141
1.13.17.67 Entity.IsDynamicsActive . . . 141
1.13.17.68 Entity.IsDynamicsAutoDeactivationEnabled . . . 141
1.13.17.69 Entity.IsInterpolatedModeEnabled . . . 141
1.13.17.70 Entity.IsKeepUprightEnabled . . . 141
1.13.17.71 Entity.IsLayerCollisionEnabled . . . 141
1.13.17.72 Entity.IsVisible . . . 142
1.13.17.73 Entity.LookAt . . . 142
1.13.17.74 Entity.LookAtWorld . . . 142
1.13.17.75 Entity.ParticleIsAlive . . . 142
1.13.17.76 Entity.ParticleReset . . . 142
1.13.17.77 Entity.ParticleSetAttachType . . . 142
1.13.17.78 Entity.ParticleSetEffect . . . 142
1.13.17.79 Entity.ParticleSetEmitting . . . 142
1.13.17.80 Entity.ParticleSetUpdate . . . 142
1.13.17.81 Entity.ParticleSetUserParam . . . 142
1.13.17.82 Entity.PauseAnim . . . 142
1.13.17.83 Entity.PlayAnim . . . 142
1.13.17.84 Entity.RegisterCollisionCallback . . . 142
1.13.17.85 Entity.Reset . . . 142
1.13.17.86 Entity.ResumeAnim . . . 142
1.13.17.87 Entity.SetActionForce . . . 142
1.13.17.88 Entity.SetActionPosition . . . 143
1.13.17.89 Entity.SetActionTorque . . . 143
1.13.17.90 Entity.SetAngularDamping . . . 143
1.13.17.91 Entity.SetAngularVelocity . . . 143
1.13.17.92 Entity.SetAttachType . . . 143
1.13.17.93 Entity.SetCenterOfMassLocal . . . 143
1.13.17.94 Entity.SetCollision . . . 143
1.13.17.95 Entity.SetCollisionLayer . . . 143
1.13.17.96 Entity.SetCullDistance . . . 143
1.13.17.97 Entity.SetDynamicsActive . . . 143
1.13.17.98 Entity.SetFrictionCoeff . . . 143
1.13.17.99 Entity.SetGravity . . . 143
1.13.17.100 Entity.SetLight . . . 143
1.13.17.101 Entity.SetLinearDamping . . . 143
1.13.17.102 Entity.SetLinearVelocity . . . 143
1.13.17.103 Entity.SetLocalMatrix . . . 143
1.13.17.104 Entity.SetMass . . . 144
1.13.17.105 Entity.SetMaxContacts . . . 144
1.13.17.106 Entity.SetModel . . . 144
1.13.17.107 Entity.SetPosition . . . 144
1.13.17.108 Entity.SetRestitutionCoeff . . . 144
1.13.17.109 Entity.SetRotationQuat . . . 144
1.13.17.110 Entity.SetRotationX . . . 144
1.13.17.111 Entity.SetRotationXyz . . . 144
1.13.17.112 Entity.SetRotationY . . . 144
1.13.17.113 Entity.SetRotationZ . . . 144
1.13.17.114 Entity.SetScale . . . 144
1.13.17.115 Entity.SetScreen . . . 144
1.13.17.116 Entity.SetSkeleton . . . 144
1.13.17.117 Entity.SetVisible . . . 144
1.13.17.118 Entity.StopAnim . . . 144
1.13.18 Environment . . . 144
1.13.18.1 Environment.DebugProfileBegin . . . 144
1.13.18.2 Environment.DebugProfileEnd . . . 144
1.13.18.3 Environment.EnableCallbackGc . . . 144
1.13.18.4 Environment.EnableFrameGc . . . 145
1.13.18.5 Environment.SetCallbackGcInterval . . . 145
1.13.18.6 Environment.SetCallbackGcStep . . . 145
1.13.18.7 Environment.SetFrameGcStep . . . 145
1.13.19 Font . . . 145
1.13.19.1 Font.Create . . . 145
1.13.20 Furniture . . . 145
1.13.20.1 Furniture.ClearLayout . . . 145
1.13.20.2 Furniture.ExportLayout . . . 145
1.13.20.3 Furniture.GetCategories . . . 145
1.13.20.4 Furniture.GetItemCount . . . 145
1.13.20.5 Furniture.GetItemData . . . 145
1.13.20.6 Furniture.ImportLayout . . . 145
1.13.20.7 Furniture.IsSystemBusy . . . 145
1.13.20.8 Furniture.LoadLayout . . . 145
1.13.20.9 Furniture.SaveLayout . . . 145
1.13.20.10 Furniture.SetCallback . . . 145
1.13.21 GameInfo . . . 145

1.13.21.1 GameInfo.GetSearchResults . . . 145
1.13.21.2 GameInfo.IsGamesListBusy . . . 146
1.13.21.3 GameInfo.IsSearchFinished . . . 146
1.13.21.4 GameInfo.SearchByTitleId . . . 146
1.13.22 GameLaunch . . . 146
1.13.22.1 GameLaunch.CreateSession . . . 146
1.13.22.2 GameLaunch.GetCurrentFocusPersonId . . . 146
1.13.22.3 GameLaunch.GetMaxPlayers . . . 146
1.13.22.4 GameLaunch.GetNumberOfPlayers . . . 146
1.13.22.5 GameLaunch.GetSessionMasterPersonId . . . 146
1.13.22.6 GameLaunch.GetTitle . . . 146
1.13.22.7 GameLaunch.GetUserComment . . . 146
1.13.22.8 GameLaunch.InitiateLaunch . . . 146
1.13.22.9 GameLaunch.LeaveSession . . . 146
1.13.22.10 GameLaunch.OnHideUI . . . 146
1.13.22.11 GameLaunch.OnShowUI . . . 146
1.13.22.12 GameLaunch.ReturnJoinGameResult . . . 146
1.13.22.13 GameLaunch.ReturnLaunchRequestResult . . . 146
1.13.22.14 GameLaunch.SetMaxPlayers . . . 147
1.13.22.15 GameLaunch.SetNumPrivateSlots . . . 147
1.13.22.16 GameLaunch.SetUserComment . . . 147
1.13.23 GameLaunchButton . . . 147
1.13.23.1 GameLaunchButton.GetNameRef . . . 147
1.13.23.2 GameLaunchButton.SetCallbackOnPressed . . . 147
1.13.23.3 GameLaunchButton.SetText . . . 147
1.13.24 GameLaunchExporter . . . 147
1.13.24.1 GameLaunchExporter.BeginSection . . . 147
1.13.24.2 GameLaunchExporter.BeginSessionExport . . . 147
1.13.24.3 GameLaunchExporter.EndSection . . . 147
1.13.24.4 GameLaunchExporter.EndSessionExport . . . 147
1.13.24.5 GameLaunchExporter.ExportOption . . . 147
1.13.25 GameLaunchListItem . . . 147
1.13.25.1 GameLaunchListItem.AddItem . . . 147
1.13.25.2 GameLaunchListItem.AddItemRaw . . . 147
1.13.25.3 GameLaunchListItem.GetCallbackFocusItem . . . 147
1.13.25.4 GameLaunchListItem.GetFocusItem . . . 147
1.13.25.5 GameLaunchListItem.GetNameRef . . . 147
1.13.25.6 GameLaunchListItem.RemoveAllItems . . . 147
1.13.25.7 GameLaunchListItem.RemoveItem . . . 147
1.13.25.8 GameLaunchListItem.SetCallbackOnChange . . . 148
1.13.25.9 GameLaunchListItem.SetFocusItem . . . 148
1.13.26 GameLaunchMenu . . . 148
1.13.26.1 GameLaunchMenu.AddItemButton . . . 148
1.13.26.2 GameLaunchMenu.AddItemList . . . 148
1.13.26.3 GameLaunchMenu.AddItemSlider . . . 148
1.13.26.4 GameLaunchMenu.FindItemByNameRef . . . 148
1.13.26.5 GameLaunchMenu.GetFirstItem . . . 148
1.13.26.6 GameLaunchMenu.GetId . . . 148
1.13.26.7 GameLaunchMenu.GetNextItem . . . 148
1.13.27 GameLaunchMenuSys . . . 148
1.13.27.1 GameLaunchMenuSys.AddMenu . . . 148
1.13.27.2 GameLaunchMenuSys.CreateMenu . . . 148
1.13.27.3 GameLaunchMenuSys.FindMenuById . . . 148
1.13.27.4 GameLaunchMenuSys.GetFirstMenu . . . 148
1.13.27.5 GameLaunchMenuSys.GetNextMenu . . . 148
1.13.27.6 GameLaunchMenuSys.RemoveMenu . . . 148
1.13.28 GameLaunchSlider . . . 149
1.13.28.1 GameLaunchSlider.GetNameRef . . . 149
1.13.28.2 GameLaunchSlider.GetValue . . . 149
1.13.28.3 GameLaunchSlider.SetIntegerSlider . . . 149
1.13.28.4 GameLaunchSlider.SetPercentageSlider . . . 149
1.13.28.5 GameLaunchSlider.SetValue . . . 149
1.13.29 GfxEffect . . . 149
1.13.29.1 GfxEffect.Create . . . 149
1.13.29.2 GfxEffect.Enable . . . 149
1.13.29.3 GfxEffect.IsEnabled . . . 149
1.13.29.4 GfxEffect.SetAlphaRef . . . 149
1.13.29.5 GfxEffect.SetBlendMode . . . 149
1.13.29.6 GfxEffect.SetBounds . . . 149
1.13.29.7 GfxEffect.SetDepthPrepass . . . 149
1.13.29.8 GfxEffect.SetDrawRange . . . 149
1.13.29.9 GfxEffect.SetIndexData . . . 149
1.13.29.10 GfxEffect.SetRenderStage . . . 149
1.13.29.11 GfxEffect.SetShadowEnable . . . 149
1.13.29.12 GfxEffect.SetTexture . . . 149
1.13.29.13 GfxEffect.SetTransform . . . 150
1.13.29.14 GfxEffect.SetTwoSided . . . 150
1.13.29.15 GfxEffect.SetVertexData . . . 150
1.13.30 Group . . . 150

1.13.30.1 Group.Create . . . 150
1.13.30.2 Group.GetId . . . 150
1.13.30.3 Group.GetMemberCount . . . 150
1.13.30.4 Group.GetMemberList . . . 150
1.13.30.5 Group.IsInGroup . . . 150
1.13.30.6 Group.Join . . . 150
1.13.30.7 Group.Leave . . . 150
1.13.31 GroupDoor . . . 150
1.13.31.1 GroupDoor.Create . . . 150
1.13.31.2 GroupDoor.Enter . . . 150
1.13.31.3 GroupDoor.GetFailedEnterMessage . . . 150
1.13.32 Hash . . . 150
1.13.32.1 Hash.Create . . . 150
1.13.32.2 Hash.Finalize . . . 150
1.13.32.3 Hash.Start . . . 150
1.13.32.4 Hash.Update . . . 150
1.13.33 HelpSystem . . . 151
1.13.33.1 HelpSystem.GetObjectHelpText . . . 151
1.13.34 HttpPostData . . . 151
1.13.34.1 HttpPostData.AddDataHeader . . . 151
1.13.34.2 HttpPostData.AddHttpHeader . . . 151
1.13.34.3 HttpPostData.AddValue . . . 151
1.13.34.4 HttpPostData.Create . . . 151
1.13.34.5 HttpPostData.Finalize . . . 151
1.13.34.6 HttpPostData.FinalizeData . . . 151
1.13.34.7 HttpPostData.FinalizeHttpHeaders . . . 151
1.13.34.8 HttpPostData.Reset . . . 151
1.13.35 Intersect . . . 151
1.13.35.1 Intersect.AabbAabb2d . . . 151
1.13.35.2 Intersect.AabbAabb3d . . . 151
1.13.35.3 Intersect.AabbPlane3d . . . 151
1.13.35.4 Intersect.CircleAabb2d . . . 151
1.13.35.5 Intersect.CircleCircle2d . . . 151
1.13.35.6 Intersect.DistSqLineLine3d . . . 151
1.13.35.7 Intersect.LineAabb2d . . . 151
1.13.35.8 Intersect.LineAabb3d . . . 152
1.13.35.9 Intersect.LineCircle2d . . . 152
1.13.35.10 Intersect.LineLine2d . . . 152
1.13.35.11 Intersect.LinePlane3d . . . 152
1.13.35.12 Intersect.LineSegment2d . . . 152
1.13.35.13 Intersect.LineSphere3d . . . 152
1.13.35.14 Intersect.PlanePlane3d . . . 152
1.13.35.15 Intersect.RayAabb2d . . . 152
1.13.35.16 Intersect.RayAabb3d . . . 152
1.13.35.17 Intersect.RayCircle2d . . . 152
1.13.35.18 Intersect.RayLine2d . . . 152
1.13.35.19 Intersect.RayPlane3d . . . 152
1.13.35.20 Intersect.RayRay2d . . . 152
1.13.35.21 Intersect.RaySegment2d . . . 153
1.13.35.22 Intersect.RaySphere3d . . . 153
1.13.35.23 Intersect.SegmentAabb2d . . . 153
1.13.35.24 Intersect.SegmentAabb3d . . . 153
1.13.35.25 Intersect.SegmentCircle2d . . . 153
1.13.35.26 Intersect.SegmentPlane3d . . . 153
1.13.35.27 Intersect.SegmentSegment2d . . . 153
1.13.35.28 Intersect.SegmentSphere3d . . . 153
1.13.35.29 Intersect.SphereAabb3d . . . 153
1.13.35.30 Intersect.SpherePlane3d . . . 153
1.13.35.31 Intersect.SphereSphere3d . . . 153
1.13.36 Json . . . 153
1.13.36.1 Json.Decode . . . 153
1.13.36.2 Json.DecodeFromString . . . 153
1.13.36.3 Json.Encode . . . 154
1.13.37 Light . . . 154
1.13.37.1 Light.Create . . . 154
1.13.37.2 Light.SetAttenuationEnd . . . 154
1.13.37.3 Light.SetAttenuationPower . . . 154
1.13.37.4 Light.SetAttenuationStart . . . 154
1.13.37.5 Light.SetColor . . . 154
1.13.37.6 Light.SetMask . . . 154
1.13.37.7 Light.SetShadowQuality . . . 154
1.13.37.8 Light.SetSpotFov . . . 154
1.13.37.9 Light.SetSpotShadow . . . 154
1.13.38 LinearCast . . . 154
1.13.38.1 LinearCast.DebugDraw . . . 154
1.13.38.2 LinearCast.GetEndPoint . . . 154
1.13.38.3 LinearCast.GetHitCount . . . 154
1.13.38.4 LinearCast.GetHitData . . . 154
1.13.38.5 LinearCast.GetHitDistance . . . 155

1.13.38.6 LinearCast.GetHitNormal . . . 155
1.13.38.7 LinearCast.GetHitPoint . . . 155
1.13.38.8 LinearCast.GetHitType . . . 155
1.13.38.9 LinearCast.GetStartPoint . . . 155
1.13.38.10 LinearCast.IsReady . . . 155
1.13.38.11 LinearCast.SetCollisionLayersToHit . . . 155
1.13.38.12 LinearCast.SetRotation . . . 155
1.13.38.13 LinearCast.Start . . . 155
1.13.39 LocalPlayer . . . 155
1.13.39.1 LocalPlayer.AddRepertoire . . . 155
1.13.39.2 LocalPlayer.AddToPlayersMetList . . . 156
1.13.39.3 LocalPlayer.DebugRemovePortable . . . 156
1.13.39.4 LocalPlayer.DebugSetPortable . . . 156
1.13.39.5 LocalPlayer.DoAction . . . 156
1.13.39.6 LocalPlayer.EnableAutoHiding . . . 156
1.13.39.7 LocalPlayer.GetAccountRegion . . . 156
1.13.39.8 LocalPlayer.GetAccountTerritory . . . 156
1.13.39.9 LocalPlayer.GetAge . . . 156
1.13.39.10 LocalPlayer.GetAvatarRig . . . 156
1.13.39.11 LocalPlayer.GetCurrentLogin . . . 156
1.13.39.12 LocalPlayer.GetCurrentRegion . . . 156
1.13.39.13 LocalPlayer.GetFirstLogin . . . 156
1.13.39.14 LocalPlayer.GetFriendCount . . . 156
1.13.39.15 LocalPlayer.GetFriends . . . 156
1.13.39.16 LocalPlayer.GetHomeAccountId . . . 156
1.13.39.17 LocalPlayer.GetHomeRegion . . . 156
1.13.39.18 LocalPlayer.GetLastLogin . . . 156
1.13.39.19 LocalPlayer.GetLoginCount . . . 157
1.13.39.20 LocalPlayer.GetPerson . . . 157
1.13.39.21 LocalPlayer.GetPersonId . . . 157
1.13.39.22 LocalPlayer.GetPreviousUserType . . . 157
1.13.39.23 LocalPlayer.GetUser . . . 157
1.13.39.24 LocalPlayer.GetUserType . . . 157
1.13.39.25 LocalPlayer.IsChatAllowed . . . 157
1.13.39.26 LocalPlayer.IsDoingAction . . . 157
1.13.39.27 LocalPlayer.IsInWardrobe . . . 157
1.13.39.28 LocalPlayer.Relocate . . . 157
1.13.39.29 LocalPlayer.RelocateToGroupInstance . . . 157
1.13.39.30 LocalPlayer.RelocateToUniqueInstance . . . 157
1.13.39.31 LocalPlayer.RemoveRepertoire . . . 157
1.13.39.32 LocalPlayer.RestoreAvatar . . . 157
1.13.39.33 LocalPlayer.SetAvatarRig . . . 157
1.13.39.34 LocalPlayer.SetClothingItems . . . 157
1.13.39.35 LocalPlayer.SetRelocateObject . . . 157
1.13.39.36 LocalPlayer.SetRigComponentObjectId . . . 157
1.13.40 Material . . . 158
1.13.40.1 Material.GetParam . . . 158
1.13.40.2 Material.SetAlphaRef . . . 158
1.13.40.3 Material.SetBlendMode . . . 158
1.13.40.4 Material.SetDepthPrepass . . . 158
1.13.40.5 Material.SetMatrix . . . 158
1.13.40.6 Material.SetTexture . . . 158
1.13.40.7 Material.SetVector . . . 158
1.13.41 Matrix44 . . . 158
1.13.41.1 Matrix44.Add . . . 158
1.13.41.2 Matrix44.Copy . . . 158
1.13.41.3 Matrix44.Create . . . 158
1.13.41.4 Matrix44.GetRotationXyz . . . 158
1.13.41.5 Matrix44.GetRow . . . 158
1.13.41.6 Matrix44.Invert . . . 158
1.13.41.7 Matrix44.InvertAffine . . . 158
1.13.41.8 Matrix44.InvertAffineOrthogonal . . . 158
1.13.41.9 Matrix44.IsEqual . . . 158
1.13.41.10 Matrix44.IsIdentity . . . 159
1.13.41.11 Matrix44.Multiply . . . 159
1.13.41.12 Matrix44.Orthonormalize . . . 159
1.13.41.13 Matrix44.SetAxisRotation . . . 159
1.13.41.14 Matrix44.SetIdentity . . . 159
1.13.41.15 Matrix44.SetRotationQuat . . . 159
1.13.41.16 Matrix44.SetRotationXyz . . . 159
1.13.41.17 Matrix44.SetRow . . . 159
1.13.41.18 Matrix44.SetScale . . . 159
1.13.41.19 Matrix44.SetTranslation . . . 159
1.13.41.20 Matrix44.Subtract . . . 159
1.13.41.21 Matrix44.Transpose . . . 159
1.13.42 MediaLibrary . . . 159
1.13.42.1 MediaLibrary.ExportImage . . . 159
1.13.42.2 MediaLibrary.ExportVideo . . . 159
1.13.42.3 MediaLibrary.GetSearchResults . . . 159

1.13.42.4 MediaLibrary.IsExporting . . . 159
1.13.42.5 MediaLibrary.IsSearchFinished . . . 160
1.13.42.6 MediaLibrary.IsSearching . . . 160
1.13.42.7 MediaLibrary.Search . . . 160
1.13.42.8 MediaLibrary.SetPrepareSearchFiles . . . 160
1.13.42.9 MediaLibrary.SetPrepareSearchThumbnails . . . 160
1.13.42.10 MediaLibrary.SetResultsToInclude . . . 160
1.13.42.11 MediaLibrary.StartSearch . . . 160
1.13.42.12 MediaLibrary.StopSearch . . . 160
1.13.43 MemoryContainer . . . 160
1.13.43.1 MemoryContainer.Create . . . 160
1.13.43.2 MemoryContainer.DebugLoad . . . 160
1.13.43.3 MemoryContainer.DebugPrint . . . 160
1.13.43.4 MemoryContainer.DebugSave . . . 160
1.13.43.5 MemoryContainer.GetBigInt . . . 160
1.13.43.6 MemoryContainer.GetBit . . . 160
1.13.43.7 MemoryContainer.GetBits . . . 160
1.13.43.8 MemoryContainer.GetFloat16 . . . 160
1.13.43.9 MemoryContainer.GetFloat32 . . . 160
1.13.43.10 MemoryContainer.GetInt8 . . . 160
1.13.43.11 MemoryContainer.GetInt16 . . . 161
1.13.43.12 MemoryContainer.GetSize . . . 161
1.13.43.13 MemoryContainer.GetString . . . 161
1.13.43.14 MemoryContainer.GetUInt8 . . . 161
1.13.43.15 MemoryContainer.GetUInt16 . . . 161
1.13.43.16 MemoryContainer.GetUsedSize . . . 161
1.13.43.17 MemoryContainer.MemCopy . . . 161
1.13.43.18 MemoryContainer.MemSet . . . 161
1.13.43.19 MemoryContainer.ReadBigInt . . . 161
1.13.43.20 MemoryContainer.ReadFloat16 . . . 161
1.13.43.21 MemoryContainer.ReadFloat32 . . . 161
1.13.43.22 MemoryContainer.ReadInt8 . . . 161
1.13.43.23 MemoryContainer.ReadInt16 . . . 161
1.13.43.24 MemoryContainer.ReadString . . . 161
1.13.43.25 MemoryContainer.ReadUInt8 . . . 161
1.13.43.26 MemoryContainer.ReadUInt16 . . . 161
1.13.43.27 MemoryContainer.Reset . . . 161
1.13.43.28 MemoryContainer.Seek . . . 161
1.13.43.29 MemoryContainer.SetBigInt . . . 162
1.13.43.30 MemoryContainer.SetBit . . . 162
1.13.43.31 MemoryContainer.SetBits . . . 162
1.13.43.32 MemoryContainer.SetFloat16 . . . 162
1.13.43.33 MemoryContainer.SetFloat32 . . . 162
1.13.43.34 MemoryContainer.SetInt8 . . . 162
1.13.43.35 MemoryContainer.SetInt16 . . . 162
1.13.43.36 MemoryContainer.SetString . . . 162
1.13.43.37 MemoryContainer.SetUsedSize . . . 162
1.13.43.38 MemoryContainer.Tell . . . 162
1.13.43.39 MemoryContainer.WriteBigInt . . . 162
1.13.43.40 MemoryContainer.WriteFloat16 . . . 162
1.13.43.41 MemoryContainer.WriteFloat32 . . . 162
1.13.43.42 MemoryContainer.WriteInt8 . . . 162
1.13.43.43 MemoryContainer.WriteInt16 . . . 162
1.13.43.44 MemoryContainer.WriteString . . . 162
1.13.44 MiniGame . . . 162
1.13.44.1 MiniGame.AddCustomExitMenuOption . . . 162
1.13.44.2 MiniGame.CloseSession . . . 163
1.13.44.3 MiniGame.ConfirmReceivedWelcomeMessage . . . 163
1.13.44.4 MiniGame.DisableWelcomeMessages . . . 163
1.13.44.5 MiniGame.EnableQueue . . . 163
1.13.44.6 MiniGame.GetCurrentFocusPersonId . . . 163
1.13.44.7 MiniGame.GetCurrentFocusPlayerSlot . . . 163
1.13.44.8 MiniGame.GetJoinType . . . 163
1.13.44.9 MiniGame.GetMaxPlayerSlots . . . 163
1.13.44.10 MiniGame.GetNumberOfUsedPlayerSlots . . . 163
1.13.44.11 MiniGame.GetPersonIdFromPlayerSlot . . . 163
1.13.44.12 MiniGame.GetPlayerSlotUsedByPersonId . . . 163
1.13.44.13 MiniGame.GetQueueSize . . . 163
1.13.44.14 MiniGame.GetTriggerRadius . . . 163
1.13.44.15 MiniGame.IsAbleToChangeCustomExitMenuOption . . . 163
1.13.44.16 MiniGame.IsPersonJoined . . . 163
1.13.44.17 MiniGame.IsSessionOpen . . . 163
1.13.44.18 MiniGame.IsSessionOwner . . . 163
1.13.44.19 MiniGame.LocalPlayerLeaveGame . . . 163
1.13.44.20 MiniGame.OpenSession . . . 164
1.13.44.21 MiniGame.RemoveCustomExitMenuOption . . . 164
1.13.44.22 MiniGame.SetJoinType . . . 164
1.13.44.23 MiniGame.SetLocalPlayerExcludeMessage . . . 164
1.13.44.24 MiniGame.SetSafeToSendWelcomeMessage . . . 164

1.13.45 NetPropertyBag . . . 164
1.13.45.1 NetPropertyBag.Create . . . 164
1.13.45.2 NetPropertyBag.DeleteFromNet . . . 164
1.13.45.3 NetPropertyBag.GetBagIndex . . . 164
1.13.45.4 NetPropertyBag.GetEventBag . . . 164
1.13.45.5 NetPropertyBag.GetField . . . 164
1.13.45.6 NetPropertyBag.GetOwnerId . . . 164
1.13.45.7 NetPropertyBag.GetTypeId . . . 164
1.13.45.8 NetPropertyBag.GetTypeIdForBag . . . 164
1.13.45.9 NetPropertyBag.IsOwnedLocally . . . 164
1.13.45.10 NetPropertyBag.IsReplicated . . . 164
1.13.45.11 NetPropertyBag.ReplicateToNet . . . 164
1.13.45.12 NetPropertyBag.SetField . . . 164
1.13.46 Object . . . 165
1.13.46.1 Object.DebugCreate . . . 165
1.13.46.2 Object.DebugDestroy . . . 165
1.13.46.3 Object.DebugEnableDebugRendering . . . 165
1.13.46.4 Object.DebugFind . . . 165
1.13.46.5 Object.DebugGetAllInstanceIds . . . 165
1.13.46.6 Object.DebugReport . . . 165
1.13.46.7 Object.DebugRun . . . 165
1.13.46.8 Object.EnableTargeting . . . 165
1.13.46.9 Object.GetCurrentListenerData . . . 165
1.13.46.10 Object.GetInitialPosition . . . 165
1.13.46.11 Object.GetInitialRotation . . . 165
1.13.46.12 Object.GetInitialScale . . . 165
1.13.46.13 Object.GetInstanceId . . . 165
1.13.46.14 Object.GetInstanceName . . . 165
1.13.46.15 Object.GetInstanceParameter . . . 165
1.13.46.16 Object.GetListenerObjects . . . 165
1.13.46.17 Object.GetLocalizedText . . . 165
1.13.46.18 Object.GetMe . . . 165
1.13.46.19 Object.GetNameRef . . . 166
1.13.46.20 Object.GetObjectId . . . 166
1.13.46.21 Object.GetPersonId . . . 166
1.13.46.22 Object.HasFocus . . . 166
1.13.46.23 Object.HasLocalizedText . . . 166
1.13.46.24 Object.IsOwnedLocally . . . 166
1.13.46.25 Object.IsPaused . . . 166
1.13.46.26 Object.Pause . . . 166
1.13.46.27 Object.Resume . . . 166
1.13.46.28 Object.SendListenerData . . . 166
1.13.46.29 Object.SetFaceActiveTarget . . . 166
1.13.46.30 Object.SetTargetableLegend . . . 166
1.13.46.31 Object.SetTargetablePosition . . . 166
1.13.46.32 Object.SetTargetableText . . . 166
1.13.47 Osd . . . 166
1.13.47.1 Osd.GetRoot . . . 166
1.13.48 OsdBasicAnimGraphic . . . 166
1.13.48.1 OsdBasicAnimGraphic.AddFrame . . . 166
1.13.48.2 OsdBasicAnimGraphic.GetNumFrames . . . 166
1.13.48.3 OsdBasicAnimGraphic.IsAnimating . . . 166
1.13.48.4 OsdBasicAnimGraphic.SetAnimating . . . 167
1.13.48.5 OsdBasicAnimGraphic.SetCurrentFrame . . . 167
1.13.48.6 OsdBasicAnimGraphic.SetFrameRate . . . 167
1.13.49 OsdBasicChip . . . 167
1.13.49.1 OsdBasicChip.AddLegendItem . . . 167
1.13.49.2 OsdBasicChip.CreateContentObject . . . 167
1.13.49.3 OsdBasicChip.RemoveAllLegendItems . . . 167
1.13.49.4 OsdBasicChip.SetBannerHeadline . . . 167
1.13.49.5 OsdBasicChip.SetBannerIconBackTexture . . . 167
1.13.49.6 OsdBasicChip.SetBannerIconFrontTexture . . . 167
1.13.49.7 OsdBasicChip.SetBannerIconVisible . . . 167
1.13.49.8 OsdBasicChip.SetBannerSubheader . . . 167
1.13.49.9 OsdBasicChip.SetBannerWantVisible . . . 167
1.13.49.10 OsdBasicChip.SetContentHeight . . . 167
1.13.49.11 OsdBasicChip.SetContentHeightSetFromChildren . . . 167
1.13.49.12 OsdBasicChip.SetContentHeightSetFromChip . . . 167
1.13.49.13 OsdBasicChip.SetContentObjectInFocus . . . 167
1.13.49.14 OsdBasicChip.SetContentPointInFocus . . . 167
1.13.49.15 OsdBasicChip.SetContentWidth . . . 167
1.13.49.16 OsdBasicChip.SetContentWidthSetFromChildren . . . 168
1.13.49.17 OsdBasicChip.SetContentWidthSetFromChip . . . 168
1.13.49.18 OsdBasicChip.SetHeightSetFromContent . . . 168
1.13.49.19 OsdBasicChip.SetSize . . . 168
1.13.49.20 OsdBasicChip.UpdateContentHeight . . . 168
1.13.49.21 OsdBasicChip.UpdateContentWidth . . . 168
1.13.50 OsdBasicLegend . . . 168
1.13.50.1 OsdBasicLegend.AddItem . . . 168

1.13.50.2 OsdBasicLegend.BeginAddingItems . . . 168
1.13.50.3 OsdBasicLegend.EndAddingItems . . . 168
1.13.50.4 OsdBasicLegend.RemoveAllItems . . . 168
1.13.50.5 OsdBasicLegend.SetBorderWidth . . . 168
1.13.50.6 OsdBasicLegend.SetFixedWidth . . . 168
1.13.50.7 OsdBasicLegend.SetItemSpacing . . . 168
1.13.50.8 OsdBasicLegend.SetMaxWidth . . . 168
1.13.50.9 OsdBasicLegend.SetMinWidth . . . 168
1.13.50.10 OsdBasicLegend.SetText . . . 168
1.13.51 OsdBasicMenu . . . 168
1.13.51.1 OsdBasicMenu.AddItem . . . 168
1.13.51.2 OsdBasicMenu.BeginChanges . . . 169
1.13.51.3 OsdBasicMenu.EndChanges . . . 169
1.13.51.4 OsdBasicMenu.GetCurrentItemId . . . 169
1.13.51.5 OsdBasicMenu.GetItemById . . . 169
1.13.51.6 OsdBasicMenu.IsCurrentItemSet . . . 169
1.13.51.7 OsdBasicMenu.OnInputDown . . . 169
1.13.51.8 OsdBasicMenu.OnInputUp . . . 169
1.13.51.9 OsdBasicMenu.Reset . . . 169
1.13.51.10 OsdBasicMenu.SetCurrentItem . . . 169
1.13.51.11 OsdBasicMenu.SetItemEnabled . . . 169
1.13.51.12 OsdBasicMenu.SetItemSpacing . . . 169
1.13.51.13 OsdBasicMenu.SetOnSetItemEnabledCallback . . . 169
1.13.51.14 OsdBasicMenu.SetOnSetItemSelectedCallback . . . 169
1.13.52 OsdBasicPopup . . . 169
1.13.52.1 OsdBasicPopup.SetMessage . . . 169
1.13.52.2 OsdBasicPopup.SetTitle . . . 169
1.13.53 OsdBasicSingleGraphic . . . 170
1.13.53.1 OsdBasicSingleGraphic.SetBackColorType . . . 170
1.13.53.2 OsdBasicSingleGraphic.SetBackTexture . . . 170
1.13.53.3 OsdBasicSingleGraphic.SetFrontColorType . . . 170
1.13.53.4 OsdBasicSingleGraphic.SetFrontTexture . . . 170
1.13.54 OsdBasicText . . . 170
1.13.54.1 OsdBasicText.GetText . . . 170
1.13.54.2 OsdBasicText.SetClipping . . . 170
1.13.54.3 OsdBasicText.SetFont . . . 170
1.13.54.4 OsdBasicText.SetIgnoreSpecialCharacters . . . 170
1.13.54.5 OsdBasicText.SetScrolling . . . 170
1.13.54.6 OsdBasicText.SetText . . . 170
1.13.55 OsdObject . . . 170
1.13.55.1 OsdObject.Activate . . . 170
1.13.55.2 OsdObject.CreateChildObject . . . 170
1.13.55.3 OsdObject.Deactivate . . . 170
1.13.55.4 OsdObject.DestroyObject . . . 170
1.13.55.5 OsdObject.FindObject . . . 170
1.13.55.6 OsdObject.GetColorValue . . . 170
1.13.55.7 OsdObject.GetHeight . . . 170
1.13.55.8 OsdObject.GetPositionX . . . 170
1.13.55.9 OsdObject.GetPositionY . . . 171
1.13.55.10 OsdObject.GetUniqueId . . . 171
1.13.55.11 OsdObject.GetWidth . . . 171
1.13.55.12 OsdObject.HandleInput . . . 171
1.13.55.13 OsdObject.IsActive . . . 171
1.13.55.14 OsdObject.SetAlphaType . . . 171
1.13.55.15 OsdObject.SetAlphaValue . . . 171
1.13.55.16 OsdObject.SetAngle . . . 171
1.13.55.17 OsdObject.SetColorType . . . 171
1.13.55.18 OsdObject.SetColorValue . . . 171
1.13.55.19 OsdObject.SetHeight . . . 171
1.13.55.20 OsdObject.SetHeightFromChildren . . . 171
1.13.55.21 OsdObject.SetLocalSpaceAlignPoint . . . 171
1.13.55.22 OsdObject.SetName . . . 171
1.13.55.23 OsdObject.SetParentSpaceAlignPoint . . . 171
1.13.55.24 OsdObject.SetPosition . . . 171
1.13.55.25 OsdObject.SetScale . . . 171
1.13.55.26 OsdObject.SetUniqueId . . . 171
1.13.55.27 OsdObject.SetVisible . . . 172
1.13.55.28 OsdObject.SetWidth . . . 172
1.13.55.29 OsdObject.SetWidthFromChildren . . . 172
1.13.56 OsdSelectionParent . . . 172
1.13.56.1 OsdSelectionParent.BeginChanges . . . 172
1.13.56.2 OsdSelectionParent.EndChanges . . . 172
1.13.56.3 OsdSelectionParent.GetCurrentSelectionId . . . 172
1.13.56.4 OsdSelectionParent.SetObjectEnabled . . . 172
1.13.57 OsdSimpleMenu . . . 172
1.13.57.1 OsdSimpleMenu.AddItem . . . 172
1.13.57.2 OsdSimpleMenu.SetItemTitle . . . 172
1.13.58 OsdSkinnedBox . . . 172
1.13.58.1 OsdSkinnedBox.ResetTextures . . . 172

1.13.58.2 OsdSkinnedBox.SetColorGradientEnabled . . . 172
1.13.58.3 OsdSkinnedBox.SetColorGradientMagnitude . . . 172
1.13.58.4 OsdSkinnedBox.SetColorGradientReversed . . . 172
1.13.58.5 OsdSkinnedBox.SetColorGradientSubtractive . . . 172
1.13.58.6 OsdSkinnedBox.SetTexture . . . 172
1.13.58.7 OsdSkinnedBox.SetTextures . . . 172
1.13.59 OsdTextLines . . . 173
1.13.59.1 OsdTextLines.GetNumTextLines . . . 173
1.13.59.2 OsdTextLines.SetFont . . . 173
1.13.59.3 OsdTextLines.SetJustification . . . 173
1.13.59.4 OsdTextLines.SetLineSpacing . . . 173
1.13.59.5 OsdTextLines.SetText . . . 173
1.13.60 OsdTextPanel . . . 173
1.13.60.1 OsdTextPanel.GetTextBorder . . . 173
1.13.60.2 OsdTextPanel.GetTextLines . . . 173
1.13.60.3 OsdTextPanel.IsFitHeightToText . . . 173
1.13.60.4 OsdTextPanel.IsPanelVisible . . . 173
1.13.60.5 OsdTextPanel.IsScrollBarVisible . . . 173
1.13.60.6 OsdTextPanel.SetFitHeightToText . . . 173
1.13.60.7 OsdTextPanel.SetFont . . . 173
1.13.60.8 OsdTextPanel.SetJustification . . . 173
1.13.60.9 OsdTextPanel.SetLineSpacing . . . 173
1.13.60.10 OsdTextPanel.SetPanelVisible . . . 173
1.13.60.11 OsdTextPanel.SetScrollBarVisible . . . 173
1.13.60.12 OsdTextPanel.SetScrollType . . . 173
1.13.60.13 OsdTextPanel.SetText . . . 173
1.13.60.14 OsdTextPanel.SetTextBorder . . . 174
1.13.60.15 OsdTextPanel.SetTextColorType . . . 174
1.13.61 Osk . . . 174
1.13.61.1 Osk.Close . . . 174
1.13.61.2 Osk.GetText . . . 174
1.13.61.3 Osk.IsAvailable . . . 174
1.13.61.4 Osk.Open . . . 174
1.13.62 OutboundMessage . . . 174
1.13.62.1 OutboundMessage.Create . . . 174
1.13.62.2 OutboundMessage.GetRemainingBufferSize . . . 174
1.13.62.3 OutboundMessage.Reset . . . 174
1.13.62.4 OutboundMessage.Send . . . 174
1.13.62.5 OutboundMessage.SendUnguaranteed . . . 174
1.13.62.6 OutboundMessage.WriteBigInt . . . 174
1.13.62.7 OutboundMessage.WriteFloat . . . 174
1.13.62.8 OutboundMessage.WriteInt8 . . . 174
1.13.62.9 OutboundMessage.WriteInt16 . . . 174
1.13.62.10 OutboundMessage.WriteInt32 . . . 174
1.13.62.11 OutboundMessage.WriteString . . . 174
1.13.63 Pad . . . 175
1.13.63.1 Pad.CreatePads . . . 175
1.13.63.2 Pad.GetAnalogExtent . . . 175
1.13.63.3 Pad.GetPad . . . 175
1.13.63.4 Pad.GetSensorForwards . . . 175
1.13.63.5 Pad.GetSensorRight . . . 175
1.13.63.6 Pad.GetSensorUp . . . 175
1.13.63.7 Pad.IsCircleAccept . . . 175
1.13.63.8 Pad.IsConnected . . . 175
1.13.63.9 Pad.IsHeld . . . 175
1.13.63.10 Pad.Reserve . . . 175
1.13.63.11 Pad.SetActuator . . . 175
1.13.63.12 Pad.UnReserve . . . 175
1.13.63.13 Pad.UnReserveAll . . . 175
1.13.63.14 Pad.WasAnyJustPressed . . . 175
1.13.63.15 Pad.WasJustPressed . . . 175
1.13.63.16 Pad.WasJustReleased . . . 176
1.13.64 Person . . . 176
1.13.64.1 Person.AttachToEntity . . . 176
1.13.64.2 Person.DoAction . . . 176
1.13.64.3 Person.DoActionBroadcast . . . 176
1.13.64.4 Person.EnableChatBubbles . . . 176
1.13.64.5 Person.EnableCollision . . . 176
1.13.64.6 Person.FindInInstance . . . 176
1.13.64.7 Person.GetAnimReg . . . 176
1.13.64.8 Person.GetAvatarStatusType . . . 176
1.13.64.9 Person.GetBehavior . . . 176
1.13.64.10 Person.GetBehaviorCategory . . . 176
1.13.64.11 Person.GetBoneCount . . . 176
1.13.64.12 Person.GetBoneIndex . . . 176
1.13.64.13 Person.GetBoneLocalMatrix . . . 176
1.13.64.14 Person.GetBoneWorldMatrix . . . 176
1.13.64.15 Person.GetCurrentAction . . . 176
1.13.64.16 Person.GetGroundUserId . . . 177

1.13.64.17 Person.GetHeight . . . 177
1.13.64.18 Person.GetId . . . 177
1.13.64.19 Person.GetIdsOfFriendsInInstance . . . 177
1.13.64.20 Person.GetIdsOfRemotePeopleInInstance . . . 177
1.13.64.21 Person.GetInstanceSessionMasterId . . . 177
1.13.64.22 Person.GetInventoryItemObjectId . . . 177
1.13.64.23 Person.GetLocalBounds . . . 177
1.13.64.24 Person.GetMood . . . 177
1.13.64.25 Person.GetName . . . 177
1.13.64.26 Person.GetPosition . . . 177
1.13.64.27 Person.GetRigComponentObjectId . . . 177
1.13.64.28 Person.GetRigComponentTypes . . . 177
1.13.64.29 Person.GetRigField . . . 177
1.13.64.30 Person.GetRotationY . . . 177
1.13.64.31 Person.GetUser . . . 177
1.13.64.32 Person.GetVelocity . . . 177
1.13.64.33 Person.GetWidth . . . 177
1.13.64.34 Person.GetWorldBounds . . . 178
1.13.64.35 Person.HasLock . . . 178
1.13.64.36 Person.InterpolatePosition . . . 178
1.13.64.37 Person.IsCollision . . . 178
1.13.64.38 Person.IsLoaded . . . 178
1.13.64.39 Person.IsLocked . . . 178
1.13.64.40 Person.IsOnGround . . . 178
1.13.64.41 Person.IsRigUsingComponent . . . 178
1.13.64.42 Person.IsValid . . . 178
1.13.64.43 Person.IsVisible . . . 178
1.13.64.44 Person.Lock . . . 178
1.13.64.45 Person.ResetCurrentStatusText . . . 178
1.13.64.46 Person.SetAnimReg . . . 178
1.13.64.47 Person.SetBehavior . . . 178
1.13.64.48 Person.SetCurrentStatusText . . . 178
1.13.64.49 Person.SetLabelText . . . 178
1.13.64.50 Person.SetPosition . . . 179
1.13.64.51 Person.SetRotationY . . . 179
1.13.64.52 Person.SetVisible . . . 179
1.13.64.53 Person.Unlock . . . 179
1.13.65 Phys3dWorld . . . 179
1.13.65.1 Phys3dWorld.Create . . . 179
1.13.65.2 Phys3dWorld.Reset . . . 179
1.13.65.3 Phys3dWorld.SetEnabled . . . 179
1.13.66 Quaternion . . . 179
1.13.66.1 Quaternion.Add . . . 179
1.13.66.2 Quaternion.Conjugate . . . 179
1.13.66.3 Quaternion.Copy . . . 179
1.13.66.4 Quaternion.Create . . . 179
1.13.66.5 Quaternion.Divide . . . 179
1.13.66.6 Quaternion.Dot . . . 179
1.13.66.7 Quaternion.Length . . . 179
1.13.66.8 Quaternion.Multiply . . . 179
1.13.66.9 Quaternion.Negate . . . 179
1.13.66.10 Quaternion.Nlerp . . . 179
1.13.66.11 Quaternion.Normalize . . . 180
1.13.66.12 Quaternion.SetAxisAngle . . . 180
1.13.66.13 Quaternion.SetIdentity . . . 180
1.13.66.14 Quaternion.SetRotationMatrix . . . 180
1.13.66.15 Quaternion.SetW . . . 180
1.13.66.16 Quaternion.SetX . . . 180
1.13.66.17 Quaternion.SetXyzw . . . 180
1.13.66.18 Quaternion.SetY . . . 180
1.13.66.19 Quaternion.SetZ . . . 180
1.13.66.20 Quaternion.Slerp . . . 180
1.13.66.21 Quaternion.Subtract . . . 180
1.13.66.22 Quaternion.ToAxisAngle . . . 180
1.13.66.23 Quaternion.W . . . 180
1.13.66.24 Quaternion.X . . . 180
1.13.66.25 Quaternion.Y . . . 180
1.13.66.26 Quaternion.Z . . . 180
1.13.67 Raycast . . . 180
1.13.67.1 Raycast.GetEndPoint . . . 180
1.13.67.2 Raycast.GetHitEntity . . . 181
1.13.67.3 Raycast.GetHitPoint . . . 181
1.13.67.4 Raycast.GetReflectedVector . . . 181
1.13.67.5 Raycast.GetStartPoint . . . 181
1.13.67.6 Raycast.GetSurfaceNormal . . . 181
1.13.67.7 Raycast.GetSurfaceUserId . . . 181
1.13.67.8 Raycast.HasHit . . . 181
1.13.67.9 Raycast.IsReady . . . 181
1.13.67.10 Raycast.Start . . . 181

1.13.68 ReceivedMessage . . . 181
1.13.68.1 ReceivedMessage.GetEventMessage . . . 181
1.13.68.2 ReceivedMessage.GetRemainingSize . . . 181
1.13.68.3 ReceivedMessage.GetSenderPersonId . . . 182
1.13.68.4 ReceivedMessage.GetTotalSize . . . 182
1.13.68.5 ReceivedMessage.GetType . . . 182
1.13.68.6 ReceivedMessage.ReadBigInt . . . 182
1.13.68.7 ReceivedMessage.ReadFloat . . . 182
1.13.68.8 ReceivedMessage.ReadInt8 . . . 182
1.13.68.9 ReceivedMessage.ReadInt16 . . . 182
1.13.68.10 ReceivedMessage.ReadInt32 . . . 182
1.13.68.11 ReceivedMessage.ReadString . . . 182
1.13.69 Renderer . . . 182
1.13.69.1 Renderer.Camera2dGetPosition . . . 182
1.13.69.2 Renderer.Camera2dGetRotation . . . 182
1.13.69.3 Renderer.Camera2dGetScale . . . 182
1.13.69.4 Renderer.Camera2dSetPosition . . . 182
1.13.69.5 Renderer.Camera2dSetRotation . . . 182
1.13.69.6 Renderer.Camera2dSetScale . . . 182
1.13.69.7 Renderer.Create . . . 182
1.13.69.8 Renderer.DrawCircle2d . . . 183
1.13.69.9 Renderer.DrawLine2d . . . 183
1.13.69.10 Renderer.DrawLineList2d . . . 183
1.13.69.11 Renderer.DrawQuad2d . . . 183
1.13.69.12 Renderer.DrawQuads2d . . . 183
1.13.69.13 Renderer.DrawRect2d . . . 183
1.13.69.14 Renderer.DrawSprite . . . 183
1.13.69.15 Renderer.DrawSprites . . . 183
1.13.69.16 Renderer.DrawText2d . . . 183
1.13.69.17 Renderer.DrawTriList2d . . . 183
1.13.69.18 Renderer.GetOverlay . . . 183
1.13.69.19 Renderer.GetPosition . . . 183
1.13.69.20 Renderer.GetScale . . . 183
1.13.69.21 Renderer.GetSize . . . 183
1.13.69.22 Renderer.GetTextSize . . . 183
1.13.69.23 Renderer.SetBlendMode2d . . . 183
1.13.69.24 Renderer.SetFont . . . 183
1.13.69.25 Renderer.SetFontHorzAlignment . . . 184
1.13.69.26 Renderer.SetFontScale . . . 184
1.13.69.27 Renderer.SetFontVertAlignment . . . 184
1.13.69.28 Renderer.SetOverlay . . . 184
1.13.69.29 Renderer.SetPosition . . . 184
1.13.69.30 Renderer.SetScale . . . 184
1.13.69.31 Renderer.SetSize . . . 184
1.13.69.32 Renderer.SetTarget . . . 184
1.13.70 Resource . . . 184
1.13.70.1 Resource.CancelRequest . . . 184
1.13.70.2 Resource.DebugDumpToTty . . . 184
1.13.70.3 Resource.DebugSave . . . 184
1.13.70.4 Resource.Exists . . . 184
1.13.70.5 Resource.Find . . . 184
1.13.70.6 Resource.GetData . . . 184
1.13.70.7 Resource.GetDownloadProgress . . . 184
1.13.70.8 Resource.GetName . . . 184
1.13.70.9 Resource.GetStatusCode . . . 184
1.13.70.10 Resource.GetSystemResource . . . 185
1.13.70.11 Resource.IsLoaded . . . 185
1.13.70.12 Resource.IsLoading . . . 185
1.13.70.13 Resource.Release . . . 185
1.13.70.14 Resource.Request . . . 185
1.13.70.15 Resource.RequestSecure . . . 185
1.13.70.16 Resource.RequestTexture . . . 185
1.13.70.17 Resource.Run . . . 185
1.13.70.18 Resource.StartLoading . . . 185
1.13.71 ResourcePack . . . 185
1.13.71.1 ResourcePack.DebugGetLoadError . . . 185
1.13.71.2 ResourcePack.GetLoadProgress . . . 185
1.13.71.3 ResourcePack.GetLoadState . . . 185
1.13.71.4 ResourcePack.IsLoading . . . 185
1.13.71.5 ResourcePack.Load . . . 185
1.13.71.6 ResourcePack.Unload . . . 185
1.13.72 Rewards . . . 186
1.13.72.1 Rewards.AddTicket . . . 186
1.13.72.2 Rewards.CollectTicket . . . 186
1.13.72.3 Rewards.DoesHaveTicket . . . 186
1.13.72.4 Rewards.GetCurrentFocusObjectId . . . 186
1.13.72.5 Rewards.GetCurrentFocusState . . . 186
1.13.73 RtGame . . . 186
1.13.73.1 RtGame.AddCustomExitMenuOption . . . 186

1.13.73.2 RtGame.CreateGame . . . 186
1.13.73.3 RtGame.CreateGameObject . . . 186
1.13.73.4 RtGame.DestroyGameObject . . . 186
1.13.73.5 RtGame.GetConnectionInfo . . . 186
1.13.73.6 RtGame.GetMaxPlayers . . . 186
1.13.73.7 RtGame.GetMemberIdFromUser . . . 186
1.13.73.8 RtGame.GetNumTokens . . . 186
1.13.73.9 RtGame.GetOwnerMemberId . . . 186
1.13.73.10 RtGame.GetPlayer . . . 186
1.13.73.11 RtGame.GetPlayerCount . . . 186
1.13.73.12 RtGame.GetRoomId . . . 187
1.13.73.13 RtGame.GetSentInstanceTime . . . 187
1.13.73.14 RtGame.GetTokenOwnerMemberId . . . 187
1.13.73.15 RtGame.GetUserFromMemberId . . . 187
1.13.73.16 RtGame.IsAbleToChangeCustomExitMenuOption . . . 187
1.13.73.17 RtGame.IsJoined . . . 187
1.13.73.18 RtGame.IsSessionOpen . . . 187
1.13.73.19 RtGame.JoinGame . . . 187
1.13.73.20 RtGame.Leave . . . 187
1.13.73.21 RtGame.ReleaseToken . . . 187
1.13.73.22 RtGame.RemoveCustomExitMenuOption . . . 187
1.13.73.23 RtGame.ReplicateToNet . . . 187
1.13.73.24 RtGame.RequestToken . . . 187
1.13.73.25 RtGame.ReturnJoinGameResult . . . 187
1.13.73.26 RtGame.SendMessage . . . 187
1.13.73.27 RtGame.SetSessionClosed . . . 187
1.13.74 RtGameObject . . . 187
1.13.74.1 RtGameObject.GetInstanceId . . . 187
1.13.74.2 RtGameObject.GetName . . . 188
1.13.74.3 RtGameObject.GetOwnerMemberId . . . 188
1.13.74.4 RtGameObject.GetOwnershipType . . . 188
1.13.75 SaveData . . . 188
1.13.75.1 SaveData.Create . . . 188
1.13.75.2 SaveData.GetState . . . 188
1.13.75.3 SaveData.StartLoading . . . 188
1.13.75.4 SaveData.StartSaving . . . 188
1.13.76 Scene . . . 188
1.13.76.1 Scene.FindEntity . . . 188
1.13.76.2 Scene.FindScreen . . . 188
1.13.76.3 Scene.FindTriggerVolume . . . 188
1.13.76.4 Scene.GetGraphicsEngineParam . . . 188
1.13.76.5 Scene.GetInstanceId . . . 188
1.13.76.6 Scene.GetLaunchParam . . . 188
1.13.76.7 Scene.GetLayerCollisionDefault . . . 188
1.13.76.8 Scene.GetListenerObjects . . . 188
1.13.76.9 Scene.GetMaxSaveDataSize . . . 188
1.13.76.10 Scene.GetSaveDataPermission . . . 188
1.13.76.11 Scene.GetStartupInstanceParam . . . 189
1.13.76.12 Scene.GetStartupValueParam . . . 189
1.13.76.13 Scene.GetVirtualObject . . . 189
1.13.76.14 Scene.IsLoading . . . 189
1.13.76.15 Scene.IsLoadingObjects . . . 189
1.13.76.16 Scene.IsSaveDataBusy . . . 189
1.13.76.17 Scene.LoadData . . . 189
1.13.76.18 Scene.LockPadControls . . . 189
1.13.76.19 Scene.OverrideAvatarCloseUpCamAspect . . . 189
1.13.76.20 Scene.OverrideAvatarCloseUpCamFov . . . 189
1.13.76.21 Scene.OverrideAvatarCloseUpCamZFar . . . 189
1.13.76.22 Scene.OverrideAvatarCloseUpCamZNear . . . 189
1.13.76.23 Scene.OverrideAvatarStandardCamAspect . . . 189
1.13.76.24 Scene.OverrideAvatarStandardCamFov . . . 189
1.13.76.25 Scene.OverrideAvatarStandardCamZFar . . . 189
1.13.76.26 Scene.OverrideAvatarStandardCamZNear . . . 189
1.13.76.27 Scene.ResetAvatarCamOverrides . . . 190
1.13.76.28 Scene.SaveData . . . 190
1.13.76.29 Scene.SetGraphicsEngineParam . . . 190
1.13.76.30 Scene.SetLayerCollisionDefault . . . 190
1.13.76.31 Scene.SetLightProbeData . . . 190
1.13.76.32 Scene.SetLightProbeTransform . . . 190
1.13.77 SceneDownload . . . 190
1.13.77.1 SceneDownload.CancelDownload . . . 190
1.13.77.2 SceneDownload.GetCurrentDownload . . . 190
1.13.77.3 SceneDownload.GetDownloadProgress . . . 190
1.13.77.4 SceneDownload.GetDownloadSize . . . 190
1.13.77.5 SceneDownload.QuerySceneCacheStatus . . . 190
1.13.77.6 SceneDownload.StartDownload . . . 190
1.13.78 SceneInfo . . . 190
1.13.78.1 SceneInfo.GetSceneId . . . 190
1.13.78.2 SceneInfo.GetSceneName . . . 190

1.13.78.3 SceneInfo.GetUniqueInstanceId . . . 190
1.13.79 SceneTransition . . . 190
1.13.79.1 SceneTransition.ExitScript . . . 190
1.13.79.2 SceneTransition.GetDestinationScene . . . 190
1.13.79.3 SceneTransition.GetObject . . . 191
1.13.79.4 SceneTransition.GetOriginScene . . . 191
1.13.79.5 SceneTransition.GetSceneChangeProgress . . . 191
1.13.79.6 SceneTransition.IsSceneChangeComplete . . . 191
1.13.79.7 SceneTransition.StartSceneChange . . . 191
1.13.80 Screen . . . 191
1.13.80.1 Screen.AreSceneSoundsMuted . . . 191
1.13.80.2 Screen.Create . . . 191
1.13.80.3 Screen.EnableBackground . . . 191
1.13.80.4 Screen.GetCenter . . . 191
1.13.80.5 Screen.GetContentCount . . . 191
1.13.80.6 Screen.GetCurrentContentIndex . . . 191
1.13.80.7 Screen.GetCurrentTime . . . 191
1.13.80.8 Screen.GetMatrix . . . 191
1.13.80.9 Screen.GetNormal . . . 191
1.13.80.10 Screen.GetSize . . . 191
1.13.80.11 Screen.GetTotalTime . . . 191
1.13.80.12 Screen.GetTriggerPosition . . . 192
1.13.80.13 Screen.GetTriggerRadius . . . 192
1.13.80.14 Screen.GetVolume . . . 192
1.13.80.15 Screen.IsContentPlaying . . . 192
1.13.80.16 Screen.IsContentReady . . . 192
1.13.80.17 Screen.IsFogEnabled . . . 192
1.13.80.18 Screen.IsHDFullScreenModeSet . . . 192
1.13.80.19 Screen.IsInFullScreenMode . . . 192
1.13.80.20 Screen.IsProtected . . . 192
1.13.80.21 Screen.IsVisible . . . 192
1.13.80.22 Screen.Pause . . . 192
1.13.80.23 Screen.RestartStream . . . 192
1.13.80.24 Screen.Resume . . . 192
1.13.80.25 Screen.SetChannel . . . 192
1.13.80.26 Screen.SetContent . . . 192
1.13.80.27 Screen.SetCurrentContentIndex . . . 192
1.13.80.28 Screen.SetCurrentTime . . . 192
1.13.80.29 Screen.SetData . . . 193
1.13.80.30 Screen.SetFogEnabled . . . 193
1.13.80.31 Screen.SetHDFullScreenMode . . . 193
1.13.80.32 Screen.SetHighPriority . . . 193
1.13.80.33 Screen.SetMuteSceneSounds . . . 193
1.13.80.34 Screen.SetProtected . . . 193
1.13.80.35 Screen.SetTargetable . . . 193
1.13.80.36 Screen.SetTriggerPosition . . . 193
1.13.80.37 Screen.SetTriggerRadius . . . 193
1.13.80.38 Screen.SetType . . . 193
1.13.80.39 Screen.SetVisible . . . 193
1.13.80.40 Screen.SetVolume . . . 193
1.13.81 Screenshot . . . 193
1.13.81.1 Screenshot.Capture . . . 193
1.13.81.2 Screenshot.Create . . . 193
1.13.81.3 Screenshot.IsInProgress . . . 193
1.13.81.4 Screenshot.Succeeded . . . 193
1.13.82 Sound . . . 193
1.13.82.1 Sound.Create . . . 193
1.13.82.2 Sound.GetVolume . . . 194
1.13.82.3 Sound.IsPaused . . . 194
1.13.82.4 Sound.IsPlaying . . . 194
1.13.82.5 Sound.IsValid . . . 194
1.13.82.6 Sound.Pause . . . 194
1.13.82.7 Sound.Play2d . . . 194
1.13.82.8 Sound.Play3d . . . 194
1.13.82.9 Sound.Resume . . . 194
1.13.82.10 Sound.SetPosition . . . 194
1.13.82.11 Sound.SetRegister . . . 194
1.13.82.12 Sound.SetVolume . . . 194
1.13.82.13 Sound.Stop . . . 194
1.13.83 SoundBank . . . 194
1.13.83.1 SoundBank.Find . . . 194
1.13.83.2 SoundBank.Play2d . . . 194
1.13.83.3 SoundBank.Play3d . . . 194
1.13.84 SoundStream . . . 194
1.13.84.1 SoundStream.Find . . . 194
1.13.84.2 SoundStream.GetCurrentTime . . . 194
1.13.84.3 SoundStream.GetTotalTime . . . 194
1.13.84.4 SoundStream.GetVolume . . . 195
1.13.84.5 SoundStream.IsPaused . . . 195

1.13.84.6 SoundStream.IsPlaying . . . 195
1.13.84.7 SoundStream.Pause . . . 195
1.13.84.8 SoundStream.Play2d . . . 195
1.13.84.9 SoundStream.Play3d . . . 195
1.13.84.10 SoundStream.Resume . . . 195
1.13.84.11 SoundStream.SetCurrentTime . . . 195
1.13.84.12 SoundStream.SetPosition . . . 195
1.13.84.13 SoundStream.SetVolume . . . 195
1.13.84.14 SoundStream.Stop . . . 195
1.13.85 Sprite . . . 195
1.13.85.1 Sprite.Create . . . 195
1.13.85.2 Sprite.GetColor . . . 195
1.13.85.3 Sprite.GetElapsedAnimTime . . . 195
1.13.85.4 Sprite.GetPosition . . . 195
1.13.85.5 Sprite.GetRotation . . . 195
1.13.85.6 Sprite.GetScale . . . 195
1.13.85.7 Sprite.GetTotalAnimTime . . . 195
1.13.85.8 Sprite.IsAnimPaused . . . 196
1.13.85.9 Sprite.IsAnimPlaying . . . 196
1.13.85.10 Sprite.PauseAnim . . . 196
1.13.85.11 Sprite.PlayAnim . . . 196
1.13.85.12 Sprite.ResumeAnim . . . 196
1.13.85.13 Sprite.SetColor . . . 196
1.13.85.14 Sprite.SetPosition . . . 196
1.13.85.15 Sprite.SetRotation . . . 196
1.13.85.16 Sprite.SetScale . . . 196
1.13.85.17 Sprite.SetTexture . . . 196
1.13.85.18 Sprite.SetUvs . . . 196
1.13.85.19 Sprite.StopAnim . . . 196
1.13.86 SpriteAnim . . . 196
1.13.86.1 SpriteAnim.AddFrames . . . 196
1.13.86.2 SpriteAnim.Create . . . 196
1.13.86.3 SpriteAnim.IsLooping . . . 196
1.13.86.4 SpriteAnim.SetLooping . . . 196
1.13.87 StreamOps . . . 196
1.13.87.1 StreamOps.AddFloat32 . . . 196
1.13.87.2 StreamOps.BoundsVec4 . . . 197
1.13.87.3 StreamOps.DivFloat32 . . . 197
1.13.87.4 StreamOps.Float32ToUInt8 . . . 197
1.13.87.5 StreamOps.HorizAddFloat32 . . . 197
1.13.87.6 StreamOps.MulFloat32 . . . 197
1.13.87.7 StreamOps.ShufFloat32 . . . 197
1.13.87.8 StreamOps.ShufInt8 . . . 197
1.13.87.9 StreamOps.SubFloat32 . . . 197
1.13.88 System . . . 197
1.13.88.1 System.AddChatLogMessage . . . 197
1.13.88.2 System.CommercePointIsClosed . . . 197
1.13.88.3 System.CommercePointOpen . . . 197
1.13.88.4 System.DrawText2dInScene . . . 197
1.13.88.5 System.EnableIdleDisconnect . . . 197
1.13.88.6 System.ExecTitle . . . 197
1.13.88.7 System.GetChatLogMode . . . 198
1.13.88.8 System.GetConsoleTestValue . . . 198
1.13.88.9 System.GetExecTitleErrorCode . . . 198
1.13.88.10 System.GetNatType . . . 198
1.13.88.11 System.GetRegionsTerritory . . . 198
1.13.88.12 System.GetScreenBottomRight . . . 198
1.13.88.13 System.GetScreenTopLeft . . . 198
1.13.88.14 System.GetUserValue . . . 198
1.13.88.15 System.GetVideoUploadResult . . . 198
1.13.88.16 System.GetVideoUploadStatus . . . 198
1.13.88.17 System.GetVideoUploadUrl . . . 198
1.13.88.18 System.IsGameLaunchSessionActive . . . 198
1.13.88.19 System.IsLookXInverted . . . 198
1.13.88.20 System.IsLookYInverted . . . 198
1.13.88.21 System.IsVideoSystemLocked . . . 198
1.13.88.22 System.IsWideScreen . . . 198
1.13.88.23 System.PostScore . . . 198
1.13.88.24 System.ProfanityFilterText . . . 199
1.13.88.25 System.SendNpMessage . . . 199
1.13.88.26 System.SetChatLogMode . . . 199
1.13.88.27 System.UploadVideo . . . 199
1.13.88.28 System.VideoSystemLock . . . 199
1.13.88.29 System.VideoSystemUnlock . . . 199
1.13.88.30 System.WebBrowserIsClosed . . . 199
1.13.88.31 System.WebBrowserOpen . . . 199
1.13.88.32 System.WriteScriptedGameLaunchXML . . . 199
1.13.89 Texture . . . 199
1.13.89.1 Texture.Find . . . 199

1.13.89.2 Texture.GetHeight . . . 199
1.13.89.3 Texture.GetWidth . . . 199
1.13.90 Time . . . 199
1.13.90.1 Time.AddDays . . . 199
1.13.90.2 Time.AddHours . . . 199
1.13.90.3 Time.AddMinutes . . . 199
1.13.90.4 Time.AddMonths . . . 199
1.13.90.5 Time.AddSeconds . . . 200
1.13.90.6 Time.AddWeeks . . . 200
1.13.90.7 Time.AddYears . . . 200
1.13.90.8 Time.ConvertToLocal . . . 200
1.13.90.9 Time.ConvertToUtc . . . 200
1.13.90.10 Time.Copy . . . 200
1.13.90.11 Time.Create . . . 200
1.13.90.12 Time.GetAsPosixTimestamp . . . 200
1.13.90.13 Time.GetDate . . . 200
1.13.90.14 Time.GetDayOfMonth . . . 200
1.13.90.15 Time.GetDayOfWeek . . . 200
1.13.90.16 Time.GetDifference . . . 200
1.13.90.17 Time.GetDifferenceInDays . . . 200
1.13.90.18 Time.GetDifferenceInHours . . . 200
1.13.90.19 Time.GetDifferenceInMinutes . . . 200
1.13.90.20 Time.GetDifferenceInMonths . . . 200
1.13.90.21 Time.GetDifferenceInSeconds . . . 201
1.13.90.22 Time.GetDifferenceInWeeks . . . 201
1.13.90.23 Time.GetDifferenceInYears . . . 201
1.13.90.24 Time.GetHour . . . 201
1.13.90.25 Time.GetMinute . . . 201
1.13.90.26 Time.GetMonth . . . 201
1.13.90.27 Time.GetSecond . . . 201
1.13.90.28 Time.GetTime . . . 201
1.13.90.29 Time.GetYear . . . 201
1.13.90.30 Time.IsValid . . . 201
1.13.90.31 Time.SetDayOfMonth . . . 201
1.13.90.32 Time.SetFromLocal . . . 201
1.13.90.33 Time.SetFromPosixTimestamp . . . 201
1.13.90.34 Time.SetFromServer . . . 201
1.13.90.35 Time.SetHour . . . 201
1.13.90.36 Time.SetMinute . . . 201
1.13.90.37 Time.SetMonth . . . 201
1.13.90.38 Time.SetSecond . . . 201
1.13.90.39 Time.SetYear . . . 202
1.13.91 TriggerVolume . . . 202
1.13.91.1 TriggerVolume.AddCallback . . . 202
1.13.91.2 TriggerVolume.Enable . . . 202
1.13.91.3 TriggerVolume.EnableDebugRender . . . 202
1.13.91.4 TriggerVolume.EnableLayerOverlaps . . . 202
1.13.91.5 TriggerVolume.GetLocalMatrix . . . 202
1.13.91.6 TriggerVolume.GetOverlapCount . . . 202
1.13.91.7 TriggerVolume.GetOverlapData . . . 203
1.13.91.8 TriggerVolume.GetOverlapType . . . 203
1.13.91.9 TriggerVolume.GetPosition . . . 203
1.13.91.10 TriggerVolume.IsEnabled . . . 203
1.13.91.11 TriggerVolume.IsLayerOverlapsEnabled . . . 203
1.13.91.12 TriggerVolume.RemoveAllCallbacks . . . 204
1.13.91.13 TriggerVolume.RemoveCallback . . . 204
1.13.91.14 TriggerVolume.SetLocalMatrix . . . 204
1.13.91.15 TriggerVolume.SetPosition . . . 204
1.13.92 User . . . 204
1.13.92.1 User.GetName . . . 204
1.13.92.2 User.GetPerson . . . 204
1.13.92.3 User.GetSceneOwner . . . 205
1.13.92.4 User.GoToLocation . . . 205
1.13.92.5 User.InviteToLocation . . . 205
1.13.92.6 User.IsFriend . . . 205
1.13.92.7 User.IsInHome . . . 205
1.13.92.8 User.IsOnline . . . 205
1.13.93 Vector4 . . . 205
1.13.93.1 Vector4.Abs . . . 205
1.13.93.2 Vector4.Add . . . 205
1.13.93.3 Vector4.Calculate2dLineIntersection . . . 205
1.13.93.4 Vector4.Copy . . . 205
1.13.93.5 Vector4.Create . . . 205
1.13.93.6 Vector4.CreateBroadcast . . . 205
1.13.93.7 Vector4.Cross . . . 205
1.13.93.8 Vector4.Dist2Sqr . . . 205
1.13.93.9 Vector4.Dist3Sqr . . . 205
1.13.93.10 Vector4.Dist4Sqr . . . 205
1.13.93.11 Vector4.Divide . . . 205

1.13.93.12 Vector4.Dot2 . . . 206
1.13.93.13 Vector4.Dot3 . . . 206
1.13.93.14 Vector4.Dot4 . . . 206
1.13.93.15 Vector4.HorizAdd3 . . . 206
1.13.93.16 Vector4.HorizAdd4 . . . 206
1.13.93.17 Vector4.Length2 . . . 206
1.13.93.18 Vector4.Length2Sqr . . . 206
1.13.93.19 Vector4.Length3 . . . 206
1.13.93.20 Vector4.Length3Sqr . . . 206
1.13.93.21 Vector4.Length4 . . . 206
1.13.93.22 Vector4.Length4Sqr . . . 206
1.13.93.23 Vector4.Max . . . 206
1.13.93.24 Vector4.Min . . . 206
1.13.93.25 Vector4.Multiply . . . 206
1.13.93.26 Vector4.MultiplyAdd . . . 207
1.13.93.27 Vector4.Negate . . . 207
1.13.93.28 Vector4.Normal2 . . . 207
1.13.93.29 Vector4.Normal3 . . . 207
1.13.93.30 Vector4.Normal4 . . . 207
1.13.93.31 Vector4.Normalize2 . . . 207
1.13.93.32 Vector4.Normalize3 . . . 207
1.13.93.33 Vector4.Normalize4 . . . 207
1.13.93.34 Vector4.Recip . . . 207
1.13.93.35 Vector4.RecipSqrt . . . 207
1.13.93.36 Vector4.Rotate . . . 207
1.13.93.37 Vector4.SetBroadcast . . . 207
1.13.93.38 Vector4.SetW . . . 207
1.13.93.39 Vector4.SetX . . . 207
1.13.93.40 Vector4.SetXyzw . . . 207
1.13.93.41 Vector4.SetY . . . 207
1.13.93.42 Vector4.SetZ . . . 208
1.13.93.43 Vector4.Sign . . . 208
1.13.93.44 Vector4.Signum . . . 208
1.13.93.45 Vector4.Sqrt . . . 208
1.13.93.46 Vector4.Subtract . . . 208
1.13.93.47 Vector4.Transform . . . 208
1.13.93.48 Vector4.TransformPoint . . . 208
1.13.93.49 Vector4.TransformVector . . . 208
1.13.93.50 Vector4.W . . . 208
1.13.93.51 Vector4.X . . . 208
1.13.93.52 Vector4.Y . . . 208
1.13.93.53 Vector4.Z . . . 208
1.13.94 View . . . 208
1.13.94.1 View.CalculatePickRay . . . 208
1.13.94.2 View.WorldToScreen . . . 208
1.13.95 Wardrobe . . . 208
1.13.95.1 Wardrobe.ResetCommercePoints . . . 208
1.13.95.2 Wardrobe.SetCommercePoint . . . 209
1.13.96 Xml . . . 209
1.13.96.1 Xml.Create . . . 209
1.13.96.2 Xml.FindAttribute . . . 209
1.13.96.3 Xml.FindElement . . . 209
1.13.96.4 Xml.FindNextElement . . . 209
1.13.96.5 Xml.GetAttributeName . . . 209
1.13.96.6 Xml.GetAttributeValueBool . . . 209
1.13.96.7 Xml.GetAttributeValueFloat . . . 209
1.13.96.8 Xml.GetAttributeValueInt . . . 209
1.13.96.9 Xml.GetAttributeValueText . . . 209
1.13.96.10 Xml.GetElementName . . . 209
1.13.96.11 Xml.GetElementValueBool . . . 209
1.13.96.12 Xml.GetElementValueFloat . . . 209
1.13.96.13 Xml.GetElementValueInt . . . 209
1.13.96.14 Xml.GetElementValueText . . . 209
1.13.96.15 Xml.IntoElement . . . 209
1.13.96.16 Xml.OutOfElement . . . 209
1.13.96.17 Xml.SetData . . . 209
1.13.96.18 Xml.SetDataFromContainer . . . 209
1.14 HDK API Reference . . . 210
1.14.1 HdkApi.Core.Collections . . . 210
1.14.1.1 HdkApi.Core.Collections.HdkDictionary . . . 210
1.14.2 HdkApi.Core.Communication . . . 210
1.14.2.1 HdkApi.Core.Communication.Target . . . 210
1.14.2.2 HdkApi.Core.Communication.TargetException . . . 210
1.14.2.3 HdkApi.Core.Communication.TargetManager . . . 210
1.14.2.4 HdkApi.Core.Communication.TargetResult . . . 210
1.14.3 HdkApi.Core.Debug . . . 210
1.14.3.1 HdkApi.Core.Debug.Logger . . . 211
1.14.4 HdkApi.Core.Maths . . . 211
1.14.4.1 HdkApi.Core.Maths.Converters . . . 211

1.14.4.2 HdkApi.Core.Maths.Vector3 . . . 211
1.14.5 HdkApi.Core.Utilities . . . 211
1.14.5.1 HdkApi.Core.Utilities.PathUtilities . . . 211
1.14.6 HdkApi.Data.AgeRating . . . 211
1.14.6.1 HdkApi.Data.AgeRating.AgeRatingItem . . . 211
1.14.7 HdkApi.Data.Collect . . . 212
1.14.7.1 HdkApi.Data.Collect.Collect . . . 212
1.14.8 HdkApi.Data.Object . . . 212
1.14.8.1 HdkApi.Data.Object.ComponentTypes . . . 212
1.14.8.2 HdkApi.Data.Object.HomeObject . . . 212
1.14.8.3 HdkApi.Data.Object.LocalisationItem . . . 212
1.14.8.4 HdkApi.Data.Object.ObjectComponent . . . 212
1.14.8.5 HdkApi.Data.Object.ObjectManager . . . 213
1.14.8.6 HdkApi.Data.Object.ObjectResource . . . 213
1.14.8.7 HdkApi.Data.Object.TerritoryTypes . . . 213
1.14.8.8 HdkApi.Data.Object.Uuid . . . 213
1.14.9 HdkApi.Data.Object.Components . . . 213
1.14.9.1 HdkApi.Data.Object.Components.ClothingComponent . . . 213
1.14.9.2 HdkApi.Data.Object.Components.FurnitureComponent . . . 213
1.14.9.3 HdkApi.Data.Object.Components.GameLaunchComponent . . . 213
1.14.9.4 HdkApi.Data.Object.Components.HeaderComponent . . . 213
1.14.10 HdkApi.Data.Packager . . . 214
1.14.10.1 HdkApi.Data.Packager.Packager . . . 214
1.14.11 HdkApi.Data.Reporting . . . 214
1.14.11.1 HdkApi.Data.Reporting.MessageType . . . 214
1.14.12 HdkApi.Data.Scene . . . 214
1.14.12.1 HdkApi.Data.Scene.GameObject . . . 214
1.14.12.2 HdkApi.Data.Scene.GameObjectFolder . . . 214
1.14.12.3 HdkApi.Data.Scene.GameObjectTypes . . . 214
1.14.12.4 HdkApi.Data.Scene.HomeScene . . . 214
1.14.12.5 HdkApi.Data.Scene.SceneConverter . . . 214
1.14.12.6 HdkApi.Data.Scene.SceneTypes . . . 214
1.14.13 HdkApi.Data.Scene.GameObjects . . . 214
1.14.13.1 HdkApi.Data.Scene.GameObjects.GameObjectGroup . . . 215
1.14.14 HdkApi.Data.Validator . . . 215
1.14.14.1 HdkApi.Data.Validator.Validator . . . 215
1.14.15 HdkApi.ObjectEditor . . . 215
1.14.15.1 HdkApi.ObjectEditor.ObjectEditor . . . 215
1.14.16 HdkApi.ObjectEditor.UserInterface . . . 215
1.14.16.1 HdkApi.ObjectEditor.UserInterface.Menu . . . 215
1.14.16.2 HdkApi.ObjectEditor.UserInterface.MenuItem . . . 215
1.14.16.3 HdkApi.ObjectEditor.UserInterface.MenuManager . . . 215
1.14.17 HdkApi.RepertoireEditor . . . 215
1.14.17.1 HdkApi.RepertoireEditor.Action . . . 216
1.14.17.2 HdkApi.RepertoireEditor.Behaviour . . . 216
1.14.17.3 HdkApi.RepertoireEditor.BooleanRegister . . . 216
1.14.17.4 HdkApi.RepertoireEditor.EaseCurve . . . 216
1.14.17.5 HdkApi.RepertoireEditor.EaseCurveType . . . 216
1.14.17.6 HdkApi.RepertoireEditor.IntegerRegister . . . 216
1.14.17.7 HdkApi.RepertoireEditor.Locomotion . . . 216
1.14.17.8 HdkApi.RepertoireEditor.LocomotionState . . . 216
1.14.17.9 HdkApi.RepertoireEditor.NumberRegister . . . 216
1.14.17.10 HdkApi.RepertoireEditor.Repertoire . . . 216
1.14.17.11 HdkApi.RepertoireEditor.RepertoireEditor . . . 217
1.14.17.12 HdkApi.RepertoireEditor.RepertoireEvent . . . 217
1.14.17.13 HdkApi.RepertoireEditor.RepertoireRegister . . . 217
1.14.17.14 HdkApi.RepertoireEditor.RepertoireTargetRig . . . 217
1.14.17.15 HdkApi.RepertoireEditor.State . . . 217
1.14.17.16 HdkApi.RepertoireEditor.StringRegister . . . 217
1.14.17.17 HdkApi.RepertoireEditor.Transition . . . 217
1.14.18 HdkApi.RepertoireEditor.Animation . . . 217
1.14.18.1 HdkApi.RepertoireEditor.Animation.BlendNode . . . 217
1.14.18.2 HdkApi.RepertoireEditor.Animation.BlendTree . . . 217
1.14.18.3 HdkApi.RepertoireEditor.Animation.BlendTreeAnimation . . . 218
1.14.18.4 HdkApi.RepertoireEditor.Animation.SkeletalAnimation . . . 218
1.14.19 HdkApi.RepertoireEditor.Conditions . . . 218
1.14.19.1 HdkApi.RepertoireEditor.Conditions.And . . . 218
1.14.19.2 HdkApi.RepertoireEditor.Conditions.CompareCurrentBehaviour . . . 218
1.14.19.3 HdkApi.RepertoireEditor.Conditions.CompareCurrentNetworkAge . . . 218
1.14.19.4 HdkApi.RepertoireEditor.Conditions.CompareNetworkAge . . . 218
1.14.19.5 HdkApi.RepertoireEditor.Conditions.ComparePreviousBehaviour . . . 218
1.14.19.6 HdkApi.RepertoireEditor.Conditions.ComparePreviousState . . . 218
1.14.19.7 HdkApi.RepertoireEditor.Conditions.CompareRegister . . . 218
1.14.19.8 HdkApi.RepertoireEditor.Conditions.CompareStateAge . . . 218
1.14.19.9 HdkApi.RepertoireEditor.Conditions.ComparisonType . . . 218
1.14.19.10 HdkApi.RepertoireEditor.Conditions.Condition . . . 218
1.14.19.11 HdkApi.RepertoireEditor.Conditions.Or . . . 218
1.14.19.12 HdkApi.RepertoireEditor.Conditions.RandomValue . . . 218
1.14.20 HdkApi.RepertoireEditor.Emotes . . . 218

1.14.20.1 HdkApi.RepertoireEditor.Emotes.ActionEmote . . . 219
1.14.20.2 HdkApi.RepertoireEditor.Emotes.BehaviourEmote . . . 219
1.14.20.3 HdkApi.RepertoireEditor.Emotes.EmoteContext . . . 219
1.14.20.4 HdkApi.RepertoireEditor.Emotes.EmoteGroup . . . 219
1.14.20.5 HdkApi.RepertoireEditor.Emotes.GlobalEmotes . . . 219
1.14.20.6 HdkApi.RepertoireEditor.Emotes.RegionalEmotes . . . 219
1.14.21 HdkApi.RepertoireEditor.Outputs . . . 219
1.14.21.1 HdkApi.RepertoireEditor.Outputs.ClearRegister . . . 219
1.14.21.2 HdkApi.RepertoireEditor.Outputs.Output . . . 220
1.14.21.3 HdkApi.RepertoireEditor.Outputs.OutputType . . . 220
1.14.21.4 HdkApi.RepertoireEditor.Outputs.SetNextBehaviour . . . 220
1.14.21.5 HdkApi.RepertoireEditor.Outputs.SetRegister . . . 220
1.14.21.6 HdkApi.RepertoireEditor.Outputs.SpawnAction . . . 220
1.14.21.7 HdkApi.RepertoireEditor.Outputs.TriggerEvent . . . 220
1.14.22 HdkApi.RepertoireEditor.UserInterface . . . 220
1.14.22.1 HdkApi.RepertoireEditor.UserInterface.Menu . . . 220
1.14.22.2 HdkApi.RepertoireEditor.UserInterface.MenuItem . . . 220
1.14.22.3 HdkApi.RepertoireEditor.UserInterface.MenuManager . . . 221
1.14.23 HdkApi.SceneEditor . . . 221
1.14.23.1 HdkApi.SceneEditor.SceneEditor . . . 221
1.14.24 HdkApi.SceneEditor.Camera . . . 221
1.14.24.1 HdkApi.SceneEditor.Camera.Camera . . . 221
1.14.24.2 HdkApi.SceneEditor.Camera.CameraManager . . . 221
1.14.24.3 HdkApi.SceneEditor.Camera.SceneCamera . . . 221
1.14.25 HdkApi.SceneEditor.Selection . . . 222
1.14.25.1 HdkApi.SceneEditor.Selection.SelectionManager . . . 222
1.14.26 HdkApi.SceneEditor.UserInterface . . . 222
1.14.26.1 HdkApi.SceneEditor.UserInterface.Menu . . . 222
1.14.26.2 HdkApi.SceneEditor.UserInterface.MenuItem . . . 222
1.14.26.3 HdkApi.SceneEditor.UserInterface.MenuManager . . . 222
1.14.27 HdkApi.SceneEditor.Utilities . . . 222
1.14.27.1 HdkApi.SceneEditor.Utilities.SynchronizationObject . . . 222
1.15 Samples . . . 222

HDK Help

Welcome to the HDK Help

Here you will find PS Home Development Kit documentation.

HDK User Help

What's New

- What's New in this Release
- Lua API Reference
- HDK API Reference
- Recent Changes in the HDK Help Wiki

Getting Started

- HDK Overview
- Installing the HDK
- Samples

Issues

- Bug Fixes
- Known Issues

Support and Assistance

- Wiki Help
- Contact the Knowledge Management Team



Recent Changes in the HDK Help Wiki

Recently Updated

- Portable Objects
  updated by Siobhan Viviers
  (view change)
  about an hour ago

Welcome

PS Home Development Kit Documentation

Welcome to the Home Development Kit Help Wiki

In this area you can find out what's new and exciting about the latest release of the HDK, the Lua API, and the HDK API.

Read about the Known Issues and Bug Fixes in this release.



Support and Assistance

Contact the Support Team



What's New in this Release

Release 1.86

HDK 1.86 has been released and is now available to everyone on the Home Developer Network - https://home.scedev.net/.

The release includes all the tools and exporters required to create content for PlayStation®Home, as well as a developer's build of the upcoming PS Home 1.86 client.

New Features and Improvements

HDK 1.86 provides developers with 4 new Inventory Items which will allow users to interact with each other in new and fun ways, as well as leave their mark on spaces with objects they can deploy anywhere. These new Inventory Items are:

Portable Item:

Deploy an animated model in any space that allows portable items
Play different animations based on player proximity and interaction

For more information see Portable Items

Avatar Interaction Pack:

Set custom animations on the player and allow other players to interact with you directly
Hug, high-five, shake hands, chest bump - players can interact like never before!

For more information see Avatar Interaction Packs

Group Animation Pack:

Set custom behaviours on the player and allow other players to join you in a group activity
Move in perfect unison with the members of your group

For more information see Group Animation Packs

Sound Pack:

Set custom sounds to your avatar and play them to those around you
Attach custom animations to play while certain sounds are playing

For more information see Sound Packs

These new templates add a huge variety of new ways for PS®Home users to interact and express themselves, and will help make PS®Home an even more lively place to spend your time.

Lua API

There are no new functions in either the Lua or HDK APIs.

One change has been made to a function in the Lua API.
`NetPropertyBag.GetBagIndex` has been updated slightly, as listed in What's New in Lua.

Known Issues

none

Bugfixes

A number of bug fixes and general stability improvements have been made.

Previous Releases

Release 1.83

New features

Support for creation of 'BodySuit' clothing type. These are a composite of the Torso, Legs, Feet and Hands components and covers all of the avatar below the neck.

Known Issues

None

Bugfixes

Online Help URL is now fixed.
ObjectEditor 'Objects View' shows the Object's UUID as a tool tip. Useful if you have multiple objects with the same name.
QA Request Form fixes.

Release 1.82

This HDK update provides an updated version of the developer build of the Home client, containing bug-fixes and optimisations to underlying systems and support for future templates. There are no API or feature additions.

There are no changes to any developer-facing functionality except for the behavior of the 'Object Resource Override' feature. Content created with HDK 1.82 that makes use of this feature will use the mapped localization to determine the resource to load, rather than the region.

For example, French Canadian users would see the French rather than English localized resources.

Release 1.80

New Features and Improvements

Improvements to Feature and Client Development

Significant changes have been made to the client to allow the development and release of future features without requiring a client update.

This means that future client patches will be more focused on bug fixing and stability while New Features will be delivered on an individual and more regular basis outside of a client update.

Developing features in this way means we can make changes and implement bug fixes much faster, potentially on a weekly basis rather than having to wait until the next client release.

As this is a big change to the client we aren't including a headline feature with this update to ensure it is as smooth, clean and focused as possible.

New features based on the 1.80 technology will be released throughout the coming months; here is an idea of some that are currently in development:

- More item types -- Including portable items with gameplay features and new types of Locomotion object.
- Player Inspect - Allowing players to inspect other players, see what items they are wearing and where they got them.
- Improved moderation -- Providing enhanced tools to help Home moderators and admins provide a better service.

- Improved Commerce Points - New systems to make the commerce process smoother and easier
- Events - Supporting broader and more engaging social & gaming events within Home.
- More use of server-side Save Data - Extending the range of save-data now that it is kept on Home's servers.

General Improvements

- A feature has been added to allow developers to choose what emotes they wish to use with specific Full Body Suit outfits. This gives developers more freedom when creating Full Body Suit outfits as they do not have to ensure that every default emote can work.
- Players you have interacted with as you journey through Home will be added to your Players Met list on the XMB™. Players can also be added manually with the new 'AddToPlayersMetList' Lua API command, in the LocalPlayer Library.
- It is now possible for the client to display user presence information on your friends list that can show where or what you and your friends are playing while in PS Home.
- Developers can now specify specific spawn points for users that have been invited to a scene so that you will always appear in the right place.
- A local service IDs list can be used to develop and test commerce items without publishing them to a sandbox.
- The commands `AsyncCommands.TicketRequest` and `AsyncCommands.EntitlementRequest` now take an optional version parameter, allowing v3.0 tickets to be retrieved. This version allows for a much higher number of entitlements per ticket.
- Voucher codes can now be redeemed whilst Home is running, using the command `AsyncCommands.AsyncCmdCommerceRedeemVoucher`.

Bug Fixes

- An issue with a specific Hair Color being reset has been fixed.
- Remaining issues with the sorting of the Inventory have been addressed.
- Some cases of download and relocation to scenes taking longer than necessary have been fixed.
- A bug that caused problems accessing mini-games and using queues after a network error has been fixed.
- Various additions have been made to make Home more tolerant of poor network conditions.
- A number of other bug fixes and general stability improvements have been made.

Client

Score Posting

There is a new API to access our new scoring service that will be available soon. This will form part of our events plans in the future and allows your content to hook into this. The new `System.PostScore` API will access a developer generated GUID to identify your scores, the score itself and optional metadata. The API is available to use now and further details will be provided when the service is available.

Future Documentation

From 1.80 onwards, we're dropping versions!

The documentation will be available as always on the wiki, and it will be updated each time a change is made.

Live content, kept fresh, whenever there is a change to the HDK.

You can check on the latest changes made on the Recent Changes in the HDK Help Wiki page, or set a watch on specific pages or the whole space.
Feel free to continue commenting

What's New in Lua

The following tables summarize the additions and modifications to the Lua API Reference.

New functions

No new functions

Changed Functions

Function
`NetPropertyBag.GetBagIndex`

Deleted Functions

Function
None

What's New in the HDK API

No changes or additions to the HDK API Reference.

Known Issues



See also the technotes on SCE DevNet - https://home.scedev.net/technotes

New Issues Reported in this Release

No new issues for 1.86

Issues Reported in Previous Releases

Self/Package (**HomeDeveloper.self**)

1. Character facial expressions do not reset whilst moving using a Locomotion Item. However the facial expressions do reset after the user has stopped moving. [HDK 1.75 Known Issue] (#12344)
2. Object Profiler doesn't accurately reflect garbage collected memory [HDK 1.70 Known Issue]. (#11246)
3. User will be unable to launch into a new clubhouse from Scene Editor if they have a pending clubhouse invite. (#5448)
4. The CTC (or "Change The Classification") icon that is used for Australian age ratings is too large in comparison to the text that it contains. (#3818)
5. When creating scenes and mini-games (specifically sessioned mini-games) that use the group door functionality, you may find that not all users are able to relocate out of the scene after joining the mini-game. If this happens try one or all of the following:
    a. Create a new user account and re-test.
    b. Amend the mini-game to be a sessionless mini-game and re-launch the scene through the Scene Editor.
    c. Create a new session with a different user and then attempt to have all users relocate.
    d. Wait for a period of time for the servers to be updated and then re-test.
6. `Renderer.Camera2dSetRotation` does not work with text rendering [HDK 1.51 Beta Known Issue]. (#5509)

Lua API

1. Setting Renderer position has no effect
2. Setting the trigger radius through the Object Editor does not work with RtGames

HDK API

1. The editor crashes if you assign None to the keyboard shortcut of a MenuItem [HDK 1.65 Alpha Known Issue]. (#8869)

Maya

1. Skin Color and Skin Specular Maps are included in the BAR file when packaging Clothing Components. [HDK 1.70 Known Issue]. (#6281)
2. [Windows XP] Many of the character components have black shading on the textured view of the mesh that shouldn't be there. [HDK 1.70 Known Issue]. (#5048)
3. [Maya 2010] The Attribute Editor may become corrupted when viewing AtgMaterials. This is fixed in Maya 2012 and loads faster in Maya 2013.

    Workaround: If corruption occurs, select 'Copy Tab' (found on the bottom of the Attribute Editor). This will show a workable Attribute Editor in a floating window.

Art Tools

1. The default camera flip plane settings in Maya may sometimes result in meshes not being clearly visible in the viewport when changing views.
2. Users may experience some graphical issues when using Maya 2010 x64 bit in Windows 7, particularly with meshes in the view port and with tabs disappearing.
3. Maya: In Maya, if character components originally used placeholder textures, when different textures are defined in the object xml these textures may not display and the placeholder textures are used in runtime instead. If this issue occurs,

the original Maya scene has to be modified to use non-placeholder textures and re-exported.
4. In the Sky Tool, if there is no shader texture applied (or the texture that was applied does not exist at the location), the scene will not display placeholder textures, but will instead not display any clouds.
5. After creating and launching a scene as a clubhouse in the Scene Editor, if the user then attempts to launch the same scene in Maya or 3ds Max using the Scene Viewer option, it will cause an error in PS Home. Users, after using the Scene Editor to launch the scene as a clubhouse, should only use the Scene Editor to launch the scene rather than using the Scene Viewer in the art tools. (#2437)
6. Applying Capsule or Sphere collision shapes to collision geometry in environments will sometimes have no effect when used in PS Home.
7. Geometry video screens that have multiple video sources are not supported currently. This will result in a crash in PS Home when the scene is loaded when the video feed is changed. It is recommended that a plane geometry video screen is used if multiple video sources are required when creating the scene.
8. For furniture item export in Maya and 3ds Max, individual collision shapes less than 10 cm thick in any direction may not work robustly in PS Home.
9. If you assign a texture containing the sub-string 'placeholder' to a texture slot in an ATG Material, the corresponding effect will be disabled. For example, if you assign placeholder_normal_map.dds to the Normal Map slot, the texture will be ignored by the runtime graphics engine and normal mapping will be disabled. If you want to edit and use any of the existing placeholder textures, make sure that you rename them so that it doesn't include the text 'placeholder'.
10. When creating a Scene in 3ds Max or Maya, upon export the Light Volume is replaced by its own bounding volume. This will cause extra probes to be generated if the Light Volume contains more than one shape.
11. For both Maya and 3ds Max, vertex animation is not supported and may produce undefined results.

Scene Editor

1. When loading a Scene Project file, the cancel button may not function as the user expects. Rather than cancel the loading of a Scene, this button will instead cancel the loading of a Model file. [HDK 1.75 Known Issue] (#10615)
2. Live editing and work area views do not match when using the Arcball Camera to rotate the view in the Scene Editor. (#6540)
3. User is only able to add seats through the Project tab in the Scene Editor. (#6658)
4. Only one script file is supported in a Scene. Any further scripts in a scene would have to be added as embedded objects.
5. The user found that if the Asset Window is not docked with the UI, the user may be unable to add assets to the scene due to the options being disabled.
6. When using an OnlineMediaRSS screen in a scene, the Fall off/end values set in the Scene Editor will be ignored by the client when an mp3 file is played. Instead of playing inside the point sound for the screen, it will be heard everywhere in the scene. When using videos however, the screen's point sound behaves normally.

Object Editor

1. HDK 1.45 Alpha: If the user updates the object files outside of the Object Editor whilst it is still open and then attempts to package, incorrect versions of the files may end up in the package. In order to work around this, close the object editor before editing the files and then reload the object after finishing, which will ensure the correct version is loaded into the editor and then is able to be packaged.
2. It is possible that the template for Companion Objects may require changes in the future and there might be some circumstances where we will need developers to update and repackage their Companion Objects. Therefore we recommend that you keep all of the assets used in creating your Companion Objects. You may also be asked to supply the Companion Object assets to your Home Account Manager.

Repertoire Editor

1. An unexpected error occurs when renaming an Emote node in the Property Editor window and clicking away before confirming the text entry with "enter".[HDK 1.75 Known Issue] (#12325)
2. [Object Editor] Using Object > Edit Repertoires may load the Repertoire Editor without loading the Object as you'd expect. The preferred workflow is to open the Object from the Repertoire Editor using the Object Browser.
3. Nodes are still validated against, even if the Parent node has been disconnected. [HDK 1.70 Known Issue]. (#10544)
4. [Windows 7 x64] The Repertoire Editor cannot always be restored from minimized. [HDK 1.70 Known Issue]. (#7999)
5. Application error occurs for first Edit Repertoires after user installs HDK. (#7982)

Particle Editor

1. When running the ATG Particle System Tool for the first time, the relative path of textures is not set. This means that launching a scene with a particle effect will result in a "Not Found" texture in PS Home. However, when you run the tool a second time the relative path is set, which nullifies the problem and the particle effect is displayed correctly in the scene.
2. In the Particle System Tool, the File menu keyboard shortcuts do not function as expected when the Effect Grid window is maximized.
3. In PS Home currently, particles with alpha do not sort correctly- if multiple instances of the same particle are spawned in a script they will sort over one another in the order they are called.

Network

1. Before installing a new version of the HDK, clubs created in previous versions need to be disbanded, as club information cannot be moved from one server to another (or accessed from different accounts) without causing errors. If you have created a club in a previous version of the HDK, do the following:
    a. In the client in which you created the club, disband all the clubs that you are tied to so that there is no club data left on your PSN account.

b. Uninstall the Developer Package as normal, removing all save and game data along with clearing the cache in the Debug menu on the XMB™.
c. Install the new client as normal (package or HDK), and load the client as normal, with your desired CSO to your CDEV sandbox.
d. When in the default apartment, add the scene entitlement object for the clubhouse you want to test into your test inventory.
e. Create a new club as normal, and change the clubhouse to the created clubhouse.
f. Select the "Go to Clubhouse" option in the club menu for your club and you should be able to enter the clubhouse as normal and not get the C-991 server error.

Media

1. Screens in Home in some circumstances can sort incorrectly against meshes with semi-trans shaders applied to them. To avoid this issue try to limit the use of alpha in materials that will be seen in areas with screens.

Other

1. Realtime Physics - Rigid bodies penetrate too readily. Thin objects can pass through each other. Cubes will roll as if they have curved edges, etc.
2. When viewing high polygonal scenes using the "Wireframe Mode" in the Profiler Sidebar, the user may experience a significant framerate drop in the HomeDeveloper.self.
3. Media RSS streams can prevent all HTTPS requests. For pre-cached Media RSS content, the client may not realize it has already downloaded files, and will re-download them. Media RSS screens can consume users' bandwidth. See Media RSS File Specification for full explanation on proper Media RSS usage.
4. When you call a sound from a Lua script, the sound file is calculated as a part of the Scene's memory budget, and not as a part of the sound budget for the scene.
5. The scenes that will be displayed in the new Navigator menu in the Developer's build package and HomeDeveloper.self are those that are for the region specified in the PSN account the user is connecting with. In order to view scenes outside of this region, the user must have a PSN account appropriate to the region of the content being tested (or select the correct country from the Developer Menu on start)
6. The HomeDeveloper.self requires a connection to the PlayStation®Network in order to obtain the current (even in offline mode). If there is no connection to PSN the default age of 0 is used. Therefore, if PSN is experiencing downtime, the user will be unable to enter the basic apartment through the Navigator.
7. Variable bitrate for mp3s is not supported.
8. Currently, users are unable to log in to the HomeDeveloper.self with a PlayStation®Network account where the age is set to 17 or under.

Bug Fixes

No bugs for 1.86, just general improvements.

Self/Package

- none

Lua API

- none

HDK API

- none

Object Editor

- none

Scene Editor

- none

Other

- none

Brand Guidelines

PlayStation®Home Brand Guidelines

All you need to know about how to refer to PlayStation®Home (and which logos to use) when making promotional material for your PS Home content.

In the PS Home Brand Guidelines PDF you can find

Information about:

- Logos
- Icons
- Color Palette
- Typography

Guidelines for:

- Sponsorship
- Partnership
- Third Party
- Merchandise

And Legal Information



Download PS Home Brand Guidelines

Download PS Home Logo Files

Getting Started

The PS Home Development Kit (HDK) documentation is for artists, developers, testers, and anyone who uses the HDK.

New to the HDK? Get started by installing the client and taking a look at the HDK Overview which introduces you to the tools for content creation within PS Home.

Artists

Try creating an object or an environment using Maya. Export the object and edit it in Object Editor and export the environment and edit it as a scene in Scene Editor. Then package the object and the scene ready for submission to quality assurance.

Designers

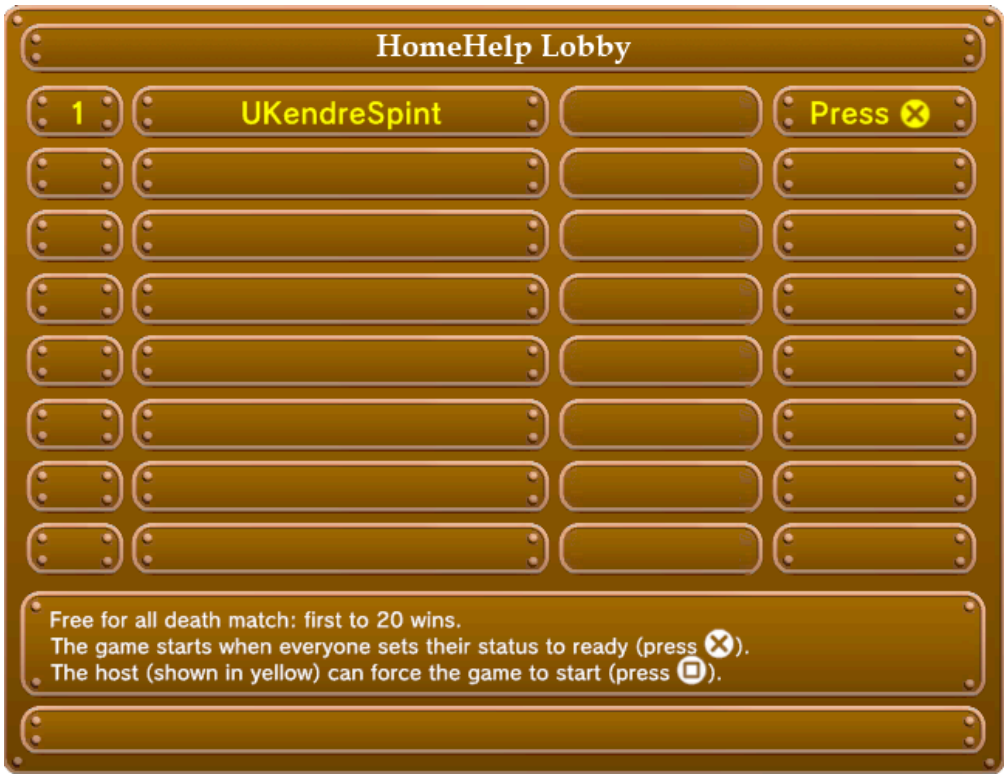
Explore and find out about the objects. ?scenes, and ?systems & services in PS Home - these sections cover everything from ?clothing to ?clubhouses, ?resource packs (DLC) and ?game launching to networking, rewards to ?commerce. Get to know what's possible with Lua Scripting and have a look at the ??general design guidelines, ?mini-game design and ?designing a Home Space.

Programmers

Explore the many features of Lua and implement dynamic behavior within Home using Lua Scripting. Customize certain tools in the HDK using the HDK API.

Testers

Check your content meets the technical requirements to function in PS Home.



HDK Overview

What is PS Home?

PlayStation®Home is part of the PlayStation®Network service (PSN). Users who have signed up to PSN can access it for free. PS Hom

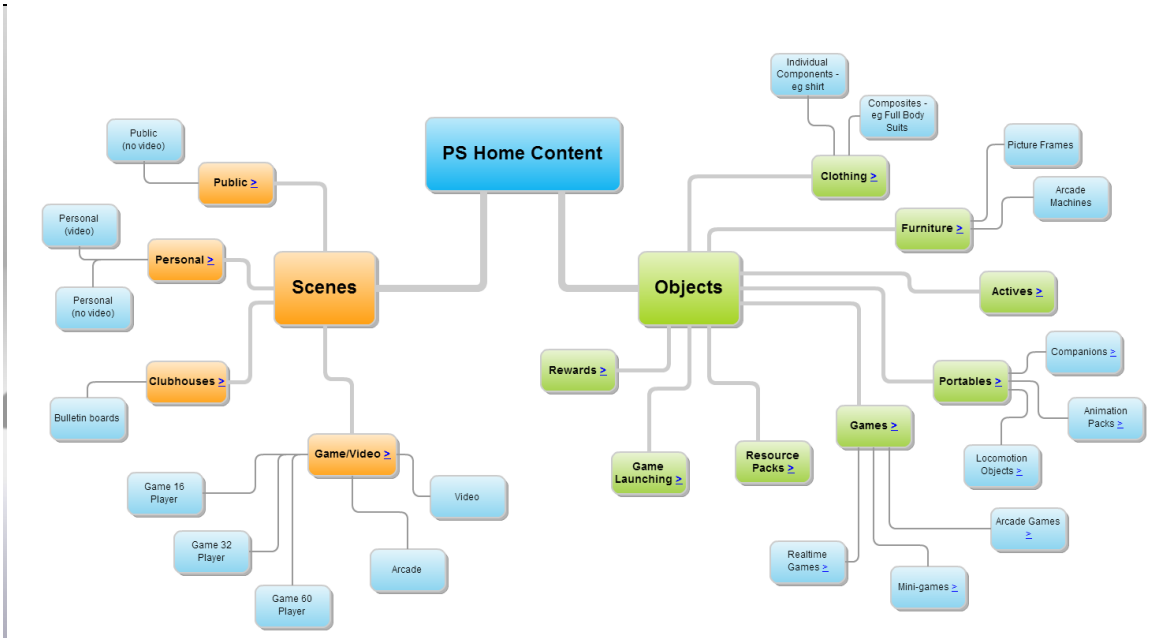
In PS Home, users can combine their interest in games with the social experience of being in a community of others with the same

After the initial download, users get their own customizable avatar and apartment, a profile that others can view. There are publ

PS Home provides a ready-made world that has many systems already in place for developers to use, including physics, commerce, UI

The HDK is designed for developers to make content for PS Home, using the platform and its existing systems to suit their needs. I

For the developer, PS Home offers an opportunity to easily create public spaces, and create and sell personal spaces (e.g. apartm



See the full diagram here

What is the Home Development Kit?

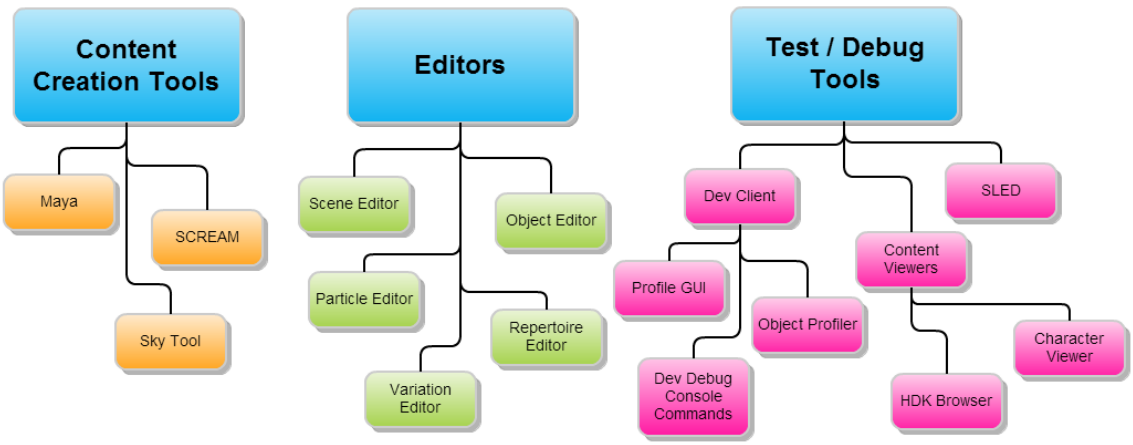
The Home Tools Team provides the means to create, edit, test and submit assets for PS Home using the Home Development Kit (HDK). Through the tools provided we empower developers to create anything from a complete public space, to a Lua-scripted 3D mini-game set inside the lobbies you create.

PS Home is constantly being developed and expanded to include ever-better features. As a result the HDK is also constantly being improved to offer the very latest innovations, workflow and feature improvements to developers.

HDK Tools

The Home Development Kit (HDK) is a bundled selection of tools and plug-ins that are downloaded from https://home.scedev.net and installed on a PC. Only registered PlayStation®3 or PS Home developers may access this toolkit. We also offer samples of scenes and assets, documentation and tutorials.

The following tools are free to registered developers and are constantly expanding and improving:



Documentation, Samples and Tutorials

Since the first pre-release of the HDK we have responded to the needs and requests of developers for more detailed documentation and support. This should help developers plan their projects for PS Home knowing the current scope of the features, best practice using the tools and the future features being worked on.

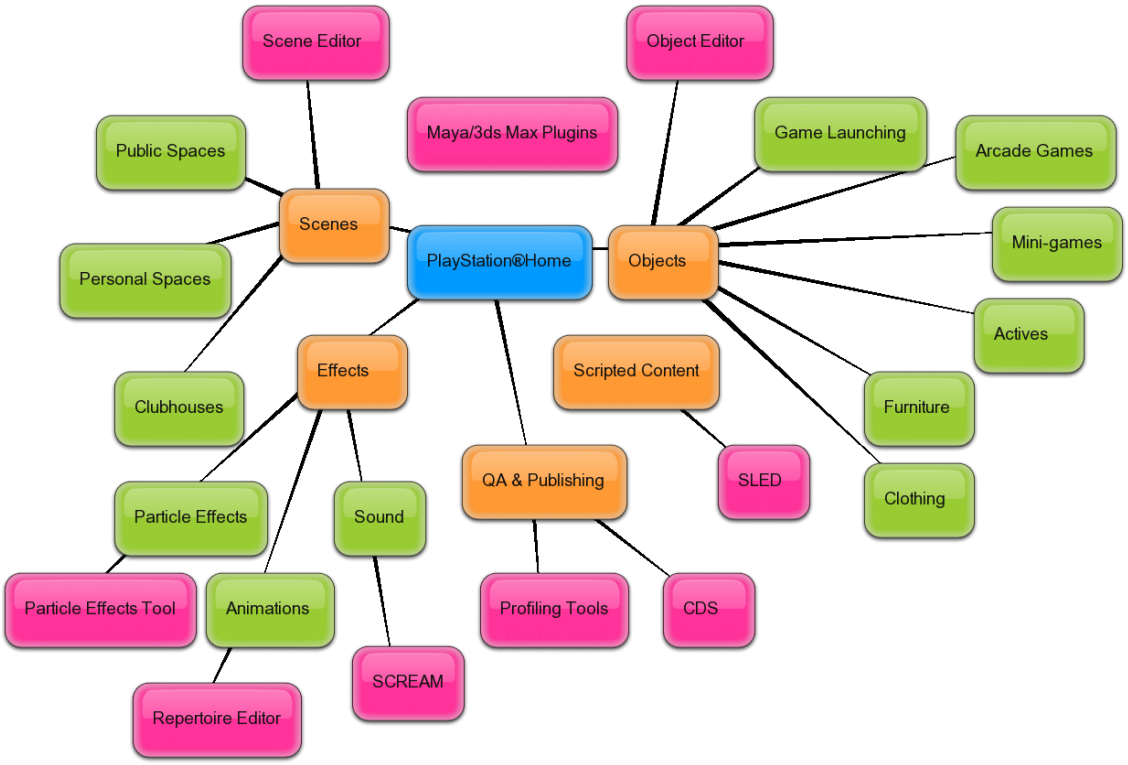
As a result, we provide regular updated documentation and samples on https://home.scedev.net for every release.

One of the most important documents is Known Issues. Please take a few moments to read through this before moving on to the HDK Installation.

Samples are offered as a separate download with the HDK - go to https://home.scedev.net and select Resources > Samples to find the latest ones. Of course, they may be used only to study, explore and analyze the capabilities of the HDK for the purposes of your own development work. They may not be used as templates for the creation of scenes or other assets or for any other purpose.

Developing with the HDK

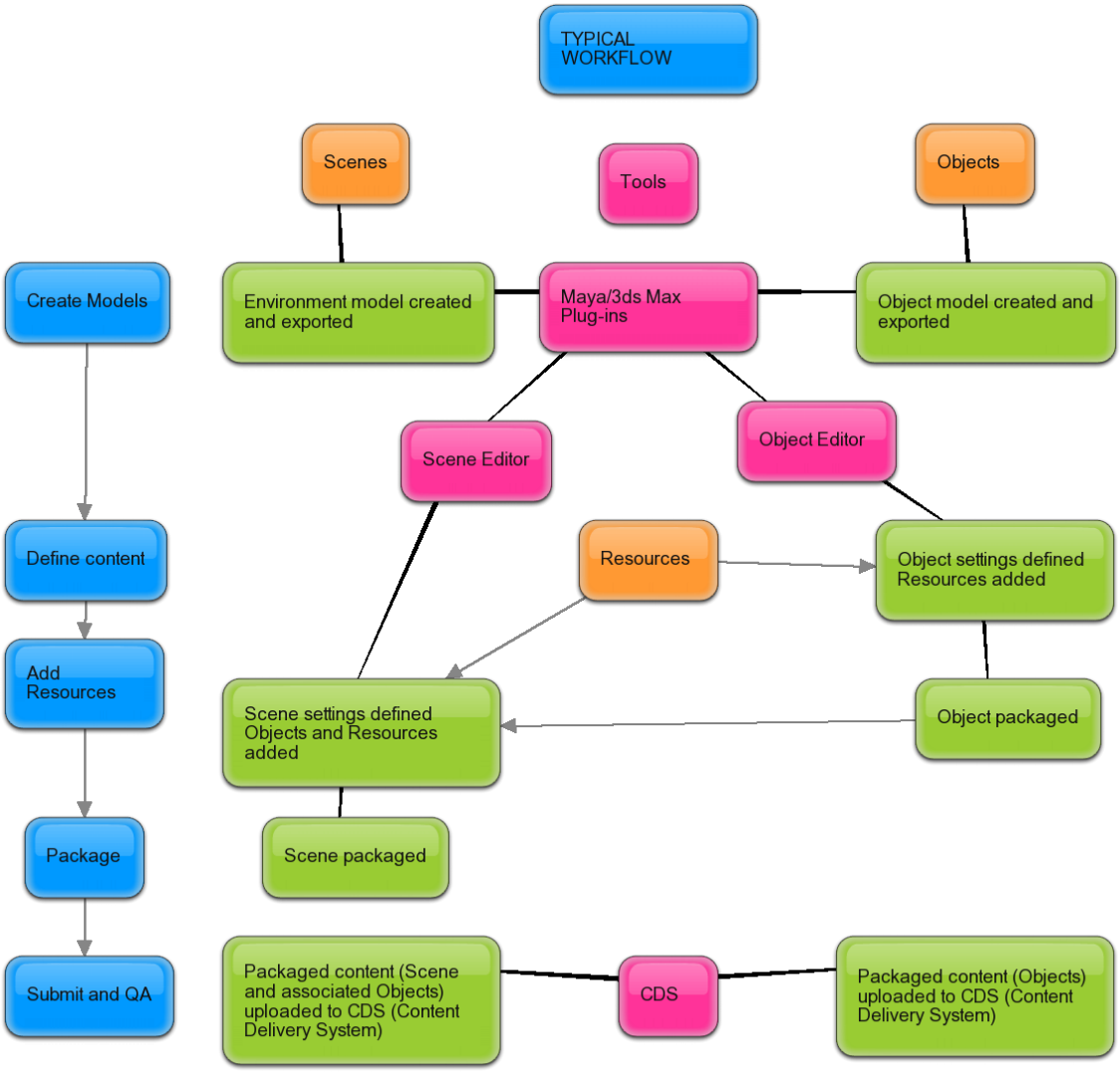
Scenes and objects are created with the HDK tools and packaged up ready for upload to Home. Broadly the content that can be created uses the following tools:



Each tool (such as the Maya plug-in) has an export option that allows you to save your content in the correct format within PS Home.

Once exported, the Scene and Object Editors give you the option to combine resources and edit settings. Which means you can add in sound effects and arcade game objects to your scene, or include Lua scripts within your mini-game or realtime game (for example).

To add your content to Home, the assets are uploaded using the Content Delivery System (CDS) - which requires you to package each asset using the Scene Editor or Object Editor as appropriate.



Art and Sound Tools

Maya Art Plugins

You add the plug-ins to Maya as new shelves and menu options. The plug-ins help you to create the Home Spaces and objects with useful automated processes to aid with lighting, lightmaps, textures, collisions and validation.

Photoshop

Adobe® Photoshop® or any other image editing tool can be used to create graphics such as sprites and backgrounds.

ATG Particle Effects Tool

You can use the ATG Particle Effects Tool to edit particle effects for use with the latest version of the runtime. Improvements have allowed a more granular approach to constructing particle effects, without loss of efficiency, and the tool was designed around this. The tool enables you to create effects easily by placing a number of 'nodes' into an 'effect graph' and tying together attributes on each node, to represent the data flowing between the nodes.

Currently, particle effects can only be added to scenes, but not scene or character components (i.e. furniture or clothing). This feature may be supported in future releases.

Static LODs

You can define additional meshes in environments as static LODs (Levels Of Detail). This allows the artist to reduce the detail, and hence the amount of triangles that need rendering in Home which improves the frame rate.

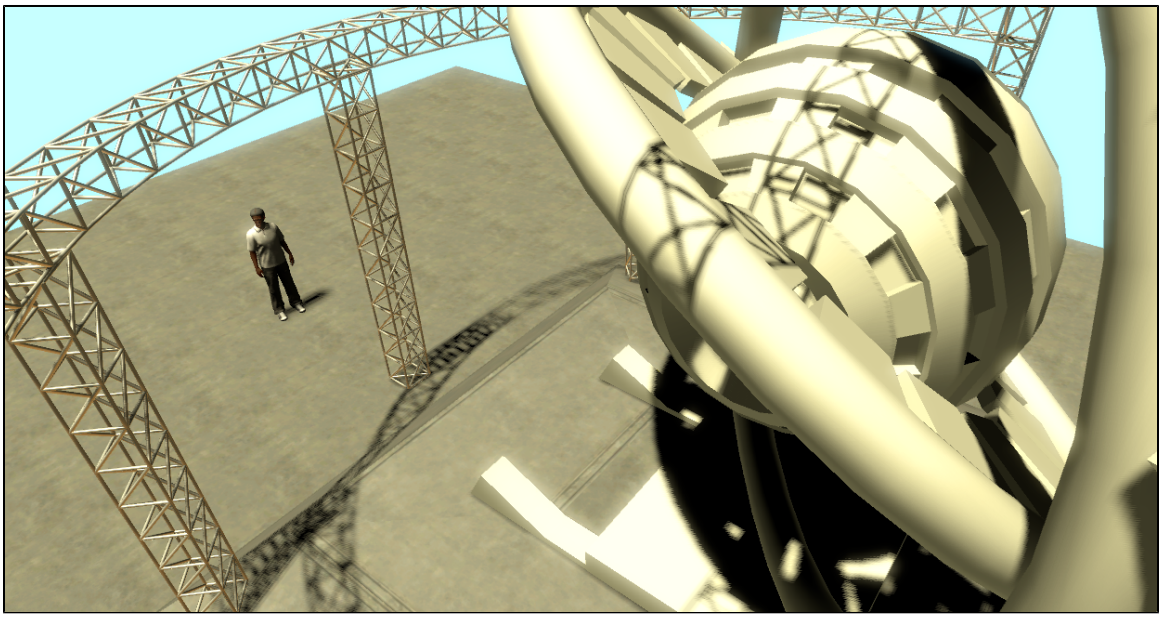
The benefits of this are clearly dependent on the particular artwork and the way in which it is constructed.

SCREAM Audio Tools

PS Home's Scene Editor works with soundbank files that are created using the recommended version of the SCREAM Tool offered to all PlayStation®3 and PS Home developers. Only audio produced with the SCREAM Tool will produce the files required.

The SCREAM Tool offers developers the ability to create complex, interactive sounds with a minimum of programmer support. It creates sound banks that work in conjunction with the SCREAM runtime engine. The tool can be used in conjunction with a Sound Tool Server (running on the local PC), which lets the sound designer audition and tune the audio before handing it to the development team.

Animation Overview



PS Home supports Joint Animation and Rigid Body Animation - see Joint Animation and Rigid Body Animation for more information.

For all the information on how you can animate thing in PS Home - create an animated object, place it in the Scene Editor and preview it - see Animation and Effects

There is also a simple rigid body animation sample (Simple Animation sample) featuring the spinning version of the screenshot displayed above. You can download the Simple Animation sample at https://home.scedev.net/projects/samples .

Scripted Assets Overview

The HDK supports the implementing of dynamic behavior within the PS Home runtime via the open source, industry standard scripting language Lua. Over four hundred Home-specific functions are exposed to the developer, and these functions are grouped into related sets called libraries.

Through Lua, you can do a great many things such as control aspects of PS Home's runtime through a command console, write 2D sprite-based arcade games, author interactive 3D experiences like multi-player networked games, and customize PS Home's game launching process.

Arcade Games

2D sprite-based arcade games will ordinarily be started up by moving an avatar inside a trigger volume associated with a virtual arcade cabinet placed within a lobby. However, the HomeDeveloper.self installed with the HDK can start up an arcade game directly, allowing the developer to bypass PS Home's menus and environment loading.



Normally, the visitor walks up to a trigger zone and will be invited to play the arcade game. Their screen will then solely display the 2D game until they exit, at which point they will be returned to their view of the Home environment.

Arcade Machines as Furniture

Arcade games are not restricted to being placed in fixed positions in specific scenes. It is also possible to assign an arcade game to PS Home furniture items. As such, the arcade game-hosting furniture can then be selected and placed anywhere within a personal space or clubhouse.

Mini-games

A mini-game is not necessarily a game. It can be a networked multi-player game or a sessionless 1-player game, but it can also be a Non-Player Character (NPC) or a scripted tutorial. It can be a tree that changes color depending on the season, or a portal that moves players around a scene. A mini-game can be whatever you script it to be.

A mini-game is an object that has a MiniGame component. The component gives it a dynamic relationship between itself and Lua scripts, giving it some callbacks that the script can react to and giving it access to certain Lua libraries.

Mini-games' ideal uses are for turn-based and single user functions, like chess, bowling, an NPC, solitaire. For the fast-paced, action junkie, try a realtime game. Mini-games are also good when you want to have spectators, when users are in a mini-game other users in the scene can watch.



Your Story

You join a scene and approach a mini-game. A small blue and white box, the Home colors, appears at the bottom of your screen inviting you take part. You accept, entering the mini-game. The mini-game, whether you see yourself in the same space or whether you appear somewhere else entirely, is still a part of that starting scene. Depending on the mini-game, other users in the scene can see what you do while in the mini-game, while you and those inside the mini-game with you can see everything happening "outside" of it. If the mini-game is not networked though, sessionless we call it, then it is just you in the mini-game with no spectators or other users.

Our Example

Chess. You and one other player accept the invitation to play a chess mini-game. Your avatars move to the seats at the chess table. Other users walking by or watching in the scene cannot directly interact with you or the mini-game, but they can watch. Within the mini-game you two pivot your pieces around the board. Mini-games send network messages to a Home servers before reaching the other users in the scene, so any animation or action that the chess game replicates can have an impact on the network. You take out the other player's queen just after shooing one of the bishops from the board, and corner their king. Check mate. A custom OSD, also in the Home colors, appears congratulating you. The game ends and you two players are returned to the scene.

> ⛔ Networked mini-games with two or more clients cannot be latency reliant.

Realtime Games

A realtime game is more likely to be a game. Realtime games user peer-to-peer (p2p) messaging to relay data back and forth between all users quickly, with (ideally) low latency. The realtime game will not be used to make chess, or an NPC, no. It will be used to make high-stakes warzones and battlefields, to pit mage against warrior in an arena. You will see fifteen other players working in together with you to set the world right under tight times. When you need to see what your teammate (or enemy) does practically as they're doing it, you will need a realtime game.

A realtime is an object that has a RealTime Game component. The component gives it a dynamic relationship between itself and Lua scripts, giving it some callbacks that the script can react to, giving it access to certain Lua libraries, and allowing it to send data peer-to-peer, instead of through Home servers.

Unlike mini-games, realtime games do not allow spectators. When you and a cadre of other players join the realtime it is just you few in there. No-one in the scene will see or hear or know what any of you do in there. The restriction on spectatorship is the necessary trade-off for p2p.

Your Story

You join a scene and approach a realtime game. A small blue and white box, the Home colors, appears at the bottom of your screen inviting you take part. You accept, entering the realtime game. Suddenly the scene changes, you're still in the same scene, but somewhere else in it and the only people with you are those who also joined. You are given your goals, and you (or your team) need to meet it before the clock runs out. Go.

Our Example

FPS sample (available from https://home.scedev.net/projects/samples). You appear in the alcove of a scene, below you is a cargo hold and in front of you is a portal. As you approach the portal you're invited to join the realtime game. You accept. A lobby appears listing the other players that want to join, and who is ready to play. When you are all ready you each appear at a spawn point in the cargo hold, armed with a loaded cannon with one mission: get 20 kills. All eight of you race around, shooting each other, going for the sweetspot, the instant kill ---the headshot. Cogs above your health bar show you not only how close you are to your 20 kills, but how close your rivals are. The last few kills are intense. You button-mash, your palms start to slide around the controller. Turning a corner you find your next target and lay into her, but at the same time someone does the same to you. The heat is on. You got the kill just before they take you out. The game ends, a custom OSD banner appears declaring you the winner and showing the stats of the other players. Then you all reappear in the alcove, asking who is ready for the next round.

Active Items: Games as Furniture

Active items are objects that the user can place within a personal space or clubhouse in a similar way to furniture. This allows the developer to create interactive or scripted behavior within an object in the user's personal space or clubhouse. Both the MiniGame component and RealTime Game component can be added to an active item (though not both components on the same object).

Read/Write to Server

It is possible to read and write to an external server using Lua. Reading from (and writing to) a server can be used if there needs to be persistent information in a scene about a user. For instance, displaying a personal high score board, targeting or personalizing the advertising, recording and storing which NPC communication loops have been completed, etc. Some of this information is personal data and would need to be handled by secure servers (https).

The data sent to the servers is XML, and the Lua functions to use are in the `HttpPostData` and `Resource` libraries. There is guidance on this function, with information on the SSL certificates you can use.

> ⚠ Writing to the server must authenticate the user with NP Ticketing.

It is also possible to access any user's leaderboard rankings for a particular title and write scores to them on the PlayStation®Network. Read-only access is available to user data for a PS3 title on the PSN, which allows you to unlock content in your space based on achievements in that title and query a user's list of trophies.

SLED - Lua Debugger

As part of our support for Lua we offer a Lua Debugger application (SLED). To use SLED, see Debugging Lua Script using SLED.

The Editors

The HDK provides two editors, an Object Editor and a Scene Editor.

Object Editor

You use the Object Editor to manage and modify objects that have been authored, validated and successfully exported from the HDK authoring tool (Home Maya). You can also use it to create object types not exported from Maya. Objects that you can compile without using Maya include non-furniture, non-clothing objects such as 2D and 3D interactive content (such as arcade games, mini-games, entities).

In most cases you use the Object Editor to work on an existing object, editing and or adding additional components, resources, localization and metadata. You also use it to specify the age restrictions for different regions, in the content metadata.

The Object Editor offers an Export option for packaging objects, post-validation. You can package all assets together in a single zip file, which you can then upload via the Content Delivery System (CDS).



For more information about using the Object Editor, see Object Editor.

Scene Editor

The Scene Editor takes the scene created in Maya, post-validation, and lets you add assets and place them in the scene as objects, triggers and scripts. For example, you can add ambient sound to the scene, an arcade game, a trigger to move a visitor elsewhere, collision details, and seating. You also use it to specify the age restrictions for different regions, in the content metadata.

The Scene Editor also offers a sophisticated viewer, taking all the elements created and opening them on the PlayStation®3 development kit or test kit so that you can view your work.

The Scene Editor offers an Export option for packaging scenes, post-validation. You can package all assets together in a single zip file, which you can then upload via the CDS.



For more information about using the Scene Editor, see Scene Editor.

Screens Overview

Screens are essentially 2D planes or 3D surfaces onto which you can stream images, video and other content. This content can be sourced from the PlayStation®3's local hard drive or from a server accessed over the internet.

Home Screen Markup Language (HSML)

Home Screen Markup Language (HSML) is an XML format for arranging images, video, audio, text, and colored boxes on a single screen. The "HSML" screen type can support only one HSML file at a time, but you can change it at any time using Lua script. HSML is particularly useful for displaying content such as leaderboards or other web content, because the markup can be dynamically generated by a server and frequently refreshed.

Media RSS

The PS Home client supports a subset of Yahoo's Media RSS specification. Media RSS can be used to create a playlist of images, audio and video. As with an "HSML" screen type, only one Media RSS file is supported per screen at any one time, but this can be changed at any time using Lua script. Feeds can also be dynamically generated by a server and retrieved by the client.

Video

The "VIDEO" screen type can display only video. As with an "IMAGES" screen type, it can display a single video, or a sequence of videos. When the sequence of videos completes, it will begin again from the start, looping continuously.

Images

The "IMAGES" screen type can display only images. This screen type can display a single image or a slideshow of images, displaying each image for five seconds with a cross dissolve transition between images. When the slideshow completes, it will begin again from the start, looping continuously.

> Additional screen types are supported within PlayStation®Home, see Screen Types.

Character Components and Furniture Creation



Character components are associated with characters, and scene components (such as furniture items and picture frames) are associated with scenes. This means that furniture and other scene components operate mainly in relation to scenes, and cannot go from one scene to another in the way that character components do.

Character Components

Character components can be equipped by the user to be worn by or to adorn characters. Because they are attached to the avatar, character components can move between different scenes.

Character components occupy certain slots on the character rig. The components must fit seamlessly onto the character, so the vertices of a component must match those of the adjacent components. The components must also integrate well with the movements the character is capable of, which means that the geometry must conform to the skeleton in all potential poses.



You can create character components using the shelf of tools provided and templates within Maya. You can then validate, preview and export the character components. The files exported from Maya can then be packaged and submitted for Quality Assurance (QA).

You can combine character components in two ways: as clothing collections or composites:

- Clothing collections are a combination of character components that have been grouped so that users can easily choose to wear all of the clothing collection's components at once. The pieces that make up a clothing collection are still individual components.
- With composites, you fuse two or more components together to make one new component. For example, you could combine the legs and feet components to make a 'legs+feet' combination component so that you can create a pair of jeans tucked into some high boots. The legs+feet combination will appear as one item in the wardrobe. You cannot select just the boots, or just the jeans, you must select both together.

You can also combine all the components together and make a FullBodySuit. A FullBodySuit is a composite component that covers the user entirely, from hair to feet. Users cannot swap in individual components while wearing a FullBodySuit. FullBodySuits can also have custom animations, and custom walking animation.

For more information on character components, see Character Components.

Furniture Items

Furniture items and other scene components, such as sofas, picture frames, sculptures, are decorations for a scene. You can create furniture items in Maya and then package them in the Object Editor.

The created components are available to users in their personal spaces, and clubhouses if they own them. To developers, furniture items are available only for public spaces as Embedded objects. Unlike character components, furniture cannot move between different scenes.

In addition to geometry, furniture items can be targeted and have collision. They can also have special attributes that allow interaction of a specific type, such as seat locators so that avatars can sit on the furniture, picture frames, and lights that can be turned on and off.

For more information on furniture items, see Furniture and Decorations.

Game Launching and Home Rewards

Game Launching

The Game System is a session management system that primarily passes information about the session to the target application.

The Game Launching System allows users in PS Home to form groups and launch into a PlayStation®3 title together. When Game Launch objects are published in the live environment, they are available to any user who has the title on the PlayStation®3 (either installed, or inserted into the system on a disc). The user does not have to purchase the object or acquire the object.

Game Launching requires development steps to support your title, but adds a new community element for your title when it is completed.

For more information about Game Launching, see Game Launch Objects.

Home Rewards

The Home Reward System allows developers to reward users by offering them special PS Home objects when they have accomplished a specific task or goal. You can grant rewards for tasks completed in an external title, or in PS Home. The Home Reward System maintains a list of reward 'tickets'. The awarded tickets are converted into Home objects that are added to the user's inventory.

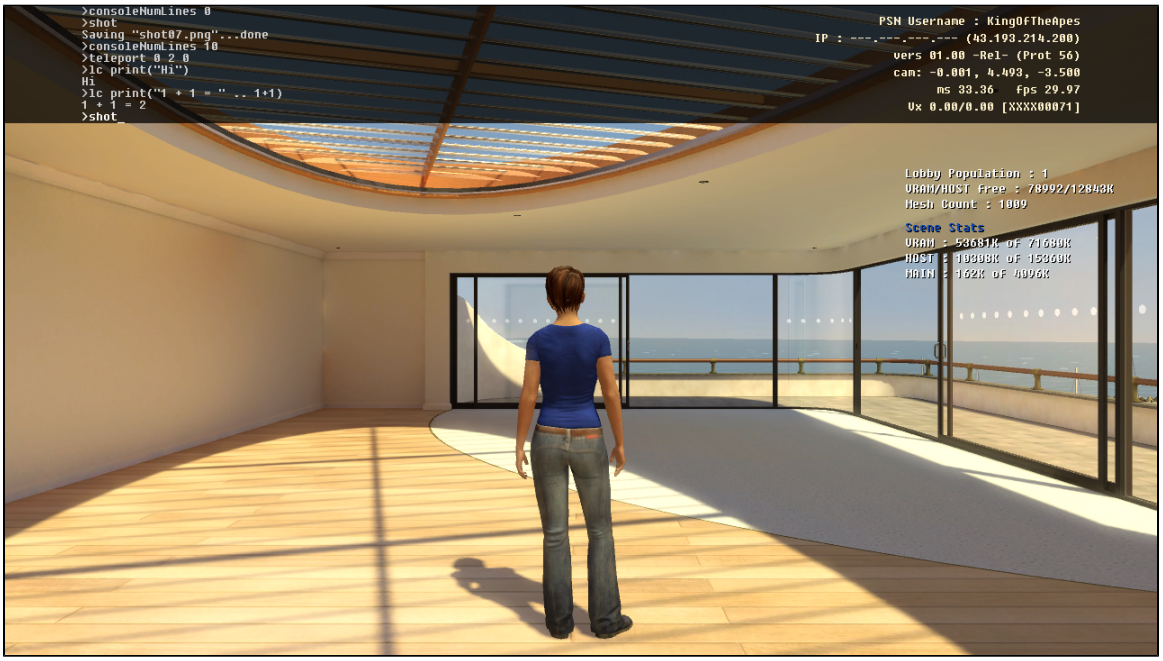
A Home reward can be any object that can be published in the live environment, such as furniture, scene entitlements, clothing, and Companion objects.

For more information on Home rewards, see PS Home Rewards.

Profiling Overview

Use the Resource Profiling tools to:

- Make sure that objects and scenes conform to the technical requirements and function appropriately in PS Home.
- Validate and profile content against parameters and constraints.



For more information on the tools available for profiling scenes and objects, see Profiling in the Client.

When profiling, see Testing, Validating and Submitting Content to ensure that your content meets Home's requirements .

Installing the HDK

This section outlines the system requirements for installing the HDK, tools you need or can use with the HDK, and how to run the installer.

System Requirements

- Microsoft® Windows® 7 (64-bit edition) or Microsoft® Windows® XP (32-bit edition) with Service Pack 3
- 2.0 GHz CPU (SSE3 instruction set support required if using Autodesk® Maya)
- 2 GB RAM
- NVIDIA® GeForce® 6 series GPU or better*
- Up to 2 GB free hard drive space (required for full installation)
- SN Target Manager 420.2.9.0
- Python 2.6.6
- Microsoft DirectX End-User Runtimes (August 2007)
- Microsoft .NET Framework 4.0

> Microsoft .NET Framework 4.5 is installed with Windows 8. It should work with the HDK tools, but it has not been tested. To avoid problems, keep to .NET 4.0

- Microsoft Visual C++ 2005 Service Pack 1 Redistributable Package MFC Security Update (32-bit)
- (Windows® 7 only) Microsoft Visual C++ 2005 Service Pack 1 Redistributable Package MFC Security Update (64-bit)
- Microsoft Visual C++ 2008 Service Pack 1 Redistributable Package ATL Security Update (32-bit)
- (Windows® 7 only) Microsoft Visual C++ 2008 Service Pack 1 Redistributable Package ATL Security Update (64-bit)
- Microsoft Visual C++ 2010 SP1 Redistributable Package (32-bit)
- (Windows® 7 only) Microsoft Visual C++ 2010 SP1 Redistributable Package (64-bit)

- OpenGL 2.0 (or higher)
- PS3 Reference tools or Debugging stations running system software version 4.10 or later:
    - DECR-1000
    - DECR-1000A
    - DECR-1400J
    - DECR-1400A
    - DECHA00J
    - DECHA00A
    - DECHJ00J
    - DECHJ00A
    - DECH-2000A
    - DECH-2000J
    - DECH-2000HR

*The HDK is developed and tested with NVIDIA® GeForce® GPUs only; it is therefore not guaranteed that other graphics hardware will produce exactly the same results in development.

Tools

Audio Tool

You need https://home.scedev.net/projects/scream_toolsPS3 SCREAM Tool 5.2 to make soundbank files that are compatible with the PS Home client. Separate versions are available for 64-bit and 32-bit Windows® editions

Art and Effects Tools

Autodesk® Maya®

You must have installed one of the following versions of Maya:

- Maya® 2013
- Maya® 2012
- Maya® 2010

Maya® System Requirements

HDK Installation

Before You Start

Before you install a new version of the HDK, do the following:

- Make sure that Maya is not running, otherwise the HDK may not install correctly.
- Disband any clubs created in previous versions, then recreate them in the new version of the HDK.

To disband an existing home club:

1. In the client in which you created the club, disband all the clubs that you are tied to so that there is no club data left on your PSN account. For more information on disbanding an existing club, see Disbanding a Home Club.
2. Uninstall the Developer Package as normal, removing all save and game data, and clearing the cache in the Debug menu on the XMB™.
3. Install the new client (package or HDK), and load it as normal, with your desired CSO to your CDEV sandbox.
4. When in the default apartment, add the scene entitlement object for the clubhouse you want to test into your test inventory.
5. Create a new club as normal, and change the clubhouse to the created clubhouse.
6. Select Go to Clubhouse in the club menu for your club. You should be able to enter the clubhouse as normal and not get the C-991 server error.

Things to note:

- Some tools, such as SLED v 4.1.2.0 (which installs with the HDK), may prompt you to install updates to more recent releases. Note that the HDK supports only the version packaged with the HDK or listed in the Installation Guide. If you update the software for these tools to a version other than the version provided or specified, the HDK may not work as expected and content created may not pass Quality Assurance (QA).
- The HDK is developed and tested using the English version of software. For example, the Maya tools are created and tested using the English versions of Maya. Other language versions of the software are not tested, not supported, and may not install properly or work with the HDK.
- You may require local technical support to connect your PlayStation®3 Development or Test Kit to your computer in order to preview your work.
- You may require the assistance of local technical support if you do not have administrator privileges on your computer.

Installation Files

HDK 1.75 has three installation files:

Installer	Description
HDK-1.75.exe	The main installation file for the HDK. It includes all the files required to develop, export, edit and preview PS Home content.
HDK_DistributedLighting-1.75.exe	This installer contains additional files to enable the distributed processing of lighting calculations during content export. This can greatly improve export times. The Distributed light processing system includes Imogen, which requires the Java Runtime Environment (JRE). Imogen may not be compatible with the latest version of the JRE. If you experience problems when running Imogen, please downgrade your JRE version to one that is specified in the Imogen documentation.
HDK_DeveloperClient-1.75.exe	This is a standalone installation package for the PS Home Developer client that runs independently of the HDK. You need it to test content that uses Game Launching features and it is useful when previewing content that has been uploaded to a developer sandbox.

Running the HDK Installer

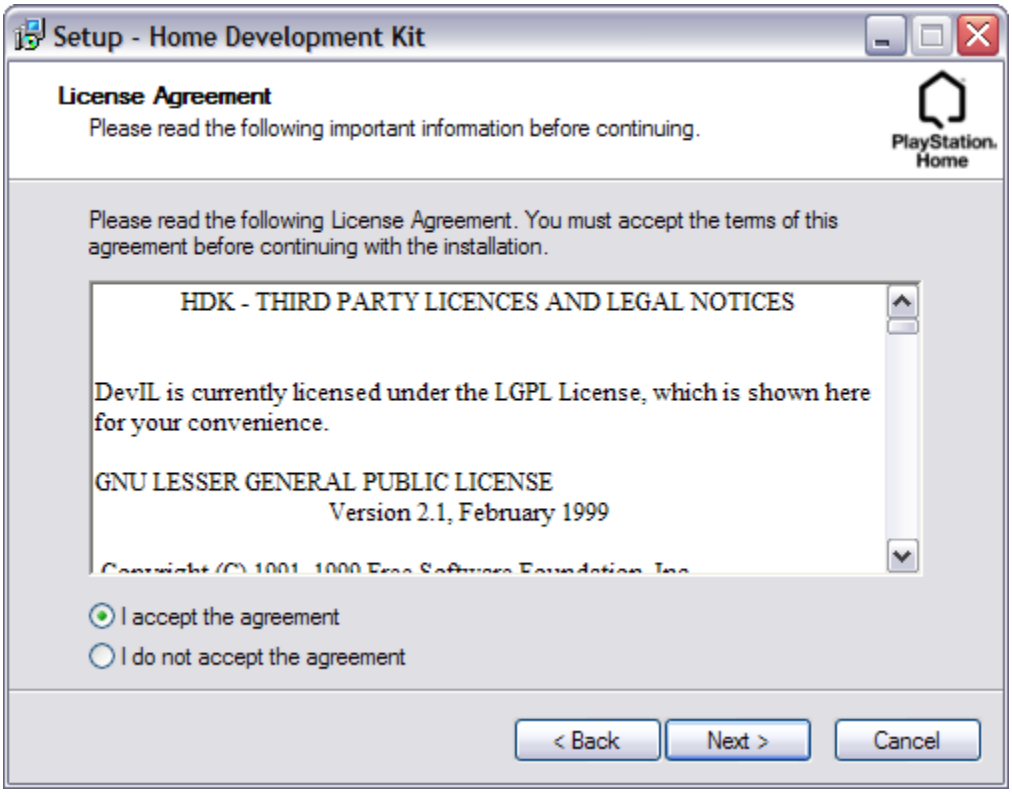
When updating your version of the HDK on a Test Kit, data from previous versions might cause issues. For example, if you have created a character component and you update your HDK you may receive "File not found" error messages when running the latest .self file. In this case, delete the Home Install data, Home Game data and Home Save data from the relevant areas on the XMB™.

To install the HDK:

1. Run the HDK-<version>.exe installer.
2. On the Welcome to the HomeHDK Setup Wizard screen, click Next to continue the installation:

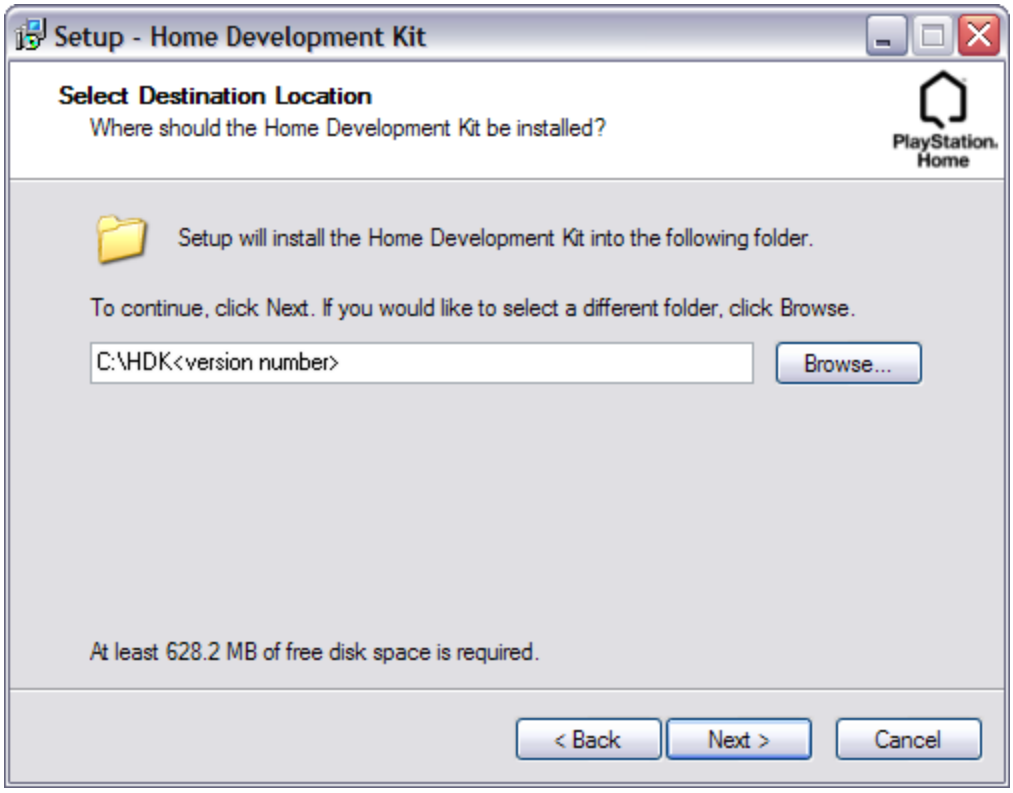


3. On the License Agreement screen, read the agreement, accept it and click Next:



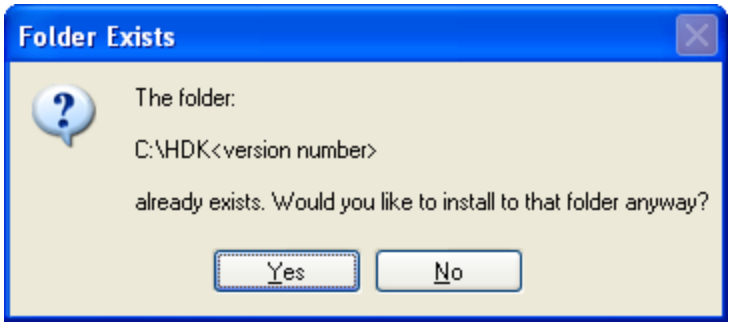
If you do not accept the agreement, the installation will stop.

4. On the Select Destination Location screen, either keep the default installation location, or select a different installation folder:



By default the installer copies all files to C:\HDK<version number>.

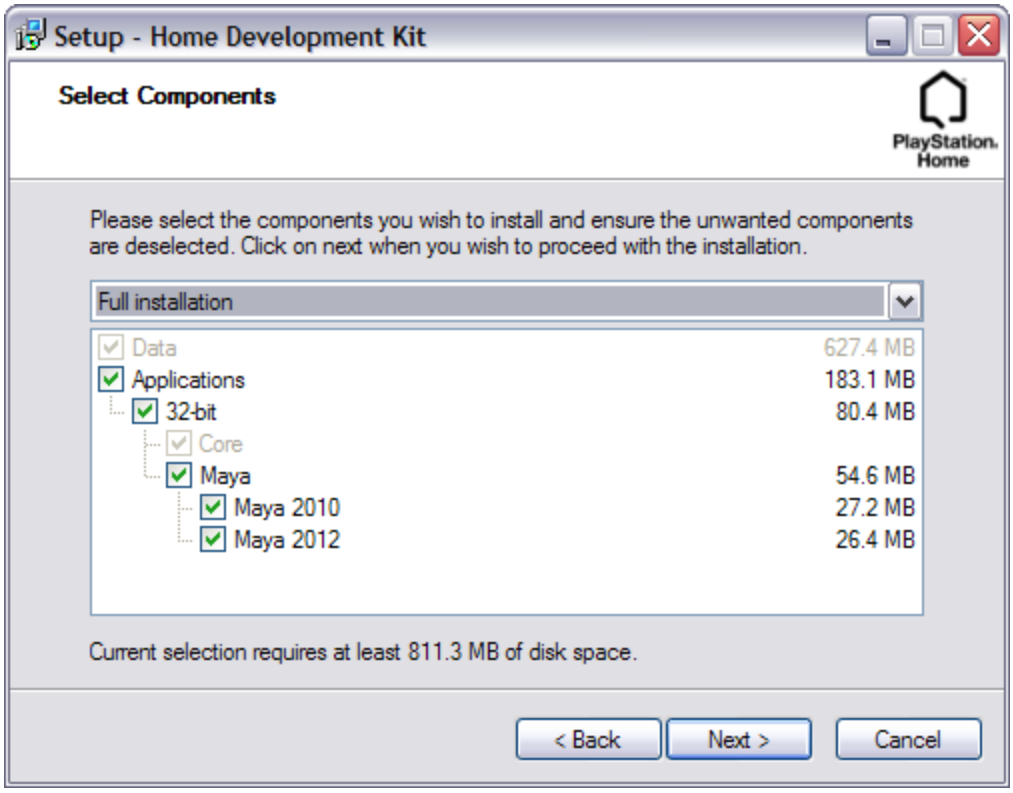
If you are upgrading from a previous version, you may receive the following warning message stating that C:\HDK<version number> already exists:



5. Click Yes to proceed, then click Next to continue the installation.

   The directory is not completely erased when you uninstall. Any files created in the previous version, such as scene files, are kept.

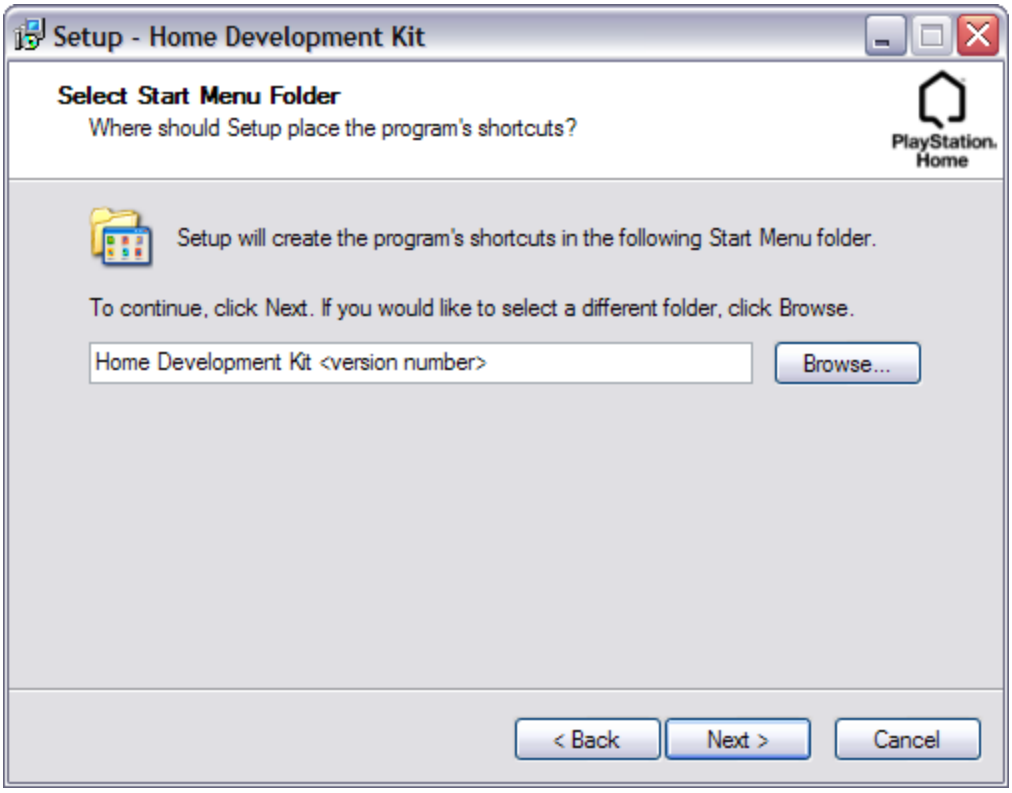
6. On the Select Components screen, select which components you want to install:



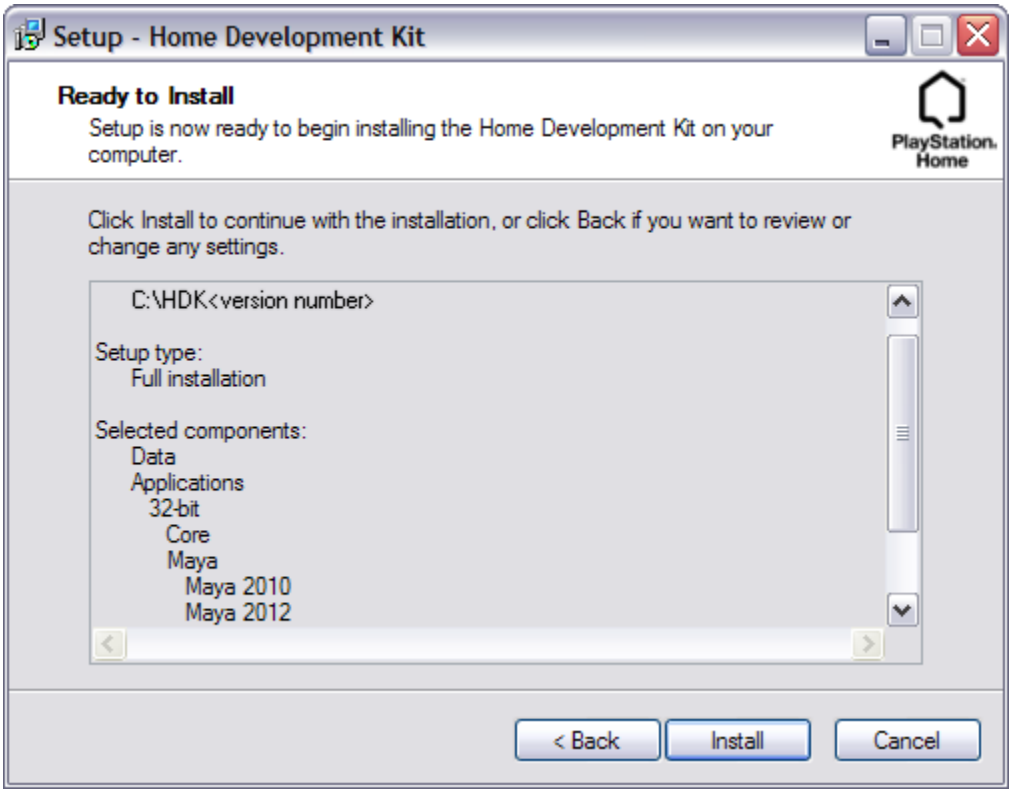
The default selection installs all the components listed. As you select and de-select components, the total disk space requirement changes at the bottom of the page.

> ⚠ Please note:
>
> - Only deselect components if you are certain that they are already installed. Unchecking components may result in the HDK not functioning correctly. If you are unsure, check with your local technical support.

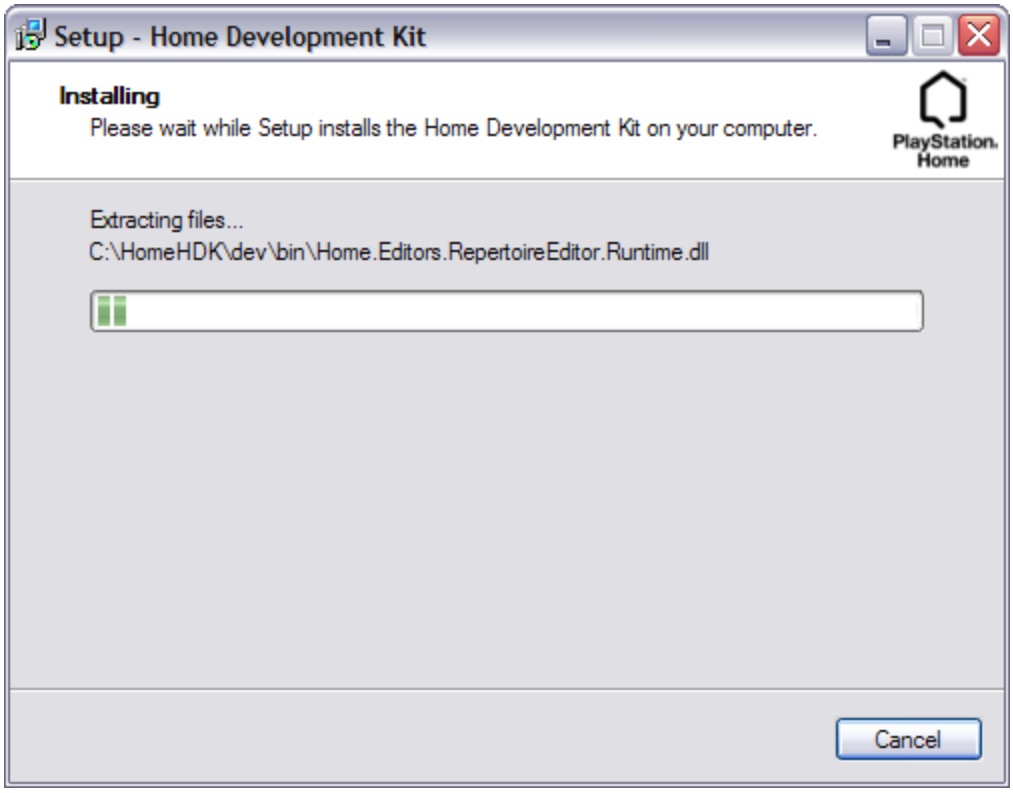
7. By default, the installer creates a start menu called HomeHDK. If you want to rename the menu, overtype it with the name you want, then click Next:



8. On the Ready to Install screen, review your installation selections. If you want to change any of the options, click Back to return to previous screens and make your changes, otherwise click Install:



9. Wait while the installation takes place and the appropriate files are copied to your system:



10. On the Completing the HomeHDK Setup Wizard screen, select Run HdkDiagnostics.exe if you want to check that you have all the necessary components installed on your computer:



11. Click Finish

If you chose to run the HDK diagnostics, the Home Diagnostics screen is displayed. This screen shows which components the HDK uses are already installed and which ones, if any, are missing. For more information on the Home Diagnostics tool, see Validating the Installation.

Localization Migration Wizard

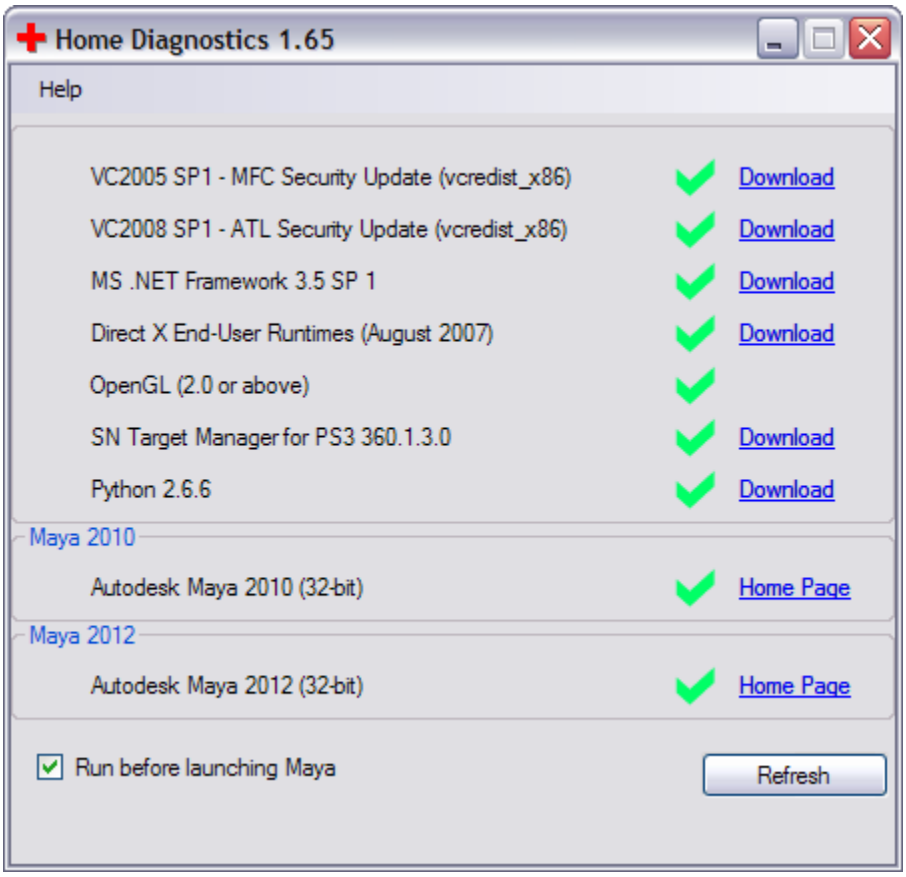
> ⚠ If you have objects that you created using HDK 1.2.3 or earlier, run the Localization Migration Wizard immediately after installing the latest HDK (if it has not been run already). The wizard creates the master localization file for all your existing objects - for more information, see Object Editor.

Validating the Installation

If you check the Run HdkDiagnostics.exe box during installation, the HDK Diagnostics Tool launches automatically when the HDK and components are installed.

The Home Diagnostics dialog indicates the status of the installation:

- A green check indicates a successful installation.
- A red cross indicates that an application is missing.



> ℹ You do not need more than one version of a 3D Graphics application or DCC Tool installed.

If any Windows Components or Miscellaneous items are missing, click the Download link to open your default browser at the necessary web page and then follow the on-screen instructions.

When everything you need is installed, you can close the HDK Diagnostics Tool.

Content Folders

> For Japanese Live Page
> 最新の日本語版ページはこちら 「コンテンツフォルダ」

HDK Content Folders

The HDK deals with a variety of content (such as textures, models, scenes and object files).

By default the HDK places all its content files within the same directory it was installed, under the following sub folders:

- artsource
- build
- intermediate

Creating Content Folders

We recommend that rather than using this default content folder you create a separate content folder for each Home project you are currently working on.

For a guide on how to create a new content folder and switch the HDK to use it see the Creating Content Folders page.

Upgrading Content Folders

Once a content folder has been created it is set to work with your current version of the HDK. If you install a newer version of the HDK you will have to upgrade the content folder to work with this new version.

For instructions on how to change the version of the HDK which a content folder works with see the Upgrading Content Folders page.

Switching Content Folders

If you are working on multiple projects at the same time you may wish to switch between using different content folders.

For a step by step tutorial on how to switch the content folder which the HDK tools use see the Switching Content Folders page.

> ⚠ If you use a network drive as a content folder, it may cause errors in the tools when trying to load content. Use a local drive.

Creating Content Folders

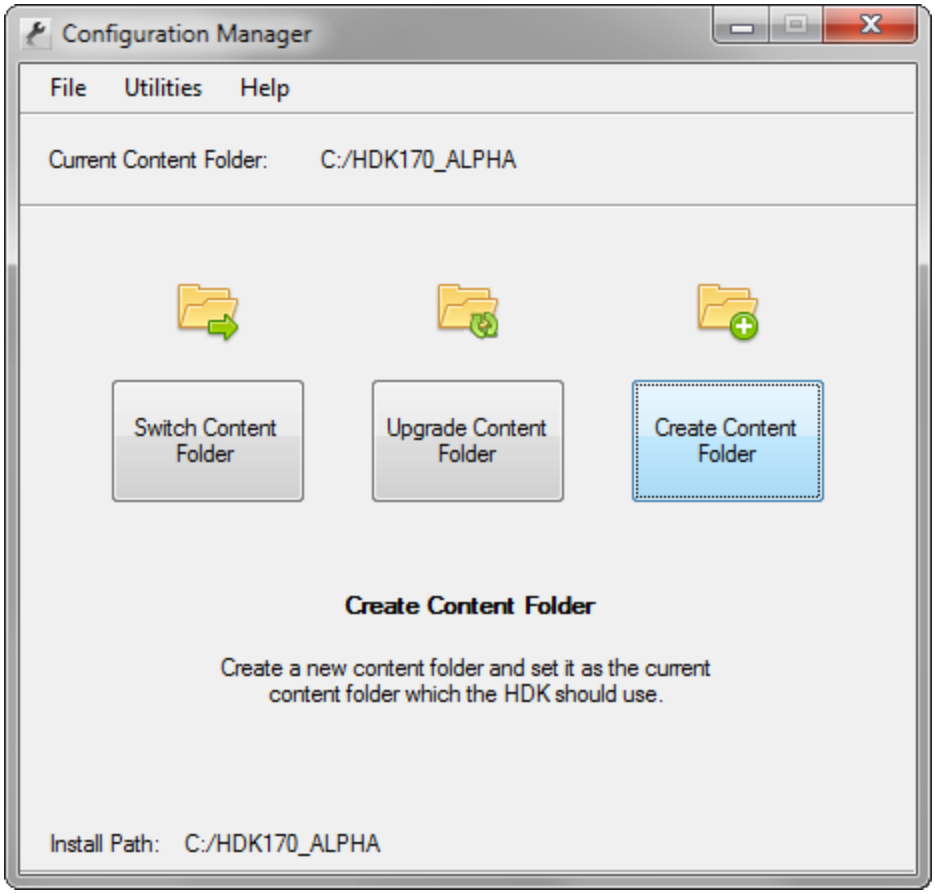
> For Japanese Live Page
> 最新の日本語版ページはこちら 「コンテンツフォルダの作成」

When starting a new Home project it is recommended that you create a new content folder to place all the project's content files.

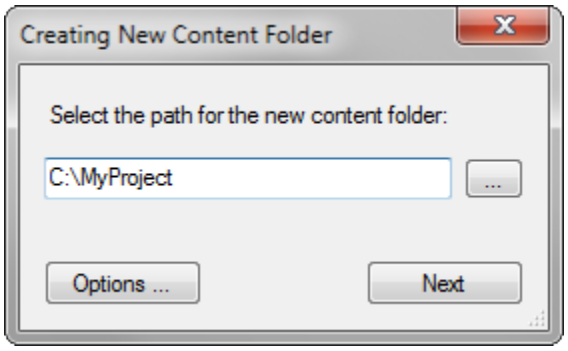
New content folders can be created using the Configuration Manager tool. This tool is installed with the HDK and can be found within the HDK's entry in the Start menu.

To create a new content folder:

1. In the Configuration Manager, click the Create Content Folder button.



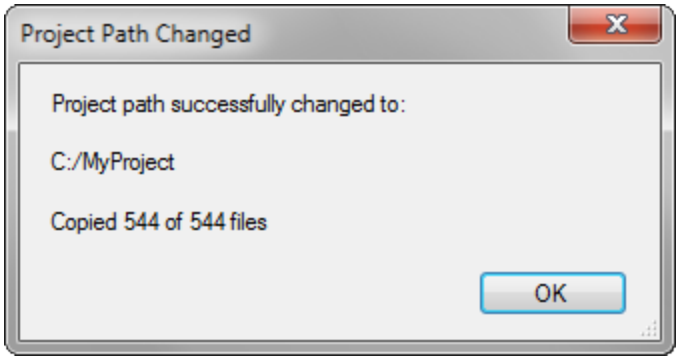
2. Enter the path to the new content folder you wish to create. Or use the … button to browse for the folder. Once you have entered the path click the Next button.



> The Options button allows direct control over which core HDK files are added to your new project. By default all of the core files are copied. This button is an advanced feature which can be ignored in most cases.

3. The new content folder will now be created. A progress bar will appear showing the progress of the creation process.

4. Once the new content folder is ready a dialog will be shown confirming that the HDK is now using your new content folder. Click OK to close the dialog. You can now start using your new content folder.



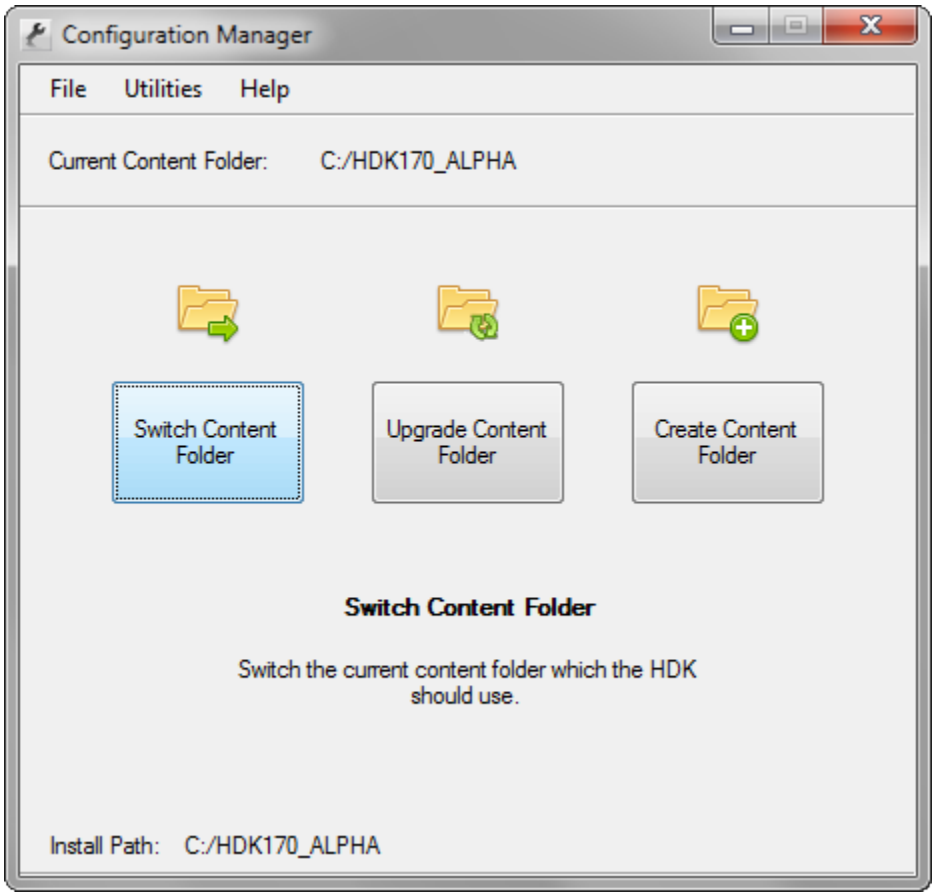
Switching Content Folders

For Japanese Live Page
最新の日本語版ページはこちら 「コンテンツフォルダの切り替え」

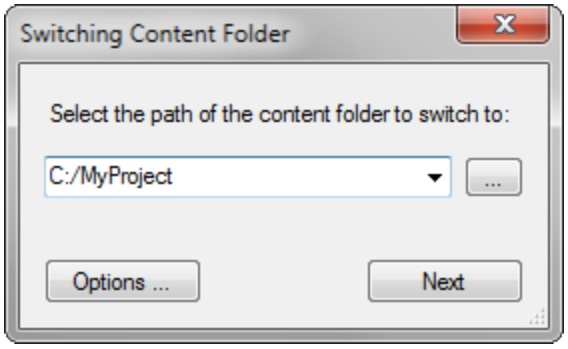
If you are working on multiple Home projects at the same time each with their own content folder you will need to configure which of these content folders the HDK uses. This can be achieved using the Configuration Manager.

To switch which content folder the HDK uses:

1. In the Configuration Manager, click the Switch Content Folder button.

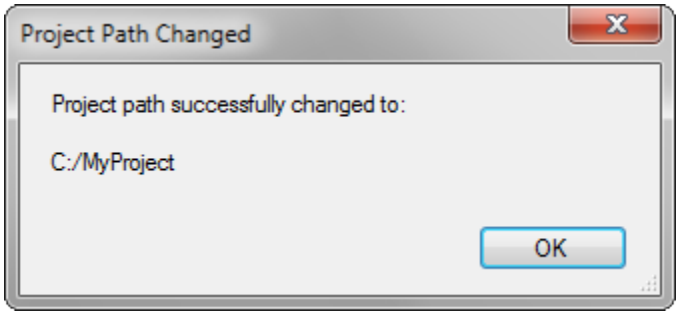


2. Enter the path to the content folder you wish to switch to. Or use the  button to browse for the folder. Alternatively you can use the drop down box to select a content folder which you have used previously. Once you have entered or selected the path, click Next.



3. A dialog will be shown confirming that the HDK is now using a different content folder. Click OK to close the dialog.

   The HDK tools will now use your newly selected content folder.



Upgrading Content Folders

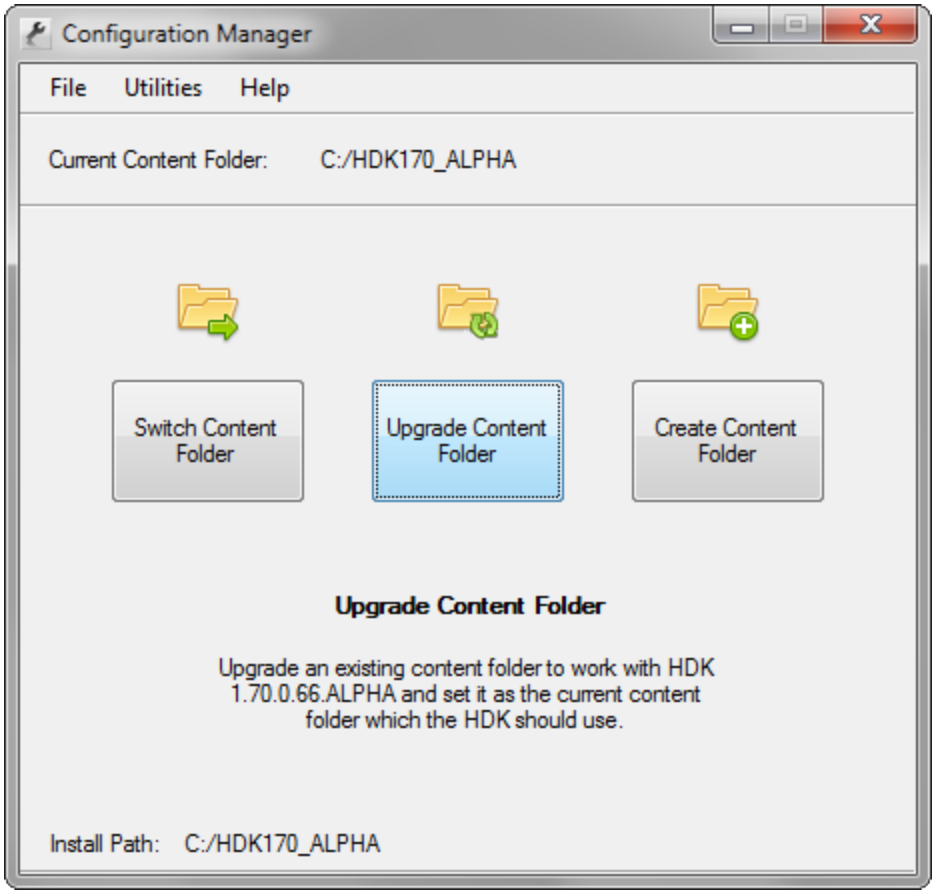
For Japanese Live Page
最新の日本語版ページはこちら 「コンテンツフォルダのアップグレード」

When installing a new version of the HDK, you may encounter difficulties working with existing content folders. This is because each content folder contains core HDK files such as the HomeDeveloper.self and BAR files, which are unique to each HDK release.

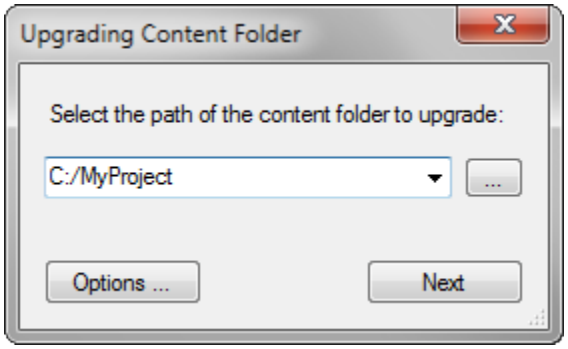
To solve these problems you can upgrade an existing content folder to work with your new version of the HDK. This can be done using the Configuration Manager.

To upgrade a content folder:

1. In the Configuration Manager, click the Upgrade Content Folder button.

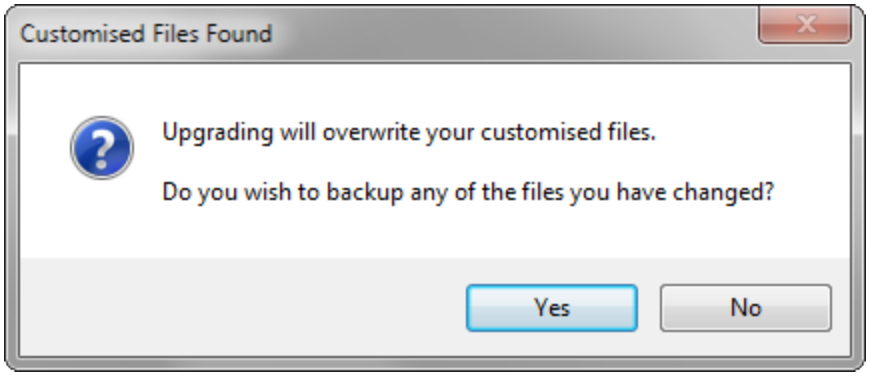


2. Enter the path to the content folder you wish to upgrade. Or use the ... button to browse for the folder. Alternatively you can use the drop-down box to select a content folder which you have used previously. Once you have entered or selected the path click Next.

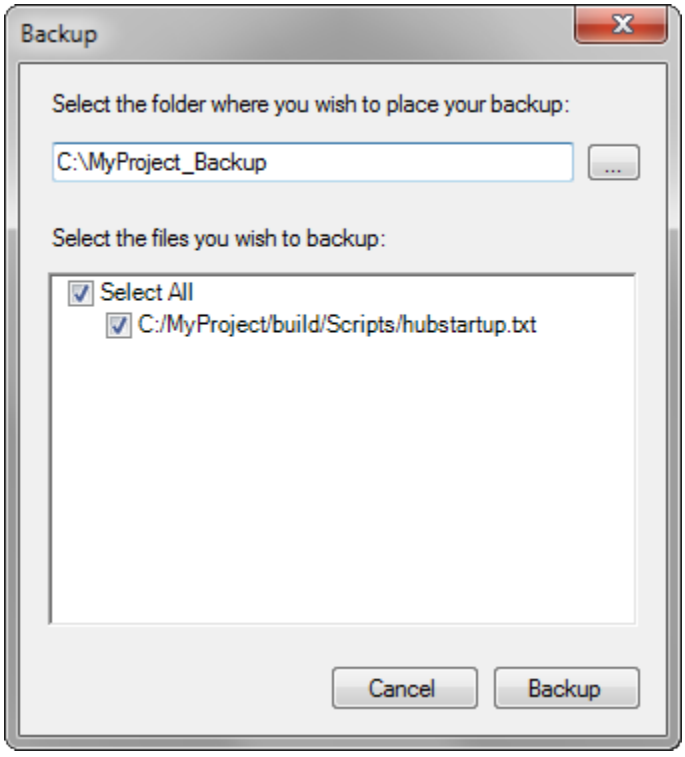


> The Options button allows direct control over which core HDK files are upgraded. By default all of the core files are upgraded. This button is an advanced feature which can be ignored in most cases.

3. The content folder will be scanned for customizations which you have made, such as edits to your HubStartup script. If any customizations are found you will be asked whether you wish to back them up before proceeding with the upgrade. These customizations will be lost as part of the upgrade so it is important that you backup any changes you wish to keep.

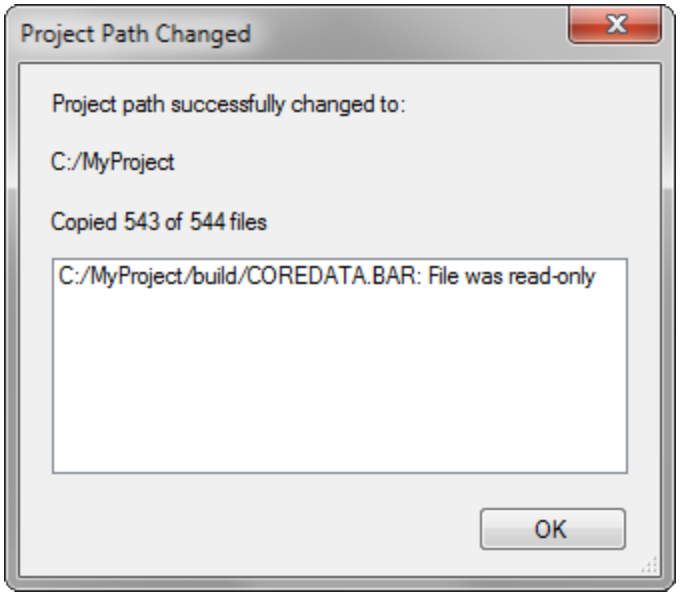


4. If you select Yes you will be presented with the following dialog. Enter the directory where you would like to place your backup and then check the customized files which you wish to backup. Click Backup to begin backing up your files, or Cancel to abort the backup and continue with the upgrade process.



5. Once the backup is complete or if you skipped the backup, the upgrade of the content folder will begin. A progress bar will appear showing the progress of the upgrade process.

6. Once the content folder has been fully upgraded, a dialog will be shown confirming that the HDK is now using your newly upgraded content folder.
   If there were any problems upgrading the files a summary of the skipped files and why they failed will be displayed. These skipped files may cause issues if ignored so you should address the errors and re-attempt the upgrade. If all the files did upgrade successfully click OK to close the dialog. You can now start using your upgraded content folder.



Installing the Home Developer Package

You can download the Home Developer package (.pkg) file from https://home.scedev.net/projects/hdk.

> ⚠ If you are reverting your version of the HDK package (for instance, returning to the 1.66 package after trying out the 1.70 alpha package), you may need to first delete the 'Home Developer' package from the XMB™ Game column, before installing the older version. (As the newer version may have created files that the old version does not understand and which may cause crashes on startup).
>
> In rare cases when upgrading, you may also need to delete your PS Home game data and saved data from the XMB™ to avoid conflicts with old data.

To install the Home Developer package, you can either use the Target Manager, or install files from a USB memory stick.

Installing Package Files Using Target Manager

To install package (.pkg) files using Target Manager:

1. Copy the .pkg file, (for example, Home.DeveloperBuild_1.75.pkg) to your File Server Directory.
2. Open Target Manager and connect to your PlayStation®3 target.
3. Right-click the target and set the file server directory to locate the package on your PC. Refer to the Target Manager documentation for details on setting your file server directory.
4. Restart your development kit or test kit in System Software mode.
5. From the XMB™, select Game > Install Package Files and install the required package from the list.

Installing Packages Files using a USB Memory Stick

To install package (.pkg) files via a USB memory stick:

1. Copy the .pkg file to a USB memory stick and insert the memory stick into the USB port of your PlayStation®3 kit.
2. Restart your kit in System Software mode.
3. From the XMB™, select Game > Install Package Files and install the required package from the list.

You can now run the installed package from the XMB™ Game menu. The name of the installed title is PlayStation®Home Developer.

Configuring the Network

The HDK includes content creation tools that run on a PC and a special version of PS Home (the Home client) that runs on the PlayStation®3 target. For the Home client to connect to the Home servers (which channel communications and provide downloadable content), PS Home must be able to connect to the PSN/NP (PlayStation®Network) servers, using specific IP addresses, TCP and UDP ports.

Connecting the Home client to PSN

The Home client can connect to the PSN/NP (PlayStation®Network) servers using these IP addresses, TCP and UDP ports:

IP addresses:

- 199.108.4.0/24
- 199.108.5.0/24
- 198.107.156.0/22
- 203.105.76.0/22
- 210.175.169.130/32
- 198.107.128.0/22

Ports:

- TCP: 80, 443, 3478, 3479, 3480, 5223, 8080
- UDP: 3478, 3479

> ⚠ These IP addresses and port numbers are subject to change. Always check the latest information. If you are a registered PlayStation®3 developer, you can obtain a list of the latest IP addresses, TCP and UDP ports at https://ps3.scedev.net/technotes/view/406. Alternatively, obtain them from your Regional SCE.

In addition, make sure that your firewall rules comply to the specifications outlined in the PSN tech notes.

For voice communications to work in PS Home, the client must be able to send to any address on UDP port 3658. For added reliability, if possible, the kit should also be able to receive from any address on UDP port 3658 or at least be able to receive from addresses that it has sent to on that port.

Contact your network administrator if you require assistance in setting up your firewall to enable these ports.

> ℹ The following error message is displayed when a firewall is blocking access to these ports: `D5028 the connection to the server was lost`

See Error Codes.

Network Connection Troubleshooting

- You cannot access a SP-INT environment with an NP account, or vice versa. If you do, you get a C-995 network connection error.
- There is a regional FAQ to help you make sure your kit is set up to connect to the PSN correctly.
- If you need to clear cached data:

    1. Delete any Game and Save data and any 'Home (non-executable)' items from your games list.
    2. Select Settings > Debug Settings > Format System Cache.

> ℹ If you still have issues, you can get support from your regional SCE.

Launching the Tools

When the HDK and components are installed, you can launch the Home tools.

To launch the tools:

1. Select Start > All Programs > Home Development Kit.
2. Select the tool you want to launch:

    - Select Launch Home Maya to open the Home Art Tools version of Maya. You use Home Maya to create environments, animated models, clothing and furniture items for PS Home. You validate and export a .mdl files. For more information, see Introduction to Maya.

    - Select Object Editor to open the Object Editor. Objects generated by Maya are completed in the Object Editor. Using the editor you can specify additional resources and components, and details such as descriptions and localization. You can also use the Object Editor to create new objects without exporting anything from Maya. When you have packaged an object in the Object Editor, you can submit it for testing and Quality Assurance (QA).

    - Select Scene Editor to open the Scene Editor. You can complete the scene file produced in Maya by loading and placing assets in the scene using the editor (see Scene Editor, Lua Scripting and Introduction to Audio). When you have packaged a scene in the Scene Editor, you can submit it for testing and QA.

The HDK includes other tools that are useful for developing content for PS Home, such as:

- The ATG Particle Effects Tool.
- The SCE Lua Editor and Debugger (SLED) Debugger.

You can also download the audio tool (PlayStation®3 SCREAM Tool) from https://home.scedev.net/projects/scream_tools.

The Home Client

The HDK is a collection of tools used to create content for PS Home, and the runtime application required to preview the content on a PlayStation®3 target. The runtime application is supplied in the following ways:

- The HomeDeveloper.self. This file is installed with the main HDK. It is a modified build of a release version of the Home client which can be used to test the content that you have generated in your PC's hard drive. It can be launched from Target Manager, Scene Editor, Object Editor, or Maya.
- The Home Developer Package. This is an installable package for your PlayStation®3 target. You must install this for the HomeDeveloper.self to run correctly. It is a modified build of a release version of the Home client which can be used to test the content that you have uploaded to a content server. It can only be launched from the XMB.



Ensure you have carefully followed the instructions on Installing the HDK. Two of the most important things to check are:

- Make sure you have installed the Home Developer Package on your PlayStation®3 target. Without this, the Home client does not run correctly.
- Make sure your PlayStation®3 target is flashed to the correct revision.

For more information on connecting the Home client to the PSN/NP (PlayStation®Network) servers, see Configuring the Network.

Launching the Home Developer SELF

The Home runtime executable resides in the following directory:

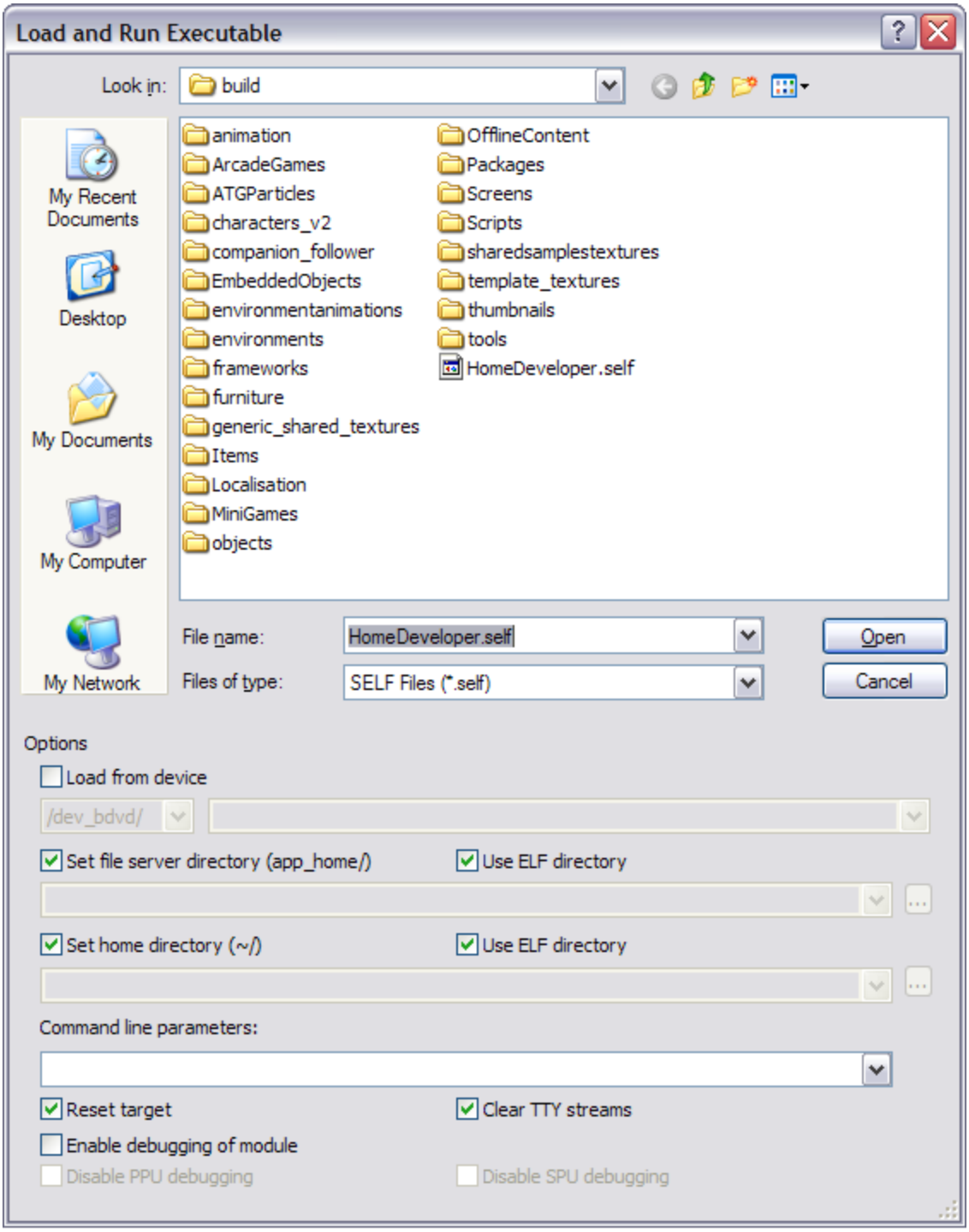
```
<HDK_INSTALL_DIR>\build\HomeDeveloper.self
```

You can run PS Home in a number of ways:

- From Maya via the Scene or Model Viewer.
- From the Scene Editor via the Launch Home option.
- From the Target Manager.

If you launch PS Home from the Target Manager, make sure that the following boxes are checked when you select Load & Run Executable…:

- Set file server directory (app_home/)
- Use ELF directory
- Set home directory (~/)
- Use ELF directory
- Reset target
- Clear TTY streams



You can use the following command line parameters to change the way the executable boots. Note that all command line options are case sensitive and are prefixed by 2 dashes.

Option	Description

--hdkbrowser	Launch in browser mode. This mode allows you to view .mdl (model), .efx (particle effect) and .atmos (atmosphere) files from the file serving root directory hierarchy. The HDK Browser allows models, particles and atmos files to be previewed. Its operation is very similar to the Character Viewer. Navigate the menu using the directional buttons and the cross/circle buttons. You can select the following main menu items: • CameraMode: Changes the camera control mechanism. • SlowFly/Fly/FastFly: These are 'free' camera modes of varying sensitivity levels that use the standard debug camera controls (left stick = adjust position, right stick = adjust rotation, R1/R2 = adjust height). • Lock To Model: Fixes the 'look at' position of the camera at the model and allows you to adjust the position/rotation of the camera using the left and right sticks, respectively. • Grid: Enables/disables the grid on the X/Z plane. • Model: Allows you to load a model by browsing your Build directory for a .mdl file. • Particle Effect: Allows a particle effect to be loaded by browsing your Build directory for an .efx file. • Atmosphere: Allows an atmosphere to be loaded by browsing your Build directory for an .atmos file.
--characterviewer	Launch in character viewer mode. This mode allows you to load character components to view, with several options you can set. For example, you can view the character components while the character is animated using the Animation Viewer menu and selecting a Body Animation (confirm by pressing the X button), or the different LODs by choosing the level in your options. Use the [START] button on the pad to cycle from Character Viewer to other view options.
--game <OBJECT_ID>	Launch an arcade game in full-screen mode. The object ID is the unique identifier assigned to a newly created arcade game in the Object Editor. The ID takes the following format: **XXXXXXXX-XXXXXXXX-XXXXXXXX-XXXXXXXX**
--mapregion @<COUNTRY_CODE>	This option has been replaced with the Options > Change Account Region functionality on the Home self launch menu. See Additional Developer Options.
--object <OBJECT_ID>	Launch in Object Memory Profiler mode. The supplied object ID must be that of a furniture item or character component. The specified object is displayed, along with statistics about MAIN, HOST and VRAM usage. To view character components on a rig, type in this command followed by another, i.e.: `--object <OBJECT_ID> --context clothing --rig 00000000-00000000-00000010-00000001`
--scene <SCENE_FILE>	Launch a scene in offline mode. SCENE_FILE is the relative path to the .scene file from <HDK_INSTALL_DIR>\build. For example, `environments/myscene/myscene.scene` Viewing a scene offline is useful for previewing graphics but you cannot interact with other avatars via the server. Also note that certain Lua API functions do not work in offline mode.
--sceneId <SCENE_NAME>	Launch a scene in online mode. To launch in online mode you must sign into PSN. This option is useful for running your scene on multiple targets to develop and debug a networked mini-game, because the Scene Editor can launch a scene only on a single target. You must run a scene in online mode to work with Game Launching and Home Rewards, or to use any of the networking Lua APIs. The scene must be one that is listed in the local scene list, found here: <HDK_INSTALL_DIR>\build\environments\LocalSceneList.xml Each entry in the XML is of the form: <SCENE Name="myscene">. In this case, set SCENE_NAME to myscene. See also LocalSceneList.xml.
--luadebugger <BUILD_FOLDER> --luaprotocol <deci3|tcp> --luaport <PORT>	Allow the HDK Lua debugger SLED to connect to the runtime. BUILD_FOLDER is the absolute path to the HDK's build directory. i.e. <HDK_INSTALL_DIR>\build. Set the protocol and port values based on what is entered in the target properties panel in SLED.

LocalSceneList.xml

This file by default contains the Object Profiler, Basic Apartment, and Thumbnailroom spaces. To add a scene to the list, you must first launch the scene in online mode from the Scene Editor. The scene is then automatically added to the LocalSceneList.xml file.

> To launch in online mode, you must sign into PSN.

Additional Developer Options

When PS Home launches, PS Home offers several launch options:



- `Go online`: Launches PS Home in online mode. To test your content in online mode you must download the hubstartup.txt file from your sandbox on the Content Delivery System (CDS). For more information, see Testing Online Content.
- `Offline`: Launches PS Home offline. It uses your `<HDK_INSTALL_DIR>/build` folder for content.
- `Options`: Allows you to change the account region of PS Home when you launch online. For example:



Testing Content in Online Mode

When you have published content to your developer sandbox using the Content Delivery System (CDS), you can view it using the Home Developer PKG, or by launching the .self.

> ⚠ You must have the Home Developer PKG installed on your PlayStation®3 system using the Install Package Files option on the XMB™.

You can also launch scenes online through the Scene Editor. See Launching a Scene in PS Home.

Downloading hubstartup.txt from CDS

An example hubstartup.txt is included with the HDK. This file contains a number of commands that affect how PS Home runs.

To test content in online mode, the hubstartup.txt file must also include the `contentServerOverride` console command, so that PS Home can connect to a sandbox. The command takes as an argument a URL to the CDS environment. The URL is unique for each sandbox. You can find the URL in the hubstartup.txt file that you can download from CDS.

There are several ways of entering the `contentServerOverride` command, as described in Launching PS Home in Online Mode below.

To download the hubstart.txt file:

1. Upload the content that you want to test to your sandbox on CDS.
2. In CDS, select My Details > Download hubstartup.txt.

For more information on using the hubstartup.txt file, see The Debug Console and Reference Tool Dip Switches.

Launching PS Home in Online Mode

To launch in online mode, you must sign into the PlayStation®Network (PSN).

You can launch PS Home:

- From the Target Manager, by editing or replacing the hubstartup.txt file in your <HDK_INSTALL_DIR>; or
- From the XMB™ by downloading the hubstartup.txt file onto a USB stick.

> ℹ To avoid confusion in the following procedures, the hubstartup.txt file that you download from CDS is shown in blue, and the original hubstartup.txt that is located in your <HDK_INSTALL_DIR>\build\Scripts folder is shown in brown.

Launching from Target Manager

To launch from Target Manager, you must either update the hubstartup.txt file that is located in your <HDK_INSTALL_DIR>\build\Scripts folder, or replace it with the the hubstartup.txt file that you download from CDS. This procedure describes how to update the hubstartup.txt file.

> We recommend that you take a copy of the original hubstartup.txt file, rather than updating or replacing the original.

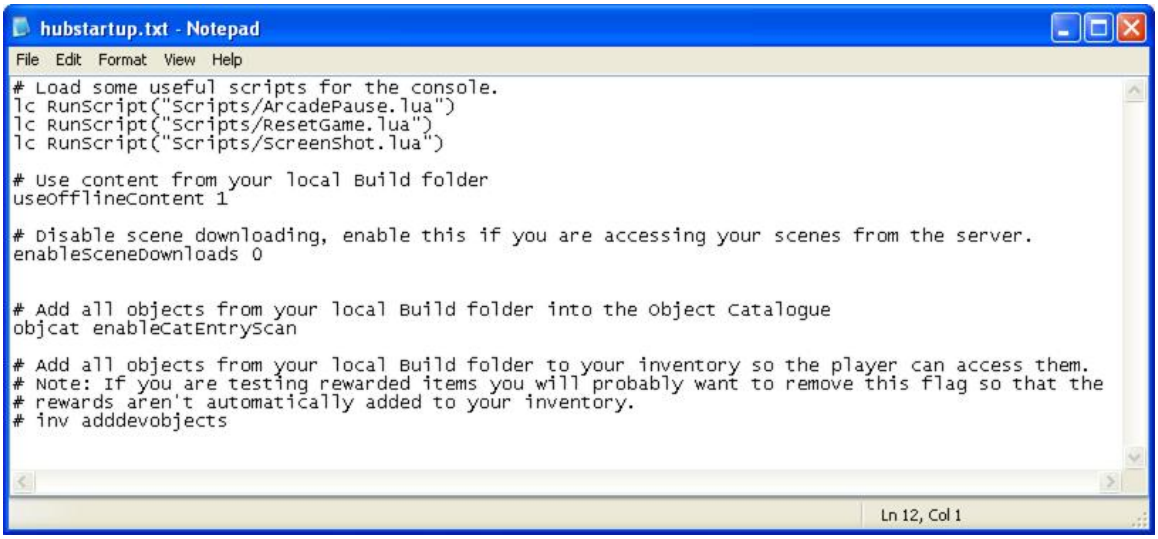
To launch from Target Manager:

1. Go to <HDK_INSTALL_DIR>\build\Scripts and copy the original hubstartup.txt file somewhere safe and easily remembered.

   > We recommend that you create a subfolder under \Scripts and put the file there. For example, <HDK_INSTALL_DIR>\build\Scripts\OriginalHub.

2. In the <HDK_INSTALL_DIR>\build\Scripts folder, open the original hubstartup.txt file.

   It should look similar to this:



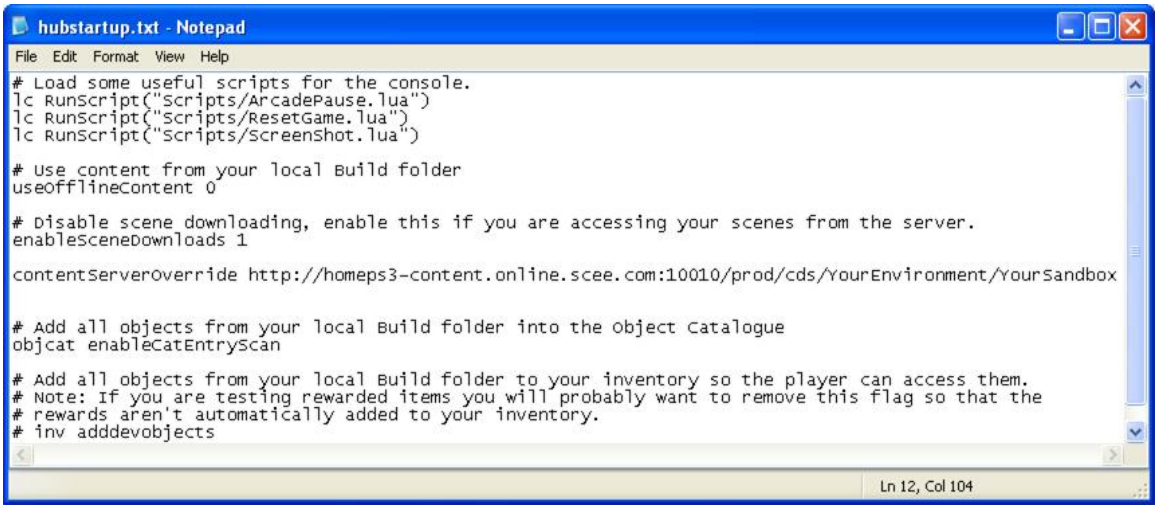
3. Change the arguments for the following commands, as follows:

Command	Original Argument	Change To	Effect
useOfflineContent	1	0	Setting the value to 0 tells PS Home to use online content.
enableSceneDownloads	0	1	Setting the value to 1 tells PS Home to download scenes from an online source.

> If you are testing the profanity filter, the argument for useOfflineContent must be set to 0.

4. Open the the hubstartup.txt file that you downloaded from CDS, and copy the contentServerOverride command, including the sandbox URL, into hubstartup.txt.

   The file content should now look like this:



5. Launch the HomeDeveloper.self from Target Manager, and select Go Online.

Launching from a USB

1. Download the hubstartup.txt from CDS and put it on a USB memory stick.
2. Insert the memory stick into the PlayStation®3.
3. Launch PS Home from the XMB™ and select Go Online.

Testing French Canadian Content

PS Home uses fr-FR for fr-CA localization. This means that when a Canadian user selects French as their language, the live client uses the French localization provided for France. The live client does this automatically, but the HDK developer client does not.

To test French Canadian content:

1. Localize the content in en-US and fr-FR.
2. Publish to the CDS.
3. Create a Canadian PSN account with the language set to French.
4. Log on to the PS Home developer client online and change the region to Canada.
5. The French version of your test is displayed.

Reference Tool Dip Switches

You can modify the behavior of the client by switching the development kit dip switches on or off. If your development hardware does not feature dip switches, such as a Debugging Station or Reference Tool DECR-1400, all Dip switches are disabled by default. The two dip switches used for development and testing are:

- Dip Switch 0
  - On
    - Switches to development mode.
    - Press the right analogue stick to enter a free flight mode. When in flight mode, press R1 to raise the camera vertically and R2 to lower it. The left analogue stick strafes the camera in the XZ plane. The right analogue stick controls the orientation of the camera.
    - Press SELECT to access the Debug Console (see Setting Dip Switches).
  - Off
    - Switches to release mode.
    - Press the right analogue stick to enter an `over-the-shoulder' third person camera mode.
    - Press SELECT to access the PS Home Safe Screen.
- Dip Switch 3
  - On
    - When the runtime boots, the save data for PS Home is cleared. This is useful if you want to reset your avatar back to a default state.
  - Off
    - Loads cached save data when booting the runtime.

Keep all other switches in the Off state.

Setting Dip Switches

When you run the Home executable on a test kit, you do not have access to the dip switches that you have on a Reference Tool. By default, the state of all switches (0 to 7) is Off. However, you can simulate setting them using the following Debug Console command:

```
SetDipSwitch <SWITCH> <VALUE>
```

Where VALUE can be:

- **1**: Override the physical switch (if present), setting it to on.
- **0**: Override the physical switch (if present), setting it to off.
- **Anything else**: Do not override the physical switch (if present).

To force setting switch 0 to On and 3 to Off, add the following commands to the end of hubstartup.txt:

```
SetDipSwitch 0 1
SetDipSwitch 3 0
```

For more information on the Debug Console and the hubstartup.txt file, see The Debug Console and Testing Content in Online Mode.

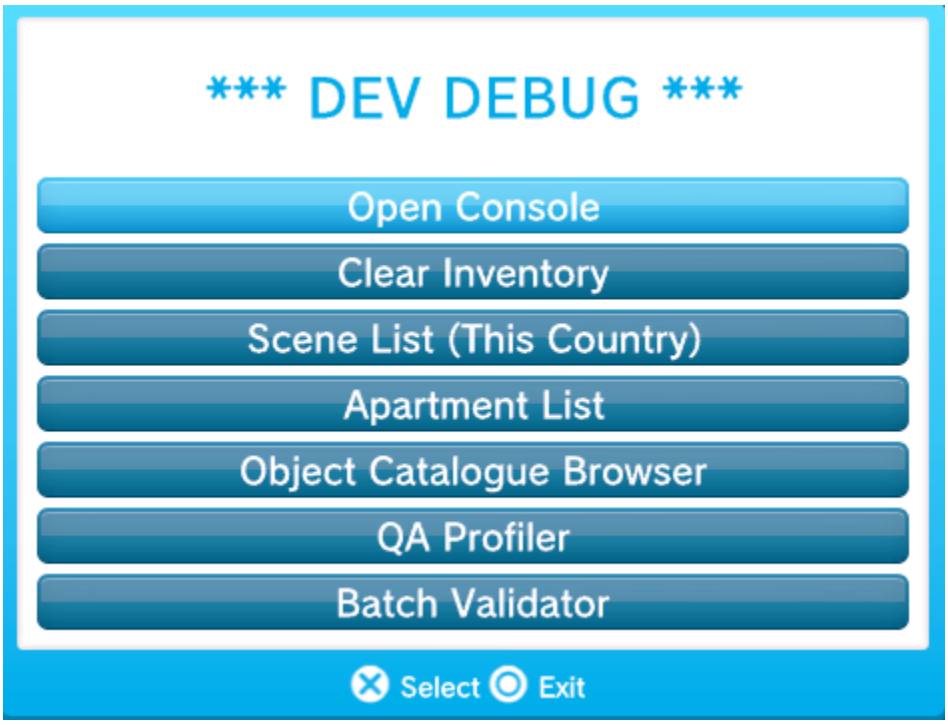
The DEV DEBUG Menu

The DEV DEBUG menu is part of the Home Safe Screen on the SELECT button. You can access the DEV DEBUG menu only when in release mode.

To activate the menu:

1. Press SELECT on the connected controller.
2. Select ***DEV DEBUG*** from the PS Home Safe Screen.

The DEV DEBUG menu provides a number of options for checking and testing your content:



The following table describes each menu option:

Menu Option	Description

Open Console	Opens the Debug Console for you to enter commands that affect the Home client.
Clear Inventory	Removes all furniture and clothing objects that have been added to your inventory. This is useful when you need to repeatedly test the rewarding of an item.
Scene List (This Country)	Displays all the scenes in the scene list that are available to your current Home account region.
Apartment List	Lists all the personal spaces from your \build directory. Select a space to relocate to that personal space.
Object Catalogue Browser	Displays a hierarchical browser of all Home objects available to the client. You can choose to browse: • All objects. • Only objects in the active object catalogue. • Only objects in the user inventory. To add an object from the Object Catalogue Browser to your inventory, press the ACCEPT button. Character components can be added and then automatically applied by pressing the SQUARE button.
QA Profiler	Opens the QA Profiler. Selecting this option is equivalent to entering `ProfileGui.Enable 1` in the Debug Console.
Batch Validator	Opens up the Batch Validator for profiling and validating multiple objects at one time.

The Debug Console

You use the Debug Console to issue commands to perform certain tasks or to affect the way PS Home runs. To enter commands when the console is active, you need to connect a USB keyboard to your PlayStation®3 target.

You access the console either through the DEV DEBUG menu, or by pressing SELECT, depending on whether you are running PS Home in development mode or release mode.

When activated, the Debug Console displays certain information about the scene:



The following table describes the data displayed on the Debug Console screen:

Data	Description
version	The current HDK version.
cam	The camera position coordinates.
lua	The combined frame rate (ms) of all active Lua scripts in the scene.
ms	The current frame rate, in milliseconds (ms).
fps	The current frame rate, in frames per second (fps).
Lobby Population	The number of people in the scene.
Mesh Count	The number has no significance. Ignore it. To get an accurate mesh count for your scene, use the ProfileGui.Memstats panel.
Visible Meshes	The number of visible meshes visible to the user. This value is more useful to QA, who populate a space with 60 users.
Scene Stats	The MAIN, HOST, and VRAM resources consumed.

When the Debug Console is active, certain keys have a special function:

Key	Special Function
TAB	Command completion. If you partially enter a command and press the TAB key, the command will either complete or if there is any ambiguity, you will be presented with all the commands that match the currently input string. If nothing is entered at the command prompt, all available commands are listed.
Page Up/Down	Scroll console by line.
SHIFT + Page Up/Down	Scroll console by page.
CTRL + Home	Go to first line in the console history.
CTRL + End	Go to last line in the console history.

For a comprehensive list of Debug Console commands, see Debug Console Command Reference.

Executing Console Commands using Target Manager

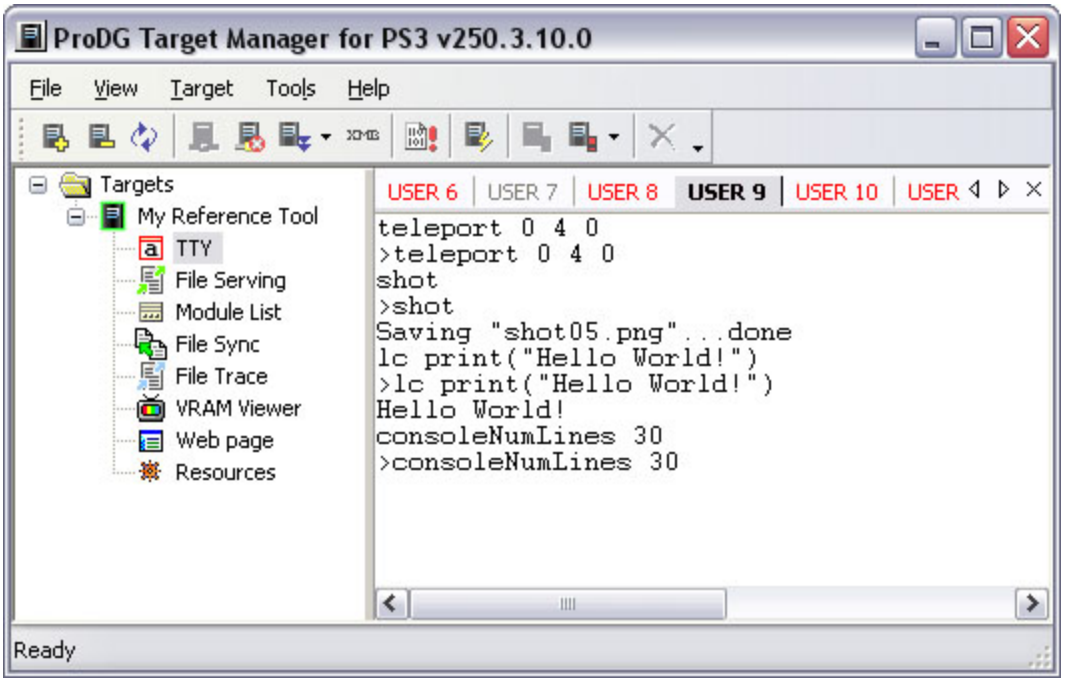
You can also execute Debug Console commands using a Target Manager TTY channel to send commands from the PC connected to the PlayStation®3 target running the Home client. Use the TTY channel USER 9.

> ⚠ Make sure that USER 9 echoes your input to the screen so that you can see the commands as you type them.

To make the commands visible in Target Manager:

1. In the Targets panel, select the PlayStation®3 target you are using.
2. Expand the item list below the target and select TTY.
3. In the TTY panel, select the USER 9 tab.
4. Right click and select Properties from the pop-up menu displayed.
5. Make sure that Echo TTY input to screen is selected and click OK.

You can now enter commands directly into the USER 9 tab, as shown below:

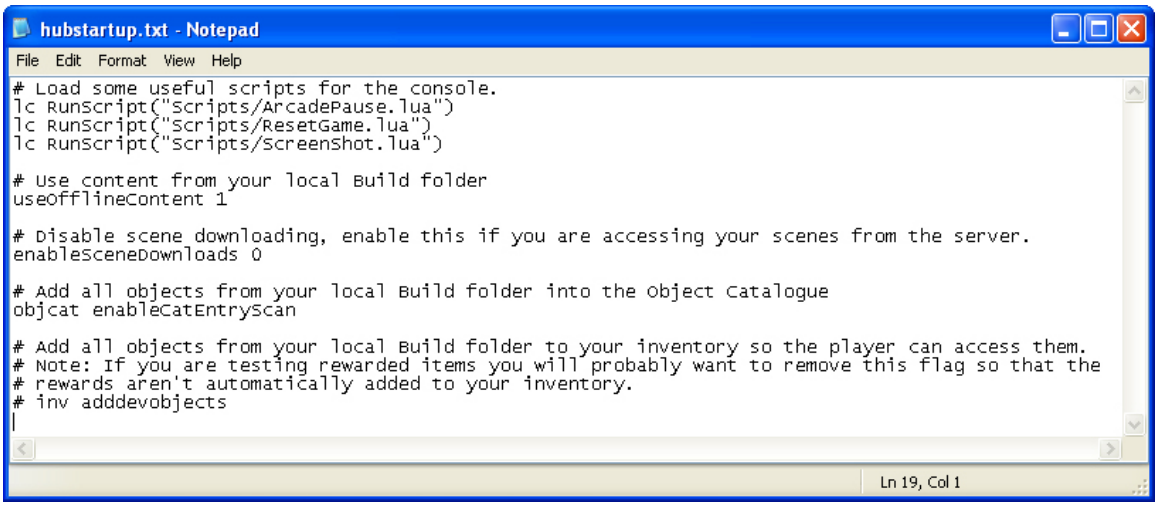


> ⚠ Pressing the TAB key to invoke the auto-complete function does not work when you use Target Manager to issue commands.

Executing Console Commands at Boot Time

When PS Home boots, it loads a text file: <HDK_INSTALL_DIR>\build\scripts\hubstartup.txt. This file issues commands that affect how PS Home runs. If there is a command that needs to run every time you run PS Home, consider adding it to your hubstartup.txt file. An example hubstartup.txt is included with the HDK.

It contains the following commands:



It calls the following Lua scripts using the console command lc:

- ArcadePause.lua: Press F1 to pause or resume the currently active arcade game.
- ResetGame.lua: Press F2 to reset the currently active arcade or mini-game. This action destroys the game, reloads the game's scripts, then restarts the game.
- ScreenShot.lua: Press F4 to save a .png image of the screen to your <HDK_INSTALL_DIR>\build directory.

It also calls the following console commands:

- useOfflineContent 1: This command disables downloading of various content (such as the inventory) from the server and defaults to using content in the build folder instead
- enableSceneDownloads 0: This command disables downloading of online scenes and defaults to loading scenes on the hard

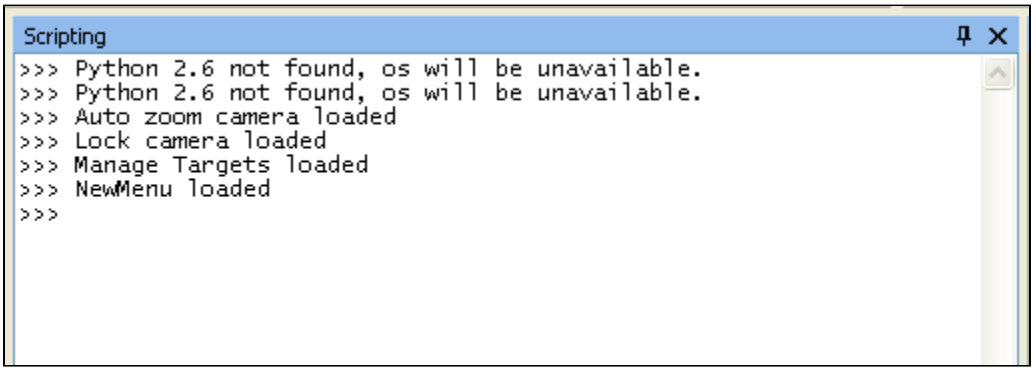
disk instead
- objcat enableCatEntryScan: This command scans the <HDK_INSTALL_DIR>\build\objects folder for objects to add to the inventory.

The hubstartup.txt file provides these commands to facilitate development and testing of content. However, you can edit the hubstartup.txt file as you like.

For more information on updating your hubstartup.txt file for testing content online, see Testing Online Content.

Executing Console Commands through the Scene Editor

The Scene Editor has a Scripting window that enables you to run HDK API scripts. You can also use it to enter console commands:



Before you can enter console commands, you must first connect the Scene Editor to a target. For information on connecting to Target Manager, see Launching a Scene in PS Home.

Taking Screenshots

You can take screenshots of the Home client in the following ways:

- Press the F4 key on your keyboard.
- Use a Debug Console command.
- Use the VRAM Viewer in Target Manager.

Taking Screenshots using the Debug Console

To take screenshots of PS Home using the Debug Console, you use the shot command.

If you issue the command with no parameters, by default the screenshot is saved back to the file serving directory on the host PC. It is saved in .png format at the same resolution as PS Home's display buffer. The default filename takes the format shotXX.png (where XX is a unique number that increments for each new screenshot). You can specify a custom filename with a .png, .tga or .jpg extension. The screenshot is then be saved using the corresponding encoder.

When running the Home Developer PKG, screenshots are saved onto your development hardware's hard drive by default. You may find it more convenient to save the file to an external USB storage device. To do this, enter the following command:

```
shot usb:<filename>.png
```

Hiding the Debug Console

When you take a screenshot, you may not want the Debug Console to be visible.

- To exclude the Debug Console from the screenshot, enter the command: consoleNumLines 0

  This command hides the console but does not deactivate it, so you can still issue the shot command.

- To redisplay the console, issue the command: consoleNumLines 10

> You can vary the number of lines but note that you can view a maximum of about 55 console lines.

Flight Mode

You can also switch the camera into flight mode. For more information, see Reference Tool Dip Switches.

Taking Screenshots via Target Manager

Target Manager has a built-in feature called VRAM Viewer that you can use to take screenshots from connected targets. For this to work correctly, you must set the capture settings for VRAM Viewer correctly.

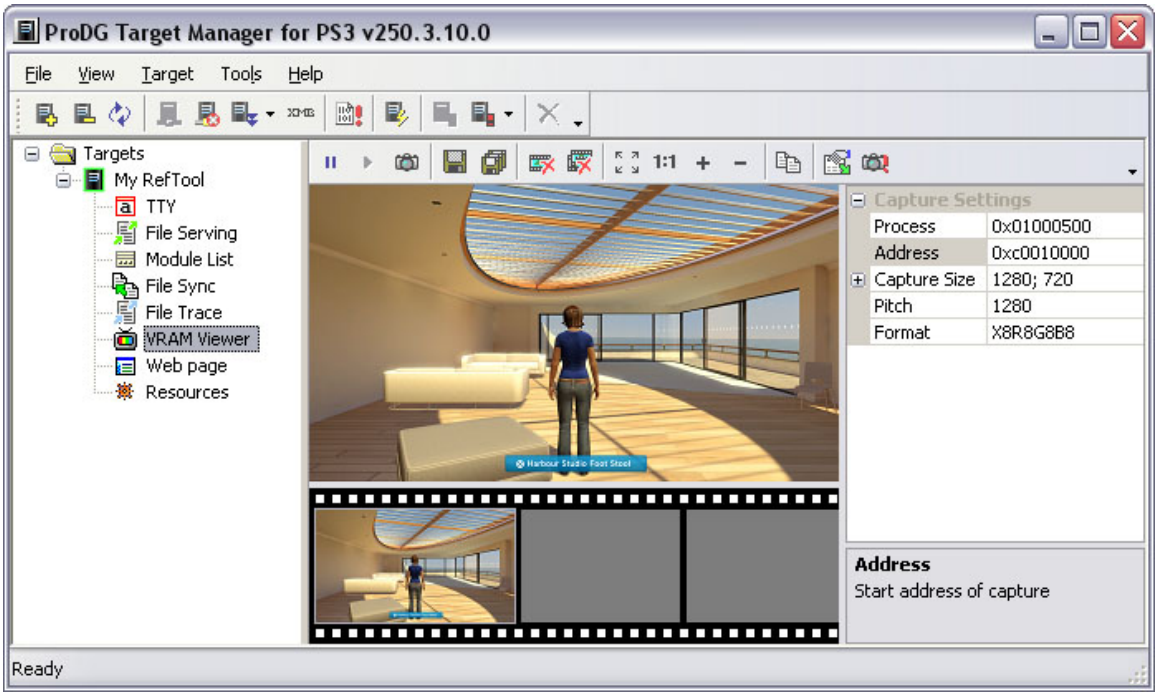
To set the capture settings for VRAM Viewer:

1. In Target Manager, expand the target connected to a running instance of the Home client.
2. Select the VRAM Viewer.
3. Enter the following values:

Field	Value
Address	0xc0010000. This field is in the Capture Settings panel.
Capture Size	1280; 720.
Pitch	1280.
Format	X8R8G8B8.

4. Click Capture to take the screenshot.

Target Manager displays the screenshot as follows:



Troubleshooting the VRAM Viewer

If you have problems taking screenshots with the VRAM Viewer, try one of the following:

- Click  to reset the capture settings, then take the screenshot.
- Download and install the latest version of Target Manager.

Debug Console Command Reference

The following table lists useful commands that you can enter into the Debug Console:

Command	Valid Parameter(s)	Default Value(s)	Description

atgParticlesReloadEffects	N/A	N/A	Reloads all active particle effects (.efx files) from disk. This is very useful for allowing a quick turnaround between authoring in the ATG Particle Effects Tool and visualization in-game.
atgParticlesRenderEnable	false, 0, off, true, 1, on	true	Disable/enable rendering of all particle systems.
atgParticlesUpdateEnable	false, 0, off, true, 1, on	true	Disable/enable per-frame evaluations of all particle system (effectively pause/unpause time for particles).
consoleFont	debugFont, consoleFont, smallFont	consoleFont	Change the font of the Debug Console.
consoleNumLines	0 to 55	12	Change the number of visible lines displayed by the Debug Console.
debugCameraCollision	false, 0, off, true, 1, on	false	Allows the debug camera to be affected by collision.
debugCameraPickCollisionItem	false, 0, off, true, 1, on	false	Allows you to pick items in the scene (excluding scene geometry) to see their collision. You must be in free camera mode.
debugCameraPickerHitPos	off, 0, false, true, 1, on	false	When using the debug camera, it tells you the <x,y> coordinates of the cursor.
debugCameraPickerLength	0 to 133	10	Sets the distance from the camera at which the debugCameraPick starts to highlight collision. You must have debugCameraPickCollisionItem set to on.
debugClothingLimits	false, 0, off, true, 1, on	false	When enabled, this allows developers to view the guideline limits to the Avatar's clothing component sizes.
EnableArcadeGameTimeout	false, 0, off, true, 1, on	true	Toggles the arcade game timeout. The timeout boots a user from the arcade game if they are idle for 3 minutes.
enableAudioRender3D	0 (off), 1 (show all emitters), 2 (show only emitters that user can hear)	0	Displays all the sound areas in a scene in the same colors as in the Scene Editor.
enableMemoryStats	0, 1, 2, 3	0	Displays a table showing all memory pools and their usage. If set to 1, the original version of the memory stats is displayed. If set to 2, a newer, more detailed version is displayed. If set to 3, a version is displayed which consumes less screen real estate.
enableNetStats	false, 0, off, true, 1, on	false	Displays the scene's network stats (for example, Net Free memory, LibNet Free memory).
enablePrivateLobbies	false, 0, true, 1	false	If set to true, forces you into a private instance of that space.
enableVideoStats	false, 0, true, 1	false	Displays the memory stats for videos in your scene.
envShot	map size (power of 2, greater than 4, max 512) filename (.tga)	N/A	Using the current camera position, generates the 6 sides of a cube map and saves them as a single image in .tga format to file serving root.
filecache	list, clear, fill, remove	N/A	Allows you to view the scene's caches, the amount (in KB) stored, and to empty it.
home	N/A	N/A	Relocates to the 'Harbour Studio' personal space.
HomeStore.Open	<commerce point path> e.g. ThreadsCP.xml	N/A	Opens a commerce point for testing.

idleTime	0 to MAXFLOAT	3540.0	Specifies the time of user inactivity in seconds before PS Home disconnects the client from the servers. Applies to online mode only.
inv adddevobjects	N/A	N/A	Temporarily includes all objects in the build directory within the object catalogue so that when the .self is launched it is easy to browse for any object on your HD.
inv adduserobj <OBJECTID>	N/A	N/A	Add the specified object to the user's inventory.
inv clear	default, user, paid, all	N/A	Remove objects from the user's inventory by type: default, user, paid, or all objects.
inv list default	N/A	N/A	Lists all object IDs of objects in the user's default inventory.
inv disableValidation	N/A	N/A	Skips the check that normally prevents non-reward items from being added into a user's list of rewards. Useful when you want to load the client without the client first removing all commerce items from your inventory, requiring you to manually re-enter them when in PS Home.
inv list paid	N/A	N/A	Lists all object IDs of purchases in the user's inventory.
inv list user	N/A	N/A	Lists all object IDs of objects in the user's inventory.
inv refreshnp	N/A	N/A	Re-requests purchase information from the Network Platform (NP), to rebuild the inventory of a user's purchases.
inv removeuserobj <OBJECTID>	N/A	N/A	Removes the specified object from the user's inventory.
lc	Arbitrary Lua command(s)	N/A	Executes Lua code in the currently active Lua environment.
map	Valid scene name in scene list	N/A	Relocates the local player to the specified space. The specified name does not need quotes. The name is taken from build\environments\LocalSceneList.xml if launching offline, or it is the name supplied to the Content Delivery System (CDS) if the scene has been uploaded to a content server.
mapregion	N/A	N/A	Previously it allowed you to select the region PS Home loaded into. It has been replaced with the region select button in the PS Home start-up screen.
objcat	See objcat	N/A	Performs a search in the object catalogue. For more information on using this command, see objcat.
osd	N/A	N/A	Enters the On-Screen Display (OSD) sub-console. You can use the ls and cd commands to traverse the OSD hierarchy.
osdhide	false, 0, off, true, 1, on	false	Disables all OSD, including system dialogue boxes.
profileEnable	false, 0, off, true, 1, on	false	Enables timing bars showing what PS Home is spending time doing.
ProfileGui.Enable	false, 0, off, true, 1, on	false	Displays the new profile GUI, which allows you to call any profiling information that you want to display. Supports a mouse.
ProfileGui.ControllerMouseEmu	false, 0, off, true, 1, on	false	Allows you to use the controller to emulate a mouse.
ProfileGui.ClearPanels	N/A	N/A	Clears any panels that you created with ProfileGui.CreatePanel.
ProfileGui.CreatePanel	Info, MemStats, Network, Cpu, Gpu	N/A	Creates a profiling panel of the specified type (such as Info, MemStats).

ProfileGui.ListObjects	N/A	N/A	Lists all objects and their UUIDs in the scene. You can also choose to open the panel in a minimized state by typing min after the panel name. For example, ProfileGui.CreatePanel Info min)
ProfileGui.ListPanels	N/A	N/A	Provides a list of possible Profile panels that you can use with Profile.CreatePanel.
profileMode	off, overview, meshtiming, overdraw	off	Allows you to set one of four profiling modes. For more information, see Profiling in the Client.
Quartermaster.OverrideLodhi, med, low, none	hi, med, low, none	none	Switches the LOD of the avatar clothing from high (LOD1), medium (LOD2) and low (LOD3). Do not use this console command in the Wardrobe.
reloadscene	N/A	N/A	Reloads the scene. Useful when testing in your sandbox.
reloadTextures	-a	No parameters	A fast way to reload all the textures in the scene memory slot. Use -a to reload all the textures in the game. Use no parameters to reload all textures in the scene memory slots. Only textures that have not changed in total data size since the creation of the texture will succeed. An unsuccessful reload attempt leaves the original texture unchanged and a console warning message is generated.
runSpeed	0 to MAXFLOAT	3.5	Changes your avatar's run speed. Useful for running around big environments quickly.
screenDebug	all, video	N/A	Displays general screen-related information, or information only on screens that are playing video.
SetDipSwitch	switch (0 to 7), state (0 or 1)	All switches set to off (on test kit)	Simulates a reference tool's dip switches on a test kit. Certain functions of the Home client activate in response to the setting of these switches. See Reference Tool Dip Switches.
setLanguage	A localization region in the form xx-YY E.g. en-GB	N/A	Partially changes the language used, by reloading specific localization files for the given language. This is for quick testing of localization text, for Objects etc. It is not intended as a PlayStation®Home-wide language change, as many Home systems translate their text once only, so changing the language completely would mean re-initializing them, where it would be easier and safer to just restart Home using a different language setting.
setNumNPCs	0 to 63	N/A	Spawns wandering Non-Player Characters (NPCs) from 0, 0, 0. Useful for artificially load testing your spaces.

setOverrideServerTime	yyyy mm dd hh mm ss	N/A	Allows you to set a fake server time. You can set a time, and when you query the server time it returns your set time plus the time elapsed since. For example, if you set the override time to be 2 minutes before an event is due to start, it starts 2 minutes later. This is particularly useful for testing if something triggers correctly at a specific time.
shot	Optional filename	shotXX.png (where XX is a number)	Saves a .jpg, .png or .tga screenshot back to the file-serving root. When running the PS Home Developer PKG, this is /dev_hdd0/game/NPIA00010/USRDIR. To save the screenshot to a USB drive, use shot usb://filename.png.
showPersonLabels	false, 0, off, true, 1, on	true	Toggles the rendering of the PSN name labels above avatars' heads. Sometimes useful for taking screenshots.
showTargets	false, 0, off, true, 1, on	false	Shows debug information about targetable areas that the player is in, including a 'Target Score' which shows which of two trigger labels will be displayed if the player is within both
sleepTime	0 to MAXFLOAT	180.0	Specifies the time (in seconds) before the local avatar 'goes to sleep' (It displays Zs and the 'Back soon…' message)
teleport	x, y, z, rotation(radians) All 0 to MAXFLOAT	N/A	Specifies a location to move your avatar to in the currently loaded environment. The rotation angle sets the avatar's heading around the vertical world space axis.
walkSpeed	0 to MAXFLOAT	2.2	Changes your avatar's walk speed. Useful for walking around big environments quickly.
wireFrameEnable	false, 0, off, true, 1, on	false	Renders the scene with textured wireframe.

objcat

Parameter	Explanation	Further parameters
clear	Empties the object catalogue.	N/A
disableCatEntryScan	Disables the scanning of the build/objects folder for locally edited objects on startup.	N/A
dumpone	Dumps the catalogue entry for a single object to the TTY.	<object UUID>
enableCatEntryScan	Enables the scanning of the build/objects folder for locally edited objects on startup.	N/A
getodc	Retrieves an object description.	<object UUID>
load	Loads in an XML database.	<catalog_filename>
memstats	Prints object catalogue memory diagnostics.	N/A
refresh	Rescans the local folder build/Objects for new catalogue entries.	N/A
search	Performs a basic search on the catalogue.	<search_fields> <search criteria> (<order by> < pagenum> <rowsPerPage>)
sqlexec	Executes an arbitrary SQL statement on the object catalog database.	<query>
sqlexecscript	Executes a script of arbitrary SQL statements on the object catalog database.	<script_filename>
verboseSql	Turns on printing of SQL queries and results rows.	on, off

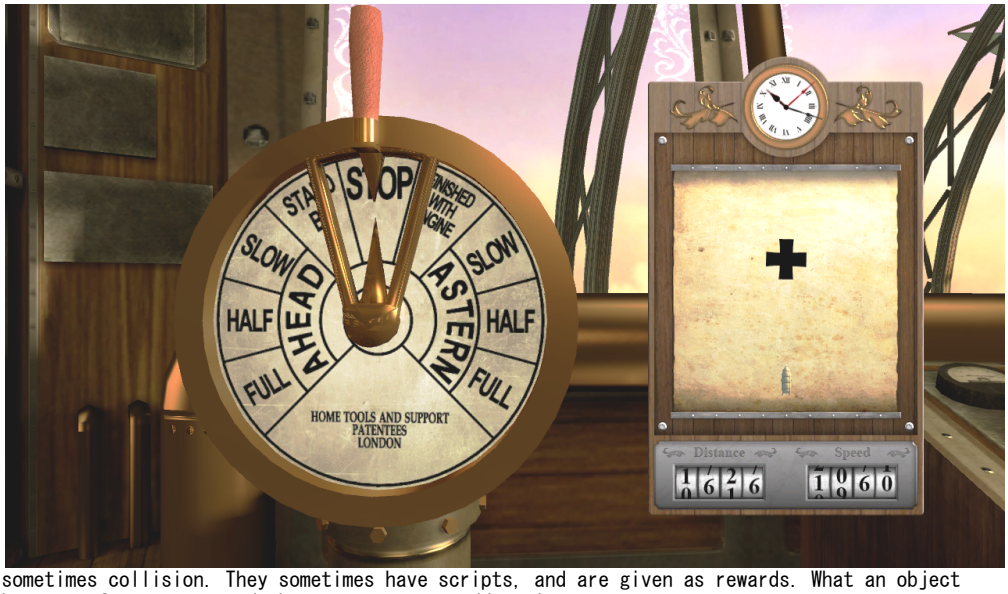
PlayStation®Home Error Codes

The following table lists errors that you might encounter when using the PS Home Developer Client:

Code	Description
C-931	Server-side error
C-991	Server-side error
C-995	NP account in use, the PS Home Developer client requires a SP-INT account
D3	An error occurred
D25	Server incompatible with current client
D3505	Disconnected from server
D4512	Server is not responding
D5028	Server connection refused
F1	An error occurred
F2	Incompatible client version
F3	Incompatible server version
F4	Server is at capacity
F5	Server does not support world id
F6	Failed to authenticate signature
F7	Encryption failure
F8	Access Key failure
F9	Failed to establish aux UDP connection
F10	The server became unavailable
F11	The session was ended
F12	The game was ended
F13	The session timed out
F14	Disconnected due to poor network performance
F15	User account banned from service
G-1	An error occurred
M-16	The user management/chat server connection was lost during the relocation process
Z(7,-1)	Selected scene not available on the current content server, delete your save data
Z(7,-2)	The space instancing server is not responding
Z(7,-3)	Error downloading content
Z(7,-4)	Cache error, delete the PS Home client from the XMB and try again
Z(7,-5)	Account age too young to access Home in this region
Z(7,-6)	The service is temporarily closed, try again later.
Z(7,-7)	Save data copied from another account, delete your save data
Z(7,-10)	PS Home client out of date, please update
Z(7,-11)	A moderator has ejected you from PS Home
Z(7,-12)	Account region does not have access to PS Home
Z(7,-13)	Regional server error
Z(7,-20)	Disconnected due to inactivity

Z(7,-100)	Failed to download user inventory
Z(7,-101)	Failed to download NP tickets, possibly due to too many Service IDs
Z(7,-102)	Failed to download user entitlement list
Z(7,-103)	Entitlement retrieval failed
Z(7,-104)	Inventory is corrupt

Objects



Objects are made up of any number of assets (or resources) and functional definitions (or components). Objects are essentially definition containers that identify the assets, paths, names and other file properties of what they contain within an XML structure.

In PS Home, objects are items that users can wear (clothing), place (furniture), or use (active items and inventory items). You can also place certain objects in scenes through the Scene Editor (for example, mini-games). When a collection of resources and components are wrapped as an object, they can be easily referenced and re-used in environments in PS Home.

Objects are sometimes model files, sometimes collision. They sometimes have scripts, and are given as rewards. What an object does or what it can do depends on the type of component and the resources you add to it.

You can make the following objects in PS Home:

- Active Items
- Animation Packs
- Arcade Games
- Character Components
- Companion Objects
- Furniture and Decorations
- Game Launch Objects
- Group Animation Packs
- Interaction Packs
- Locomotion Objects
- Mini-games
- Portable Items
- RealTime Games
- Resource Packs
- Sound Packs

You can also embed objects in scenes. See Embed Objects into Scenes.

> See also Object Information for details on scripting objects.

Active Items

For Japanese Live Page
最新の日本語版ページはこちら 「アドバンスド家具 (Active Items)」

Active Items are objects that the user can place within a personal space or clubhouse, in a similar way to furniture. Developers export the objects from Maya's furniture, then add a script component using the Object Editor.

An Active item must have the following components:

Component	Description
Furniture component	Allows users to select the Active Item when they select Redecorate from the Menu Screen.
Active Item component	Adds script capability to an item. Furniture light functionality is removed when this is added. You must add lighting to your script in order to regain furniture lighting behavior. ⚠ Active Items are limited to two seats. ⚠ Active Items cannot be listener objects, using the function `Object.SendListenerData()`.

To create an Active Item with scripted components, add the following:

- a MiniGame Component to the object
- a Lua Environment Component
- an Game Spawner Component
- a Targetable Component

> ℹ The components are automatically added if you use the Object Editor Wizard

Developers can create an additional collision file for an Active Item that is used specifically for collision when the script component of the object is activated. For more information on Active Item collision, see Collision For Active Items.
If an active item is created from furniture, it will have furniture safe collision - this needs to be edited to create an active item.

Moving and Placing Active Items

Active Items are not restricted to being placed in fixed positions in specific scenes. Users can select them and place them anywhere within a personal space or clubhouse, as long as there is room for the item, including the active collision shape. For information on Active Items and the number of standard furniture slots they occupy, see Furniture Block System.

Only users can place and move Active Items. The owner of a clubhouse can place Active Items and they persist even when the owner has left the space.

You cannot use Lua script to change an active item's placement in a scene. If you use a script, for example by calling `Entity.GetPosition()`, the active item seems to move, but at the next `Update` frame, it automatically moves back to its original position.

For example, an active item is placed at (1,0,0,0), and its update function is `OnUpdate()`, then:

```
activeEntity = Active.GetEntity();

function OnUpdate()


    print( "START -- " .. tostring( Entity.GetPosition(activeEntity)))


    --Move the Active Item

    Entity.SetPosition(activeEntity, Vector4.Create(0, 2, 0, 0) )


    print( "END -- " .. tostring(Entity.GetPosition(activeEntity)))


end
```

This code produces the following output:

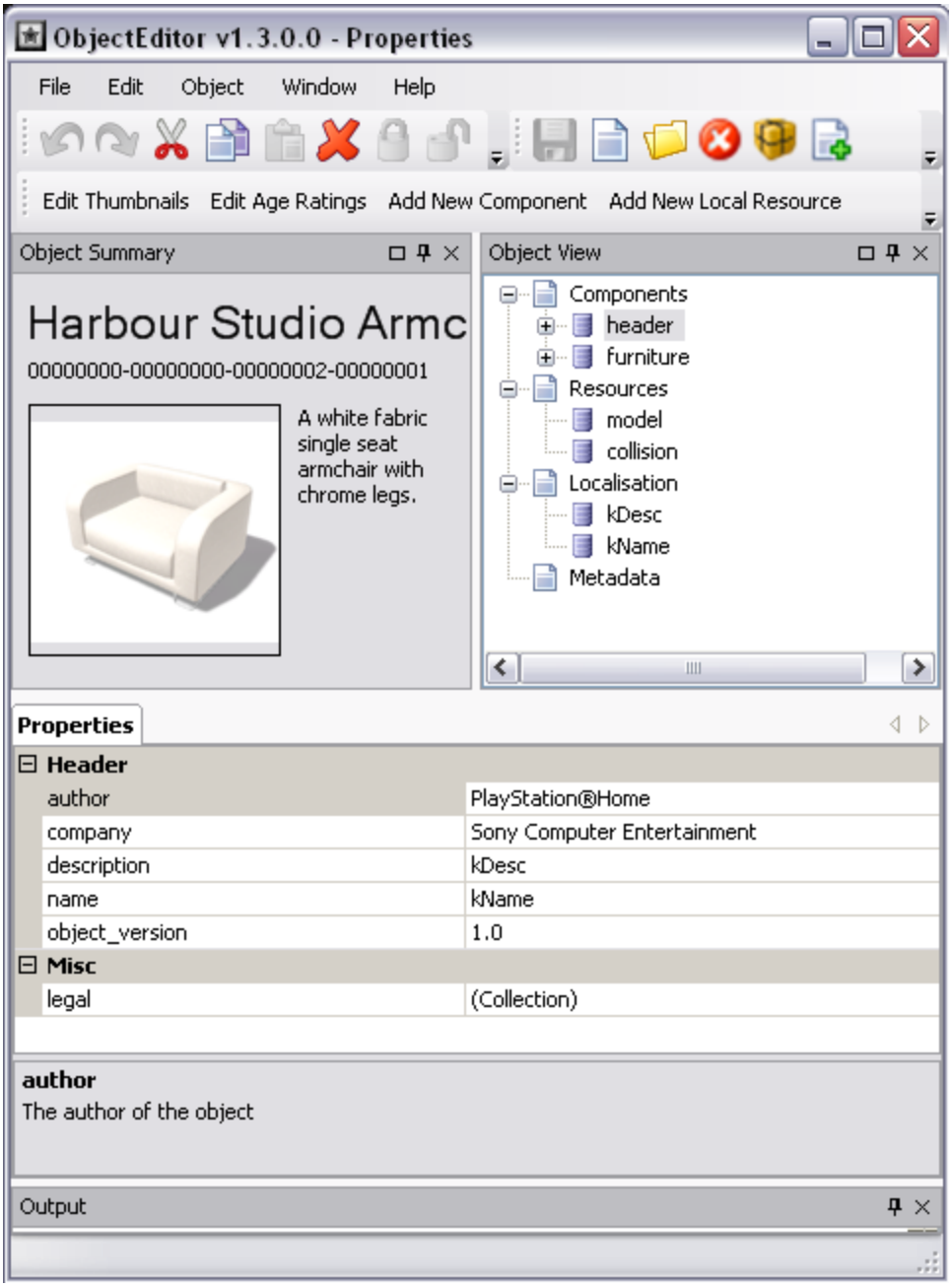
Lua	Vector	Explanation
print("START - "	[1, 0, 0, 0]	This is where the active item was placed by the user.
print("END -- "	[0, 2, 0, 0]	This is the position to which the script moved the active item.
function OnUpdate()		Resets active item position.
print("START -- "	[1, 0, 0, 0]	When OnUpdate was called it reset the active item position to where the user originally placed it.

Adding Mini-game Components

One of the ways you can create an Active Item is to add a add a Mini-game to the object. Alternatively, you can add just the Lua Environment and Active Item components to run a script (see Adding Script Components).

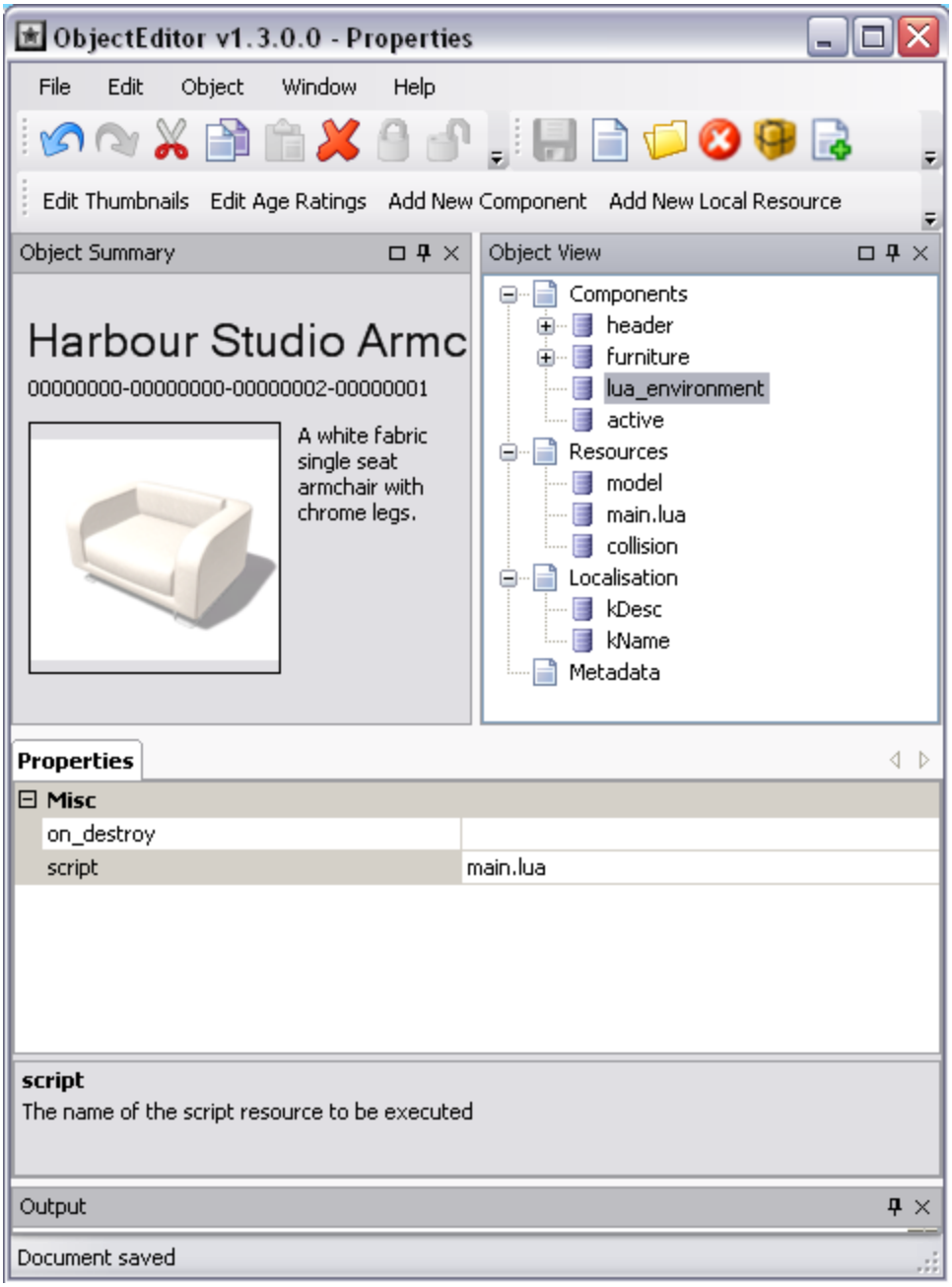
To add a MiniGame component to an object:

1. Build and export a furniture item. See Furniture Creation.
2. In the Object Editor, select the exported item of furniture. For example:



3. Associate the furniture with the Lua environment, as described in Adding Script Components. Note that for mini-games you do not need to add the OnUpdate() property.

   The furniture item in the Object Editor should now look something like this:



4. Click Add New Component in the toolbar to add the MiniGame component and set the properties that are required by your script (for more information on these properties, see Mini-games).
5. Click Add New Component in the toolbar to add the Game Spawner component and select Mini Game in the Component Options dialog so that the component spawns a mini-game. The following properties can be set when you select the Game Spawner component in the Object View panel:

    a. local_person_visible: Select whether the local player's avatar is visible in the nini-game or not.
    b. remote_people_visible: Select whether the avatars of players not in the game are visible while playing the game.
    c. remote_portables_visible: Select whether the inventory items (portables) are visible or not.
    d. use_default_camera: Choose whether to use the default camera in the Mini-game or define the control yourself within Lua.

6. The session size must be defined in the network component. Click Add New Component in the toolbar to add the Network component.
7. Create an XML file with the following format:

```
<NETWORK_DEFS>
<SESSION size="X" />
</NETWORK_DEFS>
```

(where X is the maximum number of players)

For more information on game sizes, see Maximum Number of Players in Mini-Games.

> ⚠ If you are going to use this Mini-game as part of a scene, by adding it through the Scene Editor, the Mini-game Player Slots property must be the same as the SESSION size defined in the XML.

8. Click Add New Local Resource in the toolbar and add your XML file to the active item.
9. Click on the Network component in the Object View panel and select the XML file from the definition drop-down list in the Properties panel. The network XML is now attached to your active item with the session size set.
10. Run HomeDeveloper.self from the Target Manager with no command line parameters. This takes you to the default personal apartment. Access the PS Home Menu via the Start button on the connected controller and select Redecorate Personal Space. Select and place your furniture item. You should see something like the following:

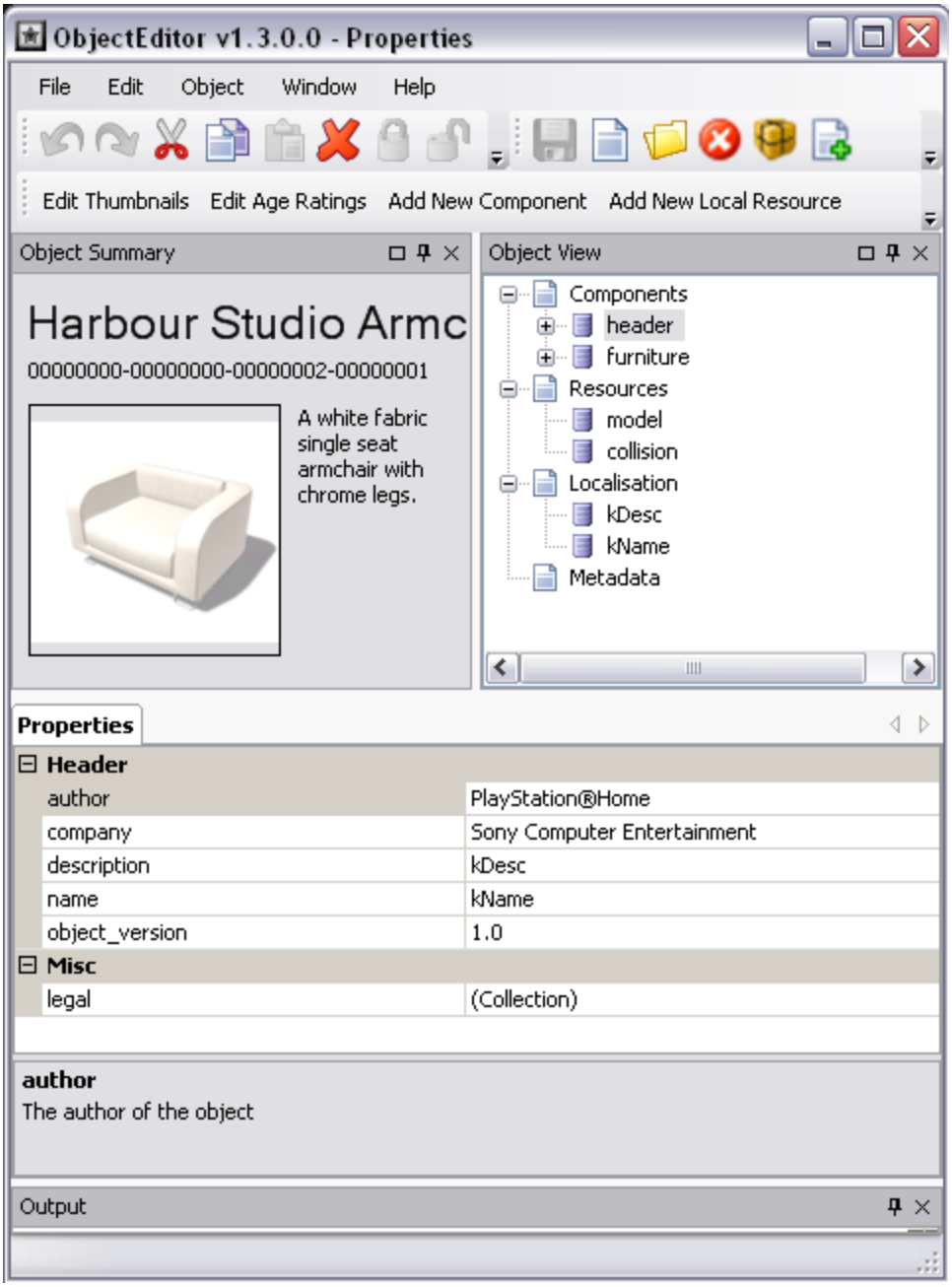


Adding Script Components

You can create interactive or scripted behavior within an object in the user's personal space.

To add script components to an object:

1. Build and export a furniture item. See Furniture and Decorations.
2. In the Object Editor, select the exported item of furniture. For example:



3. Associate the furniture with the Lua Environment by clicking Add New Component in the toolbar, then selecting the Lua Environment component.
4. Use the API function `Active.GetEntity` within your Lua file to return the entity of the active item. This is so that you can attach other entities to it or query it (for example, to find the position of the active furniture within the scene).

   When you have returned the active item entity, you can use other functions such as `Entity.AttachToParent`, which allows you to apply the parent's world matrix to the entity so that the entity's position, rotation and scale are relative to the parent.

   The following example shows a simple Lua script using the `Active.GetEntity` and `Entity.AttachToParent` functions:

```
LoadLibrary("Entity")

LoadLibrary("Active")

entityActive = Active.GetEntity()

entityBighead = Entity.Create()

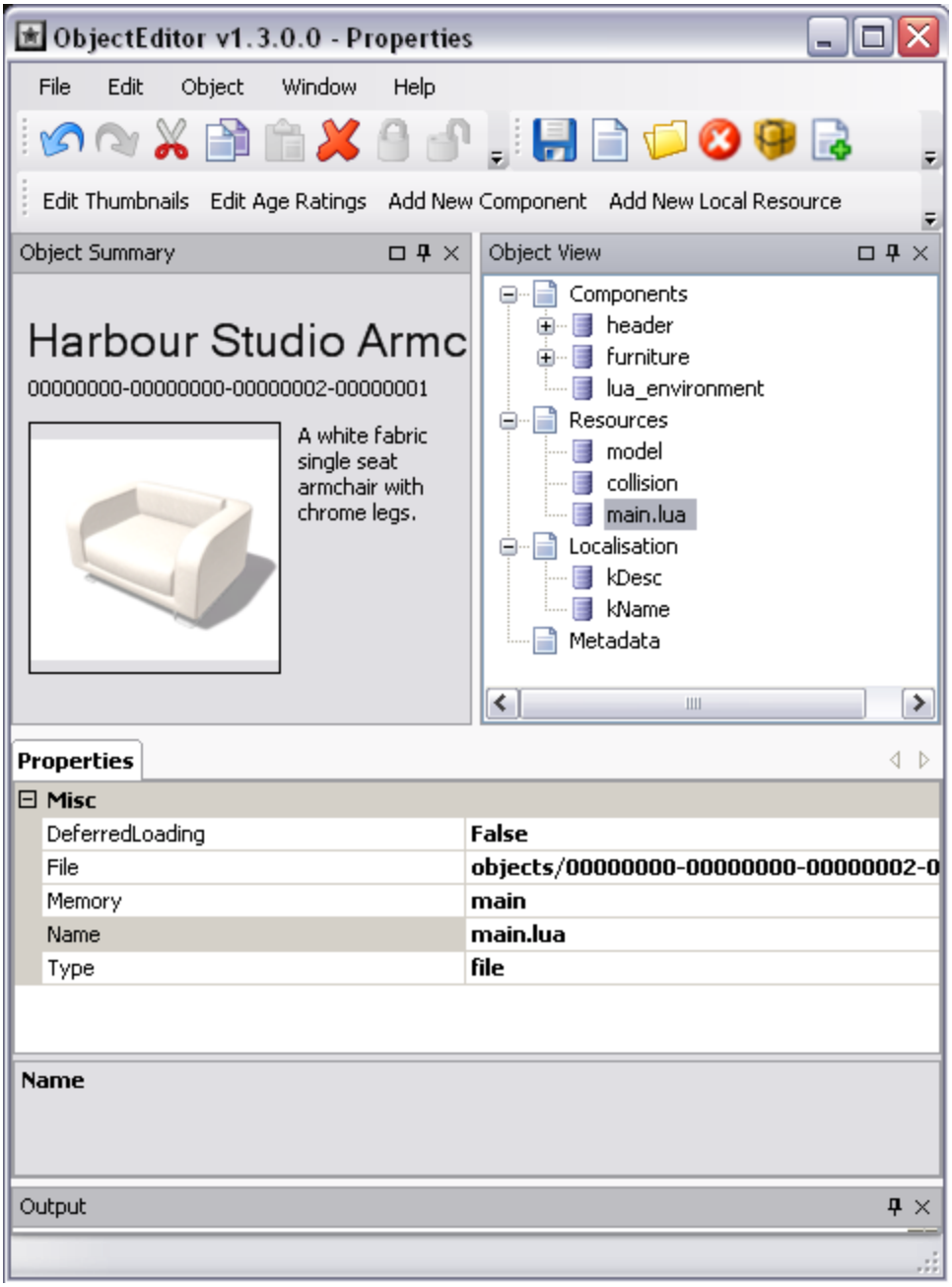
entityBighead:SetModel("bighead")

entityBighead:AttachToParent(entityActive)
```

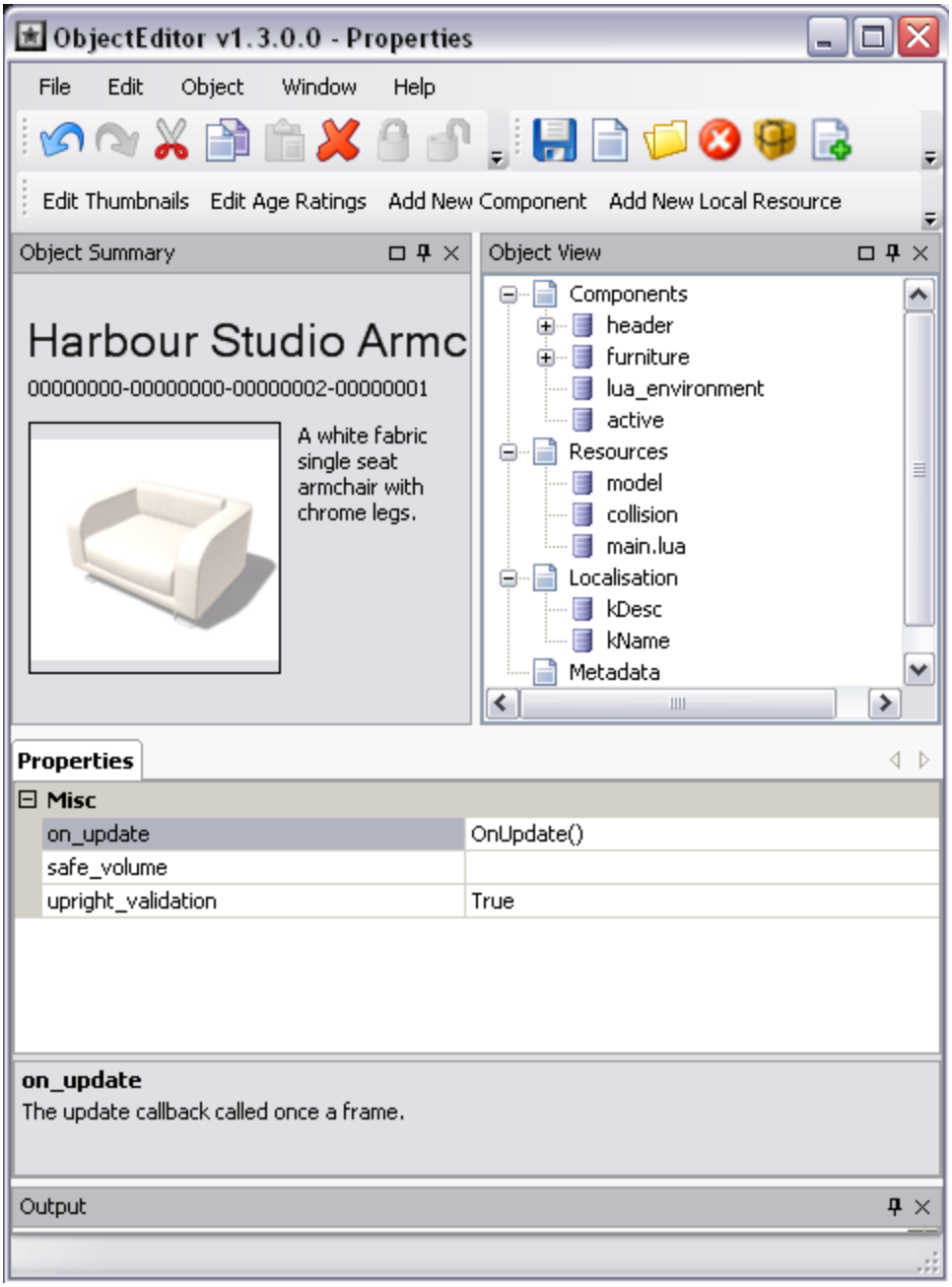
You could use this script, for example, to manipulate the model of a big head. If the active item is a table, the script attaches the model of the big head to the table. When the user moves the table around, the big head remains attached to it.

5. Click Add New Local Resource in the toolbar to associate the script with the furniture, and select your Lua file.

   The furniture item in the Object Editor should now look something like this:



6. Select the Lua Environment component in the Object View panel and set the script property to your Lua file (for example, main.lua).
7. Click Add New Component in the toolbar and select the Active Item component (unless active collision was created and exported from Maya, in which case the active component already exists). Add OnUpdate() to the on_update property in the Properties panel:



The Active Item component uses upright_validation to determine whether the item can be used only when upright. Set this to False if the item can be used even if it has been rotated by the user and is no longer upright.

8. Run HomeDeveloper.self from the Target Manager with no command line parameters. This command takes you to the default personal apartment.
9. Access the PS Home Menu from the Start button on the connected controller, then select Redecorate.
10. Select and place your furniture item. You should see something like the following:



You can also create an active item by adding a mini-game to the object. See Adding Mini-game Components.

Portable Objects

A portable object is an object that can be deployed in most spaces (unlike Furniture and Decorations or Active Items that can only be deployed in Personal Spaces and Clubhouses).

Portable objects is a collective term for the following objects:

- Companion Objects
- Animation Packs
- Locomotion Objects

Animation Packs

See Avatar Animation Packs

Locomotion Objects

For Japanese Live Page
最新の日本語版ページはこちら 「ロコモーションオブジェクト」

A Locomotion object is a portable item that allows users to get around Home in style.

Locomotion objects support the following features:

- Configurable avatar movement speed

- Up to 2 custom models attached to the avatar
- Custom animations (for both the avatar and the attached models)
- Sounds
- Particle effects

For a step-by-step guide on creating a new Locomotion object see Creating Locomotion Objects.

For details on how to configure and customize Locomotion objects see Configuring Locomotion Objects.



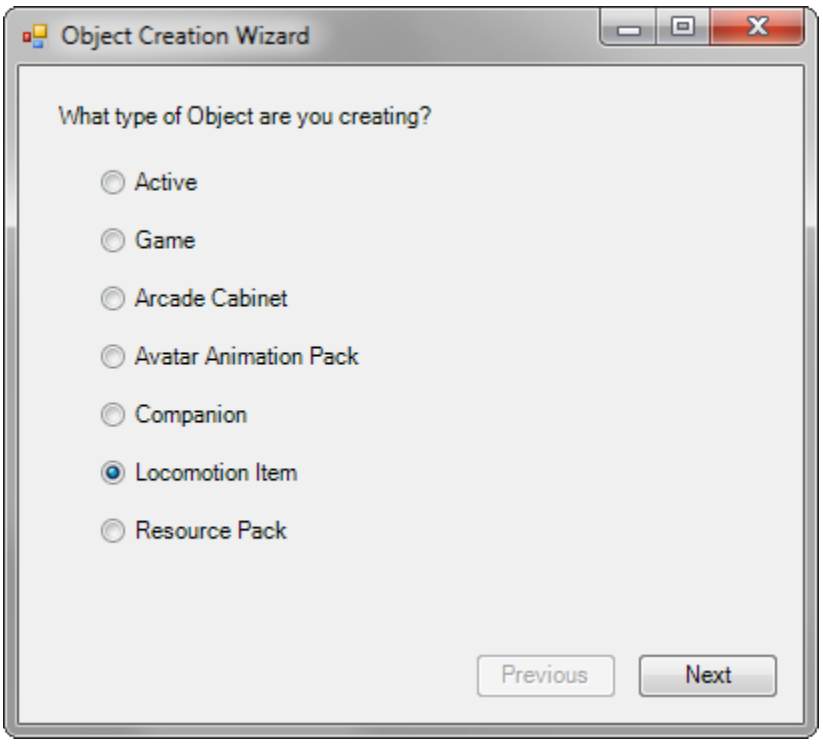
Creating Locomotion Objects

> For Japanese Live Page
> 最新の日本語版ページはこちら 「ロコモーションオブジェクトの作成」

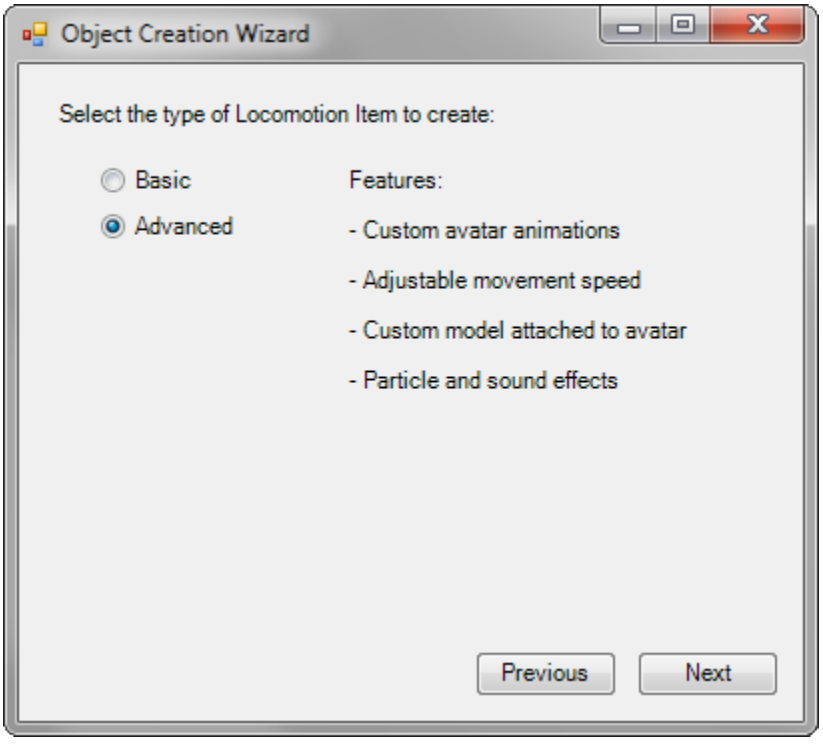
Creating a Locomotion Object

To create a new Locomotion object:

1. In the Object Editor, select File > New Object Wizard.
2. In the Object Creation Wizard dialog, select Locomotion Item and click Next:



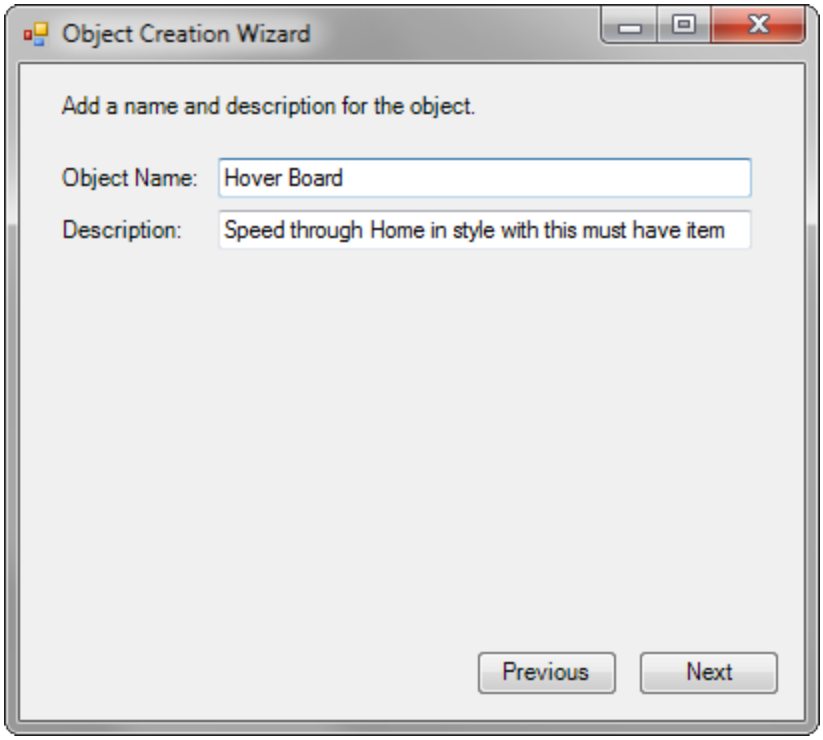
3. Select either Basic or Advanced, depending upon the type of Locomotion item you want to create.
This option affects the default configuration file and sample resources that the Locomotion object will contain.



The Basic Locomotion item provides the simplest default setup with the Locomotion object only adjusting the avatar movement speed and replacing the default locomotion animations with a faster running animation.

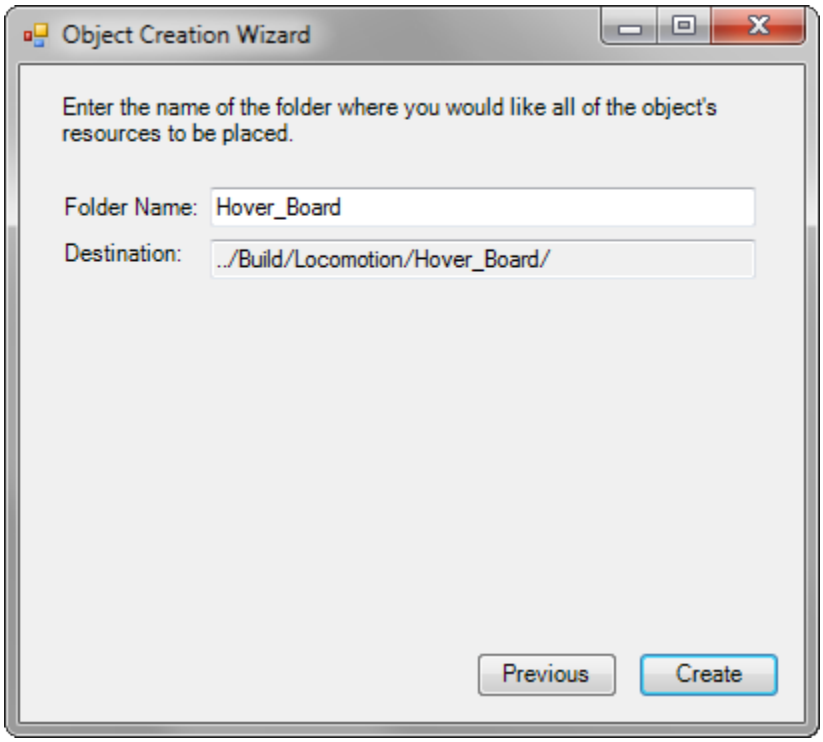
The Advanced Locomotion item provides a more complex setup with a model attached to the avatar's feet and advanced examples, including use of sounds and particle effects.

4. Enter the name of the Locomotion object, and provide a description.



> ⚠ You must provide a description. This description is visible to users when they select the Locomotion object. For text character limits, in bytes, see the HDK Tools Validations.

5. Enter the name of the folder where you want the Locomotion object's resources to be placed:



6. Click Create to complete the wizard, a new Locomotion object is created at: <HDK_INSTALL_DIR>/build/Locomotion/<Locomotion Resource Folder Name>/.

   This folder initially contains all the default template assets that are required for the Locomotion object to work (i.e.

the model, repertoire and animation files listed previously). You must replace these files with your own. The folder also includes all the other assets required for the Locomotion object to work in PS Home.

7. Preview the Locomotion object in PS Home.
    - To launch into the HomeDeveloper.self, see Launching the Home Developer SELF.
    - To add items to your inventory in the HomeDeveloper.self, see The DEV DEBUG Menu.

Configuring Locomotion Objects

For Japanese Live Page
最新の日本語版ページはこちら 「ロコモーションオブジェクトの設定方法」

This is an in-depth guide of how to configure Locomotion Objects.

If you have not yet created a Locomotion Object you can find a step-by-step creation guide on the Creating Locomotion Objects page.

You can configure a Locomotion Object through the `config.xml` file, which is referenced in the object's resources.

This page details the customizations that can be made in this configuration.

Example

A lot of PS Home users want the ability to run faster. So we use the Object Editor to create a new Basic Locomotion Item. The `config.xml` file is simple enough:

```
<xml>
    <properties run_speed='5.25' walk_speed='3.5'/>
</xml>
```

This locomotion item changes the avatar speed settings allowing for faster movement around PS Home.

Check with your regional account representative about valid speeds for your object. The top speeds have value to the users and there will be pricing considerations when submitting objects with higher speeds

Repertoires

As well as changing the basic avatar properties, you can provide a repertoire that changes the avatar animations. The basic locomotion template's repertoire simply replaces the avatar's move state. Each rig (male/female) has four animations: walk, fast-walk, jog and sprint. These are linked together using the LocomotionState, triggering based on speeds all the way up to 5 metres per second.

The animation resources should be marked to defer load (and the repertoire component should be set up so that "`is_exclusive`" is True). This saves memory by only loading up animations for the current rig (male or female).

See the Avatar Animation Repertoires documentation for more information on how to set up repertoires.

Attaching Models

You can also attach up to 2 models to the avatar. Create a new object using the basic locomotion template. This gives you a sprinting avatar. You need to then create a model and assign it to the object via the Object Editor (and flag it to defer load). The model must fit within the memory pool of a remote portable (alongside whatever other resources you are adding).

In the resource editor, rename the model as `test.mdl` and then edit the `config.xml` for the object:

```
<xml>
    <properties run_speed='5.25' walk_speed='3.5'/>
    <local_model name='test.mdl'/>
    <remote_model name='test.mdl'/>
    <fsm>
        <state name='initialise'>
            <on_entry>
                <attach bone='head'/>
            </on_entry>
        </state>
    </fsm>
</xml>
```

The new elements here are: local_model, remote_model and fsm (finite state machine).

> Note that we specify the model twice. This is because we need to let the locomotion item know which model to use on the local client and which to use remotely - this allows for a lower level-of-detail version in the memory restricted evironment of the remote portable.

The fsm element is where the locomotion item presents its flexible nature. Within the fsm element you define a finite state machine which comprises of a number of states. In this example we define a single state, called initialise, and upon entering this state we command the locomotion item to use the attach command, specifying the "head" bone as the attach point. In this case, the model will be attached to the player's head.

You can name any avatar bone as the attach point, but if no bone is given, the model will be attached to the root of the player (at their feet).

Here's a list of the bones that are available in the avatar:

- "pelvis"
- "hips"
- "leftleg"
- "leftfoot"
- "rightleg"
- "rightfoot"
- "spine"
- "neck"
- "head"
- "leftshoulder"
- "leftarm"
- "leftforearm"
- "lefthand"
- "rightshoulder"
- "rightarm"
- "rightforearm"
- "righthand"

The states in the fsm are controlled by transitions, and in effect we are programming the item in a safe manner. The first fsm that the locomotion item comes across will be run on both the local and remote portable, but any others will only be executed locally (to keep the runtime cost of the remote portable down).

Config Specs

local_model

Specifies the model to use for the local portable.

Attrib	Mandatory	Default Value	Notes
name	yes	none	the name of the resource to load
skeleton	no	none	the name of a skeleton resource to attach
scale	no	1.0	model scaling from 0.1 to 5.0
id	no	1	you can have up to two models (1,2)
autohide	no	0	should the model hide when running foreign animations (0,1)

remote_model

Specifies the model to use for the remote portable - this can be the same as the local, or one with lower detail (or perhaps you don't care for animation on the remote portable and so leave out the skeleton).

Attrib	Mandatory	Default Value	Notes
name	yes	none	the name of the resource to load
skeleton	no	none	the name of a skeleton resource to attach
scale	no	1.0	model scaling from 0.1 to 5.0
id	no	1	you can have up to two models (1,2)
autohide	no	0	should the model hide when running foreign animations (0,1)

properties

Global properties for this locomotion portable.

Attrib	Mandatory	Default Value	Notes
run_speed	no	3.5	the run speed is limited to between 1 and 5.25 metres per second
walk_speed	no	2.7	the walspeed is limited to between 1 and 3.5 metres per second
autohide	no	0	can be 1 or 0. If 1, the model is hidden when playing external animations
sound	no	none	the name of an audio bank resource

We recommend that if you want to increase the walk and run speed that you always use the maximum of 3.5 and 5.25.

Note1 - the autohide property, is set to 1, will override the autohide settings made in the local_model/remote_model elements.
Note2 - the walk_speed must be less than the run_speed.

variables

This is a list in which you can define up to 6 user variables for use in the configuration. These are always floats.

```
<variables>
    <var name='x' value='0'/>
    ...
</variables>
```

Each var element has the following attributes:

Attrib	Mandatory	Default Value	Notes
name	yes	none	a unique name for the variable
value	yes	0	any number

The var name attribute must be unique and must not be the same as any internal variables (see the Read-Only Variables section below)

registers

You can control animation registers defined in the repertoire from the locomotion portable. Reading and writing of the repertoire registers is done differently (for performance reasons). Writing to repertoire registers is done by providing a list of user variables and registers - the locomotion logic will update the repertoire register from the user variable every frame:

```
<registers>
    <register name='x' value='y'/>
    ...
</variables>
```

Each register has the following attributes:

Attrib	Mandatory	Default Value	Notes
name	yes	none	the name of your lua writeable animation register
value	yes	none	the name of a source user variable

The user variable must have been defined before the register is defined. Reading registers is done manually (see later on).

fsm

This section contains the programmable elements. You can have several fsm elements, and all of these are run on the local portable. However, the remote portable will only run the very first fsm defined in the configuration (to keep the runtime profile small).

An fsm is defined by a number of states:

```
<fsm>
    <state name='initialise'/>
    <state name='on_jump'/>
    <state name='on_run'/>
</fsm>
```

All fsm elements must contain at least one state named initialise. This will be the first state entered when the fsm is started.

States contain the following elements:

```
<state name='some_state'>
    <transition condition='x1' goto='y1'/>
    <transition condition='x2' goto='y2'/>
    <on_entry/>
    <on_update/>
    <on_exit/>
</state>
```

The state can contain many transition elements, and one each of on_entry, on_update, on_exit.

Finite State Machine

Adding models and changing the player repertoire are the easy steps. To unlock the true potential of the locomotion item you need to provide the definition for a finite state machine (fsm). Within such an fsm you define several states.

Each state has the following properties:

- transitions - a transition is a test, which if true will cause the state machine to jump to a different state
- on_entry - a set of functions that will be called once (when we enter this state).
- on_update - a set of functions that will be continually called while in this state.
- on_exit - a set of functions that will be called once (when we leave this state).

For example:

```
<xml>
    ...
    <fsm>
        <!-- entry point for this fsm -->
        <state name='initialise'>
            <transition condition='mood==Happy' goto='smile'/>
        </state>

        <state name='smile'>
            <transition condition='mood~=Happy' goto='initialise'/>

            <on_entry>
                <set_anim name='ears_happy.ani'/>
            </on_entry>

            <on_exit>
                <stop_anim layer='1'/>
            </on_exit>
        </state>
    </fsm>
</xml>
```

Here we define two states, initialise and smile. The locomotion item will always jump to the state initialise on startup. In this case, the locomotion item will do nothing, until the user changes their mood status to Happy. When this happens, the state machine will jump to the smile state.

On entry to the smile state, we set an animation. Then nothing happens, until the user changes their mood status again to something that isn't Happy. In this case the state machine will transition to initialise. Before it does, it will call stop_anim as part of the on_exit functions.

You can have several fsm elements defined in the configuration file, but only the first fsm is run on the remote portables. So locally you will see everything, but other users will only see the work done by this first fsm.

Transition

Transitions control when an fsm should change state. A state can contain a number of transition elements, so different paths can be taken based on conditions.

The general format for a transition is:

```
<transition condition='C1' goto='S'/>
```

While in a state X, the transition test is run constantly. When C1 is true, the finite state machine will jump to state S.

We can set up a transition to test a number of conditions before changing state:

```
<transition condition='C1' goto='S'>
    <condition type='T' condition='C2'/>
    ...
</transition>
```

Here, C1 and C2 must be satisfied before jumping to state S. The type T can be one of the following:

name	description
and	C1 and C2 must be true before we can transition
or	C1 or C2 can be true before we can make the transition

So what can be tested? What could C1 or C2 possibly be? Here are a few examples before the rules are presented:

Example 1

```
<transition condition='speed>0.01' goto='moving'/>
```

This is a single condition test. When the player's speed is greater than 0.01 metres per second, the state machine will jump to the moving state.

Example 2

```
<transition condition='button1==1' goto='moving'>
    <condition type='and' condition='speed>4.00'/>
</transition>
```

There are two conditions being tested here. The first tests for user input ("button1==1"), the second test checks to see if the player speed is above 4 metres per second. So, if the player is providing some input AND is moving relatively fast, we transition to the moving state.

The tests always consist of a VARIABLE on the left hand side, a comparison operator in the middle, and a number/string value on the right hand side. See the reference section for which variables can be used. The comparsion operators can be one of <, <=, ==, ~=, =>, > and work as expected in any computer language.

Commands

These can exist as elements in the on_entry, on_update and on_exit portions of any state. They effectively allow you to program a state. The available commands are:

Sounds

Sound is only played on the local portable - these commands will do nothing when run on the remote portable.

play_sound

Will play a sound from a loaded audio bank.

Attr.	Req.	Def.	Range	Description
id	no	1	1 to 4	4 Sound handles are maintained, you can use any
name	yes			The sound effect name - this is NOT validated
volume	no	1	0 to 1	The sound volume to play at
pitch	no	0	-8 to 8	The audio pitch to play at

set_sound_register

Set a scream register for a currently playing sound effect (is safe to use even if no sound effect is playing). The effect is based solely on how the sound has been set up in the scream editor.

Attr.	Req.	Def.	Range	Description
id	no	1	1 to 4	Which sound to control
reg	no	0	0 to 3	Register to change
var	yes			An existing numeric variable to use as the source
factor	no	1	-100 to 100	Applied to the var

The register is set using the formula : reg = var x factor.

If you want to set the register to a constant, then you can use the special variable 'one'

stop_sound

Stops a playing sound effect (is safe to use even if no sound effect is playing)

Attr.	Req.	Def.	Range	Description
id	no	1	1 to 4	Which sound to stop
time	no	0	0 to 1	An optional time over which to fade the sound out

Particles

Particles are only played on the local portable. These commands have no effect on the remote portable.

start_emitter

Will start emitting particles from either the player root, a player bone or a model bone.

Attr.	Req.	Def.	Range	Description
id	no	1	1 to 10	The particle emitter id
effect	yes			The name of particle effect (an existing resource)
type	no			The emitter attach type ('render' or 'emitter')
tx, ty, tz	no	0		The relative offset from some origin
rx, ry, rz	no	0		The rotation to apply to the emitter
target	no	none	1 to 2	If given, will attach to a model (else the person)
bone	no			If given, will attach to the given bone of the target

stop_emitter

Stops a particle emitter

Attr.	Req.	Def.	Range	Description
id	no	1	1 to 10	The particle emitter id

Rig

These change rig properties. Works for both LOCAL and REMOTE portables.

set_rig_run_speed

Set the avatar run speed. The original value is restored when deactivating the portable.

Attr.	Req.	Def.	Range	Description
value	no	3.5	1 to 5.25	A run speed value (metres per second)

set_rig_walk_speed

Set the avatar walk speed. The original value is restored when deactivating the portable.

Attr.	Req.	Def.	Range	Description
value	no	2.2	1 to 3.5	A walk speed value (metres per second)

Model

These commands control any models that are created.

set_anim

Blends an animation in. There are 3 layers available.

Attr.	Req.	Def.	Range	Description
id	no	1	1 to 2	Which model to use
layer	no	1	1 to 3	Which animation layer to use
name	yes			The animation resource to use
blendin	no	0.25	0 to 2	Over what period should the animation blend in
weight	no	none	0 to 1	The initial weight of the animation (cancels blendin if specified)

attach

Attaches a model to the person, a bone on the person, another model, or the bone of some other model.

Attr.	Req.	Def.	Range	Description
id	no	1	1 to 2	Which model to use
target	no	none	1 to 2	Which model to attach to (uses the person if not given)
bone	no			Which bone on the target to attach to
tx, ty, tz	no	0		The translation from the attach point
rx, ry, rz	no	0		The rotation from the attach point

Animations

These commands control any animations that have been triggered. You can have three active animations for a local locomotion object, but remote objects will only run one. Each animation you run creates a layer; these layers can be applied to a single model or split between two models (this means that you can only animate one model on a remote locomotion object). The "set_anim" command creates an animation layer, and by default it will use layer 1 - remember to change this if you try and animate both models (otherwise you'll end up starting an animation on model 1 and then recycling that layer for model 2).

set_anim_rate

Change the rate of playback for an animation

Attr.	Req.	Def.	Range	Description
layer	no	1	1 to 3	Which animation layer to use (remote portables ignore layers 2 and 3)
var	yes			An existing numeric variable to use as the source
factor	no	1	-100 to 100	Applied to the var
min	no	-10	-10 to 10	Allows clamping of the generated rate
max	no	10	-10 to 10	Allows clamping of the generated rate

The playback rate is worked out as: var x factor, clamped between min and max

set_anim_weight

Change the weight of an animation layer

Attr.	Req.	Def.	Range	Description
layer	no	1	1 to 3	Which animation layer to use (remote portables ignore layers 2 and 3)
var	yes			An existing numeric variable to use as the source
factor	no	1	-100 to 100	Applied to the var

The weight is worked out as: var x factor. The value is always clamped to be between 0 and 1.

get_anim_time

Fetch the current time progress of an animation layer.

Attr.	Req.	Def.	Range	Description
layer	no	1	1 to 3	Which animation layer to use
var	yes			A user variable to hold the result in

set_anim_time

Set the time position for an animation.

Attr.	Req.	Def.	Range	Description
layer	no	1	1 to 3	Which animation layer to use (remote portables ignore layers 2 and 3)
var	yes			A variable to read the time from

stop_anim

Blends out an animation layer

Attr.	Req.	Def.	Range	Description
layer	no	1	1 to 3	Which animation layer to use
blendout	no	0.25	0 to 1	An existing numeric variable to use as the source

Variables

These commands are for dealing with user variables (these are created via the configuration.xml in a Variables section).

set_var

Set the variable to a constant value

Attr.	Req.	Def.	Range	Description
var	yes			An existing user variable to change
value	yes			The numeric constant to set the variable to

decrement_var

Decrease a user variable by a fixed amount per second.

Attr.	Req.	Def.	Range	Description
var	yes			An existing user variable to change
value	yes			The number to reduce the variable by (over a second)
limit	no			A bound for the subtraction (variable will not go lower)

The variable is reduced every frame by value x GetDeltaTime().

increment_var

Increase a user variable by a fixed amount per second

Attr.	Req.	Def.	Range	Description
var	yes			An existing user variable to change
value	yes			The number to increase the variable by (over a second)
limit	no			A bound for the addition (variable will not go higher)

The variable is increased every frame by value x GetDeltaTime().

copy_var

Copy some other variable value (and modify if desired).

Attr.	Req.	Def.	Range	Description
var	yes			An existing user variable to change
value	yes			An existing variable(user or read-only)
factor	no	1	-10K to 10K	multiply 'value' by this factor
min	no	-10K	-10K to 10K	a minimum clamp value for our variable
max	no	10K	-10K to 10K	a maximum clamp value for our variable

get_register

Copy a repertoire register to a user variable.

Attr.	Req.	Def.	Range	Description
var	yes			An existing user variable to read into
value	yes			The name of the repertoire register

The repertoire register must also be flagged to allow lua read/write from the object editor.

Variables

There are a number of read-only variables to which you can add several user ones. These variables can be used in some commands as well as transition conditions. User variables are added BEFORE the fsm definitions in the xml file, in a "Variables" block:

```
<variables>
    <var name='fuel' value='15'/>
</variables>
```

The above settings will create a user variable called "fuel", which starts with a value of 15. User variables can only hold numbers. The player can only add six user variables.

Read-Only Variables

- speed the player's current speed in metres per second
- one is always 1.0
- button1 input from the player (0 for not pressed, 1 on the frame the button is pressed, 2 for button being held)
- ydelta change in the player's y rotation
- mood - the player's current mood setting (what the player has set via the emotes menu)
- rig - the player gender, 0 for male, 1 for female
- rigchange - 0 normally, though temporarily more than 0 if the user has exited the wardrobe and changed gender

⚠ Although it is possible to create a Locomotion object with only a single gender repertoire (eg. just a male repertoire or just a female repertoire), it is not advisable. If you have a female repertoire only, the client has no way to gracefully let a user whose avatar is male know that the item won't affect them, or stop them from equipping it. The user with a male avatar would equips an item with only female repertoires in them, there is no warning that the item won't affect male avatars, and the item will be successfully equipped but appear to do nothing. This is a bad user experience.

Registers

Repertoires can have animation registers, these allow for a script to vary them to allow for animation blending for example. The

locomotion portable can set these registers using variables, this is done in the "registers" section in the config.xml (after the variables have been defined):

```
<registers>
    <register name='direction' value='ydelta'/>
</registers>
```

The above configuration will update the animation register "direction" every frame, setting its value to match the variable "ydelta". It's up to the repertoire to make sense of the actual values.

Limits

Limits have been placed on the number of various elements and the first fsm has different limits to all the others (because the first fsm runs on the remote portable):

- fsm - Limited to five 'fsm' elements
    - the first fsm
        - states - Limited to 20 elements
            - tests - Limit of 3 tests per state
                - conditions - Limit of 3 conditions per test
            - commands - Limit of 4 commands per OnEntry, OnUpdate and OnExit
    - all further fsms
        - states - Limited to 20 elements
            - tests - Limit of 6 tests per state
                - conditions - Limit of 4 conditions per test
            - commands - Limit of 8 commands per OnEntry, OnUpdate and OnExit

Error Checks

The script attempts to catch as many errors during the parse phase as possible.

Parse Properties

- run_speed must be greater than walk_speed

Parse model(local/remote)

- model 1 must be defined before model 2

Parse Fsm

- Limit fsm count
- fsm must contain an "initialise" state
- the fsm must not contain a transition which references a non-existent state

Parse State

- Limit state count
- State doesn't contain a transition to itself
- on_entry, on_exit, on_update don't exceed their limits
- No duplicate states exist in the current fsm

Parse Transition

- Limit transition count
- Limit the condition count
- The condition will be checked too:
    - the condition must be of the form "x=y", where:
        - x is a known variable (either a user variable or read-only variable)
        - '=' is ==,~=,>,>=,< or <=
        - z is number (special cast, can be a string if x=='mood')
    - if x=='mood' then 'y' must be from the valid list (Neutral, Happy, Sad, Angry, Confused)
    - the condition attribute in the condition element can only be 'and' or 'or'.

Resource Packs

A Resource Pack is a container object for other scripted objects. The scripted objects can load and use any resource attached to the resource pack. This component either identifies an object that is a resource pack, or it identifies an object that uses resource packs.

You can use a Resource Pack to:

- Create Downloadable Content (DLC).

- Split the resources for a single object (most likely a game) across several resource packs for reuse and manageability purposes.

Use Cases

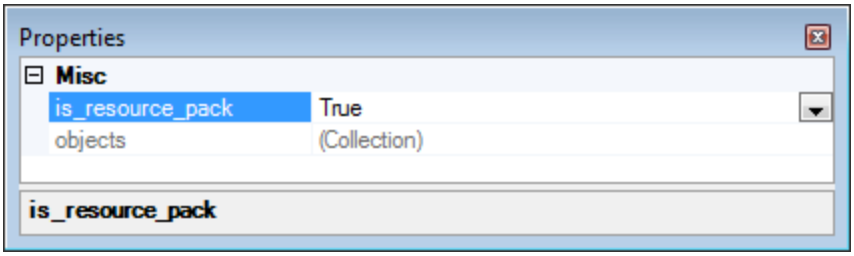
- DLC: Users receive a standard chess game as a reward when they enter your space. You sell resource packs that change the theme of the chess set, for example, a dragon theme where all the pieces are various dragons. Once the user buys the Resource Pack they can choose to change the theme of the chess set.
- DLC: A car racing mini-game has allows users to play the first two tracks for free. To gain access to new tracks and new cars, they need to buy a Resource Pack.
- Resource Management: You create a dungeon Realtime game. The basic structure of the game (i.e. battle mechanics, character stats, inventory management) are scripted into the main object. Each dungeon level is packaged into a Resource Pack instead of with the Realtime game object. When the players advance to the next level, the Realtime game unloads the previous level's Resource Pack and then loads the next level's Resource Pack.

Elements of a Resource Pack

A Resource Pack is useless on its own. It must be used by a master object. To successfully make and use a Resource Pack, you need the following elements:

Element	Description
Resource Pack object	This object contains the Resource Pack component (with is_resource_pack set to True), the header component, and any resources that you want the master object to use.
Master object	This is a scripted object (a mini-game or realtime game) that uses the Resource Pack. It contains the Resource Pack component (with is_resource_pack set to False), as well as all of the other components and resources necessary to make the mini-game or realtime game.
Master object script	The master object's Lua script(s) must use the ResourcePack Lua API to load, manage and unload Resource Packs.

Resource Pack Properties



Property	Description
is_resource_pack	A flag indicating if the object is a Resource Pack or uses Resource Packs. If the object is a Resource Pack, set to True. If the object uses Resource Packs, set to False.
object	If the object is a Resource Pack, enter the object ID for each object that uses this Resource Pack. If the object uses Resource Pack, enter the object ID for each resource pack that it uses.

Resource Pack Functions

The following functions were created/updated for Resource Packs:

- The entire ResourcePack Library
- Entity.BlendAnimIn
- Entity.ParticleSetEffect
- Entity.PlayAnim
- Entity.SetCollision
- Entity.SetModel
- Entity.SetSkeleton
- Resource.Find
- Resource.Run
- Texture.Find

For more information on each function, see the Lua API Reference.

Creating a Resource Pack

You can create a Resource Pack object by using the New Object Wizard, or by creating a new object and adding the Resource Pack component.

Creating a Resource Pack Object

To create a Resource Pack object using the New Object Wizard:

1. In the Object Editor, select File > New Object Wizard > Resource Pack. This creates the object with the necessary components.
2. Add resources to the Resource Pack.
3. Select the Resource Pack component in the Object View panel.
4. In the Properties panel, set is_resource_pack to True.
5. In the Resource Pack component, enter the Object ID for the master object that you want to use this Resource Pack. To do this:

    a. Select objects under the Resource Pack component in the Object View panel:

    Unable to render embedded object: File (Resource_Pack_Object_Selection.png) not found.

    b. Click the Add a new object button:

    Unable to render embedded object: File (ResourcePack_object002.png) not found.

    c. Add an ID to the new object:

    Unable to render embedded object: File (Resource_Pack_Object_Added.png) not found.

6. Finish making the object as normal (thumbnails, age ratings, etc.), and then package and submit it to the Content Delivery System (CDS).

> ⚠ Make sure that the Resource Pack object contains no components other than the Resource Pack and Header.

Creating a Master Object

1. Create an object as normal.
2. Add the Resource Pack component and select it in the Object View panel. In the Properties panel, set the is_resource_pack property to False.
3. In the Resource Pack component, enter the Object ID for each Resource Pack object that the master object will use.
4. In the master object's script, use the ResourcePack Lua library to load and and manage Resource Pack and their resources.
5. Finish making the object as normal (thumbnails, age ratings, etc.), and then package and submit it to the CDS.

Adding a Resource Pack to an Existing Master Object

If a master object has already been published to the live environment you can add Resource Pack to it by following steps 2-5 above.

Managing Resource Packs in Script

Loading a Resource Pack

1. Call ResourcePack.Load( ) to load the Resource Pack.
2. Call ResourcePack.GetLoadProgress( ) and ResourcePack.GetLoadState( ) to monitor the loading and make sure that the Resource Pack loads before you begin using its resources.

    If GetLoadState returns RPState.Error, call ResourcePack.DebugGetLoadError( ) to see what the error is and to respond to it. Resource Pack load errors should only occur during development.

Using the Resource Pack resources

To use the resource pack resources:

1. Call Resource.Find(string resourceName, ResourcePack pack) to load a resource from the Resource Pack.
2. Call Resource.Run(string resourceName, ResourcePack pack) to run the resource.

    To load a texture, call Texture.Find(string name, ResourcePack pack) to find a texture in a Resource Pack.

Unloading a Resource Pack

To unload a Resource Pack, call `ResourcePack.Unload( )`. Make sure all resources are unloaded first.

As with all resources including those contained in the master object directly, the client will not handle removing references to resources used by the system when the object is unloaded. It is important that you ensure that no resources from the Resource Pack are in use by the game before calling `ResourcePack.Unload()`, as the client attempts to use the resource after it has been unloaded. This means clearing models, animations, textures, particles and so on that are currently in use before unloading the Resource Pack that contains them.

Similarly to unload the Resource Pack, cancel any deferred loads of resources contained within a Resource Pack, otherwise the deferred loader will continue trying to load the resource. If a Resource Pack is garbage collected then it will automatically call `ResourcePack.Unload()`, so do not allow a Resource Pack to go out of scope if you wish to continue using its resources.

Resource Pack Memory Management

Resource Packs consume the memory of the object using them. For example:

- If a mini-game embedded in a Public Space uses resources from a Resource Pack, then those loaded resources consume and are bound by the scene's memory budget.
- If an active item uses resources from a Resource Pack, then the resources consume and are bound by the active item's memory budget.

When a script calls a Resource Pack there is also some memory consumption from the object itself (a normal object load).

> PS Home uses a portion of the PlayStation®3 HDD for its cache. Objects that dynamically load many resources from a Resource Pack can quickly consume the cache.

Consuming Memory and Loading Resources

When loading a Resource Pack you can choose to load each resource in one of two ways:

- Automatically: The resource loads with the Resource Pack.
- Deferred Loading: You can set the resource to deferred loading. By choosing deferred loading, the resource only consumes memory when specifically loaded. Deferred Loading works the same way in Resource Packs as it does with any other scripted object. Make sure that you load the Resource Pack before trying to load the resource and that you unload all resources before unloaded the Resource Pack.

Resource Pack Validation Guidelines

Use the following guidelines to ensure your Resource Pack object meets the validation requirements:

- When you submit a Resource Pack to the Content Delivery System (CDS) for publishing in the live environment, you must also repackage its master object and resubmit it to the CDS.
- When you subject a master object to the CDS for publishing in the live environment, you must also repackage all its Resource Packs and resubmit them to the CDS.
- A Resource Pack can be used by only one master object. A master object can use as many Resource Packs as you want. Bear in mind testing and CDS implications.
- Each resource that a master object uses from a Resource Pack must meet the validations, limits and TRCs of that master object.

Embed Objects into Scenes

Embedding a scene object means taking an object from the Object Editor and embedding it in a scene by adding it to the scene and selecting "scene object" as a component for that object. The object becomes part of the scene and is stored in the scene's memory. It has its own Object ID, components and resources.

You can embed an object in a scene in the following ways:

- Add a Scene Object component to the object. This component also allows you to define instance parameters, which can then be set in the Scene Editor.
- Add a Furniture component to the object. This is the only way to add a furniture object to a public space.

> You can embed objects with a Furniture component in public spaces only, however you cannot embed Character components.

After you create an object, you can embed it within a scene (see Adding Objects to a Scene) by dragging the object from the Objects section of the Palette panel in the Scene Editor. You set the object you want to embed through the Object ID property of

the created object.

Examples of Embedded Scene Objects

The following examples use embedded objects:

- Mini-games: In the Object Editor, the object has a MiniGame component, a Lua Environment component, and a Scene Object component. It is placed in a scene several times with several instance parameters set (see Instance Parameters).
- Realtime Games: In the Object Editor, the object has a RealTime Game component, a Lua Environment component, and a Scene Object component. It is placed in a scene several times with several instance parameters set that adjust the difficulty of each instance.
- Furniture: In the Object Editor, the object has only a Furniture component. It is placed as an Object node in the scene because this is the only way to add Furniture objects to a public space.
- Scripted Animation: In the Object Editor, this object has a Lua Environment component and a Scene Object component. When placed in a public space through an Object node, you can choose instance parameters that define which animations are played and in what order they are called through the object's Lua script.

An embedded object retains its full functionality, so if you place a chair it still acts as a chair, and if you place a mini-game it still acts as a mini-game.

Why use Embedded Scene Objects?

We recommend that you use embedded objects rather than scene scripts for the following reasons:

- You can reuse the object in other scenes.
- You can easily update content by resubmitting only the embedded object to the Content Delivery System (CDS). Note that whenever an embedded object is updated, all scenes that use that embedded object must go through Quality Assurance (QA) again.

For objects with their own Lua script resource, embedded objects also enable you to set instance parameters. By setting an instance parameter that your script accesses, you can have different behavior between different instances of an object in a scene.

Restrictions

- You can embed objects which have either the Scene Object component, or Furniture component.
- You cannot use Character components as embedded objects, not even those with the Scene Object component.
- Embedded furniture objects can be used only in public spaces.
- There is no defined limit to the number of instance parameters you can add to your object. However, instance parameters consume memory, so you are constrained by object memory limits.
- To set instance parameter values in the Scene Editor, you must have the object with the Scene Object component on your machine.
- If you move a scene to another machine or a directory that has furniture objects embedded in it, you can still package the scene without the objects on your machine. However, in this case you must supply the Furniture objects to the CDS when submitting the scene for publishing to the live client.

Embedding Actives

If an object with an Active Item component is embedded in a scene, it loses:

- Safe Volume: The object no longer has a safe volume.
- Save Data: The object loses access to Save Data.
- Targeting: The object loses all targeting associated with it (as do all objects that are embedded).

> We do not recommend embedding objects with an Active Item component because the active loses the features and functionality that define it as active.

Tips

- You can specify that objects download with a scene when a user downloads the scene, rather than downloading by default when the user enters the scene. See Downloading Objects with Scenes.
- Design an active item's script to handle returns. Certain returns do not always function the same way for embedded objects. For example, `System.VideoSystemLock()` does not lock the video system if a scene script or embedded object is using it, and thus will never return `True`. Active items that play video screens can play video only when this function returns `True`, so an embedded active item with a video screen will never play video.

See also:

- Defining Instance Parameters
- Adding Objects to a Scene
- Working With Objects in a Scene

Creating a Scene Object

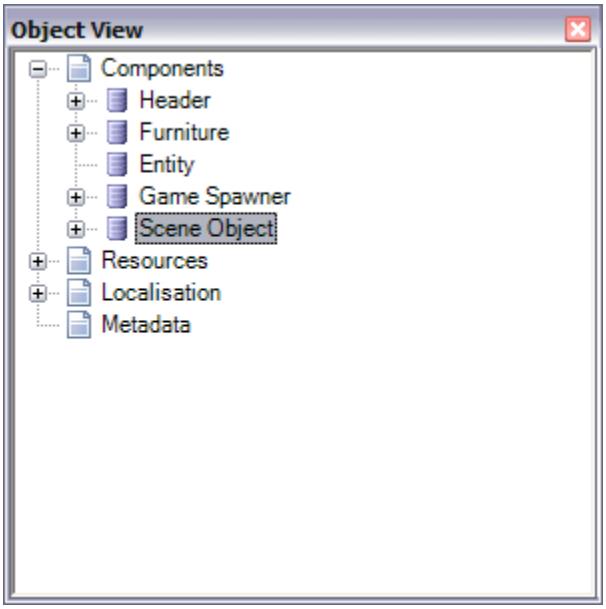
Using the Object Editor

When you submit a scene with Scene objects to be published in the live client, the Scene objects must also pass QA. This means that the objects must fulfill all the requirements for objects (such as thumbnails, localization, age rating information, memory limits).

To create a Scene object:

1. In the Object Editor, navigate to the required object.
2. Select Object > Add New Component. The Add Component dialog opens.
3. Select Scene Object and click Add.

   The Scene Object component is added to the Components list, as shown in the following example.



4. When you select the Scene Object component the Properties panel displays its additional properties. See Scene Object Component.

   You can add a number of additional properties to each embedded object, to define the instance parameters available in the Scene Editor.

Furniture Objects

When you export a piece of furniture from Maya, you can immediately embed it in a scene without making any updates or changes through the Object Editor.

Using Lua Script

This section applies only to objects with a Scene Object component.

In terms of instance parameters, the script queries the value of a parameter, and then performs an action depending on the value returned. In your script, you must:

- Define each of your instance parameters.
- Specify an action for each value of each parameter. For example, if it is a boolean parameter, you must have one action for `True` and another action for `False`.

The following Lua API function is crucial to instance parameters: `Object.GetInstanceParameter()`. This function queries a parameter value.

In the Object Instance Example, the script queries the `enum_parameter` named `'enumparam'`. `enumparam` has four possible values: e, n, u, m, which you can select in the Scene Editor. It loads a different .mdl file depending on what value is returned (e, n, u or m):

```
enumParam = object:GetInstanceParameter( "enumparam" )

enumModelList = { e = "E",


                n = "N",

                u = "U",

                m = "M",

                }

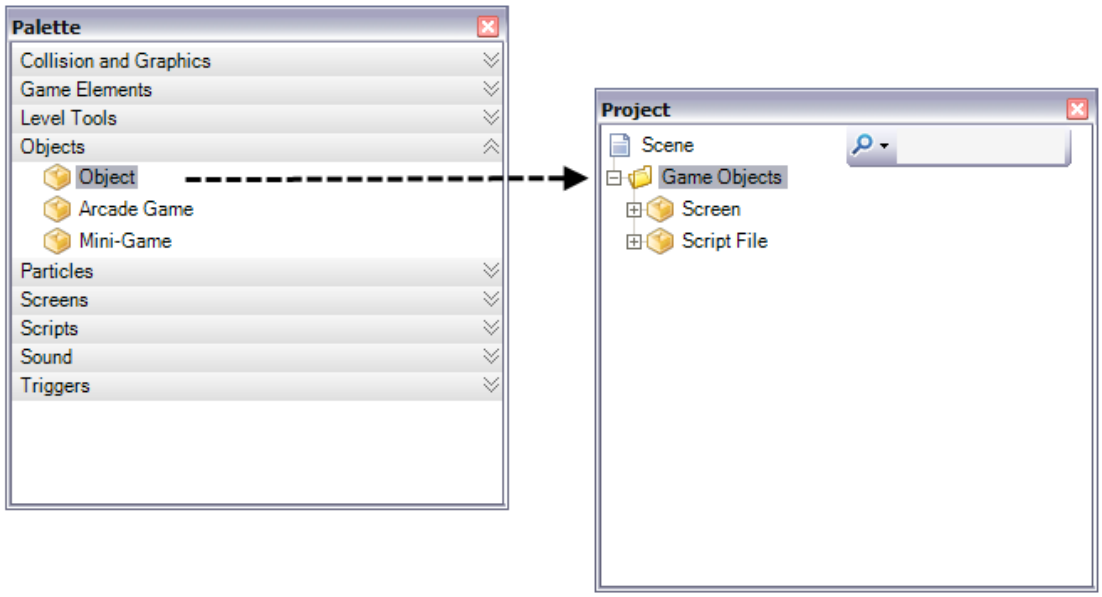

enumEnt = Entity.Create()

enumEnt: SetModel( enumModelList\[enumParam\] )
```

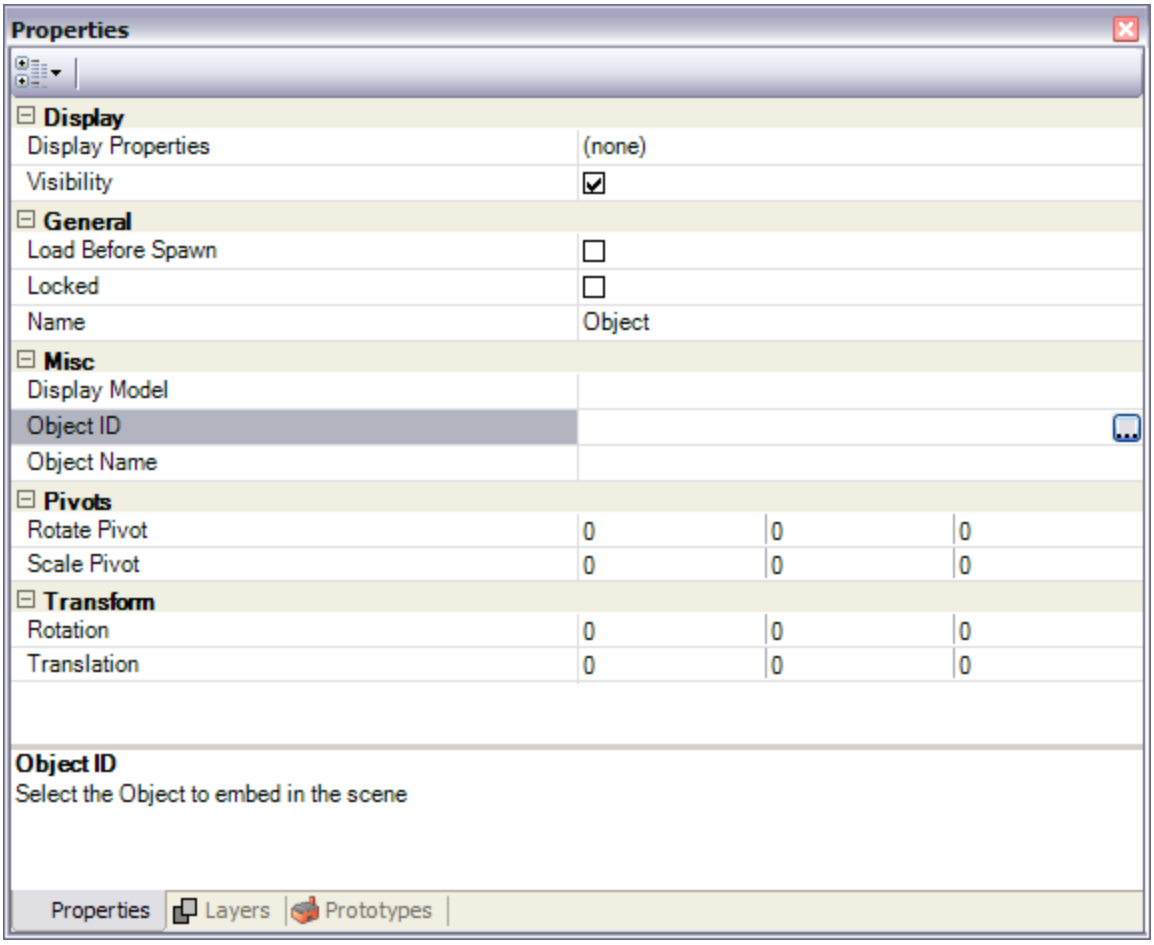
Using the Scene Editor

To add an object to a scene:

1. In Scene Editor, open the .scproj file for your public space.
2. In the Palette panel, drag the required object from the Objects section to either the Game Objects folder in the Project panel, or directly into your scene:



3. With the Object selected in the Project panel or in the Design View, go to the Properties panel, click  next to Object ID and select the object you require:



The Properties panel updates with the object's Display Model, Object ID, and Object Name properties. You set these properties when you edited the object in the Object Editor. If your object has a Scene Object component, its properties are also displayed.

4. You can choose the following instance parameters in the Scene Editor:

    a. Deselect the booleanparam option to set the value to False. Select the booleanparam option to set the value to True.
    b. Select the value you require from the enumparam drop-down list.
    c. In the Object Editor, you set the default value as well as the range of possible values. Enter a number that is within the range that you defined in the Object Editor and in your script in the numericparam field.
    d. Enter one of the values that you defined in your script for the string parameter in the stringparam field.

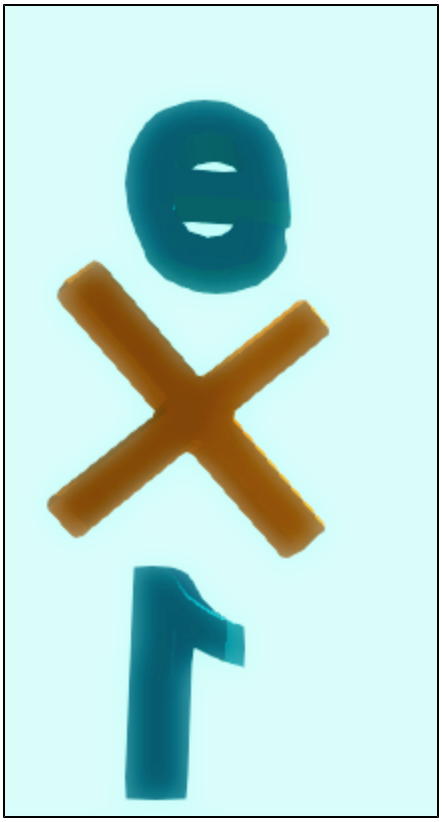
5. Repeat this step for all the scene objects that you want to use.
6. To see how the object instance parameters work in a scene, download the Object Instance Example and use it with at least two Object nodes.

Defining Instance Parameters

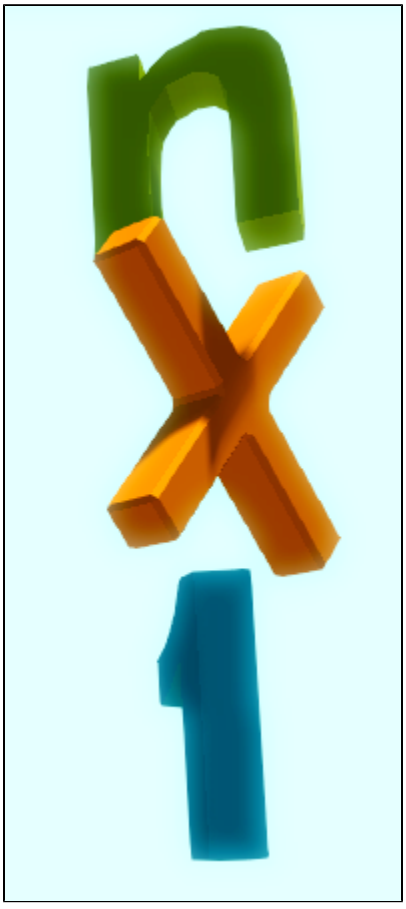
Instance Parameters are starting values or properties that you can choose for your object. The starting values or properties correspond to values and properties that you define within the object's Lua script. They allow you to have several instances of the same object in a scene, each with unique starting values or properties. For example, you can have several instances of the same object with an instance parameter named enumparam. Each instance could have possible values of e, n, u, m, with a default value of e. A different model file loads, depending on what value you choose for enumparam when you place this object as an embedded object in the Scene Editor.

> Instance parameters are only available in Public Spaces.

For example, you can set one instance of the object to e, loading the e.mdl:

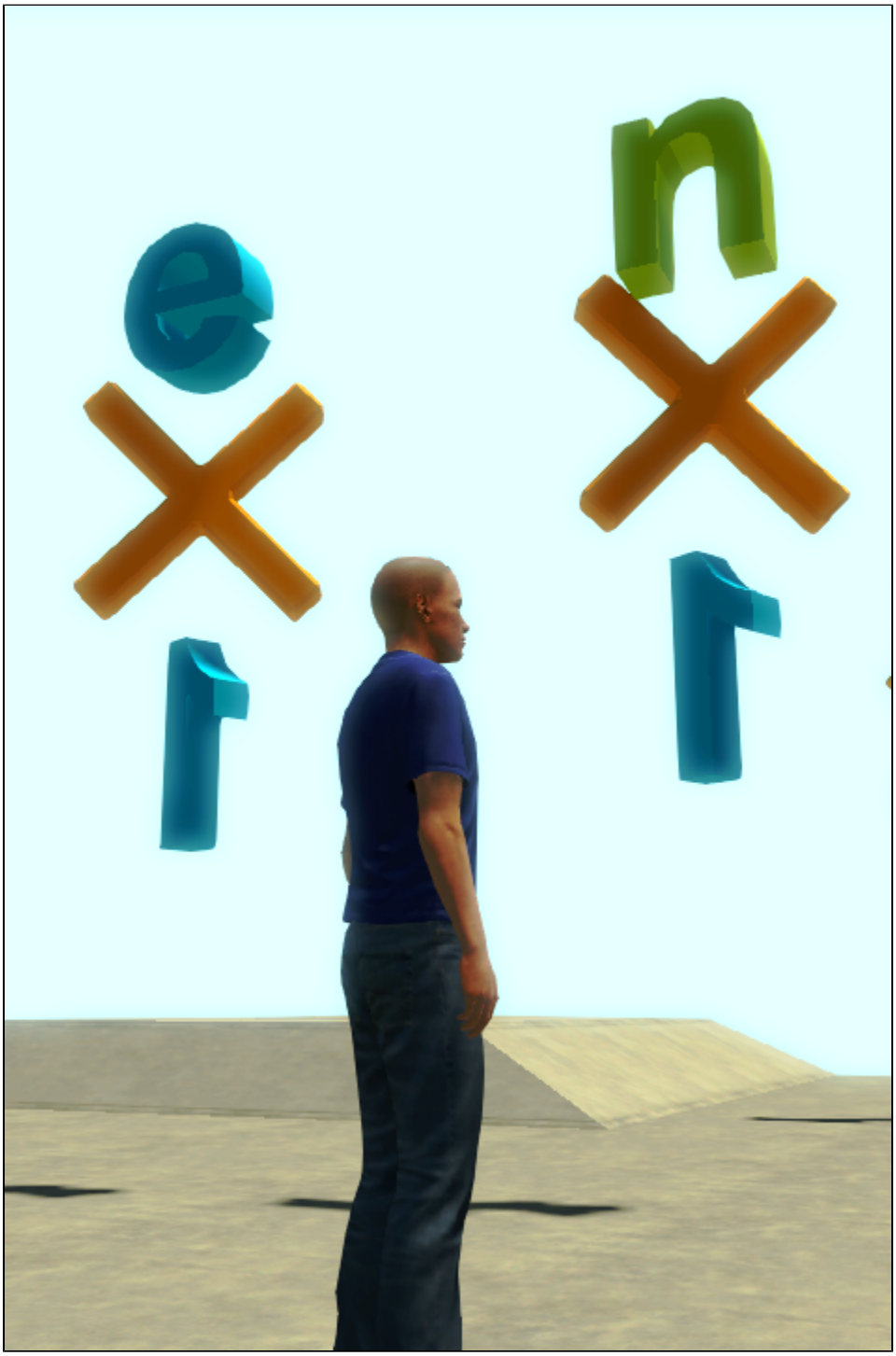


You can set another instance of the same object in the same scene to n, loading the n.mdl file.



Both instances of the same object are in the same space simultaneously.

The following object is set to e on the left and n on the right:



Instance parameters are useful only for objects with a Lua Environment component.

If you do not use a Scene Object component, the only way to achieve multiple instances of an object is to create a new object that is nearly identical, then change the parameters through the script and Object Editor, before placing the new, nearly identical object in the scene.

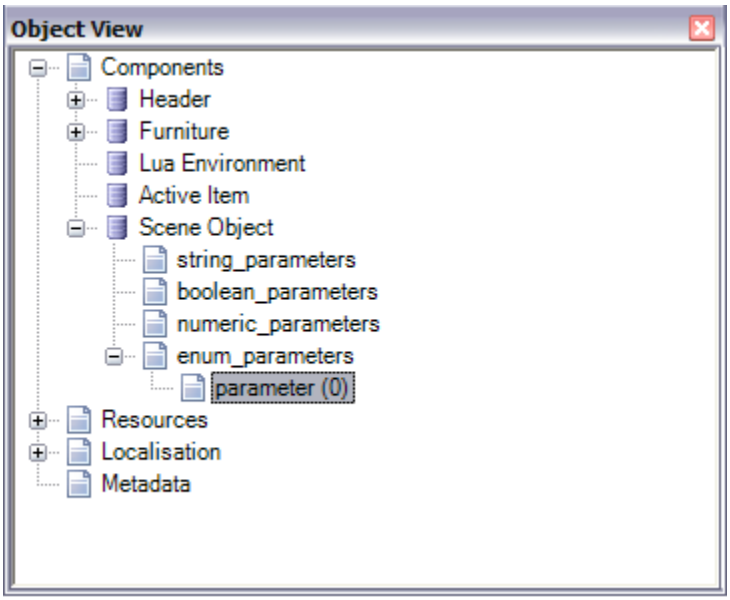
In Lua, you can use a workaround where you name a Mini-game Trigger radius and reference the name in the object's Lua script. This is not best practice because the Scene Editor has no validations to check against name mismatches. With Object nodes there are validations to ensure that the parameter values and names given in the Scene Object component match those set in the Scene Editor.

You can download a sample object called Object Instance that demonstrates what you can do with instance parameters and the Scene Object component from https://home.scedev.net/projects/samples.

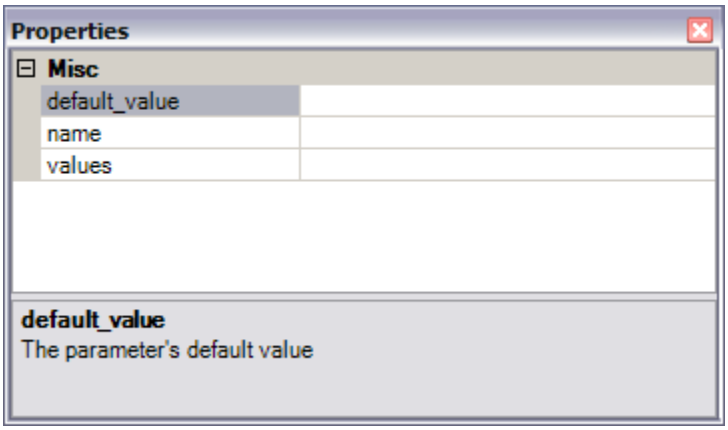
Adding an Instance Parameter

To add an instance parameter to a Scene Object:

1. In the Object View panel, select the parameter type for the Scene Object, for example, enum_parameter, and then click Add a new parameter in the toolbar.
   A new node is displayed under the selected parameter.



2. You can set the values and name of this new parameter in the Properties panel:



3. Enter the default value of the instance parameter in the default_value field.
4. Enter the name of the instance parameter that you call in the script in the name field. The name must be in lowercase.
5. Enter the other values that the enum_parameter can be set to in the values field (enum_parameter only).
6. Enter the maximum value to which the parameter can be set in your script in the max_value field (numeric_parameter only).
7. Enter the minimum value to which the parameter can be set in your script in the min_value field (numeric_parameter only).

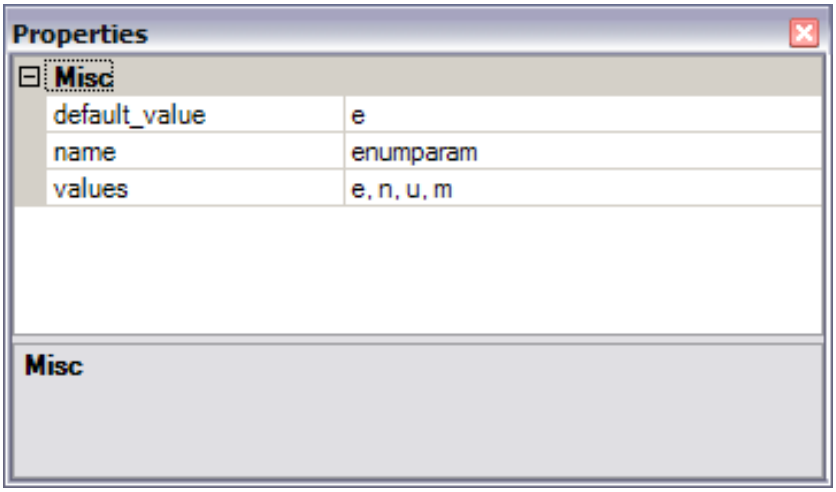
Objects with Instance Parameters

Setting Parameters

Enum_Parameter is the only parameter for which you need to define all possible values in the Object Editor. For all other parameters, you enter only the name of the parameter and its default value in the Object Editor.

To set an instance parameter:

1. Add the instance parameter.
2. In the Properties panel, enter the name of the parameter (using the name of the parameter from your script) and enter each of its values:



3. Repeat this step for as many parameters as you have defined in your script, then save your object.

> There is no limit to the number of parameters you can have on your object, only memory limitations.

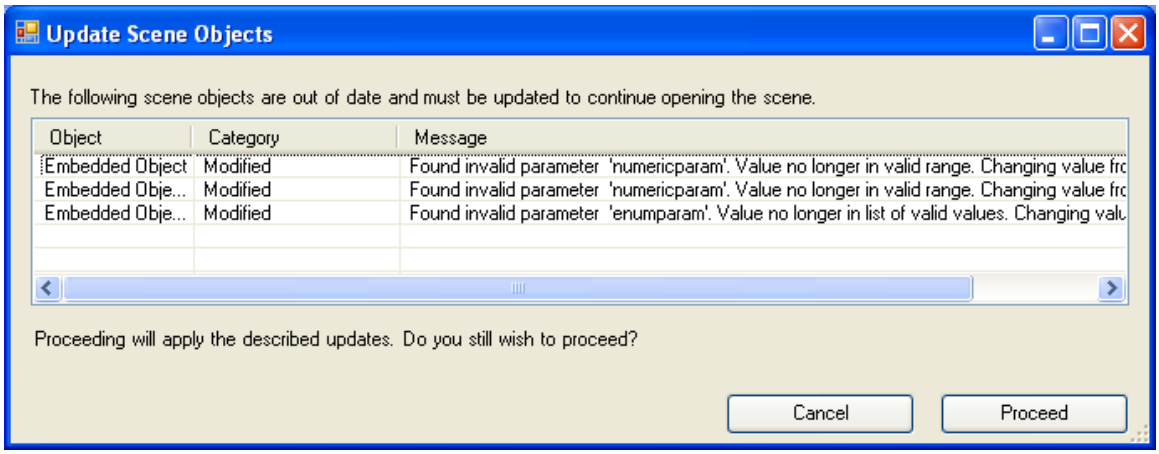
> Before you add an object in the Scene Editor, make sure that the object is successfully packaged and passes validation. All objects must be successfully packaged before submitting to the Content Delivery System (CDS) for publishing. For more information on object requirements and limits, see HDK Tools Validations.

Modified and Invalid Instance Parameters

Assume you do the following:

1. In the Object Editor, create an object with a Scene Object component with instance parameters.
2. Add the object to a Public Scene in the Scene Editor and set the instance parameters for that object.
3. Return to the Object Editor, and change the instance parameters.

After re-opening the scene, you might receive the following warning message:



You receive this warning when you remove instance parameter values in the Scene Object component (Object Editor) that were used in the Object node in your scene.
You also receive this warning if you delete an instance parameter altogether.

For example, with the Instance Object Example:

1. Two Object nodes in a scene reference the same object. This object has one enum parameter. One of the Object nodes sets this parameter to e and the other sets it to n.

2. The object is re-opened in the Object Editor and the enum parameter's list of valid values is changed from e, n, u, m to u, m.
3. When the scene is re-opened in the Scene Editor you receive the warning messages saying that the instance parameter values have been modified.

Solutions

The warning message indicates that the instance parameter values you chose in the Scene Editor no longer correspond with the values listed in the Object Editor. You can resolve the issues in several ways:

- Click Proceed in the Update Scene Objects dialog. This action resets the invalid parameter value to the default value. If the instance parameter has been completely deleted, the parameter and its value is removed from the Object node.
- If you do not want to use the default values, update the Object node manually.

Furniture and Decorations

Picture Frames and Wall Decorations

Picture Frames are a type of furniture item placed on the walls of Personal Spaces (wherever picture hooks have been placed). They can display JPG and PNG images taken from the PS3 hard drive, or from a library of stock images.



You can have any kind of wall decoration – not just picture frames.

For example: a moose head, lacrosse sticks, clocks.

The wall decorations are still created and exported as picture frames, but just do not have the components to display and swap images.

There are two types of picture frames:

- Static Picture Frames: These are wall decorations that do not have the ability to display a picture from the user's PlayStation®3 HDD.
- Dynamic Picture Frames: These are picture frames that allow users to display pictures from their PlayStation®3 HDD.

Picture Frames work like any other piece of furniture and therefore they can be acquired:

- As part of a core update
- From the PlayStation®Store (for free or paid for)
- As a redeemed Home Reward



Picture Frames are only available in America (SCEA), Japan (SCEJ) and Asia (SCE Asia). American users (SCEA), Japanese users (SCEJ), and Asian users (SCE Asia) can view each other's images that are placed within their Picture Frames. European users (SCEE) are not able to see any of them. European users (SCEE) are able to see picture frames that do not allow users to upload pictures.

Be aware of the following issues regarding the use of Picture Frames:



The system software generates thumbnails automatically whenever the user manipulates an image file on the XMB™. However, thumbnails are not generated while a game is running. This means that when the user acquires new images on their PlayStation®3 HDD while using PS Home (for example by capturing screenshots within PS Home), thumbnails will not be created for those images. Consequently, the user will not see the thumbnail preview for those images and will have difficulty putting them in Picture Frames.



The network can get overloaded or the speed may drop significantly if the network connection speed is slow, the image file size is large, or if there are a number of viewers looking at the picture in the owner's Personal Space. Keep in mind images are sent in an uncompressed state from the owner's PlayStation®3 HDD to each one of the viewers.



Users will have difficulty targeting Picture Frames if the Picture Frame is placed a great distance off the floor.

Picture Frames

You access picture frames and wall decorations from the Picture Frames category of the furniture placement system.

Format, Size and Flagging of Content

Dynamic Picture frames can display JPG and PNG image formats.

> ⚠ Progressive JPGs and interlaced PNGs require much more memory, and usually result in an error message. Use only small progressive JPGs and interlaced PNGs, or avoid these file types altogether.



> ℹ Every user is responsible for the images they choose. Other users cannot change other people's images, but other users can flag them.

Placing Picture Frames on Hooks

Users place picture frames on picture hooks. Users cannot add, delete or move picture hooks, so if you want to support picture frames in a space, you must add picture hooks to the scene, using the Scene Editor. You can add any number of picture hooks in a space, but you should have at least 3 hooks.

If a user selects a picture frame from the furniture menu, the picture frame is automatically attached to the first available picture hook. The user can then move the picture frame to a different picture hook. If all picture hooks are occupied the user is prompted to remove an existing Picture Frame before being able to place another. For more information, see Adding Picture Hooks.

You remove Picture Frames from a Personal Space like any other piece of furniture.

Interacting with Picture Frames

The interaction with Picture Frames is like the interaction with a piece of furniture.

Menu Options

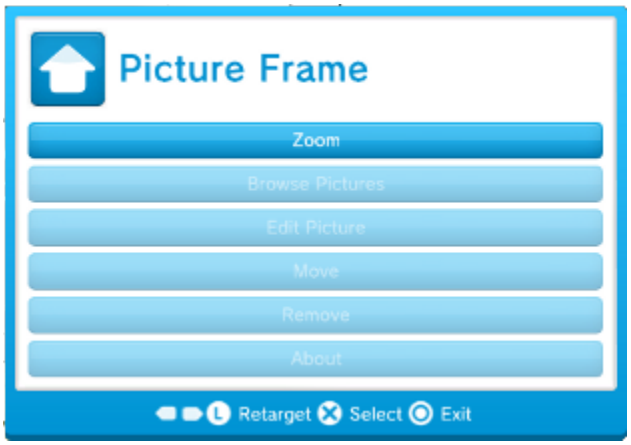
The different options are shown in the Action menu. However there are slight differences depending on the type of user.

Standard Features /Viewer Features



- Zoom: Takes the user into zoomed viewing mode in exactly the same way as for a normal poster or video screen.
- About: Displays the Picture Frame's name and its thumbnail.

Owner Features



- Browse Pictures: Browses for the images available and changes the picture in the frame.
- Edit Picture: Edits the options of the picture in the frame.
- Move: Opens the move furniture dialogue for the frame.
- Remove: Removes the frame from the wall.
- About: Displays the Picture Frame's name and thumbnail.

Dynamic Picture Frames

There are two types of picture sources:

- Standard pictures provided by PS Home:
    - Stock of default images included with the frame.
    - Packs of pictures provided free of charge (such as freely available masterpieces, images from new artists, advertising).
    - Purchased images (such as game posters, custom artwork).
- Custom pictures provided by the user:
    - From the PlayStation®3 hard drive.

Editing Pictures

After the user has selected the image for the picture frame, there are three basic options for how the picture is sized into it:

- Fit proportionally (default): This keeps the image in its original aspect ratio and fit it into the frame, placing white bars above and below or to the sides as needed
- Fill proportionally: This keeps the image in its original aspect ratio and fit it into the frame, cropping above and below or to the sides as needed
- Fill: This stretches or squashes the image as needed to fill the chosen frame

Viewing and Visibility

When they are first placed, Picture frames display the default texture. The owner of the space can choose picture frames and load images in them. The owner can change images in pictures frames at any time. They select them from their PlayStation®3 HDD, or from stock photos in the Default Images folder, if the PS Home for user's region provides them.

When an image is loading into a picture frame, the owner and everyone else in the space who can see the image sees this loading image:



When the owner removes an existing image, the frame reverts to the default image.

If an image fails to load, the frame displays the following image with an indication of the possible error:



A user viewing the picture frame might not see the loaded image because of age restrictions or parental control levels. If a user is not permitted to view a certain image, they see the default image instead of the intended image.

> Users outside of SCEA and SCEJ regions see only the original texture only.

Creating Picture Frames in Maya

To create a simple picture frame in Maya:

1. Select File > New Home Furniture Item.
2. In the New Home Furniture Item dialog, enter a name for the new furniture object and click OK:

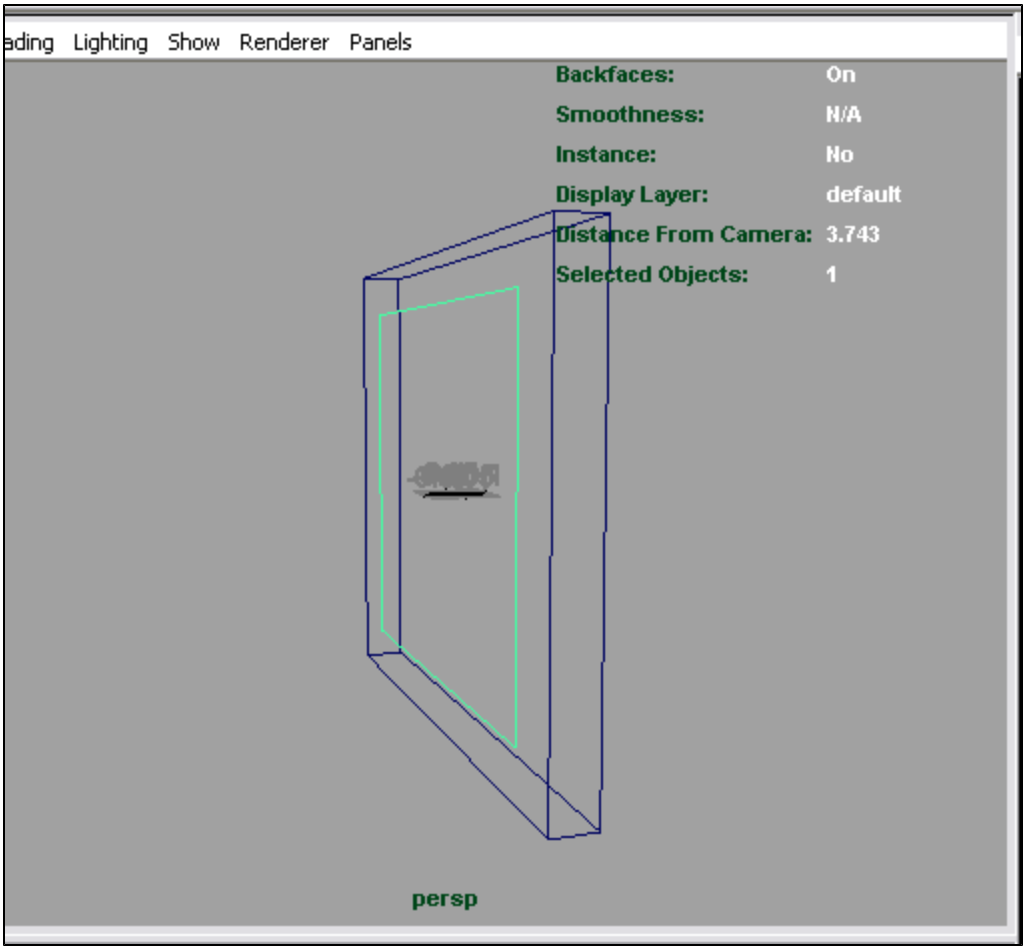


3. To create a simple cube-shaped picture frame, select Create > Polygon Primitives Cube.

> You can make your picture frame in whatever shape you want, as long as it does not exceed the dimension limits. See HDK Tools Validations.

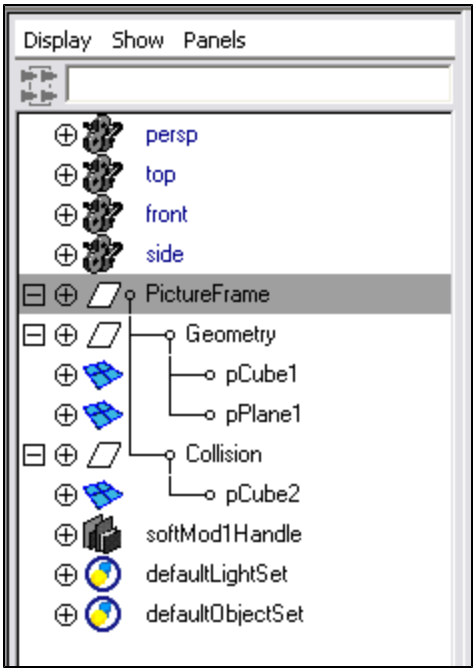
> Picture Frames are placed on picture hooks in users' personal apartments. Any part of the picture frame that is not in the positive Z-axis may not be visible since it may be rendered behind the picture hook and thus behind the wall.

Dynamic picture frames only: If you are making a dynamic picture frame, select Create > Polygon Primitives > Plane to add a plane. This is the geometry on which the user's picture will go. This sample aligns the plane with the front of the picture frame, which is on the positive side of the depth-axis.



⚠ This must be in the positive Z-axis; otherwise the 'screen' may not appear in the live environment.

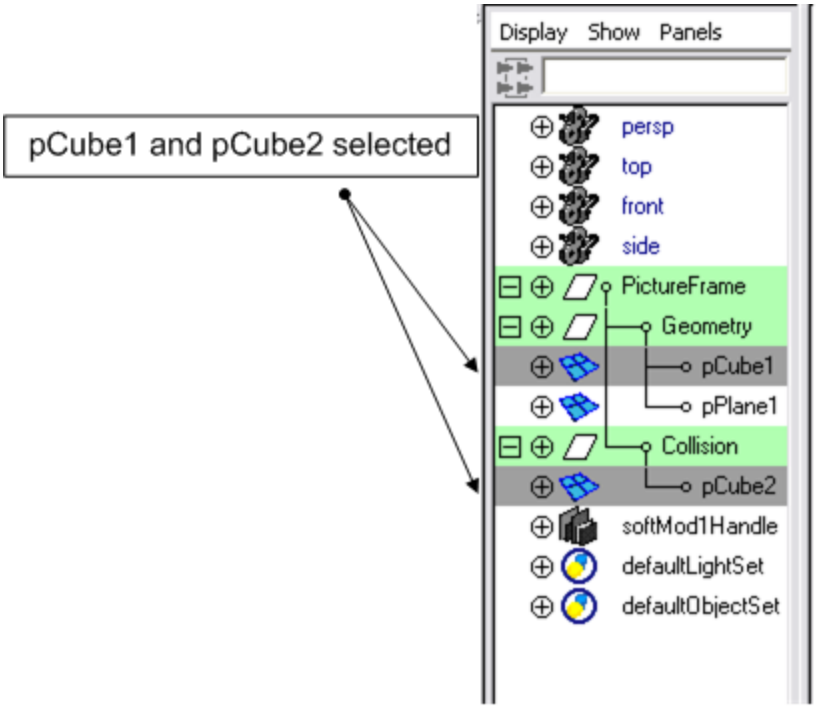
4. Specify the picture frame dimensions. The required maximum dimensions for picture frames is 2 m (X axis) x 2 m (Y axis) x 0.3 m (Z axis - for example, protrusion).
5. You can now set up geometry and collision. Display the Persp/Outliner View by clicking  .
6. Duplicate the picture frame shape and drag one to Geometry and the other one to Collision.
7. Drag the plane shape to Geometry:



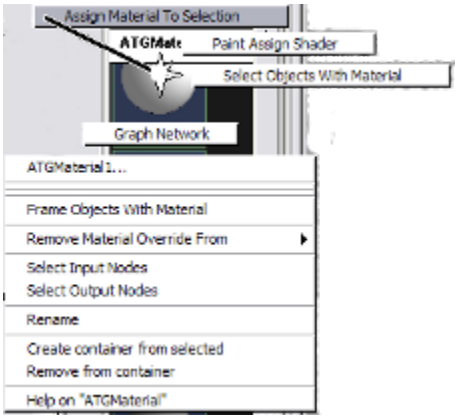
8. You can now apply ATG Shaders. Display the Hypershade/Persp View by clicking  .
9. Select the ATG material to be applied to the picture frame and click to open the Attribute Editor.
10. In the ATG Material tab, select the material type required from the Select drop-down list (in this example, the default material is selected):



11. Display the Persp/Outliner View by clicking  and select the picture frame (in this example, pCube1 and pCube2):



12. Display the Hypershade/Persp View by clicking  , right-click over the desired ATG material and select Assign Material to Selection from the pop-up menu displayed:



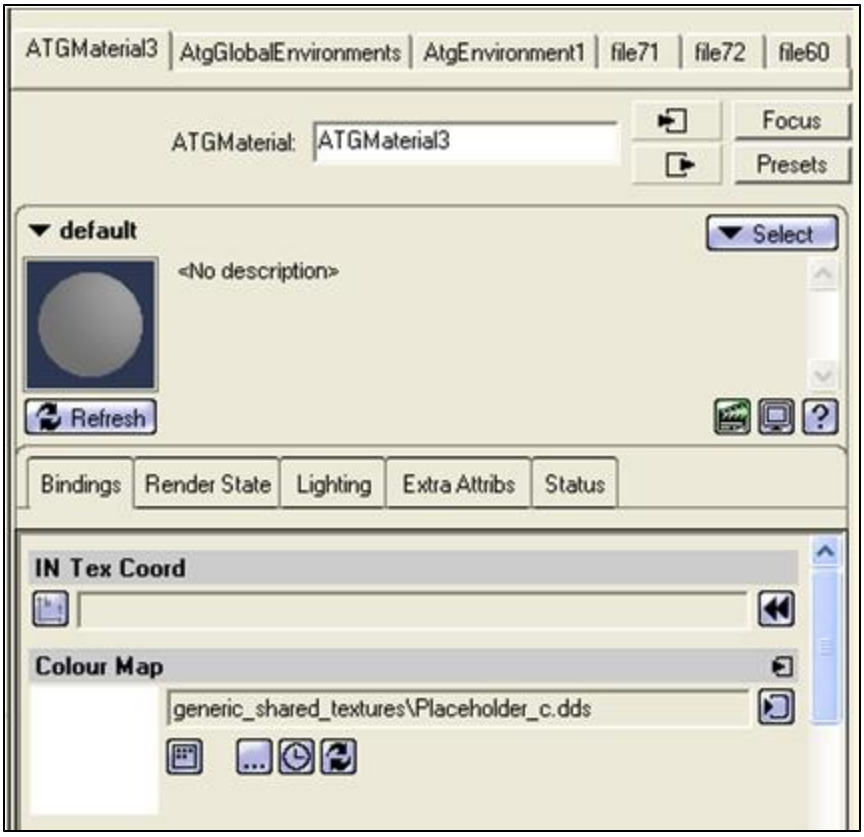
13. Dynamic Frames Only: Dynamic Frames allow users to load pictures from their PlayStation®3 HDD. The picture is loaded onto a texture that you identify as the texture for the picture to replace.

    In Maya, you assign as placeholder texture to the geometry shape on which you want to users to place pictures.

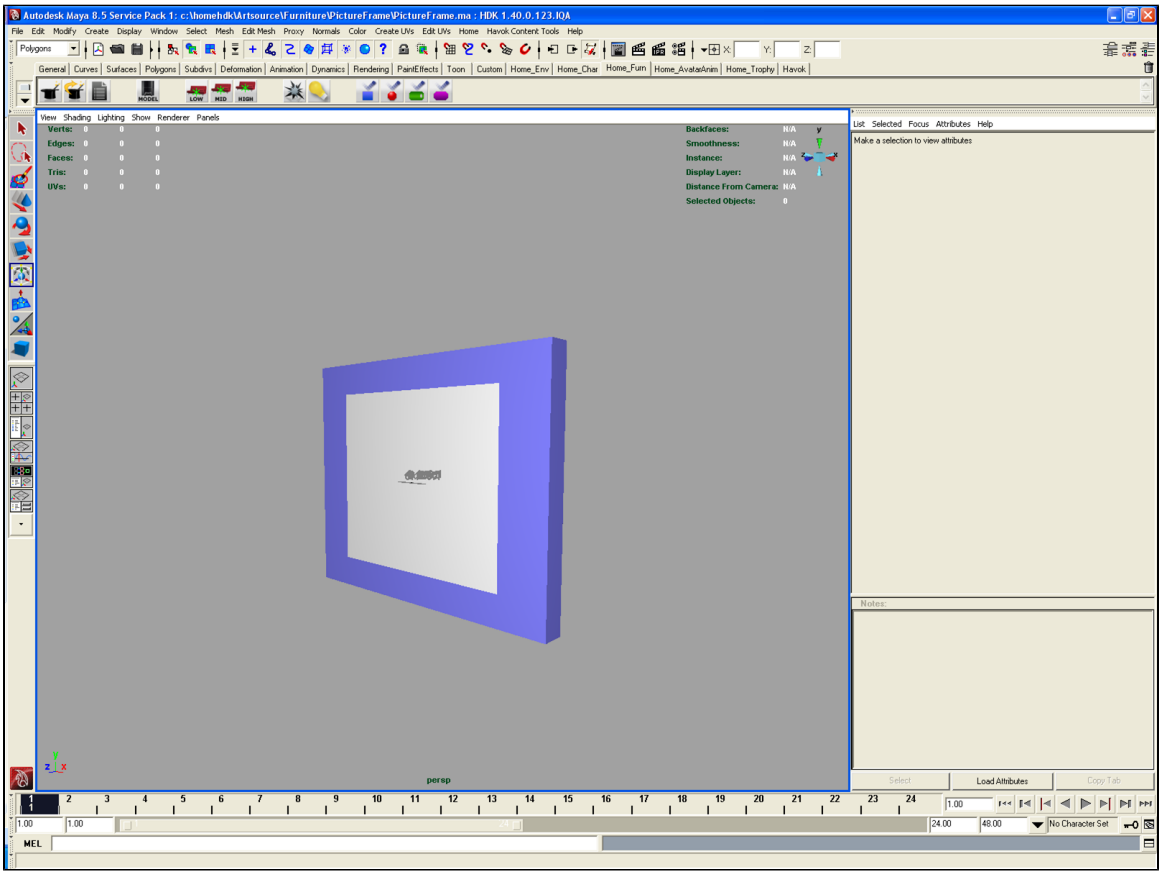
    In the Object Editor, you identify the placeholder texture in the properties of the Furniture component. See Exporting Picture Frames.

    To Assign a Placeholder Texture:

    a. Create another ATG Material, and select default type.
    b. In Material Parameters > Colour Map, choose a placeholder texture. This example uses the Placeholder_c.dds texture in <HDK_INSTALL_DIR>\artsouce\Generic_Shared_Textures:



c. After you have applied the textures, press the 6 key to see the changes.



Your picture frame is now ready to export. See Exporting Picture Frames:

For more information on creating furniture, see Furniture and Decorations and Maya.

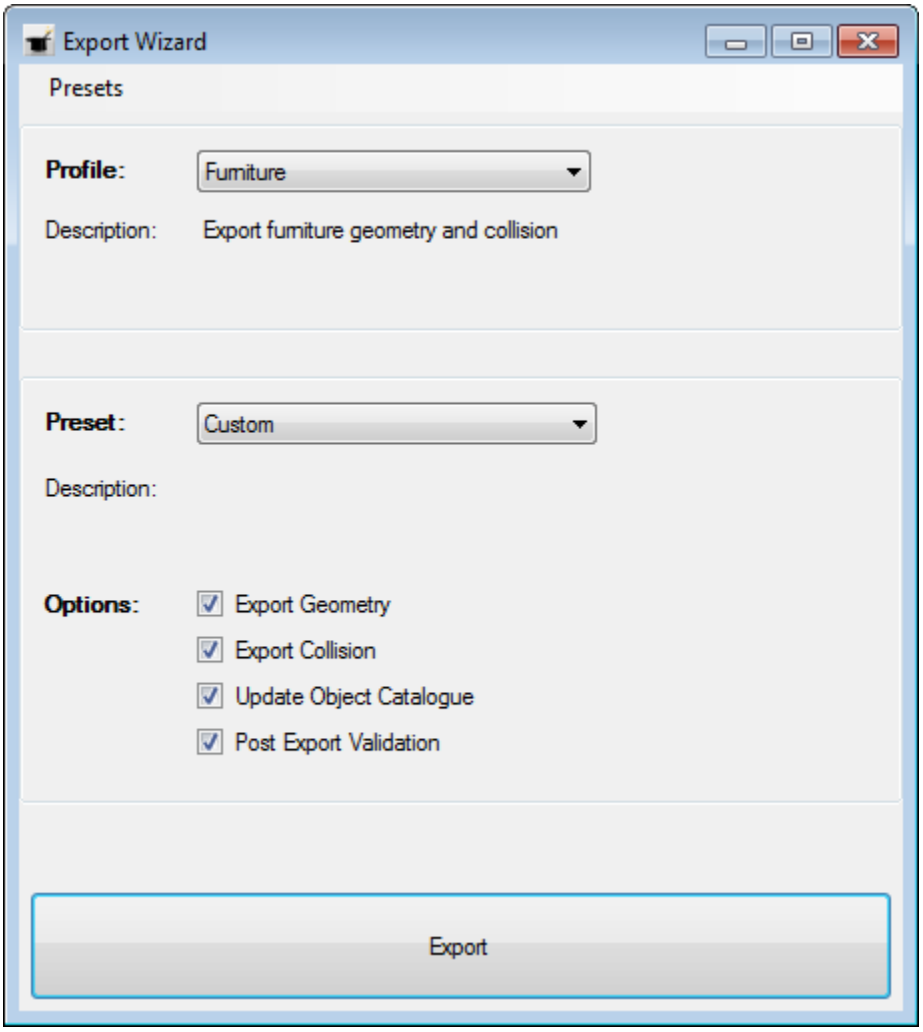
Exporting Picture Frames

This section applies to any type of picture frame made in Maya.

To export picture frames:

1. Select Home > Export Wizard or click  .

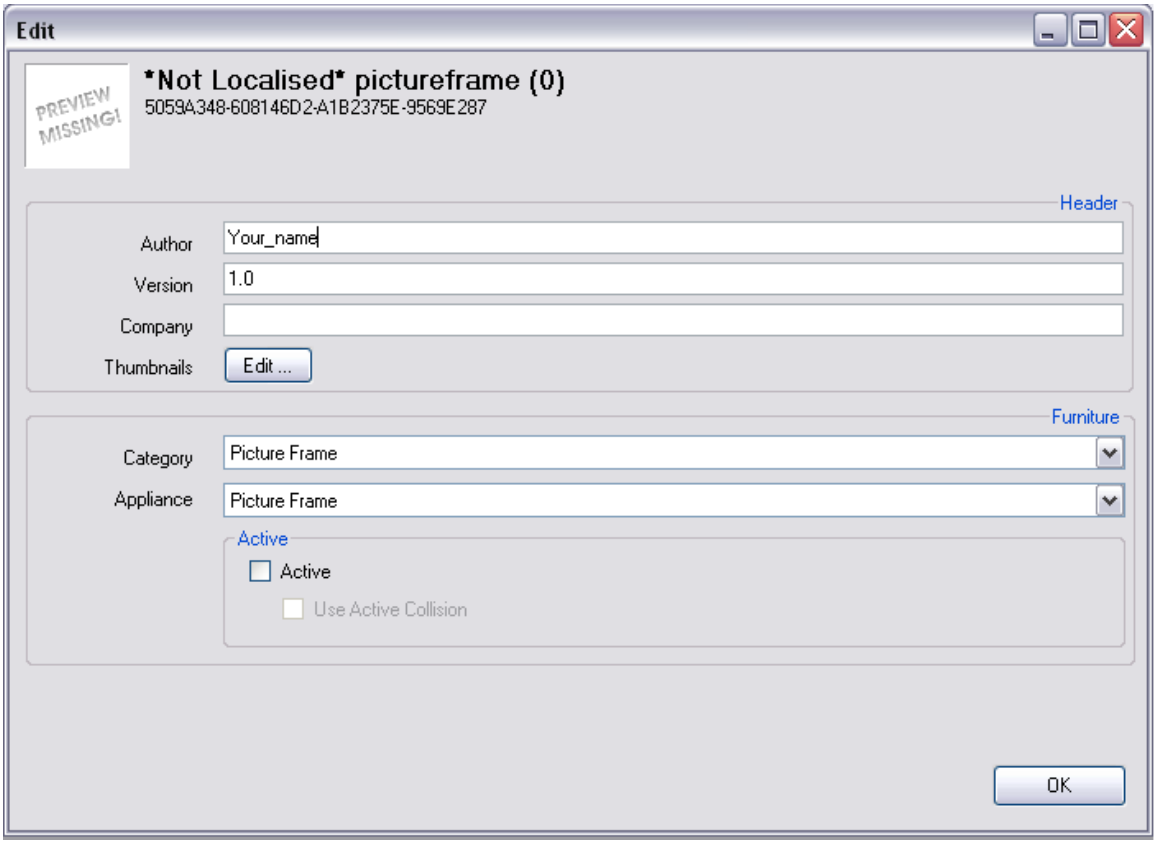
   The Export Wizard dialog is displayed:



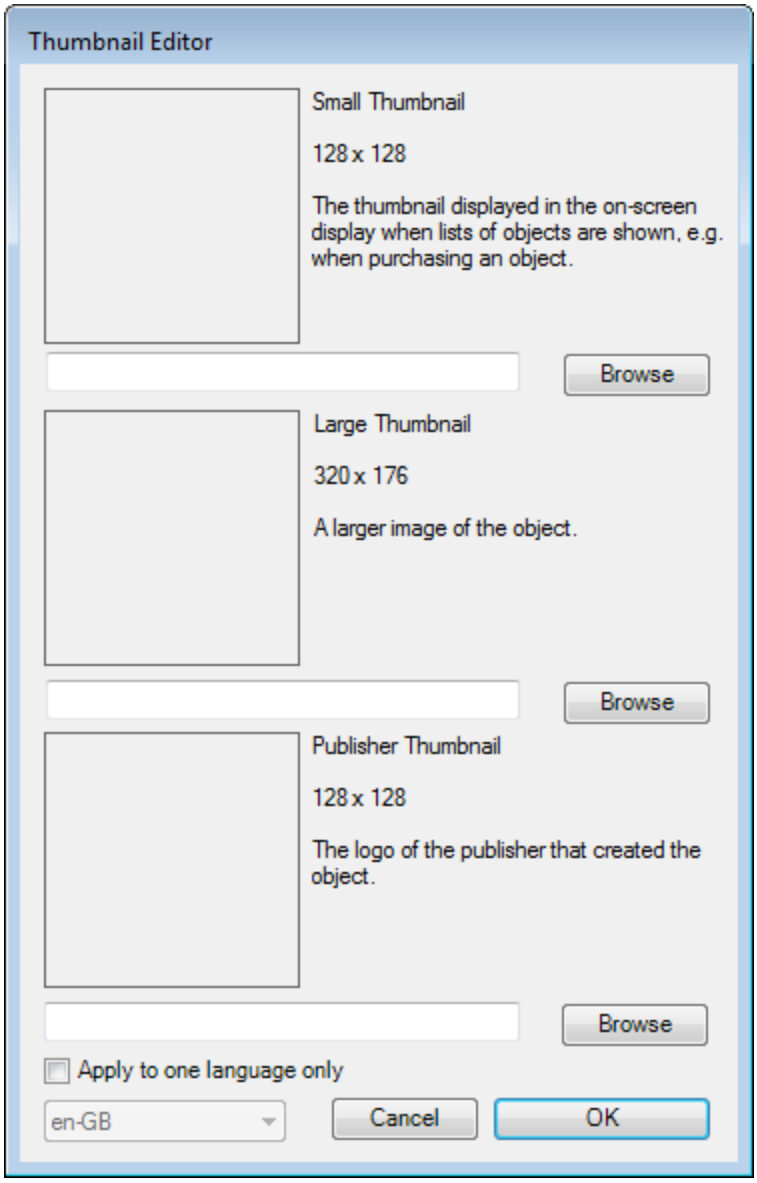
2. Select Furniture from the Profile drop-down list.
3. Ensure all of the export option boxes are checked.
4. Click Export. The Furniture Object Editor dialog is displayed:



5. Enter the name you want to use when exporting your object in the field displayed.
6. Click the pencil icon. The Edit dialog displays the parameters that need to be set for the object metadata:



7. Select Picture Frame from the Category drop-down list.
8. Select Picture Frame from the Appliance drop-down list.
9. Click Edit next to Thumbnails to select the thumbnail images of the item in the Thumbnail Editor dialog:

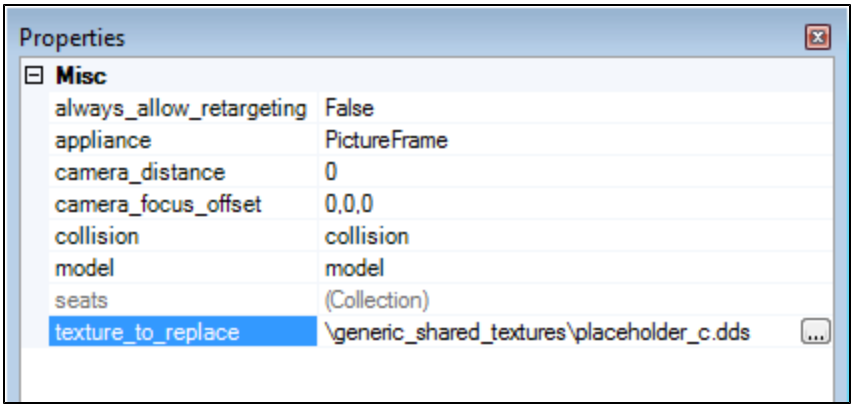


For more information on selecting thumbnail images, see Editing the Thumbnails.

10. To export the Picture Frame object, click Commit All Changes in the Furniture Object Editor dialog. At this point, Static Picture Frames are now complete. You need only complete the steps required for all objects (age rating, profiling, etc.).

11. For Dynamic Picture Frames only: complete these steps:

    a. Open the Object Editor.
    b. Select the object just created.
    c. In the Object View panel, select the Furniture component.
    d. In the Properties panel, click the button next to the texture_to_replace field and browse to and select to the texture to be replaced. In this example, \generic_shared_textures\placeholder_c.dds was used:



Your Dynamic Picture Frame is now complete. You need only complete the steps required for all objects (age rating, profiling, etc.).

Adding Picture Hooks

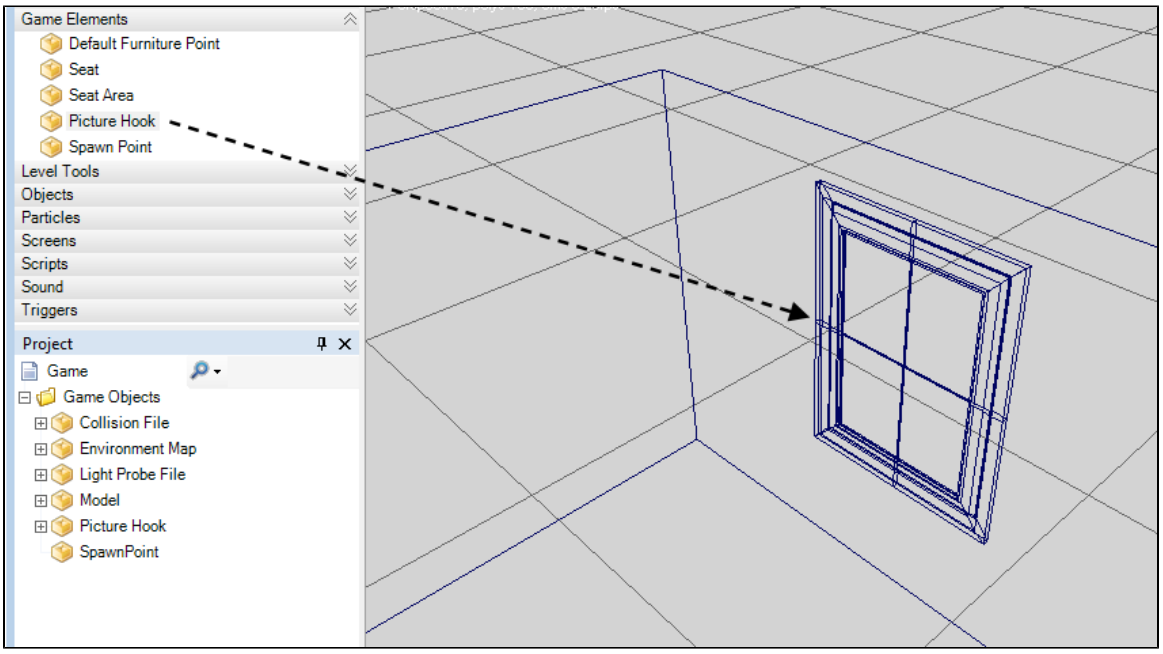
Picture Hooks allow users to place picture frames (static and dynamic) on walls in personal spaces, so that they can then hang pictures on the walls. You add picture hooks to the personal space using the Scene Editor. You don't need an asset file to add them to a scene.

You should add at least three picture hooks to your scene so that users can use their picture frames as they acquire them.

For more information on picture frames, see Picture Frames and Wall Decorations.

To add a picture hook:

1. Open the scene in the Scene Editor.
2. Drag the Picture Hook object from the Palette panel to either the Project panel, or directly onto the scene.
3. Position the Picture Hook object against a flat wall:



Picture frames are not designed to be placed on curved surfaces, so avoid placing picture hooks on curved walls.

Picture frames can be placed on picture hooks and rotated to any angle. Leave enough room around the edges for this to work.

Placing a hook very high up makes the hook and picture frame difficult to target. Test picture hooks in scenes before publishing them in the live environment.

See also Designing a Home Space.

Picture Frame Commerce and Regional Restrictions

The following restrictions apply to Picture Frames:

- Dynamic Picture Frames are only available in the SCEJ, SCEA and SCEAsia regions. They are not available in the SCEE region.
- Static Picture Frames are available in all regions.
- Picture Frames can only be purchased within the region they are available. For example, Picture Frames within SCEJ region can only be purchased in SCEJ, but not in SCEA, SCAsia, or SCEE regions.
- Pictures Frames acquired in one region cannot be sold in a different region.
- Picture Frames acquired in one region are visible in any other region where Picture Frames are available:
    - SCEJ users can see SCEA Picture Frames and vice versa
    - SCEE users will only see the default texture (texture_to_replace) in Dynamic Picture Frames.
    - SCEE users can see Static Picture Frames.

Creating a Seat

To make a piece of furniture that avatars can sit on, you must set up your furniture item as a Seat object.

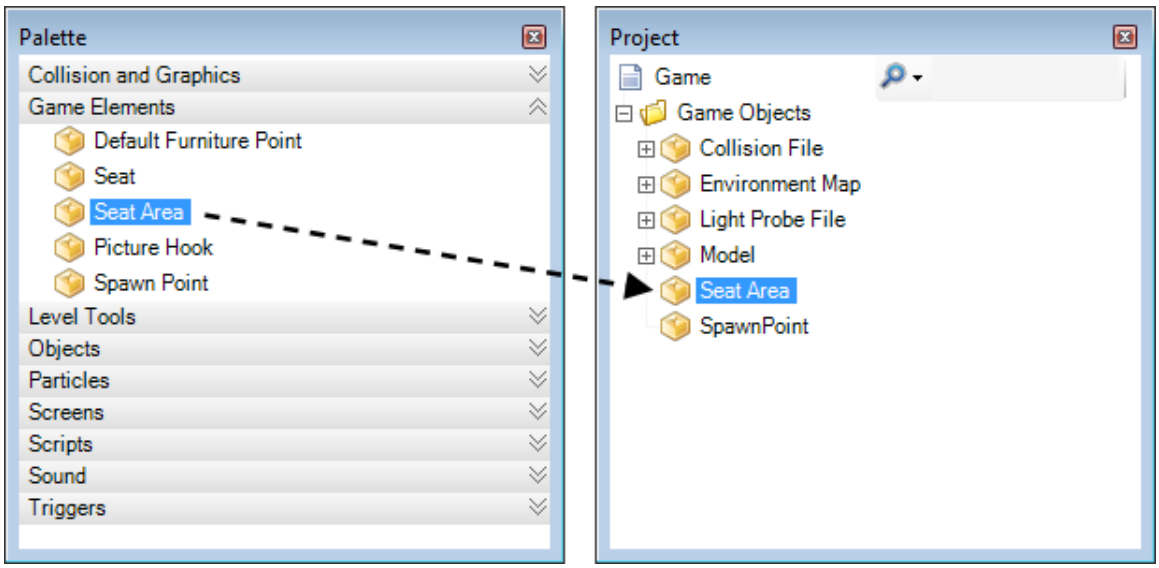
Adding Seats in Personal Spaces and Clubhouses

To add seats to your personal space or clubhouse, you cannot just import furniture as a user would if they were in the space. You must first design the geometry and collision of the seat in Maya. Using the Scene Editor, you then add the Seat Area and Seat objects from the Palette panel to turn that area of geometry into a seat.

> You can use this process for public spaces as well. With public spaces, you can also use Scene objects from the Object Editor to decorate your space. For more information, see [Embedded Objects].

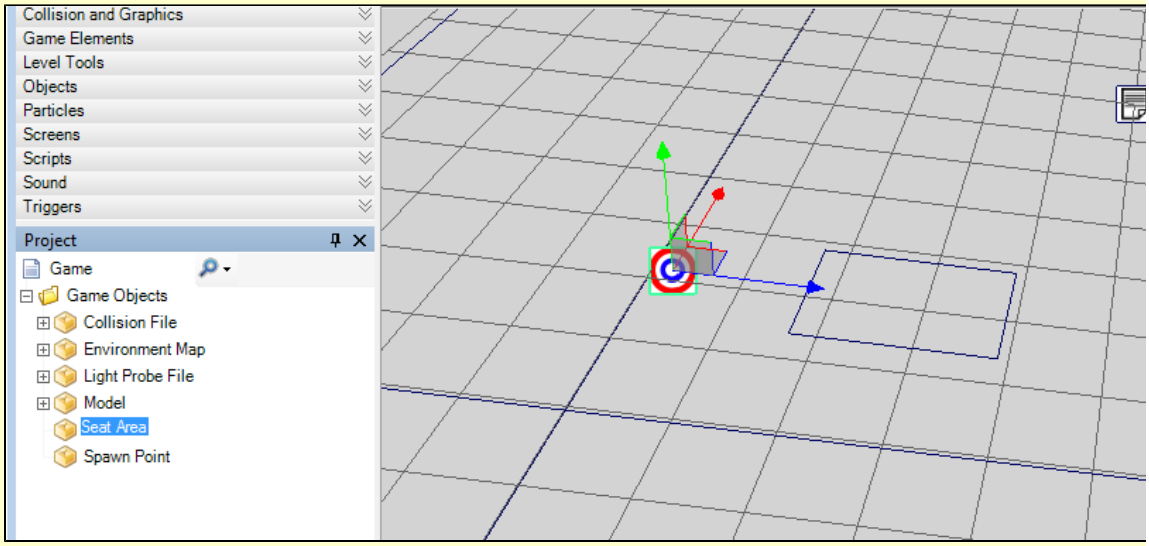
To add a seat:

1. In the Scene Editor, drag the Seat Area object from the Palette panel into the Project panel, or directly onto the scene:

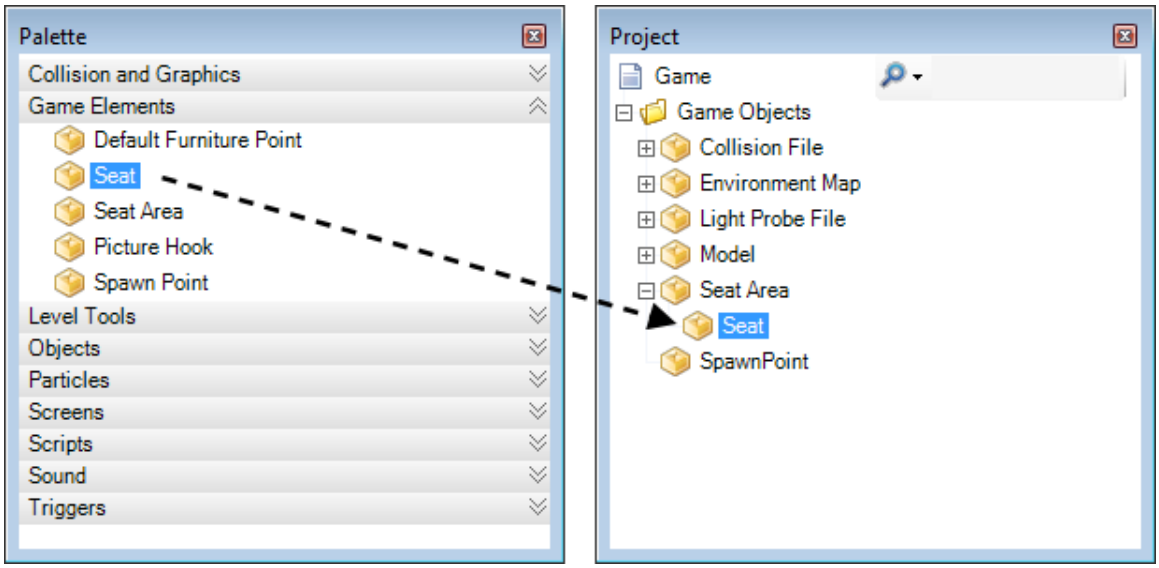


2. Position the Seat Area on the scene to where you would like users to be able to sit down.

In Maya, the Y Positive axis is the front. When using pivot rotation to align furniture with seating locators, keep in mind that the seat's front positive may not be the furniture's front positive after pivot rotation. Make sure that you rotate the seat locators only on the y-axis. If you rotate them on other axes, the position locator in Maya will not match the position of the locator in PS Home. Model all furniture pieces so that world 0 is the floor level:



3. Drag the Seat element from the Palette panel onto the Seat Area element in the Project panel:



The Seat appears in the Seat Area, and your seat is now created in the space.

Seat Locators

Select a Character Seat Locator of the appropriate height from the Furniture Tool Shelf in Maya (Home_Furn). You can choose from the following:

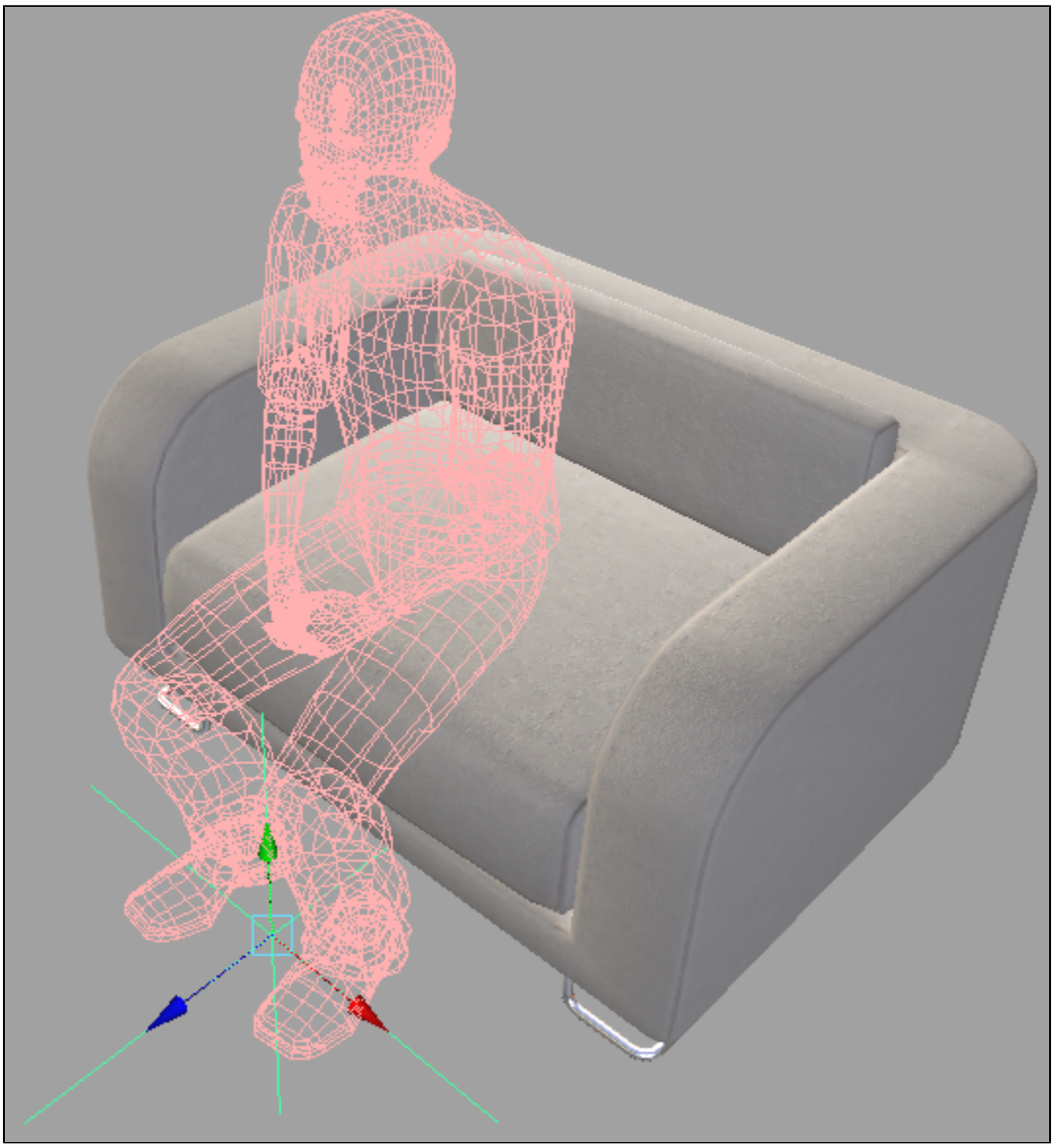
- Low: reclining seat height (0.35 m)
- Medium: standard seat height (0.5 m)
- High: bar stool style seat height (0.7 m)

The seat heights are not exact measurements, so place your Character Seat Locators in Maya before modeling your seat. You can then build the geometry as far as possible to fit the avatar pose.

For more information and illustrations of seat heights, see Dimensions.

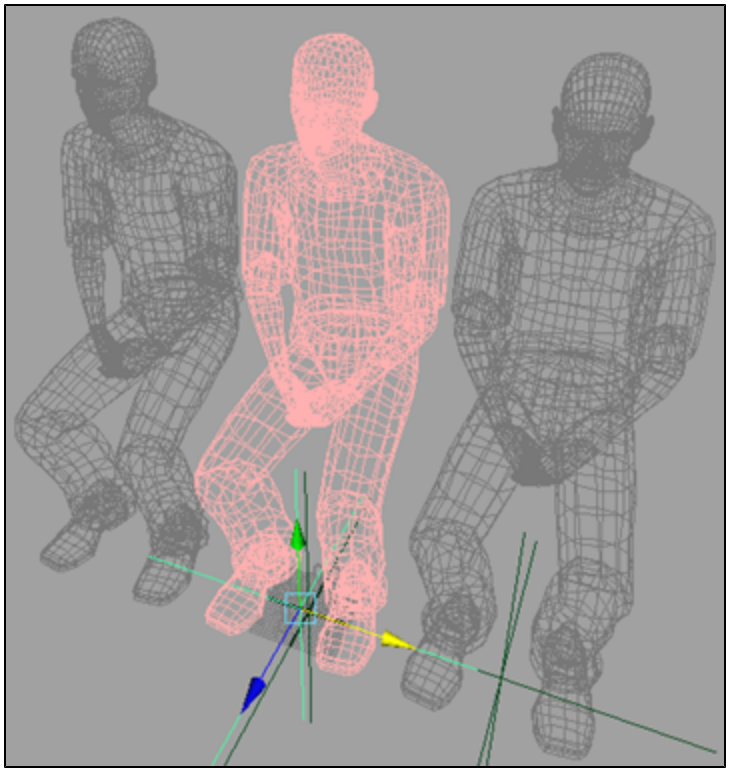
Positioning Seat Locators

You can reposition seat locators (named HomeSeat) wherever you want. The position information is automatically included when you export your model:



Multiple Seat Locators

Furniture items can have up to three seats. For furniture that seats more than one, just add more Character Seat Locators:



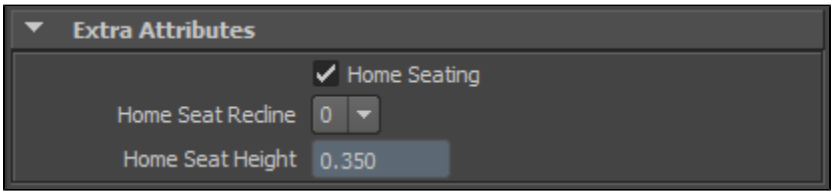
The tools (Maya, Scene Editor and Object Editor) automatically validate the number of locators and generate an error if you exceed the maximum number allowed.

Reclining Seats

When creating your seating locator, you can use the Recline parameter so that the avatar can lean back into a relaxed position (for example on a recliner sofa).

To set the Recline parameter:

1. Select a HomeSeat.
2. In the Attribute Editor, select Extra Attributes.
3. 0 (no recline) is selected in the Home Seat Recline drop-down list by default:



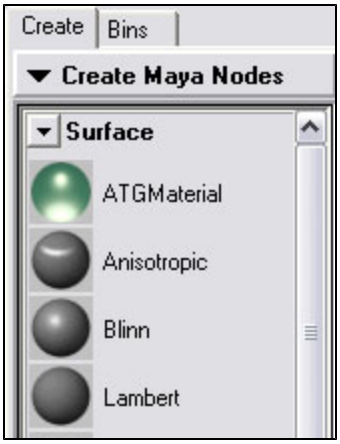
To set up a reclining seat, select 1 from the Home Seat Recline drop-down list.

Selecting Default Shaders for Furniture

The ATG Shaders used in designing Home content are not present until the Home HDK has been installed.

To select the default shaders for furniture in Maya:

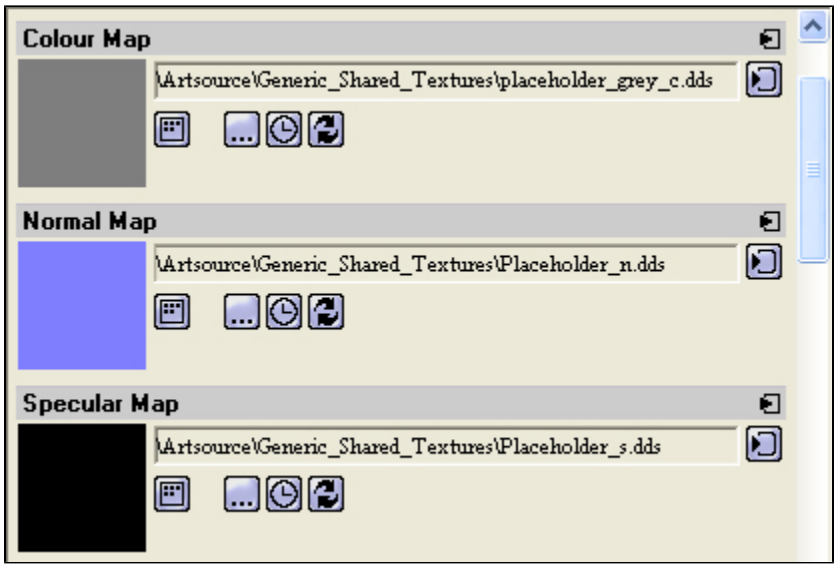
1. Open the Hypershade dialog and select ATGMaterial from the Materials list:



2. Open the Attribute Editor and select the default material type from the Select drop-down list:



The ATG Shader is automatically configured with three mapping nodes as follows:



Each map channel has a default placeholder texture. You can replace any of these textures with your own texture maps, leaving the placeholders in any unused channels.

Menu Categories for Furniture

The furniture menu in the client has specific categories where furniture and decorations are listed by type:

- Appliances
- Chairs
- Cubes
- Flooring
- Footstools
- Lights (candles, desk lights, standard lamps, etc.)
- Ornaments
- Picture frames / Wall hangings
- Sofas

- Storage
- Tables

The category a furniture item appears in depends on what is set in the metadata description of the object.

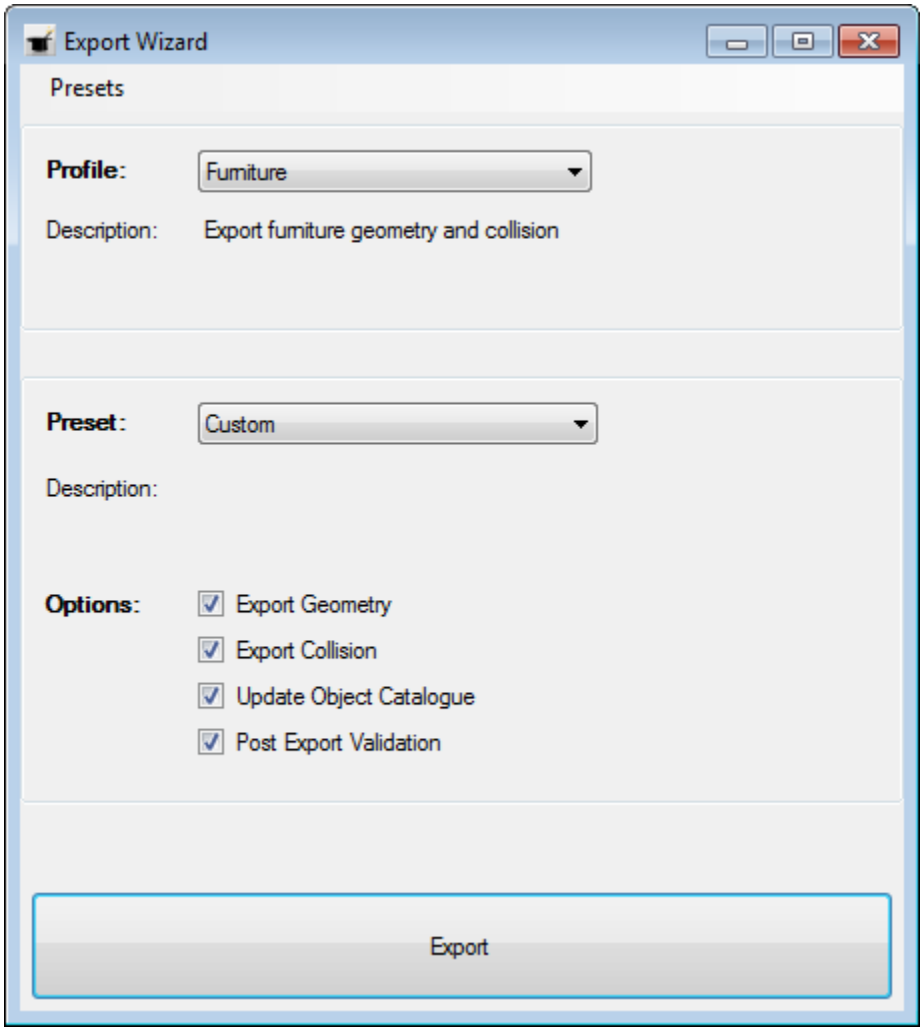
Validating and Exporting Furniture

To export a furniture item:

1. Launch the Export Wizard from the Home drop-down menu or by clicking  .

   The Quick Export option (also available from the Home Tool shelf or the Home menu) exports the scene using the last settings used.

   The Export Wizard dialog is displayed:



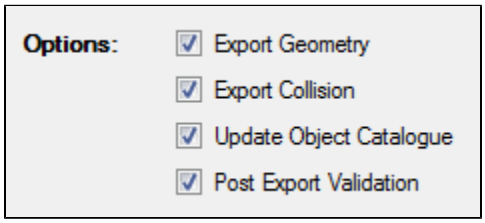
2. Select the Furniture export profile from the Profile drop-down list.

   When you export, the project's data is exported to an intermediate folder, which is automatically created with the same name as the Maya file when the first export takes place. This folder contains a number of files required by PS Home to build the final environment and an export log file that contains useful data about the export.

3. Select the export quality from the Preset drop-down list.

The Default preset is currently the only option available for furniture.

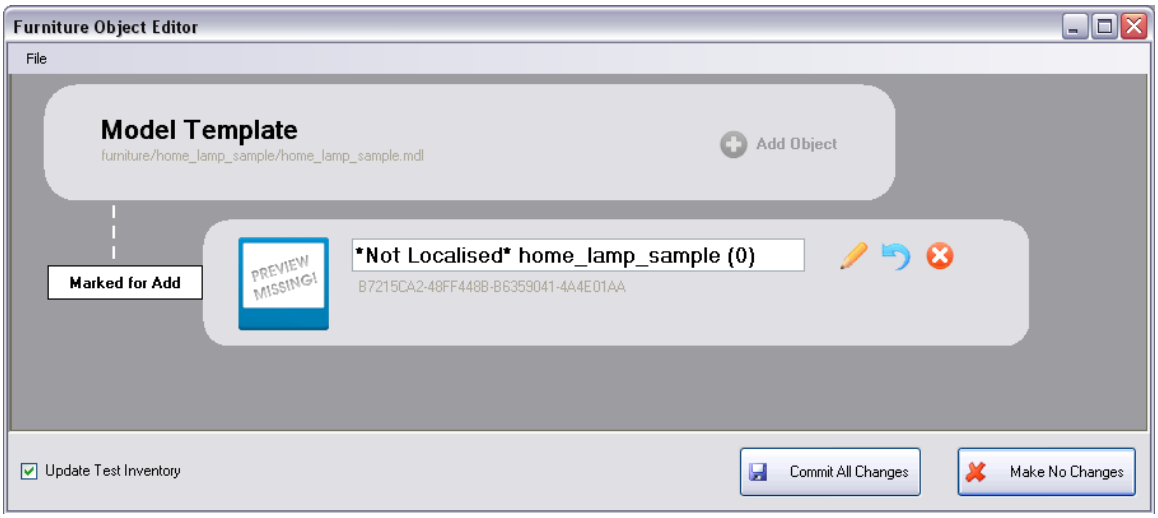
4. For custom presets, check or uncheck the export option boxes, as required:



All the boxes are checked by default and on most occasions can be left. However, if for example, you are working on just your collision geometry, you can uncheck all the boxes except Export Collision. This saves the exporter having to re-export the geometry.

If the Post Export Validation box is checked, the Export Wizard validates your content to make sure it is compliant with PS Home requirements. For more information on what is validated, see HDK Tools Validations. Only uncheck this box if you are sure that you will not be packaging the content for submission to PS Home (for example, if you are just testing the export to Scene Editor or Object Editor).

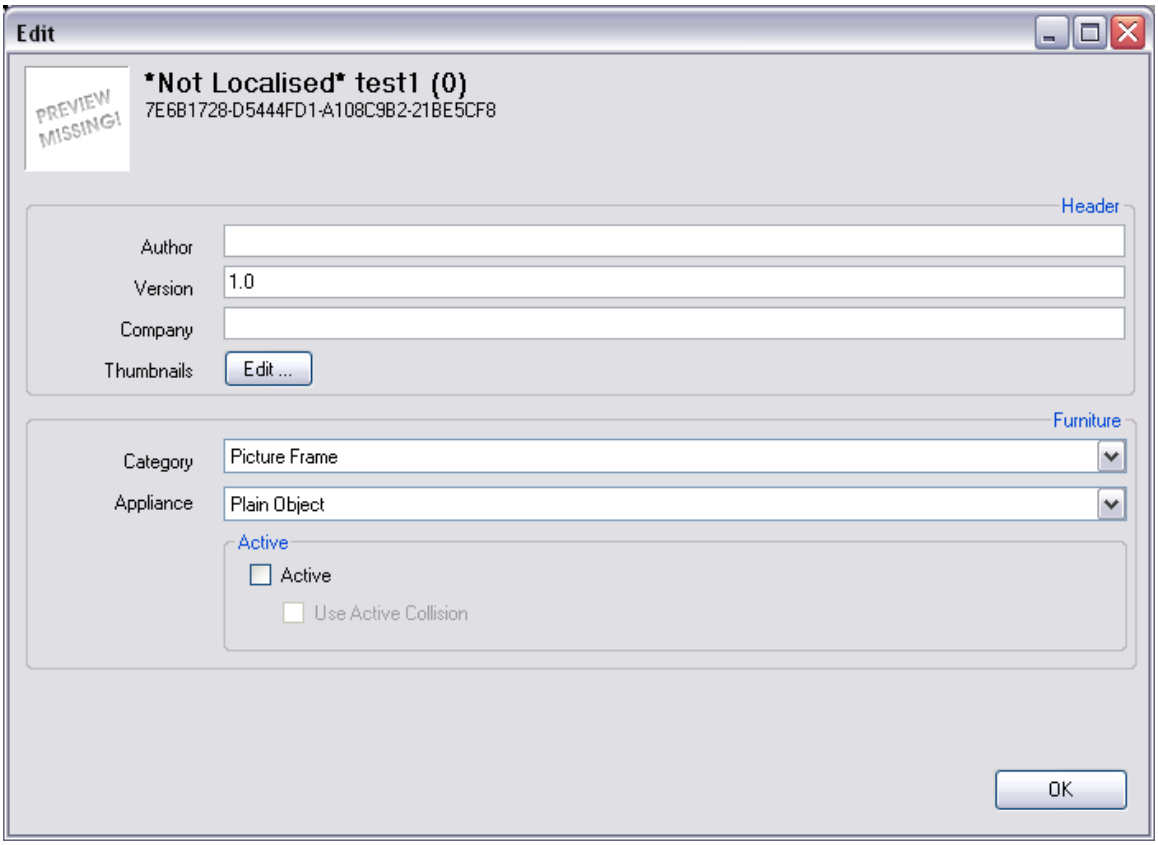
5. Click Export to start the export of the data.
6. During export, the Furniture Object Editor loads the object catalog. You can use this tool to add object data to your exported object, so that it is already recorded in your object when you open it in the Object Editor later. This can be useful, as you can add names, descriptions and thumbnail images that make it easier to identify the object in the object catalog:



To edit the object data, click the pencil icon:



The Edit dialog is displayed:

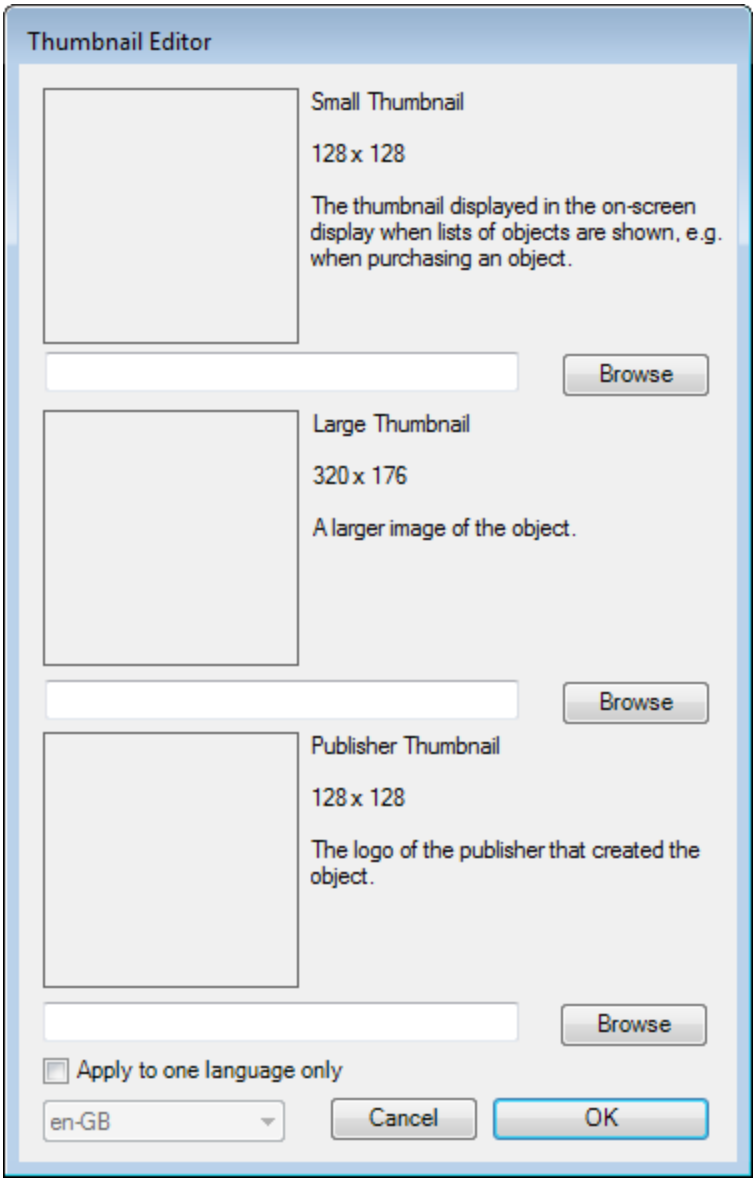


Edit each field to your requirements:

a. Enter the name of the creator of the item in the Author field. This attribute does not appear in PS Home and is only for your own reference.
b. Enter the version number of the item in the Version field. This attribute does not appear in PS Home and is only for your own reference. We recommend versioning as follows:

    - First version of the furniture: 1.0.0
    - Any update during QA: +0.0.1
    - Resubmission (for whatever reason): +0.1

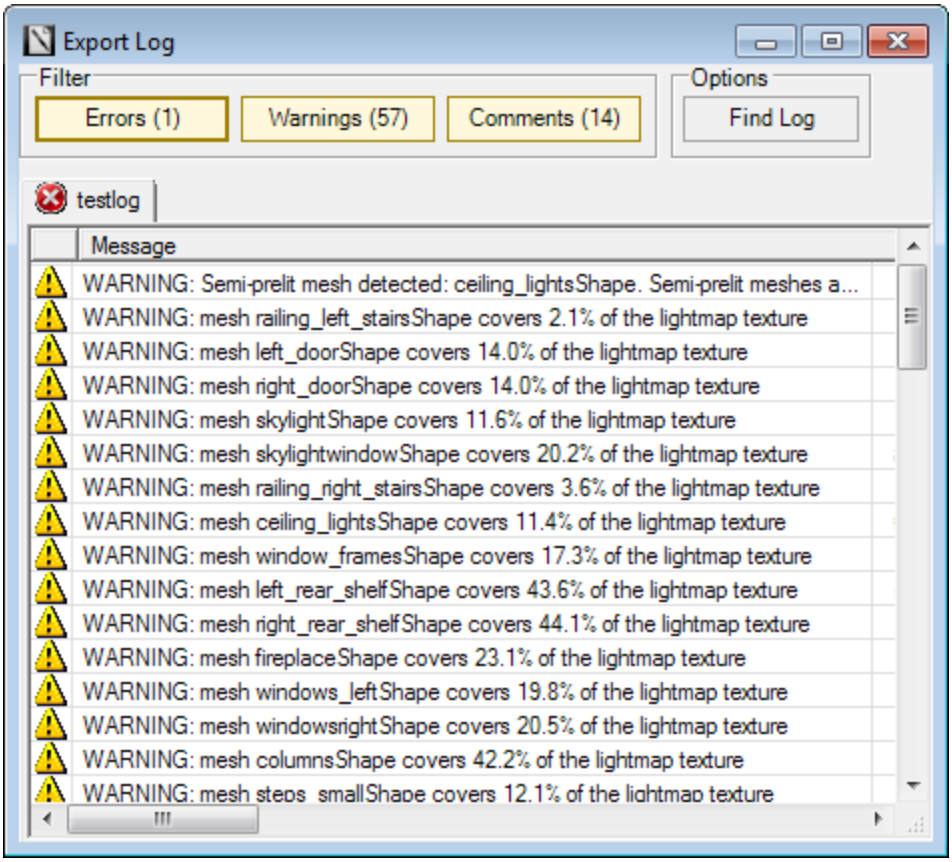
c. Enter the name of the company for whom the item has been created in the Company field. This item does appear in PS Home.
d. Click Edit next to Thumbnails to select the thumbnail images of the item in the Thumbnail Editor dialog:



For more information on selecting thumbnail images, see Editing the Thumbnails.

e. Select the category in which the item will appear in when the user is adding new furniture in PS Home from the Category drop-down list.
f. Select any special functionality that the furniture might have from the Appliance drop-down list. You should select Plain Object for all furniture items, except for Seats and Picture Frames as these items have special behaviors.

7. At the end of the export process, the Export Log dialog displays the results of the export and lists any errors  , warnings  and comments  , for example:



If there are no errors, select Next to move onto the next step. If the log reports errors, you must correct them before continuing.

If your export is successful, "Export Complete" is displayed.

A Home model file (.mdl) containing the geometry and a matching Havok file (.hkx) containing the collision data are created. In addition, an object file (.object) is exported that references all the other files to create the final Home furniture item.

When validation and export have successfully completed, you can edit the scene or object/component created in Maya (if necessary) and package it in the Editing tool. You use the Scene Editor for scenes and the Object Editor for objects or components. The scene or object/component must validly export and be packaged by the associated Editor before you can submit it for quality assurance.

Textures for Furniture

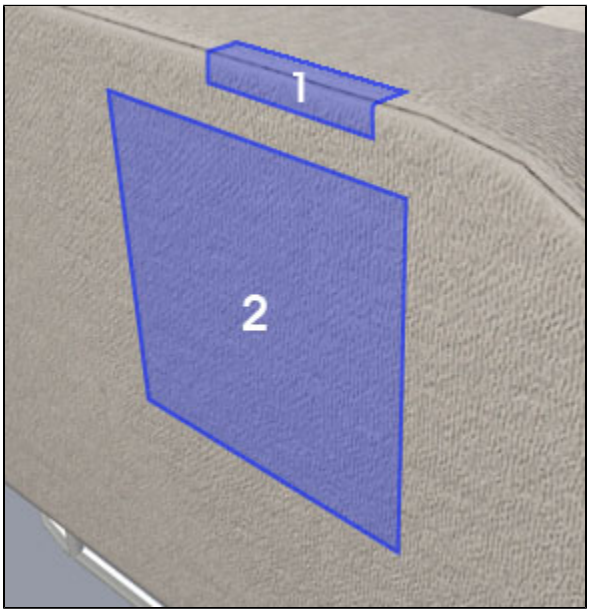
Texture Allocation

The texture allocation is the equivalent of a 512 texture set made up of three 512x512 pixel maps, a color map with alpha channel, normal map and specular map with alpha channel.The allocations are saved in the DDS NVIDIA format with the DXT5 compression. You can split this 512x512 set into any size, providing both dimensions are to the power of 2 (i.e. 16, 32, 64, 128, 256), for example, 256x256 and 128x512.

	
Texture 2 with 256x256 pixel resolution	Texture 1 with 256x64 pixel resolution

Workflow

The aim is to achieve the highest fidelity with as little texture space as possible. Tiling textures have been proved to give the level of fidelity required for things, such as fabric, so only use unique textures where absolutely necessary. The following image shows a typical example of the size of one tile of each texture:

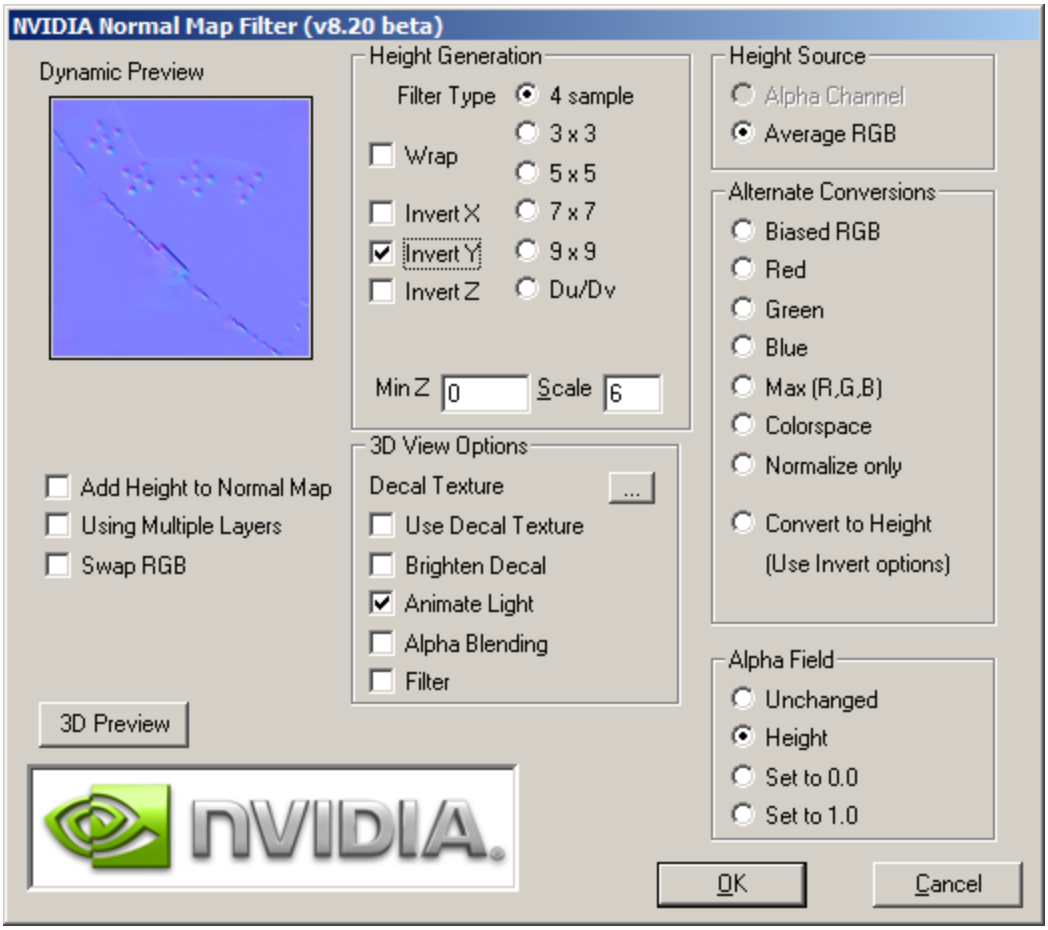


As you can see, textures 1 and 2 are laid out according to their tiling regions

Flat color materials are often present in the furniture for materials such as chrome, so you need only create a color swatch 16x16 to achieve the required result.

Normal Maps

Normal maps can be created using the NVIDIA Photoshop filter. By creating the equivalent black and white bump map and running the filter will achieve the correct results. The following image shows the correct setup for the NVIDIA filter.



The only options you should change are the Filter Type and Scale. Smaller detailed textures such as fabric would suit a sample type of 4 or 3 and a scale of 5, whereas large details would suit a higher sample type and larger scale.

Specular Maps

Home supports Specular Intensity plus Specular Power, that is Specular Color and Specular Roll Off respectively. The Specular Intensity is represented by the RGB channels in the image, and the Specular Power is represented by the alpha channel.

We recommend that you keep the original .PSD file and all its assets. These files are not needed to package PS Home content, but can be used if re-editing content. The example pack is supplied with a typical .PSD and its elements.

The Furniture Shelf and Furniture Menu

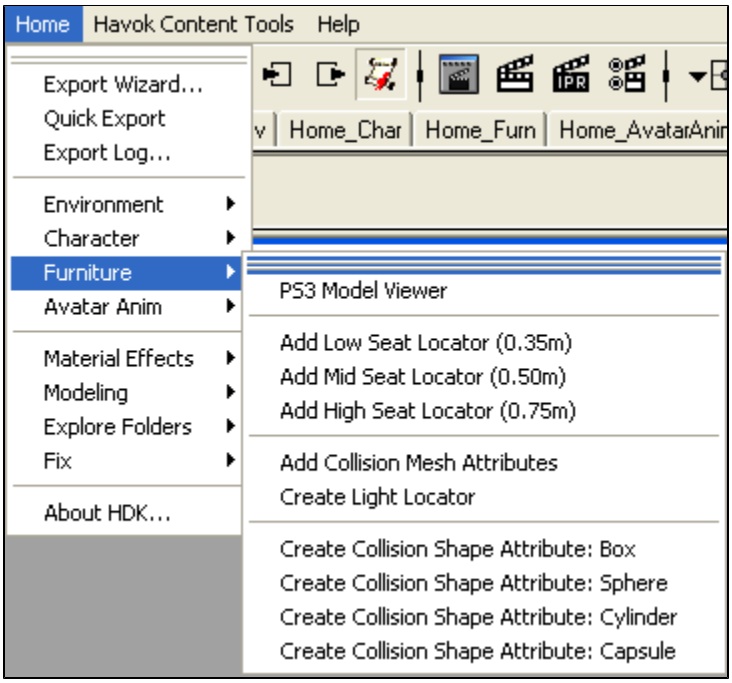
The Furniture Shelf, Home_Furn contains tools specific to creating furniture, as well as some common tools.

The Home_Furn tab has the following buttons:

Button	Name	Description
	Export Wizard	Launches the Export Wizard.
	Quick Export	Exports the scene using the last settings.
	PS3 Model Viewer	Launch on PlayStation®3.
	Add Seat Locator	Places a Character Seat Locator item at the origin of the current Maya Scene with the name 'HomeSeat'. These items show the seating positions on your furniture and display the space used by a seated character.

	Collision Mesh Attributes	Adds a Convex Mesh parameter to the currently selected shape.
	Create Light Locator	Adds a light locator item that can be moved into position to determine the location of the point light source for a Lamp. See Creating a Lamp.
Unable to render embedded object: File (collisionattribute.png) not found.	Collision	Creates a collision box, sphere cylinder or capsule shape, respectively. See Collision in PS Home.

These options are also available in the Home > Furniture menu:



Rules and Conventions for Modeling Furniture

This section gives recommendations to help you to make furniture items efficiently and economically.

To make sure that your furniture items pass export validation, see:

- Dimensions for furniture size recommendations.
- Content Requirements for the technical requirements of your Home content.
- Profiling in the Client for information on profiling your Home content.

Collision

Use primitive shapes when making furniture collision.

See Collision For Furniture and Objects for more details.

Axis

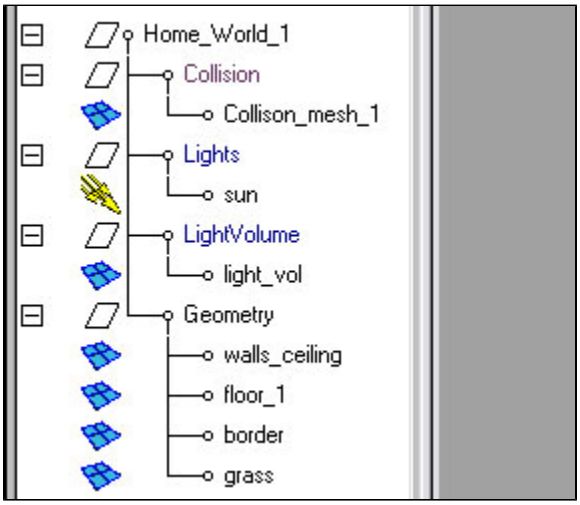
Model all furniture pieces so that world 0 is the floor level.

In Maya, the Z Positive axis is the front (a chest of drawers, for instance, would have drawers facing positive Z). When using pivot rotation to align furniture with seating locators, keep in mind that the seat's front positive may not be the furniture's front positive after pivot rotation.

Make sure that seat locators are rotated in the y-axis only. If they are rotated in other axes, the position locator in Maya will not match the position of the locator in PS Home. You must rotate it parallel to the ground and not put at an odd angle.

Hierarchy

Place your collision geometry under the Collision Group/node. The following image gives a view of the correct hierarchy setup:

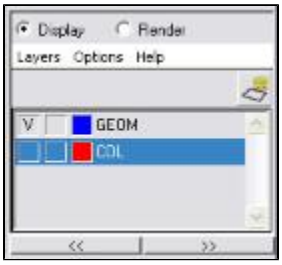


Layers

For organization purposes and ease of use, make two layers:

- A layer named GEOM containing the Geometry Group/Node and its children.
- A layer named COL containing the Collision Group/Node and its children.

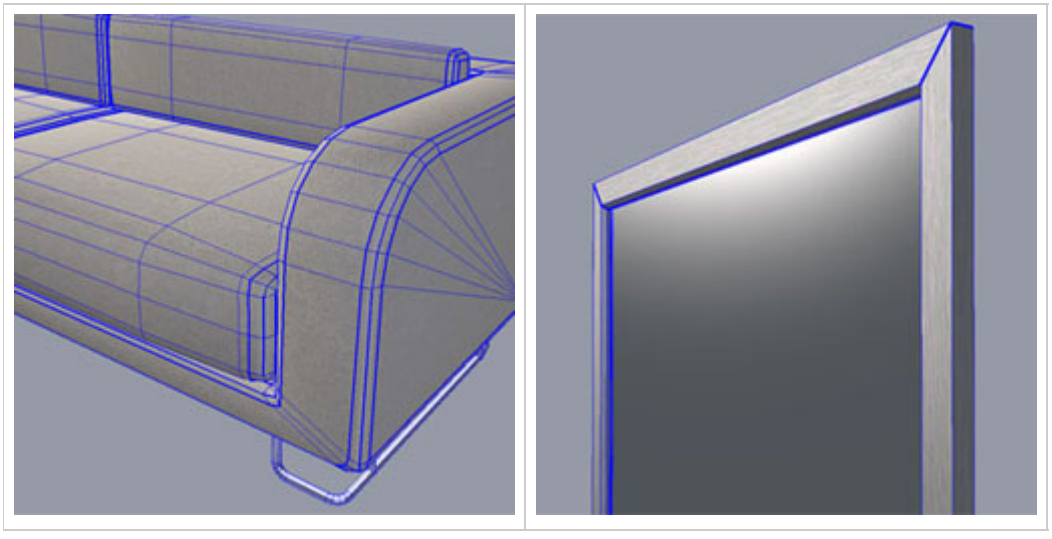
The following image shows the correct layers setup:



Level of Detail and Triangles

Although there is a recommended maximum, keep models efficient by using as little geometry as possible whilst retaining the required quality and level of detail.

For example, a sofa may be 3,850 triangles, but a picture frame consists of 84 triangles, as shown in the following images:



Sofa with 3,850 triangles	Picture frame with 84 triangles

4,000 triangles should be sufficient, so try and avoid long hard edges by beveling and cutting-in where necessary.

Naming Conventions

Assets

All assets in PS Home follow the same naming conventions. All names are capitalized and use underscores for separation of category. For example, all assets created for 'Brand Name' would be named in this format: BrandName_AssetType.

Examples of correct and incorrect naming:

Correct Name Format	Incorrect Name Format
BrandName_DesignType.ma BrandName_DesignType_c.dds BrandName_DesignType_n.dds	brandName_designtype.ma BrandName_designtype_col.dds brandname_design_type_n.dds

Textures

All Textures within Home have a specific suffix depending on whether they are color, specular or normal maps. You must name all your textures via the previous naming convention adding the appropriate suffix.

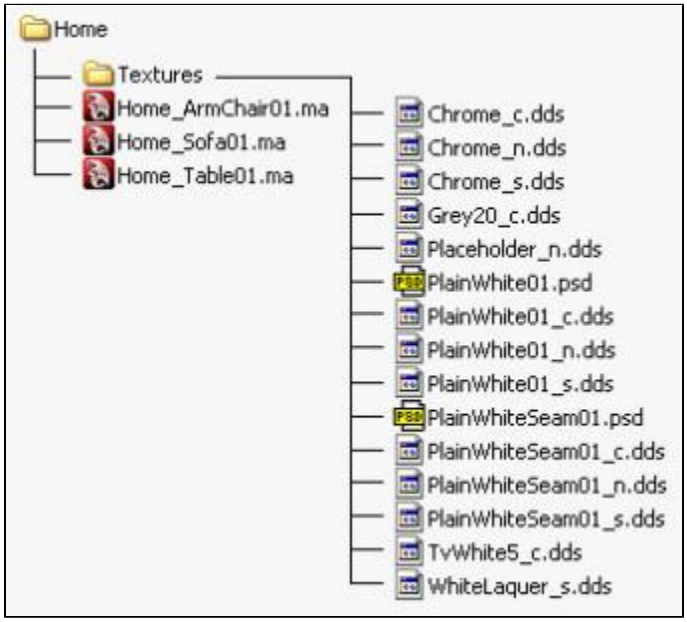
Color maps add '_c', specular maps '_s' and normal maps '_n'.

For example, a sofa upholstery color texture would be named Home_SofaUpholstery_c and the normal map would be Home_SofaUpholstery_n followed by the appropriate file extension.

Project Directory

The project directory is a folder named by the scene file, such as chair_1 or Home_ArmChair01. It contains another folder named Textures. Place all your DDS texture files in this folder. Your scene will then source textures from this folder.

The following image shows a project folder containing all relevant files:



UV Sets

You need one UV set, 'map 1'. Make sure your poly objects do not have more than 1 UV set.

The default UV set 'map 1' is used for your color maps and you can organize this to suit your texturing. Keep in mind the aim is to texture these models with tiling textures.

To face forward, the furniture should point in a positive Z direction with Y being the vertical axis. For example, if creating a sofa the seats should face outwards towards positive Z.

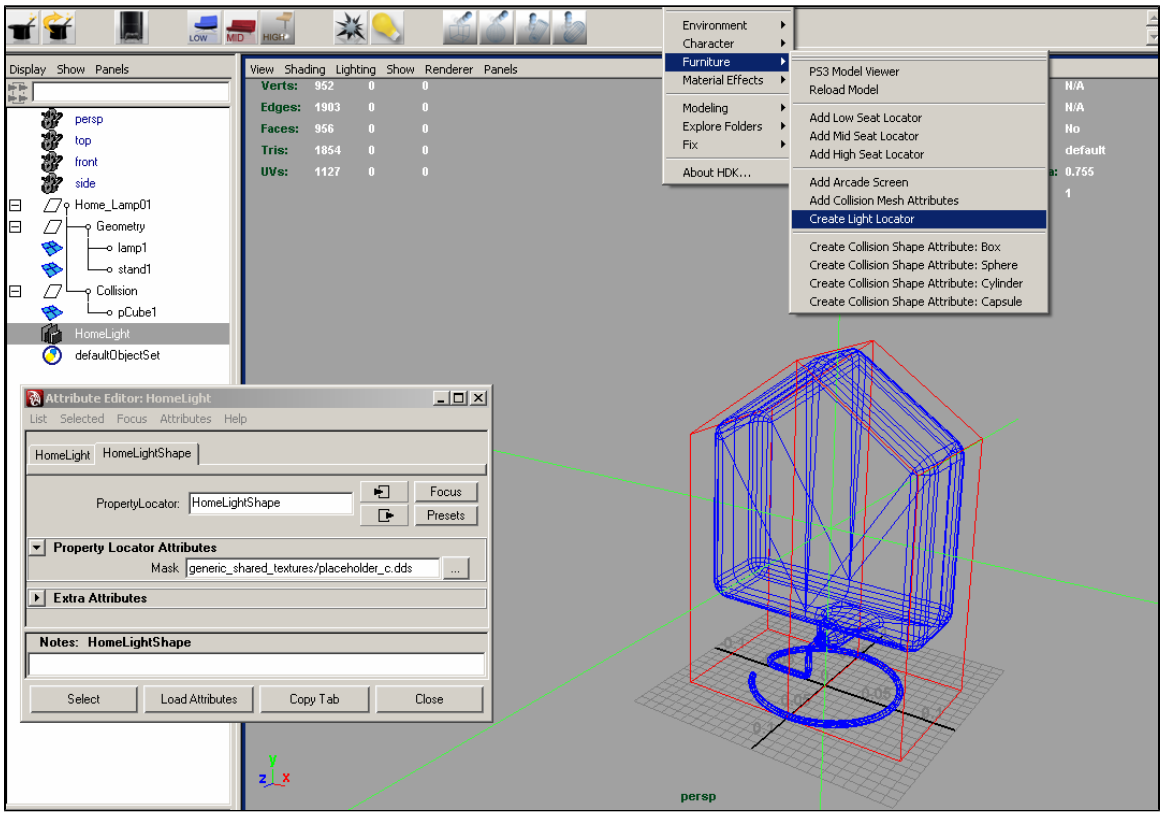
Creating a Lamp

A lamp in PS Home is an item of furniture that has simple user interaction that allows the user to activate or deactivate a light source.

> Users can turn lights on and off, but this behavior must scripted. Lamp furniture objects within Public Spaces do not allow for user interaction of the light.

To create a lamp:

1. Create the lamp in Maya, as you would any model of furniture (including collision and textures).
2. Click or select Home > Create Light Locator.
3. Position the light locator to determine the exact location of the point light source:



> There is a limit of one light locator per lamp but there must be one light locator so that the model exports as a lamp. The offset and light mask texture for the light locator is displayed in the Object Editor under Components > light > lights > light(0).

4. The texture used by the mask for the point light is displayed in the Attribute Editor (the default mask is a solid white texture).

   To mask particular areas of the light source, change the texture to a cube map (for example, when the model includes a lamp-shade you may want the light to only illuminate through the gaps in the lamp-shade)

   To change the color of the light, simply change the white within the mask to a color.

> The default values for the point light in a lamp are Attenuation Start = 0, Attenuation End = 2.5 and Attenuation Power = 2. For more information on point lights in Home, see Adding Dynamic Lighting.

5. Export the lamp furniture object, ensuring that you check the Update Object Catalog box. For more information on the export process, see Validating and Exporting Furniture.

Furniture Block System

For Japanese Live Page
最新の日本語版ページはこちら 「家具ブロックシステム」

Overview

A standard furniture item is an object with a Furniture component, but no Active Item component. An 'active item' is an object with both a Furniture and Active Item component (see Active Items).

The budget for placing furniture in a space is presented to users as consisting of 100 "blocks". Each standard furniture item takes up one block. Active items can use between 1 and 100 blocks depending on their complexity, this value is set using the Object Editor. All active items created before HDK 1.65 consume 22 blocks.

When users place items of furniture, the user interface shows them:

- How many blocks they have already used
- How many blocks are still available
- How many blocks the selected item in the catalogue requires
- Whether the object will fit in the Apartment or Clubhouse



If users remove a furniture object, the user interface tells them how many blocks will be released by removing it. The number of blocks used by a furniture item is also reported to the user at the point of purchase.

> Remote users only see that an item has been added, moved or removed when the local user has confirmed the action.

Handling Fragmentation

When users place and remove furniture, furniture memory can become fragmented. For example, there may be 22 free blocks, but if the required memory is not consecutive, the user cannot place an active item in the space. The PS Home client handles fragmention by reloading the layout of furniture in the space, which defragments the furniture memory.

Active Item Resource Usage

The resources an Active item uses can vary depending on the type of component you create. You can create Actives so that users could, potentially, add up to 50 Active items with a small memory budget, or just a handful of high-memory interactive items.

> Active items with lower memory usage are more appealing to users as it means they can place more items.

When you have created your Active items, you must profile them in the Client so that they can be accurately rated for the resources they use. See:

- Determining Active Item Resource Usage
- Optimizing Actives Memory Usage

See also:

- Furniture Validations - summarizes both HDK automatic validations and manual validations for furniture
- Active Item Validations - summarizes the automatic and manual validations that are required or recommended for active items
- Object Editor Validations - summarizes the HDK automatic validations and manual validations for the Object Editor

Determining Active Item Resource Usage

For Japanese Live Page
最新の日本語版ページはこちら 「アドバンスド家具のリソース使用量の決定」

The amount of system resource an Active Item uses determines how many blocks of the furniture budget it consumes (out of a maximum of 100). You must declare the resource usage of an Active Item before it can be packaged.

How Resource Usage Relates to Furniture Blocks

Active Item resource usage is rated in the following areas:

- Main Memory
- Host Memory
- VRAM
- PPU Time
- Network Usage

Each value is specified in its own custom units. The following table lists how much resource is allocated for each resource type:

Resource Type	Resource Usage
Main	48 KB per block
Host	112 KB per block
VRAM	304 KB per block
PPU (while object is not in use)	0.1 ms per block
PPU (while object is in use)	8 ms
Network (while object is not in use)	0.1 kbytes/s per block
Network (while object is in use)	2 kbytes/s

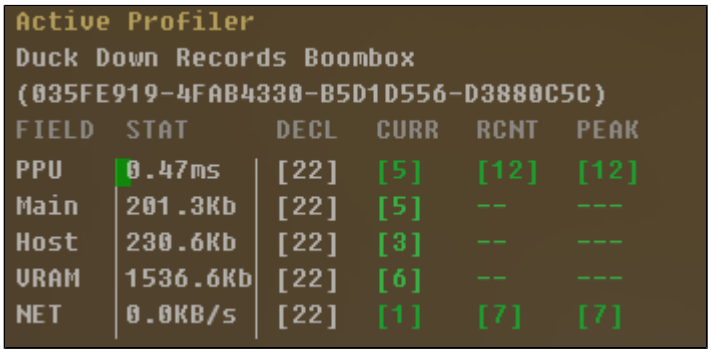
For example, a value of '2' for VRAM means that the Active item does not use more than 608 KB of VRAM.

Resource allocations for memory (Main, Host and VRAM) are strictly enforced; if the item uses more memory than has been allocated it will stop working.

Resource allocations for PPU and Network use are not strictly enforced; the occasional spike in usage is acceptable.

The Active Profiler

The Active Profiler tells you how many furniture blocks are required to properly accommodate an Active Item. You can access it using the console command `activeprofile`.



The profiler displays a table with the following columns:

Column Heading	Description
FIELD	The type of resource usage being measured

STAT	The current reading in its native format
DECL	The number of blocks currently declared in the Object Editor
CURR	The current number of blocks required by the object for the resource type
RCNT	The highest recent (within the last 10 seconds) peak value
PEAK	The highest recorded number of blocks required by the object since profiling began

The CURR values are used when reporting PPU and Network usage, the PEAK values are used when reporting memory (Main, Host and VRAM) usage.

Determining Resource Usage

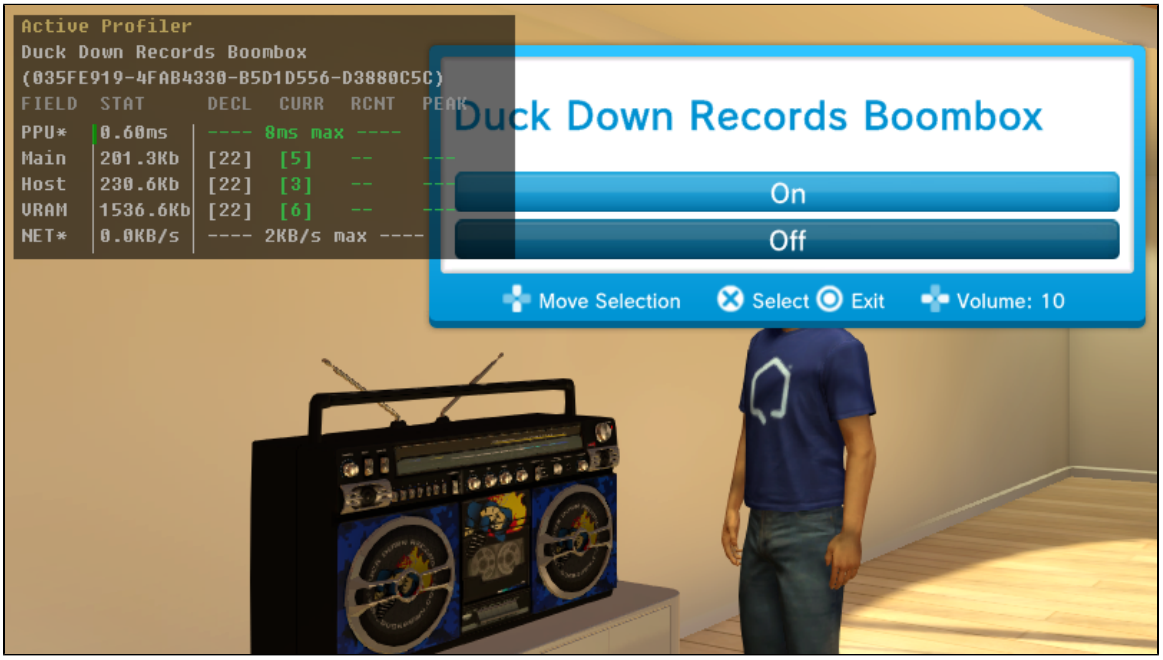
To determine resource usage values for an active item:

1. Launch the PS Home client in online mode (see Testing Content in Online Mode) and use the navigator to go to a Personal Space.
2. Use the Dev Debug Console command `activeprofile`.
3. Ensure that your Active Item is in your inventory (see Managing the User Inventory) and place the item in your scene.
4. While the item is not being used, make a note of the typical values shown for PPU time (PPU) and network usage (NET). Occasional spikes in these values are acceptable, the value recorded should be the highest value that frequently occurs in the CURR column. If the item includes features which run when the item is not being used (such as playing music), enable these features, then exit the activity before recording the PPU and Network usage values.



5. Join the Active Item and perform a full load test of the item, testing all the available functionality. If the Active Item supports networking, test the item with as many users as it supports. Note down the peak values recorded for Main, Host and VRAM. While testing, also ensure that the values for PPU and NET do not frequently exceed the stated maximum

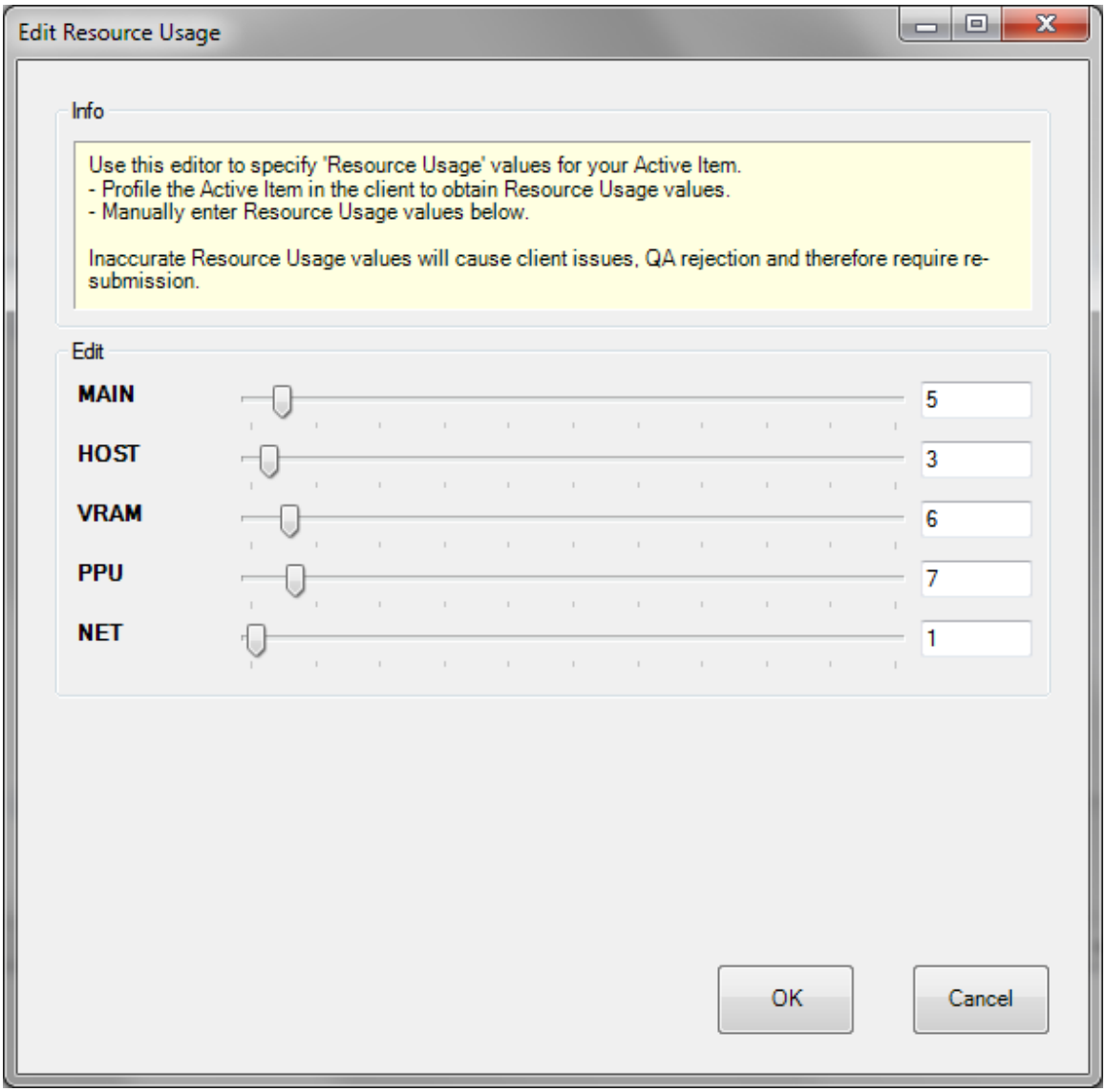
values (again, occasional spikes are acceptable).



Specifying Resource Usage Values

To specify resource usage values for an Active Item:

1. In the Object Editor, select the Active Item that you wish to set the resource usage for.
2. Select Object > Edit Resource Usage.
3. Enter the five recorded values into the relevant fields.



4. Click OK to save your changes.

> Once the resource usage values have been entered, the object should be fully tested. If the object no longer functions correctly, it is likely that the resource usage values have been set too low.

> A value of '0' or a blank field means 'undefined'. An object with undefined values will not pass validation and cannot be packaged.

See also:

- Profile GUI
- Profiling Objects

Optimizing Actives Memory Usage

> For Japanese Live Page
> 最新の日本語版ページはこちら 「アドバンスド家具のメモリ使用量の最適化」

There are certain actions you can take to help optimize memory usage and so make your active furniture item more attractive to users. For example:

- To reduce run-time costs, compile all scripts and make use of the Environment library.
- To avoid getting a high (inefficient) slot rating from the profiler, make sure that your item uses resources in a balanced way.

For example, if an item uses a small amount of Main memory, a small amount of Host memory, but a very large amount of VRAM, it is given quite a high slot rating. To reduce the slot rating, you could try to reducing the amount of VRAM used, perhaps increasing mesh complexity to compensate. This might greatly reduce the slot usage without having to compromise on item quality.

> ⚠ Avoid making changes that are likely to affect item quality. For example, reducing the texture resolution may be feasible from a design point of view, but is not desirable from the artist's point of view.

Object Editor

The Object Editor is a tool for creating, editing and packaging object definitions that are loaded by PS Home. Using the Object Editor, you can compile, edit and manage new and existing objects and their assets and components.

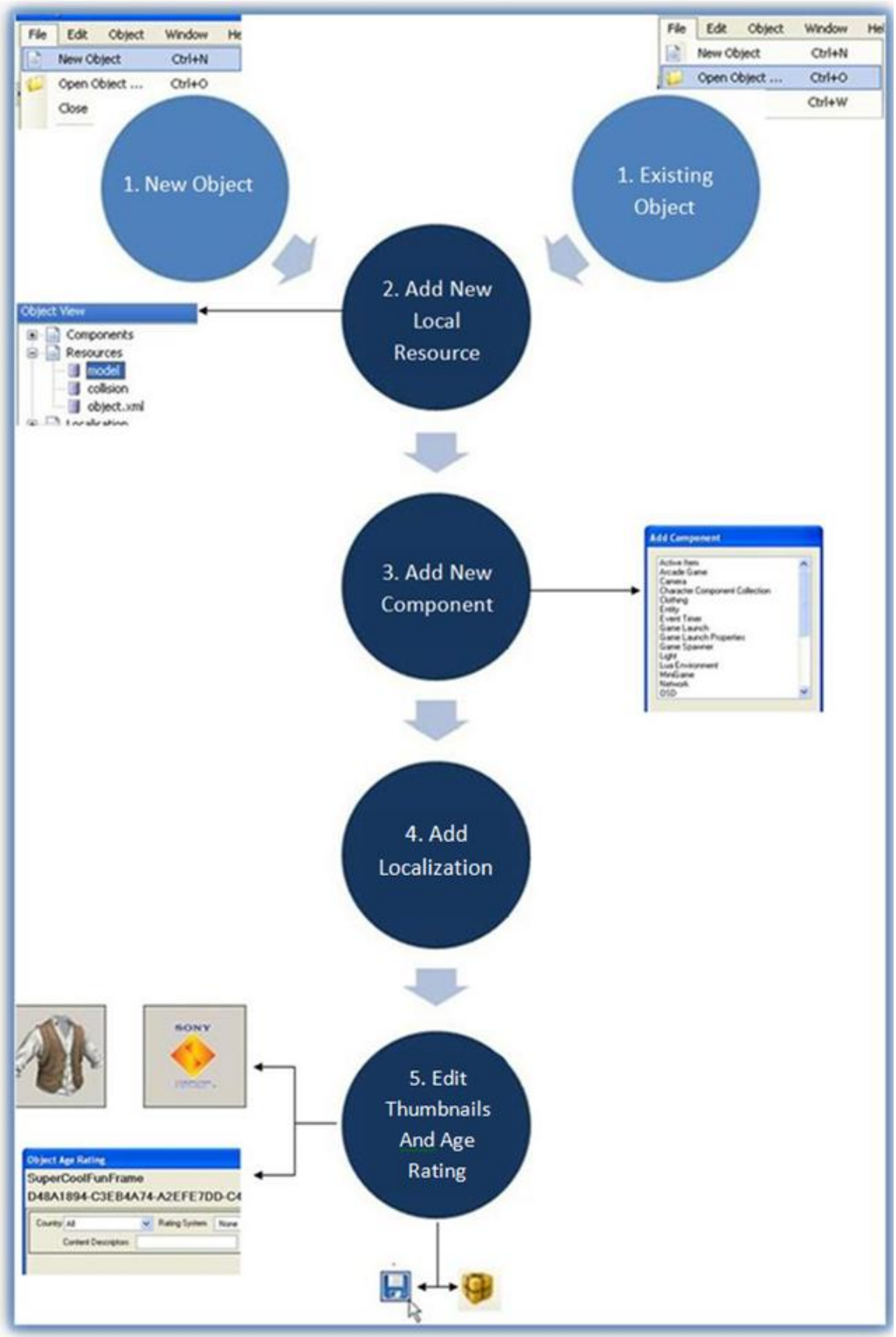
The Object Editor works as an HDK-specific XML editor that enables:

1. Easy management and modification of objects that have been authored, validated and successfully exported from the Home DCC Tools.
2. Creation of object types not exported using the HDK authoring tools (such as the Home DCC Tools).

Object Creation Workflow

The following process diagram shows a typical sequence of tasks for creating a new object in the Object Editor. Although the steps are ordered sequentially, they do not necessarily reflect the exact order in which you do things.

An explanation of the processes follows the diagram.



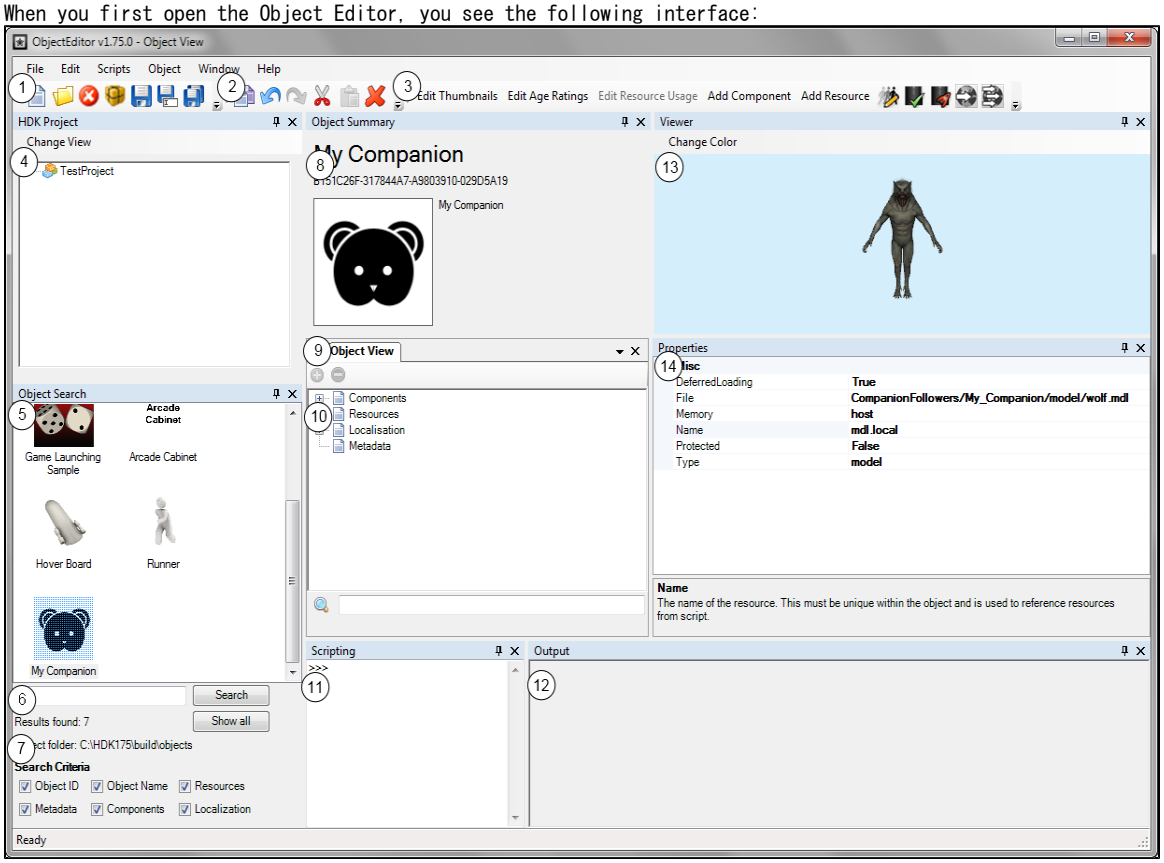
⚠ Save your object at each stage in the process of using this tool.

The following table summarizes what you do at each stage of object creation:

	Task	Description
1.	Create a new object/ Add an existing object	Create a new object, or add an existing object (made using DCC tools) to edit in the Object Editor.
2.	Add New Local Resource	Add the resources contained in your <HDK_INSTALL_BUILD> folder that you want to use with your object. For example, add a .lua file as a local resource for a mini-game object. Repeat this step for as many resources as you want to use with the object.
3.	Add New Component	Add components to define how your object will behave in PS Home. For example, if you added a .lua resource in step 2 that uses the OSD library, add an OSD component so that your object can use and access OSD.
4.	Add Localization	Add the object's information in different languages.
5.	Edit Thumbnails and Age Rating	Add thumbnails and age rating information. These items are required to successfully package your object.
6.	Save and/or package the object	Save your object frequently. When you are ready to send it to QA, package it.

Object Editor Interface

When you first open the Object Editor, you see the following interface:

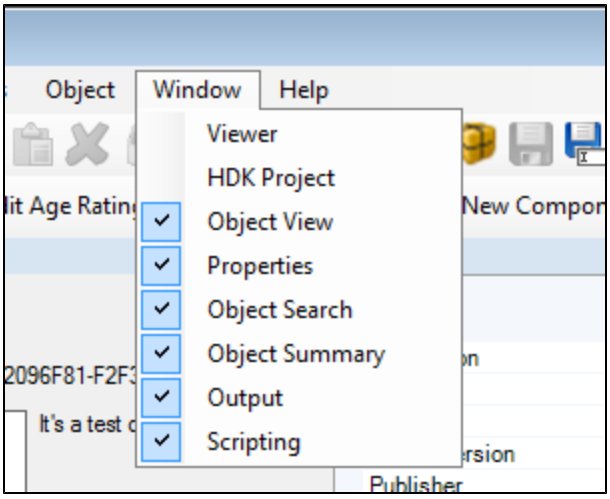


Item Number	Name	Description
1.	Quick action buttons	Actions are: Save, Create New Object, Open Object, Package Object, and Delete Object.

2.	Quick action edit buttons	These buttons are Copy, Undo, Redo Undone Action, Cut, Paste, Delete, Lock From Editing and Unlock To Allow Editing.
3.	Quick action edit object buttons	Allows you to edit thumbnails, edit age ratings, edit an object's dynamic resource usage, add a new component, add a new local resource, and edit an object's repertoires. Also allows quick testing of the selected object on your PS3 and importing localization information for the current object, or for all objects in your build folder.
4.	HDK Project panel	Displays the currently loaded HDK Project. For more information on this panel, see HDK Project Panel.
5.	Object Search panel	Displays a welcome message when first opened. When you enter a search or click Show all, the selected objects, or all the objects in your <HDK_BUILD_DIR>/build/objects directory are displayed.
6.	Search field and button	Enter search terms in the field and click Search, or click Show all.
7.	Search filter boxes	Enter search terms in the field and click Search, or click Show all.
8.	Object Summary panel	This panel displays the object name, object ID, description, and the 128x128 thumbnail. If an object is not localized, the Object Summary panel displays a warning, like in the image below:  For more information on localizing objects, see Localizing Objects.
9.	Add Item and Remove Item buttons	Many object components can contain sub items. These buttons allow you to add/ remove sub items to the currently selected object. For example, if you add a Scene Object component, then select one of the Scene Object parameter lists (String, Boolean, Numeric, Enum) in the Object View panel, you can use the Add Item button to add a new parameter to that parameter list. Conversely, selecting the newly added parameter and then pressing the Remove Item button will remove that parameter.
10.	Object View panel	Displays the Components, Resources, Localization, and Metadata nodes for the selected object.
11.	Scripting panel	Allows HDK API script commands to be executed. For more information see, HDK API Scripting.
12.	Output panel	Displays the results when you package and save your object.
13.	Viewer panel	Provides a view of the currently selected resource. For more information see, Viewer Panel
14.	Properties panel	Displays a description of the item you selected.

Showing, Hiding and Moving Panels in the Object Editor

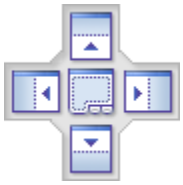
You can show and hide panels by using the Window menu:



If the name of the panel has a check next it, it's displayed in the Object Editor. Uncheck the name of the panel to hide it.

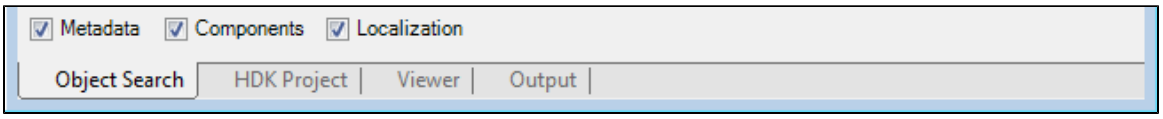
You can attach panels to the main Object Editor interface, have them as standalone windows or as tabs in the same dialog:

- To detach a panel and have it as a standalone window, place your cursor over the title of the panel and drag it out, away from the interface or any other panel it's attached to. Alternatively, double-click on the panel's title.
- To attach a panel, drag it over the interface or the panel that you want to attach it to. Alternatively, double-click on the panel's title.
- To move a panel to a particular position on the interface, start dragging the panel until the following in displayed:



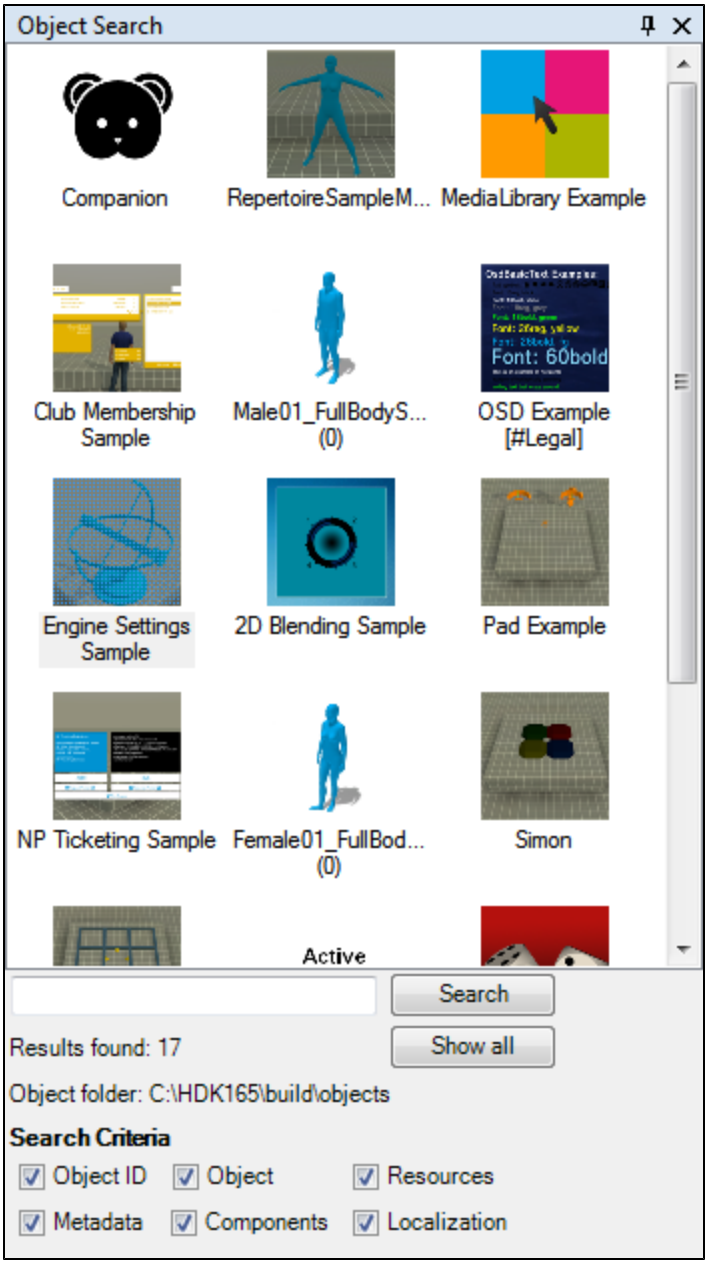
Keep dragging the cursor until it's over the position you want indicated on the above control, then drop the panel into that position.

If you attach a panel to another panel, you can view the panels using the tabs:



Object Search Panel

The Object Search panel allows you to search and browse for objects in your \build\objects folder.



You can search using any of the following criteria:

- Object ID
- Object Name
- Resources
- Metadata
- Components
- Localization

To search for an object, enter all or part of one of the above things and click Search.

To show all objects, click Show all.

By default, all criteria are included in the search. To exclude a criterion from the search, deselect the corresponding checkbox under Search Criteria.

Click on an object thumbnail to open it for editing. To edit multiple objects at once, hold Ctrl or Shift when selecting the objects.

> Not all features are available when multiple objects are selected. Unavailable features are grayed out.

Object Summary Panel

The Object Summary panel displays information about the currently selected object(s).



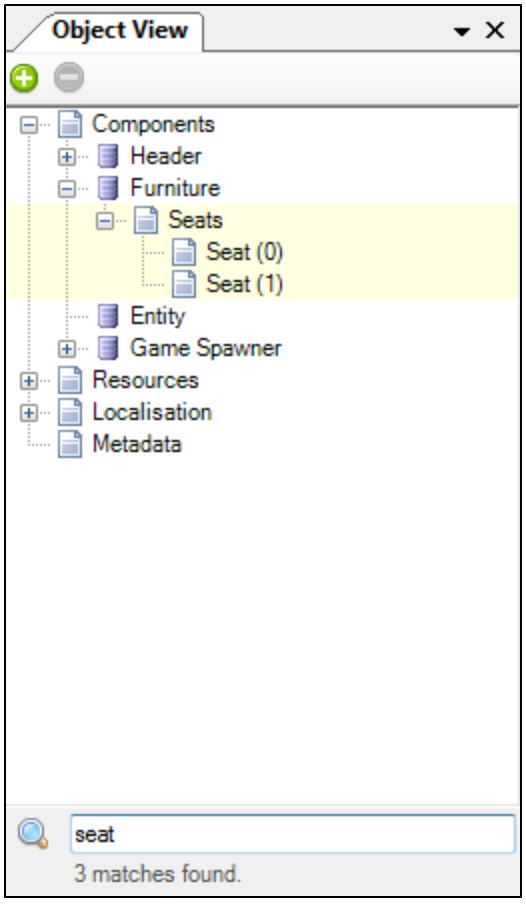
The following items are displayed:

- Object Name
- Object ID
- Object Thumbnail
- Object Description

For information on how to change these items, see Completing the Object Header, Overwriting and Copying Objects and Preparing Objects for Packaging.

Object View Panel

The Object View panel displays the full contents of the currently selected object(s).



The content is grouped into four categories:

- Components: The building blocks of an object
- Resources: External files used by the object
- Localisation: Text strings in multiple languages
- Metadata: Information about the object

The settings of components, resources and metadata can be viewed and configured in the Properties Panel after clicking on them.

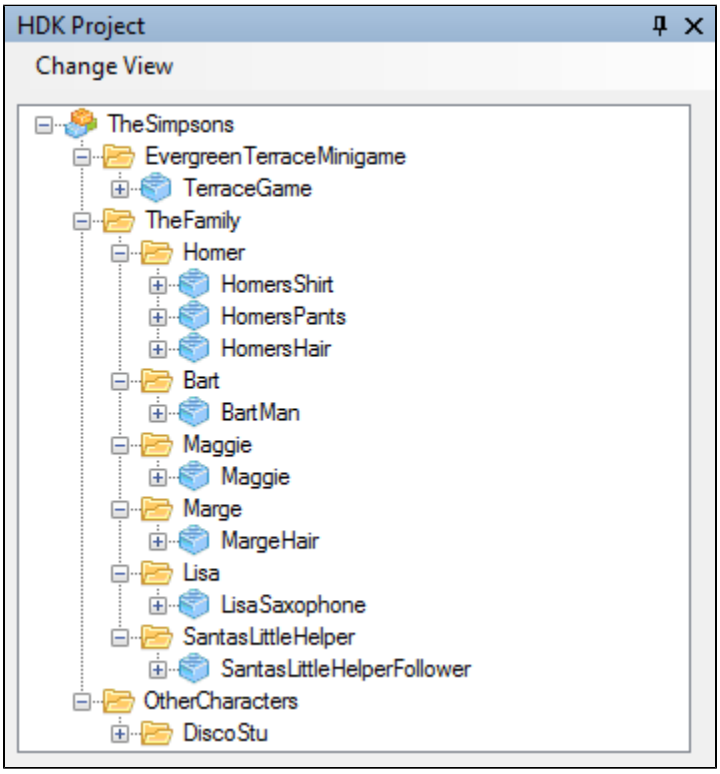
Localization entries can be viewed in the Properties Panel; to edit them see Localizing Objects. To change the default display language for objects in the Object Editor, click on the root node of the Localisation section.

The two buttons at the top of the Object View panel allow adding and removing of component sub items. In the example screenshot above the Add Item button can be used to add seats to the Furniture component as the Seats list is selected.

The search box at the bottom can be used to filter the content of the Object View panel. This is useful for finding a particular resource.

HDK Project Panel

The HDK Project panel allows you to browse and edit the currently open Project (*.hdkproj) file.



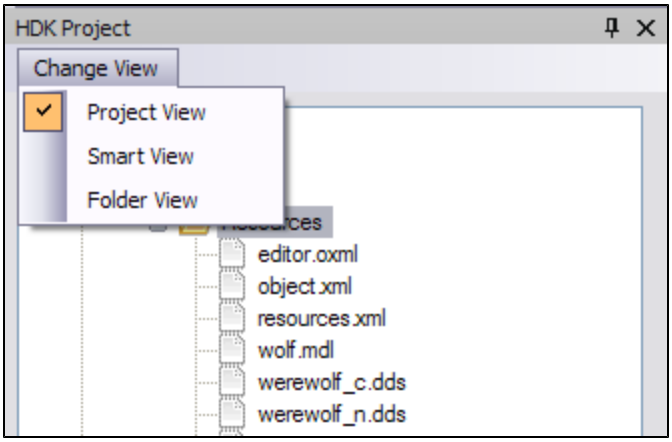
HDK Project Views

The HDK Project panel has the following views:

- Project View: Displays your objects in a hierarchical structure. This is the default view.
- Smart View: Displays your objects in a hierarchical structure and categorizes the object resources. You cannot edit content in this view.
- Folder View: Displays the contents of your entire project as a folder view. You cannot edit content in this view.

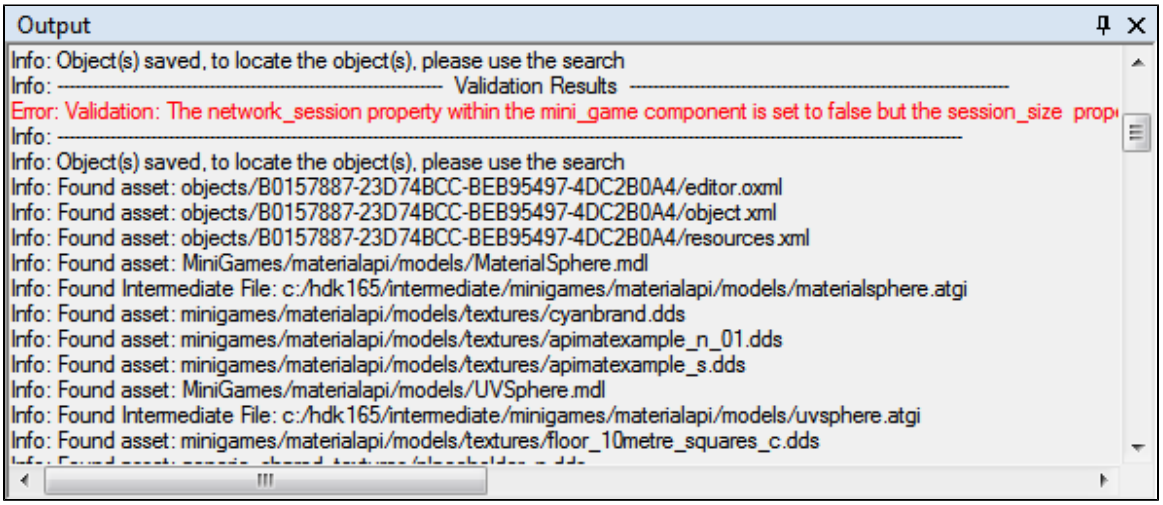
Changing the Project View

To change the project view, click Change View on the HDK Project panel and select the view you require.



Output Panel

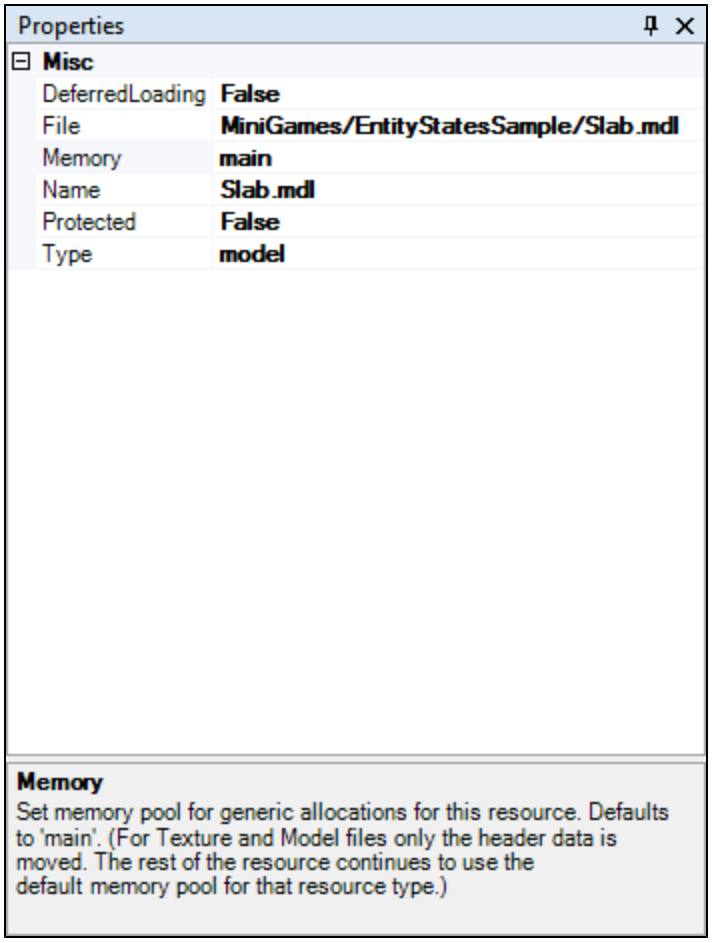
The Output panel displays information, errors and warnings during operations such as saving, packaging and validation.



Text from this panel can be copied and pasted into a text editor for searching or saving.

Properties Panel

The Properties panel allows you to view and edit items that have been selected in the Object View Panel.



The contents of the panel depends on what has been selected. Click on the name of a property to show contextual help for the property at the bottom of the panel.

> Contextual help is not available for all properties.

Scripting Panel

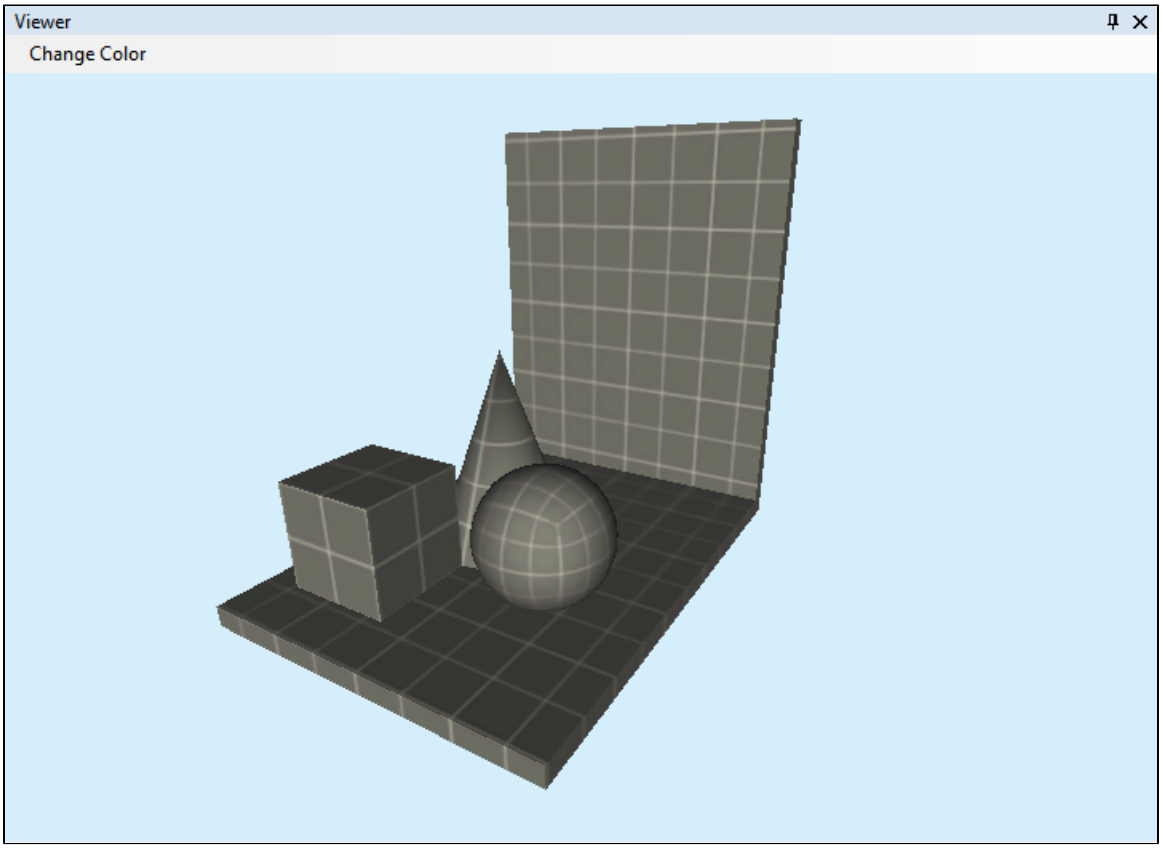
The Scripting panel displays the Object Editor's Python terminal.



This allows for HDK API scripts to be run either inline or from a file. For information on HDK API scripting, see HDK API Scripting.

Viewer Panel

The Viewer panel displays a preview of the most-recently selected resource in the Object View Panel.

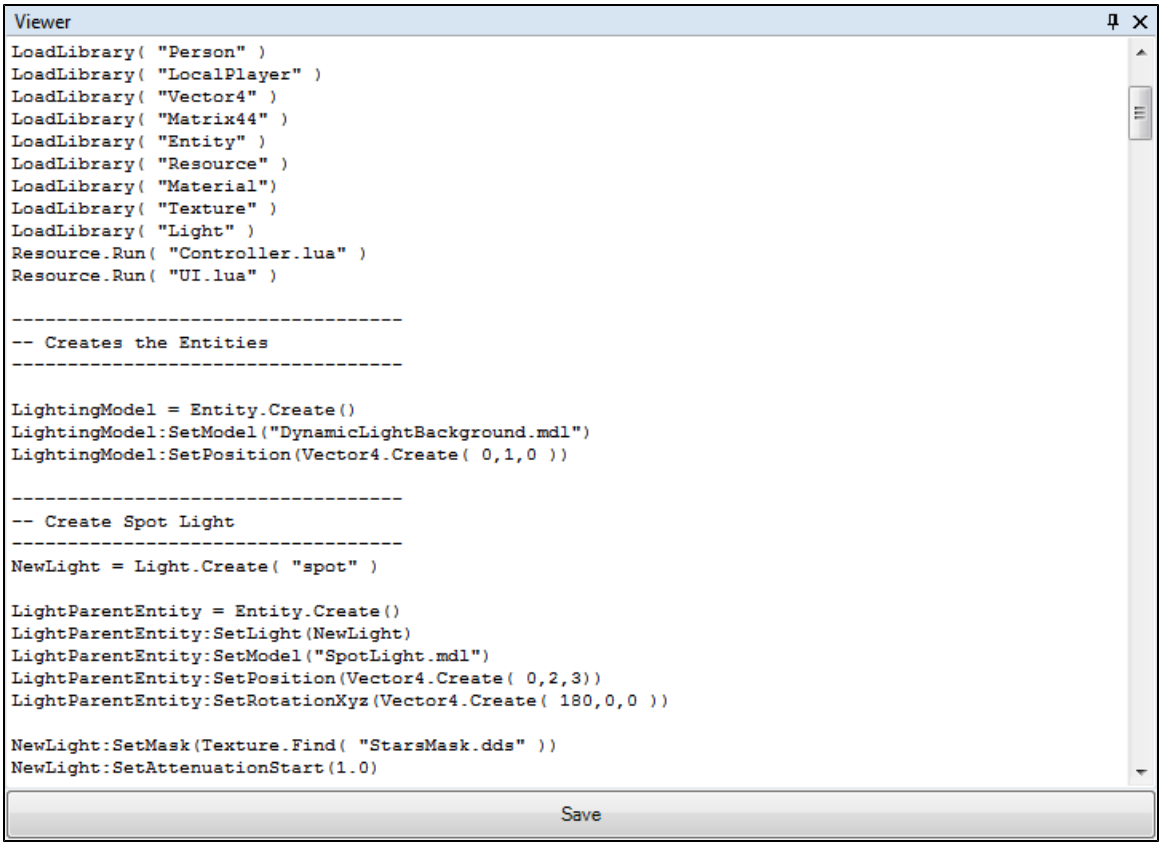


The Viewer can display models (*.*mdl), textures (.dds) and text files (e.g. Lua, XML etc).

When viewing models:

- Hold the left mouse button to rotate the model
- Hold the right mouse button to zoom
- Hold the middle mouse button to pan

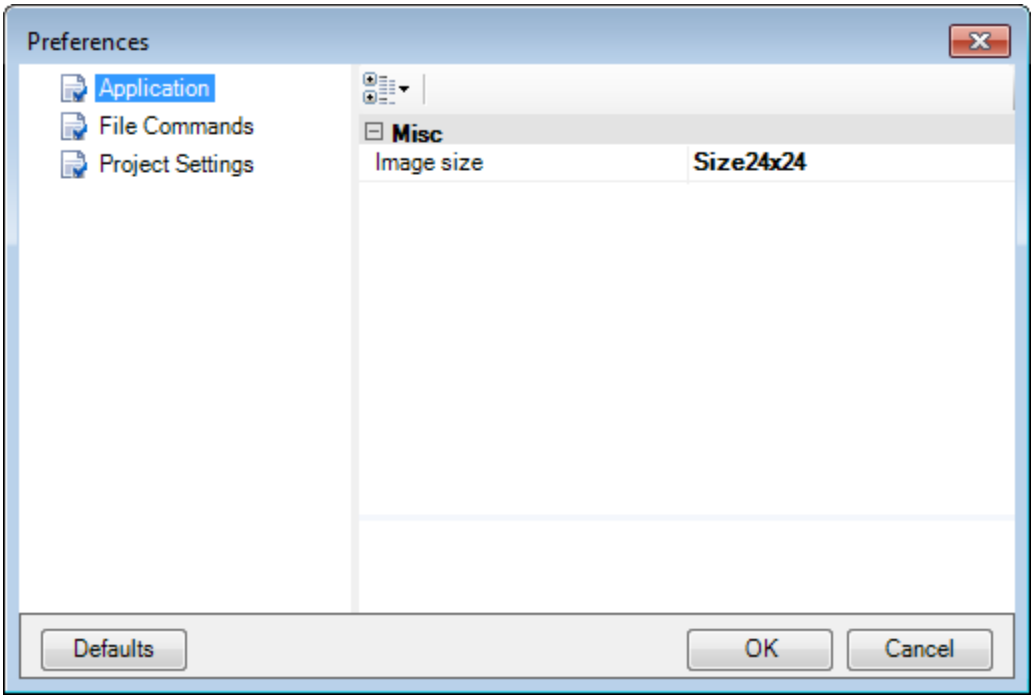
When viewing text files, you can make edits and then save the changes by clicking the Save button.



Setting Object Editor Preferences

To set your Object Editor preferences:

1. Select Edit > Preferences to display the Preferences dialog:



2. Update your preferences as required, by clicking on the area on the left side of the Preferences dialog, then making the changes needed on the right side of the dialog.

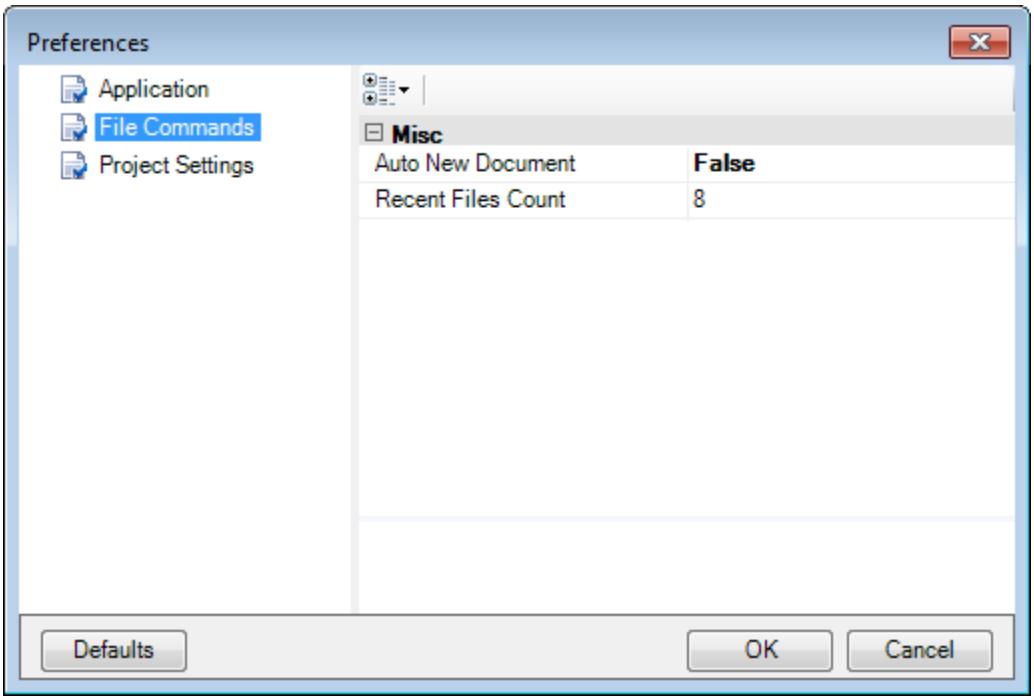
If at any point you want to return any changed values to the installation defaults, click the Defaults button.

You can change the following aspects of the Object Editor:

- Application

  Select the image size for the Object Editor toolbars from the Image size drop-down list.

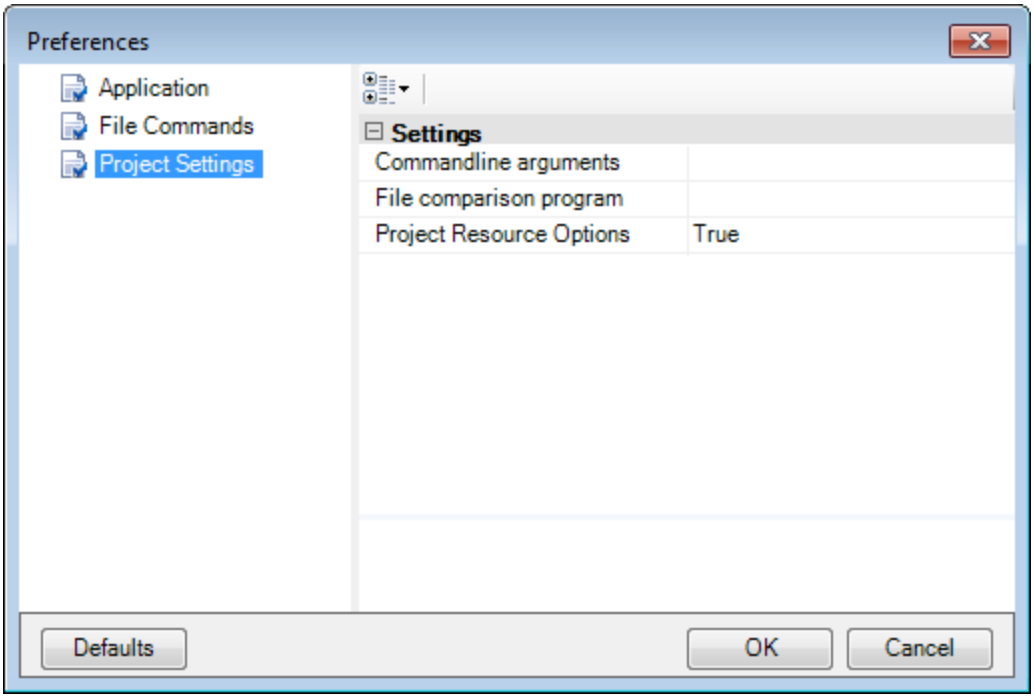
- File Commands



  To change the File Commands preferences:

  a. If you want the Object Editor to create a new .hdkproj file if an existing file has not been opened, select True from the Auto New Document drop-down list.
  b. Enter the number of recent file names saved by the Object Editor in the Recent Files Count field.

- Project Settings



To change the Project Settings preferences:

a. Specify the command line argument for your comparison program, for example, "%1" "%2" in the Commandline arguments field.
b. Specify the location of your file comparison program in the File comparison program field. For example, for Beyond Compare it is C:\Program Files (x86)\Beyond Compare 2\BC2.exe.
c. Specify if resources should be added when you drag an object to the project in the Project Resource Options field. The default value is True because some objects are extremely large and it takes time to find all of their subsequent resources.

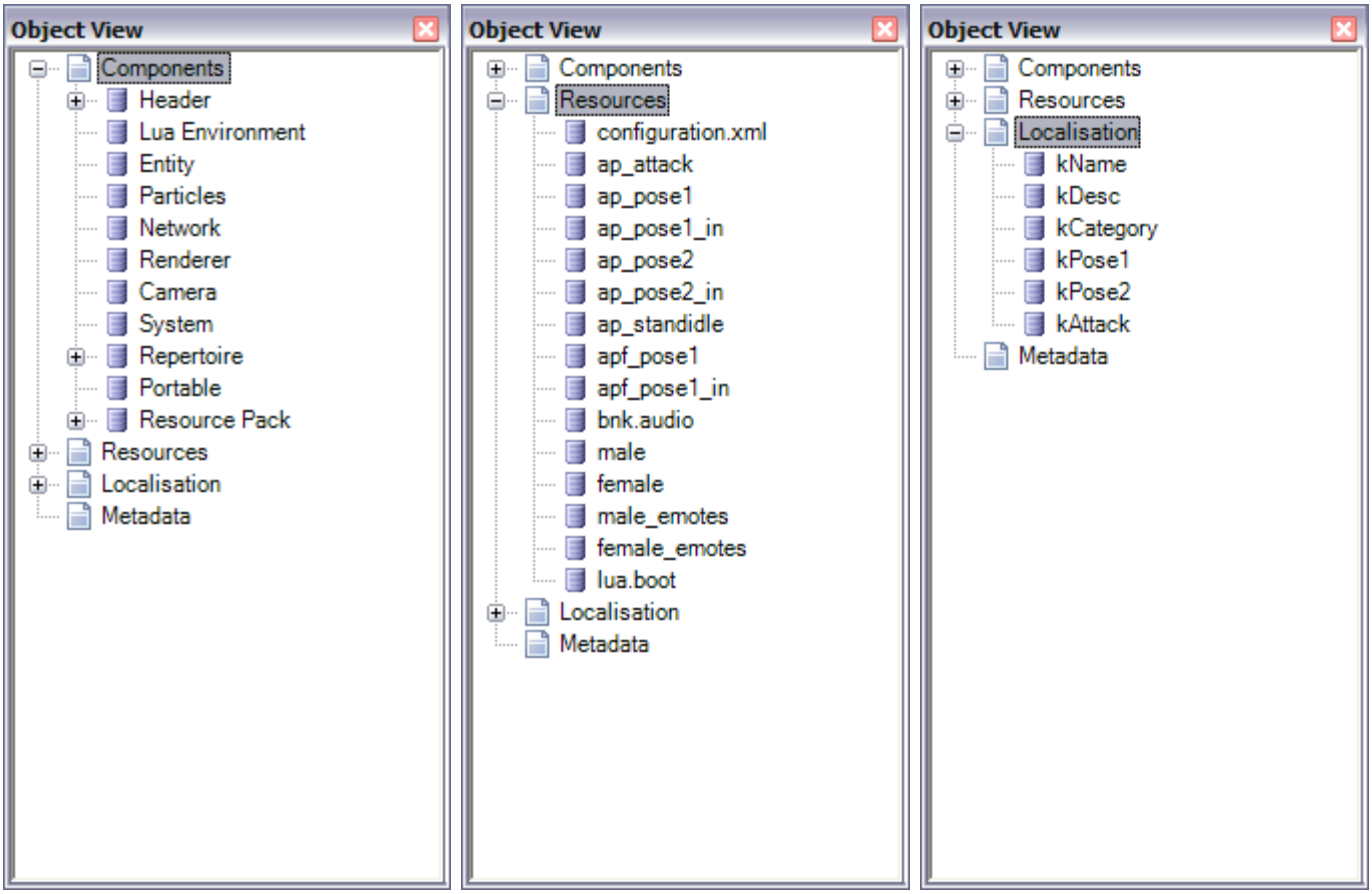
3. Click OK to save your changes.

Object Structure and Files

An object comprises: a header, components, resources, localization and metadata:

Composite	Description
Header (default and mandatory)	Contains identity tags such as author, description, display name, object version and contextual help text. See Completing the Object Header.
Components	For example, Furniture, Lua Environment, Targetable, Scene Object, Entity, Particles, Network, Event Timer, OSD, Rig, Clothing, Arcade Game, MiniGame, Screen, Pad, Renderer, Camera. See Working with Object Components.
Resources	Asset files, including textures, for example, .dds, .bnk, .xml, .ddl, .lua. See Adding and Deleting Object Resources.
Localization	If there is regional variation in objects, such as language, pricing, branding, this is where it is specified and described. See Localizing Objects.
Metadata	Information that describes the object, including information found in the header, localization information, and terms for searching and category classification. See Object Metadata.

These main composites of an object are visible in a folder structure in the Object View:



Objects are stored as a folder that is named by the unique ID of the object (for example, 0000000000000011). The folder contains the following files:

- object.xml: Describes the object.
- resources.xml: Lists any resource files.
- localisation.xml: Lists all localizable strings.
- Any resource files: For example, .mdl, .lua, .hkx, particle.xml.

When packaged with the Object Editor, these files are combined into an archive that can be uploaded to the server and submitted to the quality assurance process for PS Home.

Object Files

When you create an object, either through exporting geometry from Maya, or by creating a new object in the Object Editor, several files are automatically created that the Object Editor needs to manage the object. They are:

- catalogueentry.xml
- editor.xml
- localisation.xml (list of localization information for the object)
- metadata.xml
- object.odc
- object.xml (describes the object)
- resources.xml (lists any resource files)

When you export objects from Maya, a set of intermediate ATGI format files are created. The minimum intermediate files that the Object Editor needs are:

- .hkx
- .hkx
- .xml
- .atgi

As you add components, resources, and other information (for example, age rating) to the object, the appropriate files are updated. For example, updating the object localization updates the localisation.xml file.

Changing Object Types

To change an object type, for example from 'hair' to 'hat', you must change the object type wherever the type is defined (in both

a component and in the metadata):

- For furniture items, change the type in the Furniture component and metadata.
- For clothing items, change the type in the Clothing component and metadata.

Working with HDK Project Files

For Japanese Live Page
最新の日本語版ページはこちら 「HDKプロジェクトファイルでの作業」

HDK projects allow you to organize the objects that you are working on effectively. You can send your project file to someone else to work on, or work on a project file sent to you by someone else.

HDK project files are stored as the .hdkproj file type.

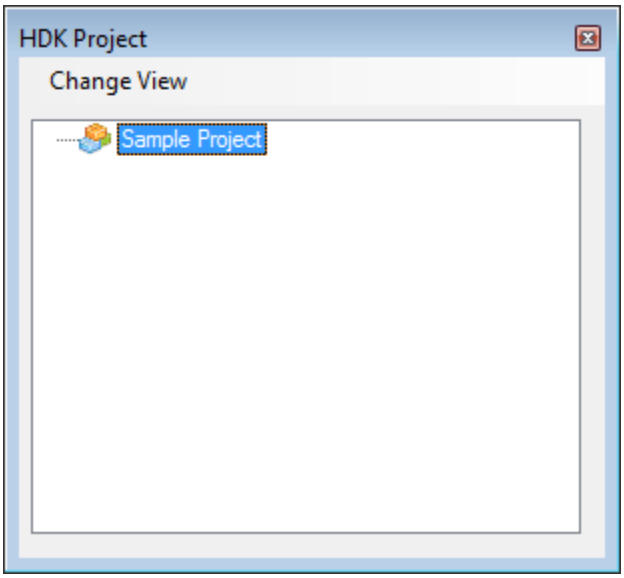
Note:

- Pre HDK 1.70, .hdkproj files also had a paired .hdkproj.computername.local file, these files are no longer needed in HDK 1.70
- The ability to organize resources into groups feature has been removed.
- Resources are no longer loaded at project load time, you need to click Show Resources in order to show the resources.

Creating a New Project

To create an HDK Project in the Object Editor:

1. Select File > New Project.
2. In the Save As dialog, enter a name for the project.
3. Click Save. The project is displayed in the HDK Project panel.



Once you've created your project, you can add groups and objects to them. For more information, see Organizing Objects Using HDK Projects.

Saving a Project

To save a project in the Object Editor, select File > Save Project. Alternatively:

1. Select File > Save Project As
2. In the Save As dialog, enter the new name for the project
3. Click Save

Zipping a Project File

You can zip your project files and send them to someone else, for example, another developer.

To zip a project file:

1. In the HDK Project panel, right-click on the project and select Zip up all assets from the pop-up menu displayed.
2. In the Save As dialog, enter a name for the .zip file.
3. Click Save.

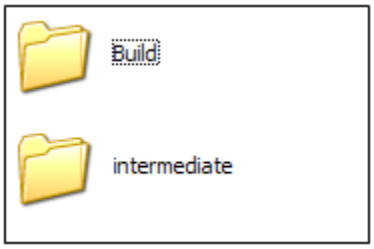
> You can also zip a single object or group of objects by right-clicking and zipping at the project level that you require.

Opening a Zipped Project File

To open a zipped project file:

1. Unzip the archived project file to your preferred location on your computer.
2. Navigate to the build folder.
3. Copy the build folder and any other folders at the same level as the build folder, and paste the folders into the root of your HDK installation directory, i.e. <HDK_INSTALL_DIR>. Overwrite the existing folders.

   In the following example, you copy the build and the intermediate folder.



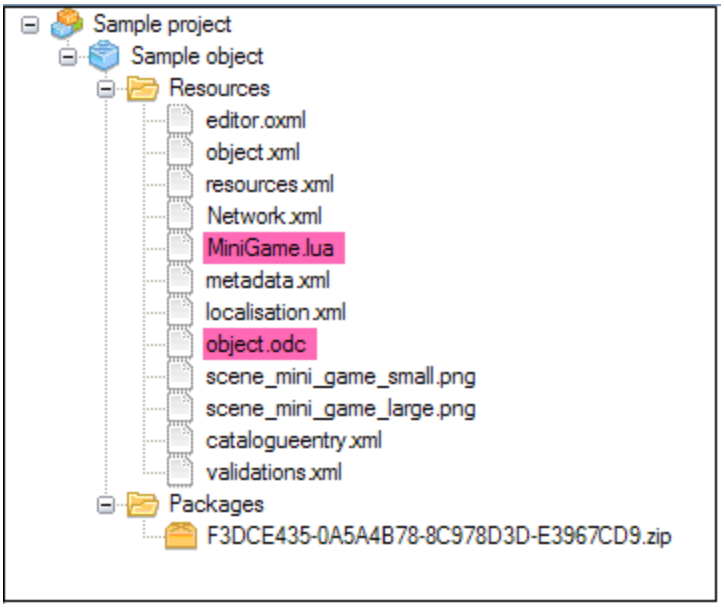
4. Copy the .hdkproj file that you received in the zip (depending on the sender's setup, you might have to search for the .hdkproj file). Paste the file to a location of your choice and open it from the Object Editor.
5. In the HDK Project panel, expand the project. You will notice that the object has no resources.
6. To display the object resources, follow the procedure in Recovering Object Resource Files.
7. Select File > Save Project to save the project.

Comparing a Packaged Object with the Copy in your Object Directory

If changes are made to object files after the object is packaged, you can run a comparison between the object files in the package and the object files in your directory.

To compare a packaged object with the copy in your object directory:

1. In the Object Editor, display the HDK Project panel and navigate to the object you are interested in.
2. Package the object.
3. Expand the object's Packages folder in the HDK Project panel to view the packaged object.
4. Right-click on the required .zip file and select Diff package against disk from the pop-up menu displayed.
5. Expand the object's Resources folder to view the resources that were modified after the object was packaged. The modified objects are highlighted, for example:



6. right-click on the highlighted files and select Compare against package from the pop-up menu displayed.

   If you have set up a comparison program for use with the Object Editor, it will open and show the differences between the two files. If you have not set up a comparison program, you are prompted to do so, see Setting Object Editor Preferences for more information on setting up your comparison program.

Reloading a Project

You might want to reload a project if you have one of the following scenarios:

- You manually edited the content of a project outside the Object Editor and wish to display the changes made in the Object Editor.
- You want to cancel the changes you made since the last time you saved the project.

To reload a project:

1. In the HDK Project panel, right-click on the project you require, and select Revert to Saved from the pop-up menu displayed. A warning message is displayed.
2. Click OK. The saved project data is reloaded from disk.

Remembering the last open Project

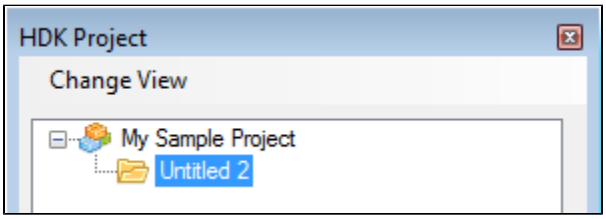
If you have a project open and you close the editor, the editor will load that project when you restart the application. You can close a project by clicking: File->Close Project

Organizing Objects Using HDK Projects

Adding a Group to a Project

To add a group to a project in the Object Editor:

1. Open the project you want to work on.
2. In the HDK Project panel, right-click on the project or existing group that you want to add the group to and select Add Group from the pop-up menu displayed. A folder representing the new group is displayed:



3. Right-click on the new group, select Rename and enter a name for the group.

You can create groups within groups to achieve a file structure and drag and drop a group into another group.

Deleting a Group from a Project

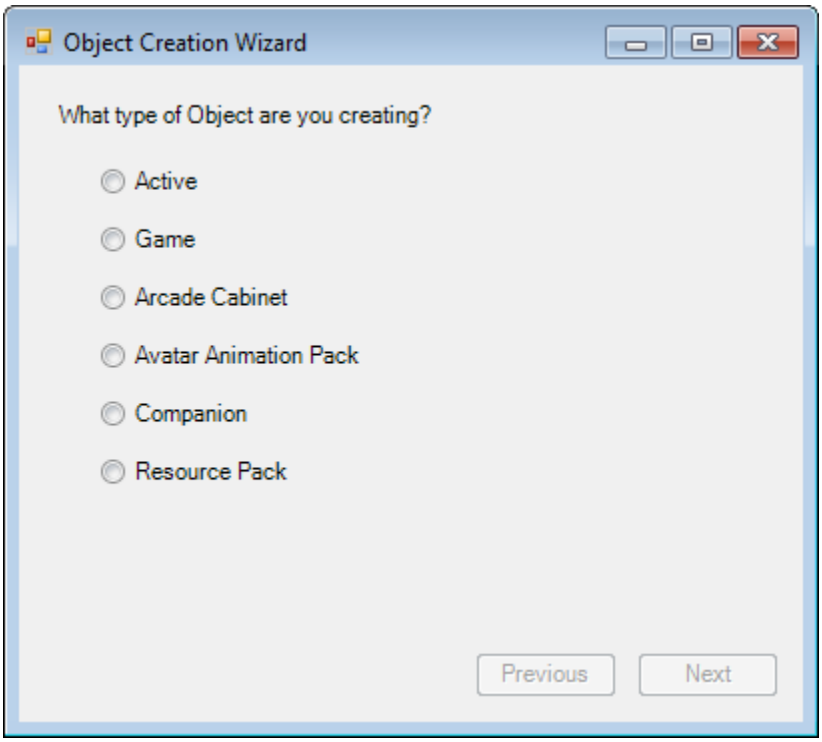
To delete a group from a project, right-click on the group in the HDK Project panel and select Remove from Project.

Deleting a group removes all object references, Some groups are read-only.

Adding an New Object to a Project

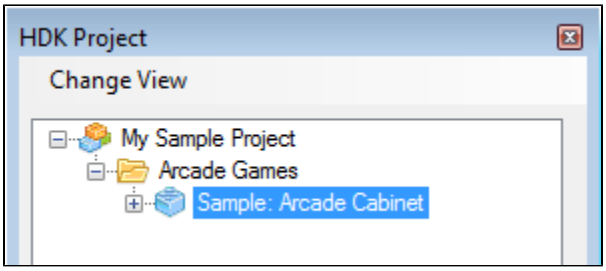
To add a new object to a project in the Object Editor:

1. Right-click on the project or group that you want to add the object to and select Create new object here from the pop-up menu displayed. The Object Creation Wizard is displayed:



2. Follow the steps in Creating a New Object.

Once an object has been added, it is displayed in the HDK Project panel, for example:



Adding an Existing Object to a Project

To add an existing object to a project you can either:

- Drag and Drop the object from the Object Search panel into the appropriate group.
- Right click in the Project View and click "Browse & Add" this will pop open an Object Search, from here you can find the

objects you wish to add to the project.

Removing an Object from a Project

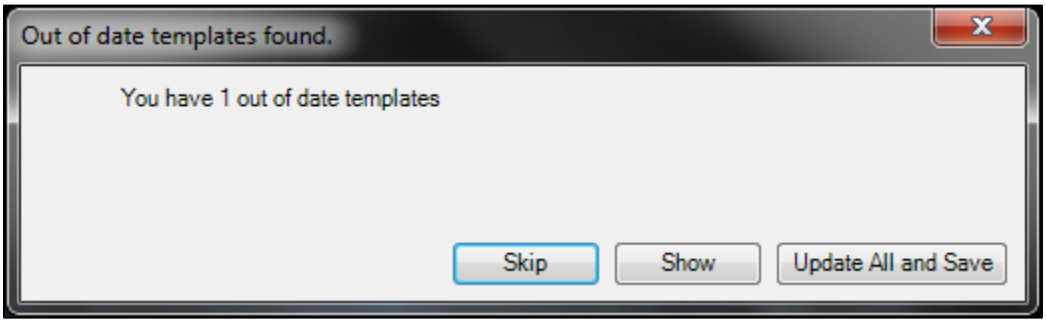
To delete an object from a project, right-click on the object in the HDK Project panel and select Remove from Project from the pop-up menu displayed. The object is removed from the project but isn't deleted.

Viewing the Location of an Object or Resource in the Build Folder

To view the location of an object or resource in the build directory, right-click on the object or resource and select Show in Explorer from the pop-up menu displayed. Windows Explorer opens and displays the object or resource that you selected.

Working with Templates in Projects

When you load a project with x number of objects you may get a prompt like this:



This prompt indicates that you have a template that is out of date.

Your choices are:

- Skip - Skips the update process and lets you carry on with your work.
- Show - Highlights the objects in the project view which are out of date.
- Update All and Save - Updates the templates - i.e. selects the objects that are out of date, updates them and saves them.

There are now also two options in the Project View menu that deal with Templates:

- Select all out of date templates - This will highlight all objects that are out of date.
- Update selected templates - If the template is out-of-date, you will get this option, this will update the object to the latest template.

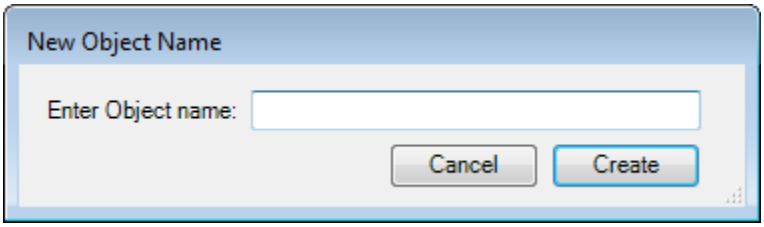
Creating a New Object

You cannot compile character components (such as clothing) or scene components (such as furniture) from scratch in the Object Editor. You must first create them and export the geometry using the DCC tool (for example, Maya).

The only objects that can be compiled without using the DCC tools are non-furniture, non-clothing objects such as 2D and 3D interactive content. These are the only types of object that you can create in the Object Editor.

To create a new object without specifying its type or automatically adding any specific components or resources:

1. Click the New Object button  on the toolbar, or select File > New Object. The New Object Name dialog is displayed:



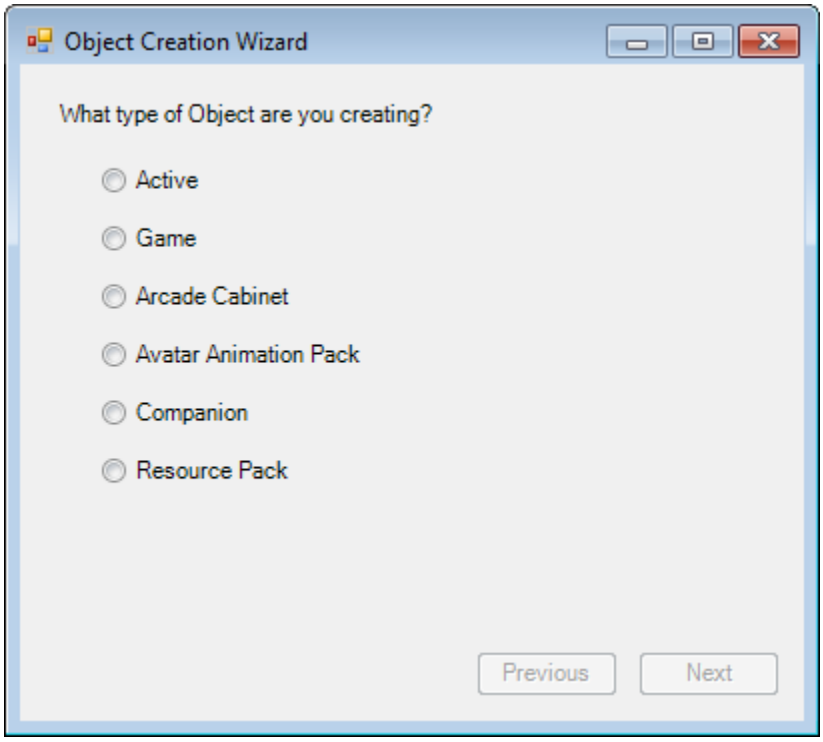
2. Enter the name of the object and click Create. The object is created and added to your <HDK_INSTALL_DIR>\build\objects folder.

To create a new object by specifying its type (ensuring that they contain the standard components and resources):

1. Select File > New Object Wizard... or right-click on the appropriate project or group In the HDK Project panel and select

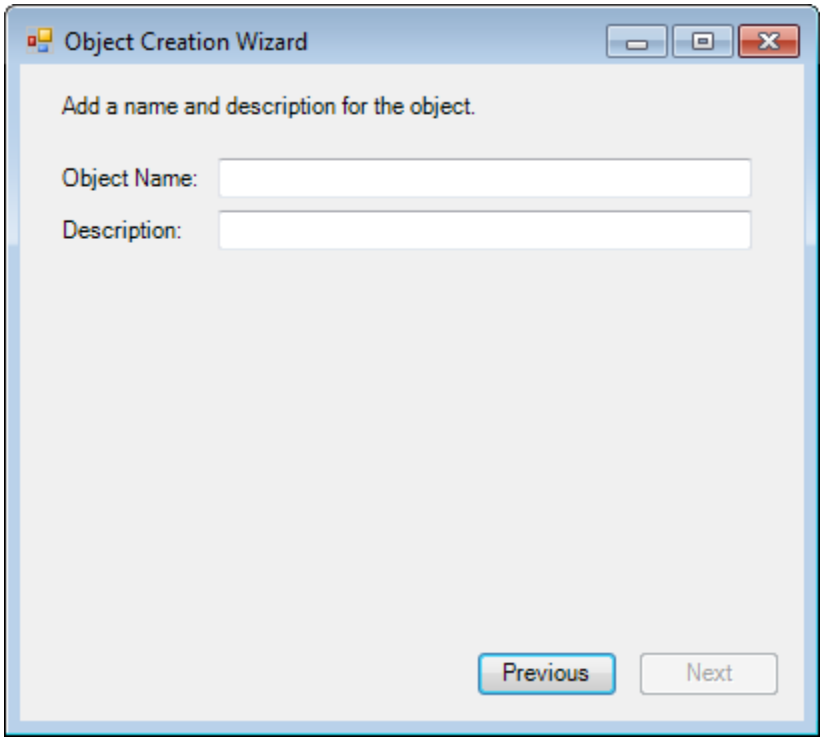
Create new object here from the pop-up menu displayed.

2. Select the required object type, then click Next:

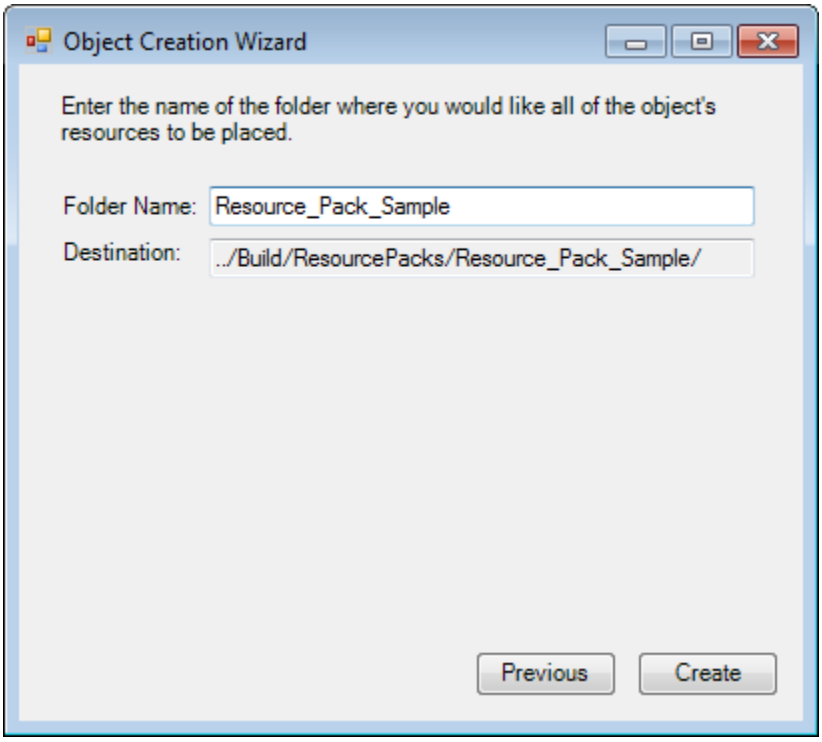


> If you select Game or Companion, you will be prompted to make additional selections based on the type of game or companion required. For more information, see Game Components and Creating Companion Objects.

3. Enter a name and description for the object, then click Next:



4. Enter the name of the folder where you want the object's resources to be placed:



By default, the name of the folder is the same as the object's name.

5. Click Create. The object is created and added to your <HDK_INSTALL_DIR>\build\objects folder.

The Resources folder in the HDK Project panel contains a list of all files related to an object, including source files of the models. If you select Resources in the Object View panel, the resource files added to the object are displayed. You can then edit the components and replace the resources with your own custom files.

The following table shows the standard components and resources that are created for each object type:

Object	Components	Resources
Active Item	Header Furniture Lua Environment Active Item	ActiveModel.mdl ActiveModel.hkx ActiveModel.safevolume.hkx Active.lua
Active Mini Game	Header Furniture Lua Environment Network MiniGame Active Item Game Spawner	ActiveModel.mdl Network.xml ActiveModel.hkx ActiveModel.safevolume.hkx MiniGame.lua
Active Realtime Game	Header Furniture Lua Environment Active Item RealTime Game	ActiveModel.mdl Network.xml ActiveModel.hkx ActiveModel.safevolume.hkx RealTimeGame.lua
Arcade Cabinet	Header Furniture Entity Game Spawner	ArcadeCabinet.mdl screen.dds ArcadeCabinet.hkx

Arcade Game	Header Lua Environment Arcade Game Screen Pad Renderer	ArcadeGame.lua
Avatar Animation Pack	Header Lua Environment Entity Particles Network Renderer Camera System Repertoire Portable Resource Pack	configuration.xml ap_attack.ani ap_pose1.ani ap_pose1_in.ani ap_pose2.ani ap_pose2_in.ani ap_standidle.ani apf_pose1.ani apf_pose1_in.ani audio.bnk male.xml female.xml male_emotes.xml female_emotes.xml boot.luac
Companion	Header Lua Environment Targetable Entity Particles Network Screen Pad Renderer Camera System Light Portable Text Label Resource Pack	wolf.mdl wolf_lo.mdl smoke.efx fly.efx trails.efx config.xml sphere.hkx idle.ani walk.ani applaud.ani an_flap.ani an_sat.ani an_land.ani an_twist.ani an_yawn.ani an_fly.ani dance.ani run.ani shakefists.ani wolf.skn audio.bnk boot.luac
Embedded Realtime Game	Header Lua Environment Scene Object RealTime Game	Network.xml RealTimeGame.lua
Resource Pack	Header Resource Pack	N/A
Scene Mini Game	Header Lua Environment Network MiniGame	Network.xml MiniGame.lua

For more information on using components, see Working with Object Components.

For information on using resources, see Managing Object Resources.

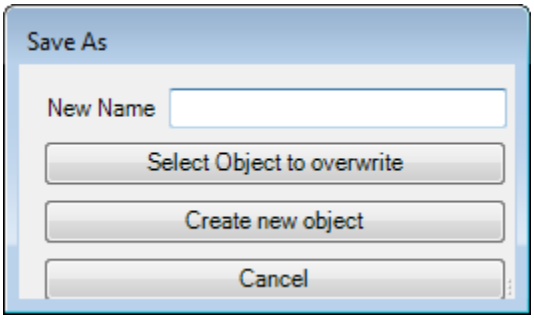
Overwriting and Copying Objects

You can overwrite or copy an object from both the HDK Project panel and the Object Search dialog in the Object Editor.

Overwriting Objects

To overwrite an object:

1. In the Object Editor, search for and select the object that contains the data you want to copy.
2. Select File > Save Object As... to display the Save As dialog:



3. Enter a name for the object, and click Select Object to overwrite. The Object Search dialog opens.
4. Click Show all or search for the object you want to overwrite.
5. Select the object and then click Open. The object is overwritten and renamed.

> You can use this option to rename an object by selecting the same object in step 1 and step 5.

Copying Objects

To create a copy of an object:

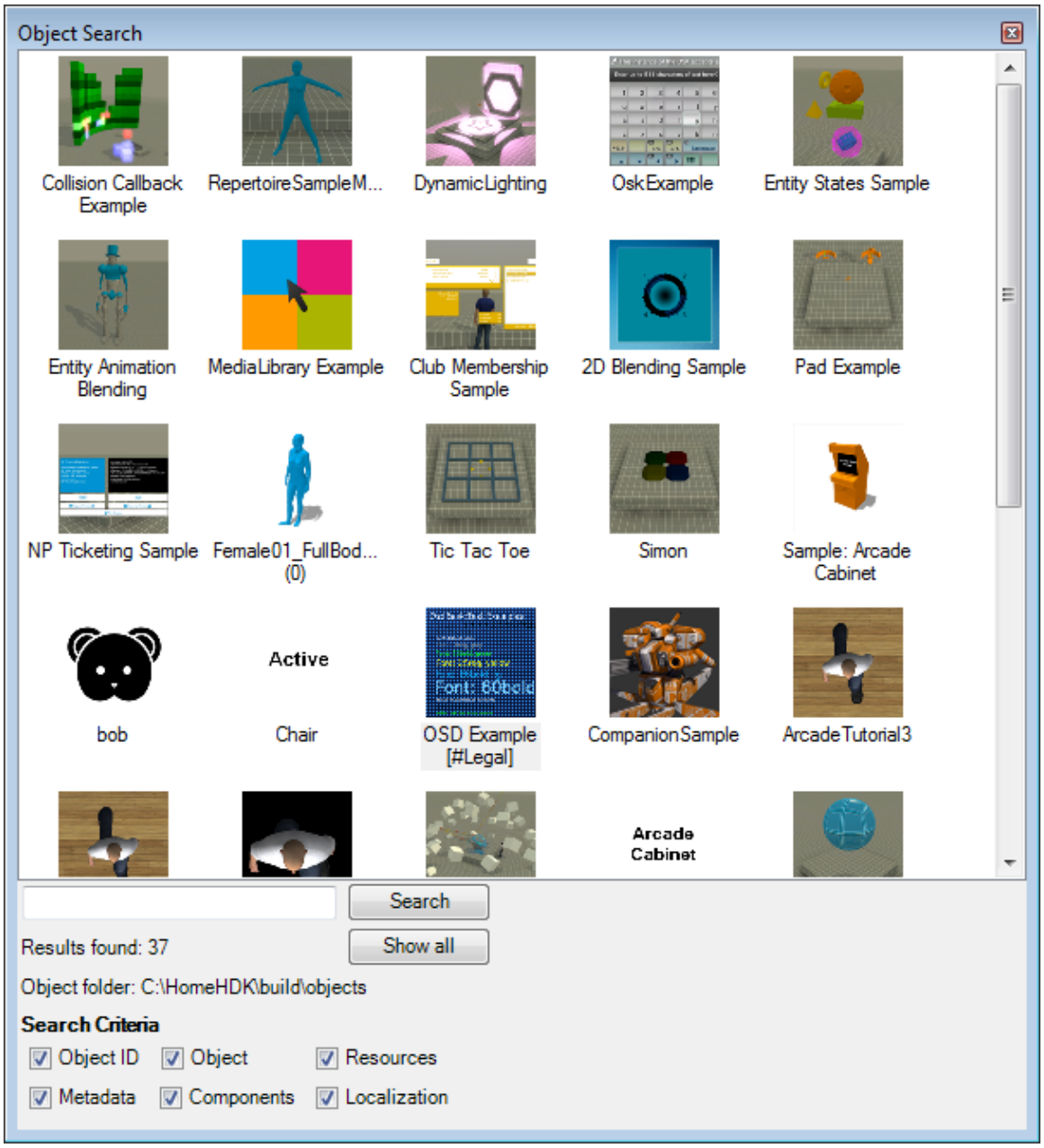
1. In the Object Editor, search for and select the object you want to copy.
2. Select File > Save Object As... to display the Save As dialog.
3. Enter a name for the object and click Create new object. A copy of the object is created.

Viewing and Searching for Objects

Viewing All Available Objects

You can use the Object Editor to view all the objects in your <HDK_BUILD_DIR>/build/objects folder.

To display all available objects in the Object Search panel, click Show all or leave the search field blank and click Search. All the objects in your <HDK_BUILD_DIR>/build/objects folder are displayed, for example:

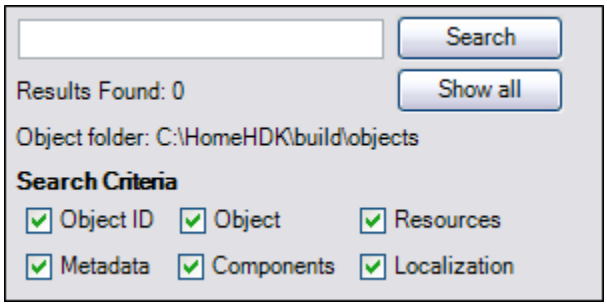


Searching for Objects

You can search for an object using their object ID, name, resources, metadata, components, and localization data.

To run a search:

1. Specify the criteria you require in the search field at the bottom of the Object Search panel:



You can enter full or partial search terms. For example, to search for a 'Knight Table' furniture item, check the Object

box and type the following:

- 'Knight Table': Returns items with 'Knight Table' in their name.
- 'Knight': Returns objects with the term 'Knight' in their name.
- 'Table': Returns objects with the word 'Table' in their name.

> Common PS Home terms such as 'Table' return a long list of results.

2. To run an 'or' search or wildcard search, you can use the following characters:

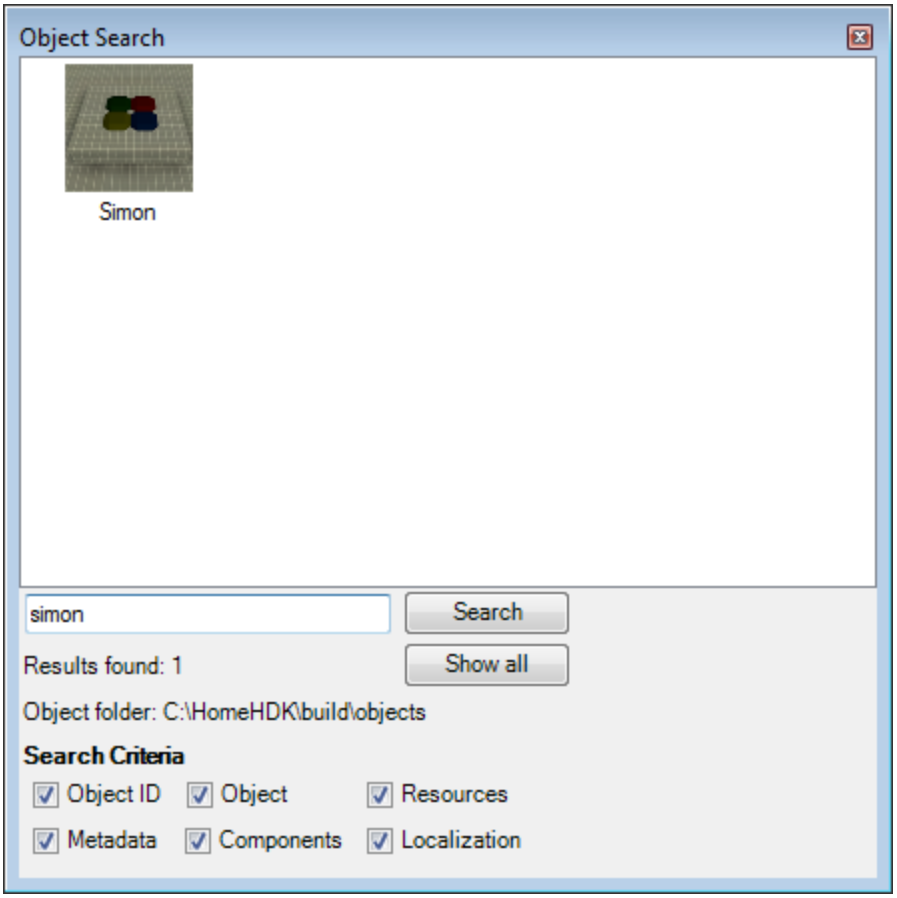
   - A semicolon (;) - acts as an 'or' term. You can tell the Object Editor to search for objects with one term OR another. For example, 'chair ; table' returns all objects that are either chairs or tables.
   - A percent sign (%) - allows you to ignore any content between search terms. For example, if you know an object ID begins with CB3 and ends with OB, enter 'CB3 % OB' as a search term. The search returns all matching objects.

Search Examples

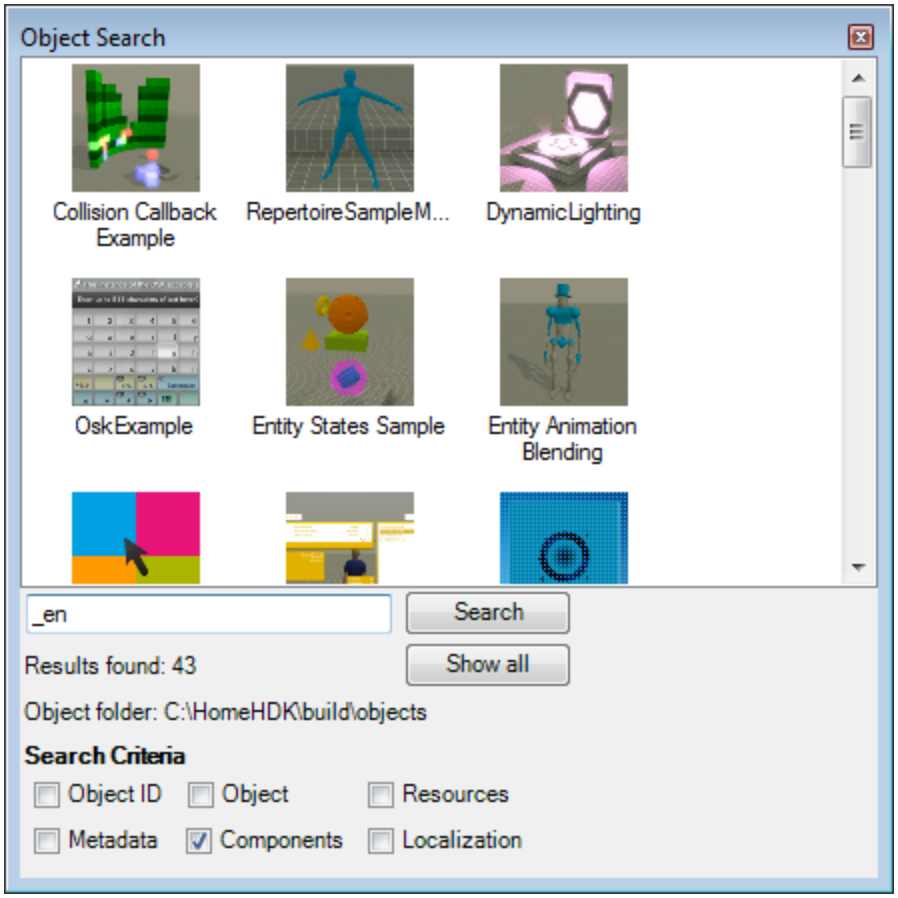
The following table shows some examples of full and partial search terms that you might use for different search criteria:

Criteria	Full term	Partial term
Object ID	xxxxxxxd-fdxxxxxxxx-xxxxxxdd	xd
Object	Oval Table	Oval Tab
Resources	AwesomeThumbnail_1.png	meThumb
Metadata	Table	Tab
Components	Furniture	Furn
Localization	Tavolo	avo

When you enter a search term, all the items that match the search criteria are displayed in the Object Search panel. The following image shows the results for a search for the Simon mini-game sample:



The following image shows results for a partial search, _en, which returns all of the objects with the Lua Environment component:



Editing Multiple Objects

Using the Object Editor, you can edit certain properties for several objects at once.

> Note that:
>
> - You cannot multiple delete content. For example, you cannot remove a component or resource from several objects simultaneously. You must do this with each object individually.
> - When you multiple edit metadata, thumbnails and age ratings, you overwrite any metadata, thumbnails and age rating information that existed on any of the objects previously. There is no way to stop the overwriting.

You can multiple edit the following properties:

- Thumbnails
- Age Ratings
- Components
- Resources
- Metadata
- Localization default language

To edit multiple objects at once:

1. Hold down the Ctrl key and select those objects that you want to edit in the Object Search panel or the HDK Project panel.
2. Edit the objects as you would for a single object.
3. Click the Save All button  in the toolbar or select File > Save All after making a change.

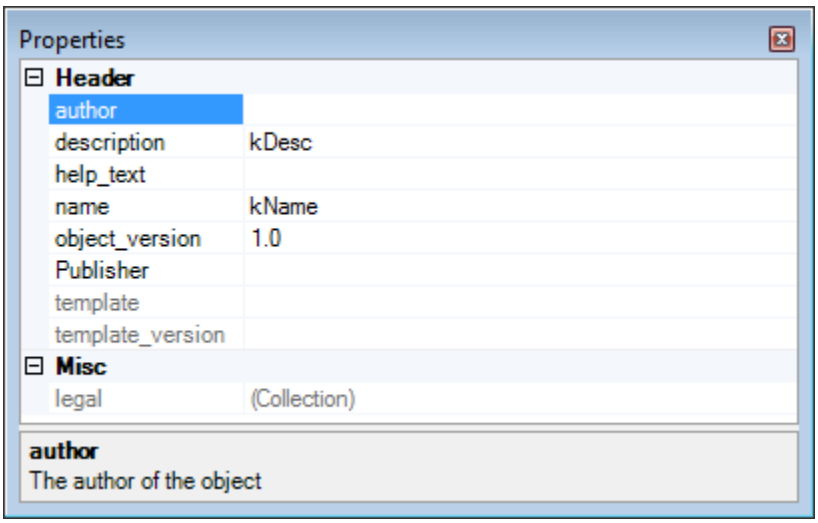
> If you do not select a save all option, a dialog is displayed, asking you to save each object in the multiple edit. This means that, if you edit 100 objects, you will be asked 100 times if you want to save the object.

Completing the Object Header

Every object must have a header. The header contains basic information about the object that is displayed in PS Home to the users, and important localization information.

To complete the header in the Object Editor:

1. In the Object View panel, select the Header component.
2. In the Properties panel, enter the author of the object in the author field. This field is never displayed in PS Home:



3. If required, select the description for the object from the description drop-down list. The list is provided by importing the localization file for the object. See Localizing Objects.
4. Add text to be displayed in the contextual help system for your mini-game or realtime game objects in the help_text field. Use this field to display useful and accurate information about what the user is interacting with in the game. As for the description field, you must add the text to the localisation.xml file, then reference it here. If you leave this field blank, no help is displayed.

> This field supports only mini-game and realtime game objects. If you add help text for any other type of object, you get the following message: "Object has an entry for 'help_text' but does not have a mini_game or a realtime_game component".

5. If required, select the name for the object from the name drop-down list. The list is provided by importing the localization file for the object.
6. If the object has a version, enter the iteration in the object_version field. A game launch object uses this property to determine which version of the PlayStation®3 game title it supports.
7. Enter the name of the publisher in the Publisher field. The name is visible to users when purchasing the item, and when they select the item and then select the About option.

Character Limits

Information about the object must not exceed the maximum number of characters:

Information	Maximum Characters
Author	Author information is not included in the object packaging, so there is no memory limit, but be reasonable with what you enter.
Object Description	4,096 bytes
Help Text	No hard character limit as the limit depends on the amount of available memory in the OSD memory pool. However, for contextual help to be useful it needs to be succinct (but not so brief as to be ambiguous or obscure).
Object display name	128 bytes
Publisher Name (can also appear as 'Company Name')	128 bytes

Entitlement ID	32 bytes
Category ID	57 bytes
Product ID	49 bytes
Element field	40 bytes
Thumbnail Publisher/Small/Large image URL	256 bytes each

Character limits are in bytes, which is the UTF8-encoded string size for the field.

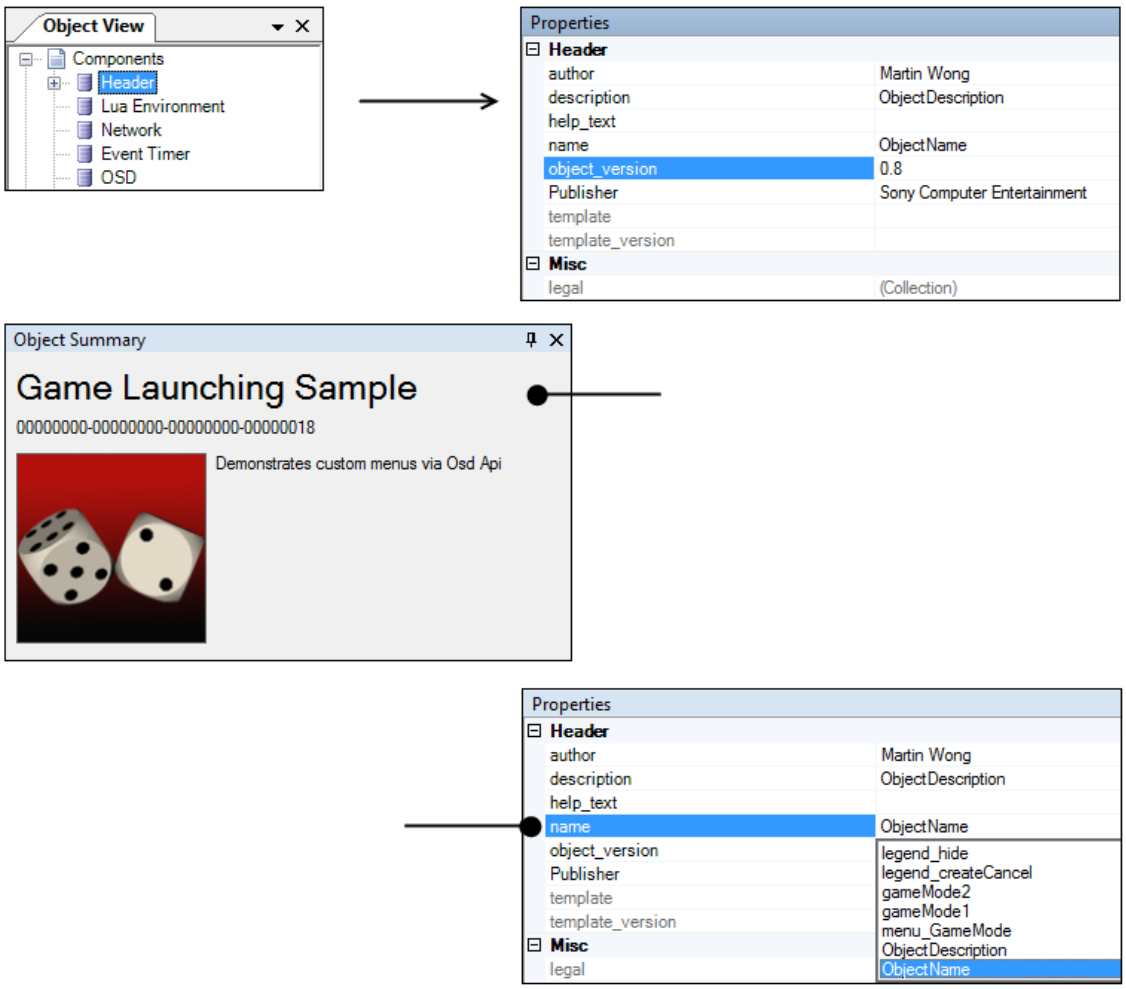
Example

The following example shows the author heading, and provides a brief explanation of characters per byte. For a full list, see HDK Tools Validations.

author is restricted to string lengths totaling 127 bytes. The number of characters available varies depending on the characters being used, due to UTF-8 encoding. For example, Chinese characters normally require three bytes to encode, so in Chinese characters, 127 bytes is approximately 42 characters.

description has a maximum of 4095 bytes, which is approximately 1365 Chinese characters.

Descriptions support new lines in the form of '¥n', which is particularly useful for developers entering Japanese descriptions.



Working with Object Components

You can add components to and delete components from single objects.

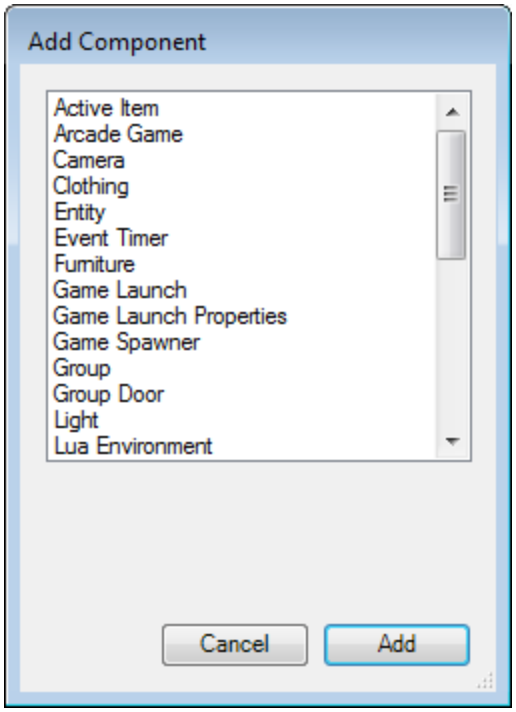
For more information on editing multiple objects, see Multiple Objects Editing.

For a list of invalid component combinations, see HDK Tools Validations.

Adding a Component

To add a component to a single object in the Object Editor:

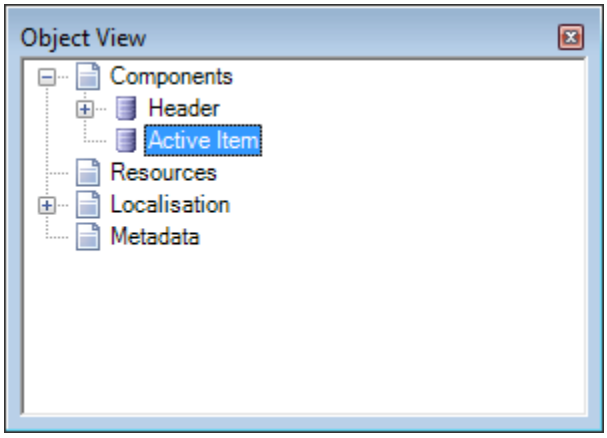
1. Click Add New Component in the toolbar or select Object > Add New Component: The Add Component dialog is displayed, listing all the available components:



2. Select the required component, then click Add.

> To select more than one component, hold down the Ctrl key while making your selection.

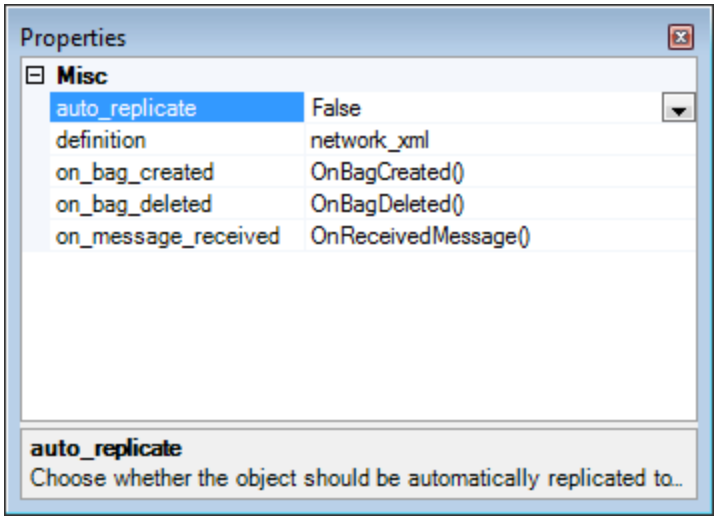
The component you added is displayed in the Object View panel under Components:



Editing a Component

After you have added a component to your object, you can edit its properties. Select the component in the Object View panel to display its properties in the Properties panel.

The following example shows the Network component, which requires several Lua functions registered against certain events common to the network framework:



Repeat this process for each component, as necessary, until the object you are creating meets the required criteria.

For more information on each component, see Object Component Properties

Deleting a Component

To delete a component:

1. select the component in the Object View panel.
2. Do one of the following:
   - Click the Delete button  in the toolbar.
   - Press the Delete key on your keyboard.
   - Select Object > Delete Component.

Object Component Properties

Components define how the object is treated and what resources are allocated for the object by the client. Certain scripting resources are available to an object only when it contains the relevant component. For example, the controller Pad scripting library is available only if the Pad component is included. This saves memory by only allocating the resources that are needed for each object.

Not all component combinations are currently supported. However, additional component combinations will be developed in the future. For a list of invalid component combinations, see HDK Tools Validations.

> ⚠ The Object Editor's packaging tool checks component combinations. When it finds invalid combinations, it stops the packaging process, and lists the conflicting components.

For information on how to add components, see Working with Object Components.

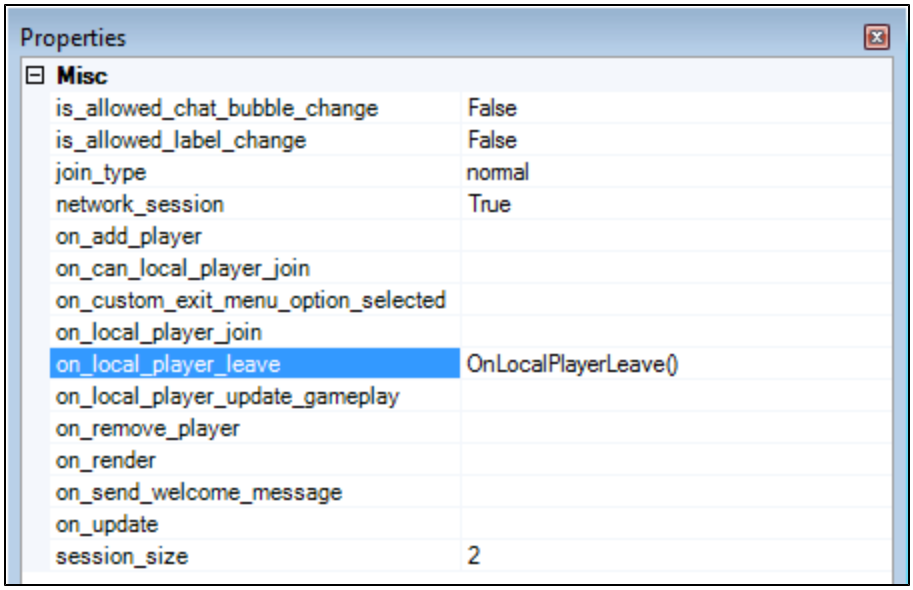
Callbacks

Several components have callbacks in their properties that can call a function in an attached script when a certain event occurs in PS Home. For example, the on_update callback calls the specified function on each frame update.

Callbacks affect the object's performance in PS Home, so do not specify empty functions. For example, an object's Lua Script contains the function:

```
function OnLocalPlayerLeave()
end
```

This function does nothing, but it is specified in the MiniGame component's properties:



With these settings, whenever a local player leaves the mini-game, the client calls the function `OnLocalPlayerLeave()`, but as it does nothing, it wastes processing time and adversely affects the mini-game's performance.

For callbacks such as `on_local_player_leave` the effects may not be noticeable, but on callbacks that occur more frequently, such as on_render, on_update, the effects can be significant.

Object Component Types

With many components, such as the Scene Object component, the button to the left of the localization buttons () is enabled to allow you to add more of that type of component to the object. After adding the component, select one of the parameters (boolean, string, enum, numeric). The button changes to Add a new parameter, which allows you to add a default parameter to your object.

Objects consisting of furniture or clothing should have already been created and exported using the HDK DCC tools, so they should already comprise those components. In some cases however, you might want to extend the interactive functionality of these type objects using additional components.

The following table lists the objects that you can edit in the Object Editor:

Component Name	Description
Active Item	An active item is an item of furniture that can have a script attached. All active items need a Furniture component as well as an Active Item component. However, they lose furniture behavior (for example, seating or lighting behavior). To regain furniture behavior, you must script it.
Arcade Game	Subscribes to the `ArcadeGame` library and related functionality common to most interactive 2D content. Also registers an update/tick function.
Camera	Subscribes to the `Camera` library and related functionality.
Clothing	Registers an object as a clothing item, along with its associated model and texture resources. Do not create clothing objects in the Object Editor. Create them using the DCC tool (Maya) and export them. After valid and successful export from the DCC tool, you can you edit the objects in the Object Editor.
Entity	Subscribes to the `Entity` library and related functionality common to most non-furniture/non-clothing 3D models.
Event Timer	Registers a Lua function to be triggered every X milliseconds.
Furniture	Registers an object as a furniture item, along with its associated model, texture and collision resources. Do not create furniture objects in the Object Editor. Create them using the DCC tool (Maya) and export them. After valid and successful export from the DCC tool, you can you edit the objects in the Object Editor.
Game Launch	Subscribes to the game launching framework with access to the `GameLaunch` and `GameLaunchExporter` libraries. See Additional Game Launching Information.

Game Launch Properties	Allows the object to launch a PlayStation®3 title without using the Home Game Launching framework. Use the Lua API function System.ExecTitle to launch the title. If session management is required then you need to develop this yourself within the Lua script. Otherwise, use the Game Launch component instead to take advantage of the Home Game Launching framework. The Game Launch Properties component does not itself have properties. For more information, see Launching From Mini-games.
Game Spawner	Required for an arcade cabinet, or to make a mini-game an active item.
Group	Adds the Group Lua library to the object. This library is used to query the status and details of the user's group or form/join a group if they are not already in one.
Group Door	Adds the GroupDoor Lua library to the object. This library allows new Group Doors to be created and entered.
Light	Add the Dynamic Light Lua library and related functionality to the object.
Lua Enviroment	Registers a Lua environment as an integral part of an object. This gives you access to most of the libraries in the Lua API (apart from the Scene library where only objects owned by the scene are granted permission to use it, including scene objects (including embedded active items) and mini-game objects. Scene Scripting also has access to the Scene library). Certain libraries require specific components to be added to the object: • Pad • Entity • OutboundMessage • ReceivedMessage • NetPropertyBag • Camera • GameLaunch libraries • Light • Osd • Renderer • Sprite • SpriteAnim • Repertoire • Rewards • Screen • Portable • Active • ArcadeGame
Memory Slots	Allows custom memory slots to be defined for the object. These memory slots can be referred to in the object's Lua scripts to load resources in to specific memory pools. For more information about memory slots see Object Memory Controls.
MiniGame	Subscribes to the MiniGame library and related functions common to most interactive 3D content. Also registers an update or tick function.
Network	Subscribes to the network framework with access to the following libraries: • NetPropertyBag • OutboundMessage • ReceivedMessage
OSD	Subscribes to the OSD framework with access to the following libraries: • Osd • OsdBasicChip • OsdBasicSingleGraphic • OsdBasicText • OsdObject • OsdSelectionParent • OsdTextLines • OsdTextPanel
Pad	Subscribes to Pad library for dealing with pad buttons.
Particles	Currently only environment effects are supported, not object particle effects.
RealTime Game	Subscribes to the RTGame and RTGameObject libraries. It allows the object to support realtime gaming.

Renderer	Subscribes to the Renderer framework for dealing 2D rendering support in full screen and gains access to libraries: • Renderer • Sprite • SpriteAnim
Resource Pack	Allows use of the Resource Pack system. Both a Resource Pack object and an object which uses Resource Packs requires this component.
Repertoire	Gives the mini-game, realtime game, or FullBodySuit access to the Repertoire Editor, allowing you to add custom avatar animations. This component is only compatible with mini-games, realtime games or FullBodySuit.
Rig	Registers object with applicable body types and other skeleton-related properties for clothing.
Scene Object	Allows an object to be embedded within a scene. Also provides access to instance parameters.
Scene Transition	Adds the SceneTransition Lua library to the object. This library is used to script custom transition sequences between scenes.
Screen	Subscribes to the Screen library for rendering media onto 2D surfaces.
System	This component does nothing. Do not use.
Targetable	Registers a Lua function to call when an object is activated by a character.

Additional Game Launching Information

The component reference above is supplemented in this section in relation to creating game launching objects:

Component Name	Description
Lua Environment	Adding this component assigns the object a Lua environment for scripting. The name of the file is denoted by the <script> tag, here, 'main' has been used which corresponds to main.lua (defined in resources.xml, see below) however any valid filename may be used.
Game Launch	This is the Game Launch component itself, it must be specified in order to subscribe to its services. The <communicationId> must be specified which will be used to determine what title IDs can be game launched together using this object.
Event Timer	The <event_timer> component has been used here to implement an update loop. It calls a function called Update() every frame. Both of these items can be user definable
Network	By subscribing to this component the Lua script environment gains permission to use the networking libraries ReceivedMessage, OutboundMessage and also has access to Network PropertyBags Here, the <on_message_received> property can be used to register a Lua callback function which is called when the Lua environment receives a network message.

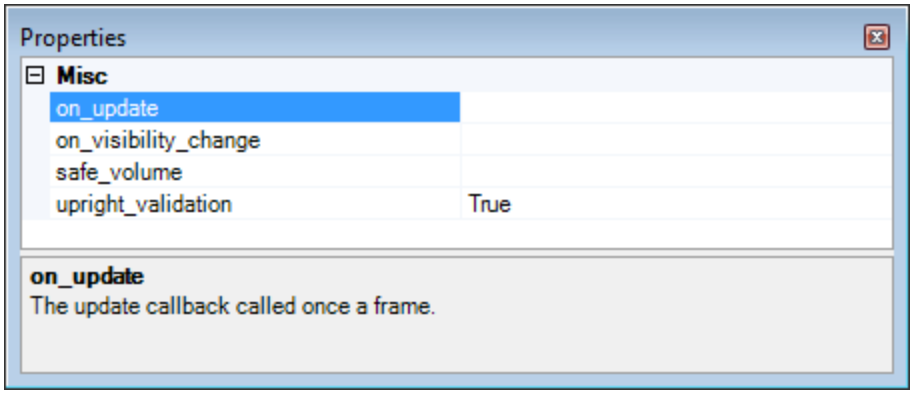
For more information, see The Game Launch Object Component.

Active Item Component

An active item is an item of furniture that can have a script attached. All active items need a Furniture component as well as an Active Item component; however, they lose furniture behavior (for example, seating/lighting). To regain furniture behavior, it must be scripted.

For more information on using the Active Item component, see Active Items.

Properties



Property	Description
on_update	This is the update callback called every frame in the associated Lua script (as defined in the lua_environment component).
on_visibility_change	A boolean callback triggered when the active's visibility changes. This callback takes a direct call (*FunctionName()). An active item's visibility changes when the user elects to remove the active from the scene (*FunctionName(False)). If the user cancels the removal, the visibility returns to True (*FunctionName(True)).
safe_volume	This is the collision file that describes the area that cannot be intersecting when the object is placed by a user.
upright_validation	This is a boolean that if set to True will not allow objects that have fallen over to be placed like that in the scene.

Arcade Game Component

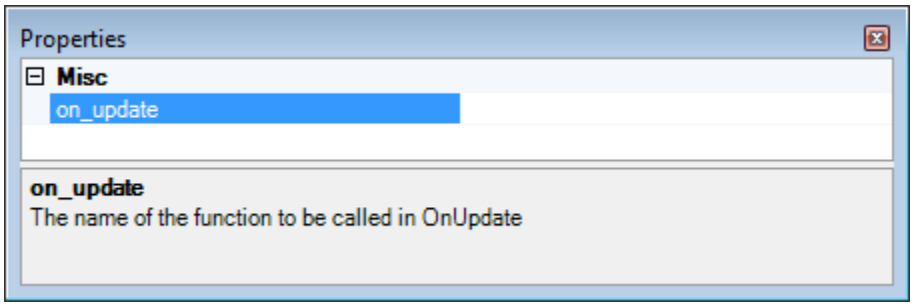
Subscribes to the ArcadeGame library and related functionality common to most interactive 2D content. Also registers an update/tick function.

For more information on using the Arcade Game component, see Arcade Games

Samples

- Games > Arcade Tutorial
- Games > Advanced Arcade Game

Properties



Property	Description
on_update	This is the update callback called every frame in the associated Lua script.

Camera Component

Subscribes to the Camera library and related functionality.

For more information on using the Camera component, see PS Home Camera.

Samples

- Media > Media Library
- Games > Simon
- Games > Tic Tac Toe
- HTS Aeronautilus > Telegraph Game

Properties

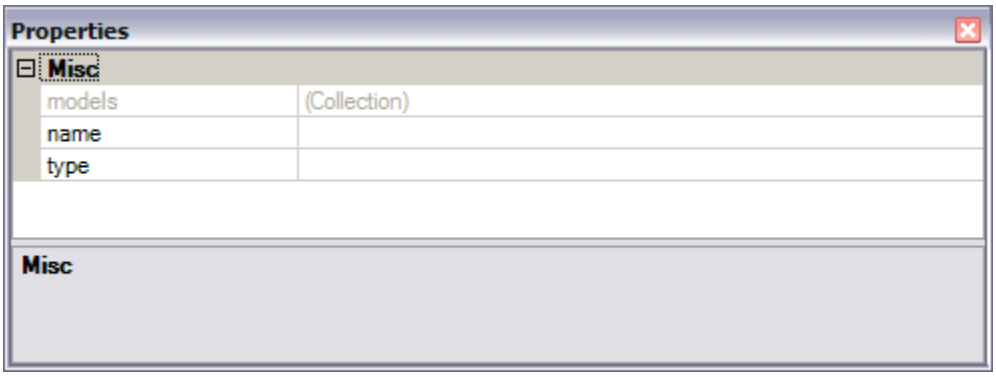
The Camera component has no Object Editor properties.

Clothing Component

Registers an object as a clothing item, along with its associated model and texture resources.

> Do not create clothing objects in the Object Editor. Create them using the DCC tool (Maya) and export them. After valid and successful export from the DCC tool, you can you edit the objects in the Object Editor.

Properties



Property	Description
Model	List of models used by the clothing item
Name	Name of the clothing component
Type	Type of clothing

Entity Component

Subscribes to the Entity library and related functionality common to most non-furniture/non-clothing 3D models.

For more information on using the Entity component, see Deploying an Arcade Game in Furniture and Adding Dynamic Lighting.

Samples

- Games > Simon
- Games > Tic Tac Toe
- HTS Aeronautilus > Telegraph Game

Properties

The Entity component has no Object Editor properties.

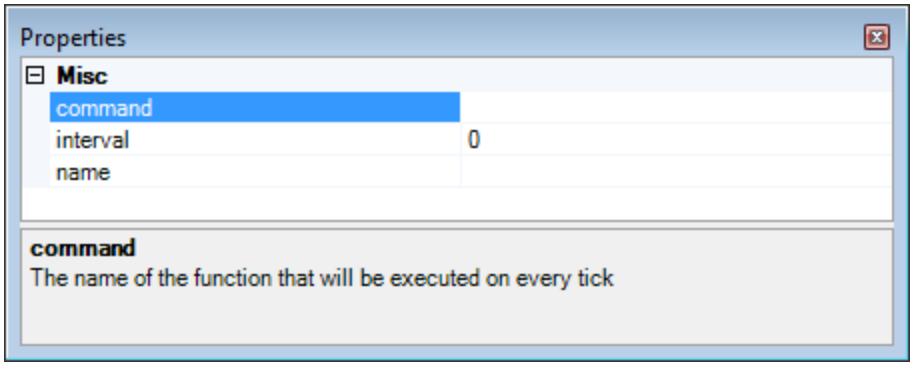
Event Timer Component

Registers a Lua function to be triggered every X milliseconds.

Samples

- Games > Simon

Properties



Property	Description
command	The Lua function to call every tick.
interval	The rate (in milliseconds) that you want to trigger the function.
name	The name of the event timer.

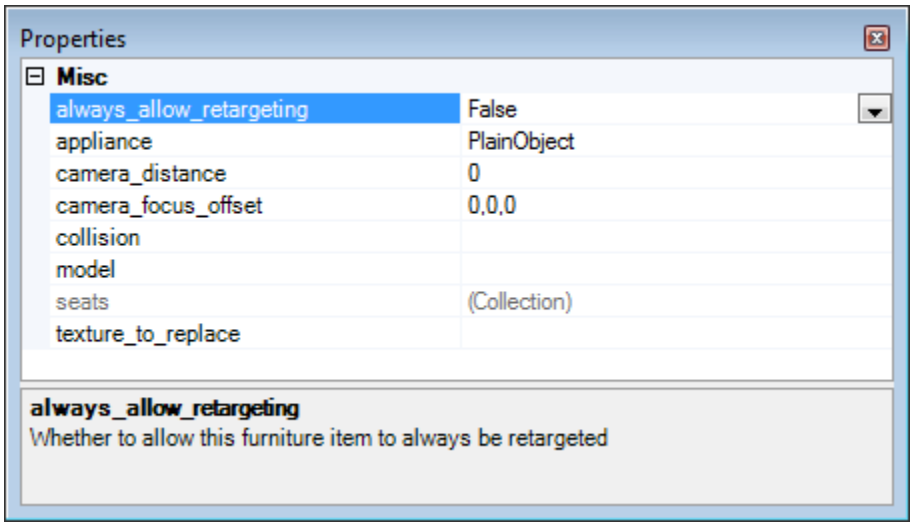
Furniture Component

Registers an object as a furniture item, along with its associated model, texture and collision resources. When you export a furniture item from Maya, the Furniture component is automatically added to the object.

For more information on using Furniture components, see:

- Furniture and Decorations
- Active Items
- Furniture Block System
- Embed Objects into Scenes

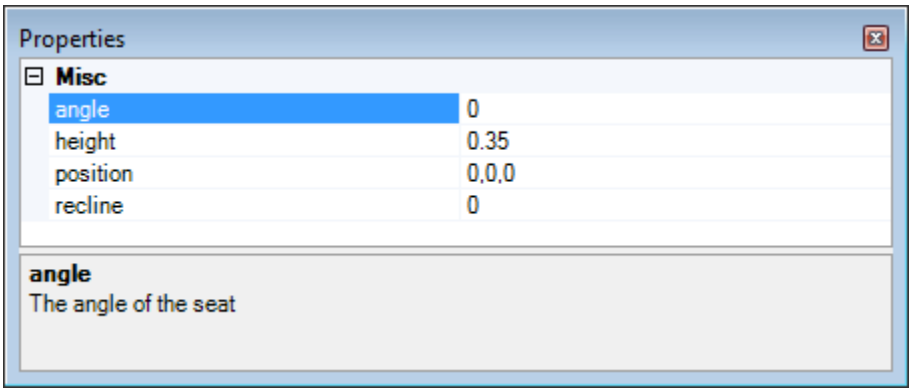
Properties



Property	Description
always_allow_retargeting	Determines whether to allow the furniture item to always be retargeted.
appliance	This is a drop-down list that lets you choose one of four furniture types. When you export your furniture item from Maya the appliance is automatically set. The four types are: • PlainObject: Any furniture type that is not a seat, picture frame, or a lamp. • Seat: This has no editable properties from this window. • Picture Frame: An object that can be placed on picture hooks on walls. • Lamp: Furniture that has a light source that users can activate. See Creating a Lamp.

camera_distance	Sets the look-at distance from the object.
camera_focus_offset	Using the furniture as the origin, this lets you set the offset for the look-at point. Specify X, Y, Z coordinates.
collision	This is the collision file to use for the furniture. When you export from Maya, it is created and allocated automatically.
model	This is the model file to use for the furniture. When you export from Maya, it is created and allocated automatically.
seats	Any seats that you added to the object through Maya or through the Object Editor's Add New Seat dialog appear under this property (see below). Remember there are 3 seating positions. See Dimensions.
texture_to_replace	This is only used for picture frames. This is the texture to replace, when you export from Maya it is automatically allocated. See Picture Frames and Wall Decorations.

seats Properties



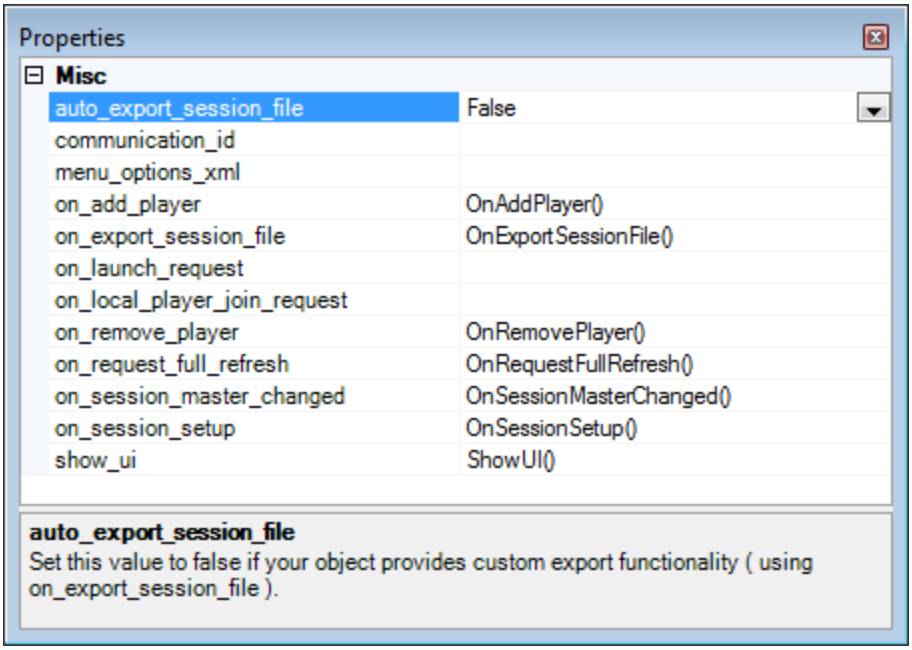
Property	Description
angle	Sets the angle of the seat.
height	A drop-down list that lets you choose three seat heights that correspond to the low, medium and high seat heights in Maya.
position	Allows you to enter the seats coordinates, with the object itself used as origin.
recline	Sets the angle of recline.

Game Launch Component

Subscribes to the game launching framework with access to the GameLaunch and GameLaunchExporter libraries.

For more information on using Game Launch components, see Game Launch Objects.

Properties



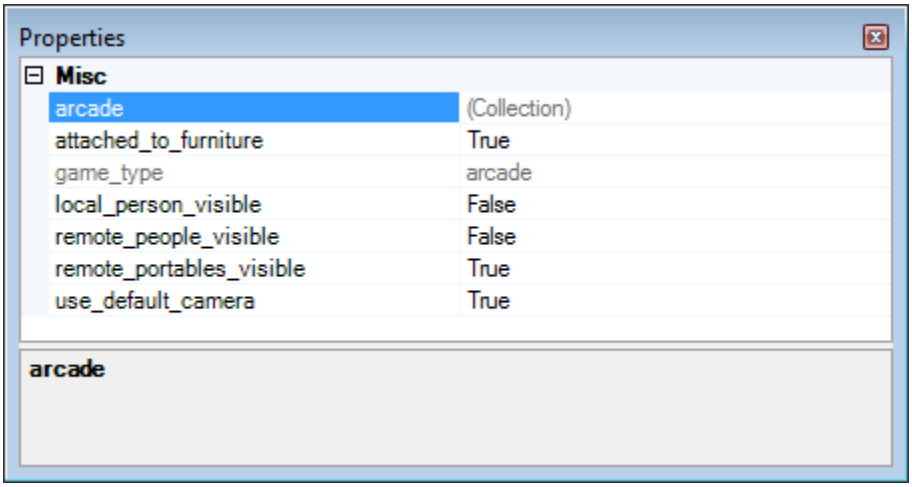
The properties for the Game Launch component are comprehensively explained in The Game Launch Object Component.

Game Spawner Component

You must use this component when making an active item that has a MiniGame component or an Arcade Game component.

For more information on using the Game Spawner component, see Active Items.

Properties

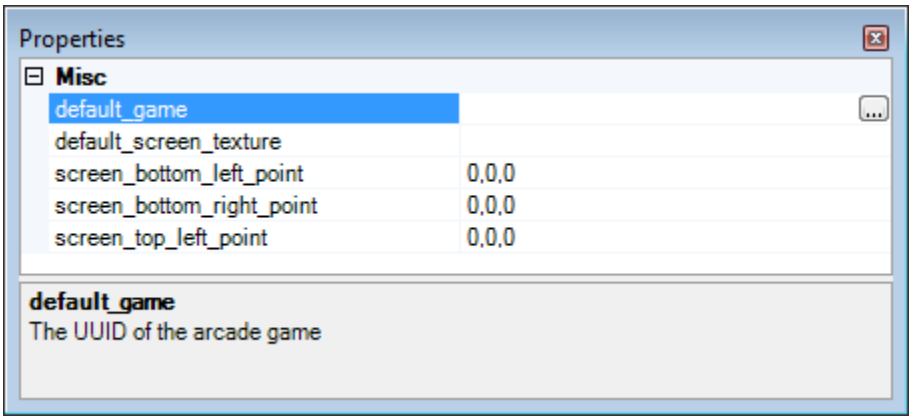


Property	Description
arcade	Only displayed if you have selected Arcade Game. You cannot edit this property.
attached_to_furniture	Allows you to set if the game is attached to a furniture object. Set True if you are making this as part of a furniture object. Set false if you are going to use it as an scene object in a scene.
game_type	Shows the type of game you chose (arcade or mini-game).
local_person_visible	A boolean that lets you choose whether the user's avatar is visible while playing the game.
remote_people_visible	A boolean that lets you choose whether other users' avatars are visible while the local user is playing the game.
remote_portables_visible	A boolean that lets you choose whether inventory items (portables) are visible while playing the game.

use_default_camera	Lets you choose to use the default camera, or to control the camera through script.

arcade Properties

If you chose mini_game, there will be no properties for this node.

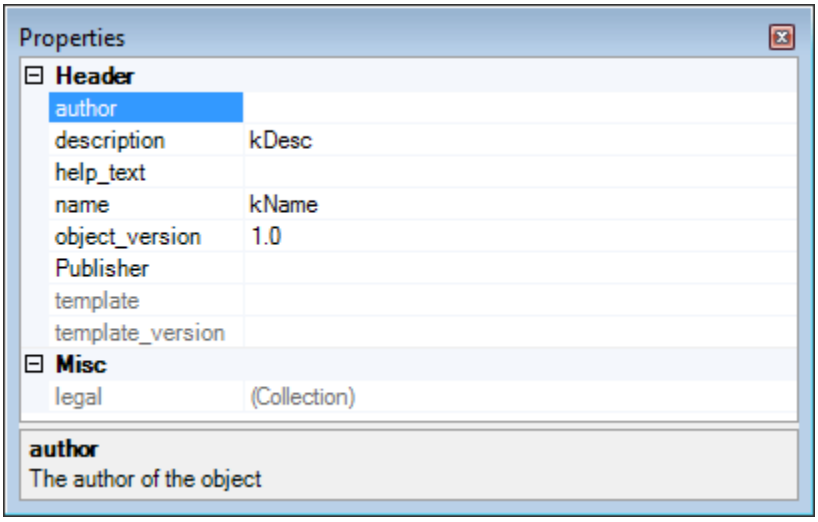


Property	Description
default_game	The object ID of the arcade game that you want tied to the furniture item.
default_screen_texture	The texture image that is displayed on the screen when the arcade game is inactive. Add a DDS texture resource to the object and assign this resource to this property.
screen_bottom_left_point	The bottom left coordinate of the arcade game screen in the local space of the furniture item.
screen_bottom_right_point	The top right coordinate of the arcade game screen in the local space of the furniture item.
screen_top_left_point	The top right coordinate of the arcade game screen in the local space of the furniture item.

Header Component

Every object must have a Header component. The Header component contains basic information about the object that is displayed in PS Home to the users, and important localization information.

Properties



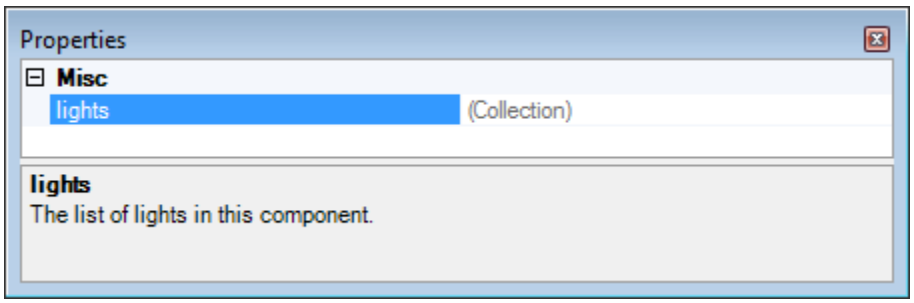
Field	Description
author	The author of the object. This field is never displayed in PS Home
description	The description for the object. The list is provided by importing the localization file for the object. See Localizing Objects.

help_text	The text to be displayed in the contextual help system for your mini-game or realtime game objects. Use this field to display useful and accurate information about what the user is interacting with in the game. As for the description field, you must add the text to the localisation.xml file, then reference it here. If you leave this field blank, no help is displayed. This field supports only mini-game and realtime game objects. If you add help text for any other type of object, the following message is displayed: "Object has an entry for 'help_text' but does not have a mini_game or a realtime_game component".
name	The name of the object. The list is provided by importing the localization file for the object.
object_version	The object's version iteration. A Game Launch object uses this property to determine which version of the PlayStation®3 game title it supports.
Publisher	The name of the publisher. The name is visible to users when purchasing the item, and when they select the item and then select the About option.
template	The template that the object was created from (if applicable). This is a read-only field.
template_version	The version of the template that the object was created from. This is a read-only field.
legal	Legal information about the object, such as the age rating. This should be edited in the Object Age Rating dialog. See Preparing Objects for Packaging.

Light Component

Adds the Dynamic Light Lua library and related functionality to the object. If an object is exported from Maya with a light locator, the Light component is automatically added, and the light locator is automatically allocated as a daughter component.

Properties

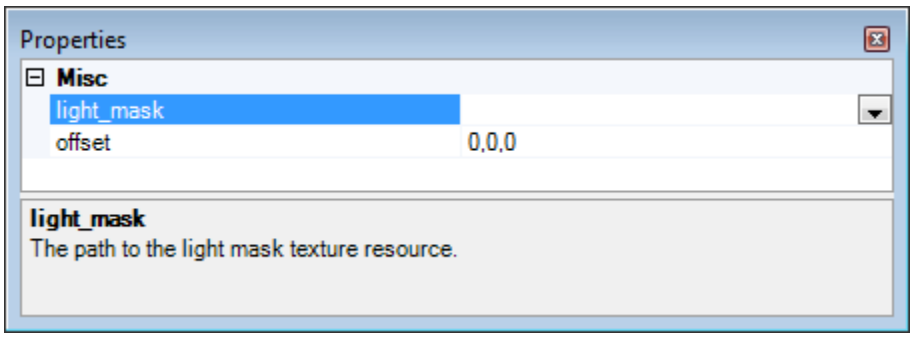


Property	Description
lights	This item has no properties that you can edit from this panel.

Light Daughter Component Properties

If you click Add a new light in the toolbar, you can add lights to your object. See also Creating a Lamp and Adding Dynamic Lighting.

When you add a light to the Light component, the following properties are available:



Property	Description
light_mask	This is the light mask's texture resource. You must first add the texture as a local resource. When you export from Maya with a light locator the resource is automatically created and allocated. If you want to use another resource, add a new local resource, and then it appears in the light_mask drop-down list.
offset	This is the offset from the object's origin for the light locator.

Lua Environment Component

Registers a Lua environment as an integral part of an object. All Lua environments get subscription to the following libraries:

- All global functions
- BasicGenX
- Debug
- HttpPostData
- LocalPlayer
- MemoryContainer
- Model
- Object
- Person
- Resource
- Sound
- SoundBank
- SoundStream
- Texture
- Vector4
- Xml

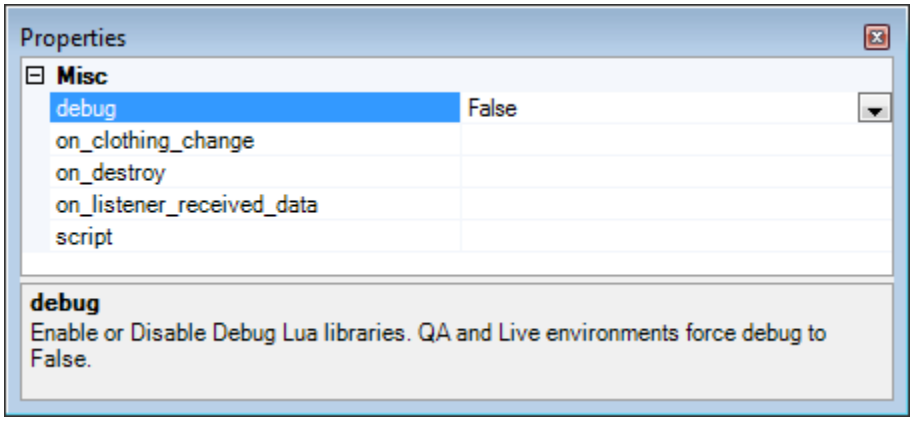
For more information on using the Lua Environment component, see:

- Adding Script Components
- Embed Objects into Scenes
- Debug Library and Public Library
- Defining Instance Parameters

Samples

- Games > Arcade Tutorial
- Games > Advanced Arcade Game

Properties



Property	Description
debug	This is a boolean that gives you access to the Debug Lua Libraries. This must be set to False when you package and submit the object for Quality Assurance (QA). For more information, see Debugging Lua Script using SLED.

on_clothing_change	Function to call when the user changes at least one character component (i.e. item of clothing). This function must be a direct call, that is, FunctionName.
	
	The client passes two lists of clothing changes (items added and items removed). The two lists are arrays of object IDs. The callback function needs to be of the form:
	
	`function OnClothingChange( added, removed )`
	`added = ( objID1, objID2, ... )`
	`removed = ( objID1, objID2, ... )`
	`end`

on_destroy	Gives the Lua function to be called on destroy.
on_listener_received_data	Gives the function to be called when receiving listener data.
script	Choose the script resource from the drop-down list. You must first add the Lua script as a local resource.

MiniGame Component

Mini-games are objects. Objects are constructed in the Object Editor and built from a number of components. The components provide the functionality that the game needs, be that pad support, network messaging, or OSD support. The only required component for a mini-game object is the MiniGame component. This component subscribes to the MiniGame library and related functions common to most interactive 3D content. It also registers an update/tick function.

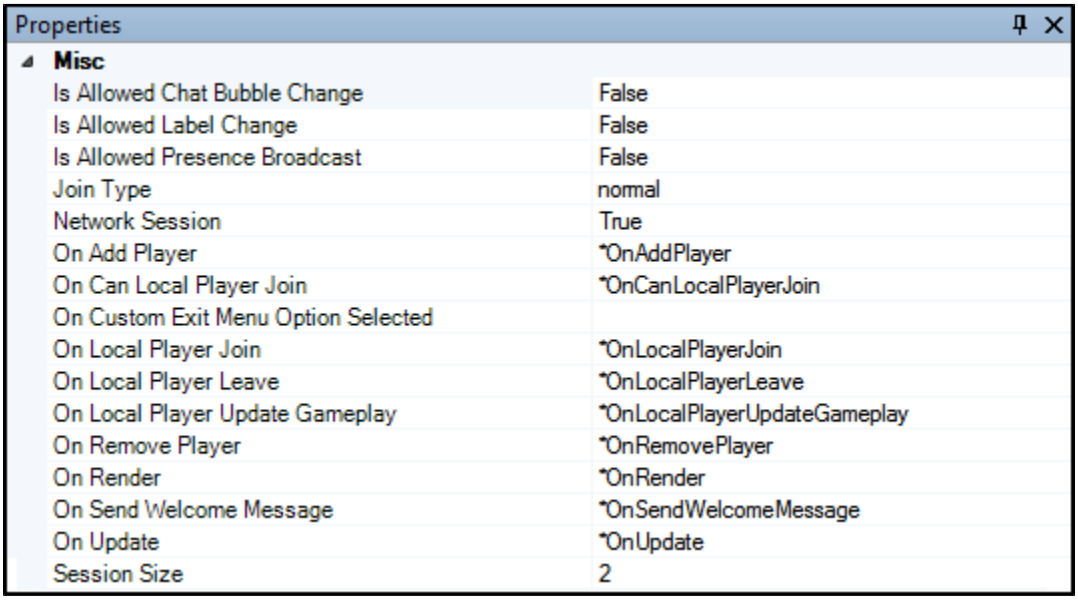
For more information on using the MiniGame component, see:

- Mini-games
- Adding Mini-game Components
- Mini-game Design
- Embed Objects into Scenes
- Scripted Assets Overview

Samples

- Games > Simon
- Games > Tic Tac Toe

Properties



If you add the MiniGame component to your mini-game object in the Object Editor, the mini-game acquires a number of properties, which you can set as required. For a list of mini-game properties, see Mini-games.

Network Component

Subscribes to the network framework with access to the following libraries:

- NetPropertyBag
- OutboundMessage
- ReceivedMessage

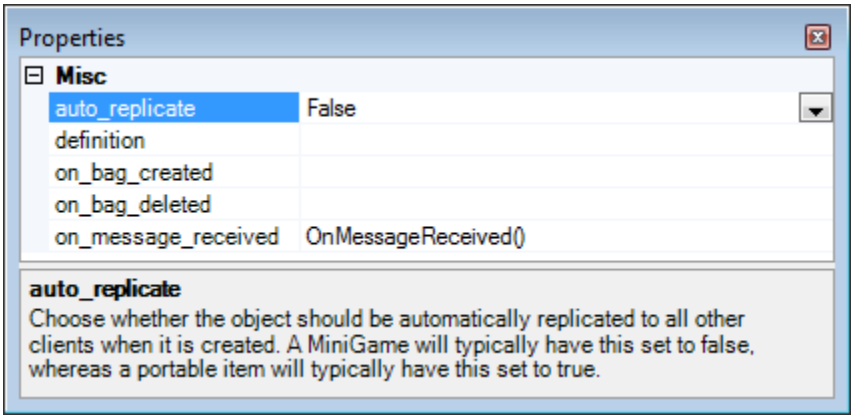
This allows your object to communicate between clients over a network.

For more information on using the Network component, see:

- Objects with Network Components
- Adding Network Communication
- Mini-games
- Realtime Games

Properties

The Network component properties are as follows:



Property	Description
auto_replicate	If set to true, when the object is created it replicates to all the clients. You would normally set this property to True for an inventory item (portable), such as the bubble machine. You would normally set it to False for a Mini-game.
definition	The XML file resource that contains the definitions of the netproperty bags.
on_bag_created	A user-defined Lua function that is called when a netproperty bag is created. It is also called when a user enters the scene.
on_bag_deleted	A user-defined Lua function that is called when a netproperty bag is deleted. It is also called when a user leaves the scene.
on_message_received	A user-defined function that is called on receipt of a network message sent from another connected client through the OutboundMessage.Send function.

OSD Component

Subscribes to the OSD framework with access to the following libraries:

- Osd
- OsdBasicChip
- OsdBasicSingleGraphic
- OsdBasicText
- OsdChipContent
- OsdObject
- OsdSelectionParent
- OsdTextLines
- OsdTextPanel

This component also lets you choose the osd .xml file resource.

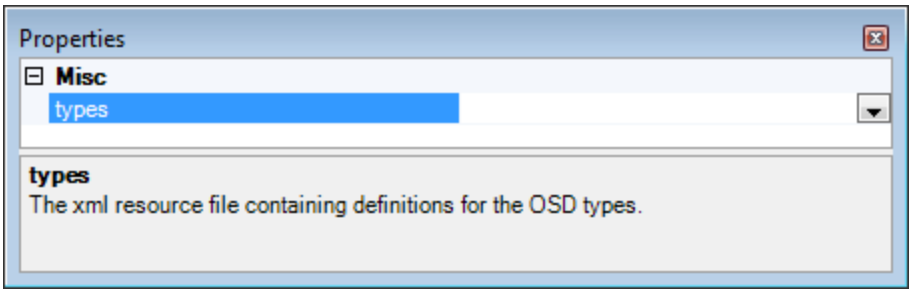
For more information on using the OSD component, see:

- Create Custom OSD Elements with XML
- Example - NP Ticketing Mini-game (uses OSD as part of the example)

Samples

- Systems > NP Ticketing

Properties



Property	Description
types	This is a drop-down list that lets you choose your OSD .xml resource. You must first add the resource to the object before it is available in the drop-down list.

Pad Component

Subscribes to Pad library for dealing with the pad button.

For more information on using the Pad component, see:

- The DUALSHOCK®3 Controller
- Handling User Input

Samples

- Games > Arcade Tutorial
- Games > Advanced Arcade Game
- Media > Media Library
- Systems > Pad API

Properties

The Pad component has no Object Editor properties.

Particles Component

Currently only environment effects are supported, not object particle effects.

Properties

The Particles component has no Object Editor properties.

RealTime Game Component

The RealTime Game component subscribes to the RTGame and RTGameObject libraries. This component also lets you choose the Realtime Gaming Realtime Network XML. Some of the RealTime Game component callbacks input arguments into the callback functions. For example, on_local_player_join inputs a memberId to OnLocalPlayerJoin(memberId). Those callbacks that input an argument are shown in the Realtime Game component's properties. The arguments are described in Callback Arguments.

For more information on using the RealTime Game Component, see Realtime Games.

Samples

- FPS Sample

Using the RealTime Game Component

You can only use the RealTime Game component with:

- Active Item components

- Scene Object components

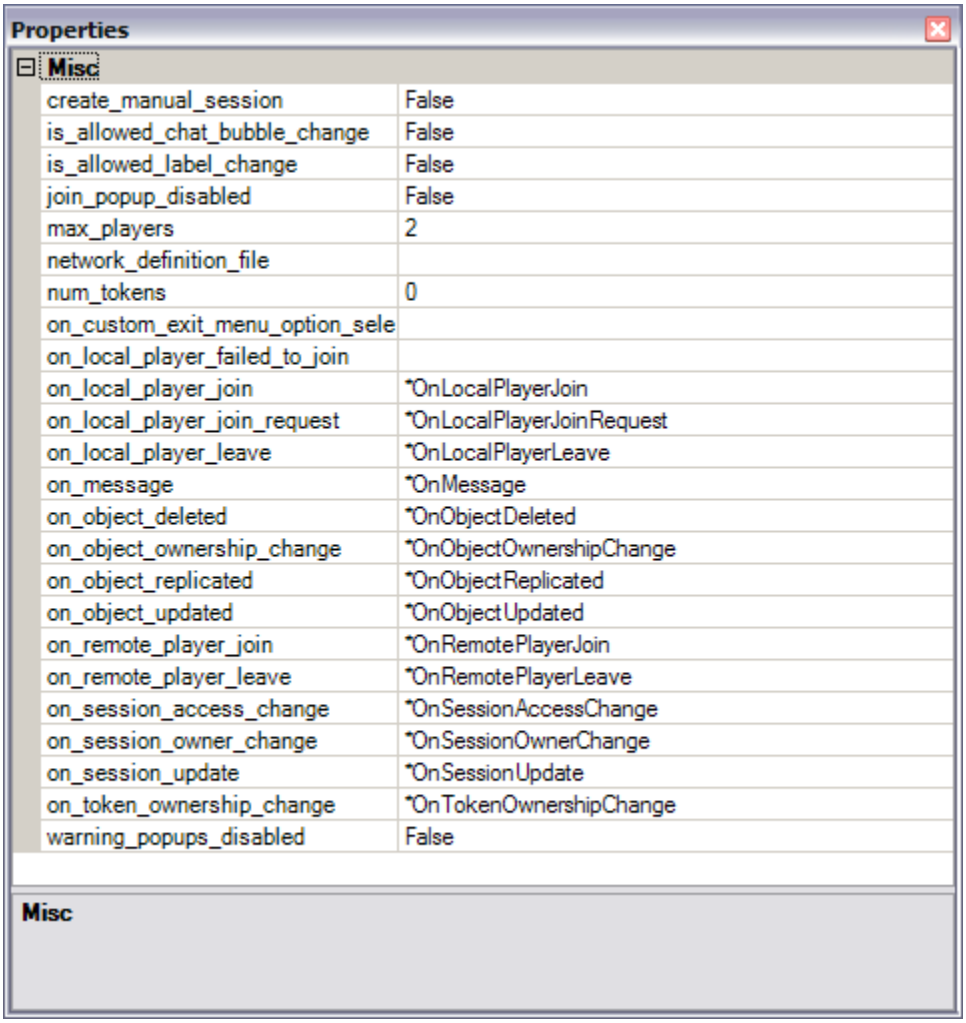
You cannot use a RealTime Game component with:

- Arcade Game components
- Character components

You can have up to 16 players in a realtime game.

Properties

The following properties are available:



Property	Description
create_manual_session	Determines whether you want the realtime game to support the manual creation of sessions (using `RTGame.CreateGame` and other related functions). The default value is False.
is_allowed_chat_bubble_change	Allows chat bubbles to be toggled on and off using the function `Person.EnableChatBubbles()`.
is_allowed_label_change	Allows the labels to be changed using the function `Person.SetLabelText()`.
join_popup_disabled	Determines whether pop-ups indicating that a user is joining the game are displayed. The default value is False (so the pop-up is displayed whenever a user joins the game). Set the value to True to disable the pop-ups. ⚠ If join_popup_disabled set to True, create_manual_session must also be set to True.

max_players	The maximum number of players that can participate in the game session. There is a system maximum of 16. Please note that for bandwidth intensive games, a maximum of 8 is recommended.
network_definition_file	The Realtime Network XML file resource. It contains definitions for network messages and network objects.
num_tokens	The number of tokens available to the realtime game session. The system maximum is 256.
on_custom_exit_menu_option_selected	Callback function when the user selects a custom menu option. Custom menus are created using `RtGame.AddCustomExitMenuOption()`.
on_local_player_failed_to_join	Callback function when a local player fails to join the game. `function OnLocalPlayerFailedToJoin( SessionError )`
on_local_player_join	Function called when a local player joins the game session. Session operations should not be performed until the on_local_player_join is triggered. Prior to this, it is possible for the on_remote_player_join to be triggered before the local player has joined, these are network events used to populate the session members list. So when on_local_player_join is triggered, the members list will be up-to-date and reflect all the members currently in the session. `function OnLocalPlayerJoin( memberId )`
on_local_player_join_request	Function called when a user attempts to join a game session. The function triggered by this callback must call `RtGame.ReturnJoinGameResult()` at some point. The callback function can perform extra processing/operations, and then decide if the local player can join, while performing the extra processing. By default, when a user joins the game, a pop-up will be displayed in PS Home, indicating that the joining is in progress. You can disable this pop-up using the join_popup_disabled property (see below). For more information, see the `RtGame.ReturnJoinGameResult` function in the Lua API Reference.
on_local_player_leave	Function called when local player leaves the game session.
on_message	Function called when a message is received. `function OnMessage( messageName, message, fromMemberId )`
on_object_deleted	Function called when an RTGame object is deleted from the game, either by the object's owner or the session owner. This callback is also called when a users leaves the game session. `function OnObjectDeleted( RTGameObject obj )`
on_object_ownership_change	Function called when the object's owner changes. This callback is triggered prior to the on_session_owner_change callback.
on_object_replicated	Function called when an RTGame object has been replicated to the network. This callback will be triggered for localled and remotely replicated RTGame objects. `function OnObjectReplicated( RTGameObject obj )`
on_object_updated	Function called when a field in a remote object is updated. `function OnObjectUpdated( RTGameObject obj, updatedFields )`

on_remote_player_join	Function called when a remote player joins the game session. `function OnRemotePlayerJoin( memberId )`
on_remote_player_leave	Function called when a remote person has left the session. `function OnRemotePlayerLeave( memberId )`
on_session_access_change	Function called when the Lua script calls the function RTGame.SetSessionClosed(). Callback is triggered only by the session owner. `function OnSessionAccessChange( isClosed )`
on_session_owner_change	Function called when the session owner changes. `function OnSesionOwnerChange( newOwnerMemberId )`
on_session_update	Once the local person has joined the game session, this callback will be triggered every frame. ⚠ Use another active or embedded object component's on_update callback to perform update processing before a player joins the realtime game.
on_token_ownership_change	Function called when a user claims or releases a token. This callback is also used to report if a token ownership request failed. A full refresh is triggered for the local user when first joining the game session and when the session owner changes. `function OnTokenOwnershipChange( tokenId, memberId )`
warning_popups_disabled	Determines whether warning pop-ups (caused by realtime game session errors) are displayed. The default value is False (so the warning pop-ups are displayed). Set the value to True to disable the pop-ups.

Callback Arguments

The following table describes the arguments that can be input to the callback functions:

Argument	Description
fromMemberId	The member ID of the player who sent the message.
isClosed	A boolean that indicates if the RTGame.SetSessionClosed() request was successful. Use this argument within the callback function to confirm that the request was successful. ⚠ Do not assume that the session is closed when the function RTGame.SetSessionClosed() is called.
memberId	The member ID of the player.
messageName	The name of the <message> as specified in the Network XML.

message	The Lua table message. See RTGame.SendMessage().
newOwnerMemberId	The member ID of the new session master.
obj	An instance of a RTGame Network XML object.
SessionError	An enumerated string saying why the local player failed to join. Possible values are: • SessionError.Closed: The session is closed. • SessionError.Full: The session is full. • SessionError.NotFound: Failed to find a session with the room ID. • SessionError.TimedOut: Create or join session timed out. • SessionError.Denied: Join failed due to false result of script callback OnLocalPlayerJoinRequest. • SessionError.NonFatal: Join/create failed - recommend for user/script to try again, after short retry period. • SessionError.Fatal: Unknown, unrecoverable error. Retry will not succeed.
tokenId	The token ID.

updatedFields	A Lua table of (fieldName, boolean) pairs. You can query the table for the existence of a particular field. For example:
	`if updatedFields.myFieldName ~= nil then`
	`  --do something`
	`end`

Renderer Component

Subscribes to the renderer framework for dealing 2D rendering support in full screen and gains access to libraries:

- Renderer
- Sprite
- SpriteAnim

Samples

- Games > Arcade Tutorial
- Games > Simon
- Media > Media Library
- Systems > OSK API
- Systems > Pad API

Properties

The Renderer component has no Object Editor properties.

Repertoire Component

The Repertoire component allows the object to access the Repertoire Editor, thus allowing you to add custom avatar animations. You can add the Repertoire component to only mini-games, realtime games and FullBodySuits.

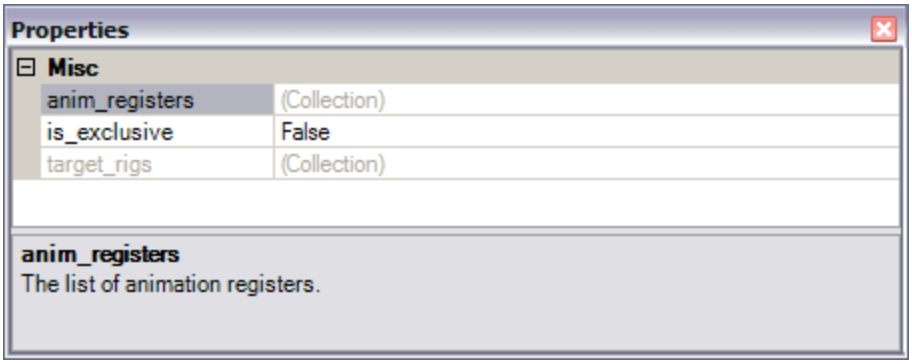
You can set the following properties for the Repertoire component:

- Target Rigs
- Animation Registers

For more information on using the Repertoire component, see:

- Repertoire Editor
- Defining Animation Registers

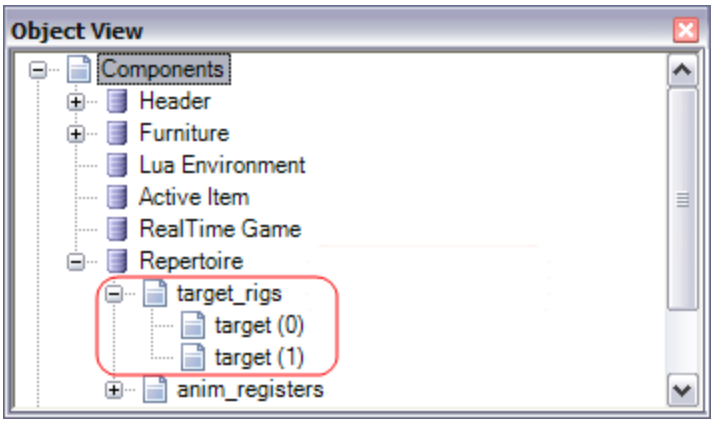
Properties



Property	Description
anim_registers	This property is automatically updated when you use the Repertoire Editor.
is_exclusive	Sets the object exclusive to one gender. It must be False for Realtime games and Mini-games.
target_rigs	This property is automatically updated when you use the Repertoire Editor.

Target Rigs

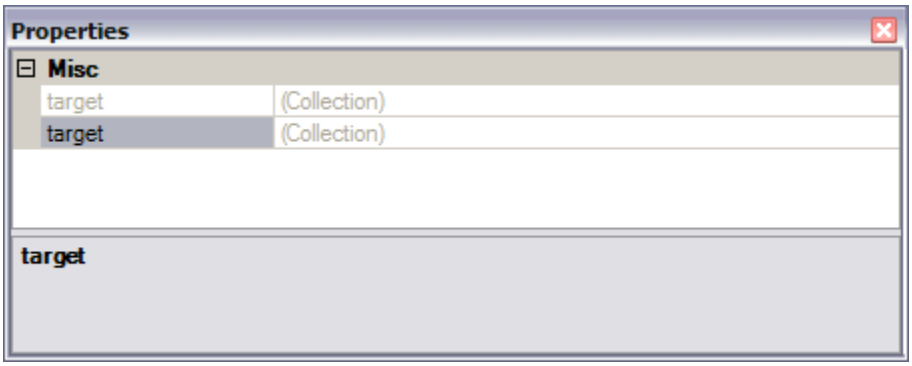
This is the repertoire component's daughter component. Each new repertoire that you create through the Repertoire Editor is displayed under the target_rigs tree.



target (0) is the first repertoire that you create for your FullBodySuit, realtime, or mini-game. For each repertoire that you create through the Repertoire Editor, a new target daughter component is created. The target daughter component is numbered, starting with target (0), and increasing sequentially. For example, if you have three repertoires, the targets will be target (0), target (1), target (2).

target_rigs Daughter Component Properties

When you select target_rigs, the Properties panel displays the following information:

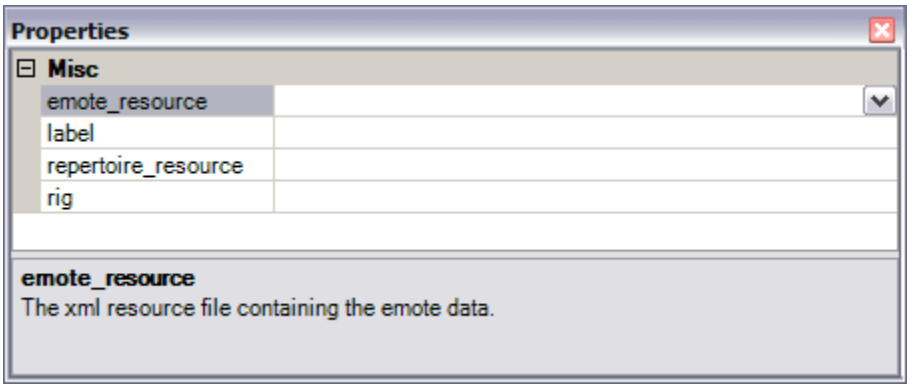


Property	Description

target	This property cannot be edited. This automatically updates when you use the Repertoire Editor.

Target Daughter Component Properties

When you select any of the targets in the target_rig tree, the Properties panel displays the following information:



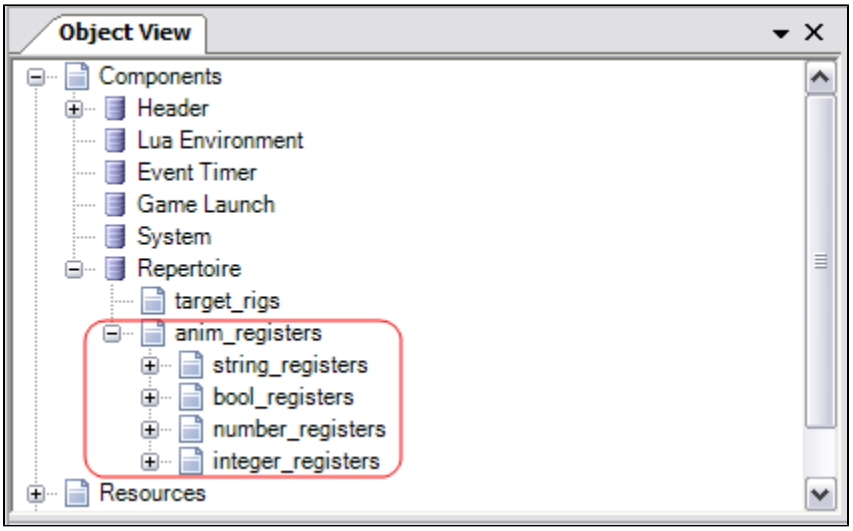
> ⚠ Do not edit any of these properties manually. They should update automatically when you use the Repertoire Editor.

Property	Description
emote_resource	The location for the emote xml file. The xml file will be automatically allocated here when you add emotes through the Repertoire Editor.
label	The label that you give to the repertoire in the Repertoire Editor. This is useful only for mini-games and realtime games, since you can call the label in your script.
repertoire_resource	The repertoire allocated to this target ( ) component.
rig	The UUID of the animation rig affected by the repertoire data.

Animation Registers

Use animation registers as control variables to give you more control over how animation states are selected for Repertoire Animation Blending. You can also use animation registers to output a value through a register to be read by a Lua script.

When you add a Repertoire component to a mini-game or realtime game, the following animation registers are created automatically:



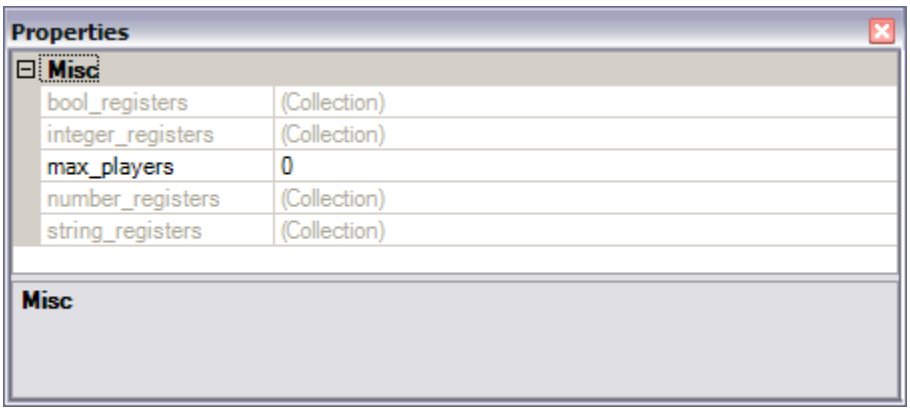
The following types of animation registers are available:

Register	Description

string_registers	A string or list of strings. For each repertoire that you create through the Repertoire Editor, a new target daughter component is created. The target daughter component is numbered, starting with register(0), and increasing sequentially. For example, if you have three repertoires, the targets will be register (0), register (1), register (2).
bool_registers	A boolean value. For each repertoire that you create through the Repertoire Editor, a new target daughter component is created. The target daughter component is numbered, starting with register(0), and increasing sequentially. For example, if you have three repertoires, the targets will be register (0), register (1), register (2) .
number_registers	A floating point range specified by a minimum and maximum value. For each repertoire that you create through the Repertoire Editor, a new target daughter component is created. The target daughter component is numbered, starting with register(0), and increasing sequentially. For example, if you have three repertoires, the targets will be register (0), register (1), register (2).
integer_registers	An integer range specified by a minimum and maximum value. For each repertoire that you create through the Repertoire Editor, a new target daughter component is created. The target daughter component is numbered, starting with register(0), and increasing sequentially. For example, if you have three repertoires, the targets will be register (0), register (1), register (2).

anim_registers Target Daughter Component Properties

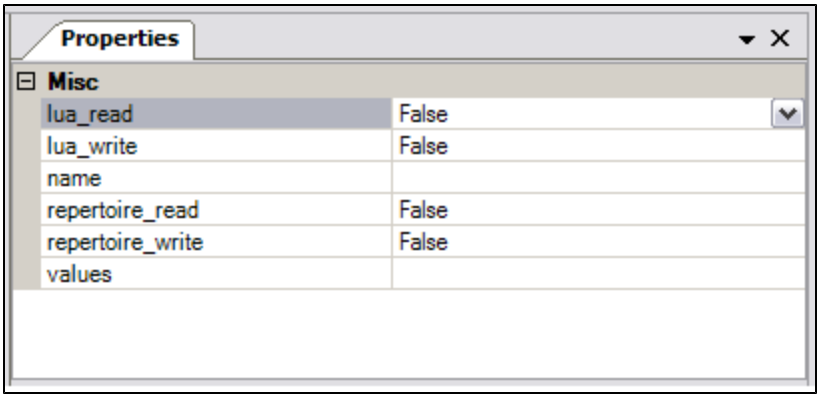
When you select anim_registers, the Properties panel displays the following information:



Property	Description
bool_registers	This property cannot be edited. This automatically updates when you use the Repertoire Editor.
integer_registers	This property cannot be edited. This automatically updates when you use the Repertoire Editor.
max_players	You must specify the number of sets for the animation registers. Each avatar requires one set of animation registers, so that the object knows how many avatars will use the object (i.e. the number of players in the mini-game). For a sessionless one-player mini-game, multiple players can play the game concurrently; set max_players to the number of players that will be able to use the game simultaneously. Note that scene capacity restrictions apply.
number_registers	This property cannot be edited. This automatically updates when you use the Repertoire Editor.
string_registers	This property cannot be edited. This automatically updates when you use the Repertoire Editor.

string_registers Daughter Component Properties

When you select any of the registers in the string_registers tree, the Properties panel displays the following information:

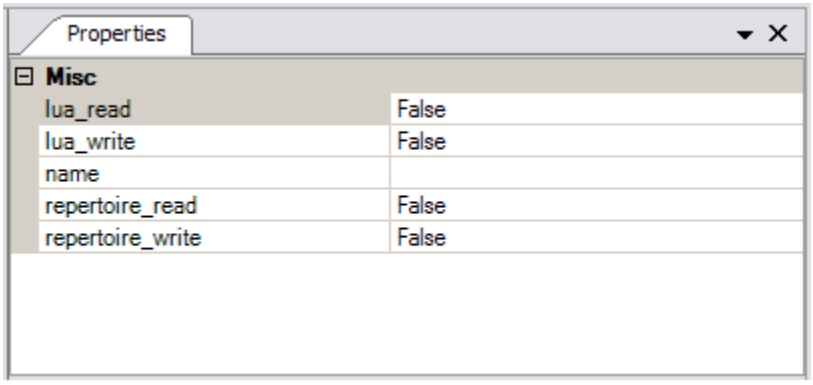


You can set the following properties for the string_registers register:

Property	Description
lua_read	If the animation register can be read from Lua.
lua_write	If the animation register can be written to from Lua.
name	The name of the animation register.
repertoire_read	If the animation register can be read by the repertoire.
repertoire_write	If the animation register can be written to by the repertoire.
values	The valid values for the animation register.

bool_registers Daughter Component Properties

When you select any of the registers in the bool_registers tree, the Properties panel displays the following information:

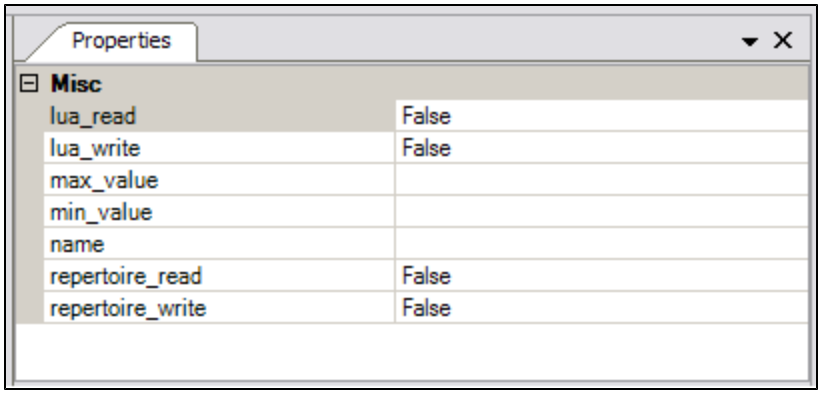


You can set the following properties for the bool_registers register:

Property	Description
lua_read	If the animation register can be read from Lua.
lua_write	If the animation register can be written to from Lua.
name	The name of the animation register.
repertoire_read	If the animation register can be read by the repertoire.
repertoire_write	If the animation register can be written to by the repertoire.

numbers_registers Daughter Component Properties

When you select any of the registers in the number_registers tree, the Properties panel displays the following information:

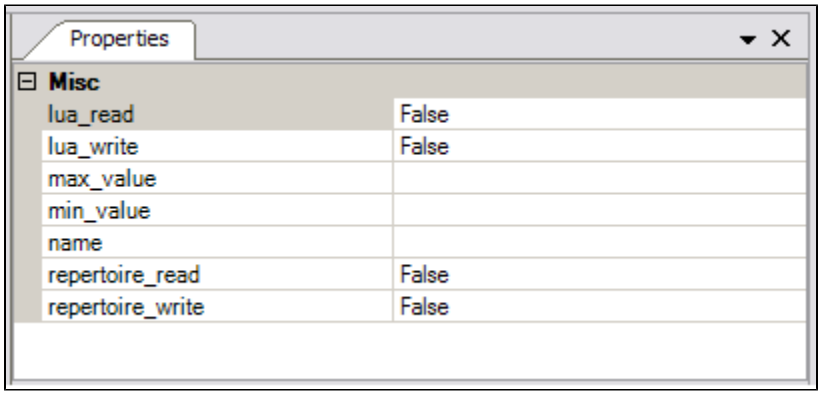


You can set the following properties for the numbers_registers register:

Property	Description
lua_read	If the animation register can be read from Lua.
lua_write	If the animation register can be written to from Lua.
max_value	The maximum value of the animation register.
min_value	The minimum value of the animation register.
name	The name of the animation register.
repertoire_read	If the animation register can be read by the repertoire.
repertoire_write	If the animation register can be written to by the repertoire.

integer_registers Daughter Component Properties

When you select any of the registers in the integer_registers tree, the Properties panel displays the following information:



You can set the following properties for the integer_registers register:

Property	Description
lua_read	If the animation register can be read from Lua.
lua_write	If the animation register can be written to from Lua.
min_value	The minimum value of the animation register.
max_value	The maximum value of the animation register.
name	The name of the animation register.
repertoire_read	If the animation register can be read by the repertoire.
repertoire_write	If the animation register can be written to by the repertoire.

Resource Pack Component

A Resource Pack is a container object for other scripted objects. The scripted objects can load and use any resource attached to the resource pack.

For more information on using the Resource Pack component, see Resource Packs.

Properties

Unable to render embedded object: File (ResourcePackComponent.png) not found.

Property	Description
is_resource_pack	A flag indicating if the object is a Resource Pack or uses Resource Packs. If the object is a Resource Pack, set to True. If the object uses Resource Packs, set to False.
objects	If the object is a Resource Pack, enter the object ID for each object that uses this Resource Pack. If the object uses Resource Pack, enter the object ID for each resource pack that it uses.

Scene Object Component

A scene object is an object taken from the Object Editor and embedded in a scene. This component gives the additional benefit of setting instance parameters. This feature enables you to set your own custom parameters for the object and then set those parameters for each instance of the object in the scene.

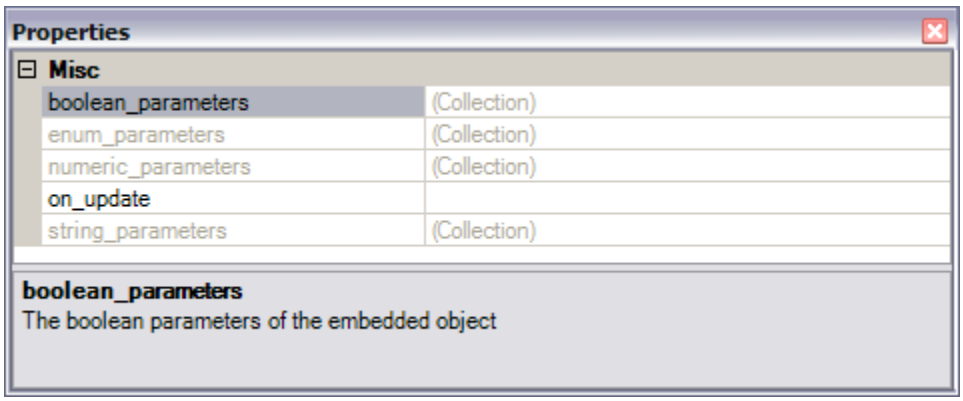
> Only objects with the Scene Object component have access to this feature, so even if you add an object as an scene object within your scene, it does not have access to instance parameters unless you add an scene object component to that object. See Creating a Scene Object.

Samples

- Arcade Tutorial
- Advanced Arcade Game

Properties

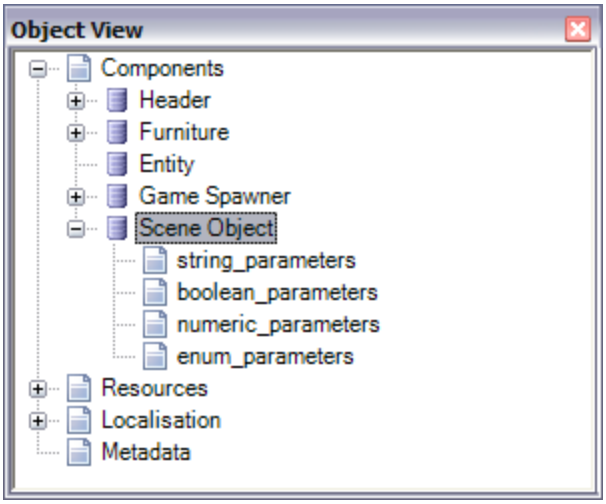
When you select the Scene Object component in the Object View panel, the Properties panel displays the following information:



Property	Description
on_update	This is the only property that you can edit. The property allows you to input the default Lua function to call at an on_update event.

Parameters

You can add a number of additional properties to each scene object, to define the instance parameters available in the Scene Editor:



The following property types are available:

Property	Description
string_parameters	Defines an initial string value.
boolean_parameters	Defines a boolean and its initial value.
numeric_parameters	Defines a numeric variable in your Lua script, its default value, and the range of possible values.
enum_parameters	Defines an enum variable, the default value, and a list containing the possible values.

Screen Component

Subscribes to the Screen library for rendering media onto 2D surfaces. See Introduction to Screens and Deploying an Arcade Game in Furniture. This does not explain the Screen component, but uses the Screen library, and explains the Arcade Tutorial samples.

Samples

- Arcade Tutorial 1
- Arcade Tutorial 2
- Arcade Tutorial 3

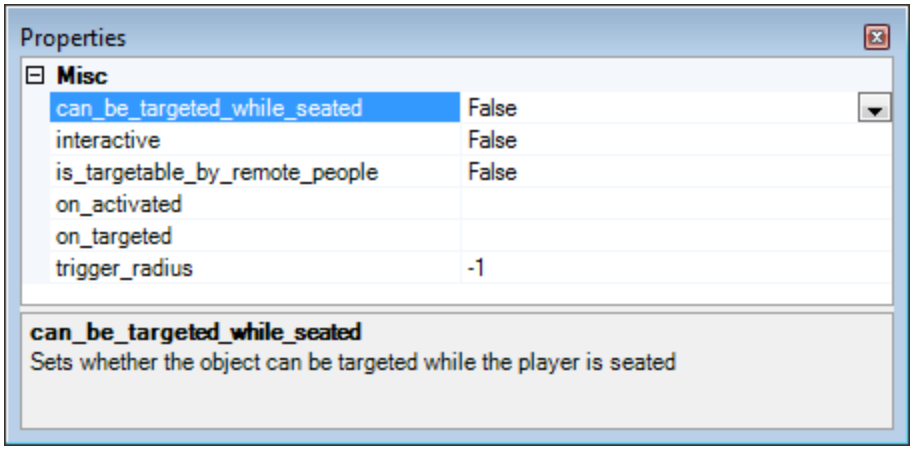
Properties

Screen component has no Object Editor properties.

Targetable Component

The Targetable component allows an object to be targeted and activated by a user. It registers a Lua function to call when activated.

Properties



Properties	Description
can_be_targeted_while_seated	Specifies whether the object can be targeted while the player is seated.
interactive	Specifies whether or not the target is interactive.
is_targetable_by_remote_people	Specifies whether the object can be targeted by the owner only (local player) or by all remote users as well.
on_activated	Sets the callback used when the targetable object is activated by a player.
on_targeted	Sets the callback used when the targetable object is targeted by the player.
trigger_radius	Sets the trigger radius. By default it is -1.

Managing Object Resources

Resources in objects can be any file or raw content type. Although you can add any type of resource to an object, the table below lists some of those currently supported in the HDK:

Extension	Type	Description
.ani	Home Animation File	Exported animation from DCC package
.bnk	Sound Bank	Audio Binary Bank file created using SCREAM
.dds	Texture	DXT 1, 3, 5 Type DDS image (with or without alpha)
.efx	Home Particle Effect	Particle effect created with the HDK Particle Effect Tool
.hkx	Collision File	Exported collision from DCC package
.lua	Lua Script	Lua script file containing Scene/Environment logic
.mdl	Home Model File	Exported model from DCC package
.mp3	Sound Stream	Generic MP3 audio stream
.skn	Home Skeleton File	Exported skeleton from DCC package
.xml	XML File	General-purpose extensible mark-up language file

To find out more about how to use object resources, see:

- Adding and Deleting Object Resources
- Overriding Object Resources
- Recovering Object Resource Files

> ⚠ For all objects created before HDK 1.3.5, you must follow the steps in Updating Object Lua Resources.

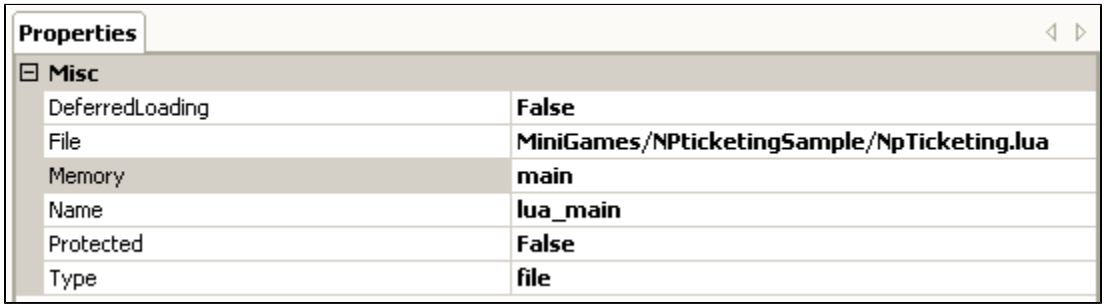
Adding and Deleting Object Resources

Adding an Object Resource

You can add resources to single or multiple objects. See Editing Multiple Objects. You can add any supported file type to your object as a resource. For a list of supported types, see Object Component Properties.

To add a resource:

1. Select Object > Add New Local Resource. The Add dialog is displayed.
2. Select the required file and then click Open.
3. Set the properties for the resource, as shown in the following example:



All resources have the following properties:

Property	Description
DeferredLoading	A boolean that toggles if the resources are deferred loaded. When set, the PS Home client defers loading resources until they are actually used. This option is often set automatically by the HDK DCC Tools.
File	The filename of the resource.
Memory	A resource type override for changing the resource type from main or host. If a file uses host memory by default you cannot change it to use main. By default, it will show main whether or not the default memory type is host (for example, .mdl files).
Name	The name of the resource.
Protected	A boolean to provide extra security to 'file' and 'lua' type content. Lua scripts by default are set to protected and must be packaged with protection set to true before you submit content to SCE QA.
Type	Sets the type of the resource (for example, model, lua, xml, soundstream).

Deleting an Object Resource

You can delete an object resource from the Object View or HDK Project panel.

> Deleting an object resource from the HDK Project panel simply deletes it from the project, as a reference.
> Deleting an object resource in the Object View panel deletes the file and removes it from the object.

Deleting an Object Resource from the Object View Panel

You can delete resources from objects only one at a time. You cannot delete resources from multiple objects.

To delete a resource:

1. In the Object View panel, expand the Resources folder and select the required resource.
2. Select Object > Delete Resource.

   The object is removed from the Object View panel.

Deleting an Object Resource from the HDK Project Panel

To delete a resource:

1. In the Object Editor, navigate to the required object and expand the Resources folder.
2. Right-click the resource you want to delete and select Remove from Project.

   The object is removed from the HDK Project panel.

To recover a deleted object resource see Recovering Object Resource Files.

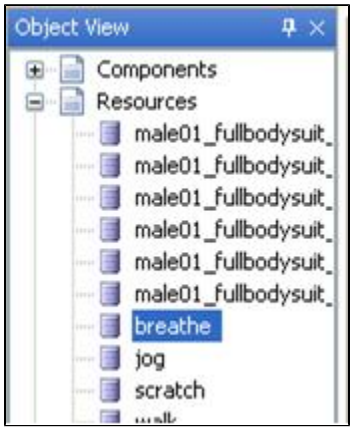
Changing the Resource Memory

All resource files (except texture, model and particle effects) use MAIN memory by default. You can switch to HOST memory using the Object Editor.

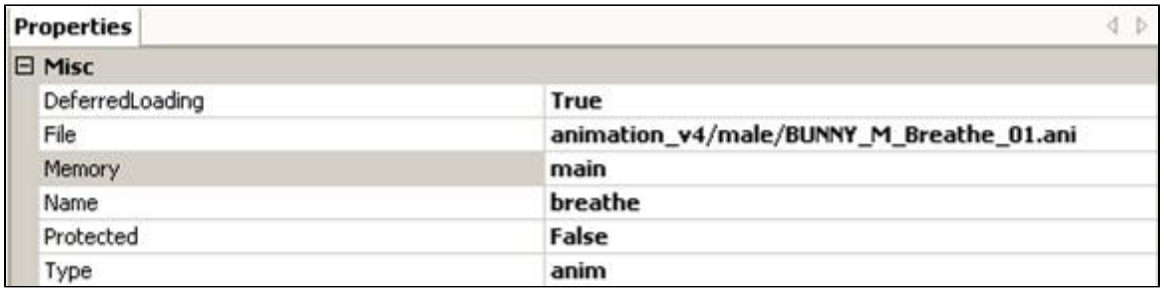
> Resource Memory Overrides for textures, models and particle information moves only the header information. Models are always in HOST, and textures are always in VRAM.

To change the type of memory that your resource uses:

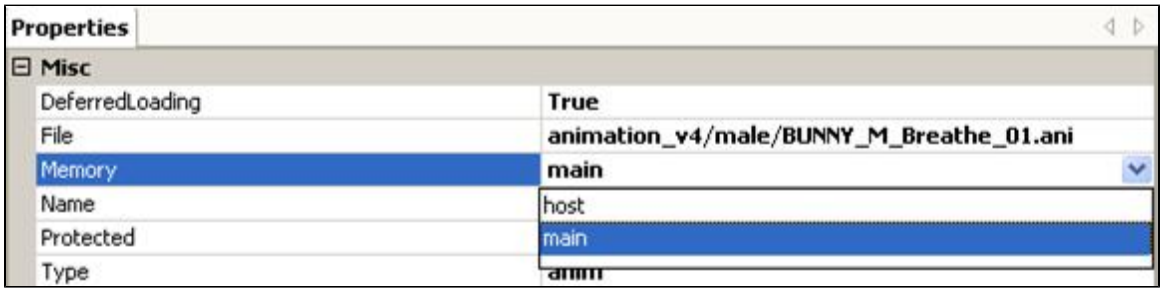
1. Select the resource file in the Object View panel:



2. Go to the Properties panel and select the Memory option:



3. Select the type of memory you want your resource to use from the Memory drop-down list:



The default memory types for the files are as follows:

Extension	Default Memory Type
.ani	main
.bnk	main
.dds*	VRAM

.efx	host**
.hkx	main
.jpg	main
.lua .luac	main
.mdl	host**
.mp3	main
.png	main
.skn	main
.xml	main

*Most of the texture data uses VRAM no matter what Memory setting you use in the Object Editor. Some small elements of a texture are not stored in VRAM. These few elements are affected by this setting.

**Changing the memory type from HOST to MAIN moves only the header data to MAIN. The rest of the resource continues to use HOST memory.

Viewing Resource Information

The Viewer panel can display 3D models, images and files, including the following file types:

- .mdl files
- .dds files
- .png files
- .xml files with write access
- .lua files with write access
- Some repertoire files

> You can save files which you have write access to.

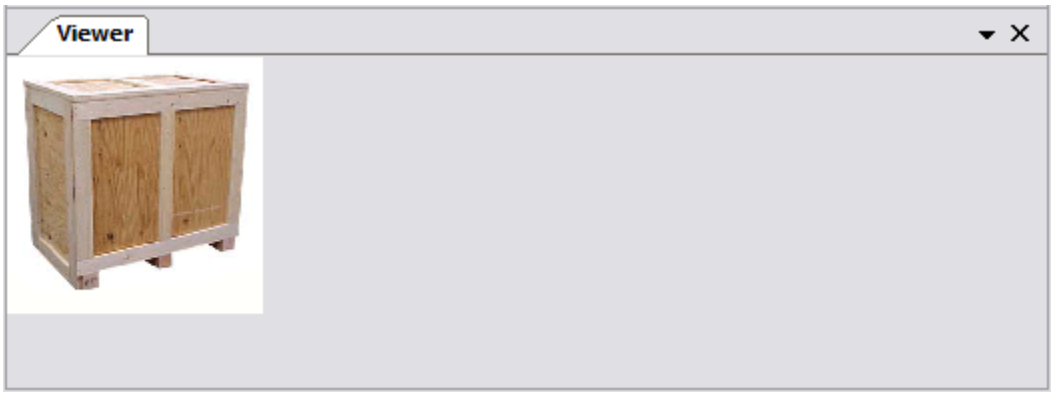
You do not have access to save the following files and file types:

- editor.oxml
- catalogueentry.xml
- object.xml
- object.odc
- validation.xml
- validations.xml
- gamelaunchoptions.xml
- Files with the following extensions: .luac, .skn, .bnk, .ani, .efx, .safevolume.hkx, .hkx, .safevolume.hkx.xml, .atgi, .hkx.xml, .ma

To view information for a resource in the Viewer panel:

1. Navigate to the object resource in the HDK Project panel or Object View panel.
2. Select the required resource, for example, woodencrate.png.

   The Viewer displays the resource as shown in the following example:



Overriding Object Resources

You can apply a Resource Override for resources that should not be used under certain circumstances, for example, based on age, parental, or regional restrictions. You swap the resource for another resource based on the user's console settings. This can be used to create regional differences in an object or to deny access to certain resources if the user is under-age (for example if the user is under 15 years old and in Germany, then an age rating and regional override can be used to automatically replace a model file).

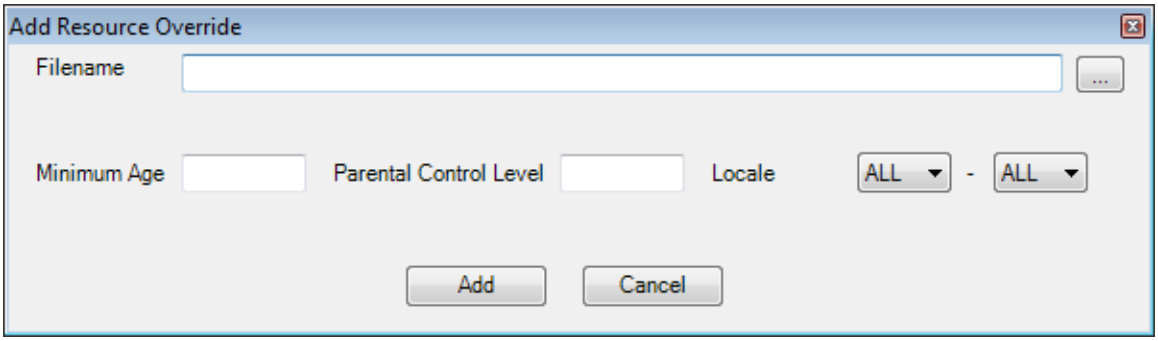
> ⚠ The Properties panel in the Object Editor contains a field called Memory, which you can use to change a resource type from main or host. If a resource has a default memory type of host, you cannot change it to main. The Memory field shows main whether or not the default memory type is host (as in the case of .mdl files).

Overriding a Resource

To override a resource:

1. In the Object View panel, select Resource.
2. Select Object > Add Resource Override.

   The Add Resource Override dialog is displayed:



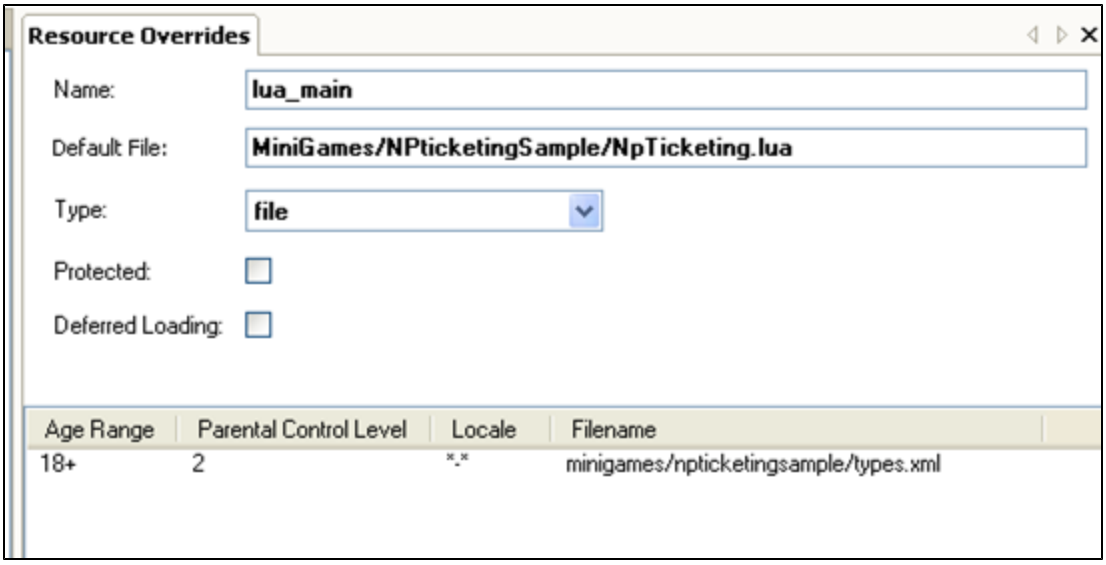
3. Click the button next to the Filename field to browse to and select the file to use as the default resource. If the conditions of the override are met, the original resource is automatically replaced by this default resource (for example, if the resource is a model file of a sofa, then the filename would be another model file that would automatically replace the original sofa). It must be of the same type as the selected resource.

4. If required, enter an age limit in the Minimum Age field (between 0 and 100). If the user's console setting is below the age entered here, the original resource is automatically replaced by the default resource (e.g. if a user needs to be 18+ years old to see a particular texture, then the minimum age would be 18).

5. If required, enter a parental control level in the Parental Control Level field (between 1 and 11). If the user's console setting is below the control level entered here, the original resource is automatically replaced by the default resource (e.g. if a user needs a console set to parental control level 10 or higher, then the value here would be 10).

6. Select ALL from both Locale drop-down lists if the override is to apply to all countries, or select a specific country. If the country code on the user's console matches the locale setting on the override, then the original resource is automatically replaced by the default resource (for example, if a user is in the US and the locale is set to en-* then the resource override is applied). Any combination of region and country can be used (for example, *-FR, en-* or -).

7. Click Add to create the resource override. The resource overrides are displayed in the Properties panel:



In the above example, the main.lua script is replaced by main2.lua if the user is under 15 years old in the US. If the user is under 12 in Spain then main.lua is replaced by main3.lua. If the user is under 12 everywhere else in the world, then main4.lua replaces main.lua.

Override Priority

The order of the overrides for each resource determines which has priority, for example, the first override has priority over all the others. Each override is checked in the order they are listed, and once the conditions are met no further checking is done. In the example above, the override for Spain is listed above ALL to ensure that users in Spain have a different override to those in the rest of the world.

To change the order, drag and drop an override to the position required.

Editing a Resource Override

To edit a resource override:

1. Right-click on an existing override and select one of the following options:
    - Add Resource Override: This is the same option as Add Resource Override on the Object menu. Use this to add a new override.
    - Delete Resource Override: Deletes the selected override.
    - Edit Resource Override: Opens the Add Resource Override dialog so you can edit the existing settings.
2. Click OK to apply your changes.

> ⚠ The default resource used to replace the existing resource in the override should not be added to the Resources of an object. Adding it as the filename in a resource override is enough to add it to the object on packaging.

Recovering Object Resource Files

You can recover missing object resource files from the HDK Project panel.

Use-cases

Scenario 1

User A's project has an object with five resources, User A packages the object and sends it to User B. User B modifies the object, deletes one of the resources, Zips up the object and sends it back to User A.

User A's local object still has five resources attached to it, even though the copy of the object received from User B has four. User A can use the Show Resources option to retrieve the missing resource file and re-add it.

Scenario 2

User A zips and sends an HDK project that contains an object with five resources to User B. User B unzips and opens the project

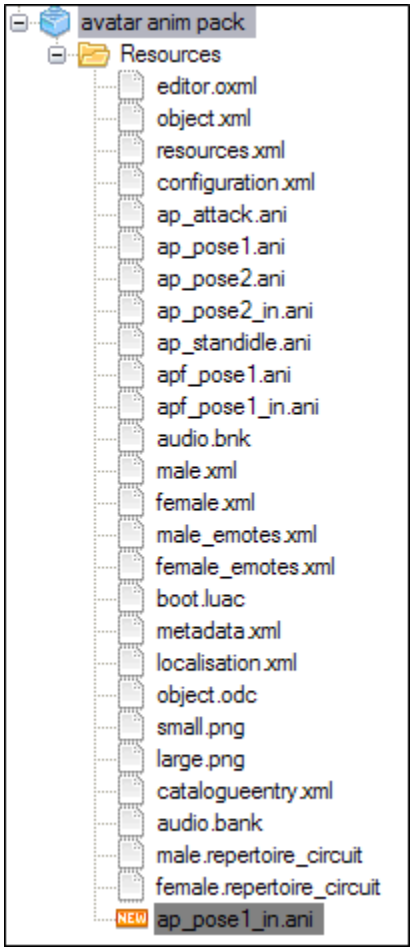
but notices that the object resources are not displayed. User B can use the Show Resources option to retrieve the missing resource files and re-add them.

Recovering a Missing Resource File

To recover a missing resource file:

1. In the HDK Project panel, navigate to the object you require.
2. Right-click the object and select Show Resources.

   The missing file is displayed as shown in the following example:



3. To add the resource file to your project, right-click the resource file and select Add to Project. To add multiple files, right-click the Resources folder that contains the missing files, and select Add all to project.

   The missing files are displayed in the Resources folder.

Localizing Objects

The Localization section of your object contains information on the default language for your object, and all the textual localization for your object, in multiple languages.

Localization data can be referenced:

- Internally by the Object Editor - for example, as references that contain object descriptions and display names (see Completing the Object Header)
- Externally, as an itemized list of strings and text that are likely to appear in actual content, such as in captions or headings in a Lua-controlled environment

You save all localization data to the object_master.xml file in the dev\localisation folder.

Localization Properties

You can use string tags to define localization properties, such as text that is likely to appear throughout your content differently depending on region, or descriptive identifiers that bear identical references to the same thing in different languages. The REF property is a compulsory part of every localization item that remains absolute yet internal to the Object. The HDK currently supports the following languages:

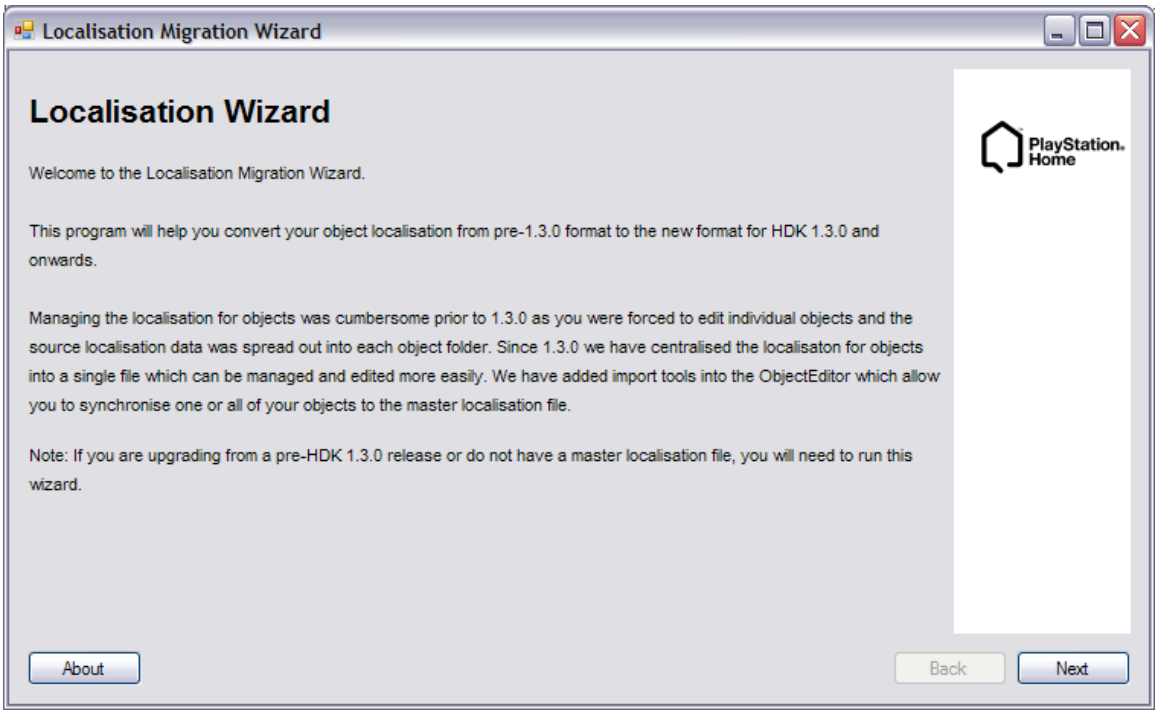
- de-DE (German-Germany)
- en-GB (English-United Kingdom)
- en-ID (English-Indonesia)
- en-MY (English-Malaysia)
- en-TH (English-Thailand)
- en-US (English-United States)
- en-SG (English-Singapore)
- es-ES (Spanish- Spain)
- fr-FR (French-France)
- it-IT (Italian-Italy)
- ja-JP (Japanese- Japan)
- ko-KR (Korean-Korea)
- zh-HK (Chinese-Hong Kong)
- zh-TW (Chinese-Taiwan)

Creating an object_master.xml File

The object_master.xml file contains all localization information for every object in your <HDK_INSTALL_DIR>/dev folder. The object_master.xml file produces a localisation.xml file for each object. To edit localization information, you must edit the object_master.xml.

To create an object_master.xml file:

1. Select Home Development Kit <version_number> > Localization Migration Wizard, to display the Localization Migration Wizard:



2. Click Next and follow the steps in the wizard.
3. Click Finish to create the object_master.xml file.

The master localization file stores all the current data in the localization files for each object.

Updating localization data

To update localization data:

1. Edit the master file.
2. Import the new localization information to the object in the Object Editor.

Editing the Master File

You can localize objects in one of two ways:

- Output Localization Strings to the master file: Use a pipeline and custom software to output your localization information into an object_master.xml using one of the following methods:
    - Use the utility provided (<HDK_INSTALL_DIR>\dev\localisation\TsvToMaster.bat) to convert TSV (tab separated values) format files to the object_master.xml file in the correct schema. See TSVToMaster.bat below.
    - Use your own software to write localization information and output your localization strings to the object_master.xml file. Make sure that you use the correct schema. See Master Localization Schema.
- Edit the object_master.xml file: You can edit the object_master.xml localization file manually.

> ⚠ This method is suitable only for small changes.

Importing Localization Information in the Object Editor

You can import localization information from the object_master.xml to an individual object, or to all objects.

To import the localization information in the Object Editor:

- To update a single object, select Object > Import Localization or click  .
- To update every object in your dev folder, select Object > Import All Localization, or click  .

Choosing a Default Language Setting

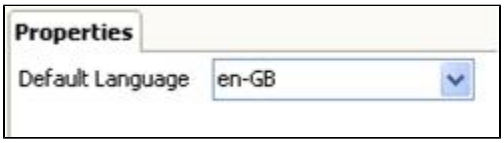
In instances where an object has no localization data for a region, the object uses the default language setting. You can choose which language to use as default.

To choose a default language setting:

1. In the Object View panel, select Localisation:



2. In the Properties panel, select the required default language from the Default Language drop-down list:



> ⚠ Make sure that you select the appropriate default language to avoid unlocalized resource messages appearing in your object.

Deleting Localization Items

You cannot delete localization items directly from the Object Editor. You must remove the localization item from the object_master.xml file and then re-import the localization information.

TSVToMaster.bat

TSVToMaster.bat is used to update the object_master.xml file based on the content of a TSV file. A TSV file is a file containing tab separated values.

To use the TSVToMaster.bat, drag the TSV (tab separated values) file onto the .bat file. This outputs the object_master.xml file in the <HDK_INSTALL_DIR>\dev\localisation folder.

> ⚠ Make sure that the object_master.xml file is closed before you run TSVToMaster.bat.

TSV Format Specification

- The file must be in UTF-8/16 with BOM.
- The first line must be this header: UUID REF en-GB fr-FR de-DE it-IT es-ES ja-JP ko-KR en-SG zh-HK zh-TW en-US
- Each line thereafter represents an item of localization whose data must align with the header description.

Master Localization Schema

The master localization file (object_master.xml) should use the following schema:

```
<GROUP name="00000000-00000000-00000000-000000A2" type="object">
   <PHRASE>
     <LANG ID="REF">example reference name</LANG>
     <LANG ID="en-GB">example localized data</LANG>
   </PHRASE>
</GROUP>
```

The following table describes each tag in the schema:

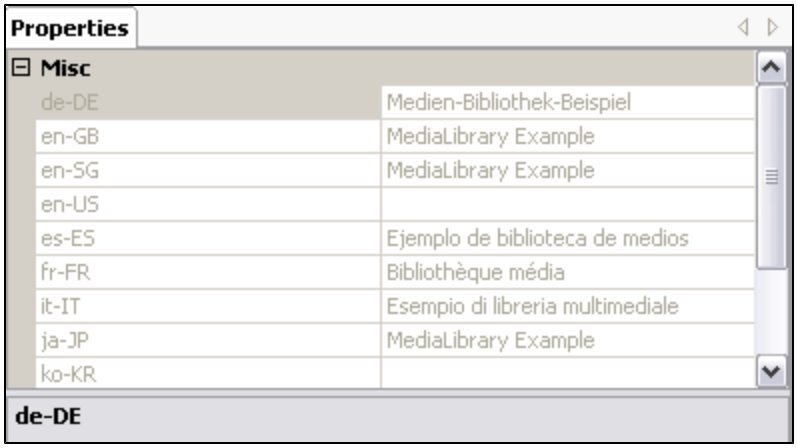
Tag	Description
<GROUP>	Each object is contained within a GROUP tag. Include the object GUID within the name parameter.
<PHRASE>	Contain each localization item in a PHRASE tag. For example, each object should have a name item and a description item defined in the Header component. Therefore each object often has at least two PHRASE tags, one containing the name and the other containing the description.
<LANG ID="REF">	Each localization item must have one <LANG ID="REF">. The ID="REF" parameter specifies that the data in this tag is used for the reference name of the localization item. The "REF" property is the absolute reference to the object. Developers can use this reference in script functions that need to call a particular localization.
<LANG ID=>	Use one <LANG ID=> for each language that the language will be translated into. The ID parameter should be the language code for the translation language. For example, for UK English the parameter is <LANG ID="en-GB">.

Localization Example

The following sample code shows an example of the localization for the MediaLibrary object supplied with the HDK:

```
<GROUP name="AE6FF8E2-7A2D4EA1-A9405FC3-286EF8F5" type="object">
   <PHRASE>
      <LANG ID="REF">Name</LANG>
      <LANG ID="en-GB">MediaLibrary Example</LANG>
      <LANG ID="fr-FR">Bibliothèque média</LANG>
      <LANG ID="de-DE">Medien-Bibliothek-Beispiel</LANG>
      <LANG ID="it-IT">Esempio di libreria multimediale</LANG>
      <LANG ID="es-ES">Ejemplo de biblioteca de medios</LANG>
      <LANG ID="ja-JP">MediaLibrary Example</LANG>
      <LANG ID="en-SG">MediaLibrary Example</LANG>
      <LANG ID="zh-HK">MediaLibrary Example</LANG>
      <LANG ID="zh-TW">MediaLibrary Example</LANG>
   </PHRASE>
   <PHRASE>
      <LANG ID="REF">Description</LANG>
      <LANG ID="en-GB">Demonstrates the features of the MediaLibrary Lua library</LANG>
      <LANG ID="es-ES">Demostración de las funciones de la biblioteca de medios de Lua</LANG>
      <LANG ID="ja-JP">Demonstrates the features of the MediaLibrary Lua library</LANG>
      <LANG ID="en-SG">Demonstrates the features of the MediaLibrary Lua library</LANG>
      <LANG ID="zh-HK">Demonstrates the features of the MediaLibrary Lua library</LANG>
      <LANG ID="zh-TW">Demonstrates the features of the MediaLibrary Lua library</LANG>
   </PHRASE>
</GROUP>
```

The following image shows the Properties panel with the localization applied for the name of the object:

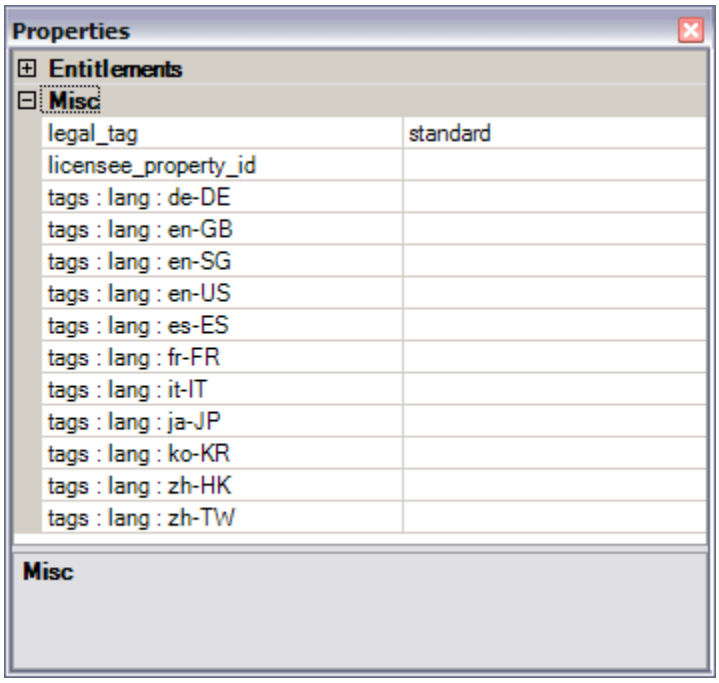


Object Metadata

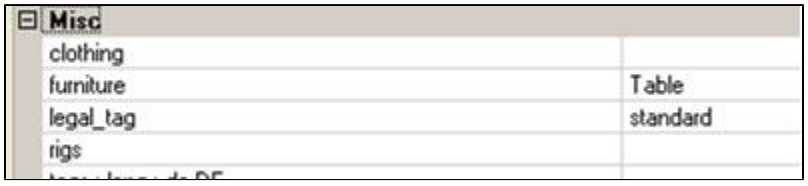
What is Object Metadata?

Metadata is descriptive data about the object. Developers use it for searching and indexing in the tools and in the run-time client. You can quickly search for an item in the .self using the `searchObjectCatalogue tags <tags_to_search>` command. Users cannot search for items by their metadata.

Metadata consist of key-value pairs, where the value can be a space-separated list of keywords. All objects have a localized set of metadata under the 'tags' keyword. These keywords can be anything helpful to you when searching for your object:



When you add components to your objects, certain components add extra keys to the metadata. For example, the Clothing component adds a 'clothing' key to the metadata list. When you export furniture, a 'furniture' key is added to the metadata list, with the type of furniture listed next to it. The following image shows the metadata automatically generated for a table:



Usually, values for these keys are determined by the tools and do not need changing. Some keys require no value at all. An example case is the clothing key, possible values for this include 'head', 'torso' or 'feet'. These are set by the export process when exporting a character component. The values for the clothing metadata are used by the client to sort clothing items into the

correct places when the player accesses their wardrobe.

You can edit the metadata of multiple objects at once. For more information, see Multiple Objects Edit.

> Reward entitlement_ids are located in Metadata. For more information about using an item as a reward and on entitlement_ids, see Rewards and Commercial Items.

Performing a Metadata Search in the Dev Debug Console

You can search the object catalogue from the PS Home console using the following command:

```
searchObjectCatalogue tags <tags_to_search>
```

When you enter a search term, those items with that term in their metadata are displayed:



> Three other GUIs appear, but they have no effect. Pressing [L2], [R2], the directional pad, the ACCEPT or the DECLINE button does not allow you to filter, sort, navigate, select, or return the objects in the metadata search.

This search is useful only for viewing which items have that metadata term in the .self. You cannot select an item after searching for its metadata tag.

Retrieving Metadata in Script

You can retrieve an object's name and description in your Lua script using the AsyncCommands library. The ObjectRetrieveMetadata functions returns the localized name and description for the region that you (or the user) are in. See Retrieving Object Metadata.

Compiling Scripts in the Object Editor or Scene Editor

You can compile scripts while developing your content. You can also compile scripts during packaging. See Compiling Scripts When Packaging

To compile your Lua scripts while developing content in the Object Editor or Scene Editor:

1. Select Scripts > Compile All Scripts.

   All the Lua scripts associated with your content are compiled using the default options. Each Lua script with the extension .lua now also has the compiled script with extension .luac.

2. Object Editor only: In the Object Editor, select multiple objects in the Object Search panel to compile scripts across all those objects.

3. Test your content within the client as follows:

   a. Select Scripts > Switch Scripts.
   b. Seelct Compiled.
   c. Click Switch.

All the resources in your content are pointed to the compiled scripts (.luac) and the properties of each resource are updated accordingly. The content now accesses the compiled scripts when run within the client.

The properties are changed to point at the compiled script, but the name of the resource does not change.

Managing the User Inventory

To view objects in PS Home, you must add the objects to your user inventory. The inventory options are available only when testing material, and can be accessed only through the HDK.

Accessing the User Inventory

The User Inventory lists all the Home objects that are owned by the user. You can access two inventory management options from the DEV DEBUG menu:

- Clear Inventory: Remove all furniture and clothing objects that have been added to your inventory. This is useful when you need to repeatedly test the rewarding of an item.
- Object Catalogue Browser: Access a hierarchical browser of all Home objects available to the client. You can browse all objects, only those objects in the active object catalogue, or just the objects in the user inventory.

To activate the menu:

1. Press SELECT on the connected controller.
2. Select DEV DEBUG from the PS Home Safe Screen.

> You can access the DEV DEBUG menu only while in 'release mode'. For more information about the DEV DEBUG menu, see The Home Client.

Dev Debug Console Commands

Use the following console commands to manage the inventory:

Command	Description
inv list user	Lists all object IDs of objects in the user's inventor.
inv adddevobjects	Temporarily includes all objects in build directory within object catalogue so that when the .self is launched it is easy to browse for any object on your HDD.
inv adduserobj <OBJECTID>	Adds the specified object to the user's inventor. You cannot use adduserobj in the hubstartup text in Target Manager.

`inv clear <default,paid, user, all>`	Removes objects from the user's inventory, depending on specified type (either default objects, paid, user, or all).
`inv list user`	Lists all object IDs of objects in the user's inventory.

For a full list of console commands, see Debug Console Command Reference.

> Restart the PS Home Client after you change any object properties, to make sure that the object behaves as expected.

Test Inventory

When you export items such as clothing, they are added to the test inventory. The inventory is an XML file: `<HDK_INSTALL_DIR>\build\TestInventory.xml`. When you run HomeDeveloper.self, it reads the test inventory and adds the items it finds to the user's inventory.

You might not always want this behavior. For example, if you want an item of clothing to be a reward, you do not want it added to the user inventory.

To prevent an object from being added to the user inventory, do one of the following:

- Remove, it using a console command
- Edit TestInventory.xml and remove the corresponding object ID

Defining Animation Registers

To manipulate the behavior of animation blend trees, you can define your own animation register in the Object Editor to use within your animation repertoires (see Repertoire Component). You use animation registers as control variables to give you more control over how animation states are selected. You can also use animation registers to output a value through a register to be read by a Lua script.

The Repertoire Editor reads which animation registers are available for use in conditions and events. See Repertoire Animation Blending.

Each avatar has a set of animation registers.

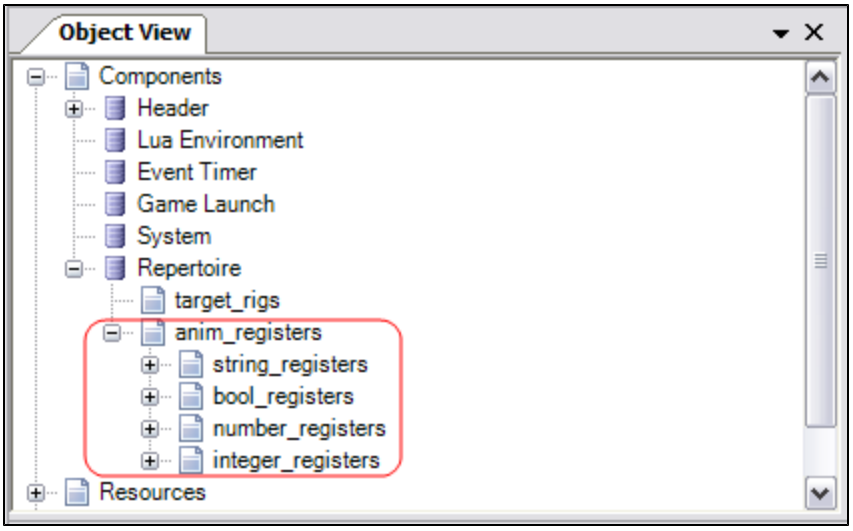
> You cannot define animation registers for FullBodySuits because they do not have a Lua environment, and a Lua environment is required to access animation registers.

Adding a New Animation Register to a Repertoire Component

You can add an animation register only to an object that contains a repertoire, and is a mini-game.

To add an animation register:

1. Open your object in the Object Editor, and go to the Object View panel.
2. Expand Components > Repertoire > anim_registers.



3. Select the required animation native type, then select Object > Add a new register.

   The new register is displayed.

4. Set the properties as required. See Animation Register properties.

Defining Conditions for Animation Networks

You can define conditions for networks, or for the states within networks.

You can also use conditions to determine which networks and states are used. See Repertoire Conditions and Logic.

See also:

- Adding a Repertoire
- Repertoire Component
- Repertoire Animation Blending Workflow
- Repertoire Animation Blending
- Repertoire Editor Node Reference

Rewards and Commercial Items

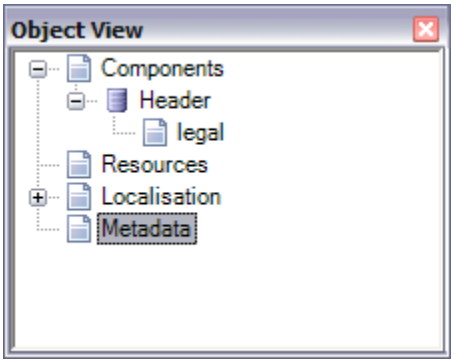
Issuing Your Object as a Reward

Issuing a reward is optional. A reward can be issued to users for free for completing a certain task, such as completing an external game title successfully or visiting a specific space in PS Home.

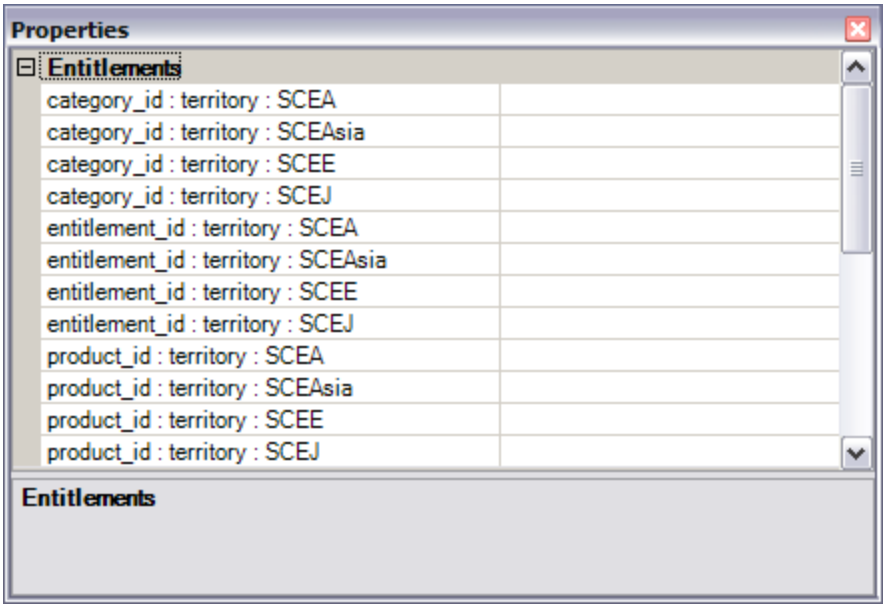
For more information on issuing your object as a reward, see PS Home Rewards

To issue your object as a reward, you need to set the entitlement_id property:

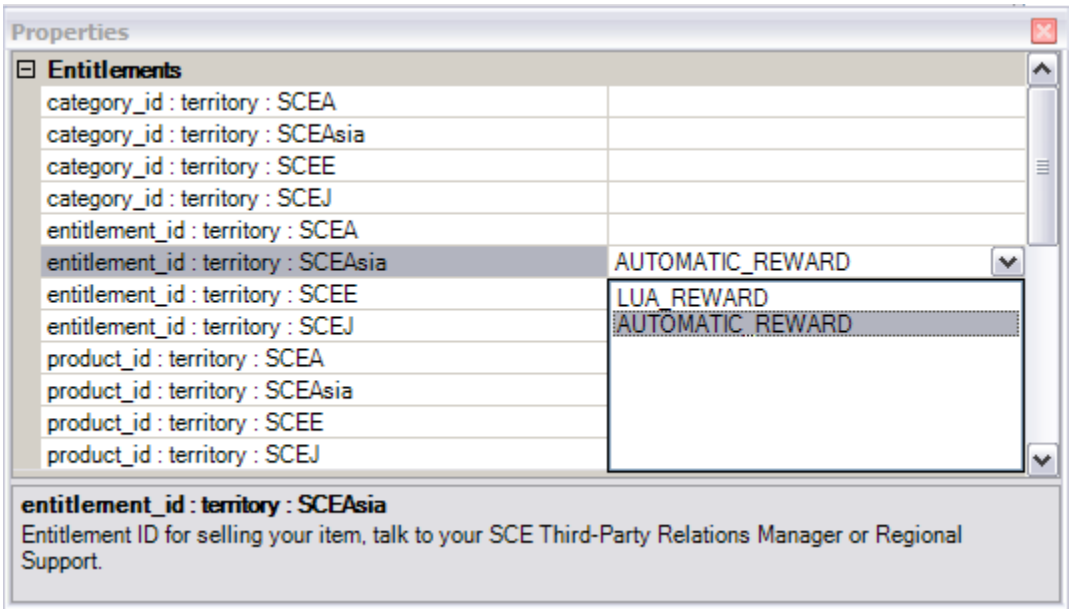
1. In the Object View panel, select Metadata:



2. In the Properties panel, select Entitlements:



3. Select the entitlement_id for the region(s) in which you want to issue the reward.
4. Select either LUA_REWARD or AUTOMATIC_REWARD from the entitlement_id drop-down list:

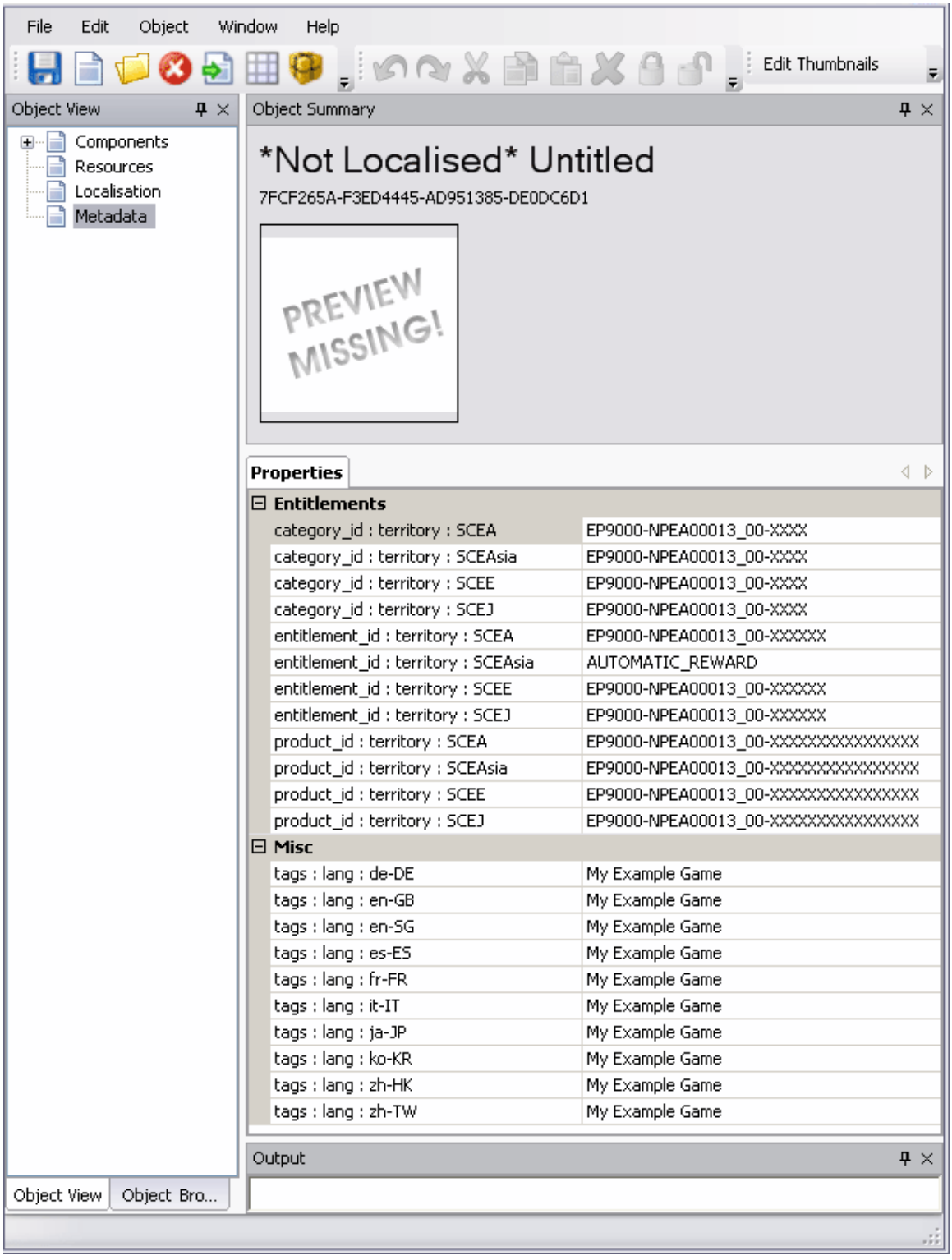


The remaining tasks for issuing an award occur outside of the Object Editor. For more information, see Awarding Reward Tickets.

Making your Object a Commercial Product

To make your object a commercial product which can be bought by users, you need to do the following:

- Ensure the object display name and thumbnails are correct, as these need to match the product name and image that will be used to display the information about your product in PS Home.
- Get the category_id, entitlement_id and product_id from your SCE Third-Party Relations Manager or Regional Support and copy these into the metadata.



Make sure that there is no white space in your PMT data (such as the entitlement IDs) because it prevents your object working properly in commerce.

See also Commerce in PS Home.

Preparing Objects for Packaging

For the Object Editor to package successfully, your object must have at least these items:

- Display name, description and object ID
- Thumbnails
- Age rating information

> ⚠ Objects such as mini-games and arcade games that are referenced by the scene must be packaged using the Object Editor. The Scene Editor does not package objects. When submitting the scene package to Quality Assurance (QA), submit all the related object packages at the same time.

Make sure that you profile your objects before packaging them. See Profiling Objects.

For information on the technical requirements, see Testing, Validating and Submitting Content.

Object Identifiers

An object has the following identifiers:

Name	Description
Name (Display Name)	A descriptive and characteristic name for the object which, along with a paragraph about the object and a thumbnail, is visible in PS Home to users.
Object ID	A unique name, not visible to users in PS Home, assigned to the object when it is first created.

After you have finalized your object, check that you have described the object, assigned a display name to the object and assigned the correct thumbnails (see below) before finally packaging the object into an archive.

> ⚠ You must provide a description and name for your object and the thumbnails must be accurately sized. See HDK Tools Validations for requirements.

Review the description and name of the object in the header (under Components in the Object View panel). Make sure this is up-to-date and describes your object accurately, as this information will be displayed to the users in PS Home. For more information on how to do this, see Completing the Object Header.

Thumbnails

Adding thumbnails is a requirement for objects to be submitted to the PS Home's quality assurance process. You can add a thumbnail representing both the object that you created, as well as the proprietor/copyright holder of the object.

If required, you can add thumbnails to multiple objects at once. See Editing Multiple Objects.

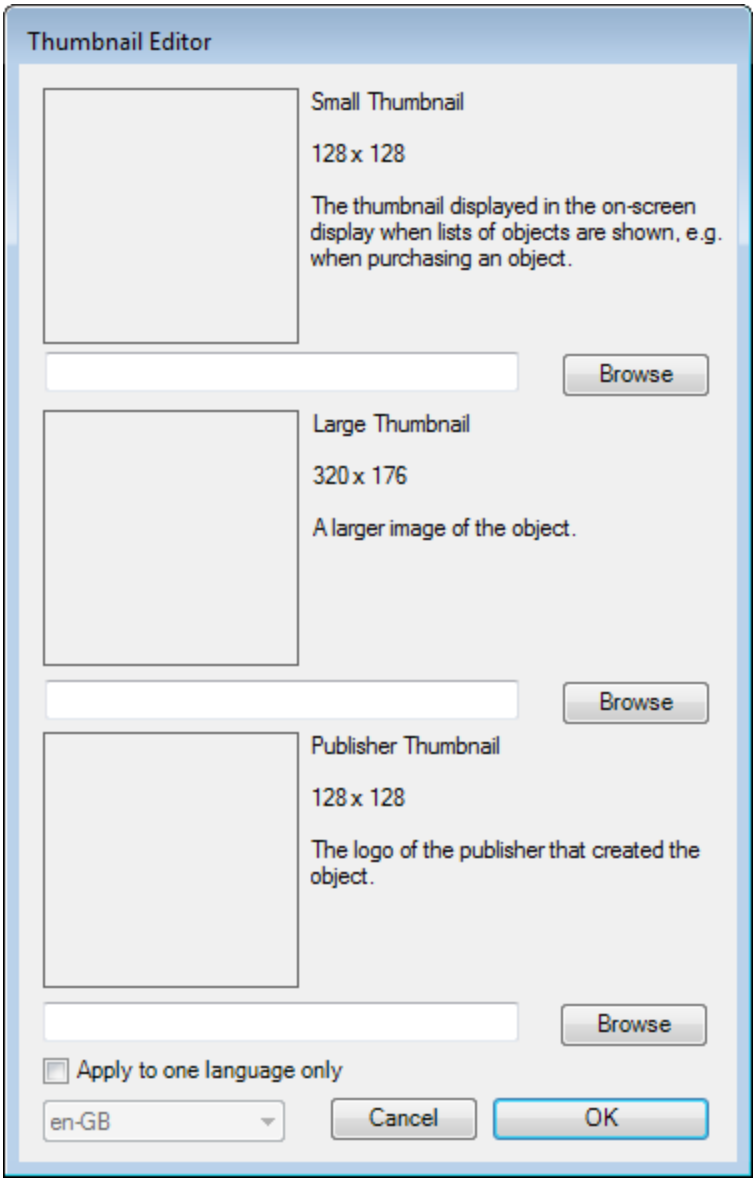
For more information on creating object thumbnails, see Thumbnails.

Adding Thumbnails

Make sure that you have the correct thumbnails of the object.

To edit the thumbnail:

1. Click Edit Thumbnails on the toolbar or select Object > Edit Thumbnails to display the Thumbnail Editor dialog:



2. Browse and select the required thumbnail images.

   The information is stored in an .odc file, and the thumbnail images are packaged during the packaging process. The .odc file must contain:

   - A description of the object
   - A small thumbnail (128 x 128)
   - A large thumbnail (320 x 176)

   > The Publisher Thumbnail is optional.

3. If you want to enable regional variation and ensure the object appears differently in different regions, check the Apply to one language only box and select the language from the drop-down list displayed (you will need to repeat this step for each region you want to localize to).
4. Click OK to save your changes.

For more information on creating your thumbnails, see Thumbnails.

Age Rating Information

To make sure that your content works as expected, set the following age restrictions:

- Age rating (if required)

- Parental control level (mandatory)
- Minimum age (mandatory)

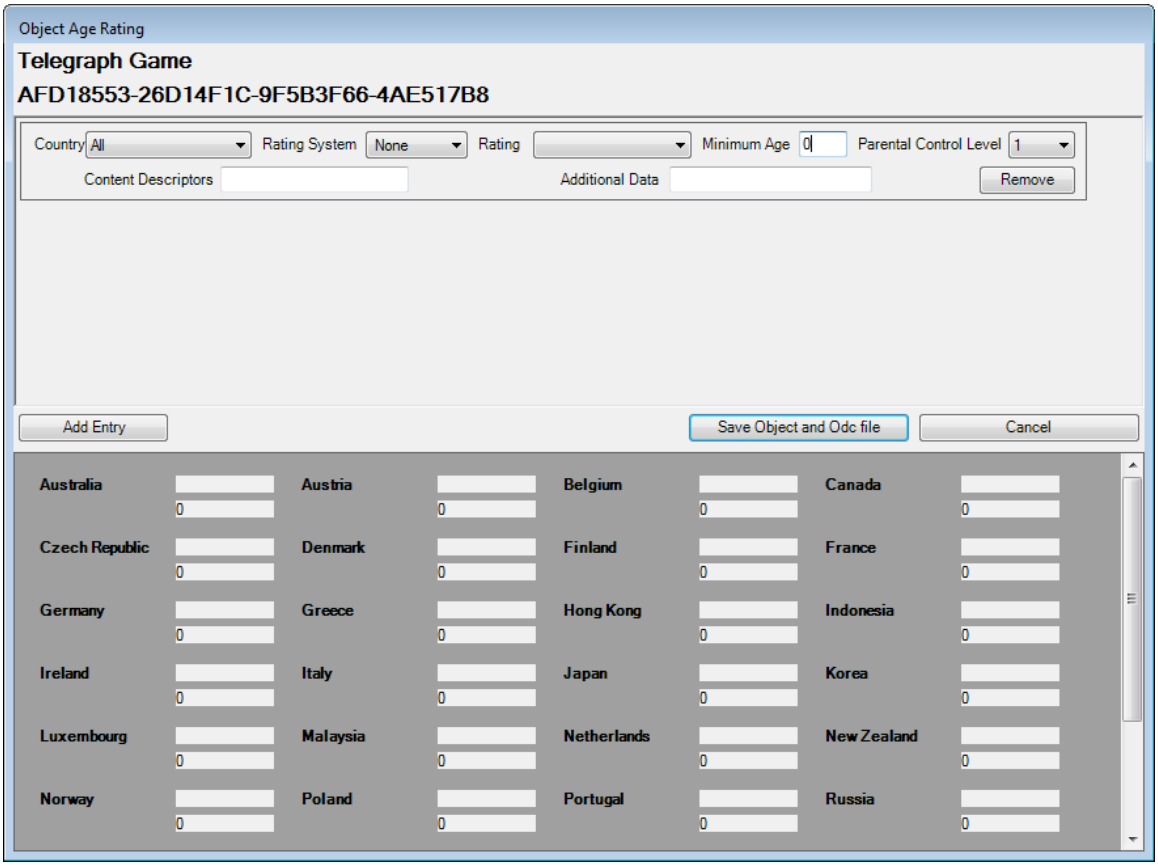
The Home Developer Build must be connected to the PlayStation®Network (PSN) to validate the age of the user, even in offline mode. If there is no connection to the PSN, the default age of 0 is used. To test content without a connection to PSN therefore, you must set the minimum age of entry to 0. You can set the minimum age to 0 only for testing. You cannot use this setting for published content.

> ⚠ Restricting content is a requirement. For information on which settings to use and where to obtain official guidance, see Age Restrictions and contact your Regional Account Manager.

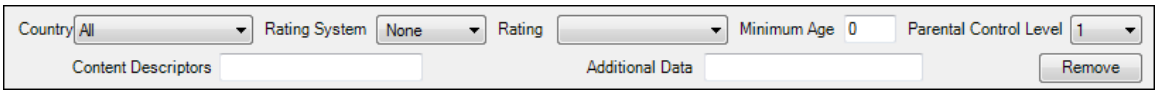
Setting Age Restrictions

To edit the age restriction information:

1. Click Edit Age Ratings in the toolbar or select Object > Edit Age Ratings to display the Age Rating Form dialog:

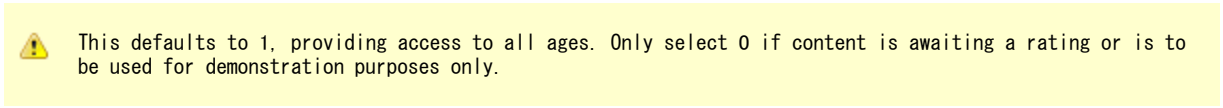


2. Click Add Entry to add a new entry:



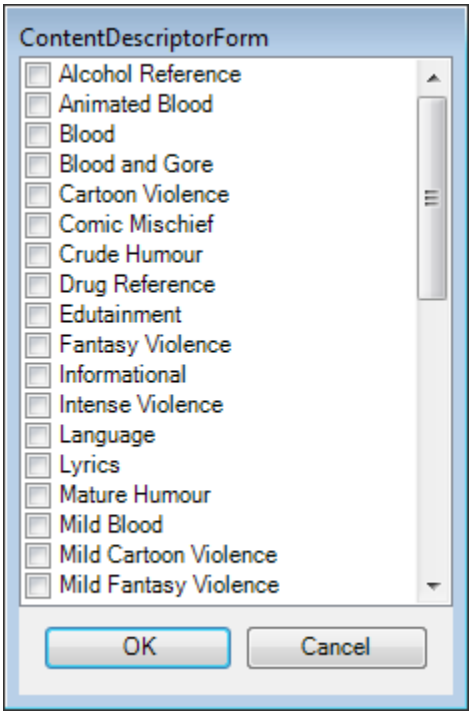
> ℹ The Additional Data field is not currently used. Leave this field blank.

3. Select the country in which the restriction applies from the Country drop-down list.
4. Select the appropriate rating system from the country selected from the Rating System drop-down list.
5. Select the rating level required from the Rating drop-down list.
6. Enter the minimum age for which the content is appropriate in the Minimum Age field. This setting must match the Rating level set.
7. Select the parental control to the appropriate level for the age from the Parental Control Level drop-down list.

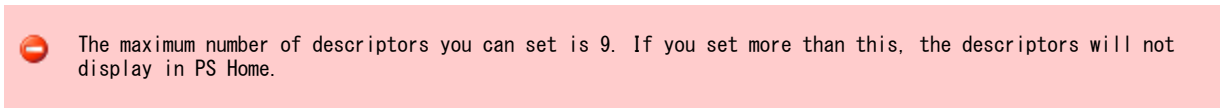


This defaults to 1, providing access to all ages. Only select 0 if content is awaiting a rating or is to be used for demonstration purposes only.

8. For regions where age rating systems require descriptions of the contents, click the Content Descriptions field to display the Content Descriptor Form dialog:

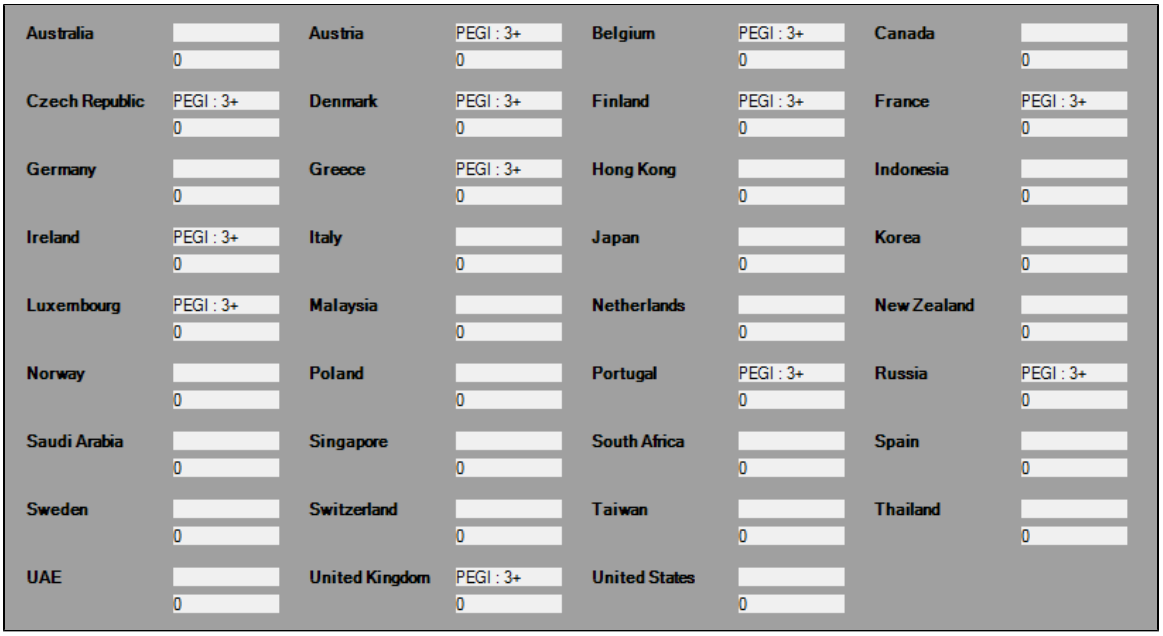


Check the boxes next to the descriptors that apply to your scene, then click OK. Each rating system has its own set of terms for describing the content of certain ratings. For example, the ESRB descriptors include such descriptors as: 'Alcohol Reference', 'Blood and Gore', 'Mature Humour'.



The maximum number of descriptors you can set is 9. If you set more than this, the descriptors will not display in PS Home.

9. For each country affected, click Add Entry and repeat steps 3 to 7. A summary of all of the restrictions that you're adding are automatically displayed in the bottom half of the Edit Scene Age Rating dialog, for example:



10. When you've add all of of the required age restrictions, click Save Object and Odc file to save the object and the .odc file.

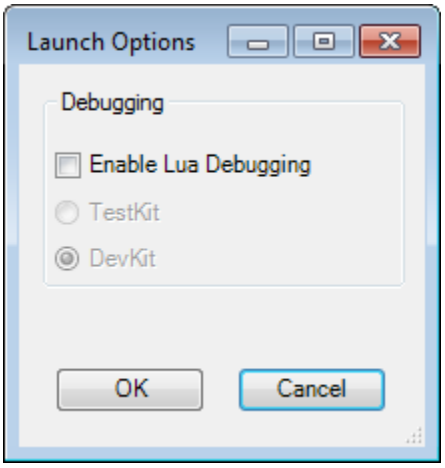
Profiling Objects

There are two options available for profiling:

- Select Object > Validate on PS3 to open the Static Object Profiler.
- Select Object >Preview on PS3 to open the Interactive Object Profiler.

For more information on profiling objects, see Profiling Objects.

You can also select Object > Launch Options to display the Launch Options dialog:



This is useful if you intend to run SLED to debug Lua code, and allows you to specify if the target platform is a development or test kit.

For more information on using SLED, see Debugging Lua Script using SLED.

Packaging Objects

You must package your objects into an optimized archive format before submitting it for Quality Assurance (QA).

Packaging Requirements

Make sure that:

- You have completed the preparation for packaging and that the object is finalized before finally packaging it. See Preparing Objects for Packaging.
- You do not modify the object content, file name or localization text after you have packaged the object. If you do, the content will fail QA.
- The object does not include any unused files. Packaged objects cannot contain unused files.
- Object packages are within the maximum size limit. Packages submitted for publication through the Content Delivery System (CDS) must be less than 100 MB. Keep the file size of packaged scenes as small as possible.

Packaging Objects

> From HDK 1.2 onwards, the Scene Editor and the Object Editor automatically optimize file compression during packaging. If you have objects that were created with versions older than HDK 1.2, open them in the Object Editor and repackage them using the packaging process described here.

From the Object Search Panel

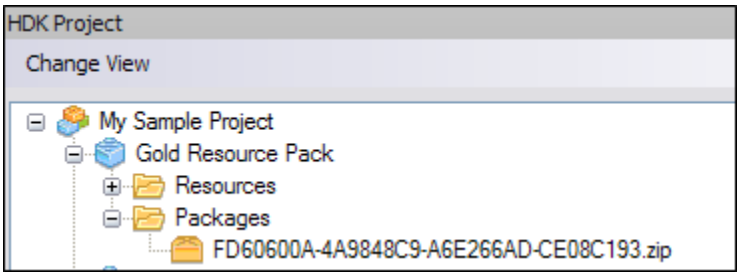
To package an object:

1. Select the object you want to package in the Object Search panel.
2. Click the Package Object button  on the toolbar, or select File > Package Object.
3. Check the Output panel to see whether or not packaging was successful (see below).

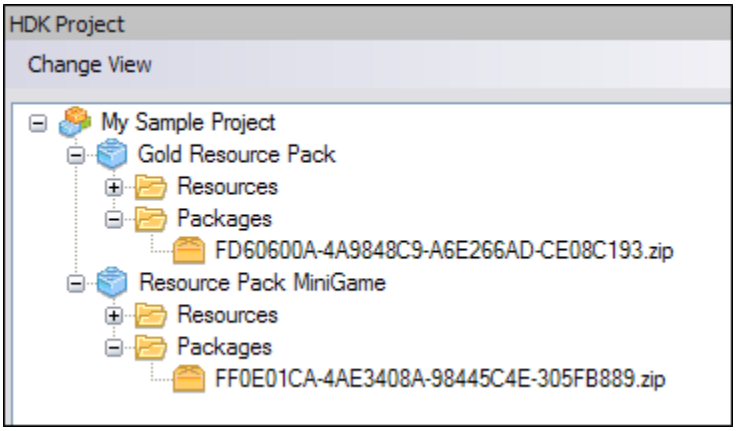
From the HDK Project Panel

1. Select the object you want to package in the HDK Project panel. To package all the objects in a project, select the project name.
2. Do one of the following:
   - Click the Package Object button  on the toolbar.
   - Select File > Package Object.
   - Right-click on the selected object/project and select Package item from the pop-up menu displayed.
3. Check the Output panel to see whether or not packaging was successful (see below).

The process creates a Packages folder and places the archived object files in it:



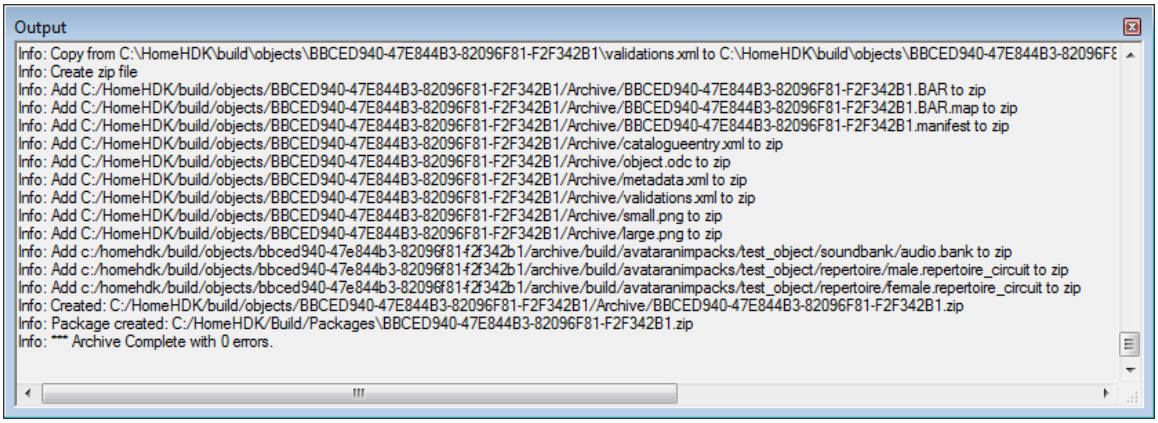
If you selected to package all of the objects in a project, the process creates individual Packages folders for each object:



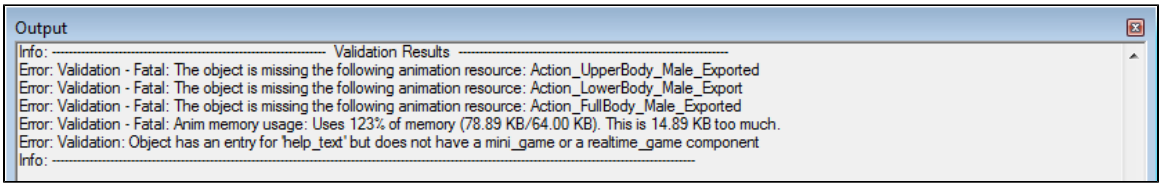
Checking the Packaging Output

Automatic validation is carried out during packaging. For information on what is validated, see HDK Tools Validations.

If the object is packaged successfully, the Output panel shows that there are no errors or warnings:



If there are validation errors (such as incorrect paths or incomplete properties), the Output panel lists them. For example, if there is a missing animation resource, you get the following output:



Viewing the Packaged Object

Objects are archived to a path matching the following pattern: `<HDK_INSTALL_DIR>\build\Packages\<OBJECT_ID>.zip`

To view a packaged object within a project, right-click the .zip file in the HDK Project panel, and select Go to File from the pop-up menu displayed. Windows Explorer displays the archived object.

Game Components

You can create different types of game components:

- Arcade Games: Games that are played on a 2D graphics screen.
- Mini-games: Objects with a MiniGame component. Mini-games in PS Home can be multi-player games such as Chess, or single (local) player interactions.
- Realtime Games: Objects with a RealTime Game component. Similar to mini-games, except they use peer-to-peer communication and do not allow spectators.

Changing User Gender or Clothing

You can change gender or clothing only in mini-games and realtime games. If you want like to change gender or clothing outside a mini-game or realtime game, you must first contact your regional SCE.

For more information on changing gender, see Changing Gender.

For more information on changing a user's clothing items, see the `LocalPlayer.SetRigComponentObjectId` function.

Arcade Games

Arcade games are games that are played on a 2D graphics screen. Arcade games support only 1-player between PS Home users, but can have up to 4 co-op players (that is, up to 4 players playing the arcade game from 1 PlayStation®3 machine). When adding an arcade game as a Game Object in Scene Editor, the object properties include Player Slots. This value must be set to 1 (the default).

> ⚠ Arcade games do not have access to the `MiniGame` library, cannot use listener data, cannot communicate over a network, and do not support commerce points. See Object Component Properties.

Ordinarily, an end-user starts an arcade game by moving an avatar within a trigger volume associated with a virtual arcade cabinet placed within a lobby. However, the HomeDeveloper.self installed with the HDK can start up an arcade game directly, allowing the developer to bypass Home's menus and environment loading. This is useful when developing an arcade game.

To start an arcade game directly, pass the following command line argument to the Home executable HomeDeveloper.self in the Target Manager:

```
\--game <OBJECT_ID>
```

The HDK includes the following example arcade games:

Name	Object ID
ArcadeTutorial1	00000000-00000000-00000000-000000a3
ArcadeTutorial2	00000000-00000000-00000000-000000a4
ArcadeTutorial3	00000000-00000000-00000000-000000a5

To load ArcadeTutorial1, specify the following command in the command line:

```
\--game 00000000-00000000-00000000-000000a3
```

When the game is loaded, you can reset and reload it from the Debug console, using the following command:

```
lc Debug.ArcadeGameReset()
```

You can thus edit the script and rerun it without having to relaunch HomeDeveloper.self, which allows for rapid iteration times. The HDK maps F2 to execute this command by default.

You can also load a different arcade game from the console. For example, to run ArcadeTutorial3, you can execute:

```
lc Debug.ArcadeGameLoad("00000000-00000000-00000000-000000a5")
```

Timeout

The PS Home client automatically boots users out of arcade games in public spaces if they are idle for three minutes, so that other users in the space can have a chance to play. The timeout applies if the arcade game is embedded and the user is not in their own apartment and not in a clubhouse.

Timeout is disabled in the following circumstances:

- All arcade games in clubhouses (whether placed as furniture or embedded in the scene).
- All arcade games placed as furniture in personal spaces.
- The owner of a personal space for embedded arcade games. Guests in a personal space are still subject to the timeout for embedded arcade games.

When testing arcade games, you can use the boolean Debug Console Command `EnableArcadeGameTimeout` to toggle the timeout to `True` (timeout enabled) or `False` (timeout disabled).

Enhanced Arcade Games with HDK 1.70+

Three new functions have been added to the ArcadeGame library in HDK 1.70 that allow you to implement smoother, more reactive arcade games running at up to 60 frames per second (fps). Due to the nature of the PS3 platform and the resulting architecture of the 3D renderer, the Home client introduces some delay into the rendering process to render 3D scenes as optimally as possible. However, this delay can be noticeable in a fast-paced arcade game and hinder the user's enjoyment, when it need not because the 3D scene doesn't need to be rendered while the arcade game is active. To address this, HDK 1.70 adds the following functions to the Lua API:

```
ArcadeGame.EnableSceneRender(boolean enable)
ArcadeGame.EnableMlaa(boolean enable)
ArcadeGame.EnableVsync(boolean enable)
```

All default to true, and you must pass false to turn them off. You should also ensure that all settings are set back to true when your arcade game exits. A brief description follows; for exact details, consult the Lua API documentation.

EnableSceneRender

If set to false, the 3D scene will no longer be rendered in the background. This includes OSD and Renderer instances owned by objects in the scene. You will not see any visual difference as the arcade game uses the whole screen, however this will return some frame time to the arcade game.

EnableMlaa

The Home client makes use of an anti-aliasing technique called MLAA, which necessitates an extra frame of render delay to perform the antialiasing. This is only performed on render of the 3D scene, not on screens or 2D objects, so it is unnecessary while an arcade game is active. Setting this to false will disable the MLAA and remove one frame of render delay.

EnableVsync

The Home client by default syncs the render at 30 fps. This is sufficient for the 3D environment, but can be insufficient for very fast-paced or "twitch" games. If you disable the vsync, the client will run at up to 60fps, which will look and feel more smooth. Note that you must time your game using GetDeltaTime and not assume a frame rate of 30 fps in order for this function to be of use to your arcade game.

Rendering a Sprite

This section shows you how to render a sprite using the Arcade Cabinet Creation Tutorial.

This tutorial shows you how to:

- Write a simple arcade game script in Lua.
- Explore objects in the Object Editor.
- Run your script on your target hardware.

The resources for ArcadeTutorial1 are here: <HDK_INSTALL_DIR>\build\ArcadeGames\ArcadeTutorial1

The object for ArcadeTutorial1 is here: <HDK_INSTALL_DIR>\build\objects\00000000-00000000-00000000-000000a3

Now we are ready to start looking at the code that defines the game. Open main.lua in the editor of your choice and you will see the code snippet shown below:

```
-- LUA Libraries
LoadLibrary( "ArcadeGame" )
LoadLibrary( "Renderer" )
LoadLibrary( "Screen" )
LoadLibrary( "Sprite" )
LoadLibrary( "Texture" )
LoadLibrary( "Vector4" )
```

The LoadLibrary function will give your script access to the PS Home Lua APIs. So if your script needs to use the Vector4 type, you must first include the corresponding library. Every library is provided by a particular object component which must first be added in the Object Editor. For a full list of components, and the libraries they provide, see Additional Object Editor Functions in Object Component Properties.

```
-- Create the renderer
screen = ArcadeGame.GetScreen()
renderer = Renderer.Create( Vector4.Create( 0, 0 ), screen:GetSize() )
renderer:SetTarget( screen )
```

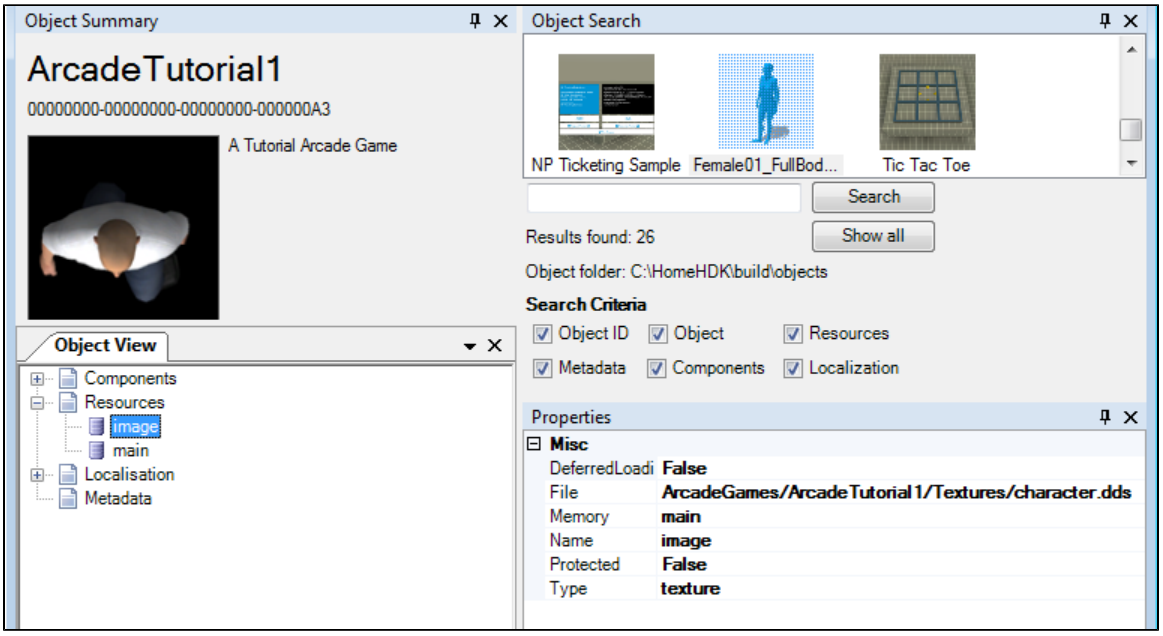
Before you can render anything to the screen, you must first set up a valid renderer. With a renderer, you can submit graphical primitives such as sprites, lines, filled rectangles and text. The renderer itself maintains a draw buffer with a virtual pixel space of 1024x720, which is independent of the display device resolution. Therefore, it is recommended that your screen be created with an equivalent aspect ratio, or the game's graphics will become stretched vertically or horizontally. In the code above, we are setting the renderer to draw to an area the same size as the screen on which the arcade game runs. The screen itself will be a quad placed somewhere within a 3D Home environment using the Scene Editor, which will be used in a subsequent tutorial.

```
-- Find the texture 'image' in the arcade game object's resources.xml
image = Texture.Find( "image" )

-- Create a sprite and set its texture to the image
sprite = Sprite.Create()
sprite:SetTexture( image )
```

Any assets that your script uses are called resources and are made available via the Object Editor. The texture above, named

'image', is set up in Object Editor as shown below:



Use a .dds format picture for the image Resource.

For more information on properties of components and resources, see Adding and Deleting Resources.

The Object Editor also allows you to set an `Update` function that is called every frame by the Home runtime (see the `on_update` property of the Arcade Game component). For ArcadeTutorial1, the `Update` function has been set to `Update` and the code is as follows:

```
function Update()

     -- Render our sprite
     renderer:DrawSprite( sprite )

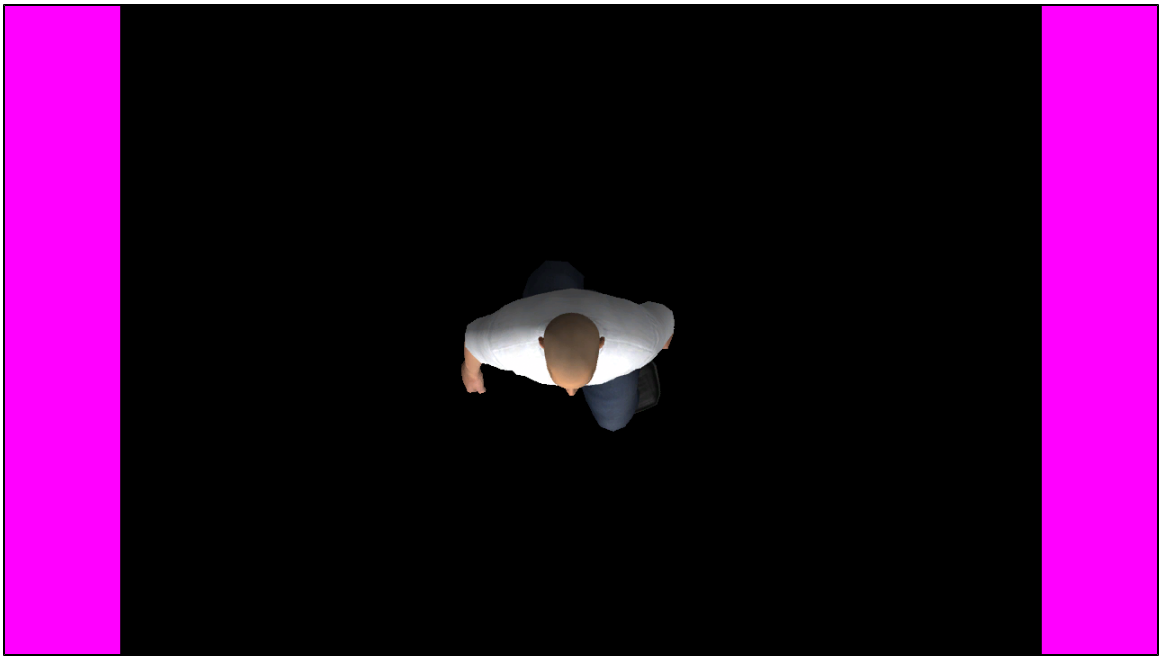
end
```

The update function simply submits the sprite to the renderer.

To run the tutorial, run `<HDK_INSTALL_DIR>\build\HomeDeveloper.self` from the Target Manager with the following command line parameters:

```
\--game 00000000-00000000-00000000-000000a3
```

You should see the following on-screen:



Animating a Sprite

This section shows you how to animate a sprite using the Arcade Cabinet Creation Tutorial.

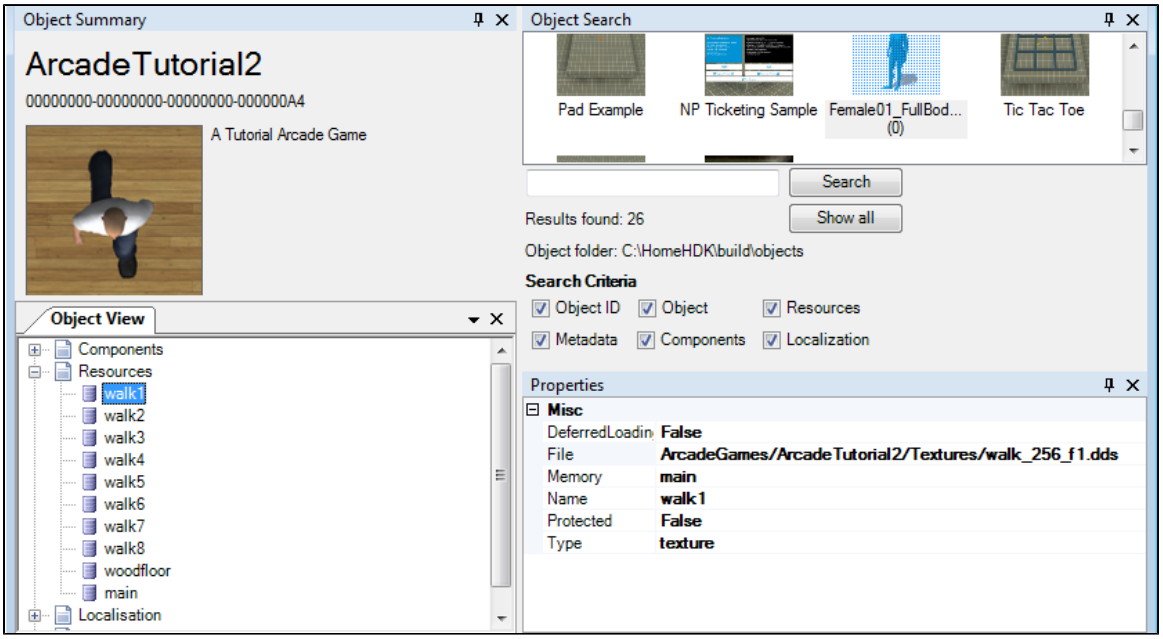
Resources for ArcadeTutorial2 are here: <HDK_INSTALL_DIR>\build\ArcadeGames\ArcadeTutorial2

The object for ArcadeTutorial2 is here: <HDK_INSTALL_DIR>\build\objects\00000000-00000000-00000000-000000a4

In the previous tutorial, we saw how you can assign a Texture object to a sprite for rendering. Now we will expand on that code to build something a little more game-like. So this tutorial will show you how to move an animated sprite around an environment.

There is another type available that can also be assigned to a sprite called 'SpriteAnim'. SpriteAnim objects reference textures as frames in an animation and determine which texture is displayed on the sprite at any given time. You are able to add many frames from one texture if required.

So, first of all, we need extra DDS images storing the frames of animation required to build our animation. ArcadeTutorial2 has a total of nine images in the Textures directory. All of these textures are referenced by the object's list of Resources. Confirm this by opening the ArcadeTutorial2 object in the Object Editor:



Now we can examine the code. If you are wondering why we do not step the animation explicitly, it is because that is being done

externally to our script every frame before our Update function is called. The code we need to build an animated sprite is as follows:

```
-- Create an animation and add its frames
anim = SpriteAnim.Create()

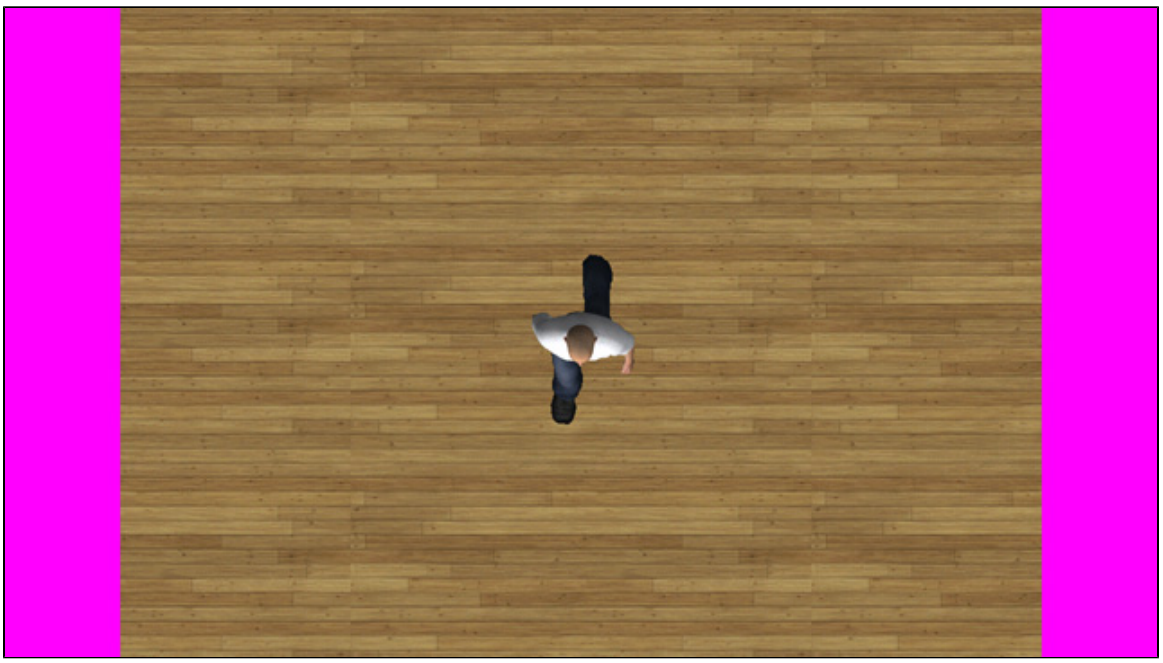
for frameIdx = 1,8 do
   -- Find the texture we have called walk in the package.xml
   image = Texture.Find( "walk" .. frameIdx )
   anim:AddFrames( image )
end

anim:SetLooping( true )

playerSprite = Sprite.Create()
playerSprite:PlayAnim( anim, 8 )
```

Notice that the SpriteAnim API allows you to loop an animation. Also notice that the speed of playback is controlled by the PlayAnim function, which specifies how many frames of the animation are played back every second. So, in our case, we have eight images in the animation and we are specifying eight images per second, so the animation sequence will loop once a second.

Run ArcadeTutorial2 and see how the sprite animates. The game should look like the image below:



You can control the heading of the character using the left analogue stick. The code in the script that reads the pad input is split into two parts. First of all, we need to map an 'action' to a pad button which is done as follows:

```
-- Get the pad and set it up
TURN = 0
Pad.CreatePads( 1 )

pad = Pad.GetPad( 1 )
pad:Reserve(PAD_JOY_LEFT_X, TURN, false, false)
```

Then, in the Update function of the script, we read the pad in response to that action occurring:

```
-- Update the player's heading based on the user's input
padValue = pad:GetAnalogExtent( TURN )
playerHeading = playerHeading - 2 * padValue
```

Every keyboard and mouse (for debug purposes only) and pad button (for debug and live environment) is represented by a predefined Lua constant and is available to your script. For the full list of these constants (and the Input handling API that accompanies

them), see the Lua API Reference.

GetAnalogExtent returns a value in the range -1.0, 1.0 for the pad's analogue sticks. Here we see this value being used to control the turn rate of the character sprite.

The last thing to point out in this tutorial is the camera. When moving a screen-centered sprite around a game world, you are faced with two choices. You could keep the sprite stationary and translate the sprites that form the environment around it. However, the PS Home Lua APIs provide some camera helper functions. This means that you can move the sprite around the world and track it with the camera. The code for this is shown below:

```
-- Update the player's position. Character faces down negative Y by default
playerPos:SetX(playerPos:X() + math.sin(math.rad(playerHeading)) * walkStep)
playerPos:SetY(playerPos:Y() - math.cos(math.rad(playerHeading)) * walkStep)

-- Track the player sprite with the camera
renderer:Camera2dSetPosition(playerPos)
```

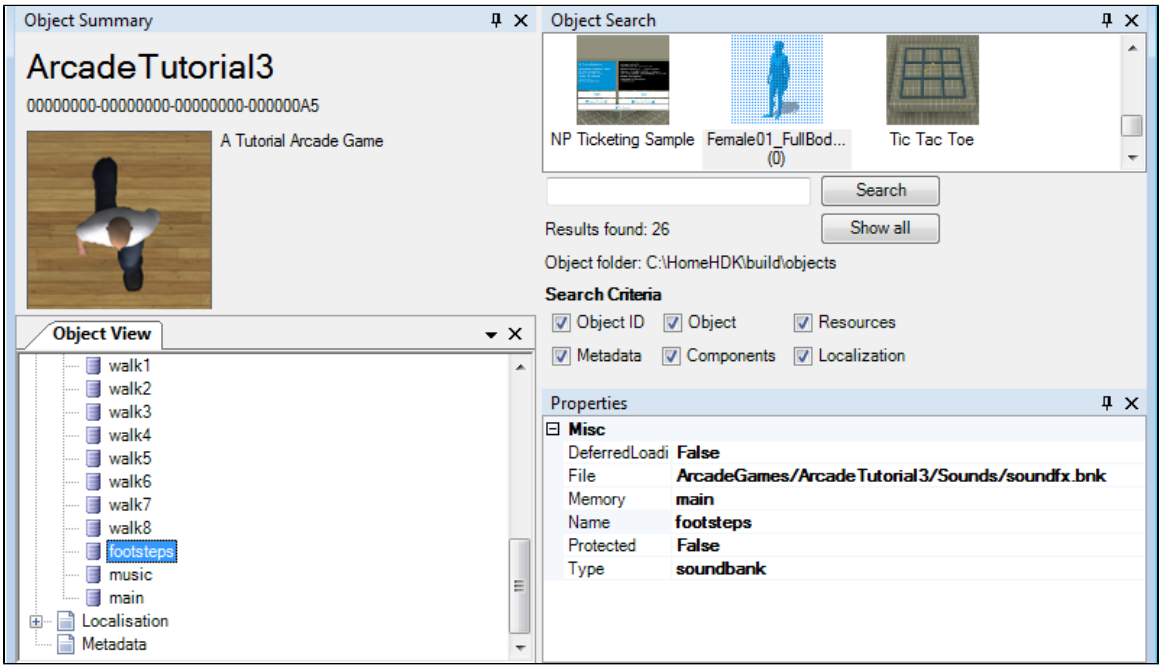
Adding Audio

This tutorial explains how you can add audio to your arcade game: both background music and sound effects. In Tutorial 2, we had a character walking across a floor. So now we will examine what must be done to play footstep sounds and a looping background audio track.

Resources for ArcadeTutorial3 are here: <HDK_INSTALL_DIR>\build\ArcadeGames\ArcadeTutorial3

The object for ArcadeTutorial3 is here: <HDK_INSTALL_DIR>\build\objects\00000000-00000000-00000000-000000a5

Open the ArcadeTutorial3 object in Object Editor and look at the list of resources. The two new entries, footsteps and music are listed in the Object View panel:



The footsteps entry references a Scream audio bank file resource (see Creating an Audio Asset). The music entry is an MP3 file resource where MP3 is the only supported format used for streamed audio using the Home Lua SoundStream API.

In your script, to play background music, you must first query the SoundStream dictionary for your music and then call the track's Play method:

```
-- Play background music

musicVolume = 0.4

musicPan= 0.0


music = SoundStream.Find( "music" )

music:Play2d( true, musicVolume, musicPan )
```

The music volume is set in the range 0, 1.0. The pan is in the range -1.0, 1.0. This controls the stereo output for the music.

To play sound effects, the process is similar. First of all, the SoundBank dictionary is queried for the string that was assigned to the SoundBank resource in the Object Editor. This returns a SoundBank which can be played as follows:

```
-- Cache footsteps sound effect

footstepsBank = SoundBank.Find( "footsteps" )


. . . .


-- Play any audio FX

time = time + GetDeltaTime()

if (time > 0.5) then

   footstepsBank:Play2d("FF_WALK_SLOW")

   time = 0.0

end
```

The SoundBank Play function returns a Sound object. This allows you to call functions on a currently playing sound, such as Stop, IsPlaying, IsPaused, Pause and Resume.

Deploying an Arcade Game in a Scene

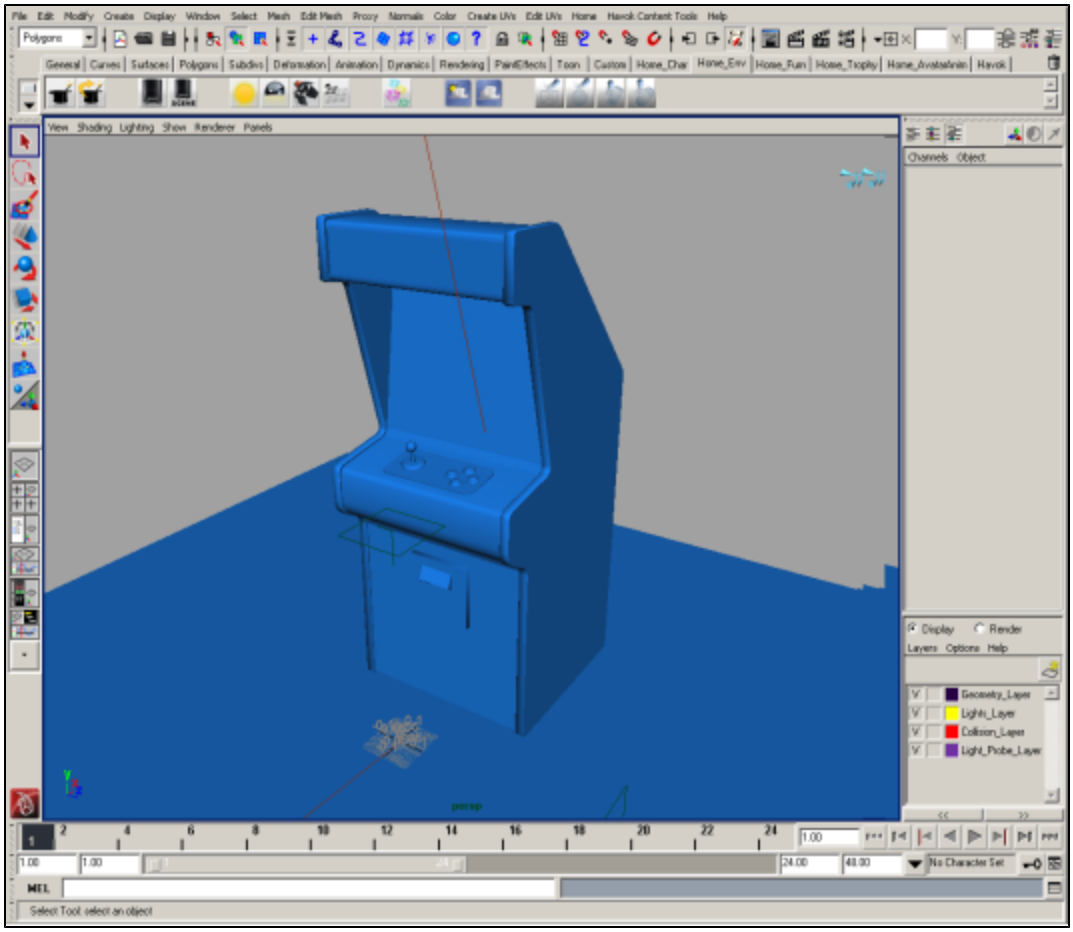
When we are satisfied that we have finished prototyping our arcade game, the next step is to install it into a PS Home space. This allows avatars to approach a virtual arcade cabinet and play the game. The steps are as follows:

1. Create a Home environment within Maya.
2. Export the environment to disk.
3. Load the environment into the Scene Editor.
4. Decorate the scene with the elements necessary to enable the arcade game.
5. Launch the finished environment into the Home runtime and test the game.

In this tutorial, we use Maya. Rather than creating the scene and geometry from scratch, a prebuilt Maya file has been created for you to inspect. Open the following Maya ASCII file which is saved in Maya 7.0.1 format:

<HDK_INSTALL_DIR>\artsource\Environments\ArcadeTutorial\ArcadeTutorial.ma

You should see the following scene:



This scene consists of:

- A ground plane and an arcade cabinet. The cabinet's screen happens to be placed at exactly the correct height and orientation to match the virtual arcade screen that we will be creating within the Scene Editor, which will essentially overlay the Maya geometry.
- Two cube collision meshes which encompass both the whole scene and the arcade cabinet.
- A light volume which determines the sample area for light level within the 3D space. This controls how bright avatars are rendered, depending on where they are standing.

Note that the scene has already been exported for you to the following location:

```
<HDK_INSTALL_DIR>\build\Environments\ArcadeTutorial
```

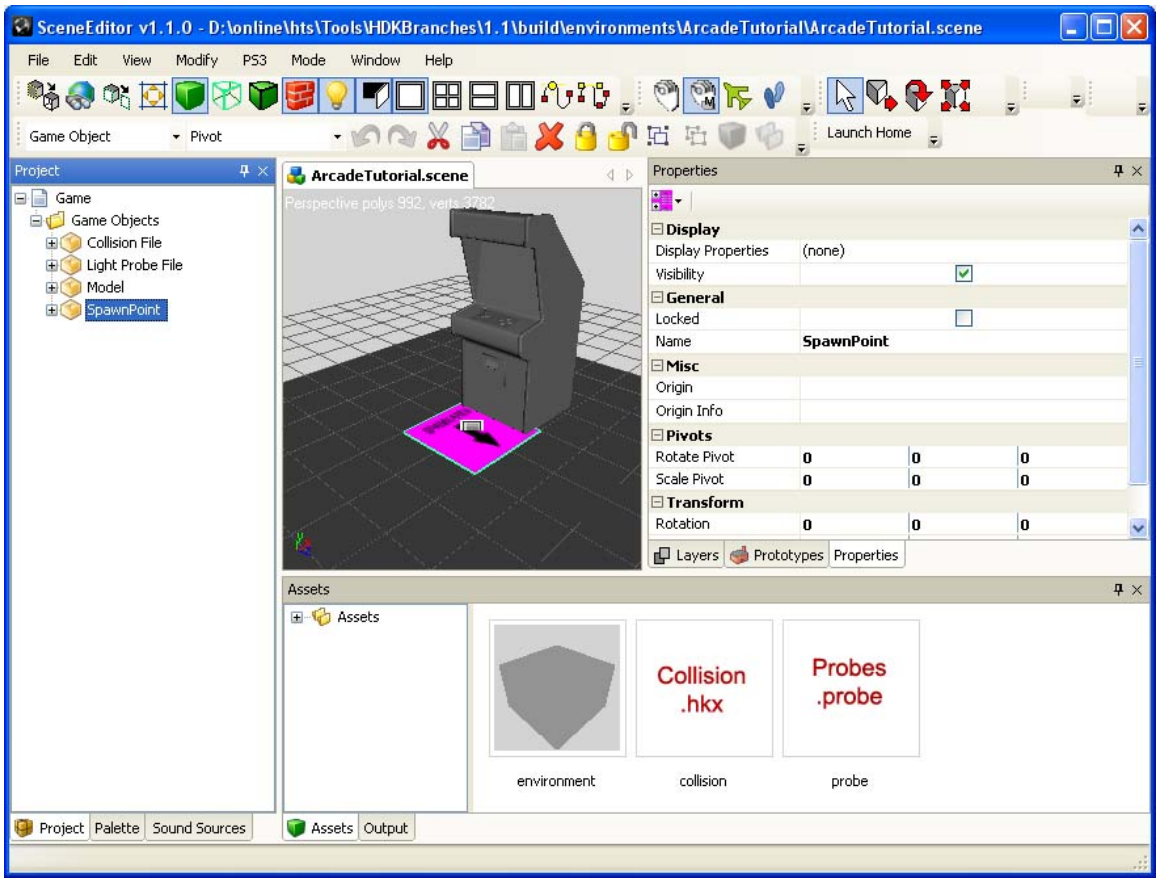
To export this scene yourself for the first time, you would use the Home Export Wizard accessible from the Maya shelf. For more information on exporting scenes from your 3D modeling application, see Maya. After exporting the scene, you must add the arcade game as 'markup' from within the Scene Editor.

To prepare the scene for the runtime:

1. Open the Scene Editor.
2. Open the exported scene:

   ```
   <HDK_INSTALL_DIR>\build\Environments\ArcadeTutorial\ArcadeTutorial.scene
   ```

   The following scene is displayed:



We need to configure two items in the scene to prepare it for the runtime: a spawn point and an arcade game. We begin with the spawn point which defines where every avatar will appear when they enter the level. The export process automatically creates a spawn point which defaults to being created at the origin pointing down the positive Z axis. This means that spawned avatars will look down positive Z.

3. To set the spawn point so that the avatar spawns facing the arcade machine, first select the Spawn Point object in the Project panel. You should be able to see the Spawn Point's properties in the Properties panel. Set the translation of the Spawn Point to -1.5, 0, 0 and the rotation to 0, 90, 0.

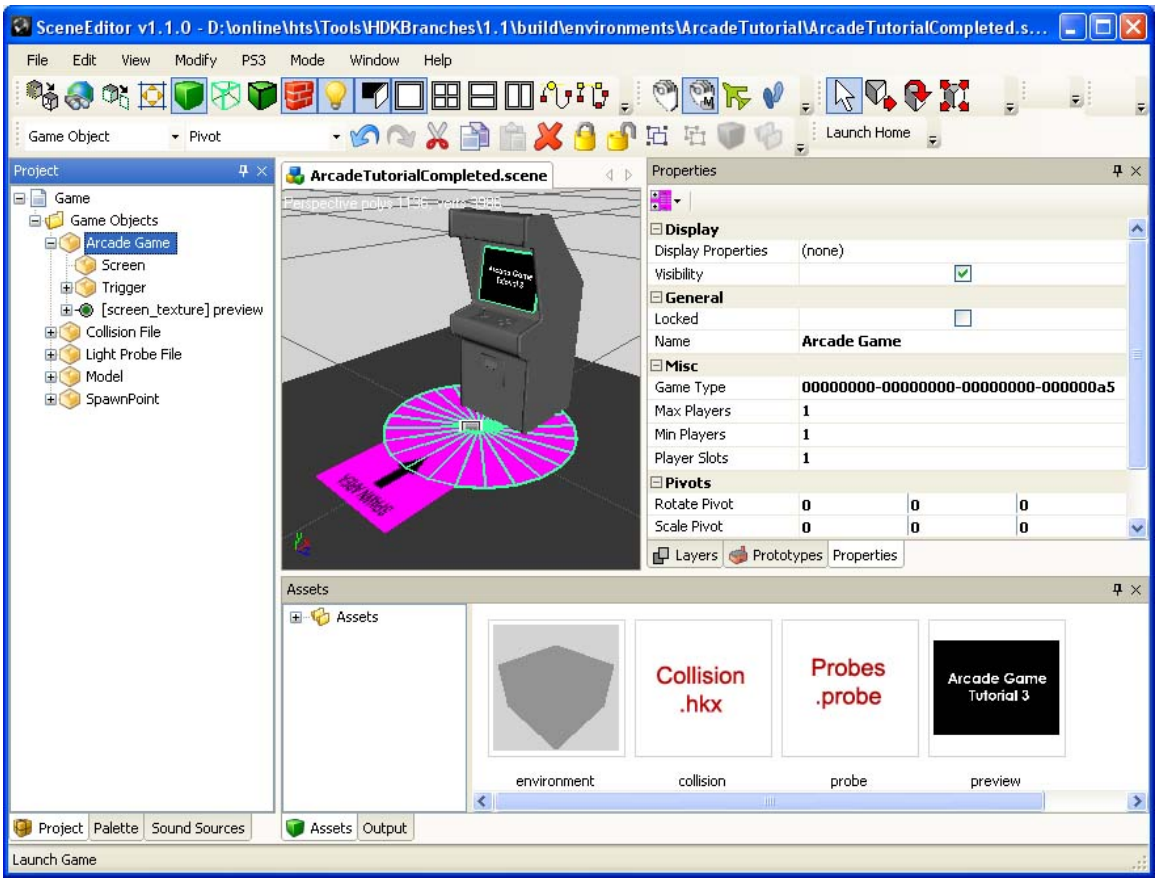
4. Now, we must create the arcade game. From the Palette panel, select the Objects section. Drag an Arcade Game object into the Game Objects folder in the Project panel. Your arcade game should be visible in the 3D view. The circular portion of the arcade game is the trigger. If an avatar enters the trigger volume, they will receive an on-screen message inviting them to play the game. The rectangular portion represents the plane on which the arcade game will be projected. This quad should align as closely as possible with the screen of the cabinet geometry. However, after initial creation of the arcade game element, you will see that it is not aligned with the arcade cabinet geometry. Set the rotation of the Arcade Game object o 0, 90, 0. Now use the translation tool (which you can access by pressing the M key, or selecting View-Move from the menu) to slide the arcade machine along the X axis until the arcade screen sits slightly in front of the arcade machine geometry.

5. Now we need to associate the Arcade Game object with a specific playable game. With the Arcade Game object selected in the Project panel, take a look at the Properties panel. Set the Game ID to the ArcadeTutorial3 object.

> ⚠ Arcade games can only be single player. Therefore, Player Slots must be set to 1 (the default).

6. The last thing we need to do is apply a preview image to the screen of the arcade cabinet. This is the image that is displayed when the game is not active. Drag the file <HDK_INSTALL_DIR>\build\ArcadeGames\ArcadeTutorial3\preview.dds into the Assets panel. Then assign the asset to the screen by dragging it to the screen_texture node of the Arcade Game object in the Project panel.

With the scene editing complete, select File > Save. The completed scene should look as follows:



Now that we have a finished .scproj project, we can load it into the runtime and play our arcade game from within the space. From the Scene Editor, select PS3 > Launch Home. Your avatar should spawn in front of the arcade machine. Walk forwards a little and a message should appear offering you a play on the arcade game.

The scene should look something like this:



Deploying an Arcade Game in Furniture

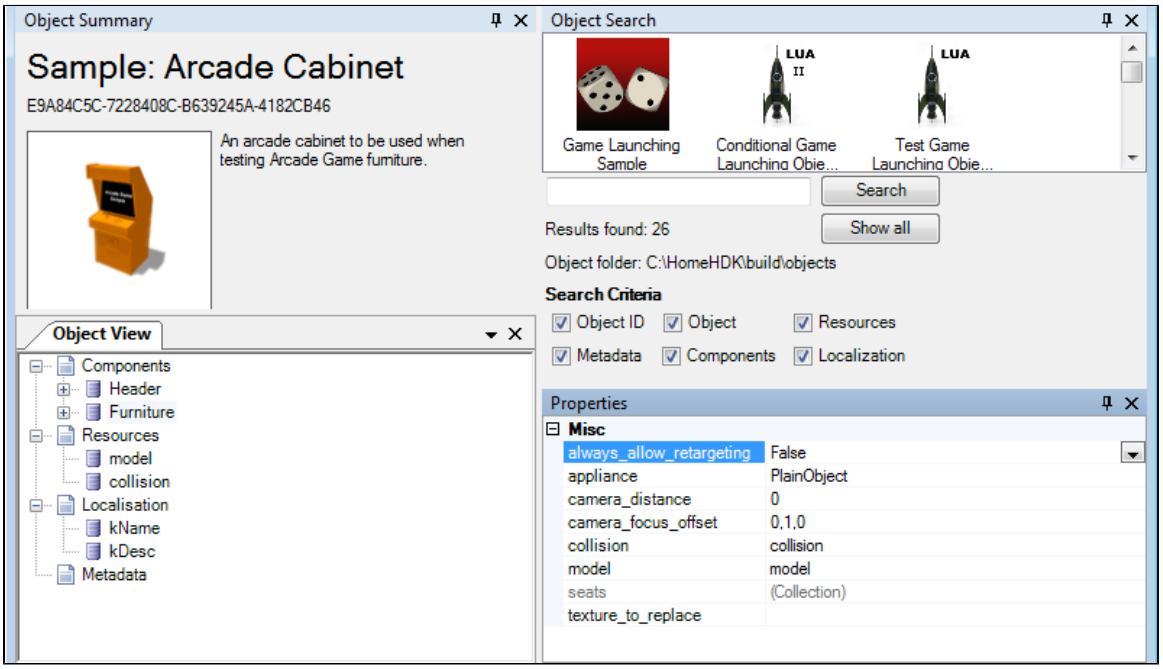
Arcade games are not restricted to being placed in fixed positions in specific scenes. You can also assign an arcade game to PS Home furniture items. As such, the arcade game-hosting furniture can then be selected and placed anywhere within a personal

space.

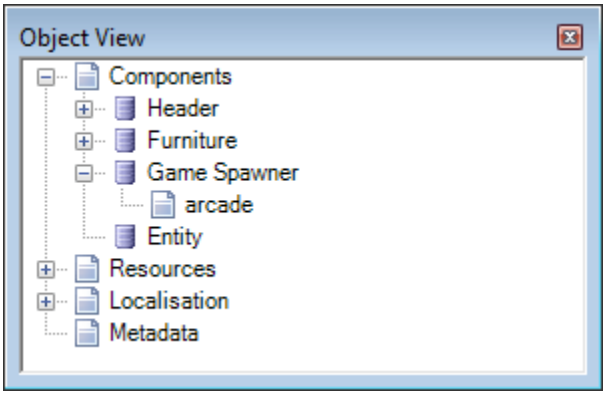
You do not require scripting to associate an arcade game with an item of furniture. You associate the furniture object with the arcade game object in the Object Editor. This association is performed by the Game Spawner component which you must add to the furniture object and configure.

To deploy an arcade game in furniture:

1. Build and export a furniture item. For more information on how to do this, see Furniture and Decorations.
2. Select the exported item of furniture in the Object Editor. It should look like the following:



3. Click Add New Component in the toolbar and then select the Entity component.
4. Associate the furniture with an arcade game object by clicking Add New Component in the toolbar, selecting the Game Spawner component, then selecting Arcade Game and clicking OK. The object view for the furniture item in Object Editor should now look as follows:



5. Select arcade under the Game Spawner object in the Object View panel and configure the following properties in the Properties panel:

    a. Enter the object ID of the arcade game that you want tied to the furniture item in the default_game field. For example, if you wanted ArcadeTutorial3, you would enter 00000000-00000000-00000000-000000a5.
    b. Add a DDS texture resource to the object and assign this resource to the default_screen_texture field. This is the texture image that is displayed on the screen when the arcade game is inactive.
    c. Enter the bottom left coordinate of the arcade game screen in the local space of the furniture item in the screen_bottom_left_point field.
    d. Enter the top right coordinate of the arcade game screen in the local space of the furniture item in the screen_bottom_right_point field.
    e. Enter the top right coordinate of the arcade game screen in the local space of the furniture item in the screen_top_left_point field.

6. Run HomeDeveloper.self from the Target Manager with no command line parameters. This takes you to the default personal apartment.

7. Press the Start button on the connected controller and select Redecorate.
8. Select and place your arcade furniture item. You should see something like the following:



Mini-games

For Japanese Live Page
最新の日本語版ページはこちら 「ミニゲーム」

What is a Mini-game?

The term Mini-game applies to objects with a MiniGame component. The component grants certain functionality (for example, callbacks) and access to the MiniGame Lua library. The MiniGame component is not a dedicated component for running games in PS Home, but a component that allows an object to run a script in a scene and to respond to users through its callbacks.

The MiniGame component:

- Automates access to an object by users (i.e. allows a user to 'join' the mini-game).
- Allows users to easily interact via an object (i.e. users can interact with each other within the mini-game).
- Can automate the replication of an object's state between all users through the Welcome Message system.
- Can be limited to local player interaction only, with no network messages.

Mini-games in PS Home can be multiplayer games like Pool, Bowling, or Chess. Mini-games can also be single (local) player interactions, like an NPC (Non-Player Character), solitaire, or a soda machine that can be used independently (i.e. up to 60 users can use the soda machine, but their state will not be replicated across the network).

A Mini-Game Sample Pack is available for download from https://home.scedev.net/projects/samples. The pack contains many examples of what can be done with Mini-games.

Networked Vs. Sessionless Mini-games

The MiniGame component contains a boolean property named Network Session, which defines whether the object can use Welcome Messages and communicate users' states to other users.

- Networked Mini-games: Set this property to True when network communication is necessary. This provides a framework through which you can send network messages and replicate game-state to all the players currently participating in the minigame, and those outside of it.

> ⚠ You must also add the Network component to the object, and script the network communication messages before players can interact or spectate.

- Sessionless Mini-games: Set this property to False when network communication is not required, and when the mini-game has no

multi-player or multi-user features.

- Sessionless mini-games cannot send Welcome Messages and cannot communicate users' states across the network.
- The actions of the player and the mini-game are seen only by the player, so unlike with networked mini-games, sessionless mini-games cannot have spectators or support multi-player.

Sessionless Mini-games

A sessionless mini-game is an object with a MiniGame component that is designed for single-player interactions. They are simpler to implement and can be accessed more quickly as they don't need to rely on communication to the session host to retrieve player list and state etc. However, it provides no network functionality and hence anything the local player does, or anything that the mini-game object does while the local player is interacting with it, is seen only by the local player. Every user in a scene can interact with the same Sessionless mini-game at any time without waiting for server responses or other users.

Sessionless mini-games must be embedded in a scene.

If you are using Sessionless mini-games, see:

- Mini-game Design
- Commerce in PS Home
- Profiling in the Client

Advantages of Sessionless Mini-games

- You do not need to create and manage Welcome Messages.
- Users do not wait for server (remote player) responses.
- Every user in the scene can play and interact with the same sessionless mini-game simultaneously. There is no queuing or instancing.

Sessionless Mini-game Restrictions

- No network communication (that is, communication between local player and remote user) is permitted within the mini-game.
- There is no access to Netproperty bags, Network component, `OutboundMessage` and `ReceivedMessage` Lua functions.
- Sessionless mini-games must be embedded in a scene..

Examples of Sessionless Mini-games

In a Carnival scene with several rides, there is an non-player character (NPC) who gives directions. The NPC asks the local user which ride they would like to go to, and then points the local user in the right direction.

- A solitaire card game. The game is a single-player game, but instead of placing 60 arcade games in a scene so that more than 1 user can play the game at a time, embed just 1 sessionless solitaire mini-game.
- Tutorials that users can read/interact with before joining a networked mini-game.

Creating a Sessionless Mini-game

To create a sessionless Mini-game:

1. In the Object Editor, select the MiniGame component in the Object View panel.
2. In the Properties panel, enter False in the Network Session field.
3. Enter 1 in the Session Size field.
4. Make sure that the object does not have a Network component.
5. Add a Scene Object component.
6. Continue making the mini-game as normal.
7. Add the mini-game as an embedded object in a scene.

Lua Scripts

PS Home mini-games are implemented in Lua script. The Home runtime executable contains a Lua VM and can therefore load and run a mini-game script. See Lua Scripting and Lua Features.

Typically, a standard Lua scripted game that you write might be of the form:

```
InitializeGame()
while (quit == false)
{
      UpdateGame()
      RenderGame()
}
TerminateGame()
```

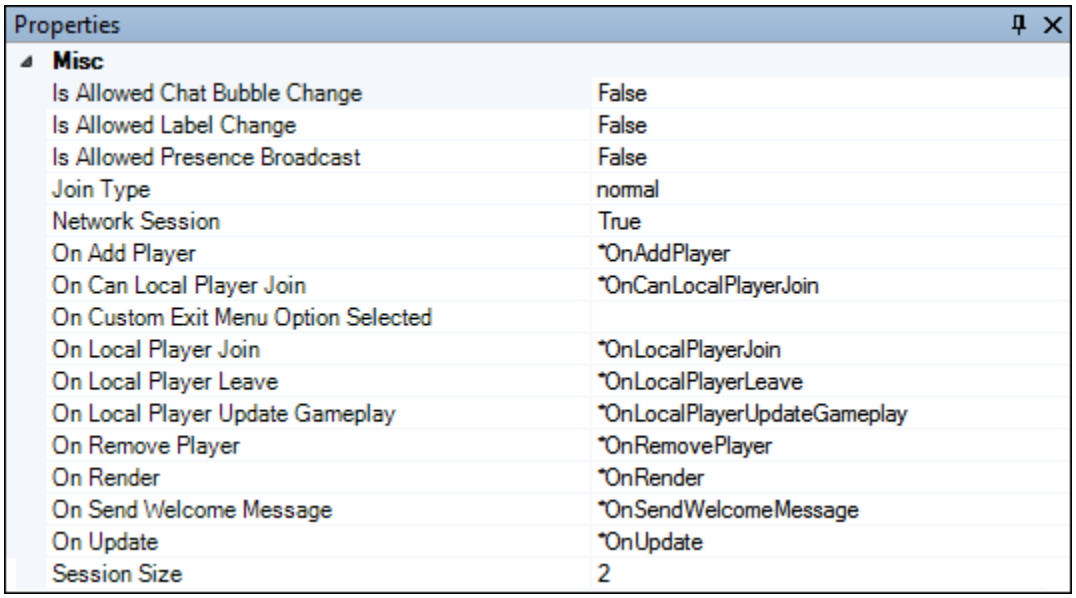
The Lua mini-game approach is somewhat different. Instead, a number of named callbacks must be provided in your script which the runtime will call. Thus it is the runtime that owns the game loop and calls into your script when certain events occur. Think of these functions as callbacks, performing various important tasks relevant to a Mini-game.

Mini-game Component Properties

Mini-games are objects. Objects are constructed in the Object Editor and built from a number of components. The components provide the functionality that the game needs, whether it be pad support, network messaging, or OSD support.

The only required component for a mini-game is the MiniGame component. This component subscribes to the `MiniGame` library and related functions common to most interactive 3D content. It also registers an update/tick function.

If you add the MiniGame component to your mini-game object in the Object Editor, the mini-game has the following properties:



You can set the mini-game properties as required:

Property	Description
Is Allowed Chat Bubble Change	Allows chat bubbles to be toggled on and off using the function `Person.EnableChatBubbles()`.
Is Allowed Label Change	Allows the labels to be changed using the function `Person.SetLabelText()`.
Is Allowed Presence Broadcast	Enables broadcasting to the XMB when a user is playing this game. The scene which the game is within must have its Broadcast Presence set to Scene and Game. See [HDKdraft:Scene, Object and Asset Properties] for further details.
Join Type	A drop-down list for you to choose how to prompt users to join a mini-game. You have three options: • `normal`: Players are shown a full prompt and then enter the normal targeting system. • `auto`: Players join when they enter the targeting region. • `minimal`: Players are offered a single prompt to join.

Network Session	A boolean that determines if a mini-game is networked (set to True), or sessionless (set to False). If set to False, leave the On Send Welcome Message field blank, the Session Size field should be set to 1, and the object should not have a network component.
On Add Player	A user-defined function that is called when a player joins the game, regardless of whether or not the player is the local player.
On Can Local Player Join	A user-defined function that is called every frame if a player has not yet joined but is inside mini-game trigger volume (as set in the Scene Editor). Returns a boolean indicating whether or not it is legal for the avatar to join the game.
On Custom Exit Menu Option Selected	A user-defined function that is called when the user selects the script defined (see `MiniGame.AddCustomExitMenuOption()`) custom option in the Exit Menu of the MiniGame.
On Local Player Join	A user-defined function that is called when a local player joins the game.
On Local Player Leave	A user-defined function that is called when a local player leaves the game.
On Local Player Update Gameplay	A user-defined function that is called every frame to update the game state for a local player that has joined the game.
On Remove Player	A user-defined function that is called when a player leaves the game, regardless of whether or not the player is the local player.
On Render	A user-defined function that is called once per frame after `OnUpdate` and `OnLocalPlayerUpdateGamePlay`.
On Send Welcome Message	A user-defined function that is called when a connected client requires a full refresh of the current game state. You don't have to use this system if you are not concerned with replicating the state of the game to 'spectators' (i.e. other people in the scene not currently playing the game), but if you do then in the callback, your 'game host' (i.e. the source of truth for the current state of the game) should construct a refresh message and send it to the client requesting the refresh. Leave this property blank if you are using a sessionless Mini-game.
On Update	A user-defined function that is called once per frame before `OnLocalPlayerUpdateGamePlay` regardless of whether or not the local player is joined.
Session Size	Use this property only if you are using an Scene Object component. Sets the maximum number of players in the mini-game. This number must equal the `<SESSION size = "X">` number in the object's network.xml. The Session Size property affects only objects with an embedded scene object component. It has no effect otherwise.

If your script does not need to respond to any of the callbacks above, you can leave them undefined.

Mini-game Design

Mini-game Design From a Local Player Perspective

When using the `MiniGame` component in PS Home you need to consider both efficient Lua script design, and how to implement network replication. Network replication communicates to remote players a local player's actions, and vice-versa.

You should try to design Mini-games from a local perspective; that is, limit the mini-game's control of the game to the local user as much as possible. Send messages to other users about the local user's actions rather than manipulating remote users directly from the local viewpoint.

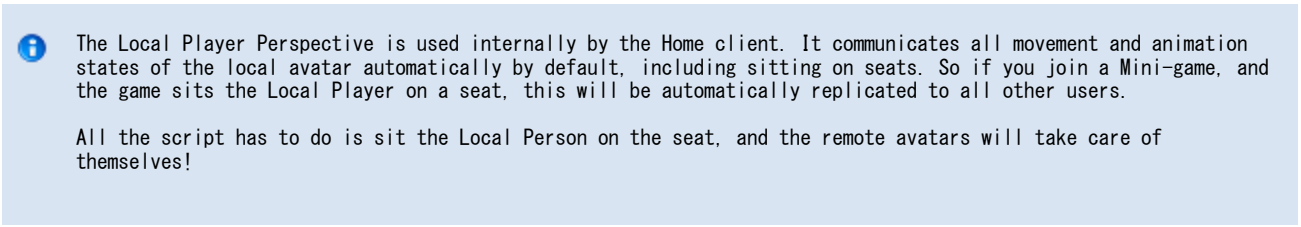
See the `LocalPlayer` library in the Lua API Reference.

Local Player Perspective

A Mini-game with a Local Player Perspective means you write the script with a view to dealing with only the Local User as much as possible, and leaving the Remote Users to handle themselves. This approach can dramatically simplify the script, and make the Mini-game more robust.

Example 1

You could make the Mini-game responsible for all the avatars' positions and animations while in the Mini-game, such as having all the avatars sitting at a table, or having them all perform certain animations in certain positions. You could just grab each avatar and tell it to sit on a seat, or position them in a pose. But, with the Local Player Perspective you don't need to, instead you can just rely on the Local avatar informing all other users about their position and animation state. Just tell the Local avatar what to do, and have it replicate its state to all users.



The Local Player Perspective is used internally by the Home client. It communicates all movement and animation states of the local avatar automatically by default, including sitting on seats. So if you join a Mini-game, and the game sits the Local Player on a seat, this will be automatically replicated to all other users.

All the script has to do is sit the Local Person on the seat, and the remote avatars will take care of themselves!

Example 2

Consider a game of pool. User 1 pots the cue ball, then User 2 pots the cue ball straight afterwards. What happens with the two shot rule?

Good Practice

The best way to communicate each user's actions is to calculate and retain the data for the local user, then send the remote user the results of their actions. So, User 1 pots the cue ball, decides User 2 is therefore entitled to two shots, and sends the results. User 2 then pots the cue ball, calculates that User 1 is now entitled to two shots, and sends the results. The users are entirely responsible for their own actions and the consequences, rather than having to work out what they and the other player did, and what the combined consequences are. This method may slightly increase network traffic, but as long as the messages are small in size it's worthwhile for the simplified Lua script.

Poor Practice

The improper way in this situation would be to have each player calculate the actions of both players and retain the game state independently. This method results in an untidy script that must constantly check both user's actions. It therefore increases the risk of errors, of data being wrong, and of the clients becoming out of sync.

Having a Mini-game Owner

There are situations where it is better that a single player acts as 'owner' (or 'server') of the Mini-game. Consider a poker game. It is normally easier for the deck of cards to be owned by one of the players, and for the owner to tell the other players which cards they are dealt. If not, each player would have to keep their own version of the deck of cards and make sure that it is synced properly with those of the other players.

During the game, the cards will be shuffled, and players will leave and join the game. Managing each player's hand, syncing the shuffling of the deck, and managing players joining and leaving the game can be difficult. To simplify the process, you can designate one player as the owner of the deck. The owner controls the deck and hands, and communicates this information to the other players. The Local Player Perspective is still used, it's just the owner has to communicate a little more information that the other users. The other users only need to tell the owner which cards they want to discard. This system may possibly increase network traffic slightly, but it makes managing the game much simpler and robust.

Messages and Network Traffic

Use messaging efficiently and sparingly to keep network traffic as low as possible. Messages should be as small as possible, and as infrequent as possible. Send only crucial information that remote users need to keep the Mini-game in sync with the Local User.

When writing script, weigh the advantages of sending a message as opposed to calculating information independently on each user's machine. Sending messages increases network traffic, but if a slight increase in traffic leads to simpler code, the cost is worthwhile.

For fast moving objects, do not send position update messages every frame, or anything close to every frame. To update fast-moving, or randomly moving positions smoothly and frequently (such as an object that is controlled directly by the player in real time), take positional values every few seconds and interpolate the positions.

Try to find ways to reduce message traffic. For example, in a game of pool, when a player takes a shot, do not send regular positional update messages telling other users where all the balls are while they are moving. Interpolating the balls' positions

may be inaccurate and look clumsy, as well as very message intensive. Instead, send a message that gives the position, angle and power of the shot to all of the remote users, who can then replicate the shot on their machine. This solution should simulate the same outcome without the need for any other messages.

> ⚠ For Mini-games that do not use network messages, you must set the network_session property to False in the Mini-game component.

For more information on profiling network statistics while in a mini-game, see Profiling in the Client.

For more information on Network Communication, see Adding Network Communication.

The Mini-game Welcome Message System

Welcome messages are used to inform users entering a scene of the state of a Mini-game in progress. For example, two users are playing a Chess game when a third user enters the scene. A script defined Welcome Message can be sent to the new arrival to tell them the positions of the Chess pieces. They can then spectate, with an accurate representation of the state of the game.

> ℹ At least one user must have joined the Mini-game for a Welcome Message to be sent.

See the MiniGame library in the Lua API Reference.

If the mini-game does not support non-playing observers, you do not need Welcome Messages. To disable Welcome Messages, call the MiniGame.DisableWelcomeMessages() function at the script initialization.

To set up Welcome Messages:

1. Write a function in the Lua script that sends Welcome Messages. This function will be assigned to the on_send_welcome_message callback:

```
WELCOME_MESSAGE = 1
function OnSendWelcomeMessage()

local newlyArrivedPersonId = MiniGame.GetCurrentFocusPersonId()
local message = OutboundMessage.Create(WELCOME_MESSAGE, 4)
OutboundMessage.WriteInt32(message, gameState)
OutboundMessage.Send(message, newlyArrivedPersonId)

end
```

2. Write Lua script to receive, recognize and process the Welcome Message. This must be in the on_message_received callback found in the network component.

   Use the Lua API function MiniGame.ConfirmReceivedWelcomeMessage() to confirm that a Welcome Message is received.

   This step is important, because otherwise the Mini-game system does not know whether a valid welcome message has been received, and therefore does not know if it has a valid one to send. Also, without confirming a welcome message, the user will be prevented from joining the mini-game.

```
function OnReceivedMessage()

local message = ReceivedMessage.GetEventMessage()
local msgtype = ReceivedMessage.GetType(message)
if (msgtype == WELCOME_MESSAGE)then

gameState = ReceivedMessage.ReadInt32(message)
MiniGame.ConfirmReceivedWelcomeMessage()

else

- Handle other message typesend

end

end
```

3. Tell the Mini-game system when it is safe to allow a Welcome Message to be sent. The system does not do this automatically, so you must set when it is safe and when it is not safe to send a Welcome Message.

   Use the function `MiniGame.SetSafeToSendWelcomeMessage()` to make it safe.

It is not always safe to send a Welcome Message. For example, in a game of pool, while the balls are moving after a player takes a shot, it is not valid to send the Welcome Message to the new arrival in the scene. Starting the ball physics while the balls are moving for a user entering the scene is not a trivial task for the system. Make sure that you send the Welcome Message only when the balls are stopped.

The following code is a sample Lua script that makes it safe to send welcome messages when all of the pool balls have stopped moving. The argument `areAllPoolBalsAtRest` returns `false` when the balls are moving, and `true` when the balls have stopped.

```
function OnUpdate()

MiniGame.SetSafeToSendWelcomeMessage(areAllPoolBallsAtRest)

end
```

In the example game of pool, the Mini-game system waits until all the balls stop before sending the Welcome Message. During this time, the new arrival does not have accurate knowledge of where the pool balls are, and is excluded from joining the Mini-game. This situation is unavoidable, but is fairly trivial.

By contrast, a Chess game may not have this problem. The Welcome Message is always safe to send, as all the moves are instant in terms of messages between players.

Welcome Message Requirements and Tips

The following guidelines apply only to Mini-games that have Welcome Messages enabled.

- For the MiniGame component's `on_send_welcome_message` callback: the function defined in this callback must send a Welcome Message. The Welcome Message should contain the mini-game's state, which includes any information that the newly arrived user needs to know before being able to join or watch a Mini-game. For example, the position of the balls in a game of pool.
- For the Network component's `on_message_received` callback: the callback must be able to recognize what the Mini-game has defined as the Welcome Message, and use `MiniGame.ConfirmReceivedWelcomeMessage()` to confirm it is received. Users cannot join or watch a Mini-game until this is done.
- Even when Welcome Messages are enabled, Mini-games do not necessarily have to allow spectators to watch the game in progress. When a Welcome Message has been sent and confirmed, it is your choice how and whether game state updates occur for spectators. However, if the Mini-game has absolutely no need to replicate any game states for users not playing the game, you may as well disable Welcome Messages.

Accessibility

Make sure that your game is available to be played whenever a user wants to use it, or within a short time of them attempting to play. The excitement around a new space or game is one of your biggest assets, and a user who is denied entry to your game on their first visit may well not be happy to wait, creating a bad first impression. An unhappy user is much more likely to be vocal on a forum regarding it than a happy one. There are several ways you can achieve accessibility:

- Create several physical instances of your game in a scene so that the chances of a slot being free are increased.
- Allow your game to create instances of itself when full. This works particularly well when the player isn't represented in the game as themselves. So in a racing game, for example, hide the cars not involved in your race, and only show spectators one race at a time.
- Use space instancing to allow new instances of your space when users find all games full. For example, if you have 3 poker tables in a room, have a doorway saying 'More Tables Upstairs'. If the user can't find a free table, they can go through the doorway and new instance of the space is created with a new complement of tables. This can be done exponentially using something like the following pseudo-code:

```
relocateInstanceParam = currentInstanceParam + 1
LocalPlayer.Relocate( "PokerHall", relocateInstanceParam )
```

- Use the queueing system so that users attempting to join know that they will be able to join shortly. Queueing has various pros and cons and requires careful consideration when being implemented, as it will have implications for the design of your space. See our Queueing Best Practice guidelines for further information.

Commerce Points in Mini-Games

You can open a Commerce Point from within a mini-game using the function `System.CommercePointOpen()`. When you open Commerce Points in mini-games be aware of the following potential issues:

- The user has restricted knowledge about anything happening in the game, which means that the user may be timed out of a turn in the game, or be removed from the game completely, without seeing any warnings.
- Users cannot play the content in a mini-game and browse a Commerce Point in a Mini-game at the same time. When a user accesses a Commerce Point in a Mini-game, they stop interacting with other users in the Mini-game, but they also prevent queued users from joining the it, because they are still in the game while accessing the Commerce Point.
- Users may need to queue for a Mini-game to access a Commerce Point.

> ⚠ It is advised not to use Commerce Points in Mini-games unless it is part of the Mini-game experience. If Commerce Points are used, it is advised not to allow queueing for the game. If at all possible, separate the Commerce Point from the Mini-game.

For more information, see Commerce in PS Home.

Testing Mini-games

The QA process checks that your content is within PS Home limits and meets the requirements and tests scripts to a degree. However, you must also test that your script (and Mini-game) performs well:

- Test not only how a player would normally progress through the Mini-game, but also the effects on users outside the Mini-game.
- Check for known potential pitfalls or problems that the Mini-game may have.
- Notify your Home Account Manager of any concerns or problems you may have with your Mini-game, so that the Account Manager can inform QA.
- If a scene contains a Mini-game with an invalid object reference or a reference to a missing Mini-game object, the Mini-game will not work in the scene.

Make sure that you test the following items:

- Welcome Message Testing: To test welcome messages you will need to run several machines, using different numbers of users in the Mini-game (with at least the minimum required for game-play), and have users leave and enter the scene. When a user enters the scene, check that the game's state is as it should be, and that the new arrivals have access to the game if there is room to join, also check that the appropriate pop-ups appear if there is not.
- Menu Screen Interaction: Make sure there are no issues when a player accesses the Menu Screen during a mini-game or Realtime Game.

For more information on manipulating the camera with script, see PS Home Camera.

Profiling

Profile your Mini-games during and after development to check that they are within HDK limits and requirements. Check that:

- Not too many messages are sent within a specified time period.
- The messages are not too large.
- Mini-game resources do not take up too much space in a scene.
- The Mini-game runs fast enough.

For more information, see Profiling in the Client.

Maximum Number of Players in Mini-Games

This section describes player definitions for networked and sessionless mini-games.

Networked Mini-games

Mini-Games

How you define the maximum number of players within a mini-game depends on where the mini-game is used:

- Mini-games in active items use the `session size` tag in the game's XML.
- Mini-games in a scene use Player Slots. If a session size is defined, these values must be the same.

Mini-Games as Active Items

When creating a mini-game as an active item, the object must include:

- A Network component

- A network XML file

This XML file must contain a definition for session size, which determines the maximum number of players.

Setting the Session Size

1. Open the object using the Object Editor.
2. Click Add New Component in the toolbar and then select Network. The Network component is now listed in the Object View panel.
3. Create an XML file in your object's folder in the <HDK_INSTALL_DIR> with the following format:

```
<NETWORK_DEFS>

    <SESSION size="X" />

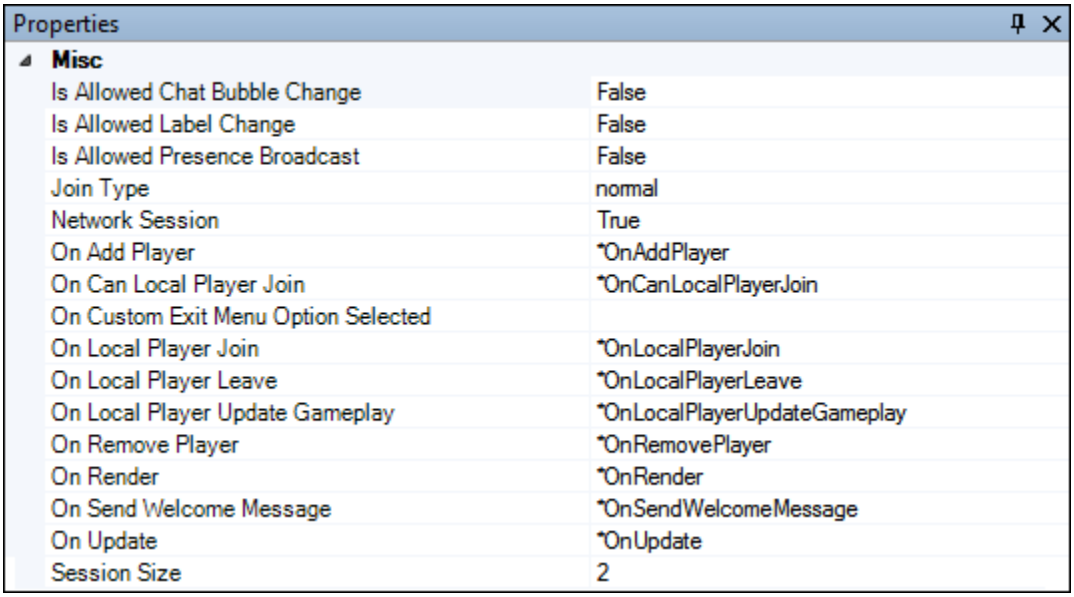
</NETWORK_DEFS>
```

(Where 'X' is the maximum number of players.)

4. In the Object Editor, click Add New Local Resource in the toolbar, and add the XML file that you created.
5. Select the Network component and select the XML resource for the definition property.

Mini-game: Set the session_size

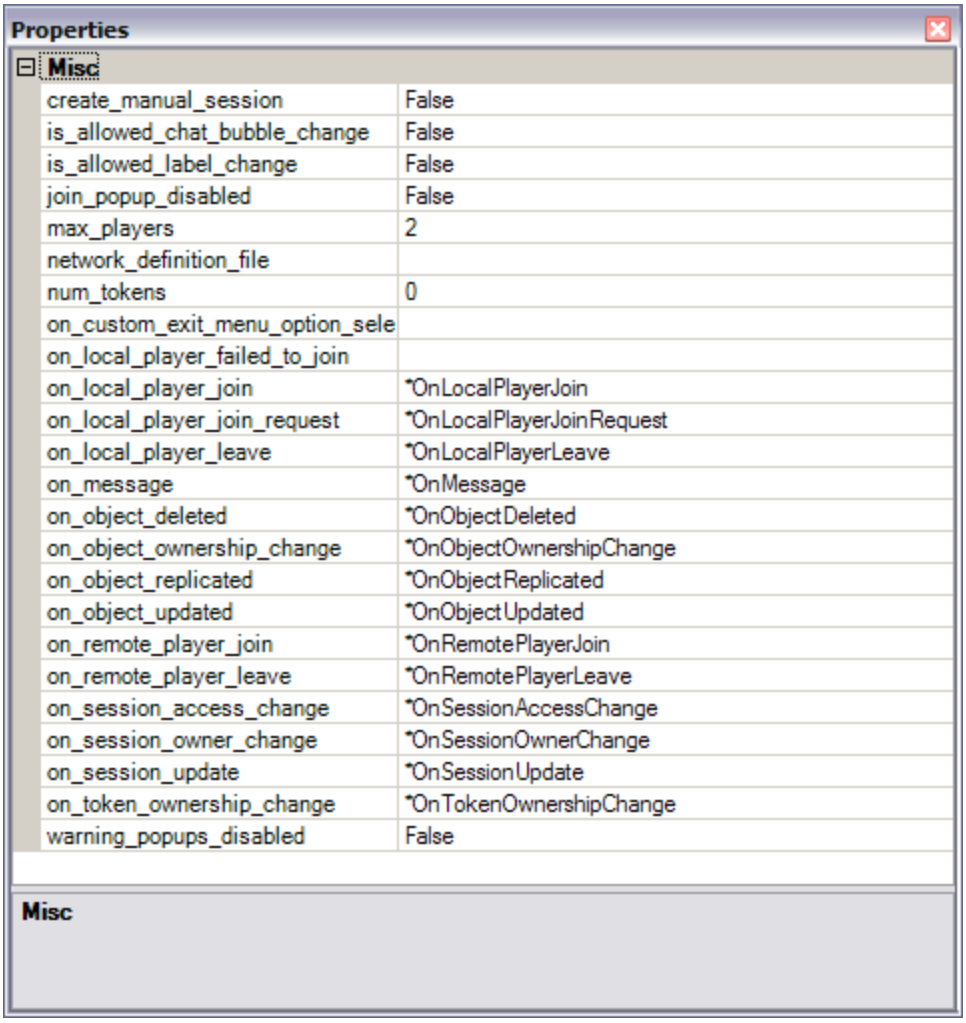
1. In the Object Editor select the mini-game component to display the following in the Properties panel:



2. Enter the maximum number of players in the mini-game in the session_size field. This number must equal the <Session size="X"> value in the object's network.xml file.

Realtime Game: Set the max_players

1. In the Object Editor, select the RealTime Game component in the Object View panel to display the following in the Properties panel:



2. Enter the maximum number of players in the game in the max_players field.

Mini-Games in a Scene

To set the maximum number of players for mini-games in a scene:

1. Go to the Scene Editor, after adding a Mini-Game object.
2. In the Properties panel, enter the maximum number of players for the mini-games in the Player Slots field.

> ⚠ If the session size is defined for the mini-game in the Network component XML, this value must be the same as the Player Slots.

Sessionless Mini-Games

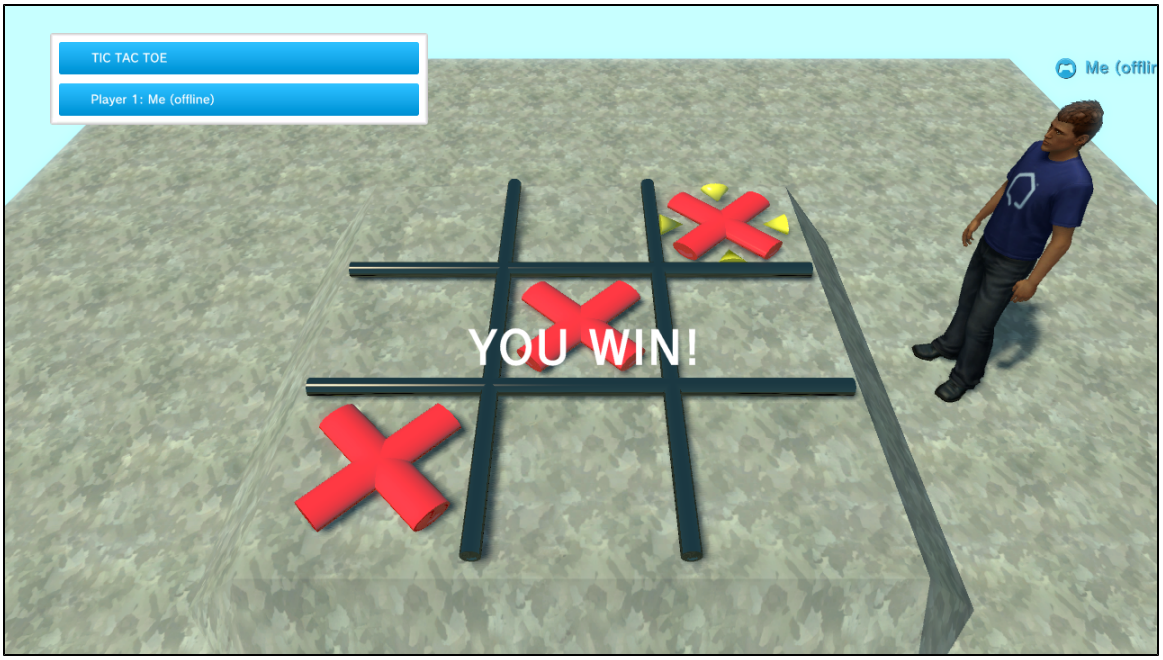
Sessionless Mini-Games are single-player. Unlike arcade games, every user in a space can play the same sessionless mini-game simultaneously.

- The session size must be 1 in the Object Editor.
- If adding the mini-game to a scene and using a Mini-Game object, the Player Slots property is ignored for sessionless mini-games.

Example Mini-games

Many example mini-games are available for download from https://home.scedev.net/projects/samples. Included in the mini-games samples are Tic Tac Toe and Simon.

Tic Tac Toe is the classic turn-based 'noughts and crosses' game:



And Simon implements the memory game where the player must remember an ever increasing sequence of colors:



The documentation makes frequent reference to the Tic Tac Toe example game in particular, so it is worth acquainting yourself with the source code (found in <HDK_INSTALL_DIR>\build\MiniGames\TicTacToe\main.lua).

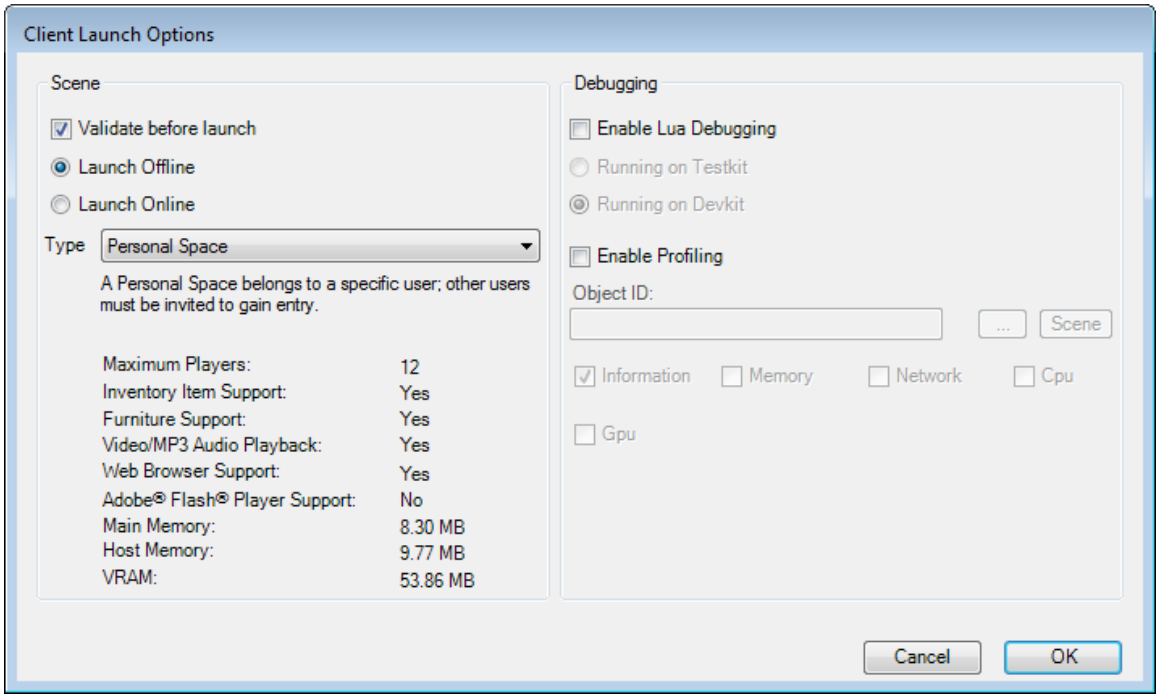
Running the Tic Tac Toe Sample

To run Tic Tac Toe:

1. Run the Scene Editor.
2. Open the following scene:

   <HDK_INSTALL_DIR>\build\environments\MiniGameTutorial\MiniGameTutorial.scene

   You will be prompted to update the file to a .scproj file.
3. Select PS3 > Launch Options to display the Client Launch Options dialog:



4. Select Launch Online.
5. Select Game Space (16 players) from the Type drop-down list.
6. Enter a scene key in the Scene Key field.
7. Click OK.
8. Select PS3 > Launch Home.
9. To play the game against another person, follow the above steps on another person's PC. If you want to launch the executable on two devkits from same PC, then open Target Manager and load and run HomeDeveloper.self on another reference tool target with the command line parameter:

```
--sceneId minigametutorial
```

If you cannot see the other player, make sure that both PlayStation®3 targets are logging in with sp-int accounts that were created against the same region.

Queueing Best Practice

For Japanese Live Page
最新の日本語版ページはこちら 「待ち行列の推奨実装方法」

Queueing can be a great way to make your scene seem busy as well as adding an extra level of social experience to a space. However, it needs to be done correctly if you're to reap the full benefits and not deny users easy access to your games. It will also have a big impact on the overall design of your space.

When considering adding queueing to a scene, make sure that you're choosing it because it's the best solution for the problem you're having, and not just a quick-fix. Have a look at the section entitled 'Accessibility' on our Mini-game Design page for alternatives.

Here are some of queuing's pros and cons:

Pros	Cons
Everyone gets a chance to play	Some users will not have the patience to queue

Time spent in space per visit increases	Can create a poor experience if implemented incorrectly
Space looks busy (avatars queueing and avatars playing visible)	Very hard to debug as there are many possible causes to queuing issues
Good for creating spectators and making players feel like the centre of attention	Queue can appear incongruous in virtual environment
Queuing gives users the time and space to create their own unique social experiences together	

In addition if the queue times are short it can help build anticipation, excitement and demand for a game.

Best Practice Guidelines

Here's a set of guidelines on how to make the most of the queueing system should you choose to use it:

Make sure there are several instances of the minigame you want to use queueing for in your space.

Users should never be queueing for more than a minute or two. Calculate how many people in your space can play at once, so you know you'll have enough minigame instances without anyone having to queue for too long.

Have several different games in the space which can be queued for

If people are playing one game while queueing for another they'll hardly notice that they're even in a queue, but will be able to jump from game to game with minimal waiting

Encourage spectating

One of the benefits of queueing is that while people are queueing they can become spectators which can add to the social dynamic and appeal of the game. If this is part of your game design make sure that your minigame is easy to see, so that players feel like they are 'performing' to a crowd. This will also draw the queuers into watching the game and build excitement around it.

Make sure there are plenty of things for your queuers to see and do

If your game does not rely on other people spectating make sure your space has plenty of things to occupy your queueing users, particularly things which will raise interest in the event. Posters, scoreboards, animated features, twitter feeds, anything which will build a sense of a busy vibrant scene that is a pleasure to be in.

Make sure your games are short

Make sure your games last no longer than a few minutes. This will keep queue times down and also encourage rejoining the queue after playing. Each extra moment of game time is multiplied by the amount of queuers in front of you and can quickly add up or undermine the desirability of your game.

Avoid single-player games

Single player games maximise queue times, and should really be instanced instead. 2 players will halve your queue times instantly. Consider having your single player games as something for people who are queueing for multiplayer games to do whilst waiting.

Ultimately, creating a compelling game or games, in a space which encourages spectating, is one of the most effective ways of engineering a competitive, fun, social experience for your visitors.

Queueing impacts the design of the space

If you do choose to enable queueing in your mini-game, this will affect how your scene is designed. As described above, to use queuing effectively your space must have plenty to see and do when not playing. It must encourage spectating by allowing plenty of viewing space. It should show current scores, top scores, friends scores and so on. You will need space for all the instances of your game. As you can see this will need to be part of your design from the beginning so make sure that you make the decision to use queuing as early as possible.

Realtime Games

Realtime games are similar to Mini-games, except they use peer-to-peer communication, do not allow spectators, and restrict players to a maximum of 16. They are intended for games like first person shooters, where the client and players need to react instantaneously to their environment and other players.

Realtime gaming features:

- Communication between clients (users), reducing lag
- Sending messages with differing priorities
- Receiving ordered and unordered messages

With PS Home's Realtime gaming, messages between clients are peer-to-peer (P2P): the local user sends a message to each remote user in the game, instead of sending a message to the a Home server, which in turn sends out the message to all remote players. This increases the amount of messages that the local user must send, but it decreases lag since a message is sent directly to the recipient.

The messages that players send to each other are predefined in an .xml file, in a way similar to network property bags with the critical except that Realtime games maintain a session master. The session master holds all the critical game information, as well as token information. When a user joins the Realtime game, the session master sends all the necessary data necessary to set up the game, similar to welcome messages.

The session master also manages tokens (a table of items, power-ups, and an owner if it has one). If the session master leaves, the system automatically selects a new session master, and continues the game uninterrupted.

When players join a Realtime game they are given a numeric memberId, which ranges from 0 to (`RtGame.GetPlayerCount()`- 1). This memberId can be queried using the `RtGame` library.

When to Use Realtime Gaming

Fast-paced, quick response games like first-person shooters benefit from the Realtime Gaming, but this feature is not limited to such games. Use realtime gaming when you want to take advantage of peer-to-peer networking, the cleaner interface, or the game objects and tokens, and a predefined format for creating, sending, and updating player and game states.

You should use Realtime gaming when:

- The game requires high performance networking
- The limitat of a maximum of 16 players is not an issue
- You want to use network messages, objects and tokens to communicate states and ownership between players
- You want to support custom gaming sessions (see Joining a Realtime Game)
- You want to support multiple sessions of a game running from a single object

While you can achieve game state replication in mini-games using welcome messages and netproperty bags, the set-up is more laborious and inherently slower since communication must first pass through Home's servers before reaching users. The `RtGame` library and supporting features are meant to bypass mini-game limitations, and support fast, constantly updating games.

Memory Limits

The memory limits for a realtime game depends on its use.

- Embedded Object: The realtime game object becomes part of the scene and uses the scene's memory budget.
- Active: The object uses the active's memory budget.

Creating a Realtime Game

This section provides an overview of the steps required to create a Realtime game. Most of the work in making a Realtime game object is creating the art assets and writing the script.

Start by creating all the required resources. You can perform the following steps in any order:

- Create the art assets: This includes all model, collision, animation, geometry, and light files that you want to use in the Realtime game.
- Write the Script: When writing the Lua files, include in the main Lua file all of the RealTimeGame component callback functions.
- Write the Network XML file: This file can be written before and while writing the Lua scripts. This file contains all of the RTGame Network XML Messages and RTGame Network XML Objects that are sent P2P.

When you have created all the resources, open the Object Editor and perform the following steps:

1. Follow the standard steps to creating an object, either with the Object Editor or through the New Object Wizard.
2. Add all the resources (art assets, Lua files, xml files, sound, etc.)
3. Add a Realtime Game component, and configure its properties.
4. Add either an Active Item or Scene Object component, depending on how the object will be used, and configure the component properties.
5. Follow the standard steps for an object (such as adding thumbnail and age rating), as well as the steps specific to completing either an active or embedded object.

To set the maximum number of players:

- In the Object Editor, select the Realtime Game component and in the Properties panel set the max_players property.

Joining a Realtime Game

There are two ways that users can join games, by joining either a static session or dynamic session.

Static Sessions

Static Sessions are the basic way that games create sessions. The user walks up to a game object, joins the game and becomes the session master. Users then subsequently join until the minimum amount of players is met and then the game can launch. Static sessions are the basic method for creating game sessions and require no extra steps.

With games that use this method, only one session per object is available. That is, once the game is in progress, other users in the scene must wait until the current session finishes before they can join and play.

To run a Static Session, the create_manual_session property on the RealTime Game component must be False.

> ⚠ NAT 3 type users are denied access to all static Realtime game sessions. See Realtime Game Network Guidelines for more information.

Dynamic Sessions

Dynamic Sessions are game sessions manually created and manually populated by the game's script. In this case, the script decides when users join the session. The session master is still the first user to join the game (i.e. the first user to be added by the script). The advantage of dynamic sessions is that it allows the script to easily use conditions (for example, players with the highest scores) to create a session, and it allows the object to run multiple sessions of the game simultaneously.

Realtime games that support Dynamic Sessions also have access to the component. This component is intended to be used to setup and manage a lobby and assemble players.

Setup a Realtime game to Support Dynamic Sessions

1. Select the RealTime Game component.
2. In the Properties panel, set the create_manual_session property to True.
3. In your script, use the following functions to create a session and to join players:

    - **RtGame.CreateGame()**: Creates a realtime game session.
    - **RtGame.GetRoomId()**: To join a user to a session, you need to use the room ID.
    - **RtGame.JoinGame()**: Adds a user to a realtime game session.

The rest of realtime game development continues as normal, with the exception of the component. Those games that support dynamic sessions can use the component to create a lobby and organize players.

> ⚠ See Realtime Game Network Guidelines for information on other important issues, such as the need to prevent NAT type 3 users from joining dynamic Realtime game sessions.

Realtime Game Tokens

Tokens are the realtime game system's method for synchronizing between clients. A token is an array that contains the token number and its owner (or nil if there is no owner).

Token Number	Owner
1	player01
2	nil

The session master controls the token list, and has write permissions to the list. All other users have read-only access. Users must request token ownership from the session master, who grants ownership to whoever requests the token first.

You can use a token in the realtime game for whichever application you want, from controlling the game, to giving items, to declaring a winner.

For example, in an FPS game:

- The script assigns a grenade launcher to token 1.
- player01 walks over the grenade launcher, causing them to request token 1.
- If the request is successful, the grenade launcher is added to player01's inventory.

Tokens are not mandatory in a realtime game. You can have a maximum of 256 tokens.

Using Tokens

To use a token:

1. In the RealTime Game component, give the game a maximum of up to 256 tokens.
2. In the script, create events that call RtGame.RequestToken()

If RtGame.RequestToken() is successful, the client will call the on_token_ownership_change callback.

3. In the on_token_ownership_change callback, create events that correlate to each token, or to a series of tokens.

The on_token_ownership_change callback is also called when RtGame.ReleaseToken() is called.

Token Ownership

- A token can have only one owner.
- Only tokens without owners (owner == nil) can be given to players.
- To change owners, a token must be released from its current owner, and then assigned to the new user.
- If the session master changes, the new session master sends a broadcast of all token states.

Check Token Ownership

Call the function RtGame.RequestToken() to run the on_token_ownership_change callback. Each time RtGame.RequestToken() is called, the client calls the on_token_ownership_change callback. In the callback function, check that the token was successfully given to the target player by calling RtGame.GetTokenOwnerMemberId().

If you want to run only the on_token_ownership_change callback for tokens that have no owner, use the following code:

```
if RtGame.GetTokenOwner()==nil then

    RtGame.RequestToken()

end
```

This code calls RtGame.RequestToken() and the on_token_ownership_change callback only if the token has no owner.

> RtGame.RequestToken() does nothing if the player calling the function already owns the token. For example, player1 owns token 1. If player1 calls RtGame.RequestToken(1), nothing will happen.

Realtime Network XML

The Network XML defines game objects and messages that can be sent quickly to users by calling RtGame.SendMessage() with the appropriate arguments. With the Network XML, all the important messages about the game, its state, and objects within it are predefined in the XML, avoiding the need to create and manage these messages in Lua.

Consider the grenade example used in Realtime Game Network XML Message Tips. The grenade has a position, linear velocity, and angular velocity when in flight, but it also has a damage, and a damage radius. Each of these values can and should be predefined in the Network XML file. The Network XML defines the name and type of each parameter (for example, position, float). Use RtGame.SendMessage() to change the value of each parameter.

There is no limit to the number of messages or objects defined in a Network XML. However, the XML, and objects and messages created from it, use the Object's memory budget.

The Realtime Network Definitions

The Realtime Network XML has two definitions: messages and objects.

RTGame Network XML Message

RTGame Network XML Messages are standard network messages that are used to send data to other users in the session. Messages are sent P2P down one of four pipelines. They are used to send events, like the launching of a grenade.

RTGame Network XML Object

RTGame Network XML Objects are persistent objects which can be used to implement states and changes to the game or player. There are two types of RTGame Network XML Objects, personal and session. PersonalRTGame Network XML Objects are owned by the user who creates the object. The owner has write permissions, while it is read-only to all other users. Session RTGame Network XML Objects are owned only by the session master, who also has write permissions, all other users can read sessionRTGame Network XML Objects.

Any time the owner updates their RTGame Network XML Object, all other players receive that update through the on_object_updated callback.

RTGame Network XML Objects are created using the RtGame library.

Realtime Network XML Template

```
<NETWORK_DEFS>

    <message name=" " transport= " " >

        <field name= " " type= " " />

        .

        .

        .

    </message>

    <object name=" " type= " " >

        <field name = " " type= " " updateFrequency= " " transport= " " thresholdChangeRule= " " />

        .

        .

        .

    </object>

<NETWORK_DEFS>
```

XML Definition Attributes

The following attributes are available:

XML	Attribute	Description
<message>	name	The message's name. This is used to identify the message in Lua. In the RtGame.SendMessage() function, it is the messageType argument.
	transport	Specifies the delivery type of message. See Transport Types.
<field> Each data item in the message is defined using a <field> tag.	name	This is the name of the field.
	type	This is the data type. The data types are listed in the Field Data Type table.
<object>	type	This can be one of the two following values: • personal: The object persists as long as the owner is part of the session. The object is destroyed when the owner leaves. • session: The object persists for the duration of the session. Session objects can only be created by the session master. If a session master leaves, a new one is automatically selected, transferring the session objects to that user.
<field> Each data item in the <object> is defined using a <field> tag.	name	The name of the field.

	type	This is the data type. The data types are listed in the Field Data Type table below.
	updateFrequency	(in milliseconds) This value specifies how often a field is checked by the client to see if the <object> needs to be updated once the function RtGame.ReplicateToNet() is called. When the object is updated depends on the thresholdChangeRule.
	transport	Specifies the delivery type of message. See Transport Types.
	thresholdChangeRule	Indicates when the value should be updated to other users. See the ThresholdChangeRule Table below.
	changeMagnitude	Add this attribute only if the thresholdChangeRule is set to 'MAGNITUDE'. An update is triggered for any changes in value greater than this number.

Transport Types

Type	Description
OUT_OF_ORDER	The order of the data's delivery is not guaranteed. This is the most efficient type of message in terms of bandwidth, but should not be used for critical updates.
OUT_OF_ORDER,GUARANTEED	The message is guaranteed to arrive at the destination, but it is not guaranteed to arrive in the order that it was sent.
IN_ORDER	The message is not guaranteed to arrive at the destination, but if it arrives it arrives in the order that it was sent.
IN_ORDER,GUARANTEED	The message is guaranteed to arrive at the destination, and in the order that it was sent.

The transport type defines through which pipeline a message travels.

Each transport type defines though which pipeline each message is sent. An OUT_OF_ORDER message travels only through a OUT_OF_ORDER pipeline, and an IN_ORDER,GUARANTEED message travels only through a IN_ORDER,GUARANTEED pipeline.

This means that messages sent in the IN_ORDER and IN_ORDER,GUARANTEED pipelines is ordered within a single pipeline, but is not necessarily ordered relative to another pipe.
For example, a script sends four messages in this order:

M1- IN_ORDER
M2 - IN_ORDER
M3 - IN_ORDER,GUARANTEED
M4 - IN_ORDER,GUARANTEED

The remote client may receive these messages in any of the following orders:

M1 > M2 > M3 > M4
M1 > M3 > M2 > M4
M1 > M3 > M4 > M2
M3 > M1 > M4 > M2
M3 > M4 > M1 > M2
M3 > M1 > M2 > M4



When the order of messages is critical, send them all through the same pipeline.

ThresholdChangeRule

Value	Description
ANY	Any change to the field triggers an update.

NONE	This object is sent once to each player, and no further updates are sent. This can be used to send the initial game state to late joiners. If 'NONE' is chosen, the updateFrequency attribute is ignored.
MAGNITUDE	Triggers an update for any change greater than X, where X is the number specified in the changeMagnitude field.

Field Data Types

Data Type	Note	<Messages>	<Object>
int8		Y	Y
int16		Y	Y
int	Limited to Lua's number data type. It is not the full 32 bits.	Y	Y
uint8		Y	Y
uint16		Y	Y
uint	Limited to Lua's number data type. It is not the full 32 bits.	Y	Y
float16		Y	Y
float		Y	Y
MemoryContainer	This allows custom data. See Message MemoryContainer Type.	Y	

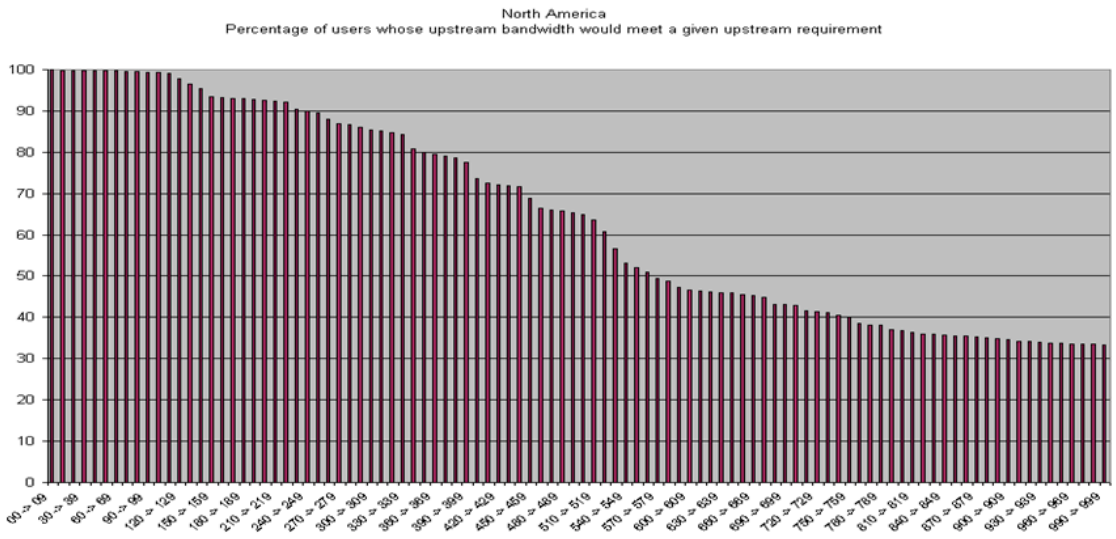
Message MemoryContainer Type

Type	Description	Example
Read start position	This uses the read/write position (i.e. the value returned from MemoryContainer.Tell()).	<field name="myData" type="MemoryContainer" ... />
size	You specify the size of the memory container. The size can be specified in two ways: 1. By using the size attribute. 2. By omitting the size attribute. The client uses MemoryContainer.GetUsedSize() to determine the size.	1. <field name="myData" type="MemoryContainer" size="10"/> 2. <field name="myData" type="MemoryContainer"/>

When a MemoryContainer is sent to the remote clients a new MemoryContainer object is created. The new MemoryContainer object is passed to the on_message callback. There is a significant increase in overhead due to MemoryContainers, and we do not advise their use for plain data types (e.g. a 4 byte integer).

Realtime Game Network Guidelines

The following graph shows the general upstream bandwidth for users in North America. The statistics for Europe, Japan and Asia are generally the same.



Realtime games have no set send and receive limits, but this does not mean that large packets of data could or should be sent every second. Instead, use general bandwidth rates to gauge acceptable send and receive rates.

Based on this graph, to make sure the a Realtime game works for 90% of North American users, limit the upstream use to < ~240kbps (~29 KB/s). This speed assumes ideal network circumstances and no other devices sharing the users' internet connections.

The Home client (and PSN) also uses some upload bandwidth, for maintaining the connection and global state, background downloads of other content (HTTP downloads consume some upstream bandwidth for ACKs / receipts), and (when available) voice communications traffic between groups and clubs. Assume that this other traffic uses about 8KB/s.

Often, other devices share a user's connection. Build in some redundancy so that a Realtime game still works well even if the connection is being abused elsewhere. Consider at least halving your estimated upstream capacity limit. Depending on how conservative you want to be, quartering may be more appropriate.

In the gauge given above, this would suggest a limit of between ~5KB/s - ~10KB/s as the maximum suitable rate for 90% of the users most of the time.

There is no local minimum for network communication. The lower the amount sent, the less likely the Realtime game will be affected by adverse conditions.

Network Limits

The Realtime game's network limits are determined by the state of the local player:

- Using the RtGame library: When the game starts, there are no explicit send/receive limits. For this reason, you must carry out extensive testing under all conditions for Realtime games. A safe, conservative gauge is between ~5KB/s - ~10KB/s.
- Not Using the RtGame library: Any network communication not using the RtGame libary, for example OutboundMessage, still needs to adhere to the network limits stated for the object type (e.g. scene network limits for an embedded object).

NAT Type Limits

There are 3 NAT types:

- NAT type 1 - No NAT (open to the Internet)
- NAT type 2 - Full-cone NAT / Restricted (port restricted) / NAT with UPnP
- NAT type 3 - Restricted (port restricted) or Symmetric NAT (not used on the PS3)

Type 1 should always be able to connect to other type 1 and type 2 users.
Type 2 should be able to connect with type 1 and type 2, especially with UPnP
Type 3 can often not connect with any other type.

> Because users with a type 3 NAT configuration can cause problems for other users in RtGame sessions, these users are prevented from joining an RtGame session.

If your script uses dynamic Realtime game sessions, always check the player's NAT type. If the player is NAT type 3, display an appropriate message informing them they are not able to connect to Realtime games.

For example: "Sorry your current network configuration means you are unable to join this game."

You can obtain whether a player is using NAT type 3 using the `System.GetNatType` Lua API command.

For static Realtime game sessions, it's mandatory that NAT type 3 users will not be allowed to join, and a popup to inform the player is automatically activated.

See Joining a Realtime Game for more information on joining Realtime games.

For more information see Documentation for PlayStation®Network Server Management Tools > NP Matching 2 Tools User's Guide

Realtime Game Network XML Message Tips

Optimization

Optimizing is minimalist; for performance reasons, keep the data set small. You need to have an idea about the data that will travel over the network. For example, when designing a projectile fired from a weapon in an FPS (first person shooter), the projectile can have many values depending on how it is modeled: caliber, velocity, position, explosive radius, explosive damage, mass (as affected by gravity), inertia, special behavior values (for example, armor piercing).

An inefficient use of bandwidth is to send all of these values each time a player shoots the projectile when only a few are needed. The most challenging part of Realtime game design is creating a compression scheme that sends as few messages as possible, and that those messages are as small as possible.

Extrapolation

In online gaming and Realtime gaming, it is impossible to ensure that all clients simultaneously have exactly the same information. However try as much as possible to create the illusion that all clients are synchronized. One way to achieve this illusion is through extrapolation.

Consider the design of a grenade. The grenade has the following data:

- position
- linear velocity
- angular velocity

This may seem like a reasonably small list, but it has 9 values total: position (x, y, z), linear velocity (x, y, z), angular velocity (x, y, z). Sending all 9 values each frame through a guaranteed pipeline is inefficient and unnecessary. Sending all 9 values each frame through an unguaranteed pipeline is no better.

Instead, extrapolate the position:

$f(x) = p_x + v_x t$

Extrapolation reduces the amount and size of messages that need to be sent, and when implemented well will be virtually identical to Realtime updates.

Execute Extrapolation

Using extrapolation, an example process is:

1. Send the three data sets in a guaranteed message at grenade's launch.
2. While in flight the remote clients extrapolate the position.
3. The local client sends unguaranteed position updates each frame. The linear and angular velocities are not sent.
4. The local client sends a guaranteed message only when the linear and/or angular velocities change.

Unguaranteed messages may arrive out of order, that is message 2 may arrive before message 1. Use a timestamp when receiving unguaranteed (and out of order) so that the system does not process messages that no longer matter. See `RtGame.GetSetInstanceTime()`.

Effects of Extrapolation

- Smaller messages
- Fewer messages
- Visually identical (or close enough) to Realtime updates.

Extrapolating and Collision

Problems With Collision

The players, local and remote, should all see the same events: the grenade launching from the local player to wherever they shot it. In a collision-free environment, there will likely be only small, indiscernible discrepancies.
With collision though, there can be glaring discrepancies. In the local player's view, the grenade flies within a few centimeters of a wall, but never hits it. In a remote player's view, the client's interpolation has the grenade a few centimeters offset, causing it to hit the wall. On that remote client the grenade will follow a completely different path as it changes direction from bouncing off the wall.

Solution

Permit only the local player to handle collision.
Make all remote clients ignore collision, instead relying on the unguaranteed messages to give the correct position updates. Only send updates when there are changes.
In the grenade example, the position is constantly changing, but linear velocity and angular velocity do not. If the grenade hits a wall it will have changed linear and angular velocities; when these changes occur send a guaranteed message.

Guarantee Critical Events

The grenade's ascent and descent are entirely cosmetic in terms of gameplay. The users will want to see where the grenade is going in order to avoid it, but watching where the grenade is going will have no effect on the game itself. With the grenade's flight, unguaranteed messages that arrive late can be disregarded; a few collisions can often be missed by the remote clients without consequence.
Where the grenade explodes does matter.

The grenade's explosion needs to be the same on all clients, because it affects the health and life of the players. You cannot have one client thinking that player A was killed by a grenade, while another thinks player A dodged it and is still going. Game changing messages, like the explosion of a grenade, need to be guaranteed.

For information on how to create a guaranteed message, see Realtime Network XML.

Realtime Design Tips

Guidelines

- Message Arrival: Never rely on messages arriving quickly. Essential systems must all effectively be turn-based, even if the result is 'realtime'.
- Group Door: When users are in a Realtimeall other scripts, screens, audio, etc are still running in that scene. Consider using the Group Door functionality to instance players to a less busy place.
- Sticky/Lock-on Targeting: As well as hiding the inevitable discrepancies in player position, a 'sticky' targeting system can help compress the data that needs to be sent. A message describing the attack can contain just the ID of the target, rather than the vector of the aim.
- Splash damage: Range-effect weapons can also hide the discrepancies between remote views. When a remote player appears to the user to be on the edge of an explosion but they don't receive damage, they're likely to accept that as part of the nature of online play, and will try to ensure your next shot hits them dead on.
- Guided weapons: Designing around guided weapons can lead to message-size savings and improve consistency of the shared view.
- Perfect Replication: Do not spend too much time comparing two screens side by side. You will want to make sure things are replicating reasonably correctly across clients, but there is a danger in putting too much emphasis on side-by-side comparisons. 99.9% of users will never see the remote players' screens while they're playing. The key factor is not that everything replicates absolutely correctly, just that it feels like it is.
- Data Compression: Reduce the amount of data sent as much as you can. For irregular messages individual packet size is less critical, but for those sent frequently every byte can make a difference. Consider the best data to squeeze into your updates. Depending on the control mode, some things will work better than others.

For a list of external resources that provide additional design guidance, see External Guidance on Realtime and P2P Design.

Position Replication

- Movement speeds: A control system with heavy inertia and/or slow turn speeds can replicate 'better' by making more use of dead reckoning and position interpolation.
- Global timestamp: Use a 'global' timestamp (acquired using the `GetInstanceTime()` or `RtGame.GetSentInstanceTime()` functions) in the position update message or replicated structure. This can be used to predict the remote user's position now, not just when the update occurred.
- Interpolate, Extrapolate and Blend: Consider keeping track of the player's last updated position (and timestamp), their current position in the local view, and their estimated position now. Blend between the positions to smooth the remote representation.
- Predict Velocity: For control systems with much inertia, replicating (and predicting) current velocity will make a huge difference.

- Replicate Controls: For some control types, replicate the current state of main controls.

For example, in developing a vehicle driving game, most of the time the user would be holding left or right (and often the 'accelerate' button) for long periods of time. The replicated view improves when the state of those main controls using a small bit-field are replicated. The remote player was updated locally by running through the standard control/physics update loop, and when the bit-field indicated the user was recently pressing 'left', the remote player representation was updated assuming the user continues to hold left. Though that can result in the remote view 'over-steering' that can be hidden by standard position, orientation blending and other tweaks to the system.

Position Replication with a Home Avatar

If using the Home avatar in your game, there are 3 main techniques to consider for handling the player position replication:

- Do nothing: Let the standard system update automatically. Infrequent updates, ~2 seconds delay, but no extra work for you
- Lock Players: Lock players and use the Lua Person library interpolation functions. This will automatically make the avatar walk/run at the desired speed.
- Lock Players and Custom Code: This is the most complex implementation. Lock the players and use custom control mechanisms and animations (for example, strafing).

Network Component

The Realtime game and players in the Realtime game are cut off from other users in the same scene. Other users in a scene cannot watch what is happening in a Realtime game, and Realtime game players cannot watch what is happening in the scene outside the Realtime game.

However, you can still use the Network Component to send Outbound and Welcome Messages to all the other users in the scene, which enables some communication with users in the scene not playing the Realtime game. For example, in a scene that contains a First Person Shooter Realtime game, you could have a scoreboard in the general area of the scene. Users not in the Realtime game can see the current score of the Realtime game, which is updated using Outbound Messages.

These messages are not sent down any of the Realtime game's peer-to-peer pipelines.

For more information about messaging, see Adding Network Communication and Mini-games.

For more information on other important network issues, including the need to block NAT type 3 users from using dynamic Realtime games, see Realtime Game Network Guidelines.

External Resources for Realtime and P2P Design

The following link leads to a good series of articles on the essentials of network programming:

http://gafferongames.com/networking-for-game-programmers/

The following link leads to a paper describing some general techniques for dead-reckoning and lag compensation:

http://web.cs.wpi.edu/~claypool/courses/4513-B03/papers/games/bernier.pdf

'Subspace' is an old game now, but it is still a great example of the execution of an online game given similar network limitations. It nicely captures the balance of design, production and technical ethic behind a title that genuinely works online:

http://snarfed.org/subspace

The following website looks mainly at the login and installation processes of World of Warcraft. It is not directly relevant to Home content developers, but it does contain some interesting notes such as:"Even today, the minimum network requirements for the original World of Warcraft client remain a 56 Kbps dialup link":

http://www.networkuptime.com/wow/page01-02.html

Companion Objects

> For Japanese Live Page
> 最新の日本語版ページはこちら 「コンパニオンオブジェクト」

A Companion object is a portable item that follows the user, performs idle animations when not moving, and can have a name.

The HDK provides a template in the Object Editor for creating Companion objects. You can use the template to develop sophisticated and desirable 'Companion' inventory items, without having to manage network replication, efficiency, or other scripting challenges. The template enables you to design ground-based or flying animals or creatures. The image here shows a green lizard companion object.

The template enables you to add the following sophisticated features to a Companion object:

- Particle effects
- Sound effects
- Flying behaviors
- Finer control of animations
- Customization through the configuration file

You can use the template to introduce a variety of states, behaviors and actions for the companion object: for example, flying or perching. Each state has settings that you can adjust, to tailor the companion object to your needs, and to specify which animations should be used. You can also customize companion objects through the Object configuration file.

To use the template, select the Companion Object option in the Object Editor's Object Creation Wizard, then provide the appropriate resources (model, skeleton and animations). Your new Companion object behaves like the default creature in the template, but by changing the models and animations you can create an entirely different creature. You cannot change the underlying scripts.

All models and animations must be compliant with PS Home Technical Requirements Checklists (TRCs), must validate and package correctly, fit into memory, and pass Quality Assurance (QA). See Content Requirements.

> The companion_follower template has been replaced by a more comprehensive template, companion_follower_2. If you have created companion objects using the companion_follower template, you might need to update and repackage your companion objects. See Recreating Companion Objects.
>
> The Maya exporter provides some additional compression tools to help achieve animation compression. For more information about this export pipeline, see Exporting a Simple Animated Object.



Creating Companion Objects

For Japanese Live Page
最新の日本語版ページはこちら 「コンパニオンオブジェクトの作成」

You create a Companion object and specify resources, properties, and idle animations for it, using the Object Editor.

Read this section in conjunction with Profiling Companion Objects and Companion Object Configuration File.

Before You Start

Before you create a companion object, you must first create and export from the DCC the following files:

- 1 x local model file (.mdl)
- 1 x remote model file (.mdl) A lower poly version of the local model (this is how remote players will see the companion)
- 1 x skeleton file (.skn)
- 1 x walk or run cycle animation (.ani)
- 1 x idle animations (.ani)

You can also export further animations if you want flying animations or more idle animations etc.

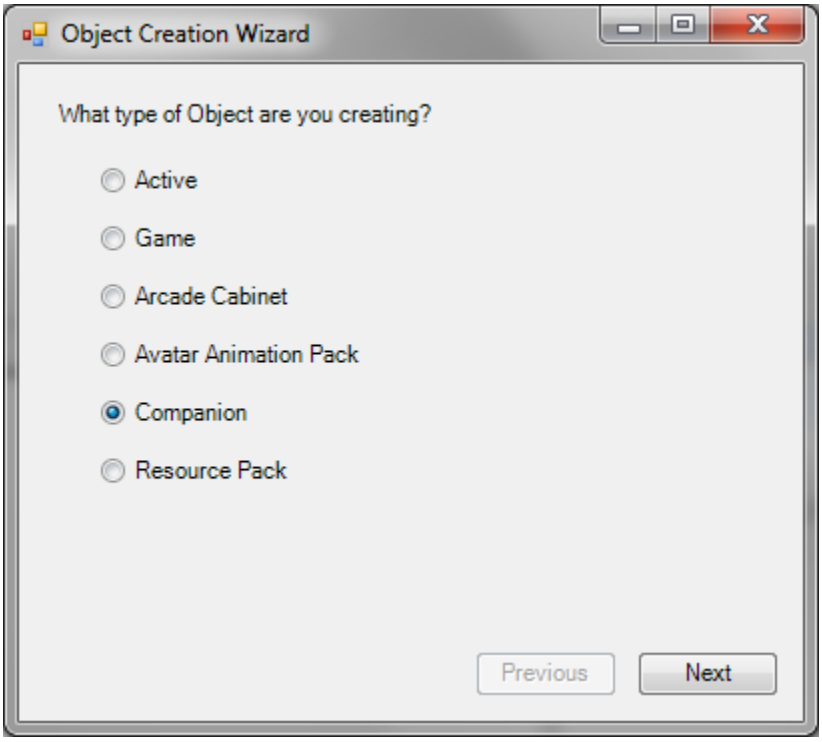
In addition you can also create particle and sound effects for your companion.

Creating a Companion Object

When you have created the companion using the wizard and prepared your assets, you need to edit the configuration file to complete the object definition. The configuration file resides in your companion directory. For details on how to update the configuration file, see Companion Object Configuration File.

To create a companion object:

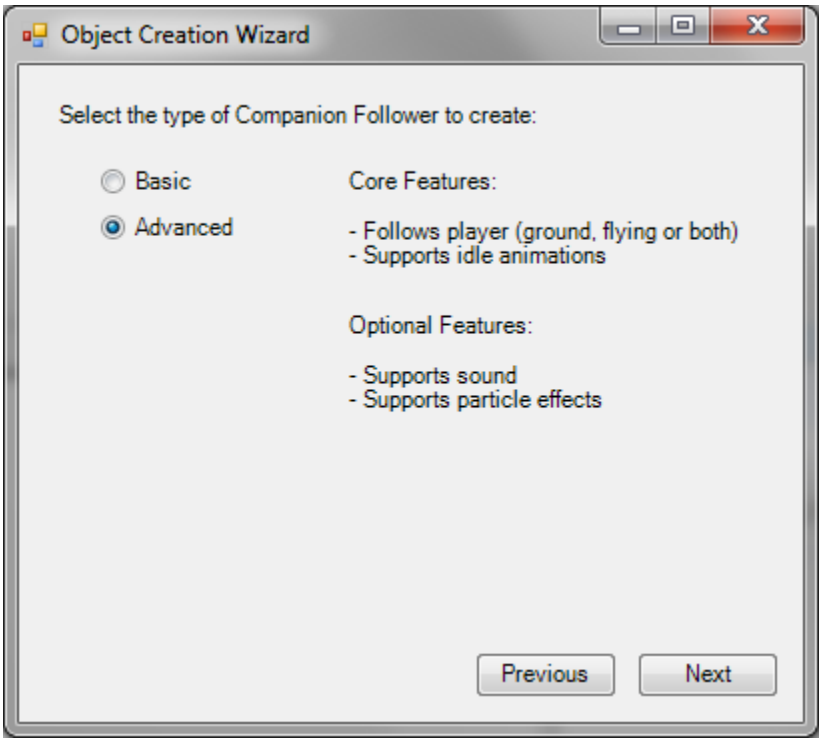
1. In the Object Editor, select File > New Object Wizard.
2. In the Object Creation Wizard dialog, select Companion and click Next:



3. Select Follower and click Next:



4. Select either Basic or Advanced, depending upon the type of companion you want to create. This option affects the default configuration file and sample resources that the companion object will contain.

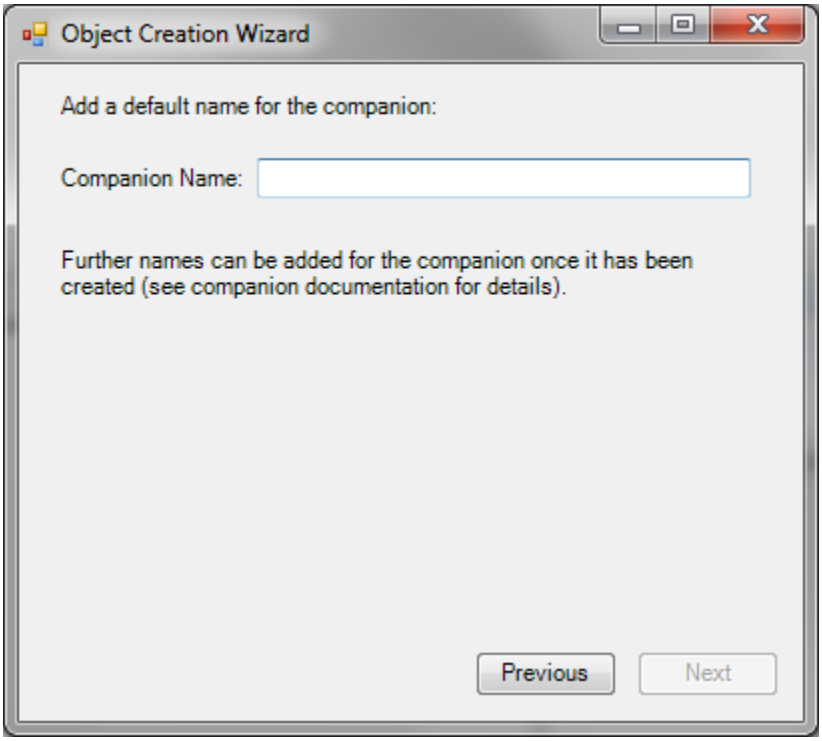


The Basic companion is ground-based. It can follow the player along the ground and play idle animations when the player is not moving.

The Advanced companion can be either ground or air based (or a combination of both). It can follow the player along the ground and by flying after the player. It can also play idle animations when the player is standing still. You can also add sound and particle effects.

You can convert a Companion object from a basic to an advanced companion and vice versa, by manually editing the object configuration file and adding or removing resources. See Companion Object Configuration File.

5. Enter the default name to give the companion. This name will be displayed above the companion when it is active.



This default name is only set for your current language. If you are releasing the companion to multiple regions with different languages you must add localized names for these regions. See Adding Companion Names.

You can specify additional names for the companion when you have created it. See Adding Companion Names.

6. Enter the name of the companion object, and provide a description.

You must provide a description. This description is visible to users when they select the Companion object. For character limits, in bytes, see the HDK Tools Validations.

A new companion template object is created in <HDK_INSTALL_DIR>/build/CompanionFollowers/<Companion Object name>/. This folder initially contains all the default template assets that are required for the companion object to work (i.e. the model, skin and animation files listed previously). You must replace these files with your own. The folder also includes all the other assets required for the companion object to work in PS Home.

7. Preview what the companion object is supposed to be and how it behaves.

   - To launch into the HomeDeveloper.self, see Launching the Home Developer SELF.
   - To add items to your inventory in the Home .self, see The DEV DEBUG Menu.

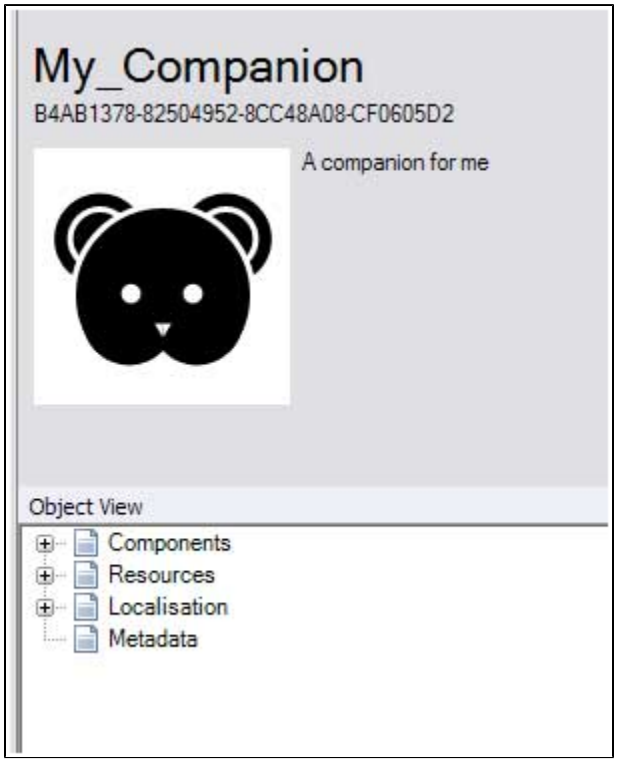
8. Move your Companion object assets (the model, skeleton, and animation files) to the <HDK_INSTALL_DIR>/build/CompanionFollowers/<Companion Object name>/ folder.

   The assets must be in the <HDK_INSTALL_DIR> directory for packaging. It is also easier to manage the files if they are in the same folder as your object.

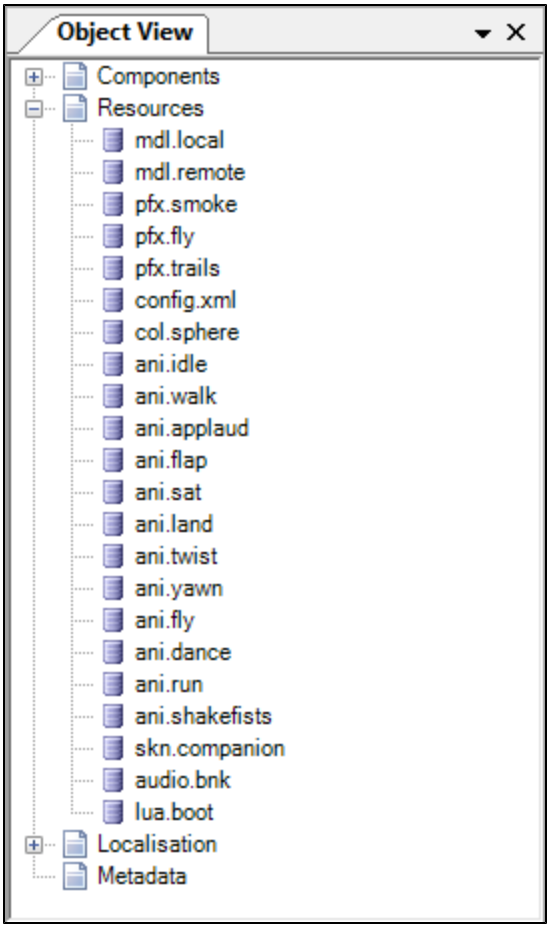
The next step is to edit the object and replace the default assets with your bespoke model, skeleton and animations.

Editing the Object

You start in the Object View panel:



1. In the Object View panel, select Resources:



2. Replace the asset in each of the following resources:

Resource Type	Associated File Type
mdl.local	.mdl
mdl.remote	.mdl
ani.walk / ani.run (you may only require one movement animation depending upon how you want your companion to appear)	.ani
ani.idle	.ani
skn.companion	.skn

> The Properties panel for lua.boot and col.sphere are grayed out. You cannot edit these resources in the Object Editor.

Note that the above files are the only core resources (apart from the config.xml) that your companion must have. All other resources can potentially be removed or replaced depending upon the behaviour that you want your companion to display. (However the config.xml will need to be updated to no longer reference any removed resources, see Companion Object Configuration File).

Below are details of some of the additional animations that you may wish to remove or delete. (Some of these resources are not included for the basic companion).

Animations used in the companion's default flying behaviour:

Resource Type	Associated File Type
ani.flap	.ani
ani.sat	.ani

ani.land	.ani
ani.twist	.ani
ani.fly	.ani

Particle effects:

Resource Type	Associated File Type
pfx.smoke	.efx
pfx.fly	.efx
pfx.trails	.efx

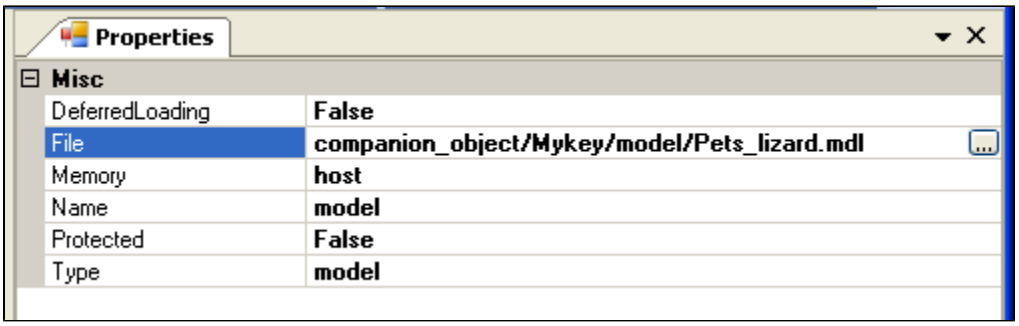
Sound effects:

Resource Type	Associated File Type
audio.bnk	.bnk

Additional behaviour animations:

Resource Type	Associated File Type
ani.applaud	.ani
ani.yawn	.ani
ani.dance	.ani
ani.shakefists	.ani

3. To replace an asset, select the resource type (for example, model), and in the Properties panel, click  next to File:



4. In the dialog displayed, select the appropriate file (for example, .mdl for model).
5. If required, change the name of the resource in the Name field. The Name property for the animation file is referenced in the config.xml file. Give animations a name that reflects what they do, such as 'walk' for the walking animation.
6. Repeat the replacement process for each of the resources that you need to replace.
7. When you have finished, remove the default assets from the Companion object's folder.

Adding Additional Resources

A Companion object can have any number of resources, limited only by the amount which can fit within the memory limits.

To add more resources such as animations and particle effects:

1. With the companion object selected, click Add New Local Resource in the toolbar.
2. Select the desired resource file.
3. If required, change the name of the resource in the Name field in the Properties panel.

For more information about limits and restrictions when creating companion objects, see Profiling Companion Objects.

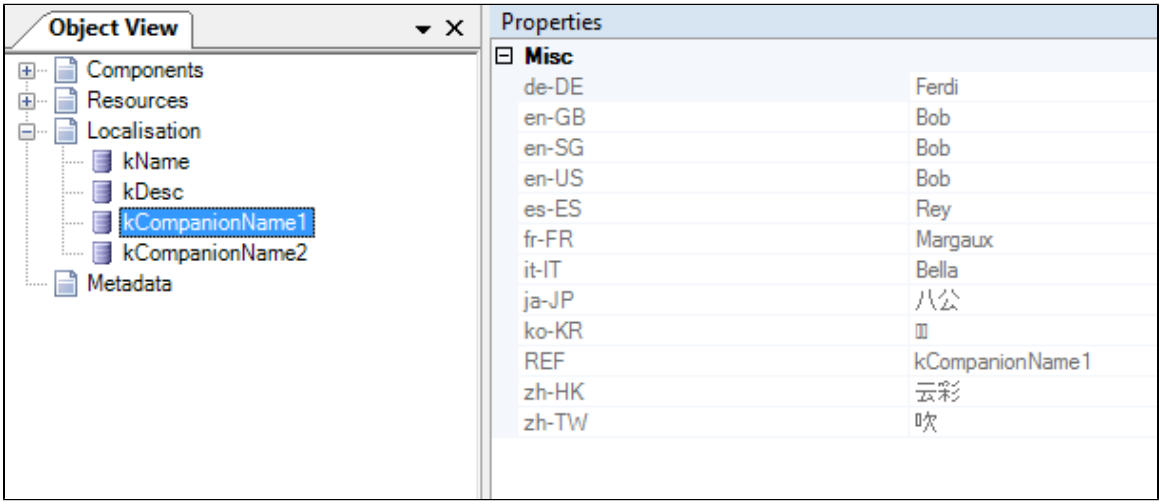
Adding Companion Names

For Japanese Live Page
最新の日本語版ページはこちら 「コンパニオンオブジェクトの名前の追加」

You can add Companion names through the localization system, in a similar way to how object name and descriptions are set-up. See Localizing Objects.

All localized items for companion names must follow the naming scheme kCompanionName1, kCompanionName2, etc.

Each companion can have up to 32 names. One of these names is selected when the companion is first activated.



Recreating Companion Objects

For Japanese Live Page
最新の日本語版ページはこちら 「コンパニオンオブジェクトの再作成」

The companion_follower_2 template enables you to create companion objects with enough local memory for object owners to add sound effects and particle effects.

The new template also provides:

- Several flying behaviors, including height variation and more freeform pathing
- A y offset for the companion label

The companion_follower_2 template replaces the companion_follower template, which catered only for ground-based companion objects. You can no longer create companion objects using the companion_follower template. However, this template is still included in the HDK so that you can modify and package any of your companion objects that were created from it.

If you need to recreate your companion object, you must use the New Object Wizard to create a new object, then re-import your assets (models and animation files).

If you are advised to update your template for the companion object:

1. Download and install the latest version of the HDK.
2. Select the companion object. It should update automatically.
3. Save and package the companion object.
4. Resubmit the companion object to the Content Delivery System (CDS).

Any additional steps will be communicated through a bulletin and documentation updates.

Companion Object Configuration File

For Japanese Live Page
最新の日本語版ページはこちら 「コンパニオンオブジェクト設定ファイル」

You create a companion object in the Object Editor. When you have created the companion and prepared your assets, you complete the object definition in the configuration file, <config.xml>. The configuration file resides in your companion directory.

You can set the following parameters for the Companion Object in the configuration file.

- Local and remote models: You can specify different model files for local and remote companions
- Particle effects: You can define a single effect or a group of effects
- Ground element: To define different states and behaviors for ground-based companions
- Air element: To define different states and behaviors for air-based companions
- Companion animations: To create “remixes” of your current animation
- Proportion attributes: To specify the frequency with which a companion adopts different animation options
- Twist layer: To make animations look more natural by applying a twist layer to it
- Actions and behaviors: To define actions and behaviors for each state

To configure your companion object, open the object's <config.xml> file in a text editor.

Please note, in the <config.xml> :

- time values are specified in seconds
- distance values are specified in meters

To look at sample configurations of varying complexity, see Example Companion Configurations.

Local and Remote Models

For Japanese Live Page
最新の日本語版ページはこちら 「ローカルモデルとリモートモデル」

The main asset related to companions is the mesh. You can specify different model files for local and remote companions. A detailed mesh can consume a large portion of the memory available to remote companions, which limits the animations they can load. If you create a lower level of detail mesh for the remote companion, a developer can save precious bytes.

To define the models, you must specify these two tags: <local_model> and <remote_model>. The model attributes you specify are the same for both tags:

Field	Required?	Description
name	Yes	Name of the model asset to load
scale	No	The model scale. The default is 1.0
label_offset	No	Height, in metres, of the companion's name label, measured from its base. Default is 0.5m

Add the model to the object resource list and mark it defer-load. It is valid for the remote and local model to be exactly the same.

Examples

The following example shows separate models for the local and remote companion, where the local model is made twice as big as normal:

```
<remote_model name="remote.mdl"/>
<local_model name="local.mdl" scale="2.0"/>
```

The following example shows the same model, but with an adjustment to the label offset for the larger local model, because the name label overlaps the scaled mesh:

```
<remote_model name="remote.mdl"/>
<local_model name="local.mdl" scale="2.0" label_offset="1.0"/>
```

See also:

- Allocating Memory for Companions

Defining Particle Effects for Companions

For Japanese Live Page
最新の日本語版ページはこちら 「コンパニオンオブジェクトのパーティクルエフェクトの設定」

You define particle effects before you define the ground and air states.

You can define two types of element for particle effects:

- effect - defines a single particle effect
- group - defines a collection of <effect> elements

You name the effects at this stage and reference them for animations in the configuration file later.

The <effect> Element

Add this element if you want to define a single effect. The following table describes how to define an effect.

Element	Required?	Description
name	Yes	The name by which you will reference this effect elsewhere in the configuration.
source	Yes	The name of the particle resource.
type	No	Specifies how the particle is attached to the entity's world matrix: <render> - The entity's world matrix is used as the render matrix of the particle effect. <emitter> - The entity's world matrix is used as the emitter matrix of the particle effect. The default is “emitter”.
attach	No	Attaches the emitter to a particular bone of your skeleton. By default, the emitter is attached to the entity origin.
pos	No	Three float values that represent the offset from the entity origin (or attach point), for example, “0, 0, -1” shifts the particle emitter 1 meter back along the z-axis
rot	No	Three float values that represent the emitter rotation (in degrees), for example, “0, 90, 0” rotates the emitter 90 degrees in the y-axis

The <group> Element

Use the <group> element if you want to define a collection of effects. For example, if you want to set up a glow particle on each hand of your companion, you create a group with two effects:

- one attached to the left hand
- one attached to the right hand

The <group> element must contain no more than four effect elements.

To use a <group> element, you need only define the group name:

Element	Required	Description
name	Yes	The name by which you will reference this effect elsewhere in the configuration

⚠ You do not define a name for the effect elements in the group.

The effects triggered are the child elements under the group (as shown in the following table):

Element	Required?	Description
source	Yes	The name of the particle resource.

type	No	Specifies how the particle is attached to the entity's world matrix: <render> - The entity's world matrix is used as the render matrix of the particle effect. <emitter> - The entity's world matrix is used as the emitter matrix of the particle effect. The default is "emitter".
attach	No	This element attaches the emitter to a particular bone of your skeleton. By default the emitter is attached to the entity origin.
pos	No	Three float values that represent the offset from the entity origin (or attach point), for example, "0, 0, -1" shifts the particle emitter 1 meter back along the z-axis
rot	No	Three float values that represent the emitter rotation (in degrees), for example, "0, 90, 0" rotates the emitter 90 degrees in the y-axis

Air Element

The <air> element sets the companion to fly, but you must also specify some other state elements to define how the companion behaves when flying and when idle. You can add the following states to the <air> element:

```
<air>
 <follow/>
 <perch/>
 <land/>
 <orbit>
  <circle/>
  <flit/>
 </orbit>
</air>
```

The following sections describe each state.

For sample code showing all behavior states, see Configuration 6 - All Behavior States

The <follow> State

As the user moves around a scene, the companion tries to follow. All companions fly to the user in exactly the same way, blending their aerial position towards the user, until they are close enough to enter an idle state (orbit). However, <follow> has some attributes that allow you to customize the companion movement: you can set an x and y oscillation.

Attribute	Required?	Description
oscillate_y_scale	No	Distance, in meters, the companion can move up and down in flight. Between 0 and 5.
oscillate_y_rate	No	The rate of y oscillation. The lower the rate, the slower the rate of oscillation. Between -200 and 200.
oscillate_x_scale	No	Distance, in meters, the companion can move left and right in flight. Between 0 and 5.
oscillate_x_rate	No	The rate of x oscillation. The lower the rate, the slower the rate of oscillation. Between -200 and 200.
twist	No	The name of an animation. See Specifying a Twist Layer for information on "twist".
twist_weight	No	The blending for the twist animation. The value for twist_weight should be between 0.0 and 1.0. See Specifying a Twist Layer for information on "twist_weight".
twist_left	No	The name of an animation. See Specifying a Twist Layer for information on "twist_left".
twist_right	No	The name of an animation. See Specifying a Twist Layer for information on "twist_right".
turn_factor	No	Controls how quickly the companion turns to face its desired direction. The value for turn_factor should be between 0.0 and 1.0. The default is 0.2. A value of 0 means the companion never turns to face its desired direction, a value of 1 equates to instant snapping in the direction.

The calculation used for the oscillation is:
f = sine(time * oscillation_rate) * oscillation_scale

You can also specify the following behaviors and actions:

Element	Required?	Description
behaviors	Yes	A list of animations for the companion to select from.
actions	No	A list of animations that can blend over the behavior animation.

The <orbit> Section

In the <orbit> section, you define the states (circle or flit) that the companion will assume when it is close to the user. You must define at least one state. If you define both states, the companion randomly selects which state to use each time it enters the orbit state. You must define these elements in the <orbit> section.

The <circle> Element

This element defines a behavior where the companion continually flies around the user. You can customize circling using the following attributes:

Attribute	Required?	Description
oscillate_y_scale	No	Distance, in meters, the companion can move up and down in flight.
oscillate_y_rate	No	The rate of y oscillation. The lower the rate, the slower the rate of oscillation. The oscillate attributes allow you to give the companion some bounce as it flies around the user.
twist	No	The name of an animation. See Specifying a Twist Layer for information on “twist”.
twist_weight	No	The blending for the twist animation. The value for twist_weight should be between 0.0 and 1.0. See Specifying a Twist Layer for information on “twist_weight”.
perch_min	No	Minimum time before the companion begins perch behavior.
perch_max	No	Maximum time before the companion begins perch behavior.
proportion	No	Weight given when selecting which orbit state to use. If the only state you have under orbit is circle, proportion does nothing. If orbit has more states, the companion uses the proportion value to calculate which orbit state to choose. The default value for proportion is 1.0, which means all states have equal chance to be selected. See Proportion Attributes for more information.
turn_factor	No	This attribute controls the speed with which the companion turns to face its desired direction. The value for turn_factor should be between 0.0 and 1.0. The default is 0.2.

The oscillate attributes allow you to give the companion some bounce as it flies around the user.

If the companion remains in the <circle> state for long enough and the configuration contains a <perch> state, the companion hovers and lands on the user's left or right shoulder. "Long enough" is defined by picking a random time between perch_min and perch_max.

If the only state you have under the <orbit> element is circle, then the proportion attribute has no effect. However, if you have more states under the <orbit> element, the companion uses the value of the proportion attribute to select the orbit state. By default the value of the proportion attribute is 1; this means that all the states have an equal chance of being selected. See Proportion Attributes for more information.

The <flit> Element

This element defines a behavior where the companion hovers at random points around the user. You can customize flitting using the following attributes:

Attribute	Required	Description
oscillate_y_scale	No	Distance, in meters, the companion can move up and down in flight. Between 0 and 5 (default is false).
oscillate_y_rate	No	The rate of y oscillation. The lower the rate, the slower the rate of oscillation. Between -200 and 200 (default is false).
points	No	The number of hover points the companion generates around the user. Between 1 and 32 points (default is 5).
time_min	No	The minimum time the companion spends at a hover point. Between 0.1 and 60 seconds (default is 1).
time_max	No	The maximum time the companion spends at a hover point. Between 0.1 and 60 seconds (default is 1).

height_min	No	The minimum height a point can be, measured from the user's feet. Between 0 and 2.2 metres (default is 1.1m).
height_max	No	The maximum height a point can be. Between 0 and 2.2 metres (default is 1.9m).
orbital_period	No	How long (in seconds) it takes to fly around the user. Between 0.1 and 60 seconds (default is 10).
perch_min	No	The minimum time before the companion begins perch behavior. Between 1 and 60 seconds (default is 10).
perch_max	No	The maximum time before the companion begins perch behavior. Between 1 and 60 seconds (default is 15).
radius_min	No	The minimum radius the point can be at (from user). Between 0.25 and 2 metres (default is 0.5m).
radius_max	No	The maximum radius the point can be at (from user). Between 0.25 and 2 metres (default is 0.5m).
face_player_proportion	No	The probability that the companion will face the user. A value between 0.1 and 10000 (default is false).
face_forward_proportion	No	The probability that the companion will face forward (matching user). . A value between 0.1 and 10000 (default is false).
turn_factor	No	This attribute controls the speed with which the companion turns to face its desired direction. The value for turn_factor should be between 0.0 and 1.0. The default is 0.2.

This state generates a fixed number of points around the user. The points are controlled by the radius_min/max and height_min/max attributes. The companion hovers at a point and stays there for a random time between time_min and time_max, and then finds a new point.

While waiting at a point, the companion bounces up and down, based on the oscillation values.

You can make the companion face the player or face forward at the hover points, or a mixture of both, by setting the face_player_proportion and face_forward_proportion attributes. When the companion stops at a point, it randomly picks one of the two choices, weighted by the proportions. For example:

- If you set both attributes to 1, they have an equal chance of being selected.
- If you set one attribute to 1 and the other to 2, the attribute set to 2 is selected twice as often as the attribute set to 1.
- If you specify only one of these attributes, only that attributed will be selected. The proportion value is irrelevant in this case.
- The scale of the proportion values is completely arbitrary. You can use 50 for face_player and 50 for face_forward to get a 50% chance of either. Or you could use 1000 for face_player and 1000 for face_forward to achieve the same thing. The parser will allow a number between 0.1 and 10000.

You can define the following elements for <flit>:

Element	Required?	Description
behaviors	Yes	Use to specify a list of animations to select from
actions	No	Use to specify a list of animations that can blend over the behavior animation
fly	No	Makes the companion fly between points, instead of hovering.

The <fly> Element

The <fly> element provides an alternative list of animations to use when the companion moves from point to point.

- Without the <fly> element, the companion always faces the user (or faces forwards) and navigates to a new point, using an animation from the behaviors list.
- With the <fly> element, the companion faces in the direction of flight and uses an animation from the <fly> list.

You can set the following attributes on the <fly> element:

Attribute	Required?	Description
threshold	No	The angle (in degrees) between points above which the companion will use <fly>. The companion hovers if moving small distances, but flies across larger ones.
turn_factor	No	This attribute controls the speed with which the companion turns to face its desired direction. The value for turn_factor should be between 0.0 and 1.0. The default is 0.2.

For example, if you set the <threshold> value to 45 degrees, and the companion is at x degrees:

- It flies if the new point is above x+45 or below x-45 degrees
- It hovers if the new point is between x+45 and x-45 degrees

The <land> State

The <land> state is optional. The state is required only if your companion defines both ground and air states. The <land> state is called when the companion is switching from air to ground. The state is a list of animations, one of which is picked as the companion descends.

The <perch> State

In the <perch> state, the companion hovers down and sits on the owner's left or right shoulder. You can customize perching using the following attributes:

Attribute	Required?	Description
sit_min	No	The minimum time the companion will remain sitting. A time between 0.1 and 60 seconds. The default value is 10 seconds.
sit_max	No	The maximum time the companion will remain sitting. A time between 0.1 and 60 seconds, but greater or equal to sit_max. The default value is 15 seconds.
turn_factor	No	This attribute controls the speed with which the companion turns to face its desired direction. The value for turn_factor should be between 0.0 and 1.0. The default is 0.2.
land_min	No	The minimum time the companion takes to land on a shoulder. Between 0.5 and 10 seconds, the default is 2.
land_max	No	The maximum time the companion takes to land on a shoulder. Between 0.5 and 10 seconds, the default is 4.

You can specify the following additional elements for the <perch> state:

Element	Required?	Description
behaviors	Yes	A list of animations to select from when the companion is sitting on a shoulder.
actions	No	A list of animations that can blend over the behavior animation.
landing	No	A list of animations to select from when the companion is hovering over a shoulder.

Ground Element

You use the <ground> element to specify a companion's ground-based movements. You must also specify some other state elements to define how the companion behaves when moving and when idle. You can add the following states to the <ground> element:

```
<ground>

 <ground_idle/>
 <ground_follow/>
 <ground_follow_slow/>
 <roam/>
 <launch/>

</ground>
```

> ⚠ The ground_idle and ground_follow states are mandatory.

For guidance on how to set up different ground-based companion configurations, see:

- Configuration 1 - Walk and Wait
- Configuration 2 - Hop and Look
- Configuration 3 - Roaming Behavior

The following sections describe each of the ground-based states.

The <ground_idle> State

The companion performs idle animations when not moving. You must specify which animations to use. For example, wait and look:

```
<ground>

     <ground_idle>
          <behaviours>
               <all name='wait.animation'/>
          </behaviours>
     <ground_idle>

</ground>
```

The <ground_idle> state occurs when the companion is close to the user. You can customize the <ground_idle> state using the following attributes:

Attribute	Required?	Description
min	No	The minimum time before the companion starts roaming
max	No	The maximum time before the companion starts roaming
turn_factor	No	Controls how quickly the companion turns to face its desired direction. The value for turn_factor should be between 0.0 and 1.0. The default is 0.2. A value of 0 means the companion never turns to face its desired direction, a value of 1 equates to instant snapping in the direction.

You can also specify the following behaviors and actions:

Element	Required?	Description
behaviors	Yes	A list of animations for the companion to select from.
actions	No	A list of animations that can blend over the behavior animation.

The <ground_follow> State

The <ground_follow> state occurs when the user moves away. As the user moves around a scene, the companion tries to follow. You must specify which animations to use. For example, walk and hop:

```
<ground>

     <ground_follow>
          <behaviours>
               <all name='walk.animation'/>
          </behaviours>
     </ground_follow>

</ground>
```

You can customize the <ground_follow> state using the following attributes:

Attribute	Required?	Description
turn_factor	No	Controls how quickly the companion turns to face its desired direction. The value for turn_factor should be between 0.0 and 1.0. The default is 0.2.
twist	No	The name of an animation.
twist_weight	No	The blending for the twist animation. The value for twist_weight should be between 0.0 and 1.0.
twist_left	No	The name of an animation.
twist_right	No	The name of an animation.

You can also specify the following behaviors and actions:

Element	Required?	Description
behaviors	Yes	A list of animations for the companion to select from.
actions	No	A list of animations that can blend over the behavior animation.

The <ground_follow_slow> State

The <ground_follow_slow> state is optional. If defined, the companion uses the state when following the user at a slow rate. You can define a value for the threshold attribute to determine the speed at which the companion goes into the <ground_follow_slow> state.

You can customize the <ground_follow_slow> state using the following attributes:

Attribute	Required?	Description
turn_factor	No	Controls how quickly the companion turns to face its desired direction. The value for turn_factor should be between 0.0 and 1.0. The default is 0.2.
twist	No	The name of an animation.
twist_weight	No	The blending for the twist animation. The value for twist_weight should be between 0.0 and 1.0.
twist_left	No	The name of an animation.
twist_right	No	The name of an animation.
threshold	No	The speed in meters per second. If the companion (not the player) runs faster than the value specified in threshold, it moves into the ground_follow state, otherwise it remains in the ground_follow_slow state.

You can also specify the following behaviors and actions:

Element	Required?	Description
behaviors	Yes	A list of animations for the companion to select from.
actions	No	A list of animations that can blend over the behavior animation.

The <roam> State

As well as having the ability to follow and stand idle, the ground companion can have a roaming behavior. The <roam> state is optional. After the companion has been in the <ground_idle> state for a specified time, it moves to the <roam> state. In this state, the companion walks around the user, pausing now and then. This mobility helps to bring the companion to life, rather than just standing idle in a single spot.

```
<ground>

     <roam>
          <random>
               <walk>
                    <all name='walk'/>
               </walk>
               <idle>
                    <all name='idle'/>
               </idle>
           </random>
     </roam>

</ground>
```

The <roam> state has the following elements:

Element	Required?	Description
random	Yes	Defines the <random> behavior type, which is the only type of roaming currently implemented.
sets	Yes	Defines the animations that are played when the companion is standing between points.

The following sections describe the elements in <roam>.

The <random> Element

This element is the only behavior implemented in <roam>. The <random> element allows the companion to walk to random points around the user, pausing for a configurable time, and then repeating the behavior.

You can customize the <random> element using the following attributes:

Attribute	Required?	Description
turn_factor	No	Controls how quickly the companion turns to face its desired direction. The value for turn_factor should be between 0.0 and 1.0. The default is 0.2.
radius_min	No	The minimum radius from the user within which the companion can roam. The minimum value is 0.25m, the maximum value is 2m, the default is 0.5m.
radius_max	No	The maximum radius from the user within which the companion can roam. The minimum value is 0.25m, the maximum value is 2m, the default is 1m.
speed	No	The speed (in meters per second) with which the companion moves from point to point.

The <sets> element

The <sets> element is a list of set elements. Whenever the companion stops whilst roaming it picks a set. Each set defines how long the companion stays idle for, and what animations are played. The set that the companion picks is based on weighted chance.

You can customize the <sets> element using the following attributes:

Attribute	Required?	Description
name	Yes	A unique name for the set.
min	No	The minimum time spent waiting in the set.
max	No	The maximum time spent waiting in the set.
proportion	No	The weight given when a set is selected. The default value is 0.

See the Proportion Attributes page for more details on how the proportion is used is selection.

You can also specify the following behaviors and actions:

Element	Required?	Description
behaviors	Yes	A list of behavior animations for the companion to play.
actions	No	The animations that can blend over the behavior animation.

The <launch> State

The <launch> state is optional. The state is the ground equivalent of the <land> state for the air element and is required only if your companion defines both ground and air states. The <launch> state is called when the companion is switching from ground to air. The state is a list of animations, one of which is picked as the companion ascends.

Companion Animations

For Japanese Live Page
最新の日本語版ページはこちら 「コンパニオンオブジェクトのアニメーション」

The <anims> list holds animation elements and <anim> defines a new animation. You use this list to create “remixes” of your current animation that run at different rates, have shorter durations, play sfx, and set commands to repeat or jump to other animations.

The following table summarizes how to define a new animation:

Element	Required?	Description
name	Yes	The name by which you will reference this effect elsewhere in the configuration.
source	Yes	The animation resource.

blendin	No	How long should the animation take to blend in. The default is 0.25s.
blendout	No	How long should the animation take to blend out. The default is 0.25s.
duration	No	Use this parameter to specify a new duration for the animation. If the value is longer than the actual resource length, it is automatically capped. A value of 0 is equivalent to the actual resource duration. You can use negative numbers to specify how much time to remove from the actual resource duration. For example '-1' runs the animation for 1 second less than the actual resource duration.
rate	No	Playback rate. The default is 1. The value can be negative for reversing animations.
sfx	No	The sound to play when this animation is triggered.
sfx_delay	No	If required, you can add a delay for triggering a sound effect. The default is 0s (no delay)
pfx	No	The particle effect to play. You can specify the name of an effect defined earlier in the Effects section OR the name of an effect defined earlier(in the Effects section).
pfx_delay	No	If required, you can add a delay for triggering the particle effect. The default is 0s (no delay)
animrate_factor	No	A factor controlling playback rate based on movement speed. The default is 1.0 This parameter applies only if you are using this animation for the walk behavior. It is ignored everywhere else.
animrate_min	No	When using <animrate_factor>, <animrate_min> sets the minimum animation rate that will be applied
animrate_max	No	When using <animrate_factor>, <animrate_max> sets the maximum animation rate that will be applied
repeat	No	The number of times the animation should repeat (this is ignored for behaviors). You must use a value between 1 and 100.
next	No	The animation (defined in <anims>) to jump to after the current animation ends.

You can set up a series of animations to play one animation after another; this enables you to chain small animations to build more complex animations with minimal memory overhead.
For example, you can have a juggler companion with the following short animations:

- Animation one: Takes out juggling balls
- Animation two: Juggles balls

You can then create animations in the list as follows:

```
<anims>
 <anim name='juggle1' source='take_out_balls.ani' next='juggle2'/>
 <anim name='juggle2' source='juggle.ani' repeat='8' next='juggle3'/>
 <anim name='juggle3' source='take_out_balls.ani' rate='-1.0'/>
</anims>
```

When you reference juggle1, the following animations start playing:

1. take_out_balls.ani
2. juggle.ani 9 times (played once and repeated 8 times)
3. take_out_balls.ani (in reverse because the rate is set to -1.0)

Defining Sound Effects for Companions

For Japanese Live Page
最新の日本語版ページはこちら 「コンパニオンオブジェクトのサウンドエフェクトの設定」

Prerequisites

Please note that sound effects will only play on the local companion.

The following prerequisites must be met to get sound effects playing for companions:

1. An audio bank must be added to the object's resources (via the Object Editor). This resource should be marked to defer load (and preferably in host memory)

2. The audio bank must be mentioned in the configuration:

```
<sound value="name_of_resource"/>
```

3. The sound to be played must be specified alongside an animation.

e.g.

```
<actions>
   <local name='anim1' proportion='2'/>
   <local name='anim2' proportion='1' sfx='explosion'/>
</actions>
```

Once the companion plays anim2 in the above example, it will also play the sound effect "explosion" from the audio bank.

Anims

You can specify the sound to play in three locations. The first is in the to define it in an anim in the anims list:

```
<anims>
  <anim name='flap' source='flap.anim' sfx='flapping'/>
</anims>
```

This is purely for convenience. It allows you to trigger the flapping sound whenever you specify the flap anim anywhere else in the configuration. You can also freely override this at an action or behaviour level. So if you had defined the above anim 'flap', you could do the following:

```
<!-- here you don't care about the sound - 'flapping' will play automatically because it's defined in
the anims element -->
<actions>
  <local name='flap'/>
</actions>

<!-- but here, you override the previous definition -->
<actions>
  <local name='flap' sfx='explosion'/>
</actions>
```

Behaviour or Action

The other two places you can specify sound is in the behaviour or action elements, in the individual animation elements:

```
<behaviours>
  <local name='idle' sfx='hover'/>
</behaviours>
<actions>
  <local name='punch' sfx='punch_sfx1'/>
</actions>
```

Putting the sfx in behaviour allows you to play a specific sound when the companion enters a state (for example "jets" when entering the following air state), whereas sfx in the actions elements will tend to synchronise with spot animations.

Proportion Attributes

> For Japanese Live Page
> 最新の日本語版ページはこちら 「Proportion 属性」

There are several areas in the configuration where you can specify a proportion attribute. Normally, you specify a proportion attribute whenever the companion can make a choice between several animation options, for example, when selecting an animation to use, or a state to enter.

The default value for all proportions is 1, so all options start off with the same weight. If you increase the proportion

attribute for one option, you increase the chance that it will be selected over the other options.

You calculate the probability of an option being chosen as follows:

- proportion value = X
- sum of all the proportion values for the available options = Y
- probability of being selected = X/Y * 100

For example, you have two options (A and B) set to the default proportion value of 1:

```
X = 1
Y = 1 + 1 = 2

Probability of selecting A or B = X/Y*100 = ½ *100 = 50%
```

In the next example, you have three options (A, B and C) with different proportion settings:

```
A has a proportion of 10
B has a proportion of 4
C has a proportion of 6

The probability of each option being selected is:

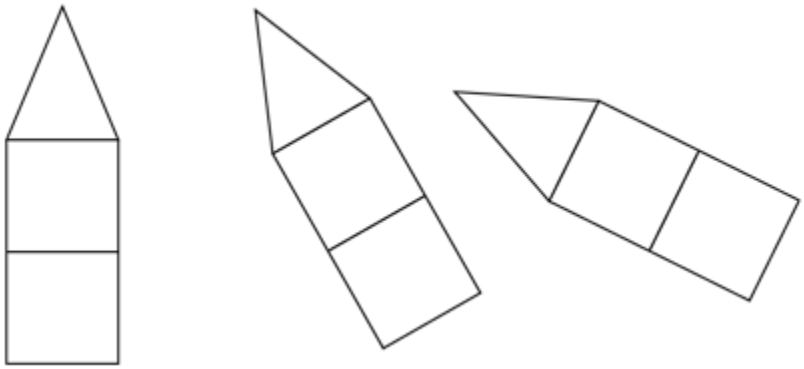
A 50%
B 20%
C 30%
```

Specifying a Twist Layer

For Japanese Live Page
最新の日本語版ページはこちら 「Twist レイヤの指定」

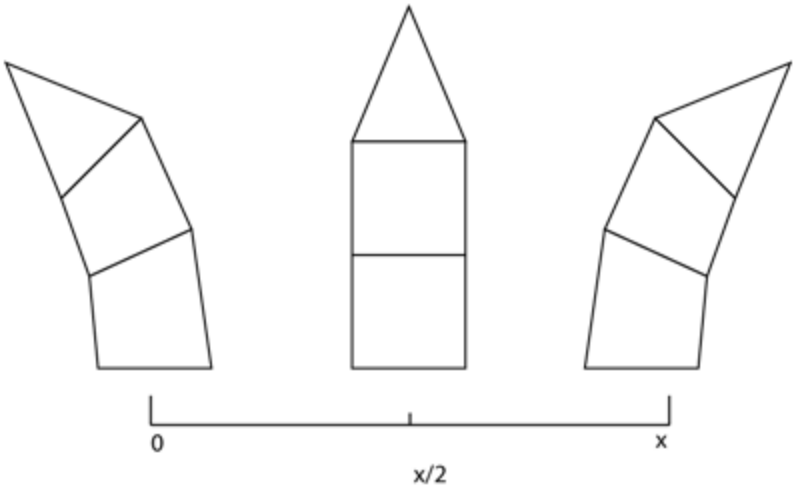
Some states allow you to specify a twist layer. This layer is an animation applied to the companion where the time position is based on how much the companion is turning.

In the following example, the companion is flying and starts to turn left. When moving, companion looks extremely stiff:



To make the companion look more natural, you can use a twist animation. A twist animation is an animation with 3 keyframes, for the companion to:

- swerve fully left
- go straight ahead
- swerve fully right



In this example, as the companion turns left, the time for the twist layer is set somewhere between 0 and x/2. As the companion turns right, the time is set to between x/2 and x. The overall effect is to make the companion seem less rigid and more organic.

You can also animate only the head (or any arbitrary bone) of the companion as it turns left or right.

Where twist is allowed, you can also specify a <twist_weight> value, which controls how much the twist layer is blended with the other layers of animation. You set this value between 0 and 1, where 0 means you do not see any effect, and 1 is the strongest effect.

Twisting Left and Right

You can use a left and right animation as an alternative to a single twist animation. This option requires more memory, however it can lead to better results.

If we use a single twist animation in a running cat, for example, we would have to control the cat's spine (extreme left, centre, extreme right). Because all the animations are multiplied in, the vertical movement on the cat's spine is reduced by any twist animation you apply. As a result, the cat looks like it is being held down.

Instead, you can create an animation with a cat running left and another animation with the cat running right, and then blend the two animations as necessary.

Actions and Behaviours

For Japanese Live Page
最新の日本語版ページはこちら 「アクションとビヘイビア」

In every state, you must specify a list of behavior animations that can be selected; the animation loops and plays until the next behavior is selected. You can also specify a list of actions that fire infrequently. Actions animate a portion of the companion or all of it, for example, waving a hand or shaking a leg.

The following example shows how the idle and idle_arms_crossed behaviors and the clap action are selected:

```
<some_state>
  <behaviours>
   <all name='idle' proportion='7'/>
   <local name='idle_arms_crossed' proportion='3'/>
  </behaviours>
  <actions min='5' max='15'>
   <local name='clap'/>
  </actions>
</some_state>
```

When the companion enters <some_state>, it selects either idle or idle_arms_crossed based on the proportion attributes. There is a 70% chance that the idle behavior will be selected. The animation plays until the companion changes state.

The companion also plays the clap animation, waiting between 5 to 15 seconds before clapping. The companion keeps playing the clap animation at a similar rate until the state changes.

The remote companion behaves differently; it always plays the idle animation when it enters <some_state>. It doesn't play the clap animation because clap is marked as local.

Changing Behaviors

If you want the companion to change behaviors when the state changes, you can specify a minimum and maximum time for the behavior list as shown in the following example:

```
<some_state>

  <behaviours min='15' max='25'>
   <all name='idle' proportion='7'/>
   <local name='idle_arms_crossed' proportion='3'/>
  </behaviours>
 </some_state>
```

In the example, the companion selects either idle or idle_arms_crossed sometime between every 15 and 25 seconds.

Example Companion Configurations

> For Japanese Live Page
> 最新の日本語版ページはこちら 「コンパニオンオブジェクトの設定例」

The following examples show how to configure the behavior of an original companion object, in several stages. The examples start with the simplest configurations for ground and air-based creatures, and adds more complexity as you progress through the examples.

The following configurations define a ground-based companion:

- Configuration 1 - Walk and Wait - Outlines the simplest configuration that you are allowed to create.
- Configuration 2 - Hop and Look - Extends the size of the behavior lists by adding two more animations.
- Configuration 3 - Roaming Behavior - Adds roaming behavior so that the companion can move around the user rather than staying in one place.

The following configurations define an air-based companion:

- Configuration 4 - Basic Air State - Provides the minimum information for basic air-only behavior.
- Configuration 5 - Multiple Air States - Adds <flit> and <perch> to the list of air behaviors.
- Configuration 6 - All Behavior States - Provides an example of a complex companion that uses all the behavior states.

Configuration 1 - Walk and Wait

> For Japanese Live Page
> 最新の日本語版ページはこちら 「設定1 - 歩行と待機」

The following sample code defines the simplest configuration that is allowed:

```
<xml>

1  <remote_model name='mdl.remote'/>

2  <local_model name='mdl.local'/>

3  <ground>

4   <ground_follow>
5    <behaviours>
6     <all name='walk.animation'/>
    </behaviours>
   </ground_follow>

7   <ground_idle>
8    <behaviours>
9     <all name='wait.animation'/>
    </behaviours>
   </ground_idle>

  </ground>

 </xml>
```

You build the configuration as follows:

(1) Define the remote model properties by naming the remote model resource. The model data is mandatory. It defines which models to load from the object. This definiton allows you to:

- Save memory in the remote companion by loading a less complex mesh; or
- Maximize the potential in the local companion by loading a more complex mesh

(2) Define the local model properties, in the same way.

> ⚠ The companion reports an error if (1) or (2) is missing.

(3) Define the ground state. The companion's ground behavior consists of following the user and waiting idle by the user. For both these states, you must specify which animations to use.

Steps (4-6) tell the companion to use the `walk.animation` when it follows the user:

- (4) Define the `<ground_follow>` state.
- (5) Add a behavior list to the `<ground_follow>` state. The behavior list must contain at least one entry. Each entry defines an animation.
- (6) Specify an element called `all`, which names the resource `walk.animation`.

Steps (7-9) define which animation the companion plays while standing idle.

This configuration creates a companion that follows the user and stops when it gets near to the user, but it is quite limited. The companion always plays the same animations and can become quite boring to the user. For more complex configurations and a description of different states and elements, see:

- Configuration 2 - Hop and Look
- Configuration 3 - Roaming Behavior
- Companion Object Configuration File

Configuration 2 - Hop and Look

> For Japanese Live Page
> 最新の日本語版ページはこちら 「設定2 - 跳ねる動作と見る動作」

This configuration builds on Configuration 1 - Walk and Wait by extending the size of the behavior list set up in that configuration. The following sample code adds two lines to the previous configuration, extending the size of the behavior lists:

- The ground_follow state has an additional behavior, `hop.animation`
- The ground_idle state has an additional behavior, `look.animation`

```
<xml>

  <remote_model name='mdl.remote'/>

  <local_model name='mdl.local'/>

  <ground>

   <ground_follow>
    <behaviours>
     <all name='walk.animation'/>
1     <all name='hop.animation'/>
    </behaviours>
   </ground_follow>

   <ground_idle>
    <behaviours>
     <all name='wait.animation'/>
2     <all name='look.animation'/>
    </behaviours>
   </ground_idle>

  </ground>

 </xml>
```

Every time a companion changes state, it starts an animation from its behavior list. If the list has more than one animation, the companion chooses an animation at random. In this example, when the companion starts following the user, it chooses either the “walk” or “hop” animation. When it changes to the idle state, it chooses from “wait” or “look”.

You can load animations on either the local companion, the remote companion or, as in this example, on both. You can use the extra memory on the local companion for additional animations or longer versions of existing ones.

Where an animation loads is based on the type of element applied to the behavior:

- local - only loaded for the local companion
- remote - only loaded for the remote companion
- all - loaded in both the local and the remote companion

In the following example, the local companion loads three animations: “wave”, “nod” and “idle”. The remote companion loads (and selects from) two animations: “idle”, “nod_quick”.

```
<behaviours>
 <local name='wave'/>
 <local name='nod'/>
 <all name='idle' />
 <remote name='nod_quick'>
</behaviours>
```

For an example of adding roaming behavior, see Configuration 3 - Roaming Behavior.
For a description of different states and elements, see Companion Object Configuration File.

Configuration 3 – Roaming Behavior

For Japanese Live Page
最新の日本語版ページはこちら 「設定3 - 歩き回る動作」

This configuration builds on Configuration 1 - Walk and Wait and Configuration 2 - Hop and Look.

As well as being able to follow and stand idle, the ground companion can have a roaming behavior. The following example shows a companion that is configured to move from point to point around the user.

```
<xml>
  ...

 <ground>

  <ground_follow> ... </ground_follow>

  <ground_idle> ... </ground_idle>

1  <roam>
2   <random>
3    <walk>
     <all name='walk'/>
    </walk>
4    <idle>
     <all name='idle'/>
    </idle>
   </random>
  </roam>

 </ground>
</xml>
```

You build the configuration as follows:

(1) Add the <roam> tag. This tag indicates that the following code will define some roaming behaviors.

(2) This step shows the single roam behavior, random. The random state is activated after the companion has been standing idle for a few seconds. In random mode, the companion chooses points around the user, walks there, waits a few seconds and then repeats these actions. This state mimics the general ground behavior available in the original companion. You must specify the animations for when the companion is walking and when it is idle.

(3) and (4) set the animations to walk and idle, respectively.

You can define several animations in both the walk and idle states, much like those listed in ground_follow and ground_idle in Configuration 2 - Hop and Look.

For a description of different states and elements, see Companion Object Configuration File.

Configuration 4 - Basic Air State

For Japanese Live Page
最新の日本語版ページはこちら 「設定4 - Airステートの基本」

You can create companions that are limited to aerial behavior. This configuration is similar to the basic ground-only file, described in Configuration 1 - Walk and Wait.

In this example, you provide the minimum information for basic air-only behaviour, which consists of the model names and the basic air states.

```
<xml>
  <remote_model name='mdl.remote'/>
  <local_model name='mdl.local'/>
1  <air>
2   <follow>
    <behaviours>
     <all name='fly'/>
    </behaviours>
   </follow>
3   <orbit>
4    <circle>
     <behaviours>
      <all name='flying'/>
     </behaviours>
    </circle>
   </orbit>

  </air>

  </xml>
```

You build the configuration as follows:

(1) The <air> tag defines the companion as an air-based, rather than a ground-based creature.

(2) You then define the states for following and being idle. Following is similar to ground_follow. You just provide the animation list for the companion object to select from. A fly companion flies at a fixed height behind the user and when close to the user enters the idle state.

(3) Define the idle state for air behaviour in the orbit section. Unlike the ground companion, air idle is defined by several states, which are all listed under <orbit>.

(4) This example uses the simplest air idle state, circle. In this state, the companion flies around the user's head.

For a more complex example of air behavior, see Configuration 5 - Multiple Air States.
For a description of different states and elements, see Companion Object Configuration File.

Configuration 5 - Multiple Air States

For Japanese Live Page
最新の日本語版ページはこちら 「設定5 - 多様なAirステート」

This example demonstrates nearly all the air states. The example in Configuration 4 - Basic Air State introduced the <circle> state in the <orbit> section. In this state, the companion flies around the user's head at a steady rate.

In the following example, you add <flit>, which is less predictable, and some other behaviors.

```
<xml>

  <remote_model name='mdl.remote'/>
  <local_model name='mdl.local'/>
  <air>
   <follow>
    <behaviours>
     <all name='fly'/>
    </behaviours>
   </follow>

1   <orbit>
    <circle>
     <behaviours>
      <all name='flying'/>
     </behaviours>
    </circle>
2    <flit>
     <behaviours>
      <all name='hover'/>
     </behaviours>
    </flit>
   </orbit>

3   <perch>
4    <landing>
     <all name='ani.land'/>
    </landing>

5    <behaviours>
     <all name='ani.sat'/>
    </behaviours>
   </perch>

  </air>

 </xml>
```

You build the configuration as follows:

(1) You define the idle state for air behaviour in the <orbit> section, and list all air idle states under this section.

(2) The <flit> behavior makes the companion hover at points around the user and then dart to some other point. it is like an air version of the <ground_roam> state.

(3) This example shows another air state definition, called <perch>. In the <perch> state, the companion hovers down and sit on either the user's left or right shoulder.

(4) For the <perch> state, you must define which animation is to be used when the companion is landing on the user's shoulder. You define the animation using the landing list.

(5) You must also provide a list of animations that will be used when the companion is actually seated on the shoulder. You define these animations using the <behaviours> tag.

For examples of ground-based animations, see:

- Configuration 1 - Walk and Wait
- Configuration 2 - Hop and Look
- Configuration 3 - Roaming Behavior

For an example of a complex companion that uses all the behavior states, see Configuration 6 - All Behavior States.
For a description of different states and elements, see Companion Object Configuration File.

Configuration 6 – All Behavior States

For Japanese Live Page
最新の日本語版ページはこちら 「設定6 - すべてのビヘイビアとステート」

The following sample code shows the most complex companion, one that uses all the behavior states. This is a simplified example, for illustration only. It would fail to load because it excludes much of the mandatory information.

```
<xml>
 <remote_model/>
 <local_model/>
 <sound/>
 <spawn/>

 <effects>
  <effect/>
  <group/>
 </effects>

 <actions>
  <anim/>
 </actions>

 <air>
  <follow/>
  <perch/>
  <land/>
  <orbit>
   <flit/>
   <circle/>
  </orbit>
 </air>

 <ground>
  <launch/>
  <ground_follow/>
  <ground_idle/>
  <roam>
   <random/>
  </roam>
 </ground>

</xml>
```

For a description of the different states and elements, see Companion Object Configuration File.

Allocating Memory for Companions

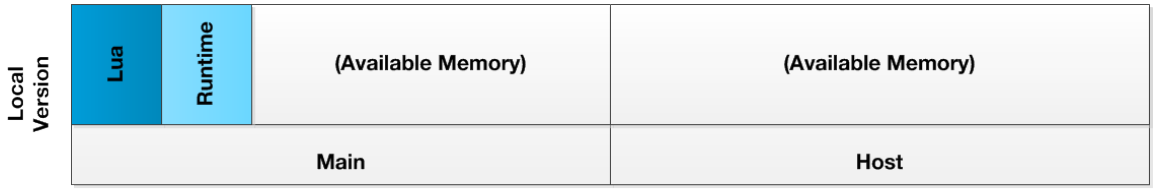
For Japanese Live Page
最新の日本語版ページはこちら 「コンパニオンオブジェクトのメモリ割り当て」

Memory Allocation

The Companion is an Inventory Item and as such has different memory budgets for the local player's object and for the version seen by remote users. The following table shows the memory allocations for local and remote inventory items:

Type	Host	Main	VRAM
Local	256 KB	256 KB	512 KB
Remote	128 KB	128 KB	256 KB

The companion loads and runs the Lua environment in the main memory pool, and a portion of the memory pool is used during runtime. The remaining memory can be used for your models, animations, sound bank and particles. The initial memory allocation is as follows:





Textures are always stored in VRAM, but you can store other assets in either Host or Main (this is set in the Object Editor on a per-object basis). A texture loaded into VRAM requires a reference to the location of that texture will be stored in Main (this can be moved to Host if needed). A texture reference uses a very small amount of memory, but if the companion contains a large number of textures, it can add up.

The complexity of the companion's configuration file will affect the memory footprint of the Lua and Runtime allocations because code features that aren't used will not be loaded into memory.

> You can use the Object Profiler to find out the available memory in each memory pool.

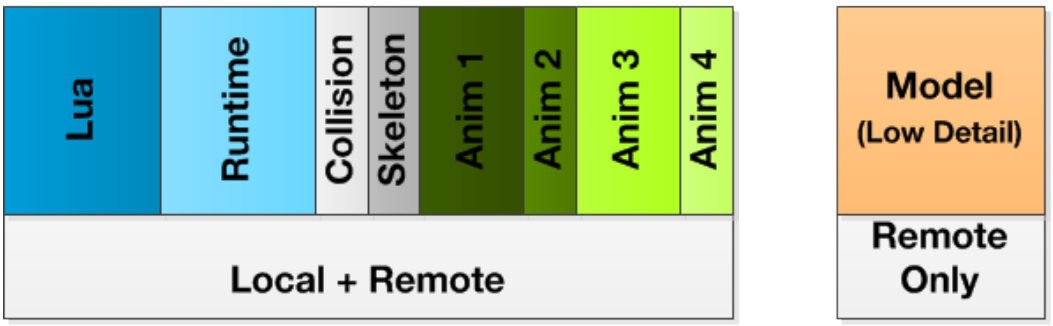
Allocating Memory for Remote Versions of Companion Objects

Remote versions of Companion Objects have half the memory budget of local versions across all pools (Host, Main and VRAM). The remote version of a companion usually consists of a subset of the functionality of the local version. You can also include a lower detail version of your model which is used in place of the usual model to save memory. A companion will therefore include resources that are used by both the local and remote version as well as resources that are used by one but not the other.

When adding resources to a companion, it is important that Deferred Loading is enabled for all resources except for the collision file and skeleton file. This prevents any resources that aren't needed from being loaded.

Example

A typical companion's resources can be grouped as follows:





There are two sets of animations, the ones marked in green play on both the local and remote companions, while the pink ones are loaded only by the local companion.

For every resource, we must decide:

- If the resource should be available for the local and remote companions.

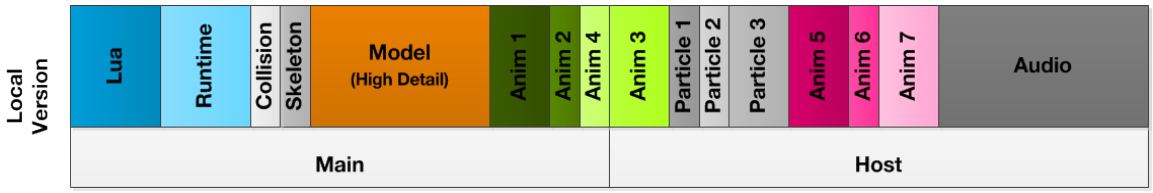
> This decision not apply to sound banks and particle effects because only the local companion can load these resources.

- If the resource should be allocated to the Main or Host memory pool.

  This decision is about prioritizing and fitting the resources into the available memory.

Space Available for Local Companion

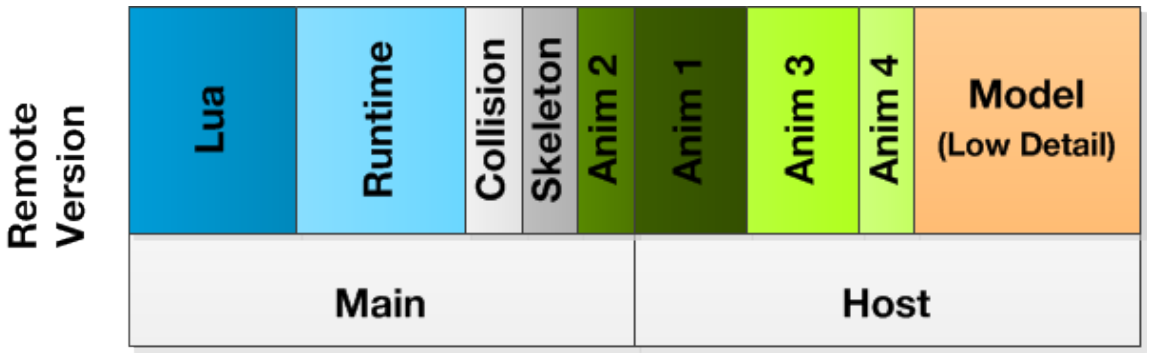
The following image illustrates the available space for the local companion and how the resources are allocated. The size of the box represents the amount of memory the resource requires.



Anim 3 was too large to fit in the remaining Main memory, so it was placed in Host.

Space Available for Remote Companion

The following image illustrates the available space for the remote companion and how the resources are allocated:



We place a single animation in the Main memory pool. The Host memory pool contains the low resolution model and the rest of the animations. No audio or particle effects are included.

Character Components

> These pages are about Character Components in general. For specific information on creating character components in Maya, see the Art Tools > Character Tools section, which includes:
>
> - Vertex Normal Tool
> - Weighting Tool
> - Replace Fat and Thin Blend Shapes
> - Character Component Reference

Character components are associated with characters, just as scene components (furniture, picture frames, etc.) are associated with scenes. Like characters, character components can move between different scenes. Character components can be equipped by the user to be worn by or adorn characters, and are available in the Wardrobe option of the inventory under the Clothing and Character sections. Character components occupy certain slots on the character rig.

The components need to fit seamlessly onto the character (i.e. the vertices of a component need to match those of the adjacent components) and integrate well with the movements the character is capable of (i.e. the geometry conforms to the skeleton in all potential poses).

Home Tools supports the production of character components by providing a shelf of tools and various templates within Maya, as

well as validation, preview and export functions. The files exported from Maya can then be packaged in the Object Editor and submitted for quality assurance.

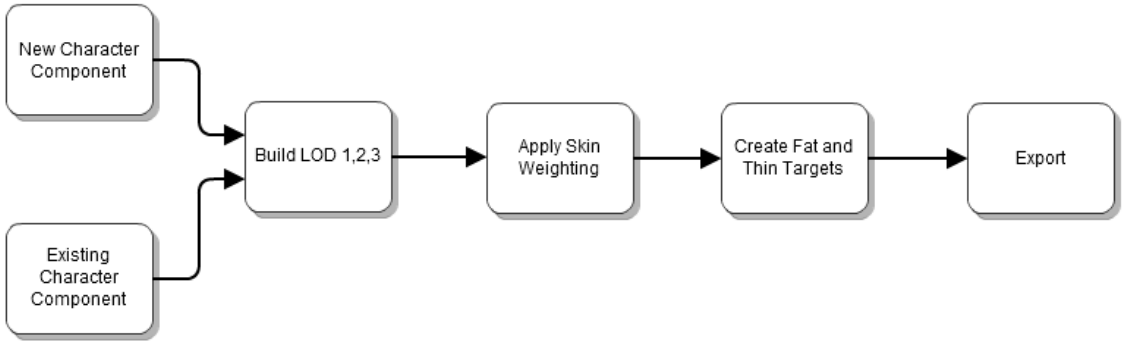
This documentation outlines the creation process and gives specific examples of particular components in more detail, such as torso and hat components. Learn the rules and conventions to help ensure the character components you create are valid, export successfully and pass Quality Assurance (QA).

See also:

- Character Component Creation Workflow
- Character Component Types
- Creating a Character Component
- Building LODs
- Applying Skinning and Weighting
- Creating Fat and Thin Targets
- Exporting a Character Component

Character Component Creation Workflow

This section provides an overview of the artist's workflow for creating PS Home characters in Maya. Although the steps are ordered sequentially, they do not necessarily reflect the exact order in which you do things.



Steps 3 and 4 - creating Fat/Thin targets and applying skinning - can be completed in any order.

You can apply skinning first or create fat/thin targets first, it doesn't matter. As long as you complete those tasks after you've built LODs and before you save/export.

	Task	Description
1.	Create a character component/ Edit an existing character component	Create a new character component or edit an existing character component
2.	Build LODs	Create the required Level of Detail (LOD) meshes
3.	Create Fat and Thin targets	Create the fat and thin blend shapes. All character clothing components in PS Home must have fat and thin versions of all the LODs
4.	Apply Skinning	Apply custom weighting which is likely to be necessary on all garments
5.	Save and/or export	Home Tools provides validation, preview and export functions

Character Component Types

A character component is an item of clothing or some form of accessory that decorates the avatar. Character components fit into designated slots.

You can create the following types of character component:

- A single atomic character component - an item of clothing that occupies only one of the avatar slots, such as a T-shirt that fits onto the Torso slot.
- A composite character component - two or more components fused together as one item (not interchangeable) that occupy

more than one slot, such as a long dress that fits onto and spans both the Torso and Legs slots.

Both atomic and composite character components must fit well – on the avatar itself and in combination with other components:

- Fit well on avatars of any blend shape (fat or thin) and fit with any avatar movement or pose (sitting, dancing), by conforming to the geometry of the skeleton.
- Work well with other components that can be equipped next to them on the avatar, by keeping to the slot or slots used by your particular character component, and by matching the vertices of the component to those of the adjacent components.

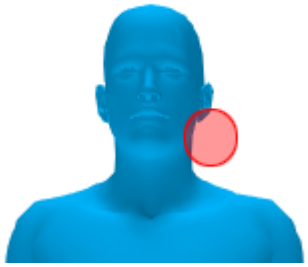
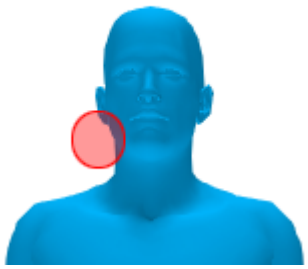
See:

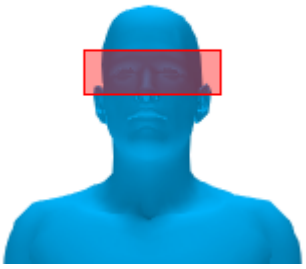
- Avatar Slots
- Composite Character Components
- Fitting Components Together
- Menu Categories for Clothing
- Creating Headgear Components

Avatar Slots

The following images shows an approximate indication of the individual slots on an avatar:

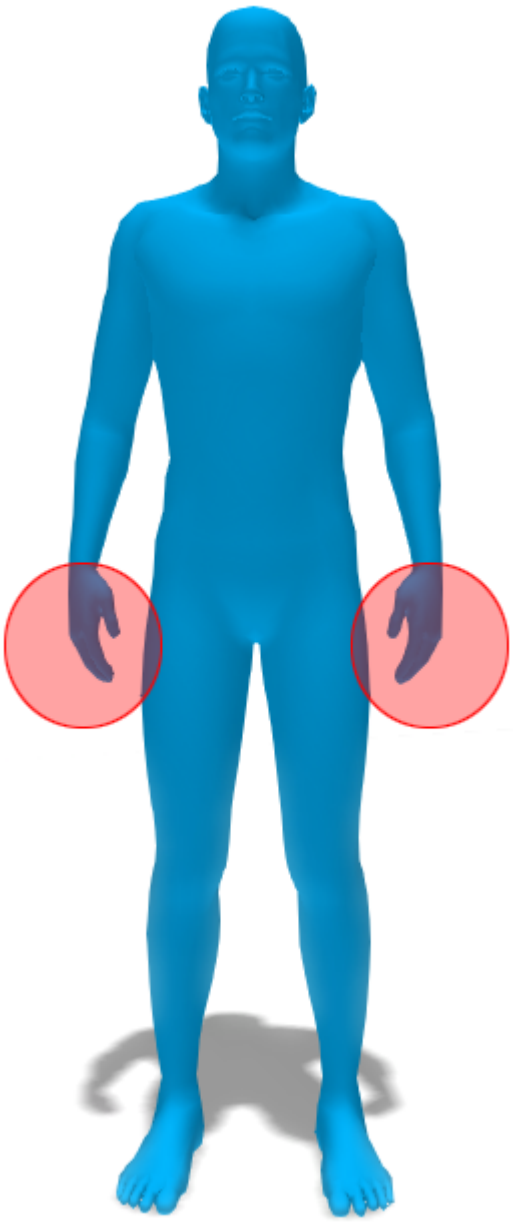
Head Slots

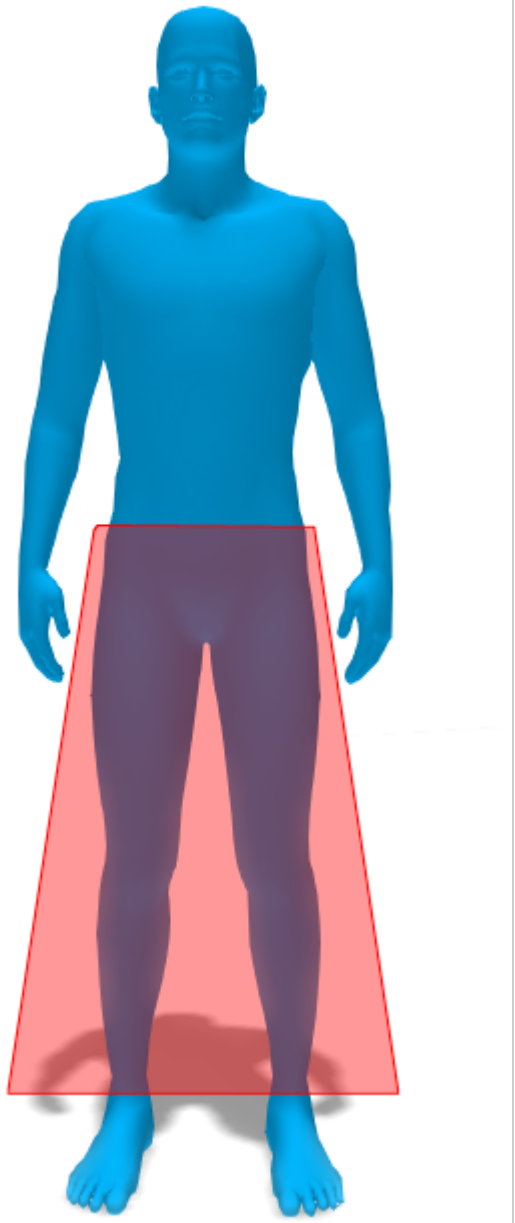
	Hair/Hat
	Jewelry (Left Ear)
	Jewelry (Right Ear)

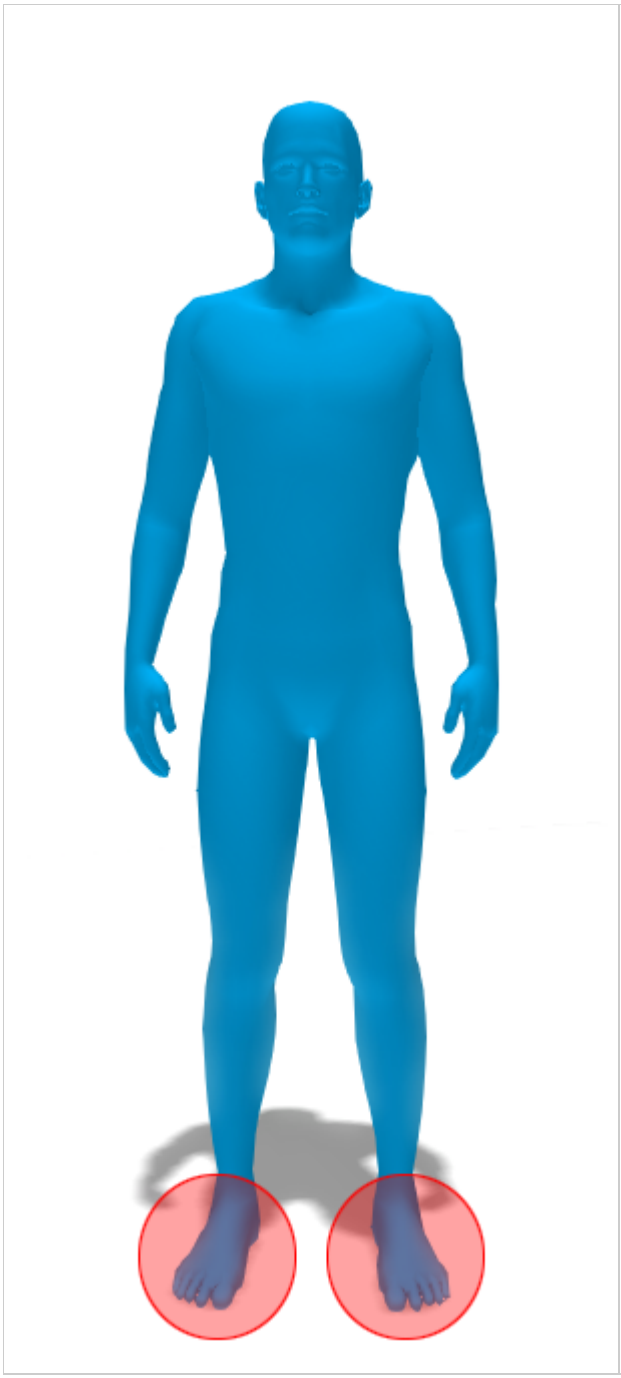
	Spectacles
	Headphones
	Facial Hair *
	Head **

Body Slots

	Torso

	Hands

	Legs



Feet

* The Facial Hair slot is only for male avatars.

** The Head slot is not normally available as a component, but is available in two situations: a FullHeadDress composite and a FullBodySuit composite. This is because both these composites cover the face entirely and free up the slot normally occupied by the avatar's head.

** A full headdress neckline must not deform with the fat and thin blendshapes.You must create a larger neck area to compensate and make sure that the neckline is covered for both the fat and thin shapes.

Composite Character Components

PS Home supports character components that occupy multiple slots.

Composite character components are always in adjoining slots, and there are a set number of supported combinations:

- Legs+Feet
- Torso+Legs
- Torso+Legs+Feet

- BodySuit
- FullBodySuit
- HeadDress
- FullHeadDress

You can combine different character component slots or adjoining areas by fusing two or more components together to make one new component. For example, you could combine the legs and feet components to make a legs+feet combination component that would allow you to make a pair of jeans tucked into some high boots. The legs+feet combination would then appear as one item in the wardrobe. You cannot wear just the boots, or just the jeans, you must wear both if you select them in the wardrobe.

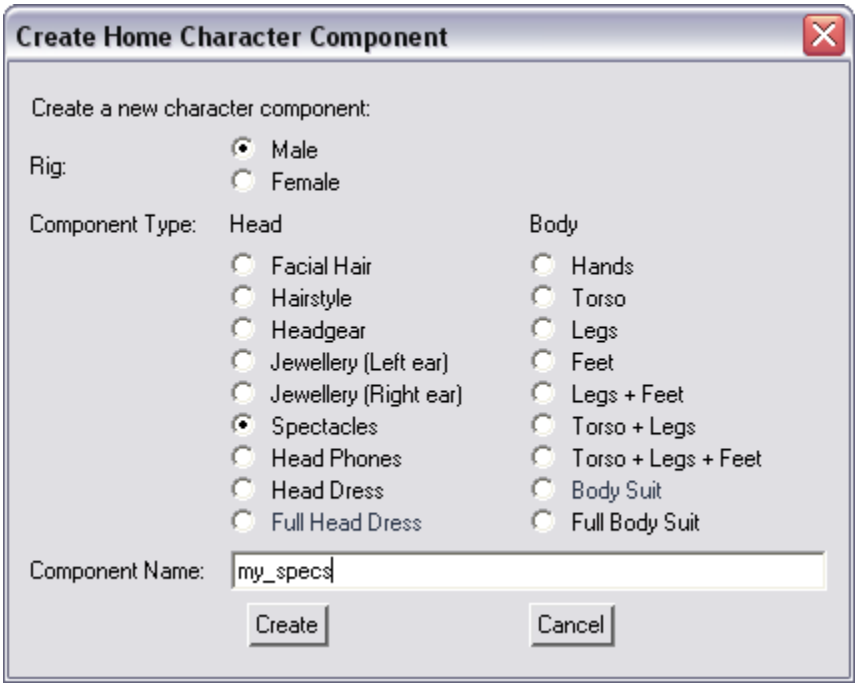
You can also combine all the components together and make a full body costume. The advantage of composites is that you can design the components to look like they overlap, because normally you cannot, for example, have feet components that go above into the legs area.

The HDK provides examples of composites. Go to the following directory and select any of the composite templates:

<HDK_INSTALL_DIR>/artsource/templates/

Composite Types

PS Home allows the preparation and export of composite character components. This allows you to combine the basic component types into single components to address seam joint and mesh intersecting problems experienced when modeling certain types of clothing design. The composites allow the following combinations:



Legs + Feet (LegsAndFeet)



Torso + Legs (TorsoAndLegs)



Torso + Legs + Feet (TorsoLegsAndFeet)



BodySuit

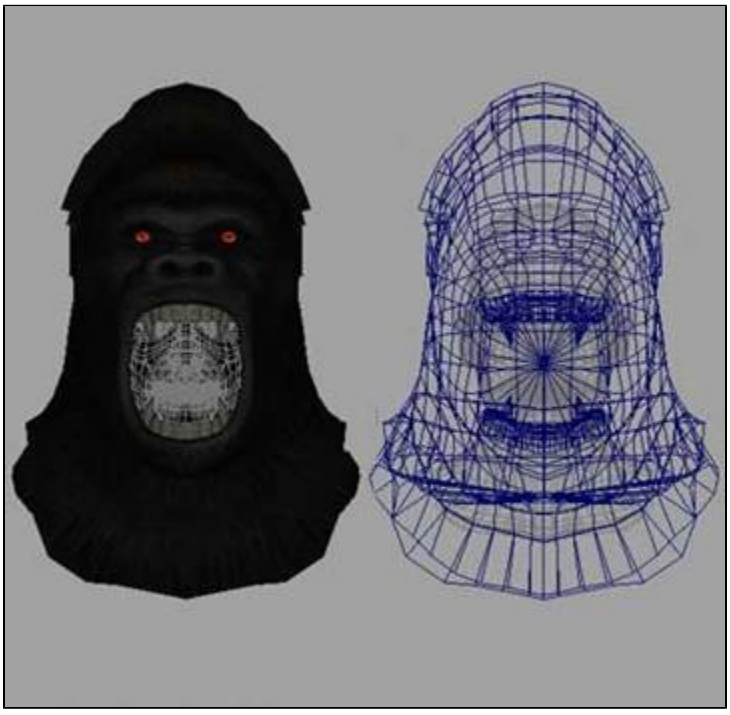


The BodySuit composite consists of Torso+Legs+Feet+Hands. This effectively replaces everything below the avatar's neck.

FullBodySuit



HeadDress (wireframe shown for clarity)



The HeadDress composite cannot use the head or spectacles slots - so the avatar's head will still be rendered and they can still equip spectacles.

FullHeadDress (wireframe shown for clarity)



The FullHeadDress composite uses all the same slots as the Head Dress composite, but also uses the Spectacles slot and (most importantly) the Head slot. This means that the avatar's head is not rendered at all. The Full Head Dress must cover or obscure the avatar's head entirely. The fact that the head slot is freed up means that both Full Head Dress and FullBodySuit have some extra memory available.

> ⚠ A full headdress neckline must not deform with the fat and thin blend shapes. You must create a larger neck area to compensate and make sure that the neckline is covered for both the fat and thin shapes.

Creating Composite Character Components

To create composite character components:

1. In Maya: slot + slot + slot = 1 piece of clothing (e.g. design a torso component, a legs component and a feet component).
2. In the Object Editor: Assign the component types to the clothing (e.g. for a tuxedo, torso+legs+feet).

In the client this appears as one clothing item, but occupies three slots on the avatar. This means that if the 'tuxedo' is equipped, all components in the torso, hands and legs are occupied at once.

Metadata for Composites

Clothing components include metadata that determines where the user access them in their wardrobe e.g. a hat appears in the hat chip within the wardrobe.
Composite character components default to the following metadata catalogue tags:

Composite	Catalogue tags
Legs+Feet (LegsAndFeet)	LegsFeet
Torso+Legs (TorsoAndLegs)	Outfit
Torso+Legs+Feet (TorsoLegsAndFeet)	Outfit
HeadDress	Hat
FullHeadDress	Hat
FullBodySuit	Outfit

The catalogue tags can be changed within the Object Editor if you want them to be listed in a different category within the wardrobe , see [Object Reference Material] for more information on metadata.

⚠ To prevent user confusion, if you change the default location for a composite, tell the user by stating this new location in the composite's description.

The chart below illustrates what atomic component each composite type is made up from:

	Legs+Feet (LegsAndFeet)	Torso+Legs (TorsoAndLegs)	Torso+Legs+Feet (TorsoLegsAndFeet)	HeadDress	FullHeadDress	FullBodySuit
Feet	X		X			X
Legs	X	X	X			X
Torso		X	X			X
Hands						X
Head					X	X
Spectacles					X	X
Hair/Hat				X	X	X
Facial Hair				X	X	X
Headphones				X	X	X
Jewellery (Right Ear)				X	X	X
Jewellery (Left Ear)				X	X	X

Notice that Hairstyle and Hat are regarded as the same class of object. This is because in PS Home, you can only wear either Hat or Hair at any one time. If the avatar is wearing Hair and then puts on a Hat, the game will remove the Hair (and vice versa).

The Head row represents the memory budget for an avatar's head. This is because when an avatar wears a FullBodySuit the head is completely covered and is removed by the game. This means that when creating these composites the memory and triangle budget for an avatar head can also be included in the overall total.

The mesh limit for each atomic component is 5 for LOD1, and 3 for LOD2 and LOD3 so, as in the other cases, the mesh limits for each composite is the number of atomic components it is made up from multiplied by five or three. Thus, a Legs + Feet component would be allowed a maximum of 10 (LOD1) and 6 (LOD2 and LOD 3) meshes and a FullBodySuit a maximum of 55 (LOD1) and 33 (LOD2 and LOD3).

Fitting Components Together

Intersections: Overlapping, Skin Weighting, Holes and Seams

All character components should work well with other character components. If character components in slots on the avatar overlap too much onto other slots, it causes problems. When creating a character component, it is important to keep in mind how well this component will work with all the other possible types of character component it may be combined with in PS Home. For example, gloves that must fit with a variety of shirts with sleeves.

You can make your character component smaller, all the way down to the skin of the avatar. You can also make your character component bigger. You can also create components that leave skin exposed on the avatar, such as a t-shirt that leaves arms exposed, or an open-neck shirt. Whatever size component you make, be mindful of how it will look when combined with other character components or with the avatar. For example, make sure that there is no intersection between the neck area and the underlying mesh.

To avoid problems with intersection, pay particular attention to the areas where components meet other components (the ankles, wrists, waist) or skin (the neck, wrists). Also, consider how components will fit on avatars that are thin or avatars that are fat. For example, you can model a t-shirt to hang off a thin target but stretch on a fat target.

Intersection on the avatar is not the only factor. Incorrectly modeled components can intersect with scenery, such as a tall hat intersecting with a low archway.

For guidance character component creation, see also the following sections:

- Building LODs
- Character Component Modeling Best Practice
- Dimensions
- Creating Fat and Thin Targets

- General Character Component Quality
- Environment Rules and Conventions
- Applying Skinning and Weighting

QA test character components against a large number of existing character components and body weights. Character components that clip, intersect, or cause corruption in the interface area with other components are unlikely to pass Quality Assurance (QA). Character components that overlap each other in a way that might be expected are not considered problematic, but avoid clipping and intersection wherever possible by testing against as many other assets as possible.

- See Character Viewer for information on how Character Viewer profiling can help you test your character components.
- You can profile The Dev Debug command `debugClothingLimits` can be used to profile character component dimensions. The command toggles geometry bubbles around an avatar that show the maximum recommended dimensions for each type of character component. For more information, see Profiling Character Components.

Watch out for some parts of the component sticking through the overlapping component. For example, widely flared cuffs may cover and overlay fitted gloves, and gloves with a wide fur trim may successfully overlay most shirts, but a shirt with flared cuffs and gloves with a fur trim may intersect at the wrist area.

Overlapping can be problematic, but so can leaving holes or not matching up seams adequately. Certain areas, such as the neck and hands, are particularly important if you need to match exactly. All components should extend to the borders, but not too much beyond the borders. Seams should invisible. For example, in the neck area, the vertices do not have to physically snap along the border, but they must intersect to create a solid connection between the neck and shirt. For more information, see the sections on intersection and holes.



Note that:

- If you are concerned about whether your character component will pass QA, or you want to make a character component that will not adhere to these guidelines, speak to regional SCE about it first.
- If you want to make a character component that needs to occupy more than the slot for it on the avatar, you can make a composite character component.

Templates

After selecting the type of component you want to make, a template shape is provided for you to get you started. These templates provide useful guidelines that help you to make components that are compliant with the guidelines and combine well with others in PS Home. For example, the template shape indicates the typical size the component should be.

Menu Categories for Clothing

The wardrobe menu in the client has specific categories where clothing of that type is listed:

- `Head`
- `Hands`
- `Torso`
- `Legs`
- `Feet`
- `Jewelry` (which includes `Left`, `Right` and `Both`)
- `Accessories` (which includes `Glasses` and `Headphones`)
- `Outfits`

The category a character component appears in depends on what is set in the metadata description of the object.

Atomic Character Components in the Menu

For atomic character components that occupies a single slot on the avatar, it is a simple correlation between the avatar slot the character component is modeled for in the tools and the menu category where the clothing will appear in the client.

E.g. If you make a t-shirt you would tag it within the metadata, because a user is likely to look for a t-shirt under the `Torso` category.

This is an automatic process - the character component is exported for a single avatar slot and the component metadata reflects this slot and assigns it the corresponding menu category.

Composite Character Components in the Menu

For composites (clothing that occupies multiple avatar slots), it is less straightforward.

E.g. FullBodySuits usually appears in the `Outfits` menu category, as does Torso+Legs+Feet. But where does one place a Legs+Feet composite such as a skirt and tall boots? In the `Outfits` or `Legs` or `Feet` menu categories? Maybe in all three?

For composite character components, you must judge where users will expect to find this composite component in the menu.

Tools Suggestions for Menu Categories

The tools suggest a certain slot in the client menu based on the type of component (atomic or composite):

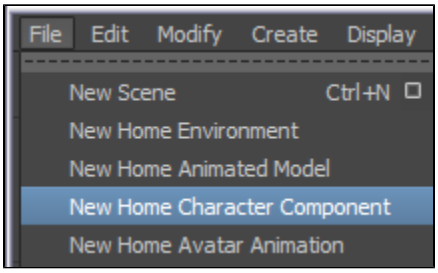
Component Metadata	Client Menu Categories
Atomic Components	
Hands	Hands
Torso	Torso
Legs	Legs
Feet	Feet
FacialHair	FacialHair
Hair	Hair
Hat	Hat
JewelLeftEar	JewelLeftEar
JewelRightEar	JewelRightEar
Glasses	Glasses
HeadPhones	HeadPhones
Composites	
LegsAndFeet	Legs Feet
TorsoAndLegs	Outfit
TorsoLegsAndFeet	Outfit
FullBodySuit	Outfit
HeadDress	Hat
FullHeadDress	Hat

Creating a Character Component

Maya provides a shelf of tools and various templates to make it easier for you to create components.

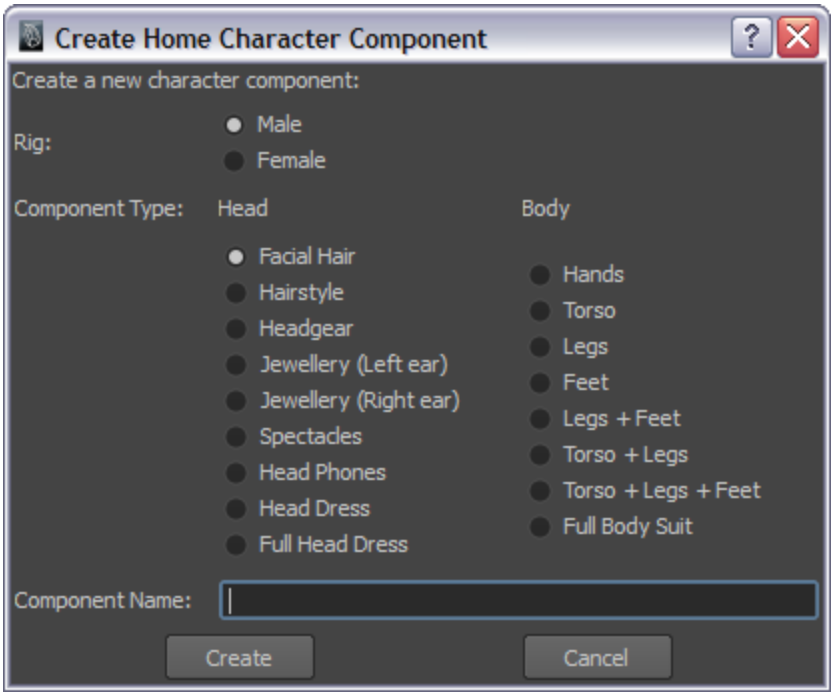
To create a character component:

1. Select the template you want to modify from the menu.
2. Select File > New Home Character Component.



The Create Home Character Component dialog opens.

3. Select the required character component type:



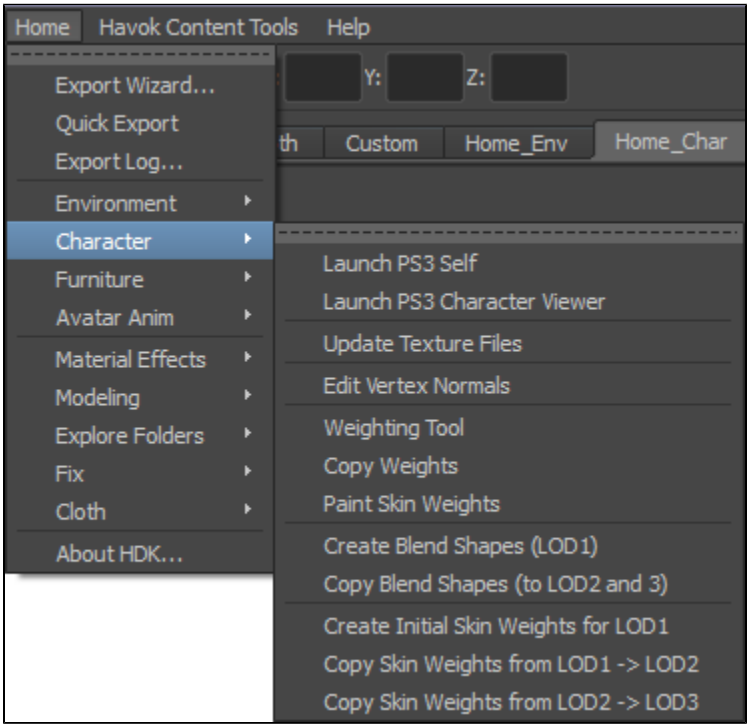
For example, to create a new custom shirt for a male character:

a. Select the Male rig and the Torso component type.
b. Enter a name for the shirt, then click Create.
   This selection loads the Home Character rig with the default Home t-shirt torso, selected and framed in the Perspective Panel.

4. Click the Home_Char tab 

to display the Character Shelf of tools available for creating character components. For a list of Home_Char icons and their functions, see The Maya Interface.

   The character component options are also available from the Home drop-down menu:



Creating Headgear Components

Creating components for the head (for example hair, hats, spectacles, facial hair, headphones, etc.) has a slightly different workflow and rig to other components. In addition, hair and hats occupy the same slot (headgear component) and only one type of component can be assigned to a character at a particular time, i.e. if the avatar has a hat equipped, the hair they had will be entirely replaced by the hat. This means that any hair visible beneath a hat on a character will have to be created as part of the hat itself.

Hats

Hats are set up in the same way as hair, but particular attention should be paid to hats which have solid regions, such as brims or hard areas. Any bone modification to the head may deform and affect the appearance of the hat. It is important to experiment and weight accordingly. See Creating a Hat for the detailed procedure.

Custom heads are not supported in PS Home, but if you wished to create a custom head covering (something such as a motorcycle helmet) you could create it as a hat. The issue to bear in mind is that the head will still be present beneath it so your motor cycle helmet 'head' (i.e. a hat) would need to be expansive enough to avoid the avatar's head geometry poking through. Hair is turned off when the avatar wears a hat, so there is no need to worry about that protruding, but you would need to consider all the other possible head decorations (such as glasses, headphones, etc.) by modeling the helmet to accommodate glasses being equipped.

Example of Headgear Component Creation

If you wanted the motor cycle helmet to completely cover the entire head (including face) and incorporate all the slots taken up by the head components - i.e. hair/hat, facial hair (for male rigs), jewellery (right and left ears), head, spectacles, headphones - you should create a Full Head Dress composite character component. See Composite Character Components for more information.

See also:

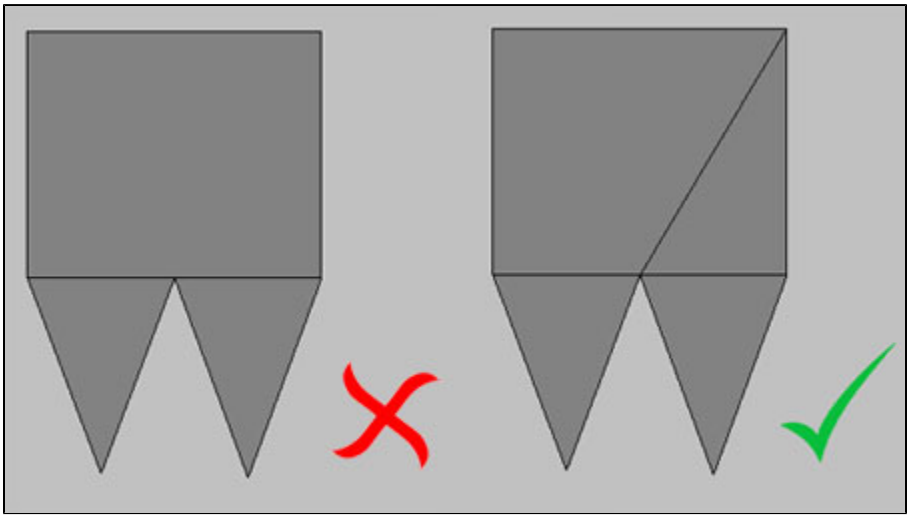
- Creating a Hat
- Creating Headphones
- Creating Jewelry
- Creating Spectacles

Hair Creation Guidelines

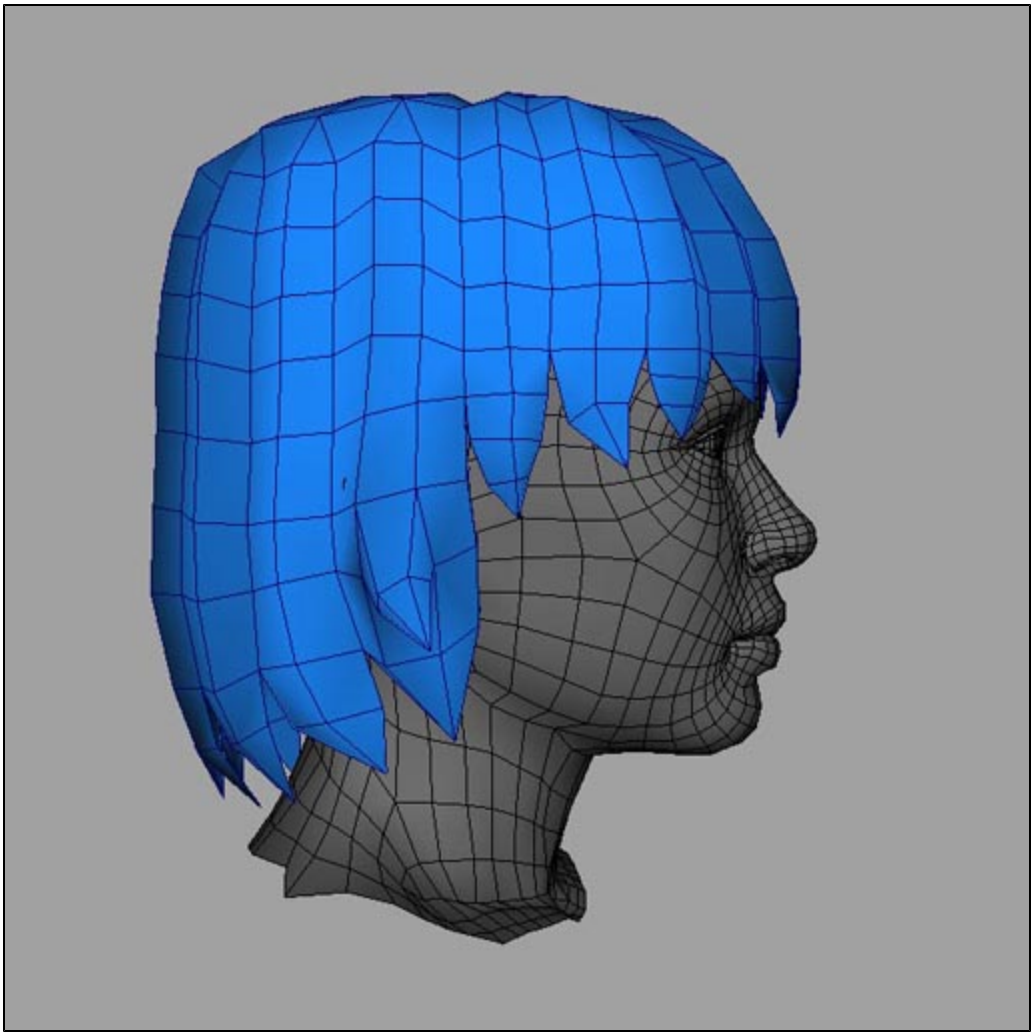
Hair components are different to other character components, as they have unique shaders and focus more on geometry than transparency.

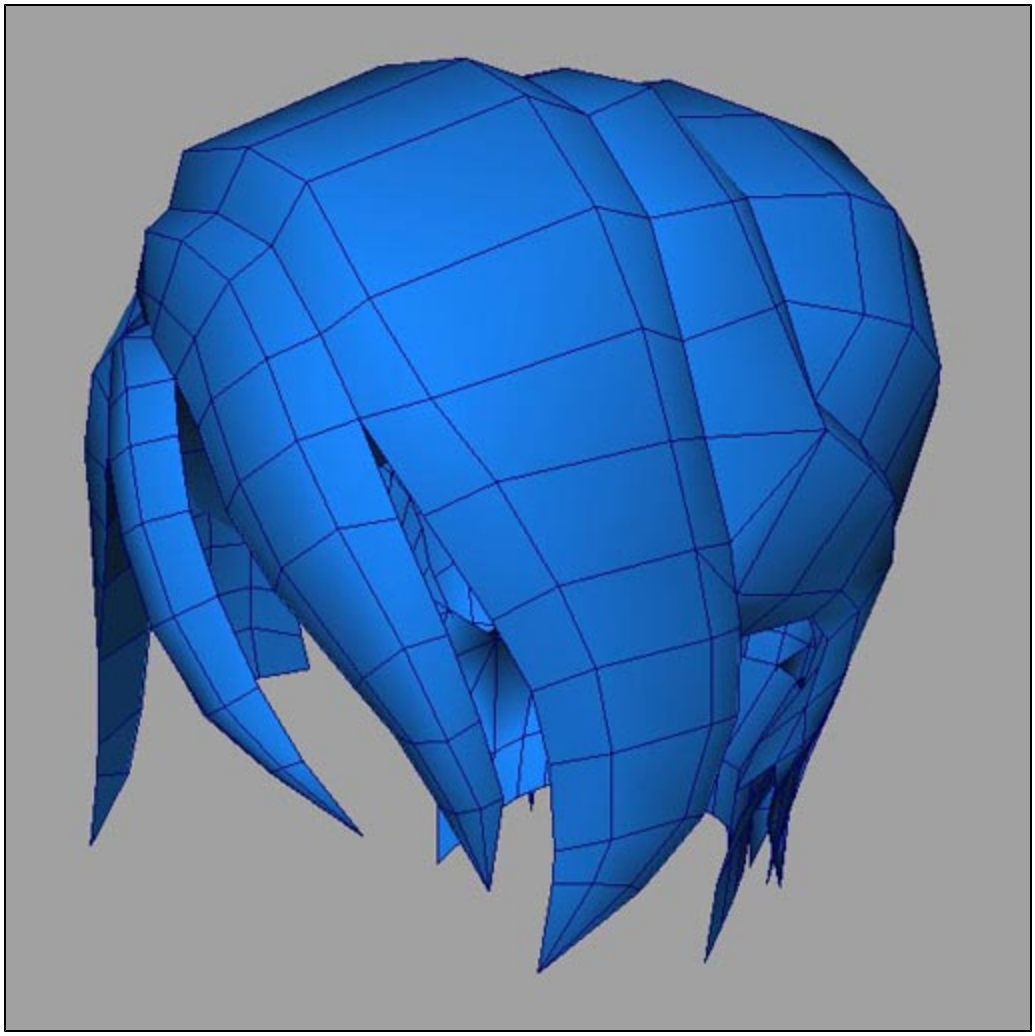
In PS Home, there are technical requirements and limitations regarding hair. In particular, reducing the amount of transparency and increasing the focus on geometry to describe the main shapes of the hair has significantly influenced how hair appears. This means that hair in PS Home takes on a stylized, almost anime style and appearance.

When exporting all UVs on the mesh must be in the 0 to 1 range. Specular will not render on geometry with UVs outside this range. Modeling with N-sided Polygons should be avoided as these will result in rendering artifacts on the console - use Quads or Triangles only. Below is an example of a 5 sided polygon and how the mesh would need to be edited to avoid problems.



As shown in the image below, the major shapes of the hair are defined as solid clumps requiring very little transparency on the main body of the hairstyle.



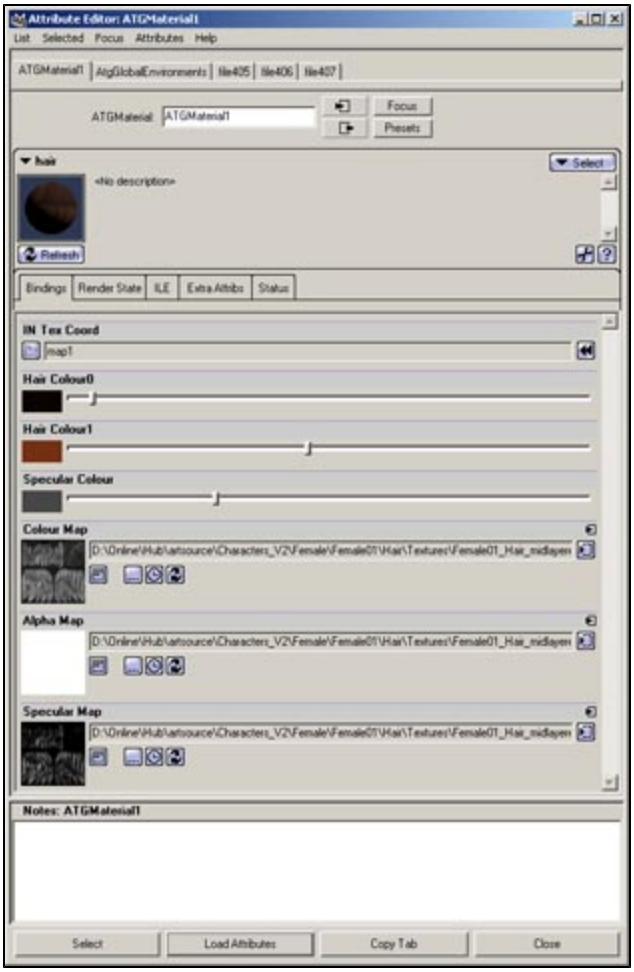


See HDK Tools Validations for triangle limits.

> ⚠ The triangle figures are upper limits. It is recommended that levels below these are used whenever possible.

Textures

The hair material is slightly different to the default, as it has no normal maps and replaces that with a Transparency layer. It also features anisotropic shading to better recreate the way light interacts with the surface of the hair.



As shown below, the texture is grayscale and the hair color is created by a color ramp between two color values specified in code, where you can also set up preview colors.

The top slider is the darkest color, the middle slider the lightest and the bottom slider is the specular color:



Specular map



Color Map



Transparency map



Specular map alpha channel

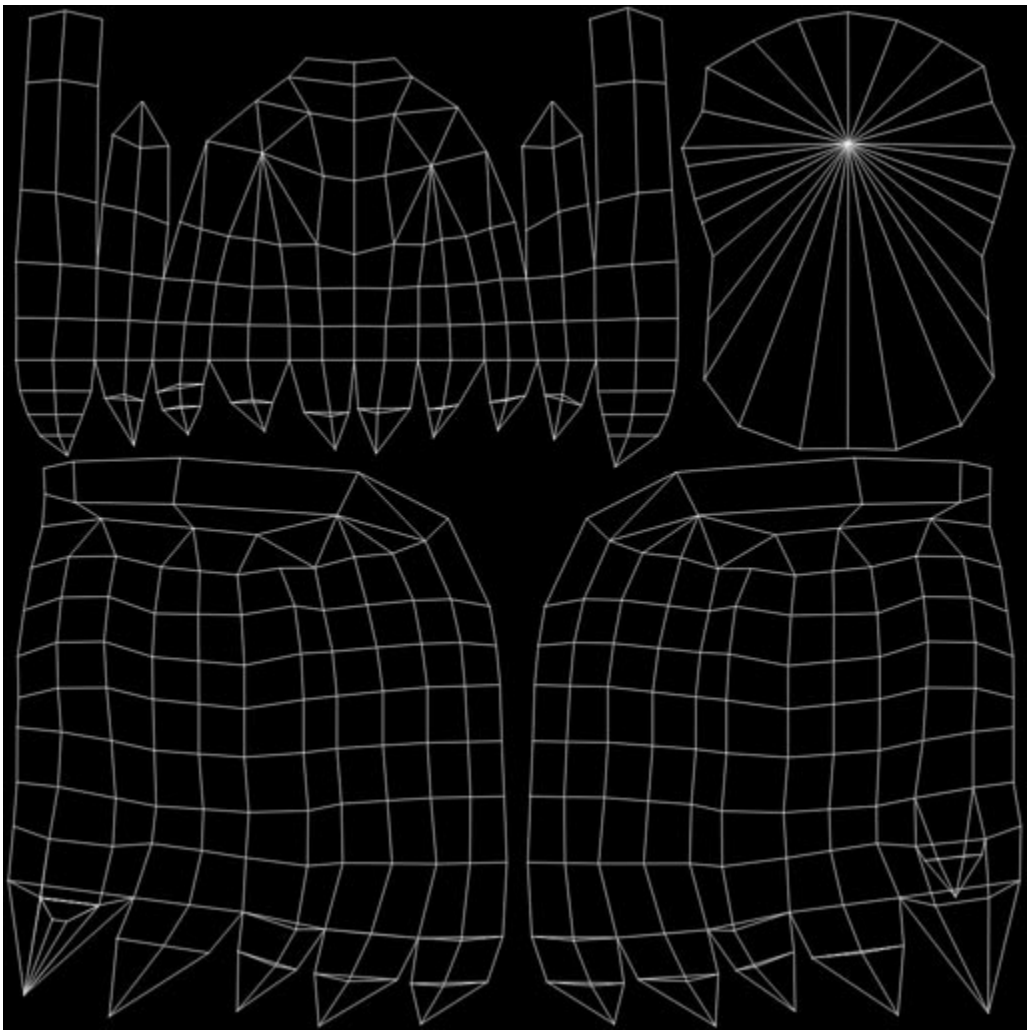
The specular map defines the amount of reflection but it also contains an alpha channel which defines how crisp the environment map is. White is mirror reflectivity, Black is blurred out.

Creating the Color Maps

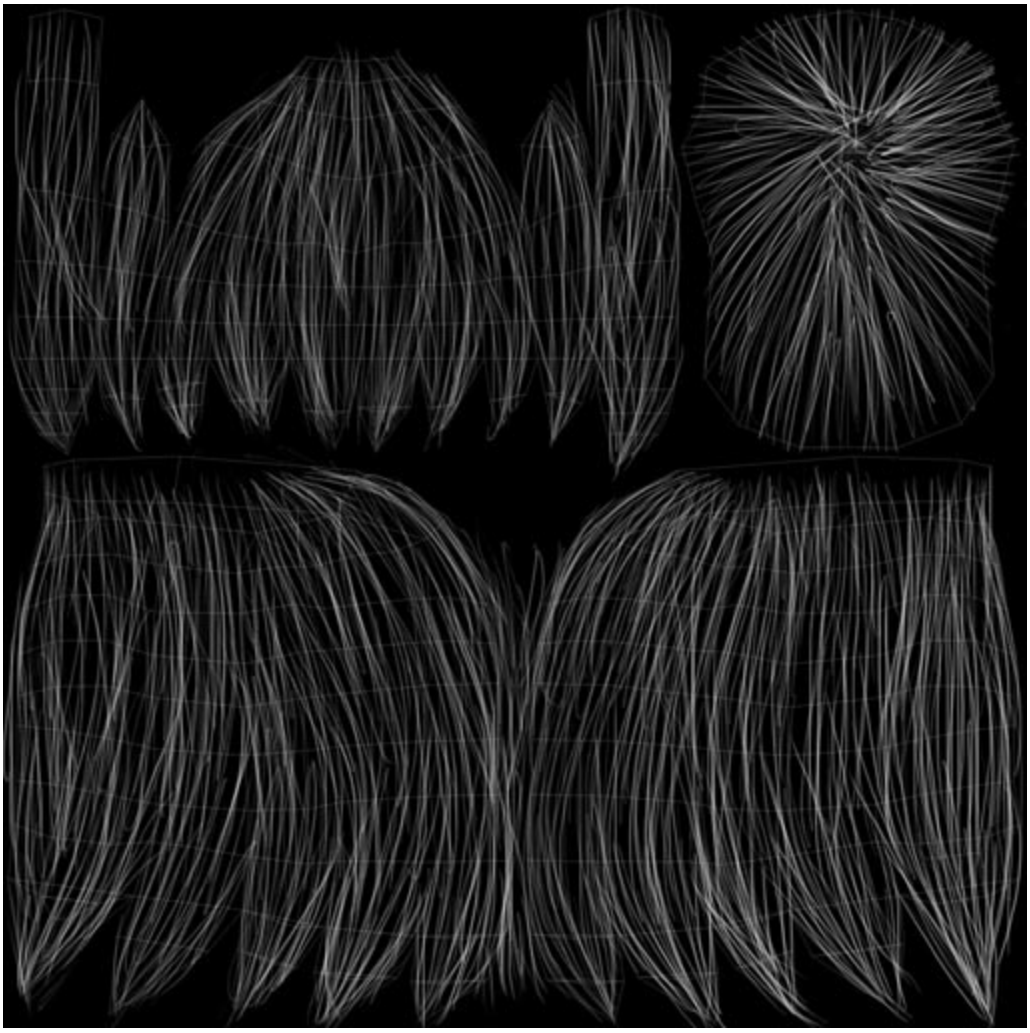
Creating the color texture requires an even amount of white to black ratio, to best use the full range of the color ramp. This

could be achieved by, for instance, working at 1024x1024 and drawing the hair with a hard 2 pixel brush.

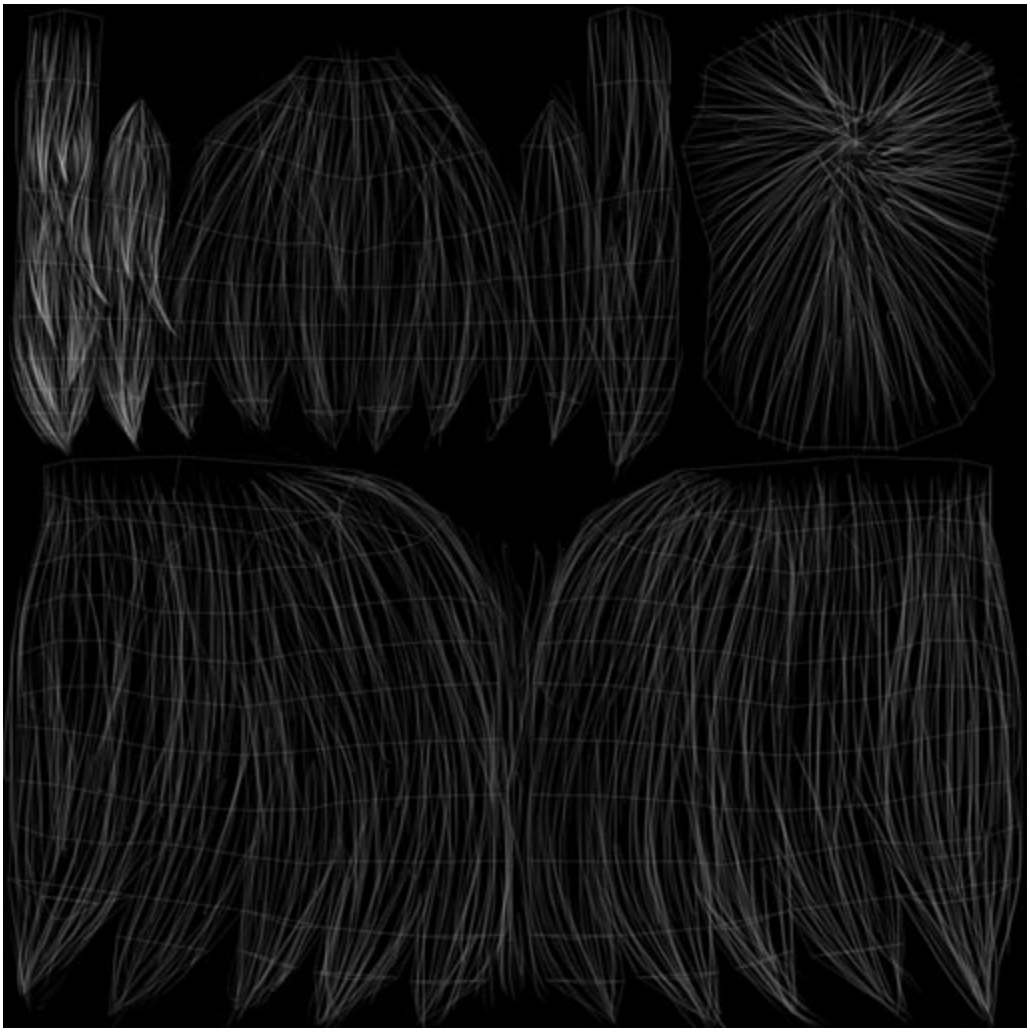
1. Start with the UVs laid out neatly, the hair strands should run top to bottom for the anisotropic shading to work



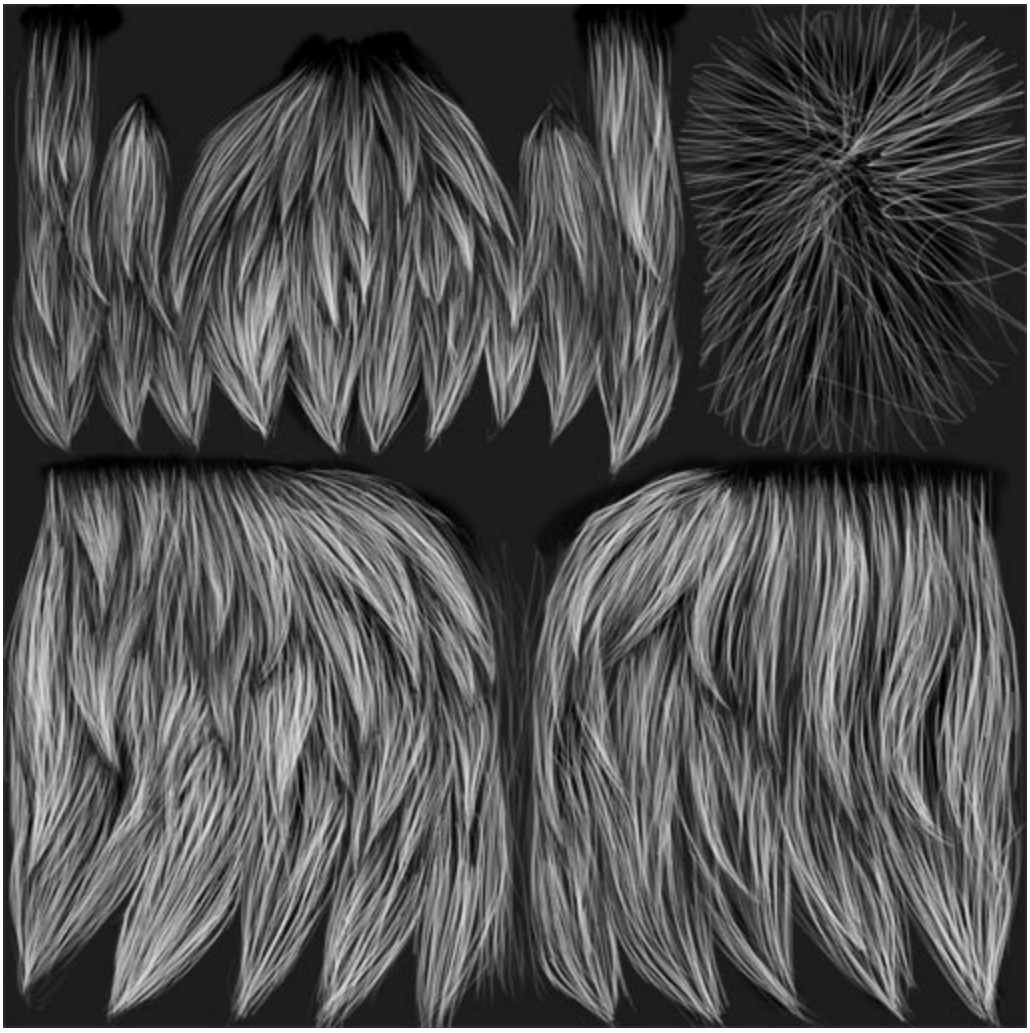
2. Next, sketch the rough hair layout to make sure the hair strands flow and the mapping works



3. Set the rough layer to 50% opacity and on a new layer begin to work over the top adding detail.

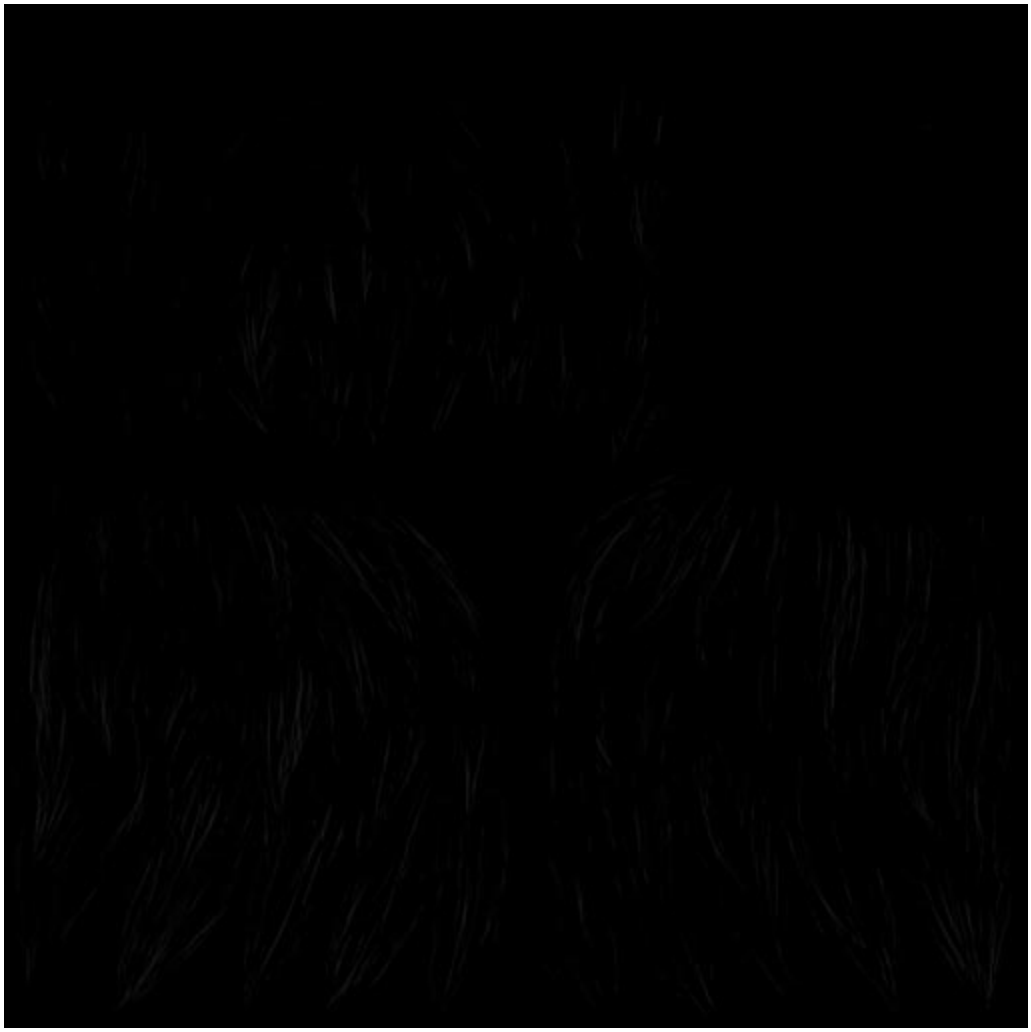


4. Finally, set the background to 90% Grey and then add some black shading behind the stands to add depth to the hair.



The specular map is the same texture, but with higher contrast, and the alpha channel for that should be very dark (almost black) to avoid bright scintillation on the hair (caused by drawing environment maps into the thin strands):





Specular Alpha Channel

Finally, the transparency map is hand drawn over the color map. It is advisable to keep the strands thick, as this suits the chunky style of the hair (see the close-up image below).



The final effect should resemble the image below:



The most important thing to bear in mind is the size of the brushstrokes used for the strands of hair and the overall tone of the image. A desirable effect can be achieved by, for instance, working at 1024x1024 and drawing the hair with a hard 2 pixel brush. To make it easier to gauge the correct tonal range of the color map, refer to table below that lists the HSV Values for the hairstyle shown above:

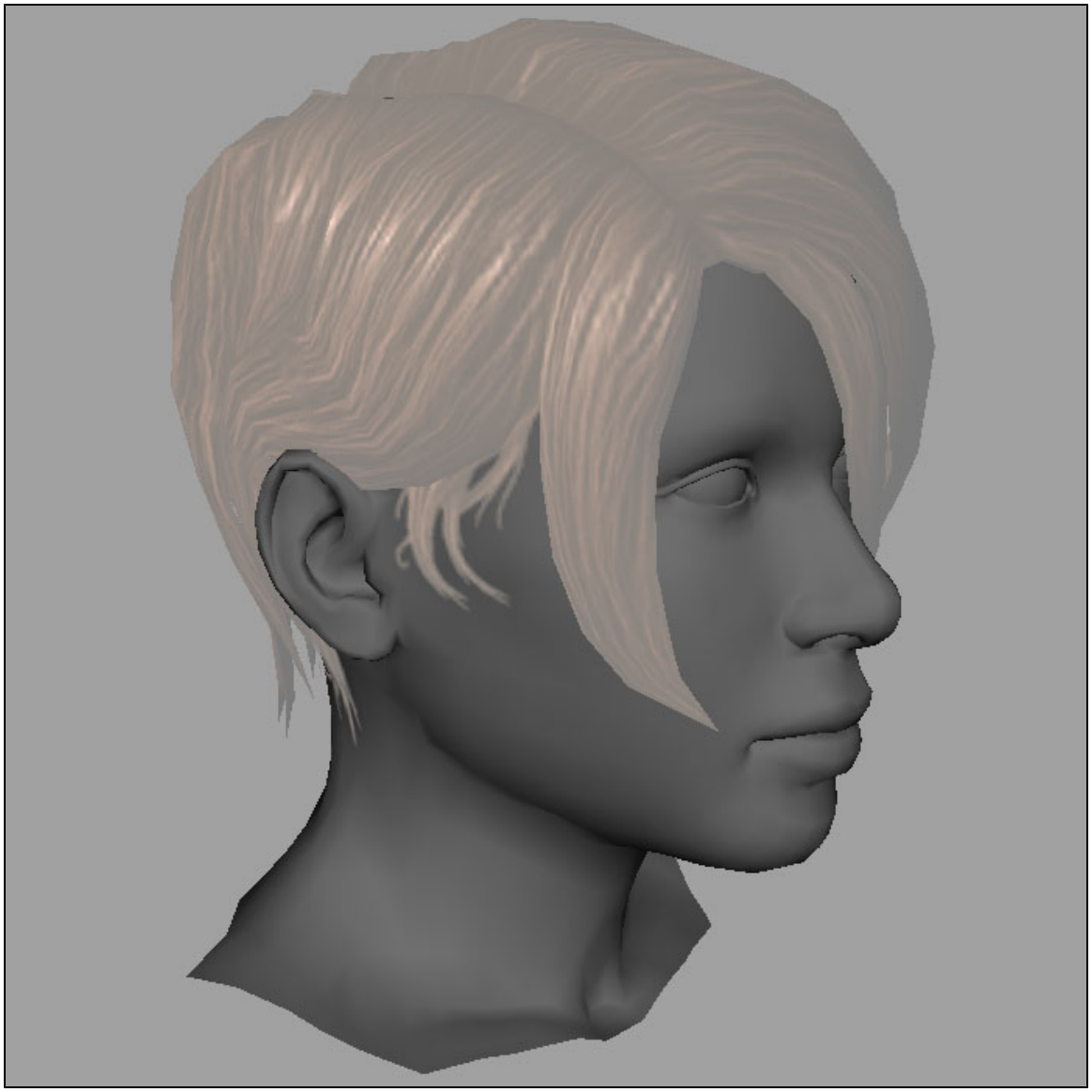
	Dark Color		Light Color
H	27.50	H	17.07
S	0.764	S	0.641
V	0.084	V	0.533





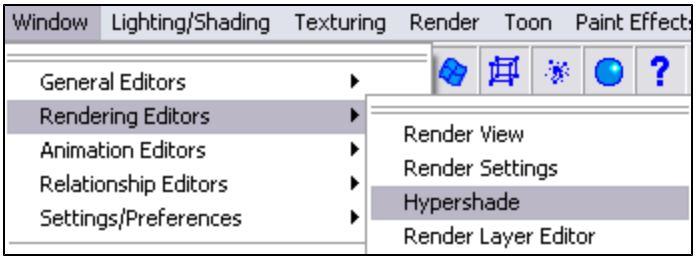
Troubleshooting Guide

A common Maya problem is that, when a hairstyle has been created, the material has been applied, and the environment set up, the mesh can look washed out, as shown below:

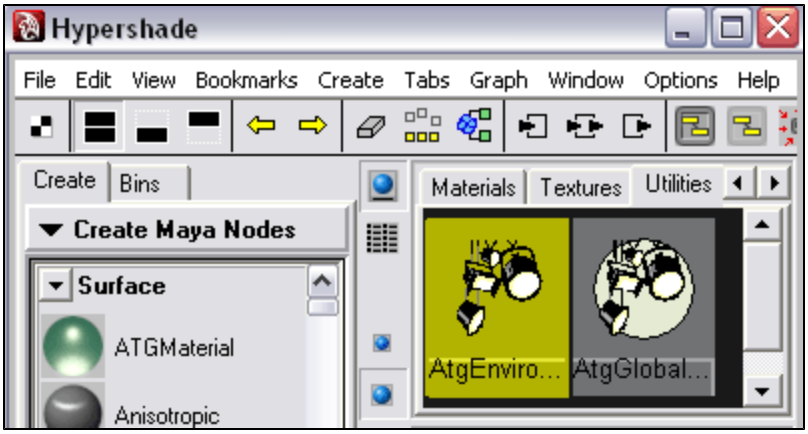


To resolve this issue:

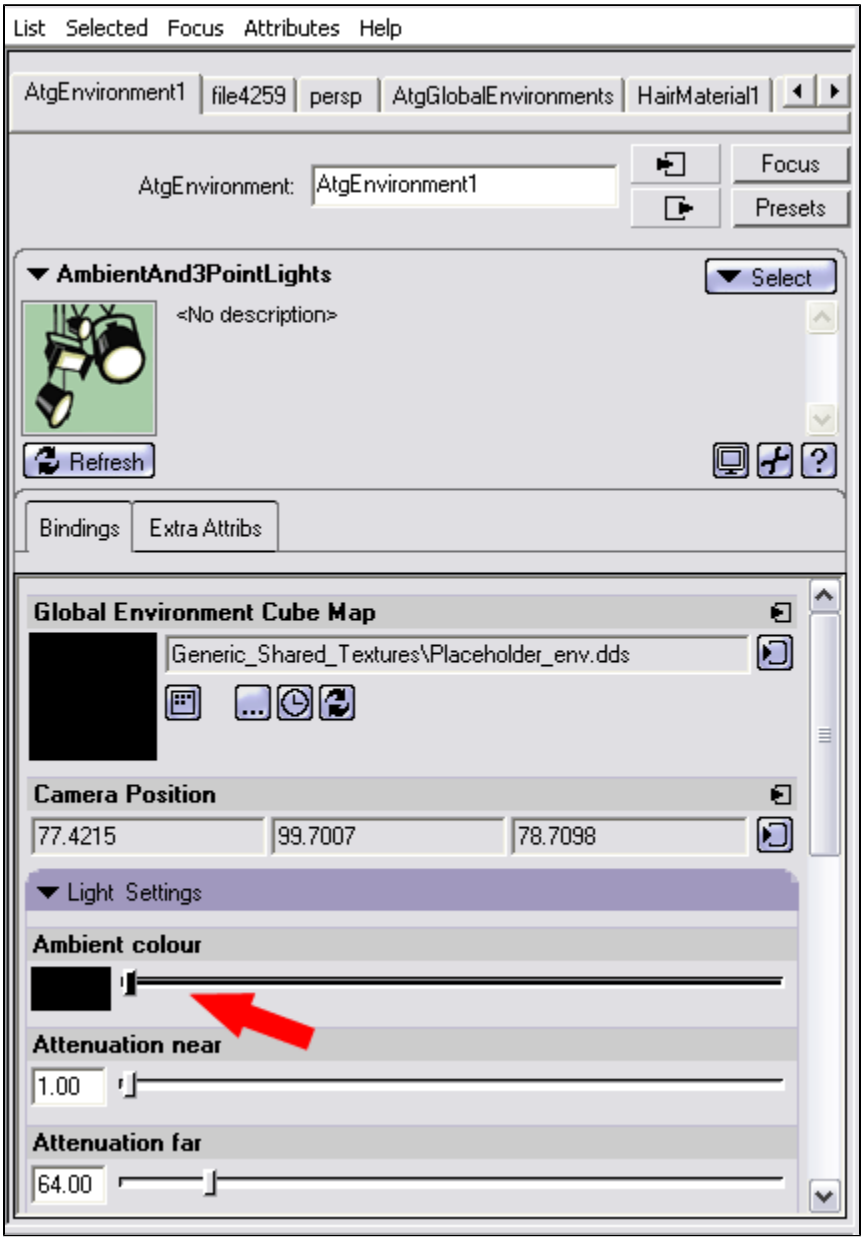
1. Select the ATG Environment Node under the Utilities tab in the Hypershade:



2. Open the attribute editor:



3. Select the Ambient Color Slider and drag it to the left to make the color black.

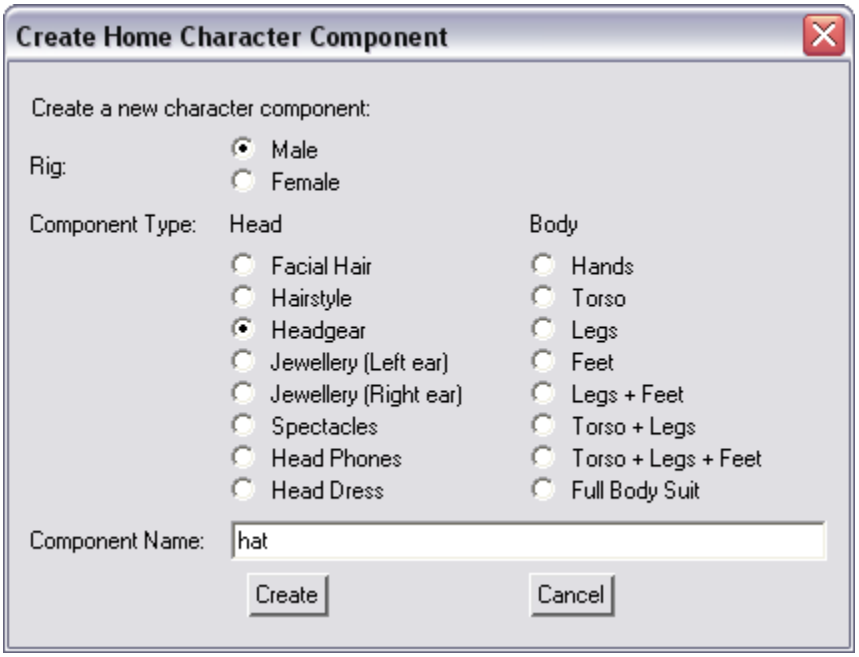


Creating a Hat

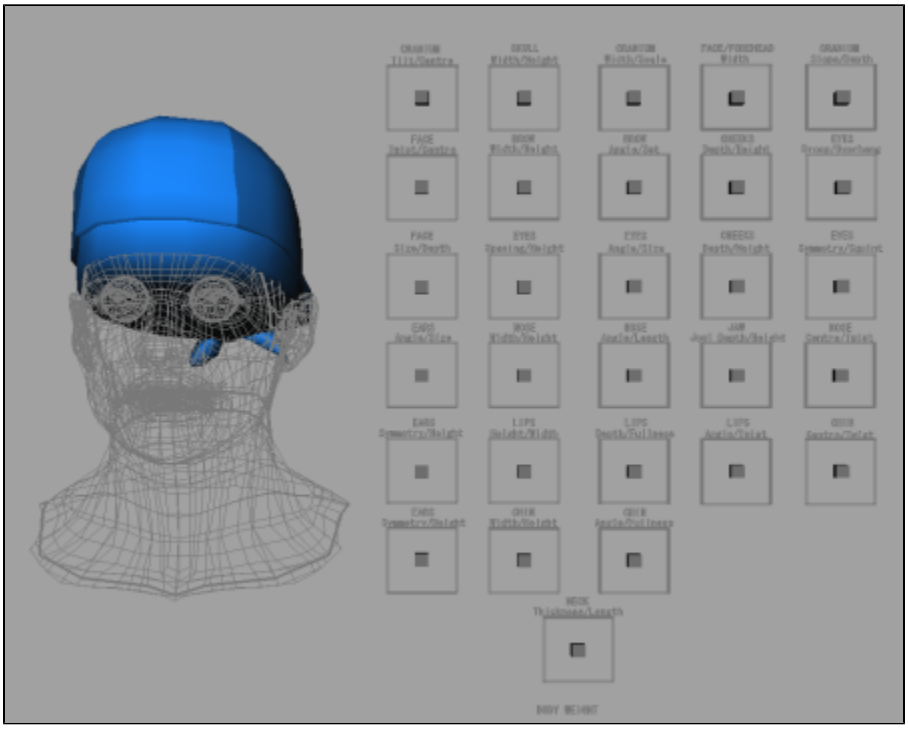
To illustrate headgear component creation, the example described here is the process of creating a hat.

To create a hat:

1. In Maya, select File > New Home Character Component.
2. Select the Rig gender and then select Headgear for the Component Type.
3. Type a component name and click Create.



This loads a different Home Character rig fitted with a default baseball cap object.



The objects to the right hand side are an array of controllers that govern the movement of various joints. These in turn control the permitted limits of the Male character's head, including the hat. These are a required part of the rig and can be manipulated by moving them left and right, and up and down. They are intended to be used to test the results of the skinning process. To the left can be found the control mesh and the default hair/hat object.
In the Outliner is a group called baseGroup. Expanding this reveals that the required three LOD meshes have been imported bearing the name Male01_Hair_your chosen name_lod1, 2 and 3, where again LOD1 is the highest resolution mesh.

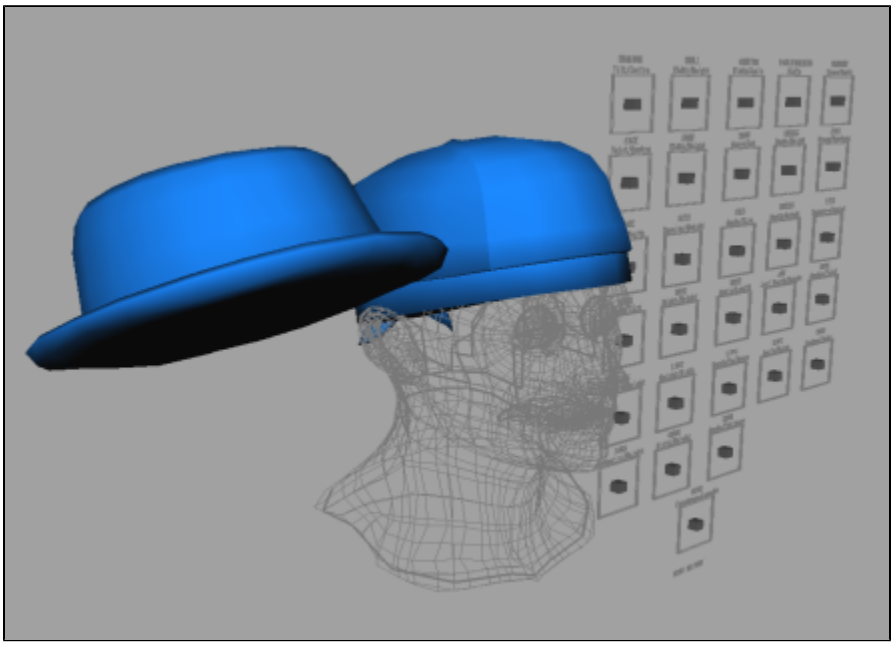


4. For now, hide LODs 2 and 3 by switching off their visibility in Maya's Layers panel.

5. You can edit the LOD1 base asset in one of three ways:

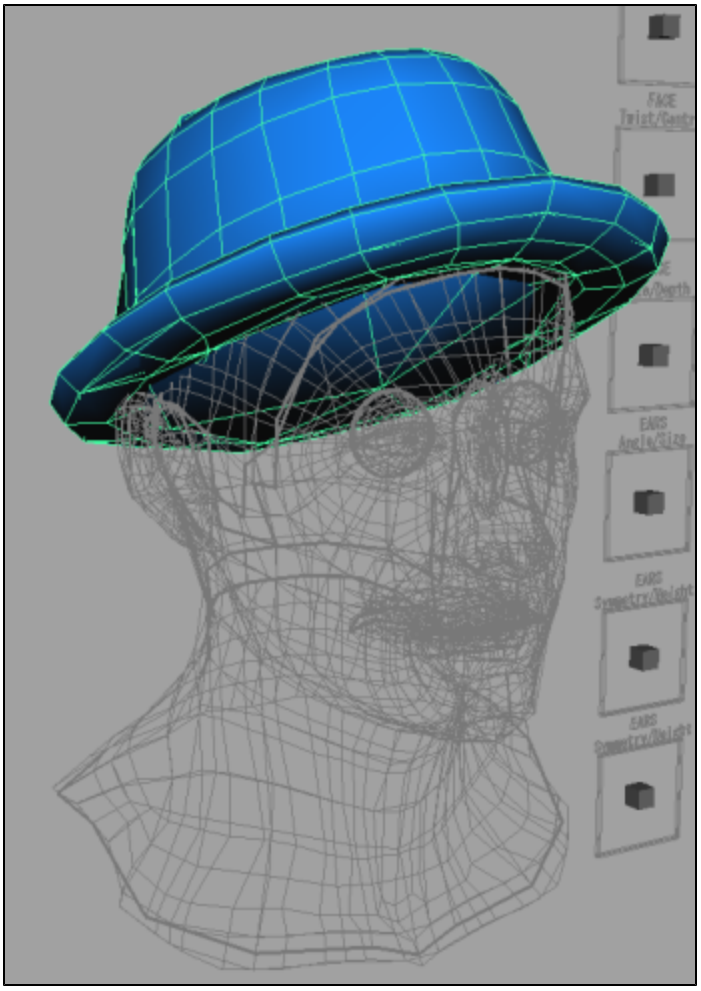
    - Replace the color texture map with a customized version, leaving the geometry unaltered.
    - Edit the provided geometry and texture to create a unique customized component mesh.
    - Delete the default meshes (including the two other LODs) completely and replace them with a custom made character component. To use customized meshes, the meshes must have the same names as the default objects that they are replacing.

    In this example, we import a bespoke character component into the scene along with its LODs - a porkpie hat.





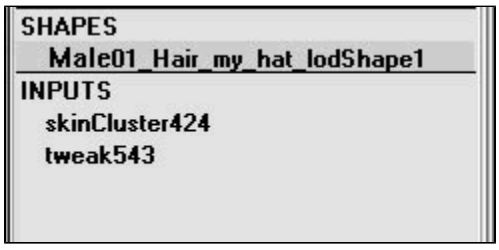
You can delete the default headgear objects, move the imported equivalents into the baseGroup and transform them to the correct place as follows:





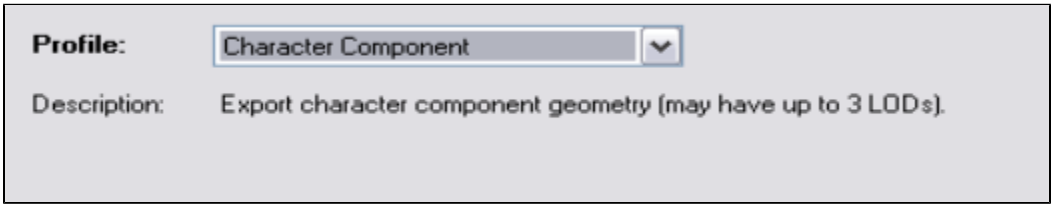
6. Select  the Home Toolbar (Character Components) to perform the initial skin weighting operation on LOD1.
7. Select  to transfer the weighting information from LOD1 to LOD2.
8. Finally, select  to transfer the weight data from LOD2 to LOD3

When completed, you will see by selecting the LODs that skin clusters have been created and are displayed in the Channel Box.



You can now use the Export Wizard to export your headgear in the usual way. Make sure you select the correct export

profile, e.g. 'Character Component'.



Creating Headphones

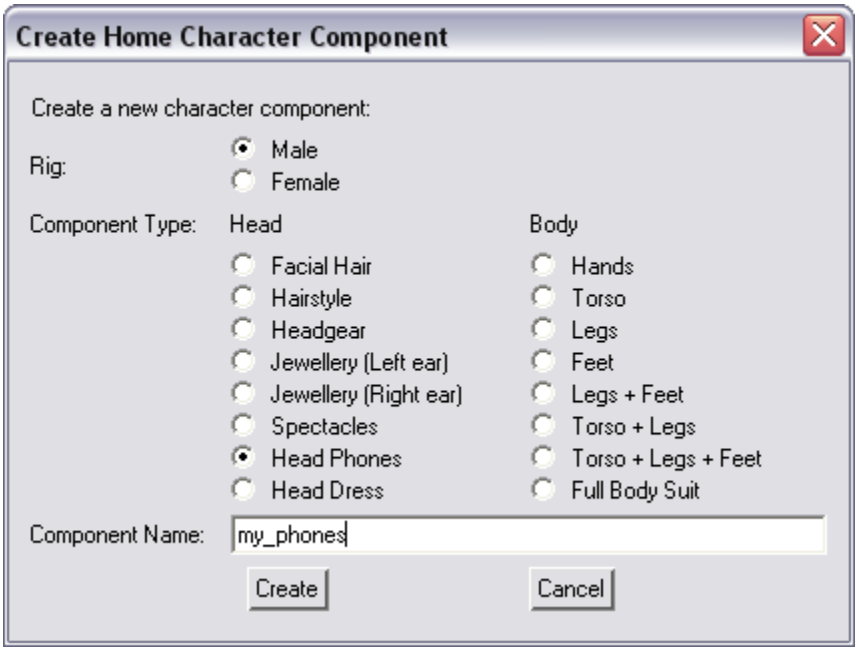
There are some limitations to skinning headphones. The key limitation is how the modification of the ears directly influences the headphone earpieces, often scaling to exaggerated proportions. Currently there is no improved solution.

To create headphones:

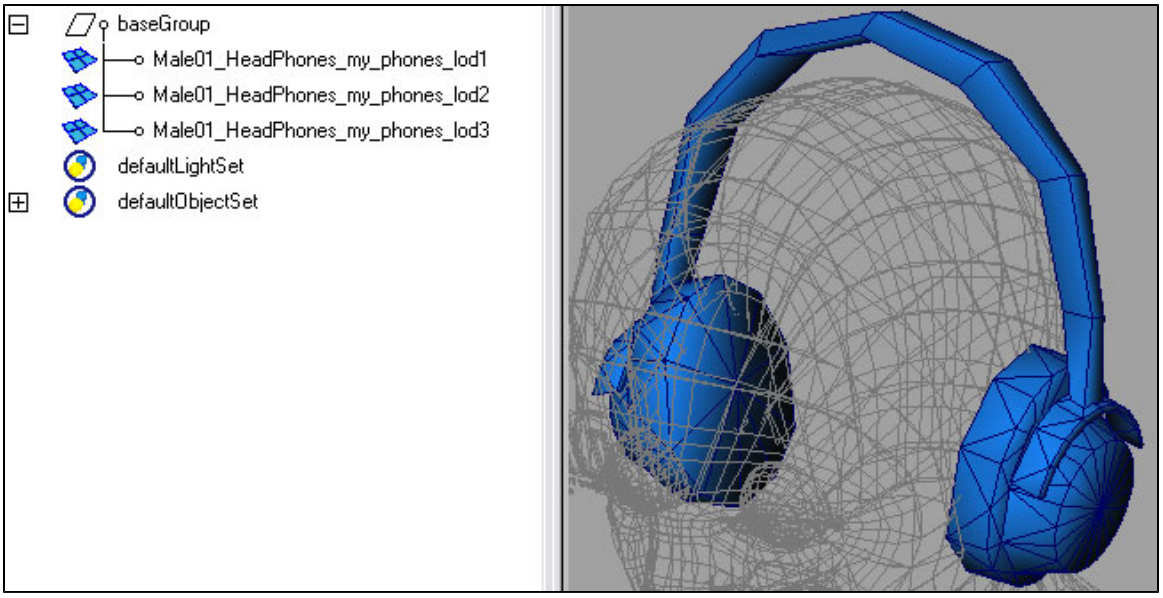
1. Open Maya and select File > New Home Character Component.

   The Create Home Character Component dialog opens.

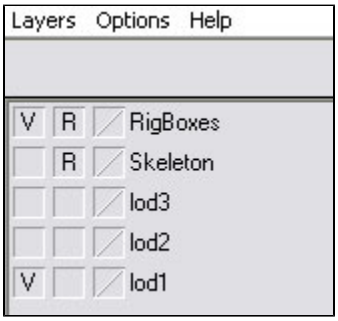
2. Select the rig (Male or Female) for the gender type, select Headphones, type a component name and click Create.



The rig is loaded and a default baseGroup of three spectacle LODs created with LOD1 visible:



LODs 2 and 3 are also loaded but their Layers are set to be invisible:

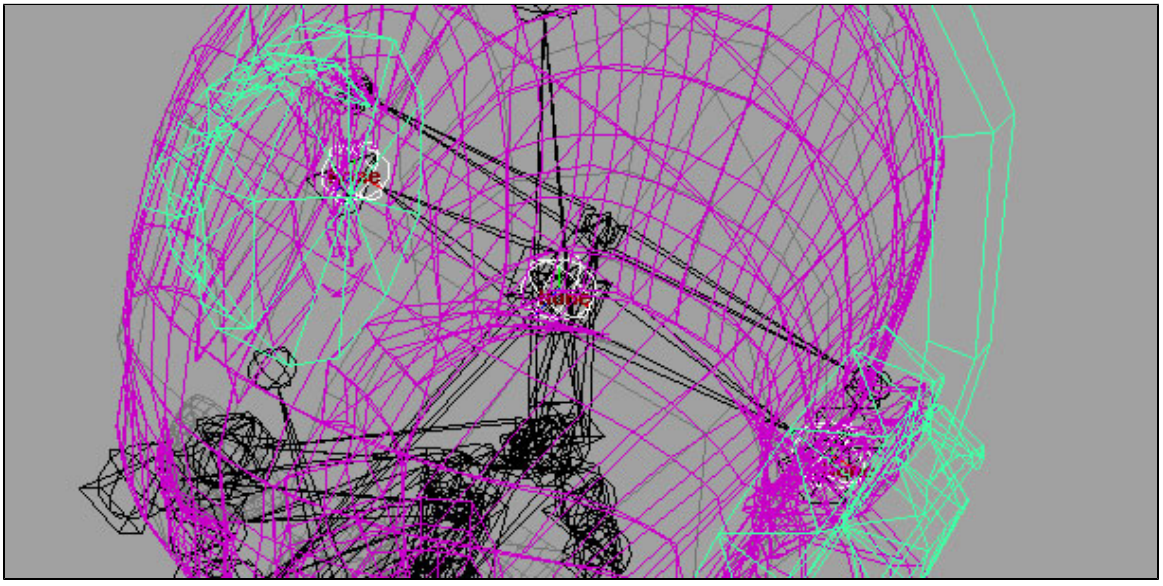


LOD1

Make sure that the Time Slider is to frame 0.

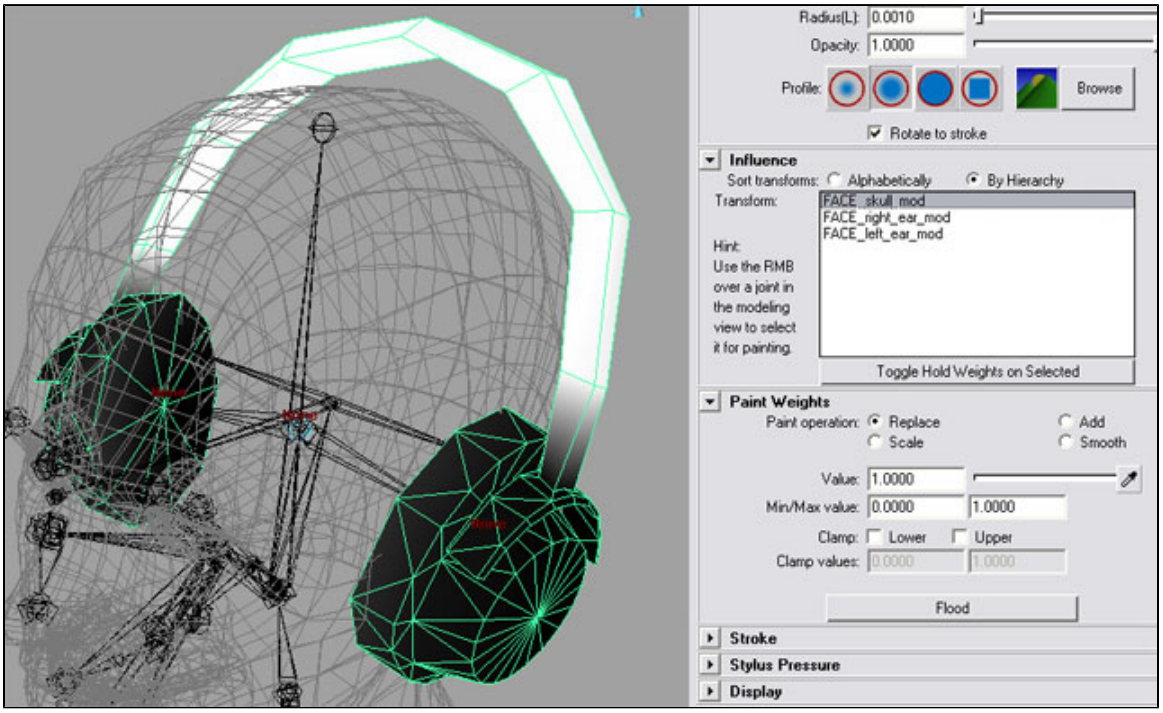
Select the LOD1 mesh and skin to the following joints:

FACE_skull_mod, FACE_right_ear_mod, FACE_left_ear_mod

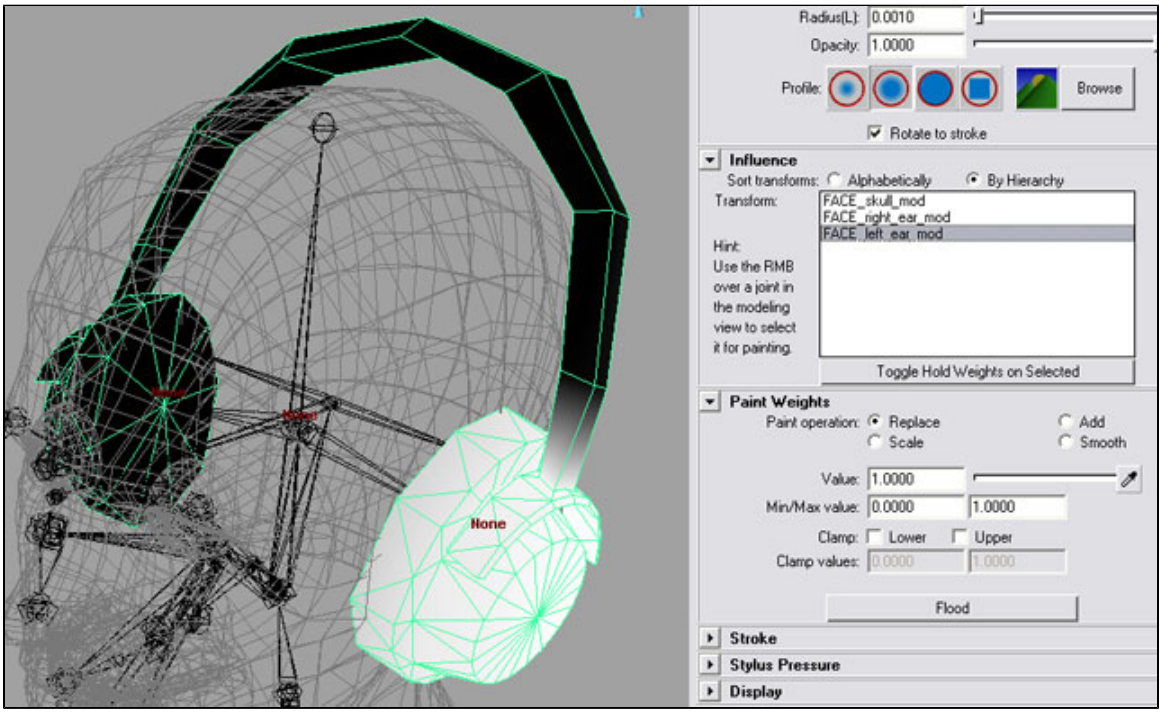


The weights will then probably need to be corrected by hand using the Paint Skin Weights option as follows:

The band should only have weighted influence from FACE_skull_mod.



The ear pieces will either be weighted to FACE_right_ear_mod or FACE_left_ear_mod, depending on which side is being modified:



In some cases, there may need to be some weight blending between the FACE_skull_mod and FACE_left/right_ear_mod influences, depending on the model.

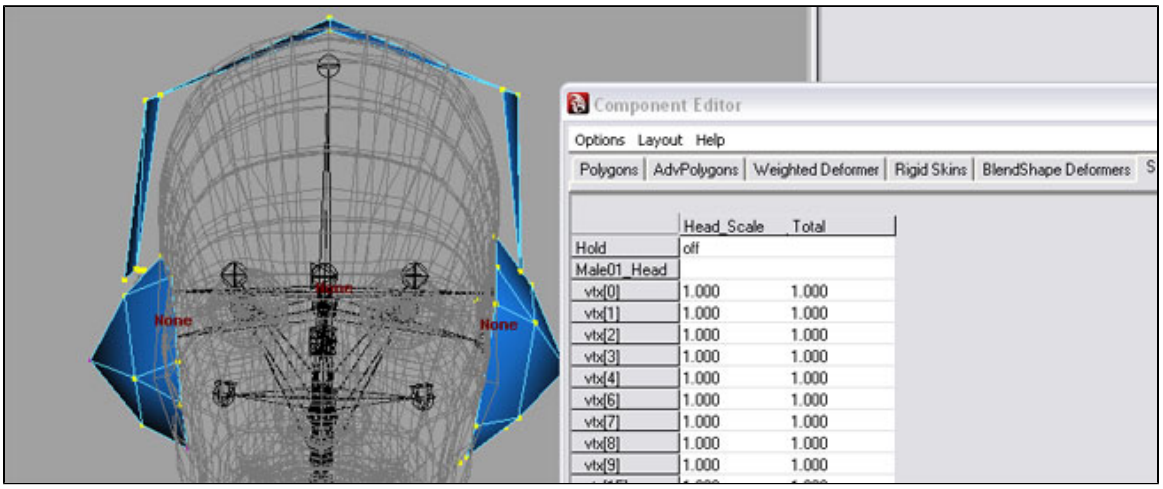
LOD2

For LOD 2 repeat the LOD1 process above.

LOD3

Reset the Time Slider to frame 0.

Skin the mesh to the Head_Scale joint. This will give it a weighting value of 1. This is all that is required for LOD3, as it will only take on the head scaling attribute. This can be verified by viewing the vertex weight values in the Component Editor:



Creating Jewelry

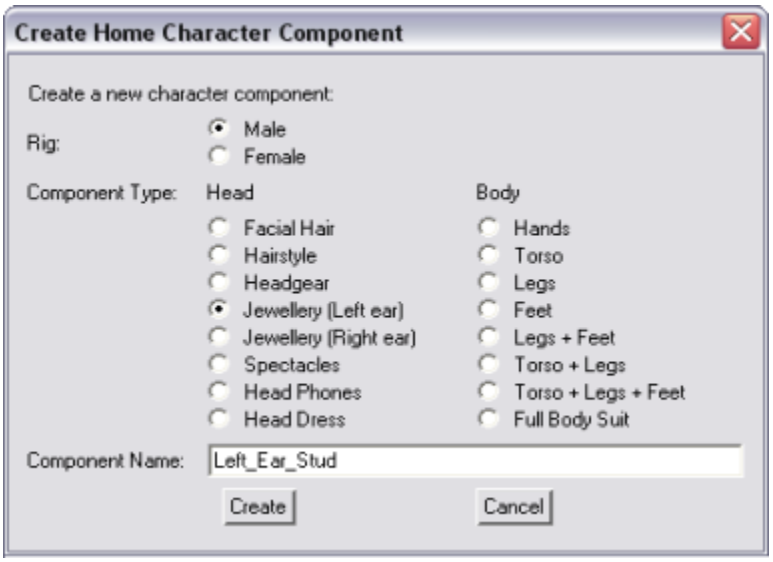
> Currently only earrings are supported.

The method for skinning and exporting jewelry is as follows.

1. Open Maya and select File > New Home Character Component.

   The Create Home Character Component dialog opens.

2. Select the rig (Male or Female) for the gender type, select Jewellery, enter a component name and click Create.



   The rig is loaded and a default baseGroup of three ear stud LODs created with LOD1 visible:

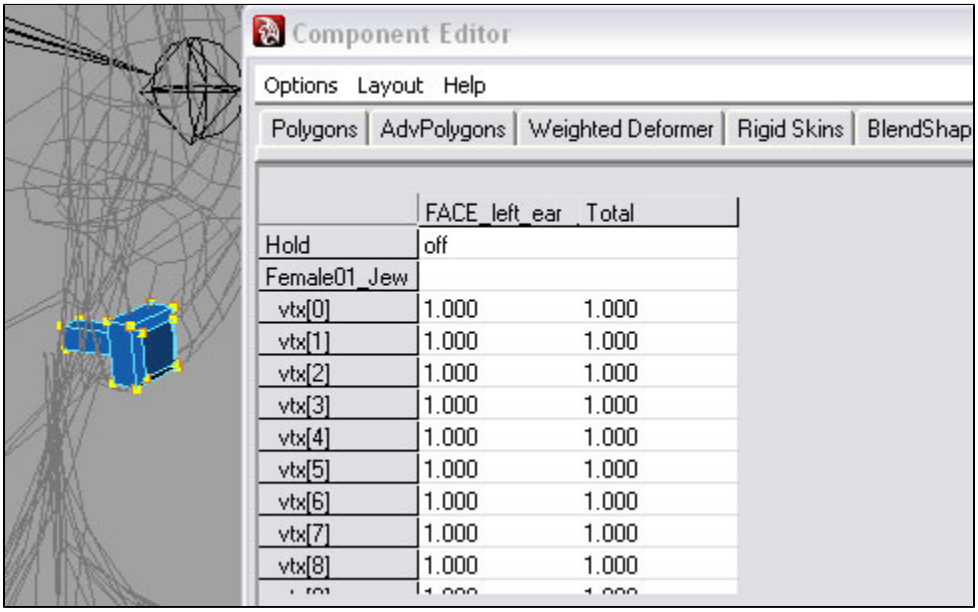


Each separate item (left or right) should be weighted to either of the following joints, with a weighted value of 1, depending on which side is being modified.

Use `FACE_left_ear_mod`, or `FACE_right_ear_mod`.

All LODs are dealt with the same way.

Skin weights can be verified or tweaked if required in the Component Editor:



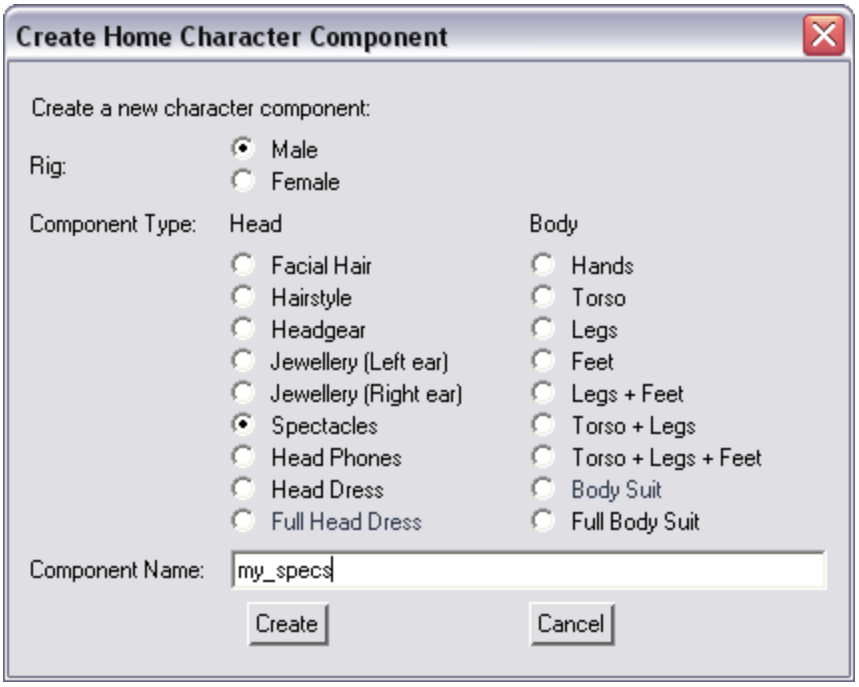
Creating Spectacles

This area covers the concept of skinning to specific bones.

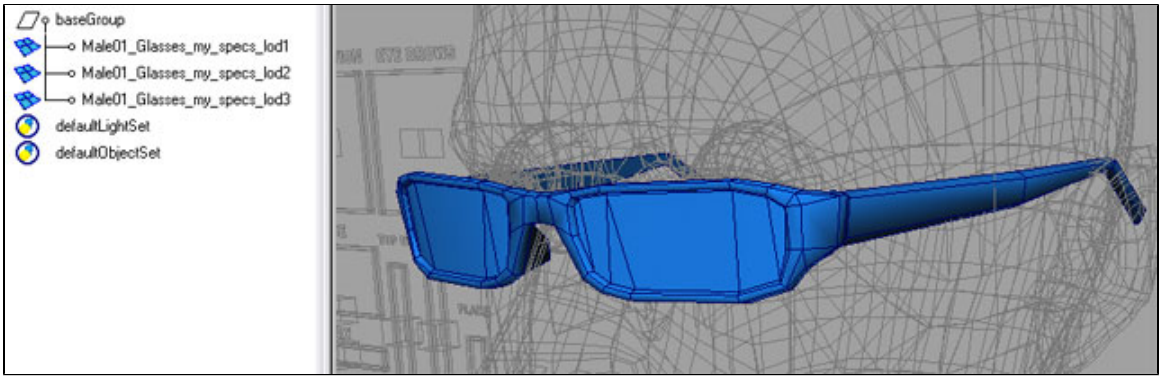
There are three categories of head and face wear: spectacles, headphones and jewelry.

To create spectacles:

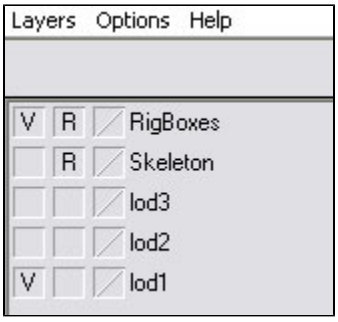
1. Open Maya and select Select File > New Home Character Component
2. Select the rig (Male or Female) for the gender type, select Spectacles, type a component name and click Create.



The rig is loaded and a default baseGroup of three spectacle LODs created with LOD1 visible:



LODs 2 and 3 are also loaded but their Layers are set to be invisible:



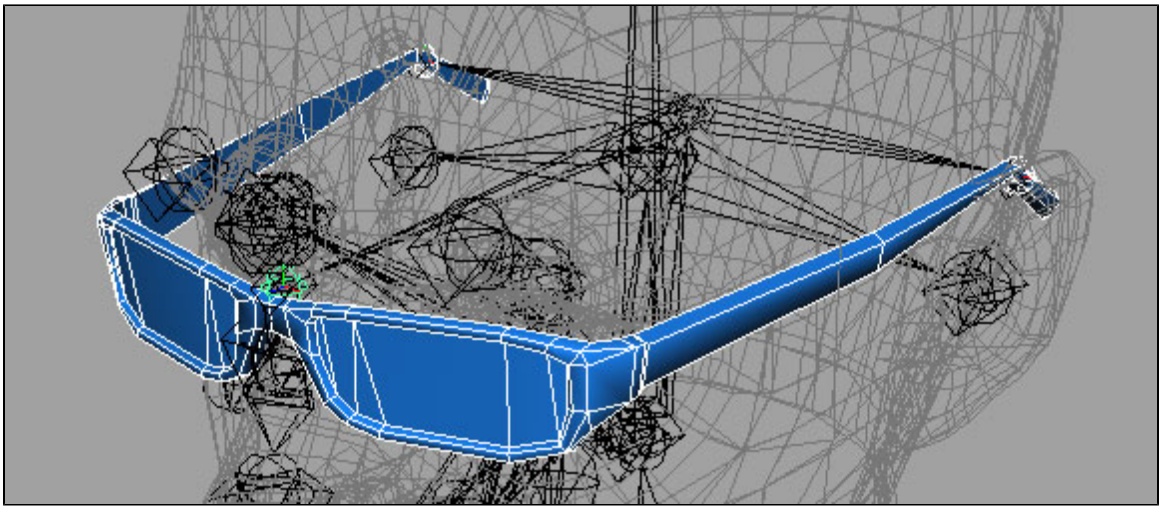
Weighting

The spectacles are skinned to bones specific to this type of face wear. There are a number of expressions driving bones, to compensate for all the face alteration. Once skinned, not only will the spectacles move and adjust to any face customization, but they will maintain a certain 'realistic' volume that is acceptable to the user.

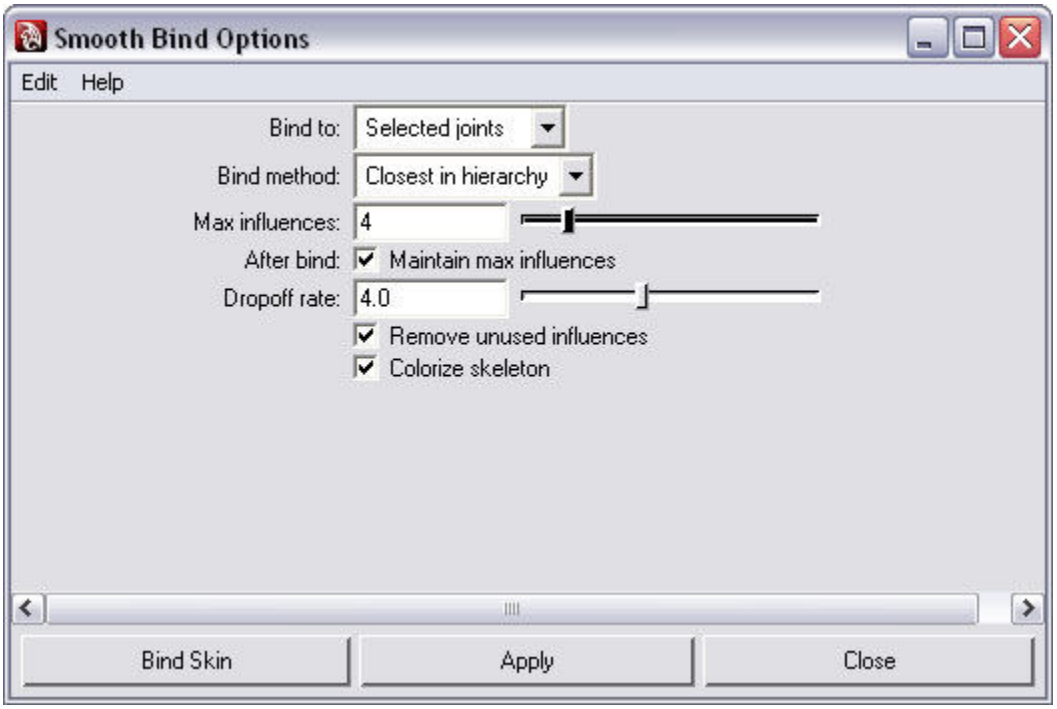
LOD1

1. Make sure the Time Slider is set to frame 0.
2. Select the mesh and the following 3 joints:

   ACCESSORY_bridge_mod, ACCESSORY_left_mod, ACCESSORY_right_mod



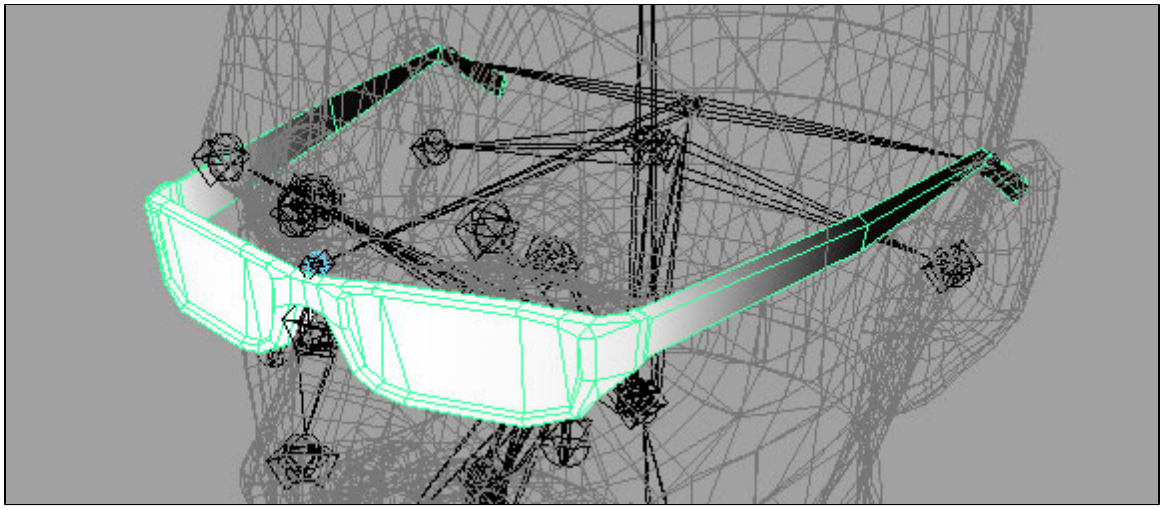
3. Before binding the skin first go to the Smooth Bind Options dialog box and make sure that `Bind to:` is set to Selected Joints and `Max influences:` is set to a value of 4.



It is important to weight the mesh correctly. The following images give guidance on how to do this:

The Main Frame and Lenses

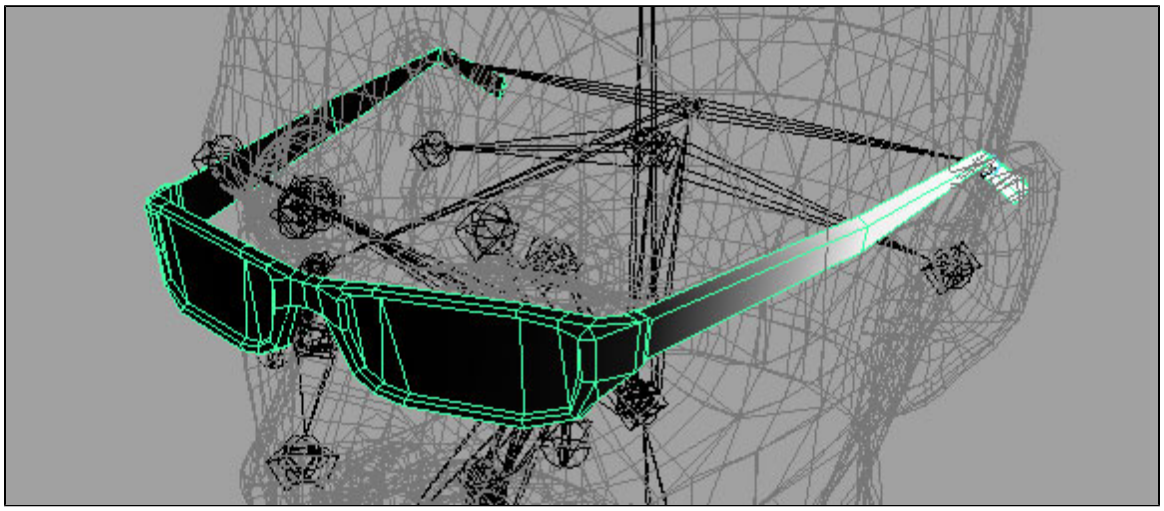
Weight this area all to bone `ACCESSORY_bridge_mod`, with a weighting value of 1.



The Frame Arms

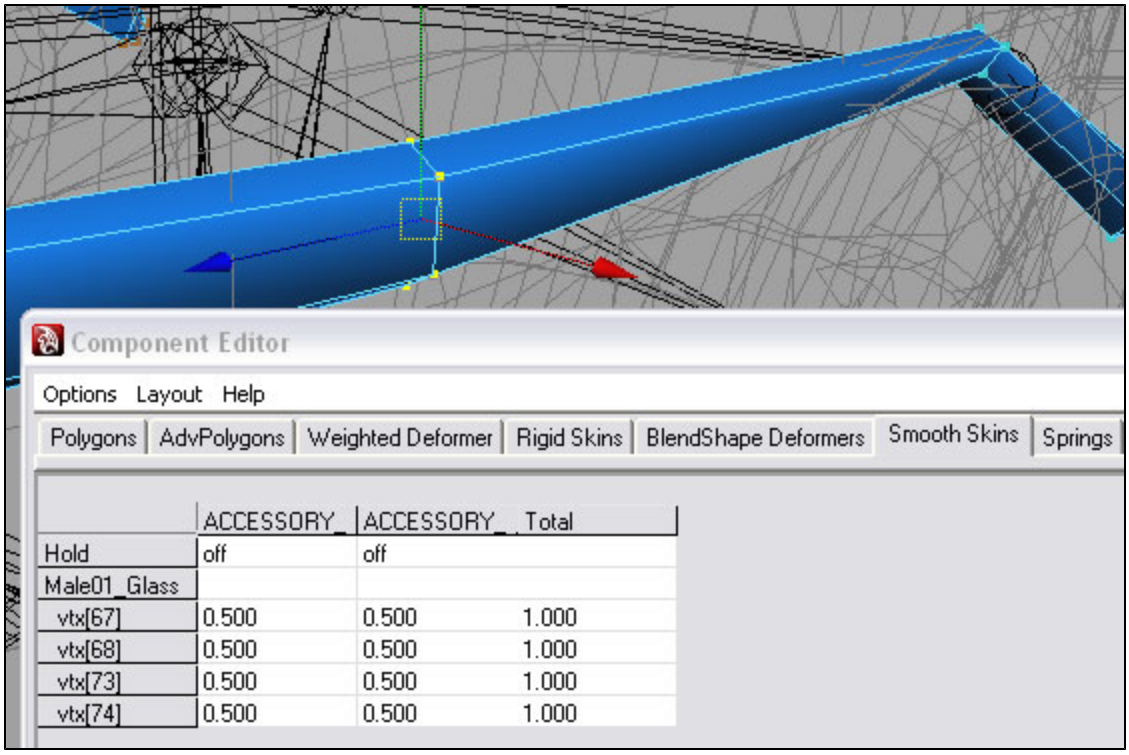
Weight this area to both ACCESSORY_left_mod and ACCESSORY_right_mod.

Use a weighting value of 1 on each side{{.}} (Left arm to the ACCESSORY_left_mod, and right arm to ACCESSORY_right_mod):



When this is complete, special attention needs to be paid to the weighted blending of the arms to the main frame. Often a value of 50/50 between the ACCESSORY_bridge_mod and either ACCESSORY_left_mod or ACCESSORY_right_mod halfway down the arm will result in satisfactory results.

A certain amount of weight value editing in the Component Editor may be required to achieve this accuracy as shown below.



This approach highlights the importance of modeling enough information in LODs 1 and 2. Vertices need to be present halfway down the arm in order for it to deform correctly.

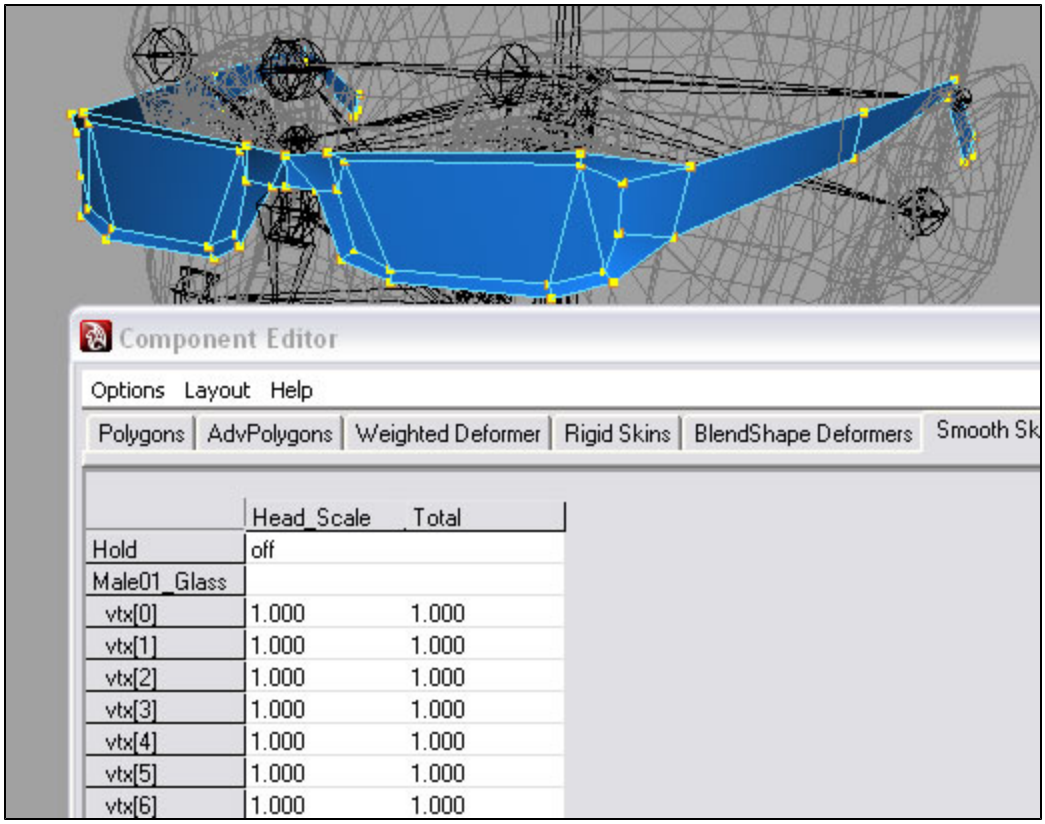
LOD2

For LOD 2 repeat the LOD1 process above.

LOD3

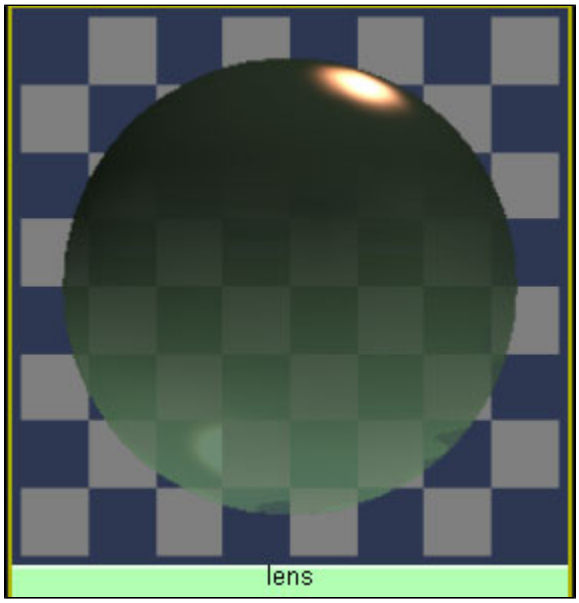
Reset the Time Slider to frame 0.

Skin the mesh to the `Head_Scale` joint. This will give it a weighting value of 1. This is all that is required for LOD3, as it will only take on the head scaling attribute. This can be verified by viewing the vertex weight values in the Component Editor:

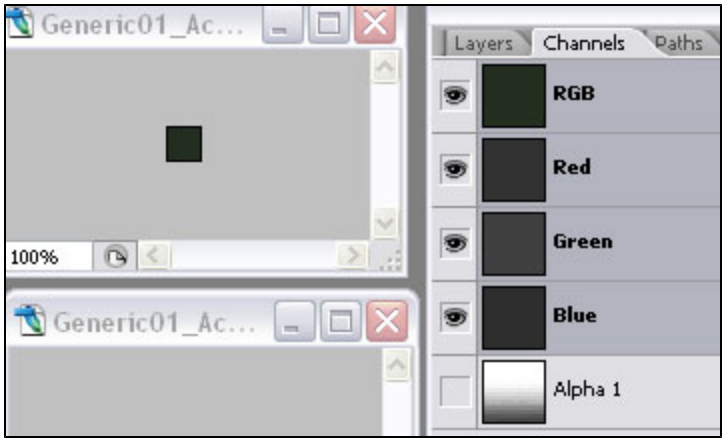


Lenses

A classic gradient sunglasses lens effect can be created in the following way using the Default AtgShader.type.

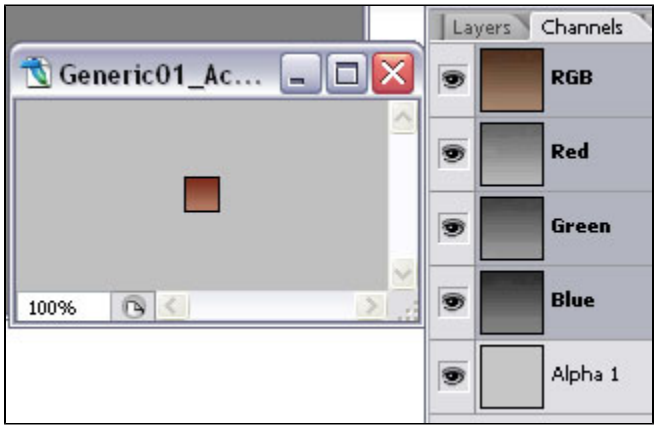


Set up the color texture map in the following way in Photoshop. Dark RGB values and a gradient in the alpha channel.

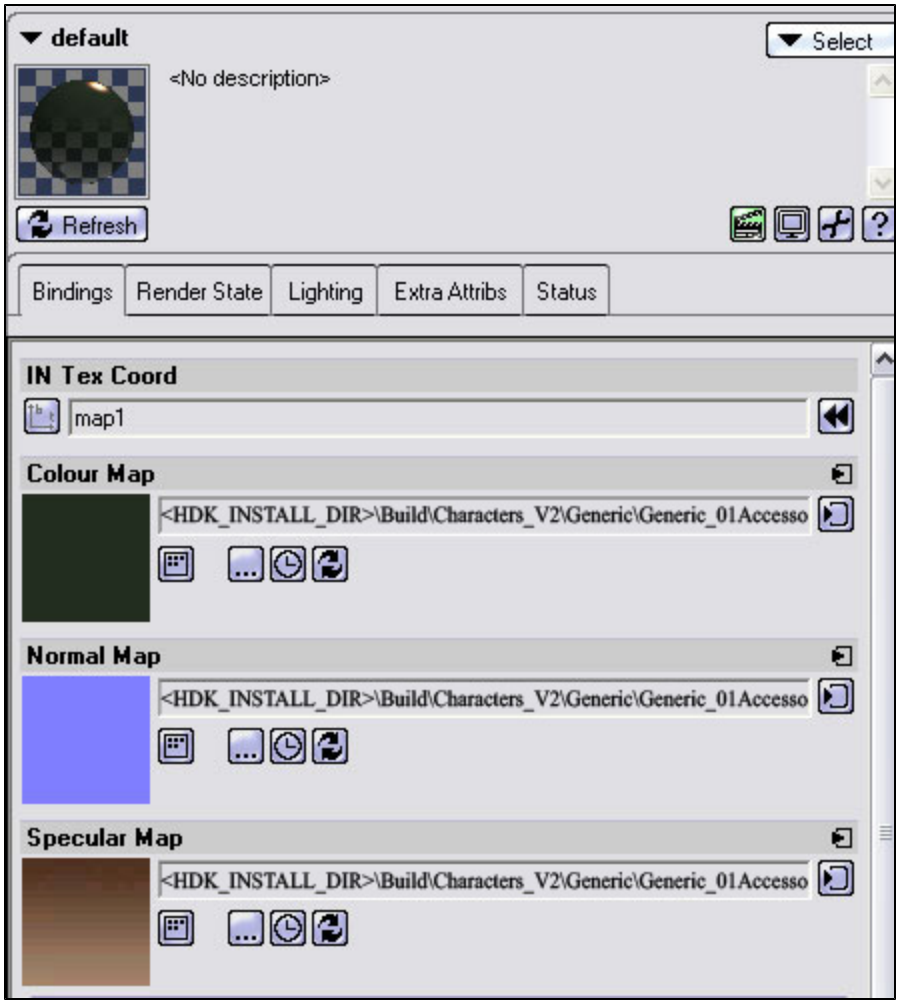


Notice that the texture does not need to be very large. 16 x 16 is ample.

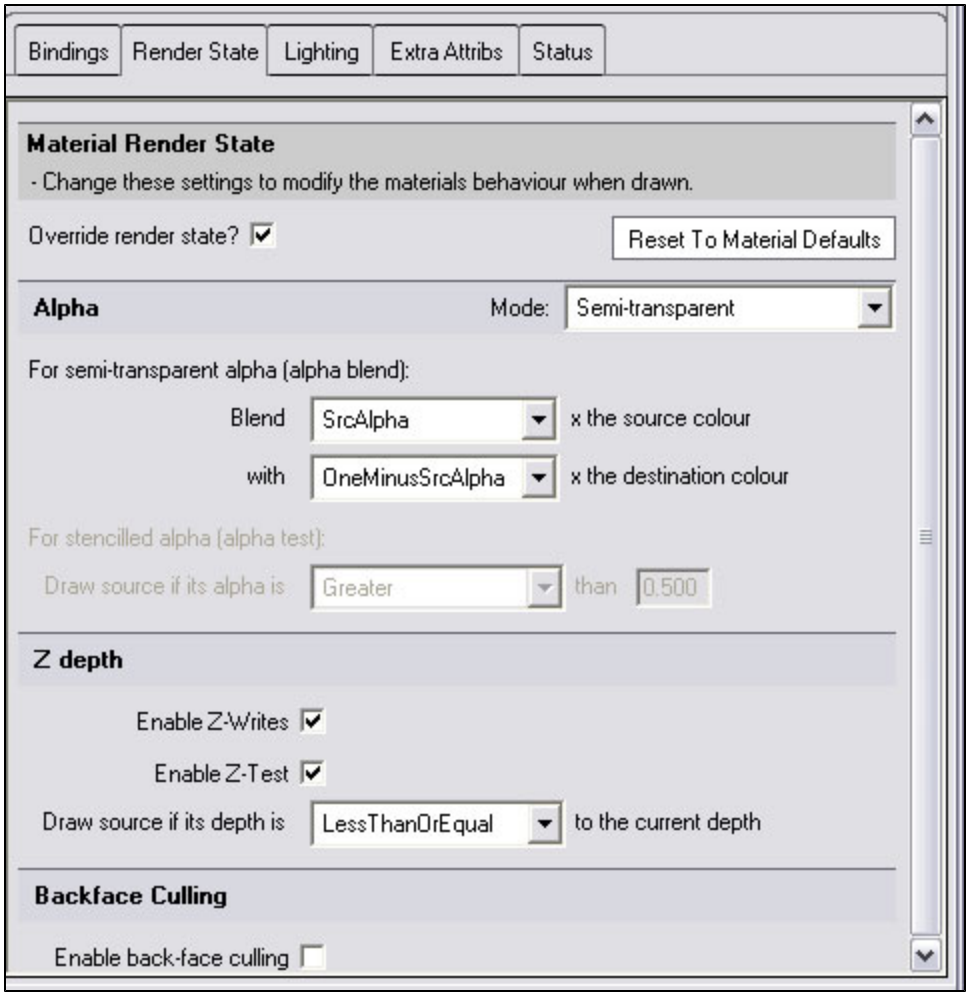
To give the specular a nice iridium tint set up a map like this:



Apply to the Shader bindings as below:



Now in the Shader's Render State settings, select the following parameters:



Creating Facial Components

The head components all currently use only the facial joints, these deal with major head deformations but do not account for the blend shape changes.

Blend shapes are used for all emotes and all the racial head types. There are about 30 on LOD1 10 on LOD2 and none on LOD3.

All internal beards and components have generated shapes that allow them to conform to the face shape. We do not currently have tools to allow the creation of these shapes for other external developers.

This simply means that facial hair components do not deform and are therefore extremely limited in quality and ability.

Technote

Avoid making close-fitting head and facial character components.

Overview

Head and facial components do not have blend shapes. Head and Facial character components are weighted to head joints only and do not deform with facial blend shapes.

Avoid making close-fitting head/facial components.

Details

Facial customization/expressions are a combination of joint deformation and blend shapes.

Head and facial components do not have blend shapes. Head and Facial character components are weighted to head joints only and do not deform with facial blend shapes.

When making head or facial components, keep in mind:
Expressions and customizations of the avatar face/head could cause clipping through your head/facial components.
Your head/facial components will not follow blend shape deformations.

Avoid making close-fitting head/facial components.

Link to this technote on SCEDev.

Building LODs

This section provides tips on making LOD assets around important vertices (see Building LOD 1) and explores how to model clothing assets, including modeling for thin and fat avatars (see Building LODs 2 and 3).

See also:

- Saving a Character Component for tips on what to do before you save and package your scene.
- Previewing Character Components in PS Home and The Home Client for a list of debug console commands that are useful for reviewing your assets in the client.

Building LOD 1

The clothing assets should be modeled around the standard male or female naked templates and have a base mesh which adheres to the triangle counts provided.

To make the shoulders rounder and more natural-looking, build them lower than the templates.

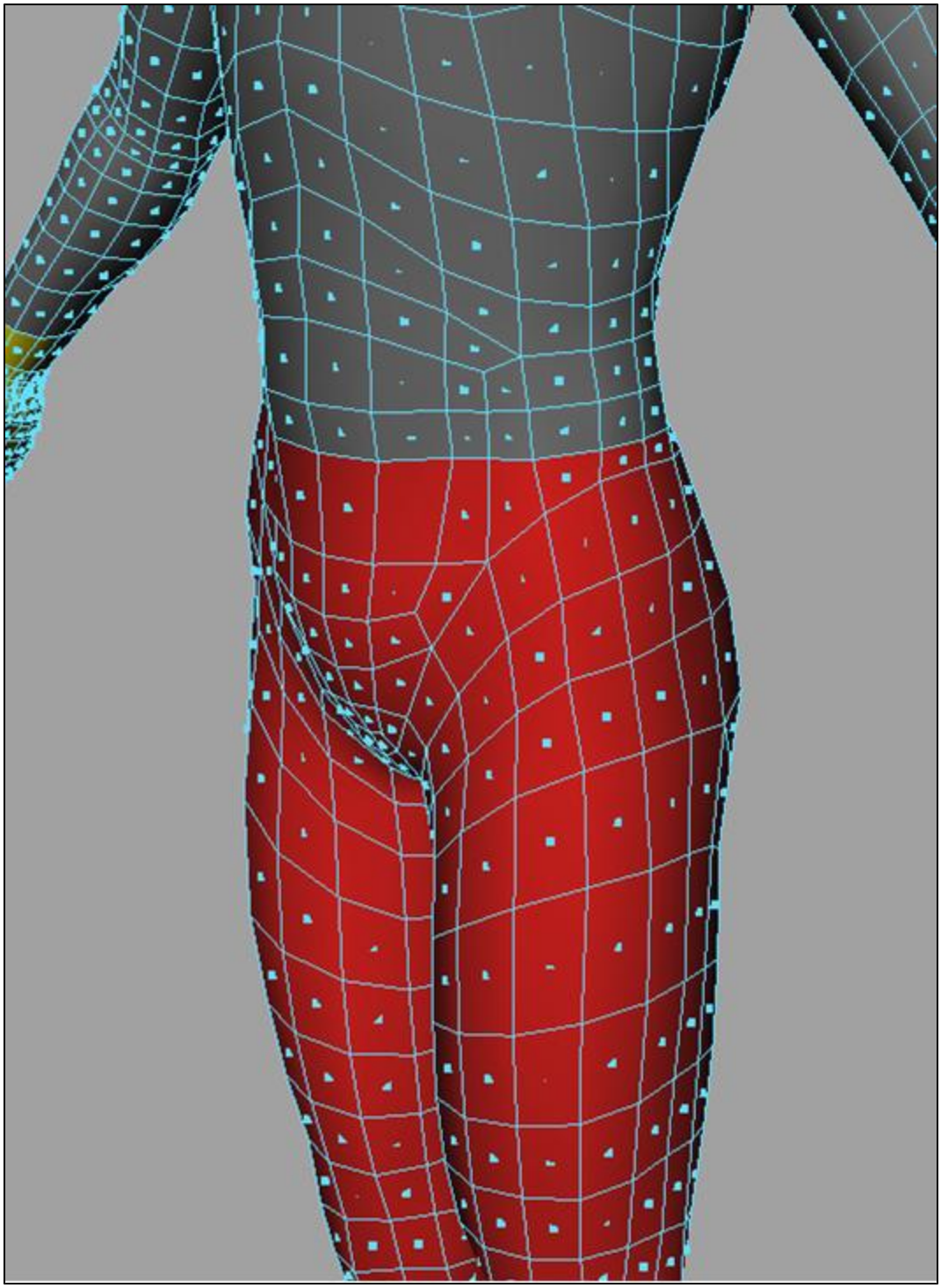
Holes in Waist

Reference the MALE01_ and FEMALE01_WRAP_DEFORMER.ma files in the <HDK_INSTALL_DIR>\artsource\template folder.

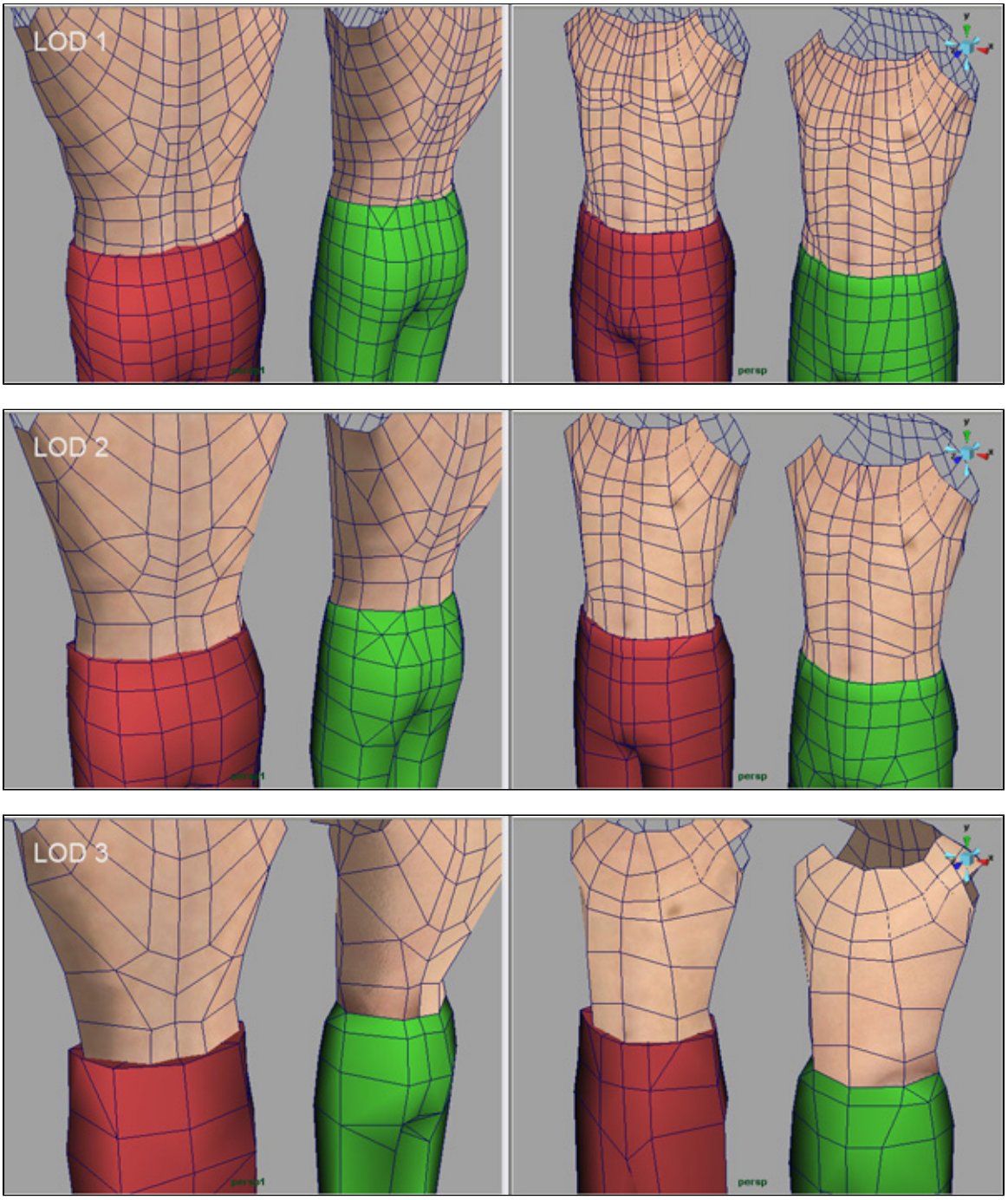
Reference the MALE01_ and FEMALE01_WRAP_DEFORMER_POLY.max files in the <HDK_INSTALL_DIR>\artsource\template\Max folder.

Load either a male or female rig into your scene, then build and snap the same amount of waistline points on all LODs as the templates. This prevents large holes appearing in game, especially on lower LODs.

The following image shows the LOD1 MALE01_WRAP_DEFORMER.ma, with a smooth wrap along the waist:



The following images show a leg asset, with good seam layouts in green and bad layouts in red. The green layouts conform to the shape of the body well in each LOD. The red layouts are more crude, and do not smoothly conform to the body shape.



The following image shows the male topology template LODs for waists:



The following image shows the female topology template LODs for waists:



Using this approach, developers gain the following advantages:

- Less intersecting and reduced bug fixing
- More styling options
- A greater variety of clothing
- Broader compatibility with clothing items

Pelvis Intersection

You should mirror the template topology guide scenes as much as possible because it is difficult to avoid intersections around the pelvis when torsos and legs overlap. This is because their topology is usually different or offset. Also, because many studios create clothing items, if you use the generic templates as guides when modeling new assets, the character in PS Home will be more robust.

The Template Topology Guide scenes are in the following directories:

- The `FEMALE01_TEMPLATE_TOPOLOGY_GUIDE.ma` is in `HOME_DIR\artsource\Characters_V2\Female\Female01\Geometry\Female template`
- The `MALE01_TEMPLATE_TOPOLOGY_GUIDE.ma` is in `HOME_DIR\artsource\Characters_V2\Male\Male01\Geometry\Female template`

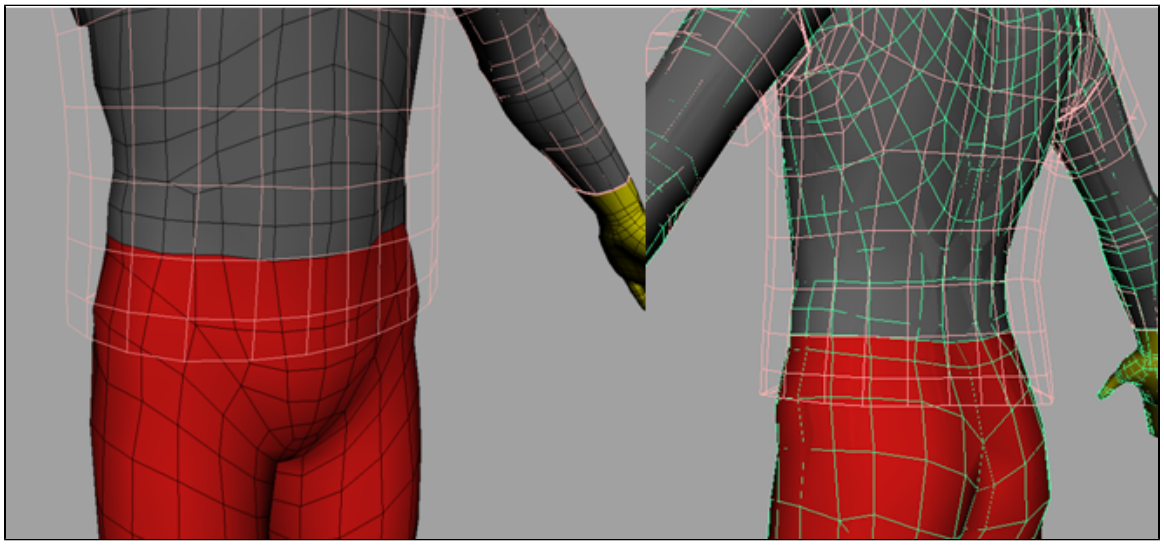
Make sure that you mirror the following:

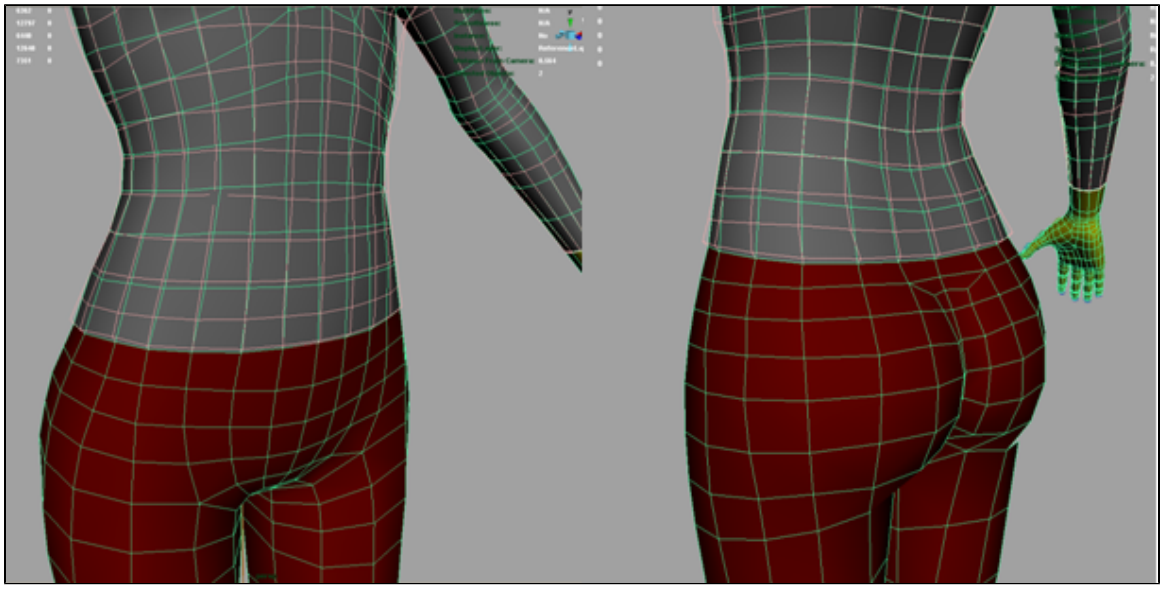
- The pelvis topology from the templates onto the pelvis area of all leg assets

- The pelvis area of the torso assets that drop below the waist seam (such as long t-shirts, untucked shirts, and three quarter length jackets)

The following examples shows how male and female topology guides have been used to build a long torso component when you select File > New Home Character Component > Torso.

With the female torso, the pink lines are difficult to see in the image, but are more clearly visible in Maya itself.

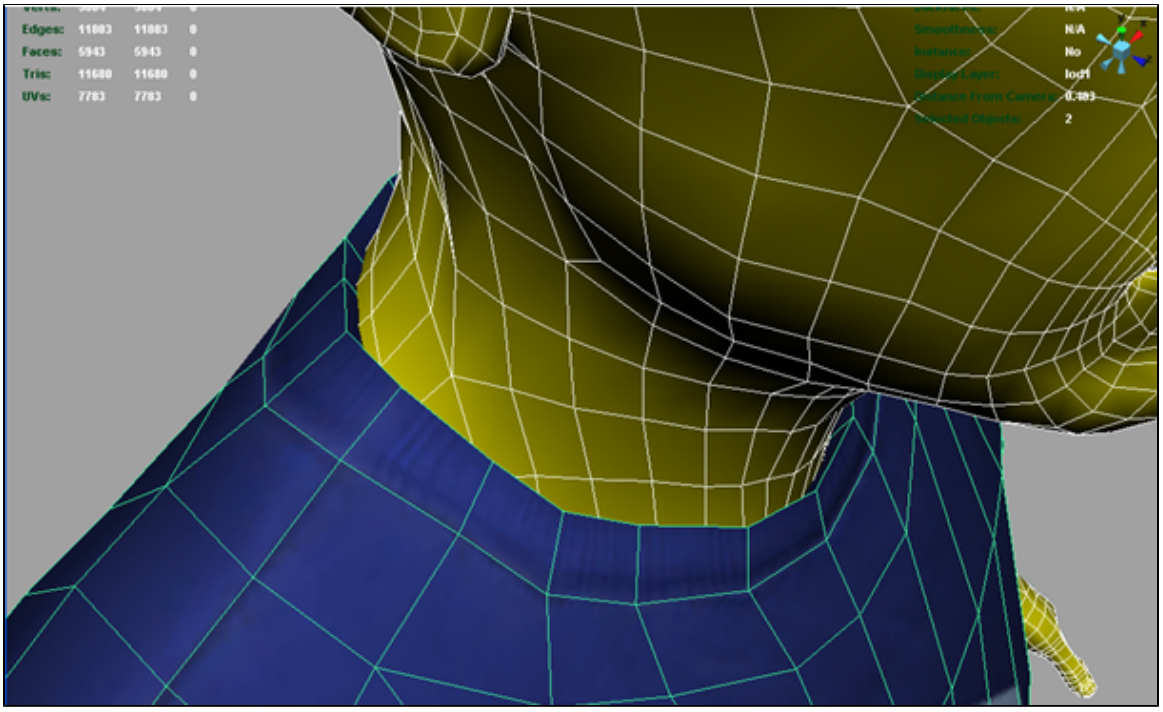


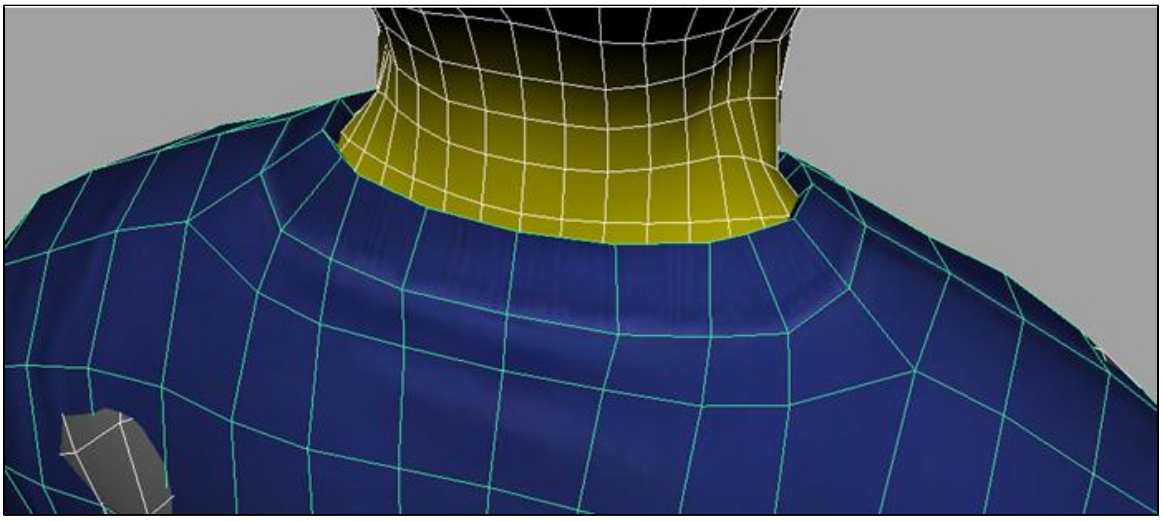


Neck Intersection

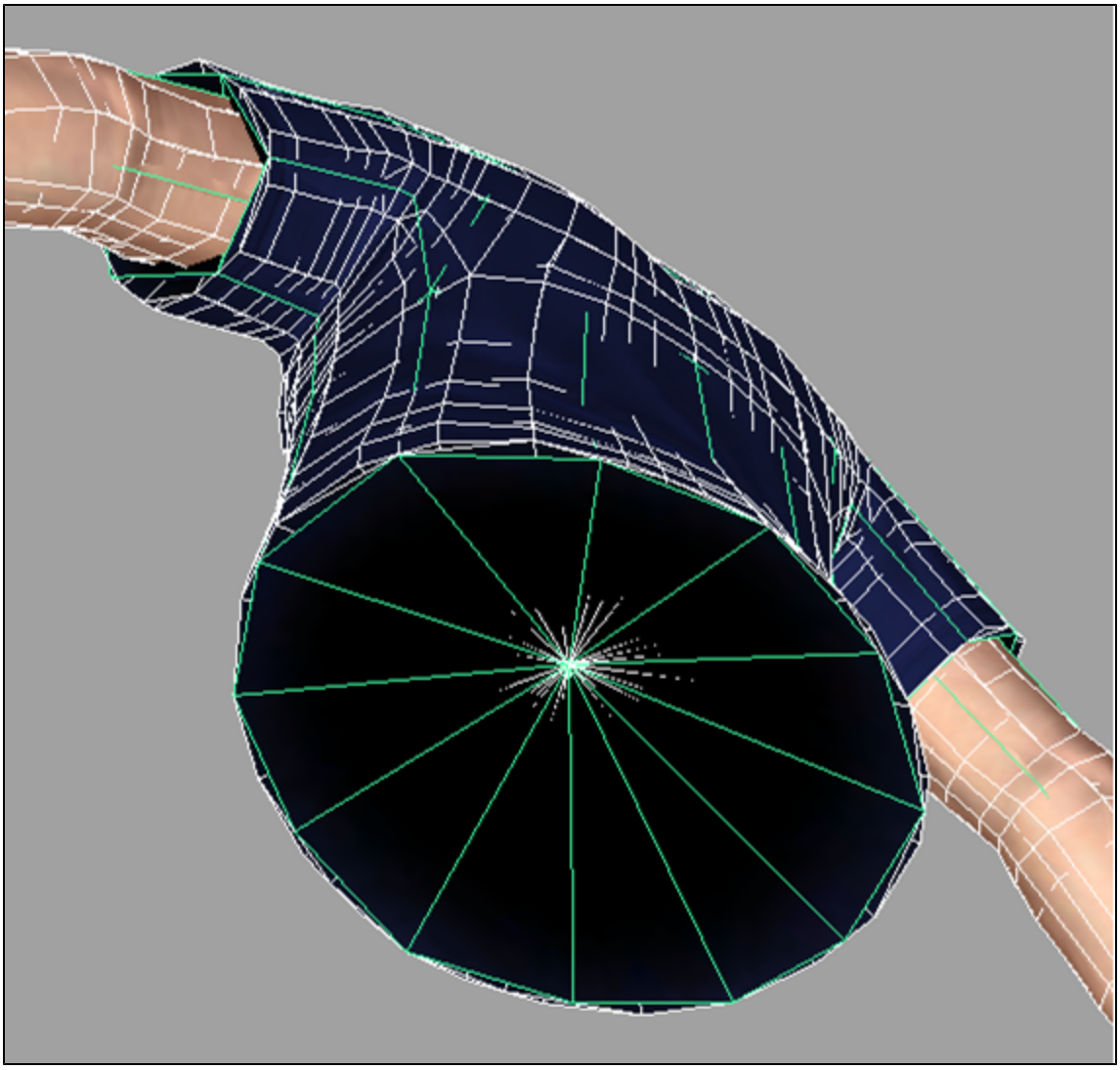
Sometimes the head mesh intersects through clothing around neck lines, due to hidden geometry under an asset. This Head geometry is fixed in PS Home. You cannot modify or delete it.

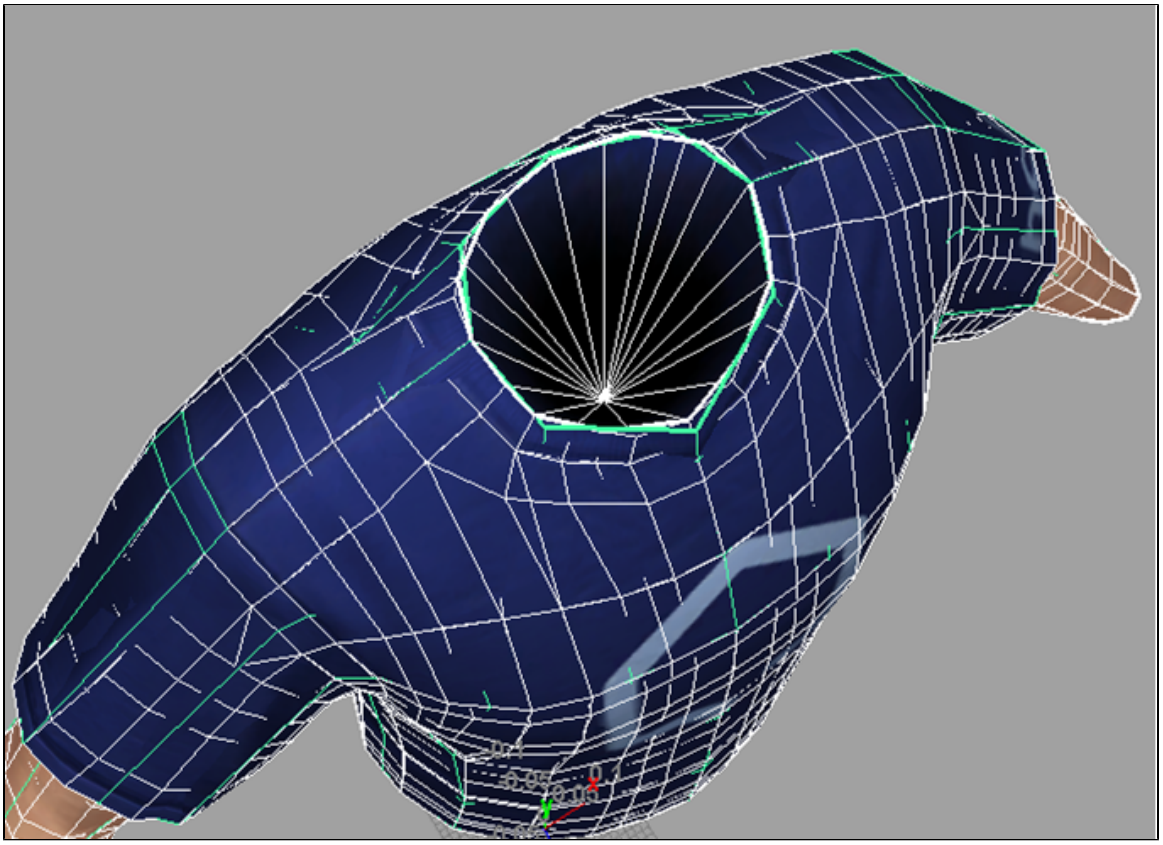
The following images show that the asset's neck area has been built to mirror the head as best as possible, to help avoid intersecting in certain animations. To view it, select File > New Home Character Component > Torso.





The following images show open edges such as necks, cuffs, waists, and trouser legs.





Unwrapping UVs

When creating assets, bear in mind the number of UV shells used on a mesh, because multiple shells require duplicates of vertex data. This data, combined with the duplicate information required for blend shapes, can double or triple the size of a model. Also, the more UV borders you have, the more difficult it is to collapse your base mesh down when creating LODs.

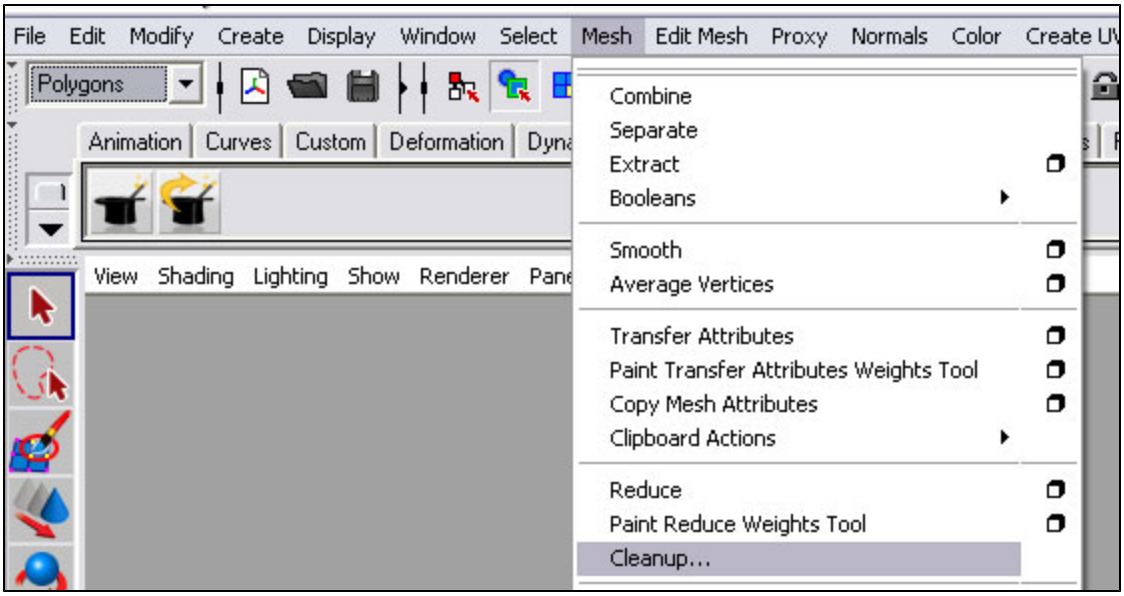
Sculpting Detail

We recommend that you use ZBrush or Mudbox to create normal maps. However, as long as you end up with a tangent space map, use whichever method you prefer.

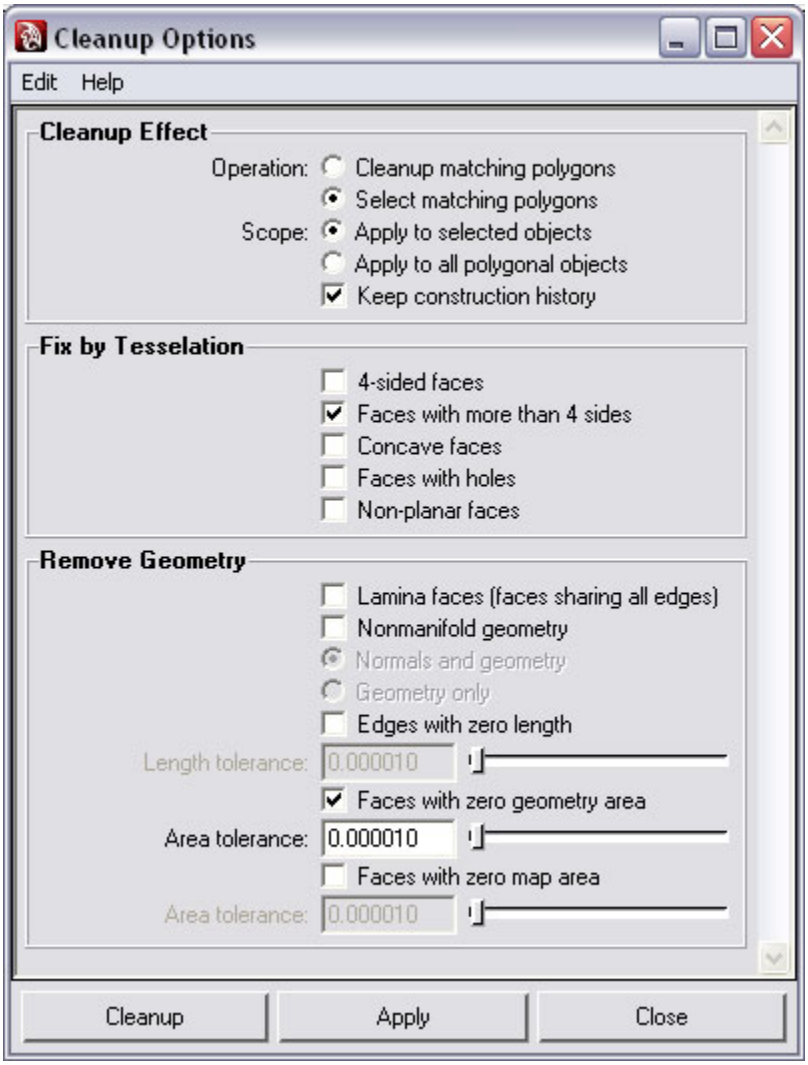
Building LODs 2 and 3

Reference the Topology Guides into your scene and mirror the pelvis topology from them into your LODs 2 and 3. When creating LODs, remove edges carefully from the stomach area on torsos, because the fat shape fat shape will not hold up later on. Also, offset the thin blend shape LODs on X by -0.5m, and the fat blend shapes by 0.5m.

Delete all history when you have finished the LODs. You should also make a poly cleanup before skinning and creating fat and thin blend shapes, to isolate any N-sided polygons and stray vertices. You can find the cleanup options under Polygons > Mesh > Cleanup….



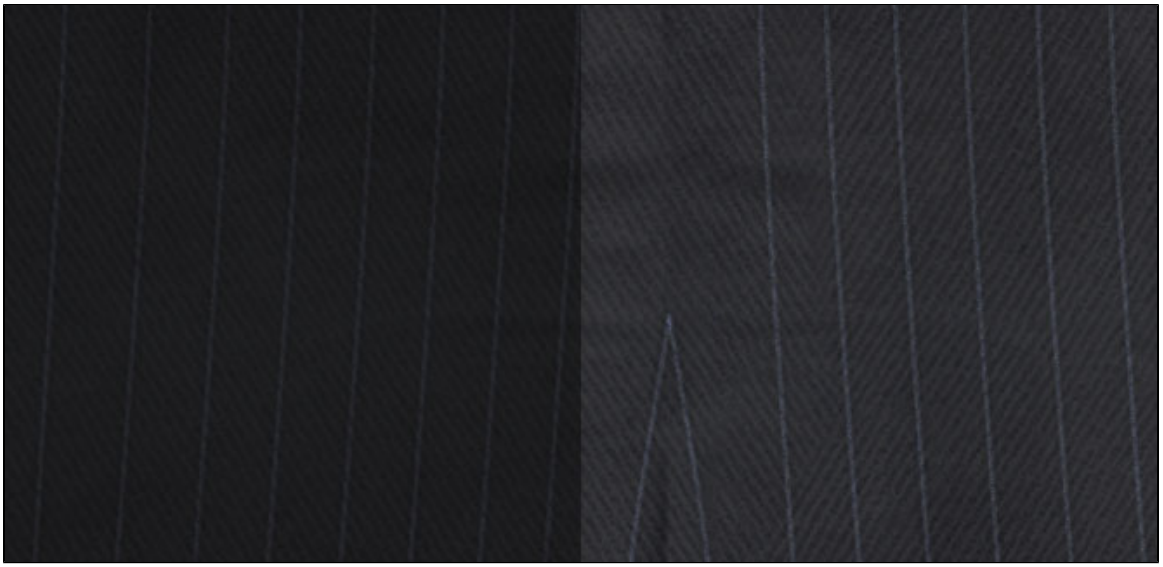
Set the options as shown below:



Use the Edit Vertex Normals tool on any skin seams  .

Texture

Black textures often appear darker in-world due to the in-world lighting. For clothing to appear similar to the sample swatch shown in the image on the left, make black textures more charcoal in appearance (similar to the texture on the right). Allow the shadow to give an overall darkening effect.



ATG Shaders

Avoid using texture maps with small areas of specular, such as buttons. Although you lose the shiny effect, you save valuable VRAM memory. Use also the specular placeholder DDS file on all soft cloth materials, unless a specular effect is crucial to the characteristics of the item.

Creating Fat and Thin Shapes

Modify manually fat and thin shapes, because the wrap deformer can distort them too much. Texture stretching can become very noticeable if pockets, zips, button details and open edges of coats break.

See also Creating Fat and Thin Targets.

Common Issues and Solutions with Fat Shapes

Pay particular attention to the chest, backside, midriff and stomach on fat shapes, as they deform significantly. The following examples illustrate common issues and solutions for the female and male fat shapes.

Female Fat Shape

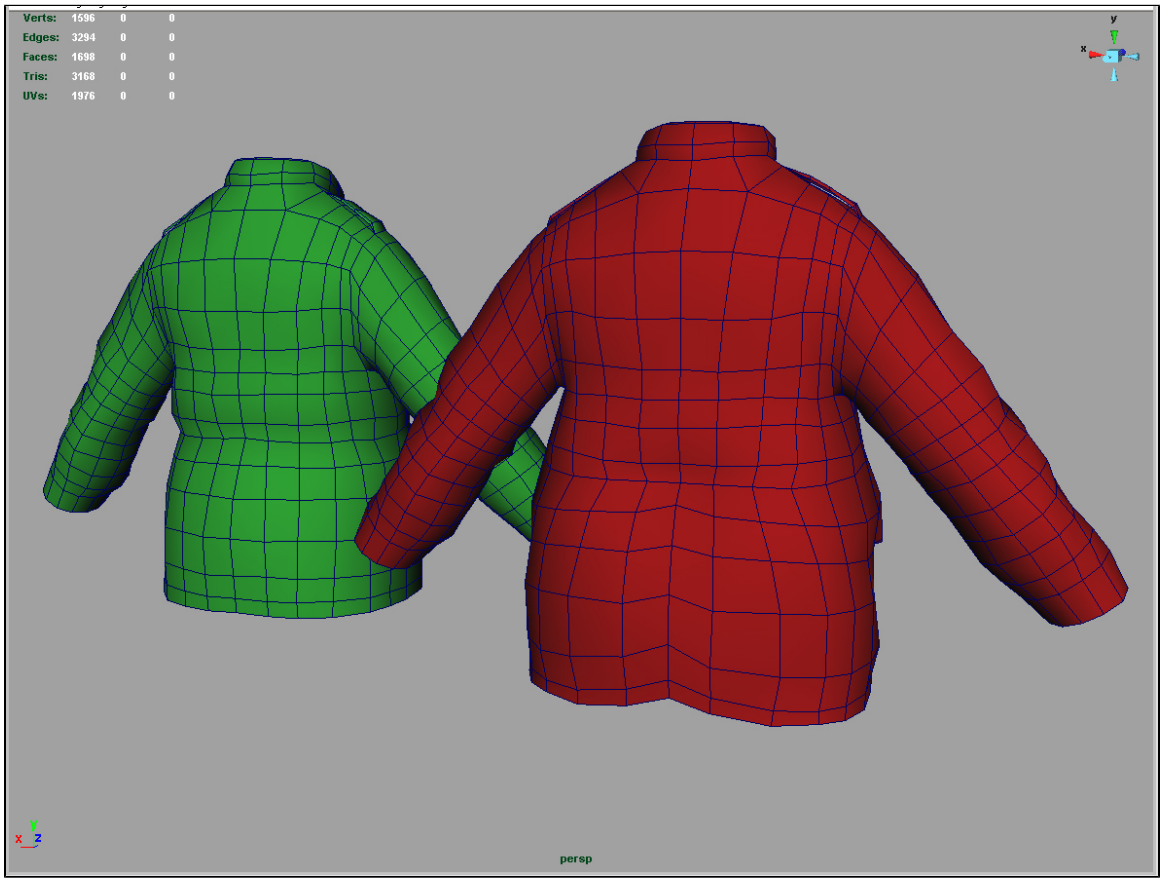
Lower and pull forward any cloth mesh between the breasts, to make the material look more natural. Vertices tend to become unevenly dispersed around the stomach area, resulting in fewer at the front and bunching on the sides. Maintain roundness around the stomach area by moving vertices towards the front. Vertices can become unevenly dispersed around that area, resulting in fewer vertices at the front, and bunching on the sides.

Male Fat Shape

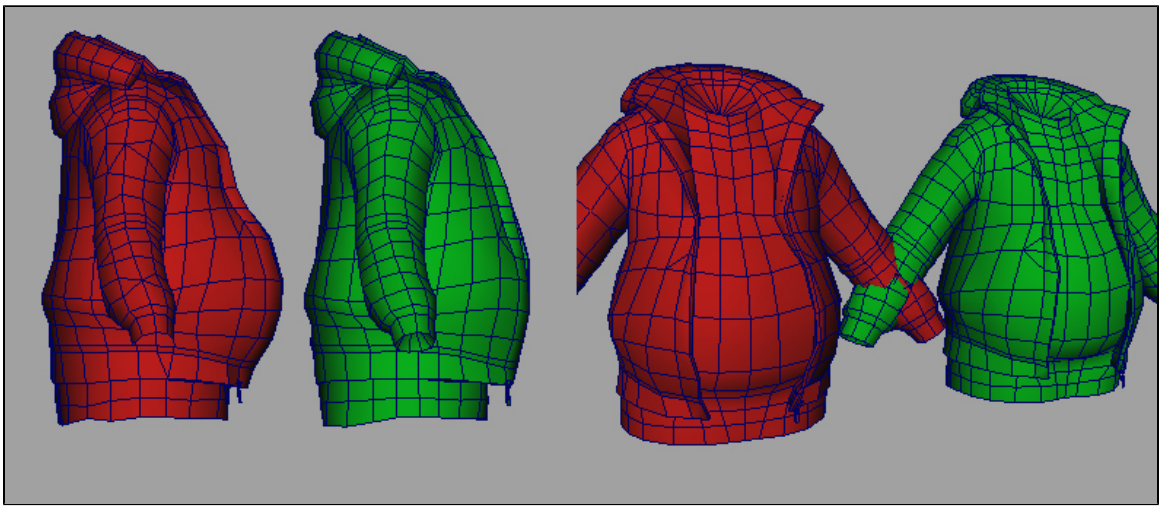
Pull out the chest area of the male fat shape, disperse more evenly the stomach vertices, and manipulate the lower back vertices. The red example shows how the pockets have initially been distorted. The green mesh has had the top and bottom pockets reshaped, the chest/zip area has been pulled out, and edges have been pulled back round so they flow evenly.

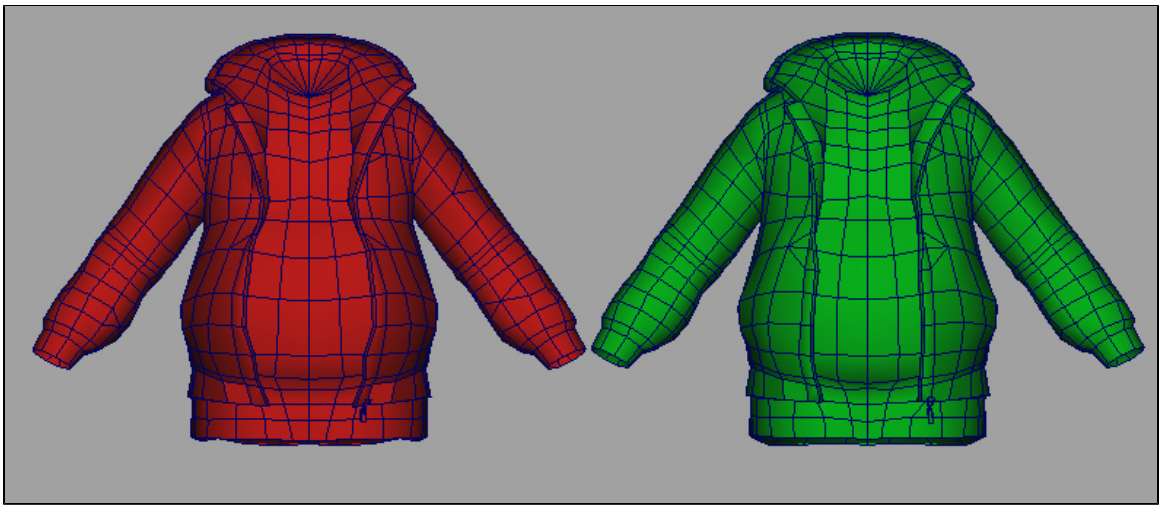
Sometimes it is faster to use a Lattice Deformer to create your fat shape.





Be aware of open jackets and how the deformer can change the initial structure. The red mesh below shows a jacket before correction and the green mesh shows how the artist has fixed the open edge.





Thin Shapes

The shoulder blades tend to distort too far out towards the upper arm. The Female chest may need volume and shape put back in.

Skin

Because Home supports skin weighting of 4 bones per vertex, skin your character assets accordingly in Maya. When you have skinned LOD1 correctly, you can use the Copy Skin Weights tool  to quickly skin LODs 2 and 3. Double-check that the wrist and neck seams have no holes.

Skinning Test

You need to do a thorough skinning test on your asset. This testing eliminates problems further down the line and reduces the amount of QA fails and added time communicating and fixing errors.

Reference items with matching test curve in your scene to test the compatibility between legs and torsos.

We generally reference a Baggy Jeans asset, as this asset has the most bulk around the waist and causes the most amount of intersecting. This means you must test 9 different meshes, but is an important part of the workflow.

Creating Fat and Thin Targets

Preparing Components

After you have modeled and textured the components, and created the required Level of Detail meshes (LODs), create the fat and thin blend shapes and perform the skin weighting using the Home Component Creation tools. All character clothing components in PS Home must have fat and thin versions of all the LODs. This ensures that users can specify how fat or thin they want their

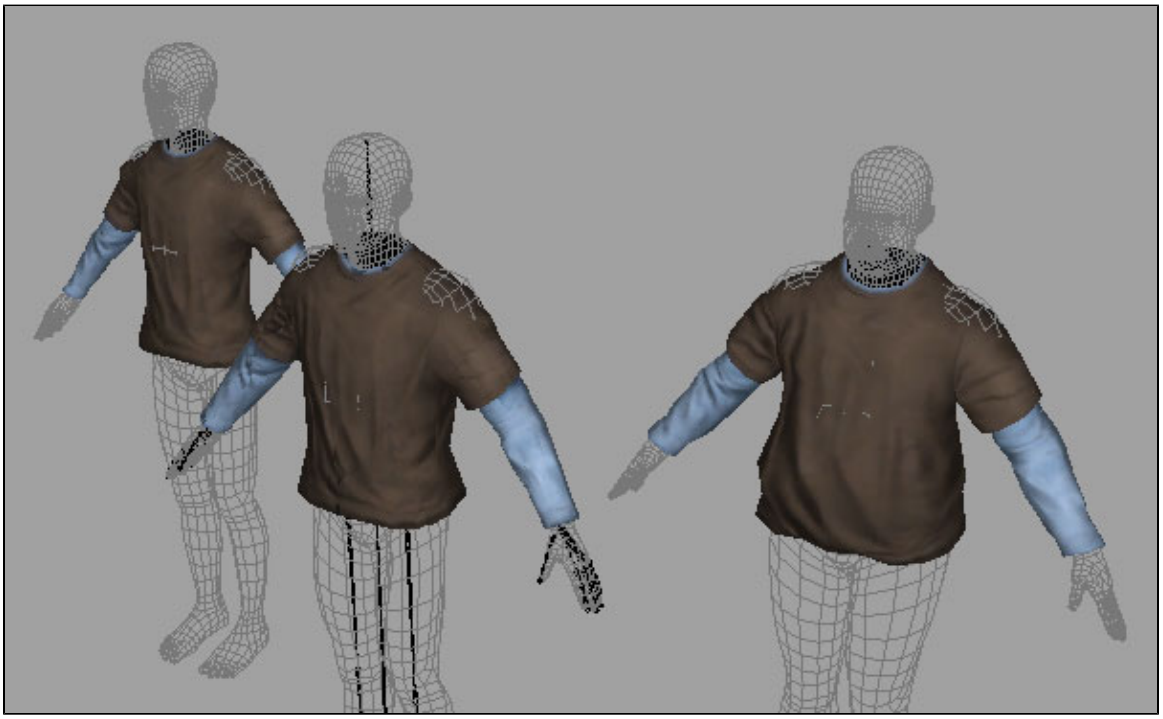
characters to be when they dress them. PS Home provides character component tools to enable you to do this.

The first step is to create fat and thin variations of LOD1.

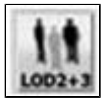
Creating Fat and Thin Variations of Components

1. From the Home_Char Tool Shelf, click the Create blend shapes (LOD 1) button.
   This is the basic (default) mesh. The process initiates the process of duplicating the component and then wrapping it onto the fat and thin template meshes.
2. Specify the LOD for Thin and Fat variations of this mesh by clicking 

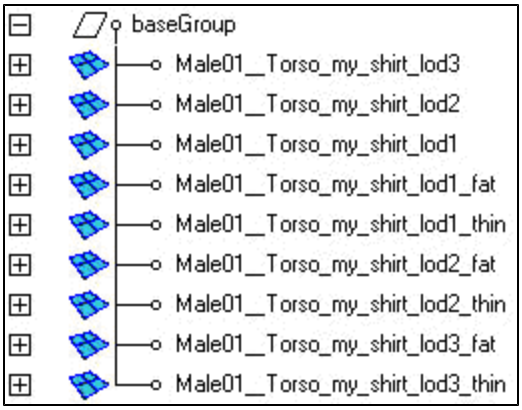
to set up LOD1 fat and thin variations for your mesh.
   This creates two variations (one Fat and one Thin) of your basic LOD1 mesh. These variations provide the weight limits that Home supports, that is, the fattest or thinnest your character component can get.
   You need only edit LOD 1 because LOD 2 and LOD 3 inherit their settings from the LOD 1 mesh and this allows you to edit the way the clothing hangs off the body.





3. Edit the newly created Fat and Thin meshes. You may need to tweak the vertex positions of the newly created Fat and Thin meshes. The amount of editing depends on the flow of garment. A baggy jumper will need additional vertex tweaks regarding how the garment hangs from the body, whereas a tight t-shirt will need less work than to describe baggy or bulky forms. The areas to be aware of are female chest, lower back, waist, posterior and arm pits. These areas will likely need the most tweaking as outlined in the Workflow section.
4. Go back to the basic mesh (in the center of the picture). Ensure LODs 2 and 3 basic meshes match LOD1.
5. Use 

to set up LOD2 and LOD3 fat and thin variations for your mesh.



Manual Modifications to Fat/Thin Shapes

When making blend shapes, using the automatic Fat/Thin creation tool will often not be sufficient and further manual modifications are necessary.

The Fat/Thin creation tool uses a wrap deformer that makes shapes based on the naked body.

This can distort the clothing asset and make it look too tight, stretch details such as pockets, zips, and buttons or mold a loosely fitted item (e.g. an open jacket) onto the body instead of hanging off the character.

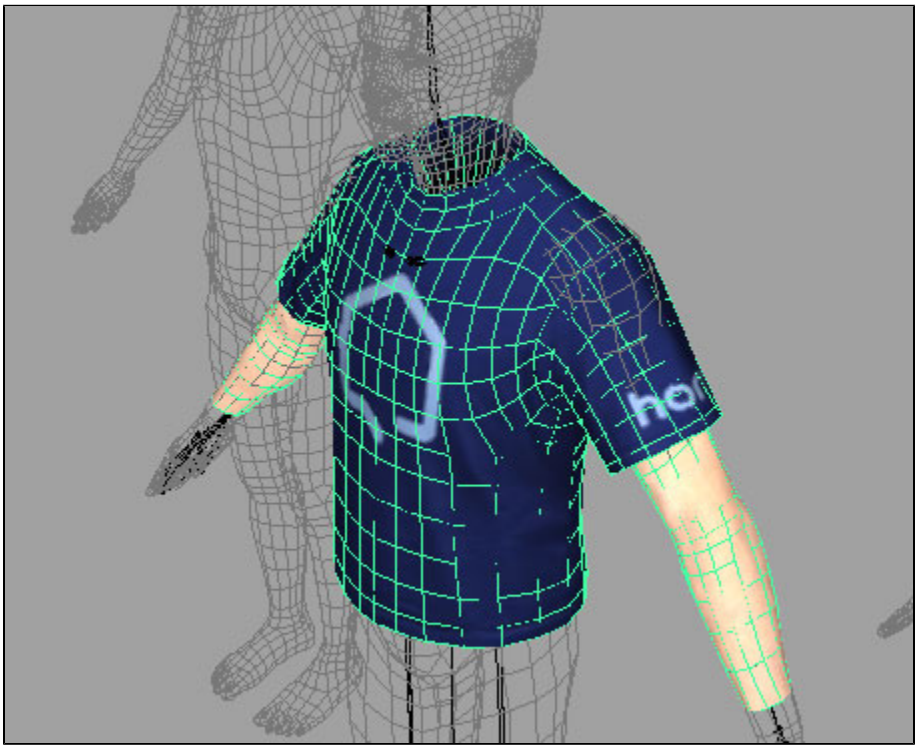
See Building LODs for more information.

Applying Skinning and Weighting

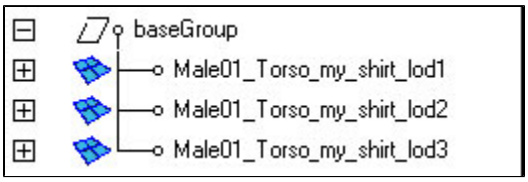
> Custom weighting is likely to be necessary on all garments, e.g. tweaking an average piece of clothing, for instance a t-shirt, to make it appear to hang loosely on thin targets and stretch on fat targets.

Skin Weighting Example

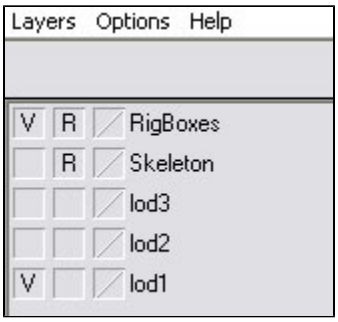
We explore how to weight and skin components by continuing with the example of creating a shirt (mentioned previously), made by starting from the Male rig and Torso component type. This template will have loaded the Home Character rig with the default PS Home T-shirt torso ready selected and framed in the Perspective Panel.



In the Outliner there is a group called baseGroup. Expanding this reveals that the required three LOD meshes have been imported bearing the name Male01_Torso_your chosen name_lod1, Male01_Torso_your chosen name_lod2 and Male01_Torso_your chosen name_lod3, where LOD1 is the highest resolution mesh.



LODs 2 and 3 are hidden by default in Maya's Layers panel.



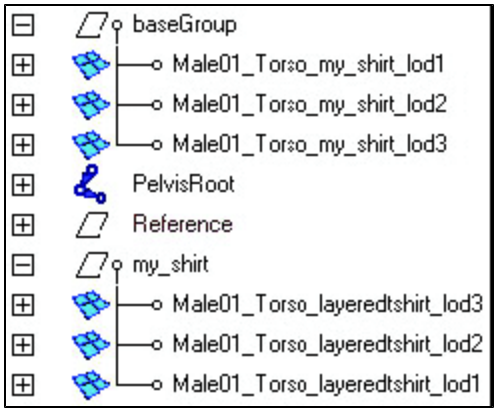
The LOD1 base asset can now be edited in one of three possible ways.

- The simplest is to replace the color texture map with a customized version, leaving the geometry unaltered.
- The second could be to edit the provided geometry and texture to create a unique customized component mesh.
- The third way would be to delete the default meshes (including the two other LODs) completely and replace them with a custom made clothing component. Custom clothing components should adhere to the modeling and texturing recommendations provided in the documentation.

> When replacing default meshes with custom meshes, give the exact names of the default objects to the customized replacements.

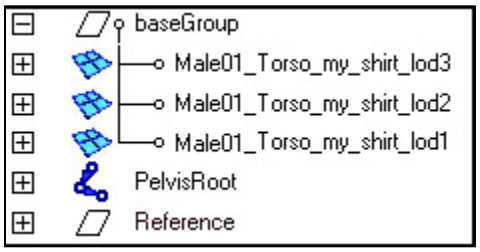
In this example the artist imports a bespoke character component into the scene along with its LODs.





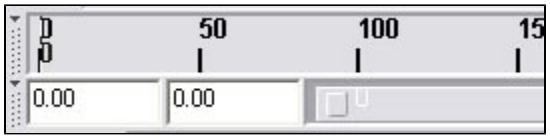
If you are importing your unique modeled asset into a scene to replace the default asset that is shipped, make sure the imported scene is clean (this includes no Display Layers) and only the unique assets are presented.

The original torso objects are then deleted and the imported equivalents moved into the baseGroup, transformed into the correct place and given the names of the original components they have replaced, as shown below:





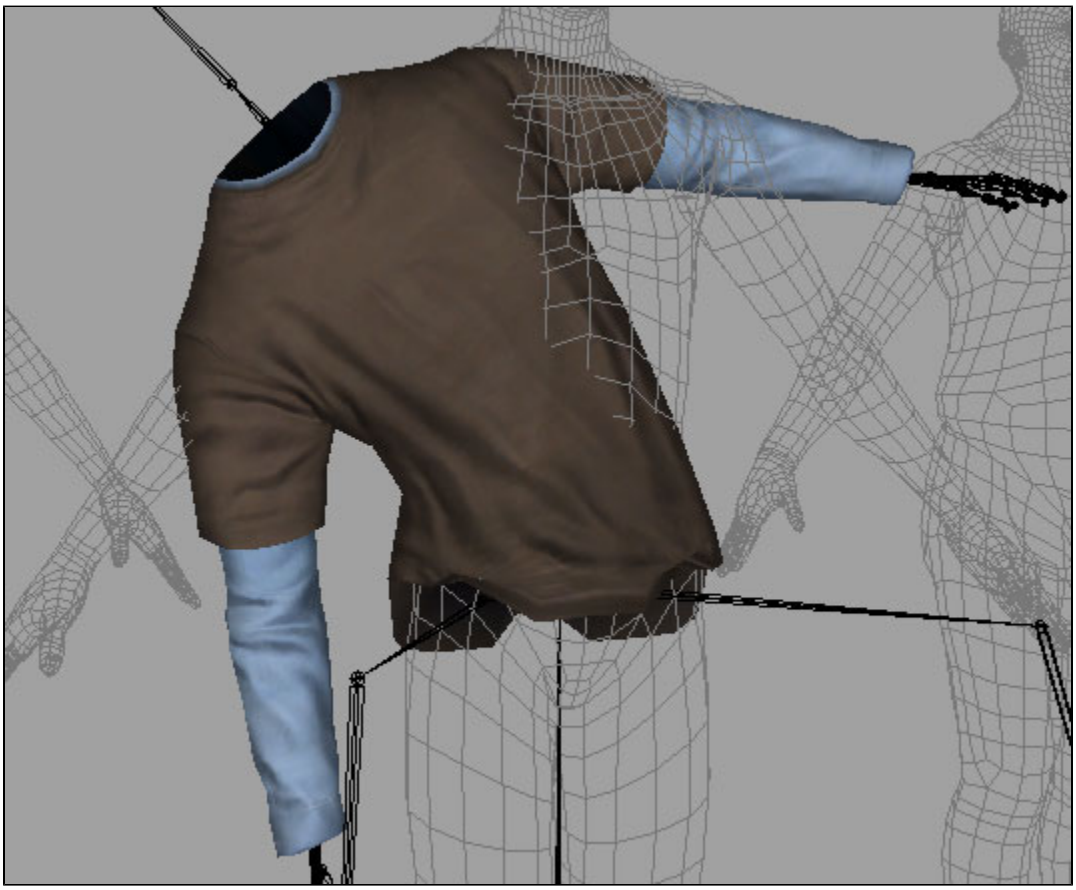
Having completed these steps, proceed with applying skin weighting. To do this first make sure the Time Slider is set to frame 0:



Clicking 

on the Home Toolbar will apply PS Home's custom weight setup to the LOD1 mesh.

Now when the Time Slider is moved back and forth the custom component will deform to the test animation of the Home character skeleton.

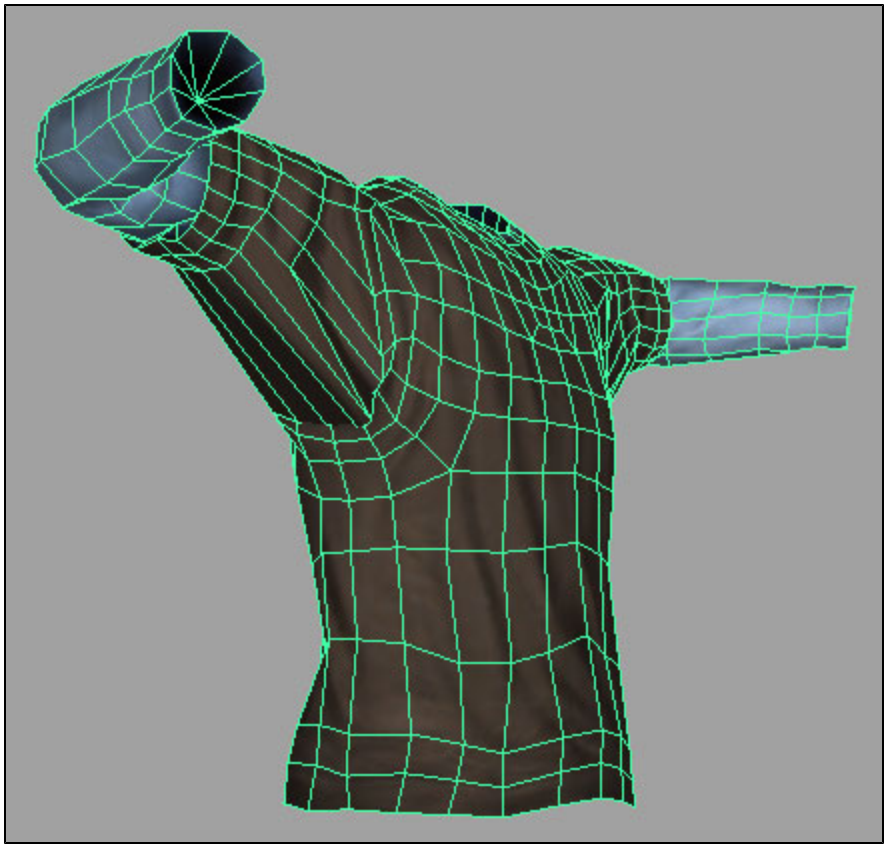


Tweaking Vertex Weights

The skin weighting should map reasonably well onto the mesh, but depending on the topology of the object it is likely to require some editing.

The idea of the test animation is to help the artist identify problem areas. In total it is 750 frames in length and systematically runs through all the major articulations a Home avatar is likely to encounter.

The main area of interest, as far as torso components are concerned, is the weighting around the shoulders and under the arms. From frame 600 the animation deals with articulating these areas to expose any problems.



As can be seen from the above image, the example component has some weighting errors beneath the arms. To fix these, use the custom Weighting GUI.

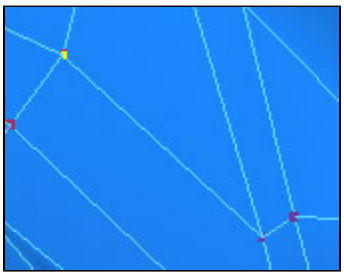
This can be accessed from the Home_Char Tool Shelf. Clicking on the  icon opens the Weighting GUI dialog:



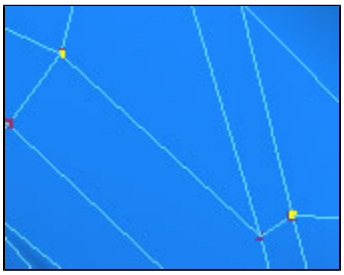
To correct a vertex's weight values, begin by selecting two correctly weighted vertices either side of the incorrect one and then

averaging its weighting values between the other two.

In Maya's Vertex select mode select the first correctly weighted vertex.



Hold down the Shift key and select the second vertex



Select 

to store these values.

Finally select the third, incorrectly weighted vertex.



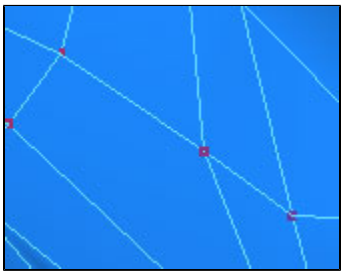
Click the average weights button 

and the last selected vertex becomes averaged between the other two.

When using the average weights button 

all joint influence needs to be unlocked by using the unlock button:





Another option is to use Maya's Paint Skin Weights Tool as well to influence weighting. Indeed, in some circumstances (outlined elsewhere in this document), it is desirable to do so.
Once the weighting of the LOD1 mesh has been tweaked and a desirable result is achieved, use the Copy Skin Weights from LOD1 > LOD2  button on the Home_Char Tool Shelf and then the Copy Skin Weights from LOD2 > LOD3  button to propagate the edited weighting down two to the other two LODs.

Saving a Character Component

Keep the following points in mind before saving the scene for your character component:

- Organize your submissions with four layers, named lod1, lod2, lod3 and skeleton (in this order) so that other artists can use your scenes efficiently.
- All objects must be visible in the Outliner and not grouped or hidden.
- Check that there is no history on all 3 LODs, such as UV tweaks or Vertice edits.
- Connect the test curve in the skeleton so that other artists can reference the asset for skinning tests. Curves will not be exported.
- Deselect Show DAG objects only in the Outliner and delete all unnecessary nodes.
- Finally, in your Hypergraph use Edit > Delete Unused Nodes.

To save your character component:

Select File > Save Scene or:

1. Select File > Save Scene As.
2. Type a name for the scene and click Save As.

Exporting a Character Component

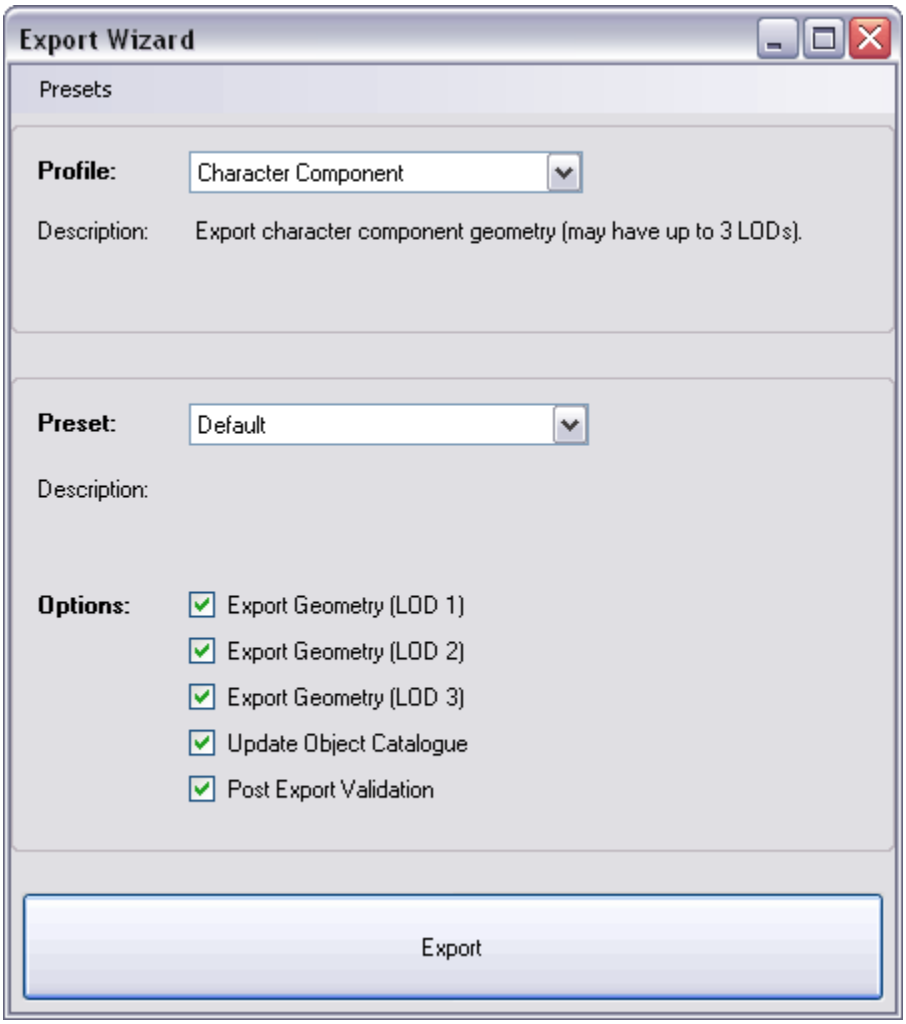
Home Tools provides validation, preview and export functions to help you when you package files exported from Maya in the Object Editor and submit them for quality assurance.

To export a character component:

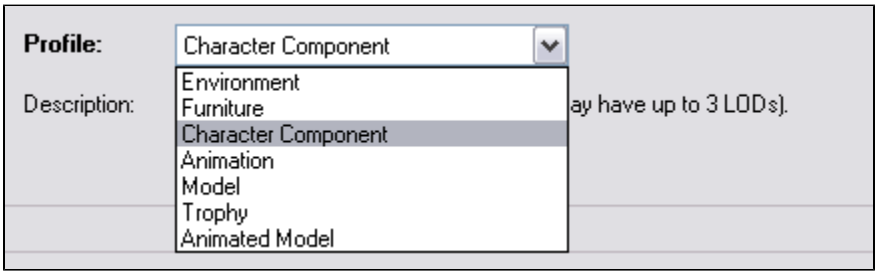
1. Launch the Export Wizard from the Home drop-down menu or by clicking  .

   The Quick Export option exports the scene using the last settings.

   The following dialog displays:



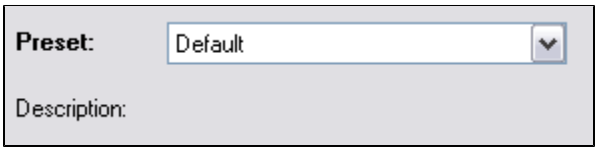
2. Select the Character Component export profile.



   The project's data is exported to the <HDK_INSTALL_DIR>\Intermediate folder which is automatically created with the

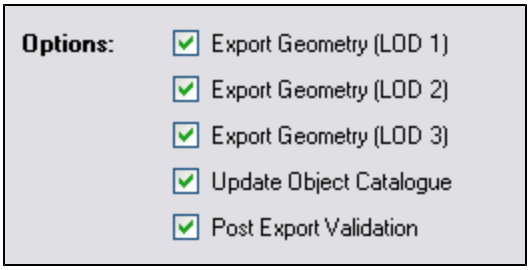
same name as the character component file when the first export takes place. This folder contains a number of files required by PS Home to build the final environment and an export log file that contains useful data about the export.

3. Select the export quality.



The default preset is currently the only option available for character components.

4. Select the export options.



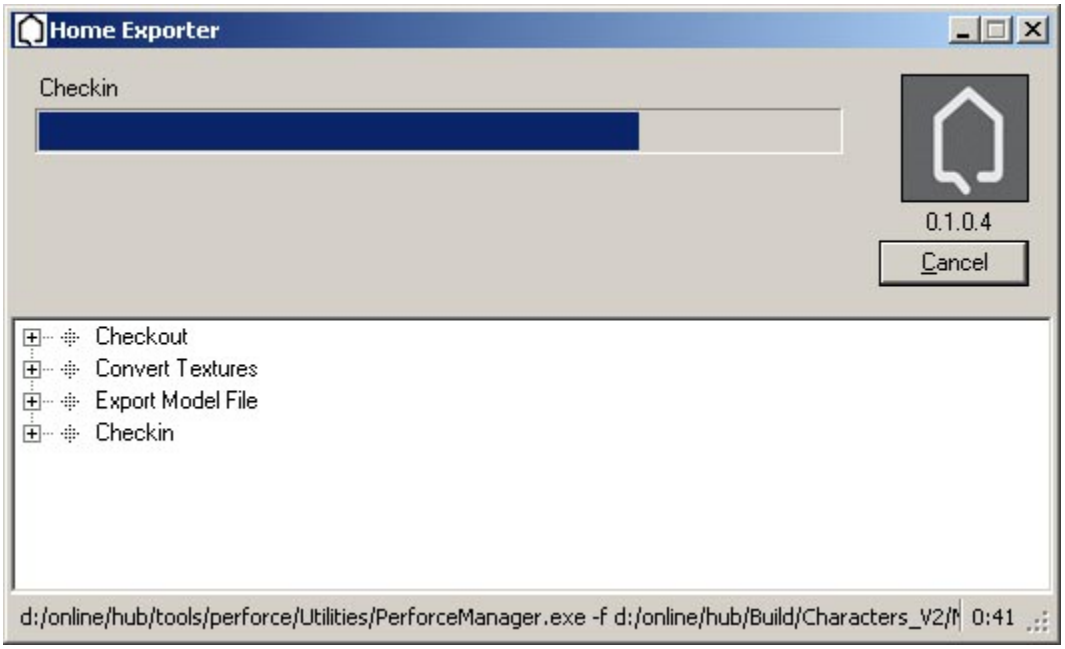
All these options are checked by default and for the most part should be left so. However in certain circumstances, for example if you are working on just a single LOD, you can deselect all options except the LOD you are working on and want to export. This saves the exporter having to re-export any other part of the character component.

5. Click Export to start the export of the data.

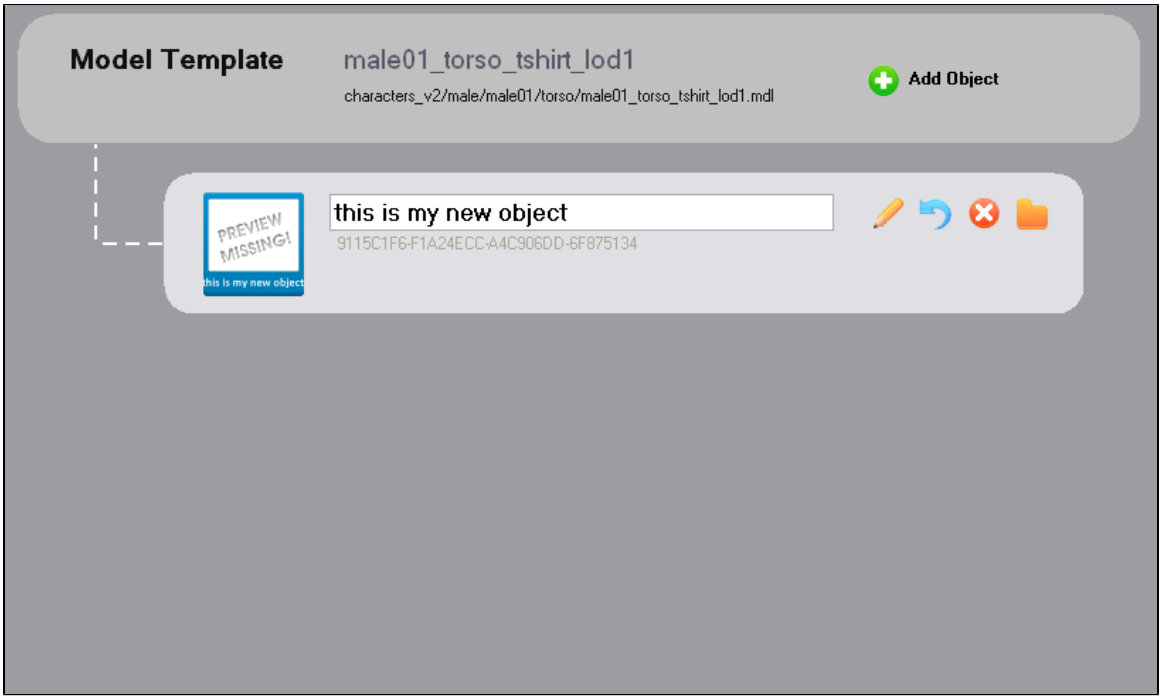
The export process creates three Home model files (.mdl) - one for each LOD. It also creates as many object files (object.odc) as were specified in the Variation Editor, which all reference the model files to create the final Home character components. You can create a number of objects from one model during export, see Creating Object Variations with the Variation Editor.

Home Exporter Process

The Home Exporter starts the export of the LODs.



Then the Clothing Object Editor appears.

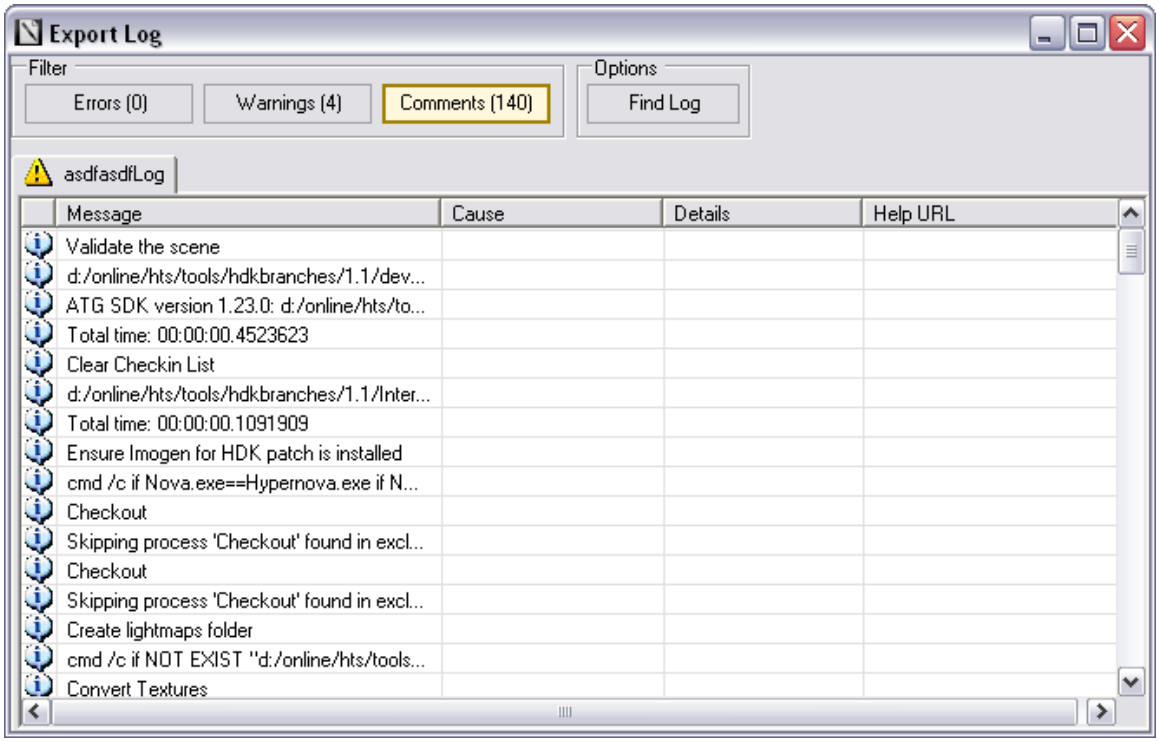


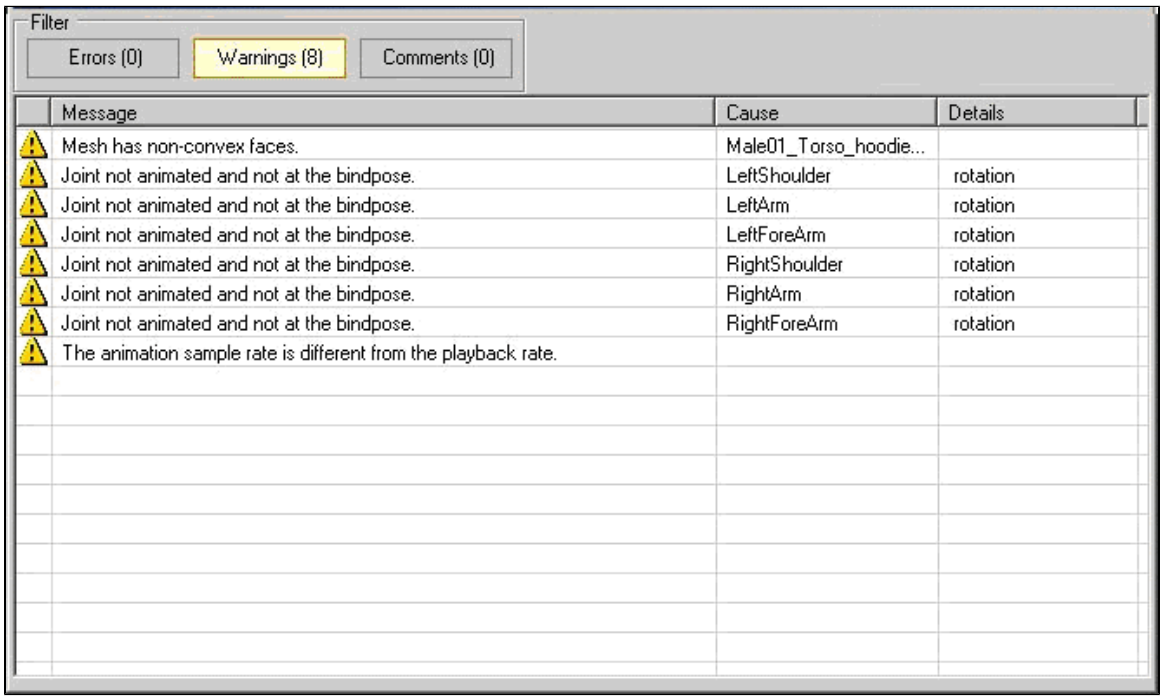
See Creating Object Variations with the Variation Editor for details.

When the export is complete, a notification appears:



An export log displays the results of the export in messages listing any errors  , warnings  and comments  , shown below:





If an export fails, the process stops immediately and the report details the particular error that caused the problem. If the Post Export Validation option is selected, the Export Wizard performs validation of your content to ensure it is compliant with PS Home requirements. For full details on what is validated, see HDK Tools Validations. Deselect this option only when you are testing the export to Scene Editor or Object Editor.

If there are no errors, select Next to go to the next step. If any errors are shown, correct these before you continue.

If your export is successful, Export Complete is displayed.

You can also select the Quick Export option from the Home drop-down menu.

> This repeats the last export sequence you performed without having to repeat the Export Wizard dialog.

When validation and export is complete, you can use the editing tool (Scene Editor for scenes and Object Editor for objects or components) to edit and package the scene or object/component created. You must export and package the scene or object/component with the associated editor before it can be submitted for quality assurance.

Previewing Character Components in PS Home

After you have created your components, check the skin weighting and fat/thin variants for all three LODS, as well as the intersections and potential clipping for poses and animations in PS Home.

You can check or profile your components inside PS Home by:

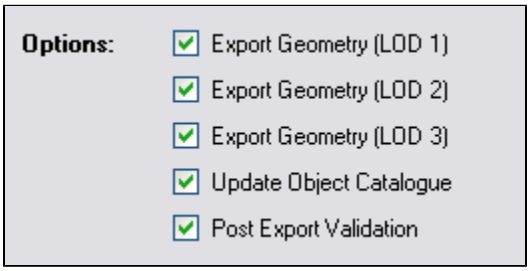
- Launching into PS Home from Maya.
- Using a viewer such as the Character Viewer or the HDK Browser (can be launched, for instance, from Target Manager or via `hubstartup` using `--characterviewer` and `--hdkbrowser` respectively).
- Using a profiling tool such as the Static Object Profiler (Batch Validator) or Debug Console commands (see the Debug Console Command Reference).

After you have skinned your component and created its fat and thin variants and levels of detail, you can export your character component.

To preview character components:

1. After you have skinned your component and created its fat and thin variants and levels of detail, you can export your character component.

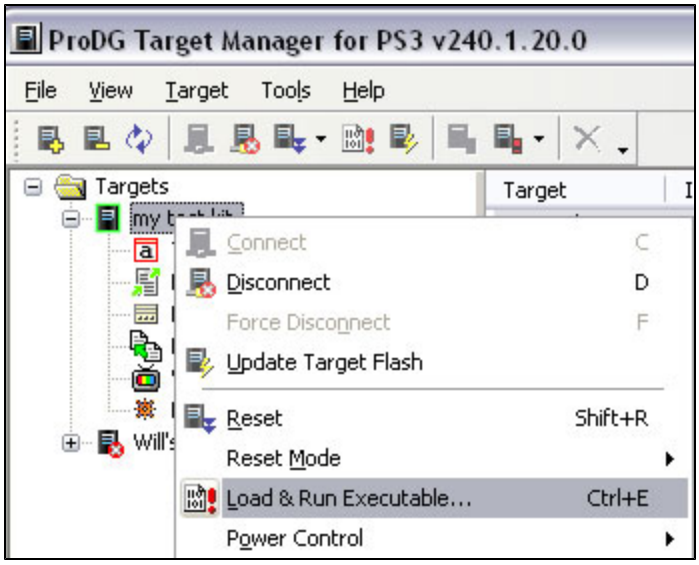
   In the Export Wizard, make sure that Update Object Catalogue is checked:

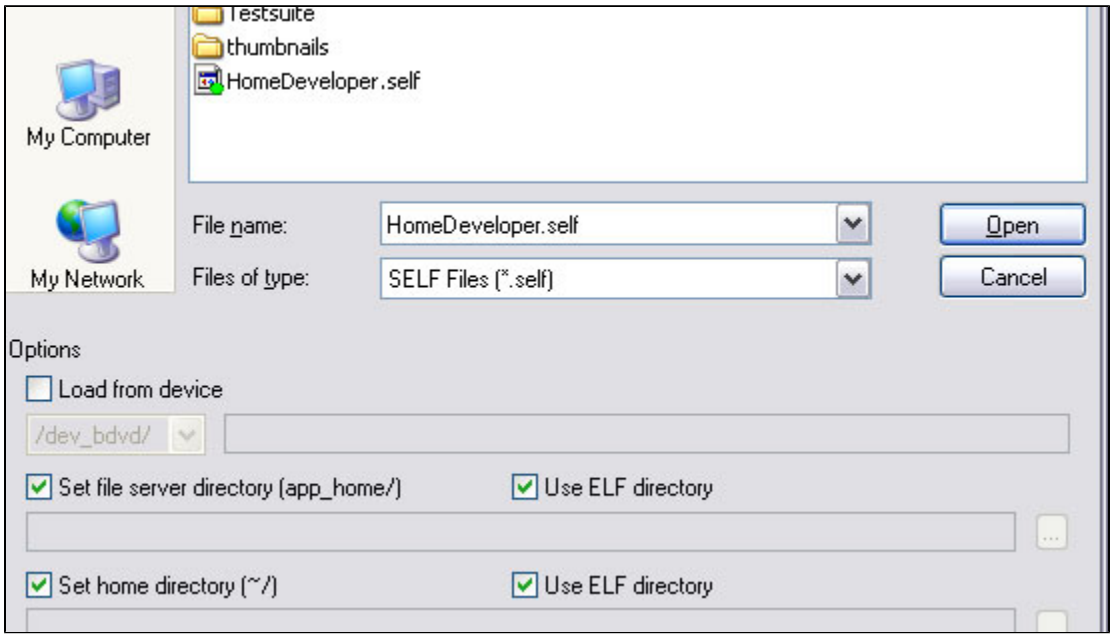
   > ⚠ If you are submitting the content to QA, ensure Post Export Validation is also checked.

   When export is complete, the Uuid is given a unique number. This is the object ID for that object that is used by PS Home and indicates that it has been added to the object inventory on your machine.

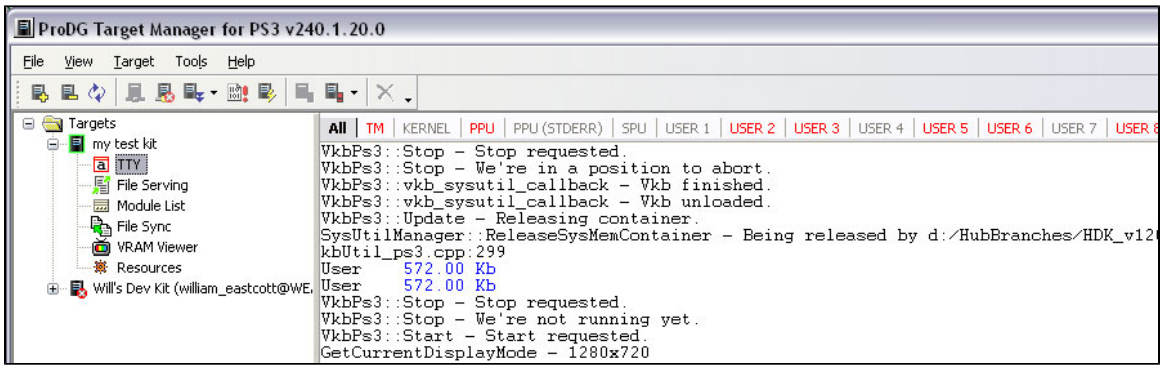
2. Run PS Home from the Target Manager by running the HomeDeveloper.self file in the `build` folder.



3. Make sure the boxes are checked as follows:



As PS Home loads you can monitor its progress by using the TTY window in the Target Manager:



When loading is complete, the Harbor Studio apartment displays.



4. Press Start on the PlayStation®3 controller to display the Menu Screen.
5. Select Wardrobe > Clothing.



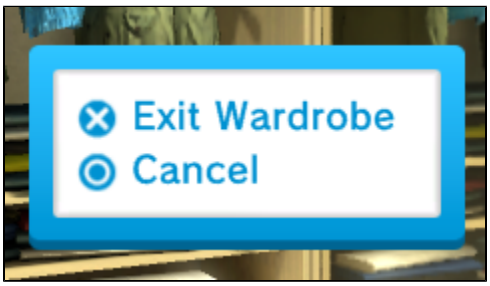
6. Go to the Torso component type:



7. Find the exported asset. At this point it does not have a thumbnail as you must create and apply these to the object separately. Using the Arrow buttons, cycle through the shirts until you find the one you exported.



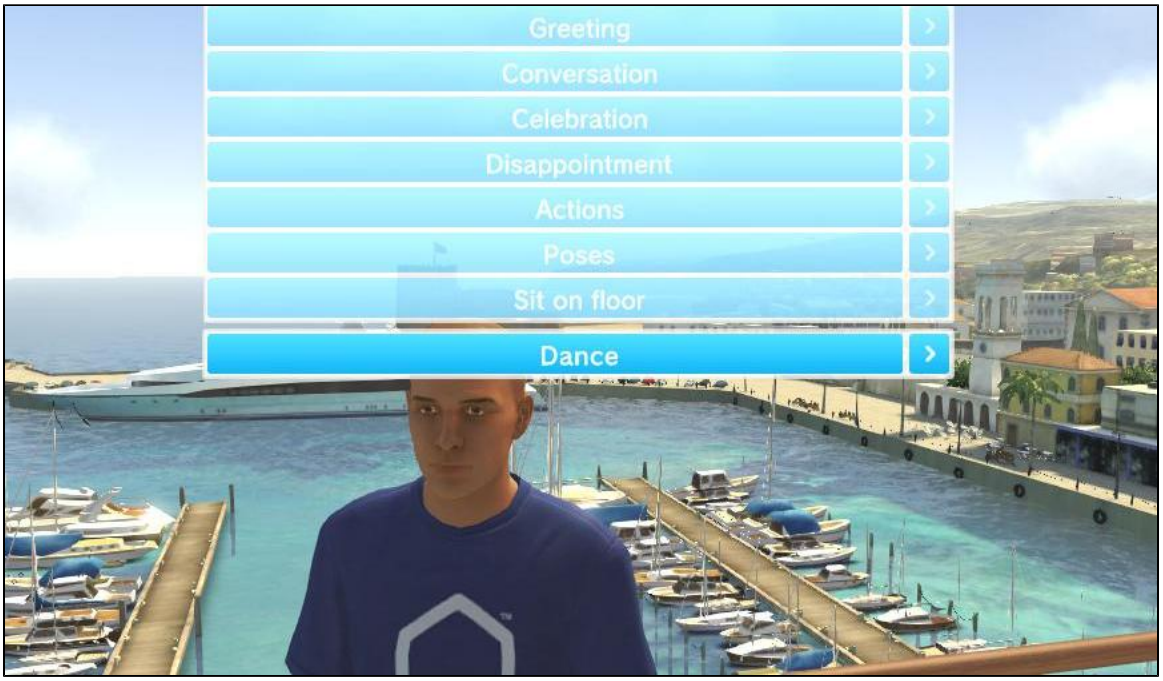
8. With your new shirt applied to the avatar, exit the wardrobe to go back into the apartment.



9. In the apartment, make your way out onto the balcony. This is a good area to get the camera to close-up on your asset using the camera collision.



10. The R1 button brings up the animations menu. Use the animations to check your vertex weighting. The dance moves are particularly useful for this task.



There are several useful console commands you can use in the dev debug menu, to profile your character components, such as `Quartermaster.OverrideLod`, which allows you to set which LOD to display (hi, med, low or none).

> ⛔ Do not use `Quartermaster.OverrideLod` in the wardrobe.

For a list of debug commands, see Debug Console Command Reference.

Character Component Creation Rules and Conventions

This section describes the rules and guidelines for making acceptable character components.

All components must:

1. Conform to the specified skeleton
    - You can tweak the basic proportions within the game, but the original meshes must have edges placed correctly in relation to the basic skeleton.
2. Conform to the correct template edges specified
    - For example, a naked wrist must use the template mesh in order to have the same number of vertices at the wrist and so conform in shape when connecting to a bare arm.
    - Wherever possible, component topology should conform to the major guide edge loops on each LOD (as defined in the template meshes).
3. Follow the rules and parameters set out by the Home Team
    - Only the Home Team can perform the following actions, because these features have not yet been fully implemented:
        - Change the fundamental head shape
        - Add bespoke heads
    - Waistlines on leg components must terminate at the edge of the component rig. There should be no midriff, as the midriff can only be part of a torso component.
    - All components must fall within their defined template volumes to avoid intersection with adjoining components.
    - Any component that does not extend to the limit of the component edge, for example a sandal, requires skin beneath it to connect to the adjacent component.
    - Hats and Hair occupy the same slot. This means that if the player selects a hat for the avatar, the hairstyle they currently have will be completely swapped out for the hat. Any hat that requires hair to be visible beneath it will need to incorporate that hair into the hat component, using the correct material types.
    - All components, textures, shading groups must follow the correct naming conventions to allow the exporter to function correctly (see Geometry Naming Conventions).
    - All textures must conform to the correct sizes.
    - All components require 3 levels of detail.
    - All components and their respective LODs (level of detail meshes) must conform to the specified memory limits.

> Keeping to the parameters set out by the Home Team will prevent unnecessary reworks and delays.

Geometry Naming Conventions

Follow these guidelines to ensure the correct functioning of the exporter:

1. Name Mesh Files according the formula:`Rig_ComponentType_ComponentName`

    ```
    Female01_Torso_tshirt.mb
    ```

    > There are three sections. Capitalization is compulsory for the first two sections but not the latter.

2. Make names as descriptive as possible to make it easier to navigate assets. For example:

    ```
    Female01_Torso_tshirtcapsleeve.mb
    Female01_Torso_tshirtvneck.mb
    Female01_Torso_tshirtlongsleeve.mb
    ```

    or

    ```
    Female_Torso_dresssummer_lod1
    Female_Torso_dresssummer_lod2
    Female_Torso_dresssummer_lod3
    ```

3. The meshes within the file inherit the filename, but with a lod suffix `Rig_ComponentType_Component Name_lod1`, e.g.

    ```
    Female01_Torso_tshirt_lod1
    Female01_Torso_tshirt_lod2
    Female01_Torso_tshirt_lod3
    ```

> As of HDK 1.75, only the lod suffix is required for a successful export; you no longer need to keep the Maya filename and mesh names in sync.
> e.g. If you were to rename the file from Female01_Torso_tshirt.ma to Female01_Torso_tanktop.ma, you would not have to rename the meshes within.

4. Name the shader groups within the file correctly, particularly components with multiple materials. The naming conventions are as follows.

   These are set by default when a component is created and should not be changed:

headSG	torsoSG
bodySG	legsSG
hairSG	feetSG
facehairSG	accessorySG
hairgripSG	handSG

> For 'Skin' and 'Head' ATGMaterials, the name of the attached ShadingGroup (SG) is important. For a body component, the 'Skin' ATGMaterial must have its ShadingGroup (SG) name set to bodySG. Similarly, a head component must have ShadingGroup (SG) name set to headSG. Otherwise the game will refer to 'TextureNotFound' instead. If you select the File > New Home Character Component template, the Skin and Head ATGMaterials are set automatically.

Texture Naming Conventions

As with the naming conventions in Geometry, please follow these guidelines:

1. Name textures after their corresponding mesh. For example, the .PSD containing all the working layers in artsource could be:

   Female01_Torso_tshirt.PSD

   The .PSD should contain all the color, normal, specular and alpha maps on layers.

2. Name .DDS files identically, using the texture type designation as a suffix. For example:

   Female01_Torso_tshirt_c.dds
   Female01_Torso_tshirt_n.dds
   Female01_Torso_tshirt_s.dds

> The suffixes showing texture types above are c (for color), n (for normal) and s (for specular).

3. Add a new color section to the name of color variants if a texture requires a number of color variants. For example:

   Female01_Torso_tshirt_blue_c.dds
   Female01_Torso_tshirt_red_c.dds
   Female01_Torso_tshirt_white_c.dds

4. If a color variant requires a unique specular or normal map, name the by adding a new map section to the name. For example:

   Female01_Torso_tshirt_sequins_c.dds
   Female01_Torso_tshirt_sequins_n.dds
   Female01_Torso_tshirt_sequins_s.dds

Character Texture Conventions

All materials require at least three textures. Default, for example, needs the following:

- a color map (c)
- a normal map (n)
- a specular map (s)

Some materials, such as hair, have a fourth texture to modulate the transparency of the alpha channel, i.e. an alpha map (a).

In terms of textures, the character template (the basic naked character that all assets are based on) is broken into two areas: Head and Body. These are defined by the Tex-Gen system and there is no need to adjust them.

Each clothing component requires a unique default material. If a material has transparency then it is defined by the alpha channel of the color map.

The Hair Material requires color (c), specular(s) and a separate alpha map (a).

Many accessories use NoNormals.dds (found in <HDK_INSTALL_DIR>\artsource\Generic_Shared_Textures\Characters), as they are small and do not require normal maps. If larger accessories require it, you can increase the color map to 256x128

Character Component Requirements and Recommendations

For the recommendations and requirements, as well as a full list of all the validations that the tools automatically run on character components (hats/hair, torso, hands, legs, feet, accessories, etc.), refer to the Content Requirements. Two sets of limits are provided in the Technical Requirements against which the components can be validated by the tools - triangle counts and host memory limits for all three LODs.

> Triangle Count Limits The triangle counts are only a suggestion and not a hard limit or requirement. They are meant to function as a handy indication for artists when planning and developing assets.

Profiling Clothing

See the Profiling in the Client, Character Viewer, Profiling Character Components, and Validation sections for more information.

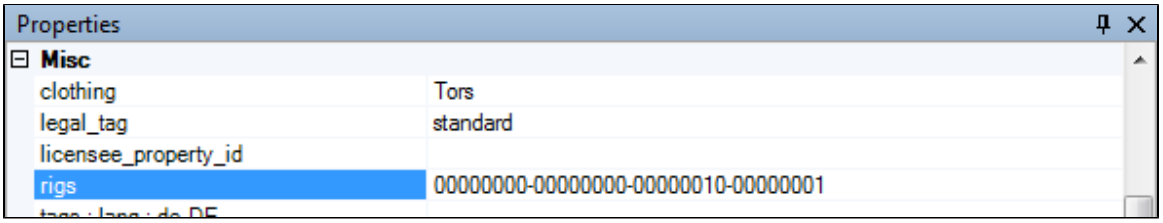
There are profiling tools available to help profiling and optimization of scenes and objects. Profiling will determine how resource-intensive the scenes and objects are.

Currently there are several methods to profile scenes and objects. The object profiler space profiles memory and mesh counts for furniture and clothing items.

Note: to view character components on a rig, you need to type in this command followed by another, i.e.:

```
--object <OBJECT_ID> --context clothing --rig 00000000-00000000-00000010-00000001
```

To find the relevant rig number for an object, you can look at its Metadata properties in the Object Editor. Under Misc you will find the clothing type, in this example Tors, followed by the Rig ID number:



See The Home Client and Testing, Validating and Submitting Content for details.

There is a Character Viewer available that lets you view character components on avatars, animate them, select different LODs, and other options. For details on this see Character Viewer. For command line parameters that allow you to view and profile content, see the Home Client and the Profiling in the Client sections.

Recommended Dimensions

See Dimensions for a full listing of recommended dimensions (including for avatars, clothing, companion objects, and other items that you place in an environment) especially the section on Clothing Dimensions.

See also Profiling Character Components and Batch Validator Clothing Limits for information on validating clothing dimensions.

General Character Component Quality

Compare the two shirt assets shown in the following images.

The first asset is quite simple:



The second asset shows more detail:



Texture Issues

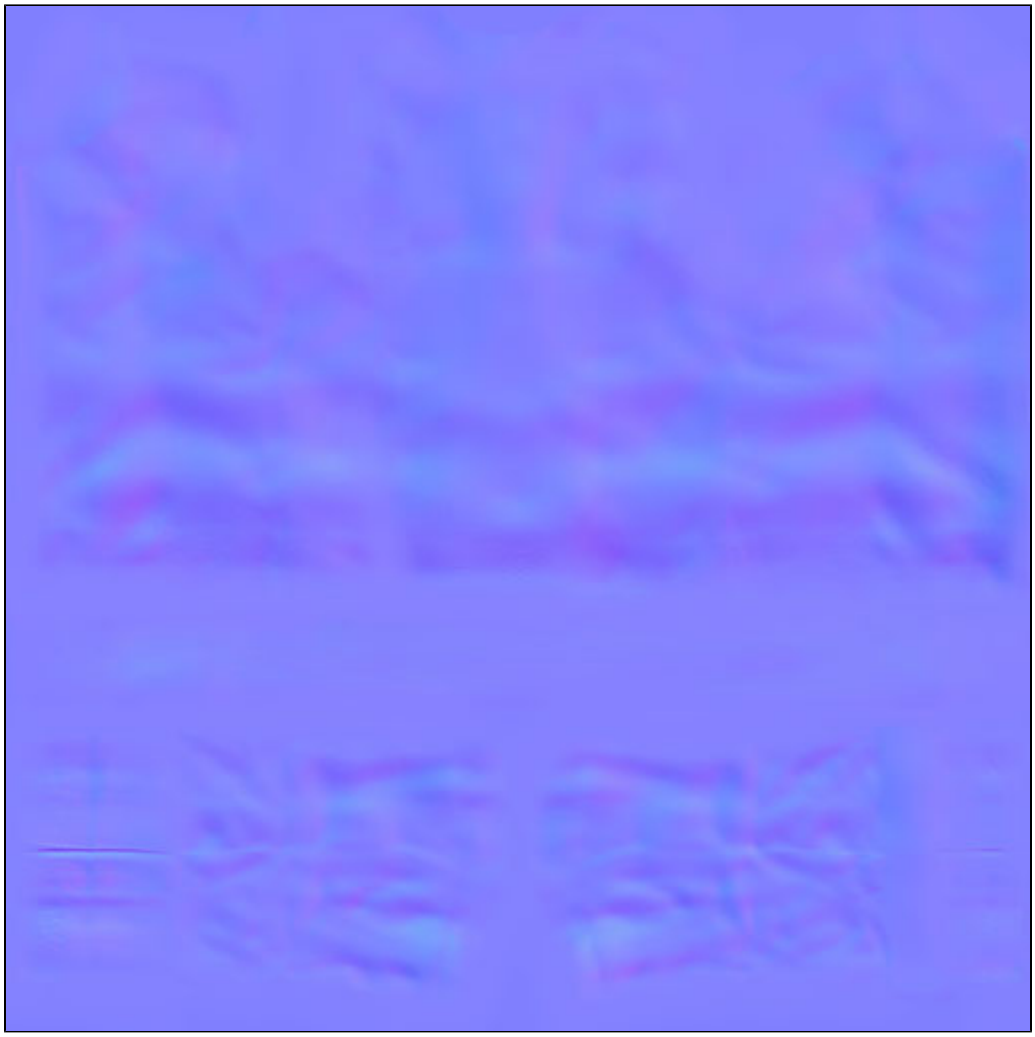
This section lists the most common texture issues and how to avoid them.

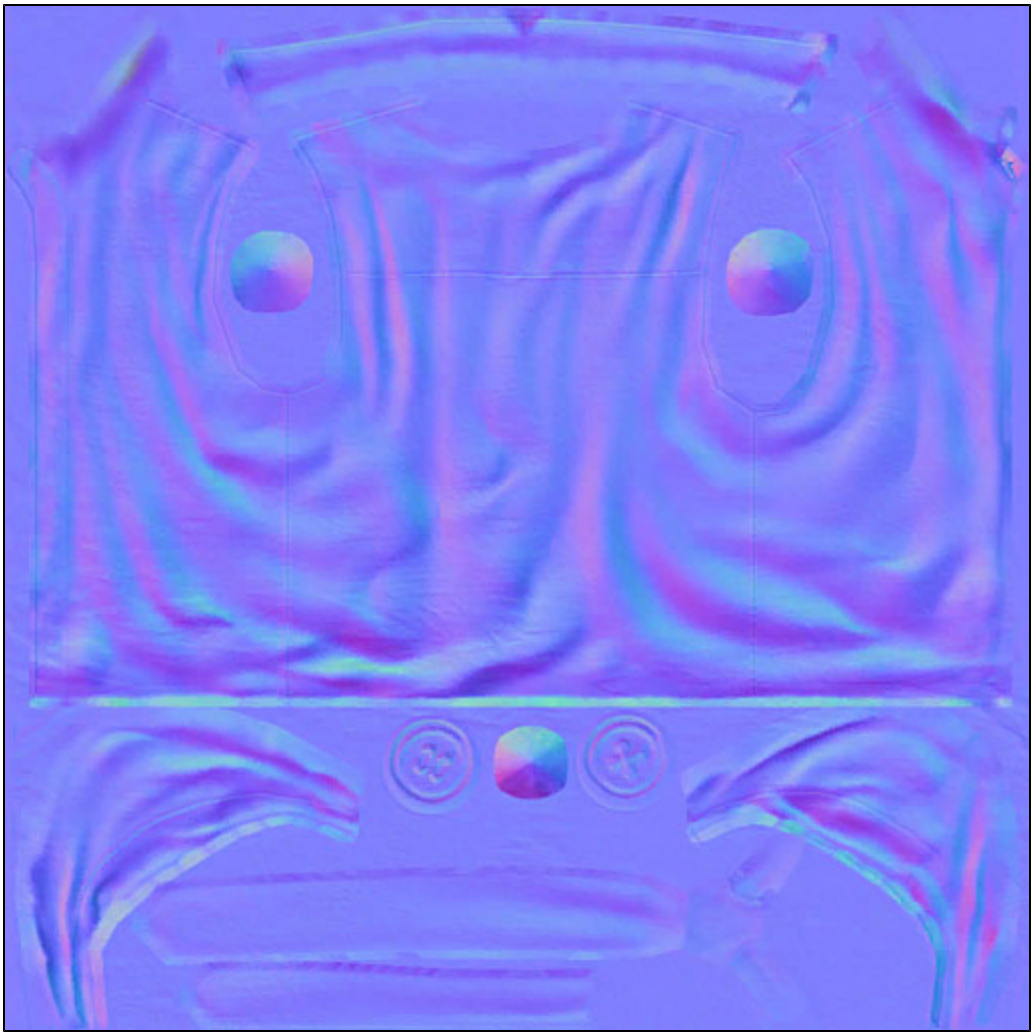
Color Maps that are too Simple

Although clean mapping is essential, you must have some shading in the texture. Shading defines texture and provides subtle shadowing, especially in large folds. Do not make the shading too strong. Make it like cavity or occlusion mapping.

Normal Maps that Lack Detail

If the normal maps lack detail, that is, they contain no major creases or folds, they seem very bare, and much of the detail looks unnatural in relation to the geometry. You should have fine creasing around buttons, seams and large folds, where the material hangs on the person wearing it. The following images show different levels of detail.





The map on the left shows little detail in the cloth. The map on the right shows a cloth that creates swags around the shoulders and bunches over the waist. The buttons have holes and cotton thread is defined in the normal and material texture.

Shades of Black

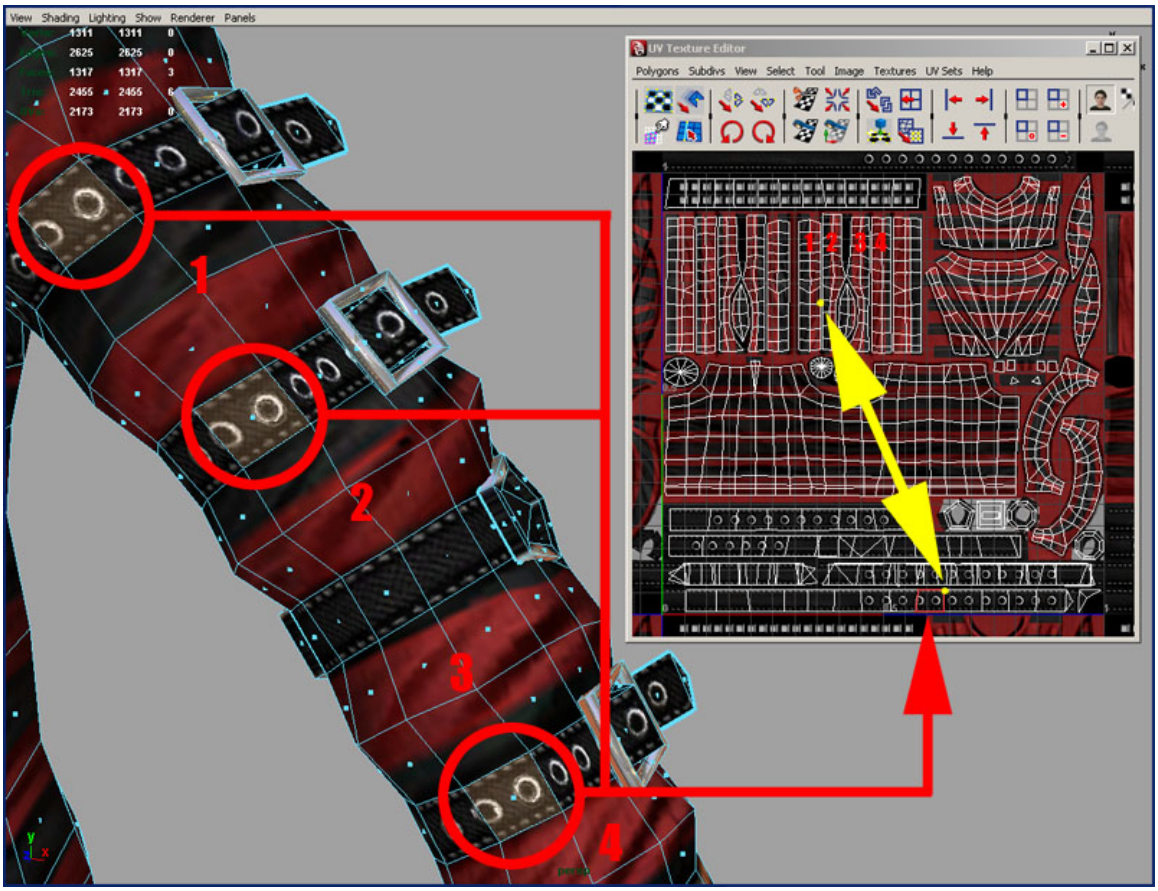
Black textures often appear darker in-world due to the in-world lighting. For clothing to appear similar to the swatch on the left, make black textures more charcoal in appearance (similar to the texture on the right), and allow the shadow to give an overall darkening effect.



UV Best Practice

The number of UV shells used on a mesh is very important. Multiple shells require duplicates of vertex data, and this (combined with the duplicate information required for blend shapes) can double or triple the size of a model.

For example, the mesh shown in the following image has been built with what would traditionally be seen as a very economical use of the texture space:



It re-uses parts of the texture that are repeated, rather than creating similar looking areas on the same texture. The circled faces all correspond to the same UV co-ordinates.

The problem arises because every character and every component has a size limit. If there are too many separate UV shells on a character component, the number of duplicated vertices increases the size of the file beyond the safe limit, even if the polygon count is within the safe limit.

For example, look at the areas 1,2,3 and 4 separated by a repeated texture in the image above. Each area has its own shell, marked correspondingly on the UV Editor. This means that all the border vertices are duplicated like the vertex marked yellow (pointed out by the yellow arrow as having the same vertices, but two co-ordinates). The result is that while the mesh only has 1311 vertices, it has 2173 UVs.

When you add the thin and fat blend shapes into the equation, it triples the number of vertices, so the file size balloons. To prevent reworking, be aware of the memory limitations and avoid creating too many UV shells.

See Technical Requirements Rules and Conventions for guidance on:

- The restrictions on content and memory usage
- The maximum recommended sizes (in KB) for the exported LOD character component .mdl files
- The Host memory used by the former objects in Home

Topology Issues

This section describes the most common topology issues and how to avoid them.

Open Polygons and Paper-Thin Geometry

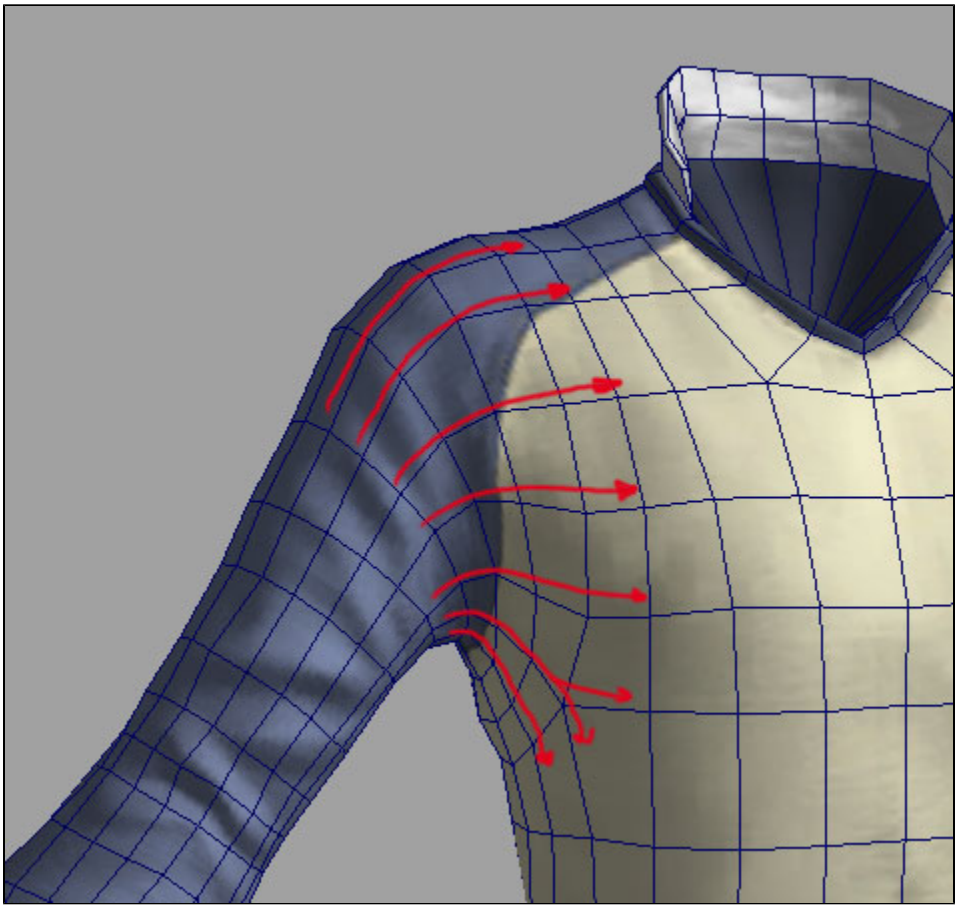
Necks and sleeves need to close at the ends, in order to seamlessly interlock with adjacent components (the head and wrists). If the reverse side is visible, such as cuffs or collars that overlap with or protrude further than the adjacent component, you can

use a number of techniques to make sure that they display correctly.

For example, build collars with some geometric depth, otherwise they look at odds with the rest of the mesh, as if they are made of paper. It also breaks the illusion of normal mapping applied to it. To prevent cuffs and collars from appearing as if they are made of paper, you can use a cone of polygons. If you use this technique, you do not need to render clothing with backface culling disabled.

Shoulders

The quality of the topology for shoulders determines how well the component animates. The following image shows how all the edges running along the arm are evenly spaced and flow into the body in much the same way that the deltoid flows into the pectoral muscle.

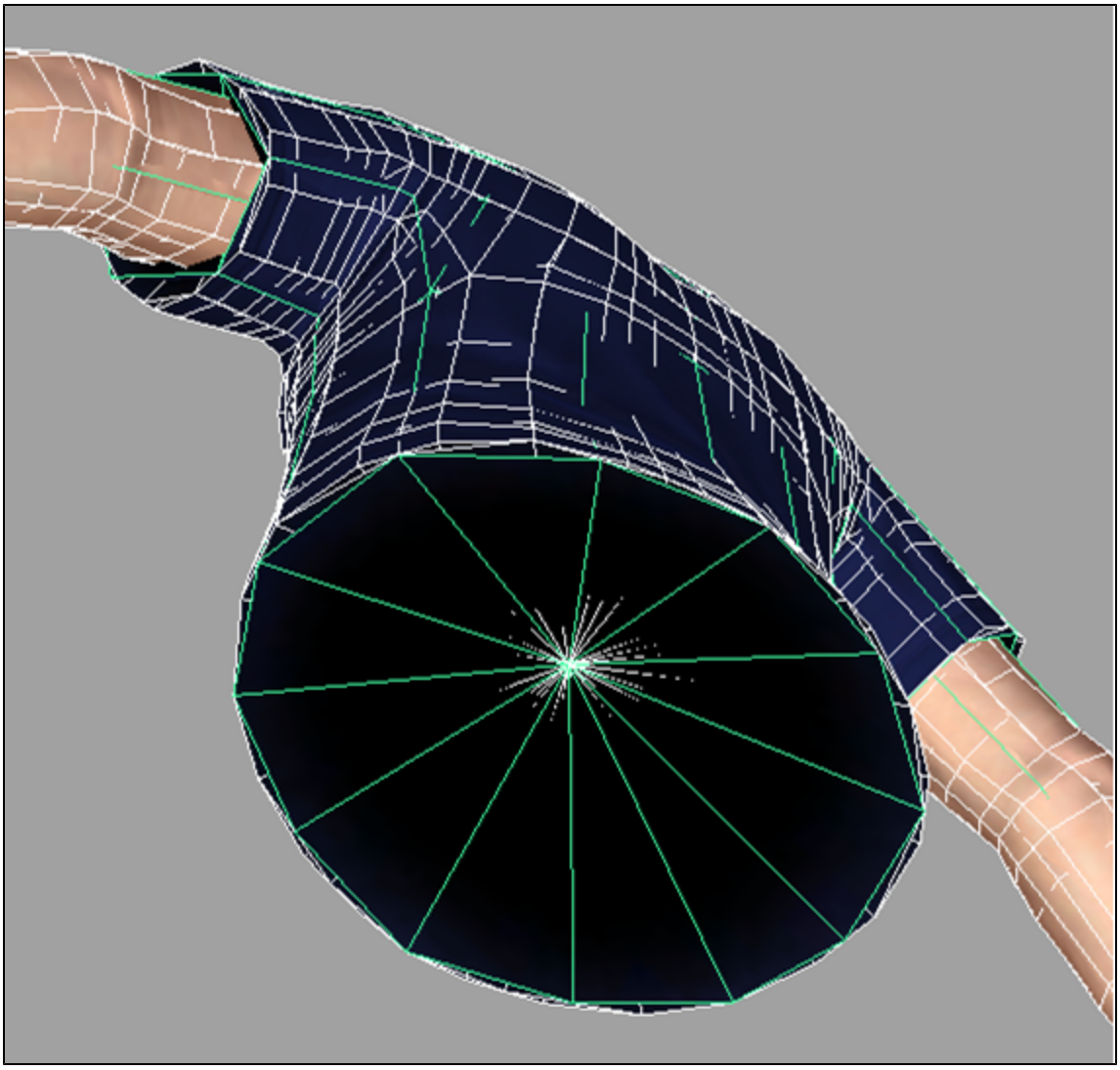


Seams and Sealing Vertex Positions

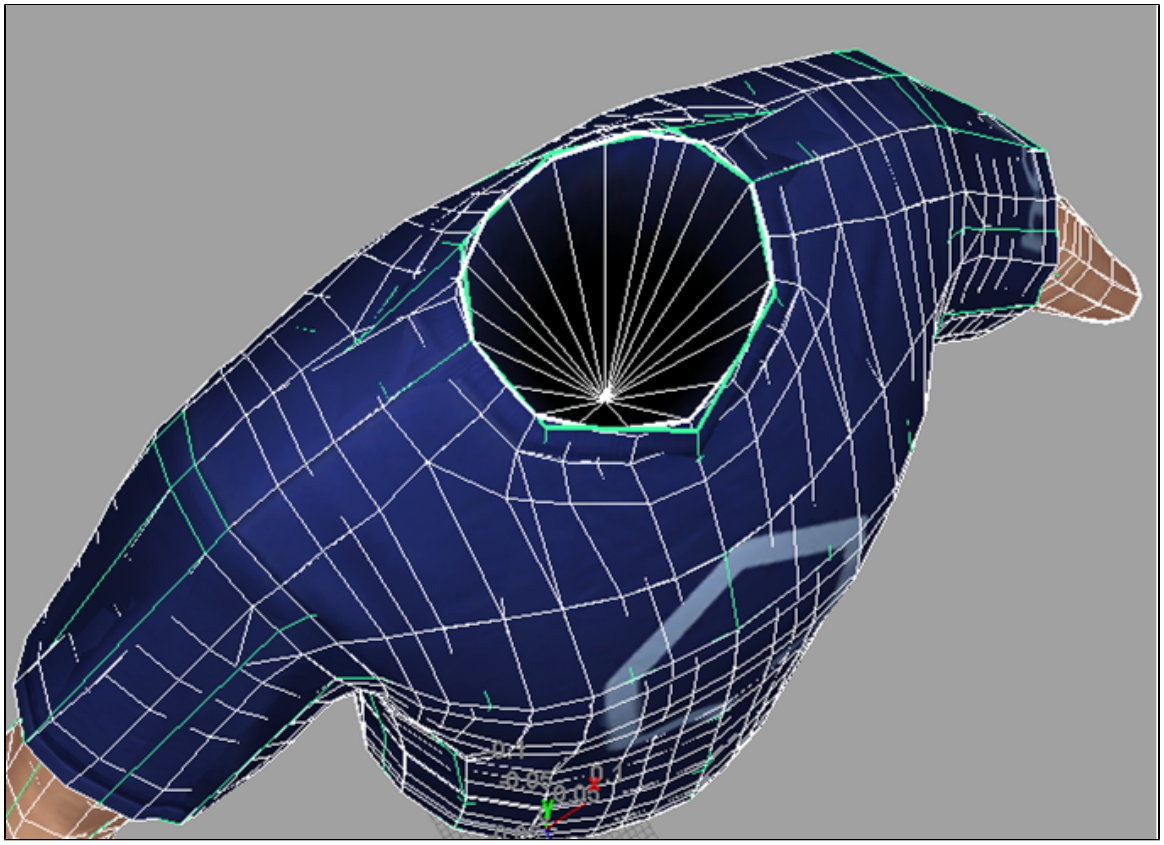
Avoid holes or unsightly seams in clothing (character components). All meshes should seal (that is, be continuous), to make sure that clothing fits characters of different sizes and joins well with adjacent components.

To avoid having holes or seams:

- Make sure that the vertex positions exactly match exactly those of the adjacent component. For example, for a torso, the position must match that of the hands, head and neck, and legs.
- Seal the character components, as shown in the following image, which shows the view of the underside of the t-shirt in the torso component template:



All areas where you would normally find holes in clothing are sealed. In the image above, the area where the torso and leg components intersect is made solid. The other area where this happens is where the torso and neck components meet, as shown in the following image.



You can avoid holes or seams for the various neck sizes by placing the sunken sealing faces of the t-shirt collar to intersect with the neck-faces. Also, make the collar indented rather than flat so that it does not appear too thick for very thin neck shapes. You can also use sealing faces to create the illusion of depth to the clothing item, and to texture the inside portion of the clothing item.

Editing Vertex Normals

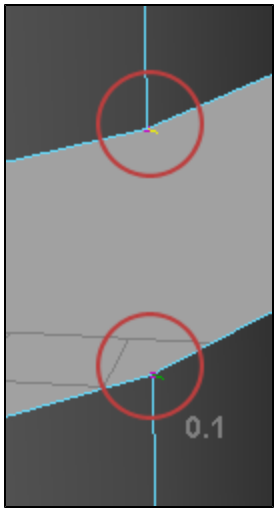
To make the edges of two meshes connect seamlessly, use the Edit Vertex Normals to copy vertex normal information from one vertex to another. Alternatively, you can average the vertex normal between two vertices. You normally use this tool on the vertices along a join between two meshes, such as the border between the arm and the wrist.

To copy vertex normals:

1. Select  to open the Edit Vertex Normals tool:



2. Select the object with the normals you want to edit and click Load. The name of the object appears in the Modify Object field.
3. Select Show Normals or Hide Normals to choose whether or not the normals are displayed.
4. Hold the SHIFT key and select two vertices. The normal information on the first vertex is copied across to the second vertex.



5. Select EDIT NORMALS to copy the normal information to the second vertex.

The vertices now share the same normal information. The second vertex appears yellow when it has been edited.

Applying the Average

To use the average of two vertex normals and apply it to a third vertex, follow the same procedure as outlined above, but select three vertices. When you select EDIT NORMALS, the third vertex takes the average of the first two vertices.

Character Component Modeling Best Practice

Best practice and lessons learned through development ensure that you get effective results.

Models and Triangle Counts

- Gain as much photo reference as possible.
- Model clothing assets around the standard male or female naked templates.
- Have a base mesh that adheres to the poly counts provided in Testing, Validating and Submitting Content.
- Adhere to the prescribed triangle counts from the beginning. Doing so will help you at the sculpting stage.
- In the early stages, inspect the triangle counts to achieve a certain look, and to make the sculpting stage easier.
- Later, when you have generated the normal maps, reduce the triangle counts.
- Use quadded geometry as much as possible, because too many triangles can produce pinching effects in Zbrush.
- Achieve better skinning results with clean edge loops, which also makes the LOD process easier and cleaner. There are various edge loop modeling tools available, commercially and for free.

UVs and Normal Maps

- Create UVs.
- Generate normal maps with Zmapper in Zbrush.
- Do not use multiple UV shells as this creates problems in the LODding process (e.g. a boot that is textured over one UV shell is much easier to reduce than one that uses a UV shell for each micro component).
- A polygon cleanup should be performed on all LODs meshes before skinning and creating the fat and thin variants. The artist should remove any N-sided polygons (polygons with more than 4 sides) and stray vertices.

UVs and Zbrush

We recommend to do UVs before sculpting in to Zbrush. However, some artists prefer to do all their sculpting in Zbrush before the UVing process because the sculpting process can sometimes have a distorting effect on them. This process is quite simple:

1. The mesh is exported as an .OBJ from Zbrush at its lowest subdivision level
2. UVs are created, and then the .OBJ is re-exported
3. The mesh .OBJ file can then be brought in to the sculpted Zbrush tool and the UVs will take effect

    - Normal maps are generated in Zbrush. UV shells placed in different UV space allow for the use of Sub Tools in Zbrush. This makes it easier to isolate specific areas of geometry (e.g. by moving the UVs of an arm into +1 UV space makes it easy to hide the geometry of the rest of the torso when sculpting).
    - When sculpting in Zbrush, it is useful to have quality photo references. However, replicating from photos exactly and in detail is not advisable as this may not always transfer well to an in-world model alongside skinning and the generation of fat shapes, etc. Zbrush has tools to generate details, such as cloth creases, that work well on an in-world model.

A normal map is often not suitable for complicated areas that fold over themselves. These may be more usefully built into the

base geometry, e.g. collars are a good example of this. It is often a balancing act with the poly count of the model.

Small Details

- Small details (such as buttons) can be created in different ways. Some artists prefer to sculpt them into their Zbrush model and rely on the texture, whereas others prefer to create separate geometry.
- Skinning is often easier when details are included as part of the texture. However, the best result sometimes requires details to be separate elements as this makes it easier to apply specific shaders.
- A recommended texture has the double of the output resolution (i.e. 1024 for the RGB and normal maps). When downsized to 512, the sharpen filter effect helps to preserve detail.

Normal Maps and RGB

- The normal map makes a good starting point to create the RGB. Some artists like to use a grey scale version of the normal map as an overlay in the PSD to better show the high and low lights of the texture. It is not enough to rely on the normal map to do this in-game as it is so dependent on the lighting.
- An ambient occlusion map can also be used in this way.

  You can create maps in Maya, or with third party plug-ins.

  These can be created in Zbrush. A mid level mesh can be brought out of Zbrush, say at subdivision level 4, and then brought into Maya to generate the Ambient Occlusion map - exporting the mesh at too high a level will slow things down drastically. The generated map can give a great high/low light overlay for the RGB.

- When the maps have been finished the mesh should then be optimized for the game. A number of edges can often be removed without having an effect on the overall quality and look of the mesh.
- Meshes with clean edge loops and quads should be relatively easy to LOD. The collapse edge tool is also invaluable here.

Creating Object Variations with the Variation Editor

What Does the Variation Editor Do?

When you create an item of clothing, a character component object is automatically created that references your model's materials and textures.

The Variation Editor enables you to create more objects that reference this model. The associated child objects are variations based on a shared Model Template.

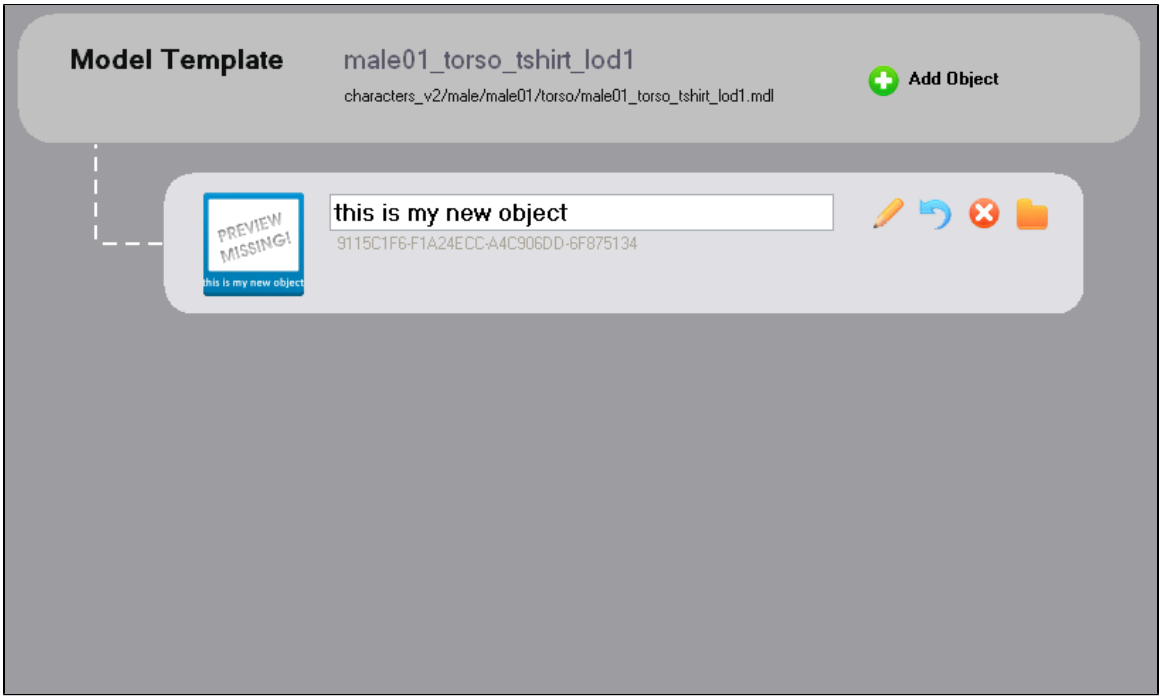
These objects all start out as clones of the Model Template, with the same material and texture information, but you can assign different textures and provide different information for each.

If you change the materials in the Model Template, you can use the Variation Editor to update the child objects you added and reassign textures.

> The name of the Variation Editor changes depending on what type of content is being exported. For example, when exporting character components it is called the Clothing Object Editor.

Default State

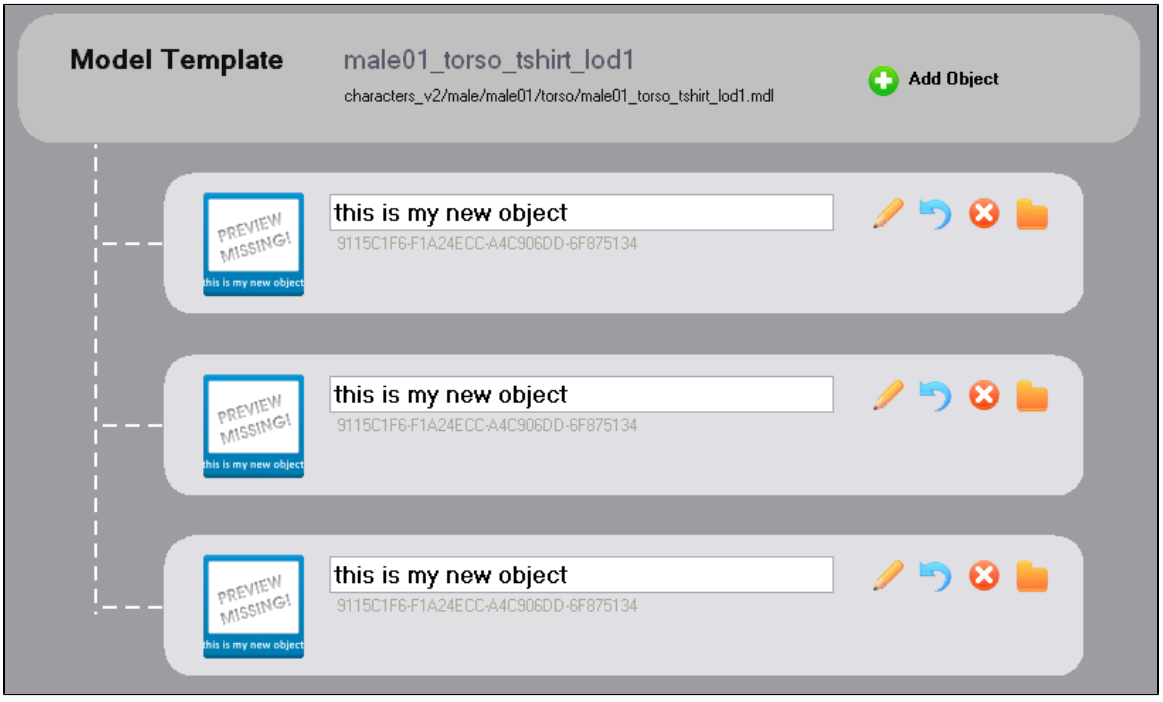
The Model Template and its object:



When you click Add Object from the Model Template, a new object will appear at the end of the list below the parent node as 'Marked for Add'.

Add Objects

Adding objects means creating clones of the first object. Below, two more objects have been added that reference the same Model Template



Why Create Variations?

Having many objects that reference one model file is useful in many ways.

It is a faster to develop multiple objects from one model file and easier to update the associated objects when you make changes to the model file.



Use variations to:

- Make more object copies and assign them different textures.
- Update your objects automatically if you change vertices in your model.
- Update the materials and reassign your textures easily.

About Object Information

Each object has information and thumbnail images. This editor enables you to edit some of the information fields attached to the object.

For access to all Object Information use the Object Editor. 

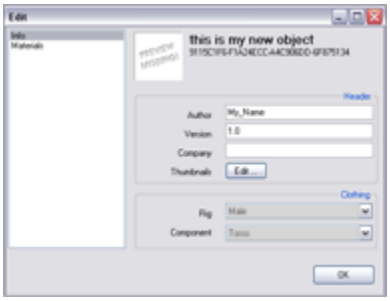
Editing Object Information

You can edit the name directly in the white box on the object child nodes.



If your object is already localized, you cannot edit its name. This is to preserve the integrity of existing any localization.

To edit object information, click the Edit button  next to the object and go to the Info tab.



You can edit:

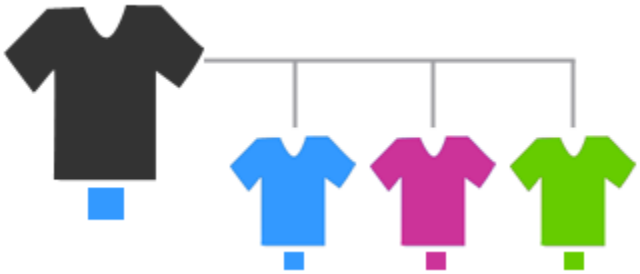
- Author
- Version
- Company/Publisher
- Thumbnails

Updating Materials

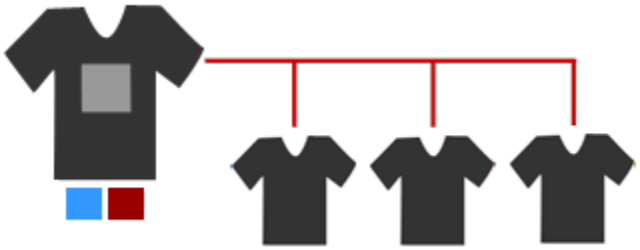
A Model Template has a certain number of materials already. The object variations reference the Model Template materials, and each object variation's material can be assigned a different texture.

But any change to the materials in the Model Template means the original material-texture relationship in the objects no longer exists.

If you change the materials of your model, you must update the objects and reassign the textures so that they relate to the new material configuration.
For example, you start with a model and one object (blue), add two more and then give the two new ones different textures (pink and green):



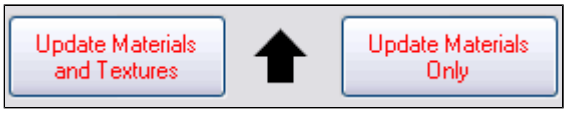
If you change the model file by adding another material (a logo on the t-shirt), the original material-texture relationship no longer exists:



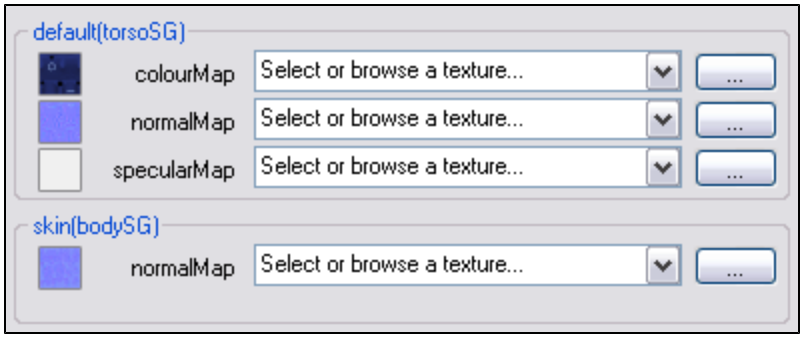
To update a material:

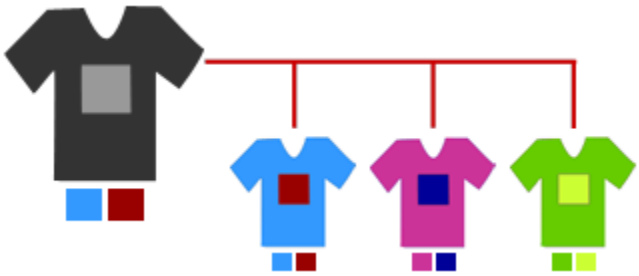
1. Click the Edit button  next to the object and go to the Materials tab in the new window.
2. Check the Update from Maya Model Template box.

   Use one the following options:

		
Option 1: Click Update Materials and Textures to (1) update the materials your objects reference and (2) copy the textures in your Model Template to the objects		Option 2: Click Update Materials Only to (1) update the materials your object references ⚠ Your object keeps the textures you specified, but you still have to reassign them from the drop-down options

3. Update the material and either copy or reassign textures for each object.





Committing Changes

When you finish making changes to the object, and adding variations, select Commit All Changes.

This adds or updates the changes to your object.

Game Launch Objects

The Game system is a session management system that primarily passes information about the session to the target application.

The Game Launching system allows users in PS Home to form groups and to launch into a PlayStation®3 title together. When Game Launch objects are published in the live environment, they are available to any user who has the title on the PlayStation®3 (either installed, or inserted into the system on a disc). The user does not have to purchase or acquire the object.

Game Launching requires development steps to support your title, but adds a new community element for your title when it is completed.

How Users Launch a Game

1. Users gather together in a group in PS Home and select their game options.
2. Users launch the new game. The Home Client writes out the options they have selected to each PlayStation®3's HDD, then resets each user's PlayStation®3 using system software GameExec functionality.
3. The new game is then launched, as though it had been selected from the XMB™ by each user in the original group.
4. When users go to quit the title, they are prompted to return to PS Home instead of returning to the XMB™.

To support Game Launching, a title must detect it was launched from PS Home and read the session file written by PS Home. When launched in this way, a game must skip its usual front-end process, and form a game session with the group of players originally gathered in PS Home.

Users can launch PlayStation®3 game titles from PS Home from:

- The Menu Screen, as a "Supported Game": These games have Game Launching Objects that allow them to create Game Launch sessions.
- The Menu Screen, as an "Other Game": These games do not have Game Launching Objects - users launch into a game without needing to first go to the XMB™.
- A Mini-game/arcade game: The process for launching from a Mini-game is very different. See Launching From Mini-games.

Activating Game Launching

To activate game launching using the developer build, you must install some packages. You can install packages using Target Manager, or from a USB memory stick. See Installing the Home Developer Package

For information on game returning and returning data from a game launched game, see Launch into PS Home from PS3 Title - Game Return to find out how to:

- Return to a specific space or spawn point in PS Home, from a game launched title

- Return data from a game launching session

Submitting Modifications to Game Launch Objects

Make sure that only one game launch object for the title is uploaded to your sandbox. If you upload more than one, the client may download the wrong object.

When you submit modifications to game launch objects, make sure that you use the same UID as for previous submissions.

Title IDs and Communication IDs

For a title to launch from PS Home, a Game Launch object must have:

- Title IDs (you can have several per game title)
- One Communication ID

The Object Editor's packaging tool does not check to see that each Title ID corresponds to the Communication ID.

Title IDs

Title IDs are unique identifiers associated with a PlayStation® package. They provide the cell SDK with a guaranteed method for identifying a particular application or game installed on the PlayStation®3. Title IDs are also referred to as product codes. During development, you can provide a temporary title ID, but when publishing the Home object, you must use the correct title ID.

In the context of game launching, Title IDs tie a game launch object to a particular title or range of titles. You can write a game launch component for any title, provided you know the title ID.

Packages are named SPID-TitleID_00-NameOfPackage.pkg. If your package is called UP12345-NPUBXXXXX_00-NAMEOFYOURPACKAGEFILE.pkg, you Title ID is NPUBXXXXX.

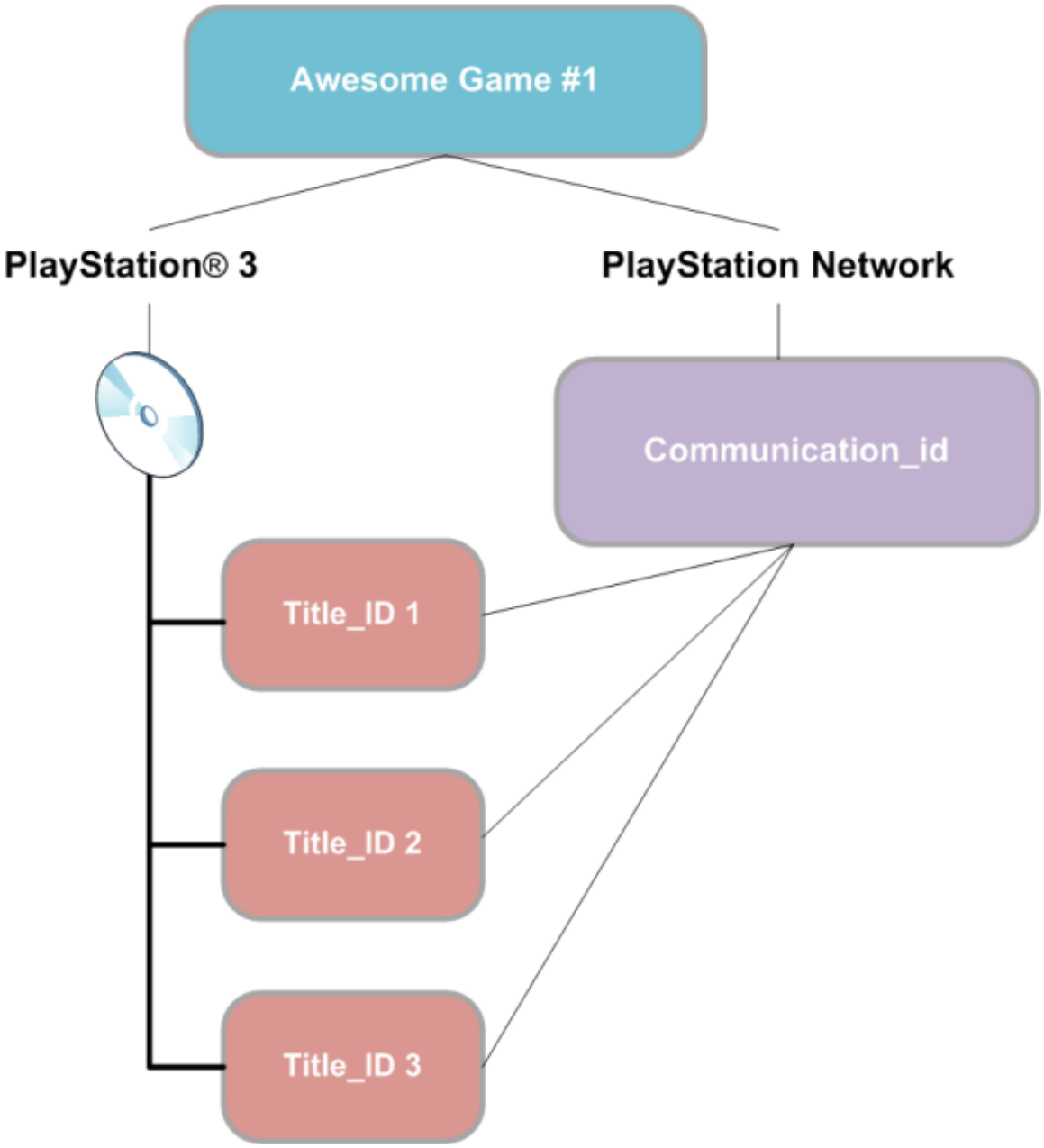
Communication IDs

Network Platform (NP) communication IDs are unique IDs that allow NP services to identify games. You normally have one communication ID for each title.

Game launching requires the communication IDs of the titles to launch correctly.

For more information about NP communication IDs, go to:
https://ps3.scedev.net/docs/ps3-en,NP_Communication-Overview,NP_Communication_ID

Relationship Between Title IDs and Communication IDs



A game title can have several title_ids. The PlayStation®3 console uses the Title ID to identify the title.

A game title can only have one communication ID. No matter which region the user is connecting from, PlayStation®3 knows which title is being accessed and played.

A Game Launch object needs both the title IDs and the communication ID to launch into the game title from PS Home.

Developing a Game Launch Object

To create and test a game launching object, you need the following:

- PlayStation®3 Developer. The HDK game launching patch is only relevant to PlayStation®3 developers as it ties in with PlayStation®3 games or applications only.
- Packages. The required .pkg files for successful Game Launch object testing.
- Communication ID and Title IDs for the title.
- Your title must be able to detect it was launched from PS Home.

The development process is as follows:

1. Create the game launching object in the Object Editor.
2. Customize the start-up sequence, telling the title how to launch, and passing parameters to the title.
3. Test the game launch sequence locally.

4. Test the Game Launch sequence on a Content Sever.
5. Submit it to QA for publishing to the live environment.

Creating and Testing the Game Launch Object

When launching a game from the Menu Screen, there are two ways to build the game launch GUI and to pass parameters:

- Static XML: An easy, but inflexible, way to specify player options, implement menus, etc. See Static XML Game Launching.
- Lua: A flexible way to enable game launching using Lua. This method has customizable options, and allows you to pass parameters via Lua-generated XML, but requires more scripting. See Custom Lua Scripted Game Launching.

This task includes steps that are common to both ways, and steps that differ, depending on whether the game launching object uses static XML or generates XML files with Lua.

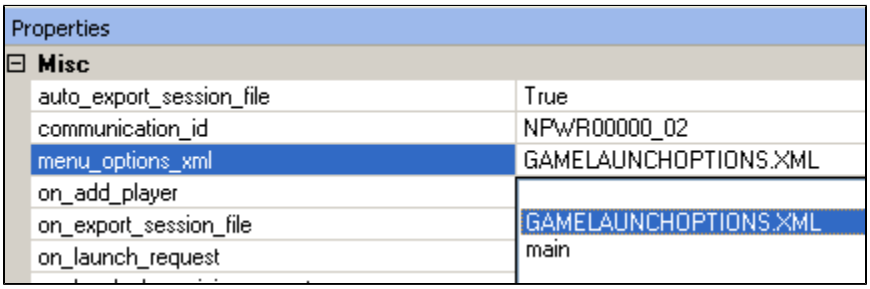
Creating a New Game Launch Object

These steps are common to Game Launch objects that use Static XML and those that generate XML files with Lua.

1. Create a new object in the Object Editor.
2. Add a Game Launch component to it.
3. Select the Game Launch component in the Object View panel.
4. In the Properties panel, set the communication_id property to the Communication ID of your title. See Title IDs and Communication IDs.
5. Select Metadata in the Object View panel.
6. In the Properties panel, set the communication_id and title_id properties to those of your title.

Setting Up the Game Launch Object for Static XML Files

1. Create a game launch options file in XML format. For format guidance, see Static XML Game Launching.
2. In the Object Editor, click Add New Local Resource in the toolbar and select the XML file.
3. Select the Game Launch component in the Object View panel. The Properties panel displays the available properties for the component:



4. Set the properties as follows:
    a. Enter True in the auto_export_session_file field. This tells Home to handle the serialization of the Game Launch Session XML file.
    b. Enter the XML resource in the menu_options_xml field.
5. Add the appropriate functions to the callbacks you use in your script. You can use any callbacks you want, and decide how you want to implement game launching. However, Game Launch objects that use a Static XML should not define functions for the following callbacks:
    - on_export_session_file: With auto_export_session_file set to True, you do not need to use this callback.
    - show_ui: The GameLaunchMenuSys handles the UI using the GAMELAUNCHOPTIONS.XML file.
6. Localize any reference strings in the game launch options XML via the object localization table.

Setting Up the Game Launch Object for Lua Generated XML

1. In the Object Editor, select the Game Launch component in the Object View panel.
2. In the Properties panel, set the auto_export_session_file to False.
3. Leave menu_options_xml empty. The fields in the Properties panel should now look as follows:

Misc	
auto_export_session_file	False
communication_id	NPWR00000_00
menu_options_xml	
on_add_player	OnAddPlayer()

Game Launch objects that use Lua generated XML use more, if not all, of the callbacks to adjust the launch options, balance session master, and to make sure that there is a minimum amount of players in the session before launching.

The Lua script needs to do the following:

- Create a UI for users to see/interact with while setting up a game launching session. The UI must be localized for all regions that have access this Game Launch Object, not just the regions in which it is published.
- Write and export a GAMELAUNCHSESSION.XML file. This needs to be triggered by the on_export_session_file callback.

See Custom Lua Scripted Game Launching.

Testing the Game Launch Object

It is good practice to test a new Game Launch object, by running HomeDeveloper.self from your development PC using Target Manager. The executable detects your Game Launch object and activates it when you select the associated game from the list of those installed by using the Menu Screen. This method does not launch a title because of PlayStation®3 Dev Tool restrictions (a .self cannot boot another .self or .pkg), but it enables you to test that:

- Your Game Launch object activates correctly on selecting the corresponding game.
- Your game launch options are correctly displayed.
- The Home client writes the correct game launch session options to the PlayStation®3's hard drive when a game launch session is created.

You should now have a fully functional Home object that allows the user to configure a session and saves session options to disk for your title to read.

Customizing the Title Start-up Sequence

See Supporting Game Launching in your Title.

To support game launching, your title must detect when it is launched from PS Home and locate the GAMELAUNCHSESSION.XML to extract session information accordingly.

Your title requires code similar to the following:

```
if (LaunchedByHome() == true)

{

// Read session options and launch directly

// into game session

}

else

{

// Start up normally

}
```

Session options written by the Home client persist on the PlayStation®3's hard drive until they are deleted or overwritten.

You can use the File Sync function of Target Manager to find the session options. For example, the following path given is an example of the path that might be returned from the function cellGameGetHomeLaunchOptionPath():

```
/dev_hdd0/game/NPEA00013DATA/USRDIR/GAMELAUNCH/GAMELAUNCHSESSION.XML
```


You cannot access this path directly and you must get the path from cellGameGetHomeLaunchOptionPath().

For example, the PS Home title ID is NPIA00005, which is different from the HomeDeveloperBuild.pkg. cellGameGetHomeLaunchOptionPath() handles the path differences and returns the correct path.

Your title retrieves this path as follows:

```
char gamePath[CELL_GAME_HDDGAMEPATH_SIZE];

char pathPersonal[CELL_GAME_HDDGAMEPATH_SIZE];

char optionsPath[CELL_GAME_HDDGAMEPATH_SIZE + 32];


ret = cellGameGetHomeLaunchOptionPath( gamePath, pathPersonal ) ;


sprintf( optionsPath, "%s/GAMELAUNCHSESSION.XML", gamePath );
```

When a valid GAMELAUNCHSESSION.XML exists, it can be used for multiple launch tests by forcing the LaunchedByHome() function to always return true. For the full specification of cellGameGetHomeLaunchOptionPath(), see the Cell SDK's document, Home_Coordination-Reference_e.pdf.

When you are satisfied that your title supports PS Home launching and normal start-up, build it as a package and install it to the PlayStation®3's hard drive.

Testing the Game Launch Sequence Locally

At this point, you have a Game Launch object that supports your title, and a 'Home-aware' version of your title installed on your PlayStation®3.

You can use the Debug Launcher icon in the XMB™ to launch HomeDeveloper.self from your development PC, and test that Home client can configure a game launch session, exit, load your title with the correct session options, and return to PS Home when the session is over.

Your title determines Debug Launcher invoked it rather than Home client, because the .self does not have a Title ID. Therefore, you may need something like the following in your codebase:

```
// Find out how this title was launched

ret = cellGameGetBootGameInfo( &type, dirName, &execdata );


#ifdef USE_DEBUG_LAUNCHER

    if ((type == CELL_GAME_GAMETYPE_HDD) &&

        (strcmp(dirName, "TEST00000") == 0))


#else


    if (type == CELL_GAME_GAMETYPE_HOME)


#endif


    {
```

This then tells your title that it launched from PS Home. If your Game Launch object works using this technique, you can be confident that it is ready for uploading to a content server and QA submission.

Debug Launcher

The Debug Launcher is located at https://ps3.scedev.net/projects/ps3_sdk/ or in the PS3_SDK_Samples-<version_number>.zip file (where <version_number> is the current version of the samples zip file. You can find this file in the directory:

PS3_SDK_Samples-360_001\cell\samples\util\debug\launcher



Remember to remove/disable support for the Debug Launcher when you complete your testing.

Testing the Game Launch Sequence on a Content Server

To test game launching, the Home Developer package must be installed on your PlayStation®3 reference tool or test kit. It must be installed for some Cell SDK functions required in your title to function correctly.

To perform final testing of your completed game launch object and Home-aware title, you must upload the object itself to a content server. To do this, you must:

- Package your Game Launch object from the Object Editor. See Packaging Objects.
- Upload the archived object using the Content Delivery System (CDS).

Then run the 'PlayStation®HomeDeveloper' icon from the XMB™ (i.e. the icon created when you installed Home.DeveloperBuild_.*.pkg). This build of Home client can connect to your personal sandbox on the content server that the CDS uploads to. Your object is downloaded from the content server and run automatically. You should now be able to:

- Create a game launch session
- Select your PlayStation®3 title
- Select your session options
- Launch the session
- Play the game
- Return to PS Home

Submitting the Game Launch Object to CQA for Publishing to the Live Environment

After Game Launch objects are published in the live environment, they are available to all users in the published regions. Users need only to have the title on their PlayStation®3, and then follow the Game Launching steps.

Publishing Checklist

Some items to check for before submitting the Game Launch Object to CQA:

- All debug settings have been removed from the object.
- (Recommended) The title that the Game Launch object will launch is already available.
- Localize all menus and options for each region in which each title ID is available.
- (Recommended) Add thumbnails. This is recommended since the process allows the object to take advantage of any options that may open up in the future to show the thumbnail.
- Add age ratings, and complete the header through the Object Editor.
- There can be only one Game Launch object per title. To update/change a title's game launch session options, edit and re-submit the existing Game Launch object. Speak to your Home Account Manager if you want to update a title's existing Game Launch object.

The HDK requires that you enter age rating data for Game Launch objects. If no age rating data is present when attempting to package, the Object Editor issues an error. The PS Home client uses the PS3 title's age rating. That is, if a player uses a Game Launch object for Fighter IV, client reads the PS3 title's age rating data and parental controls to see if the user can play the game, ignoring the Game Launch object's age rating data.

Despite this, continue to enter in age rating information into Game Launch objects, and make sure that the age rating data matches that of the PS3 titles that they launch.

PS Home Game Launching Menu

To launch a title from PS Home, the user selects the title, sets options, waits for other players, and then initiates a launch. Users select a game from the Game Launching menu. They can select this menu from the PS Home Menu Screen, or they can access it at any time by pressing START on the controller.

The Game Launching menu provides the following options:

- Set Up Game
- Launch Game
- View Player List
- Cancel Game

Set Up Game (Supported Game or Other Game)

The user can create the game from here. After a game is created, the user cannot create another game.

When the user selects Set Up Supported Game or Set Up Other Game:

1. The user is asked to choose the game they want to create. The user can choose from a list, populated by all installed games on the HDD and any present in the BD drive.

2. The user can then set the session options.
3. When the game is created, other players can join it. (The game creator is automatically included as the session master.)
4. A 'join game' option is added to the session master's targeting menu.
5. A game-specific icon appears above the session master's head.
6. The player list screen is displayed.

Session options are options specific to the game. They include the following PS Home session options:

- Number of players
- Number of private slots (see `GameLaunch.SetNumPrivateSlots()`)

Private slots can be occupied only by friends of the session master, or by users whom the session master has invited to joint the game. All other slots are public and can be occupied by anyone. Note that by private slots, it does not mean private slots within the game, but rather private slots within the context of a PS Home game launching session.

Launch Game

This option is available after the user has created the game. A new game prepares for launch when the session master selects Launch Game from the Game Launching menu, or from the Player list.

At launch there is a 10 second count-down. The count-down gives players time to opt out even if the session master chooses to launch the game manually as soon as it is created.

View Player List

This option is available when the user has created or joined a game. If selected, it displays the player list of any game the user has joined or created. The session master sees the player list as soon as they create a game. The other players see the list as soon as they have joined a game. Players can adjust their player options from the player list.

The session master can also:

- Launch game: Goes to the game launch phase.
- Cancel game: Abandons the game.

Anyone can close the screen and re-open it.

Cancel Game

Only the session master can cancel the game. Other users can leave it. The other players are informed If the game is canceled.

The game is canceled if the session master leaves the game. If a user involved in a game leaves the lobby in which the game was created, they are removed from the player list.

The Game Launch Object Component

To support game launching in PS Home for your title, you must create an object in the Object Editor and add the Game Launch component to it.

The Game Launch component also grants access to the following Lua libraries:

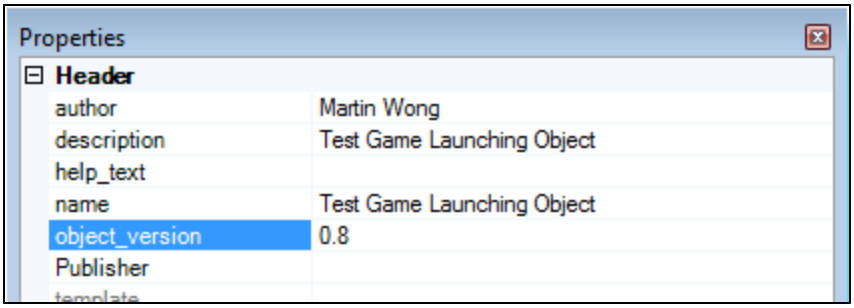
- `GameLaunch`
- `GameLaunchExporter`

Before You Start

Before reading this section you need to know how objects are built. See Objects.

Adding the Object Version to the Game Launch Header

A Game Launch object uses the `object_version` property in the object's Header to determine which version of the PlayStation®3 game title it supports.



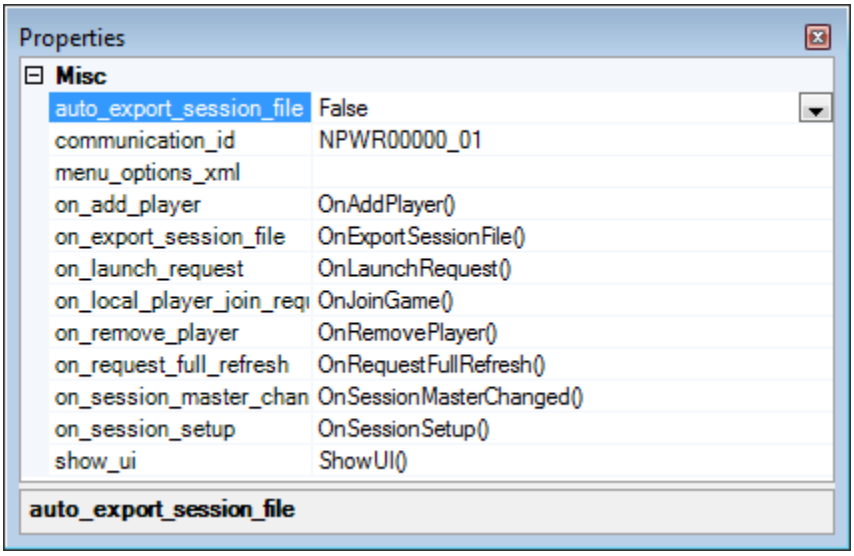
To add specify the version:

1. In the Object Editor, select the Header component.
2. In the Properties panel, enter the supported version in the object_version field.

For more information on the Header component, see Completing the Object Header.

Game Launch Component

The Game Launch component has a number of editable properties.



The component properties are Lua callbacks, except for the following properties:

- auto_export_session_file
- communication_id
- menu_options_xml

You define the functions for the callbacks in the object's script. The Game Launching System calls them in response to certain events, or to trigger certain actions.

> The Game Launch component callbacks work only until the title is launched.

Properties

You can set the following properties:

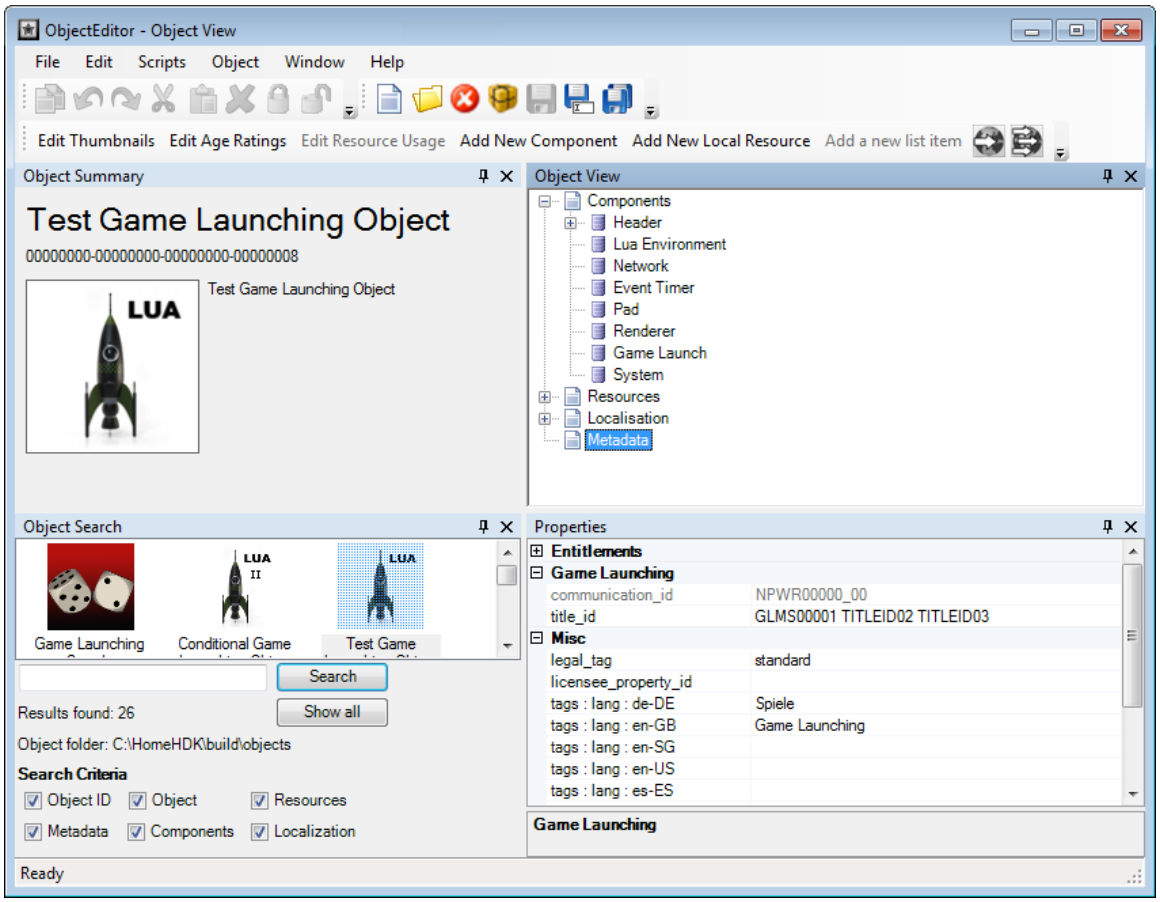
Property	Description
auto_export_session_file	A boolean to specify how PS Home creates the export session file when a game is launched. • Set to False in cases where Lua script creates and exports the session file (see on_export_session_file below). • Set to True if the standard game launching OSD is used, that is, the game launch object is created using the Static XML menu format or the GameLaunchingMenuSys APIs.

communication_id	The Communication ID of your title. A PlayStation® title can have multiple SKUs, but should only have one Communication ID per title.
menu_options_xml	Set this property only when using Static XML Game Launching. Specify the XML resource containing the Game Launch menu session options.
on_add_player	Specify which function to call when a new player is added to the session.
on_export_session_file	Use this function to write out any session data for the title to parse when a title launches. You can use the `GameLaunchExporter` library to achieve this. This callback is called just before PS Home exits to launch into a title.
on_launch_request	This callback is called only on the Session Master's object. If used, the callback is triggered when a player tries to launch a session. The function must call `GameLaunch.ReturnLaunchRequestResult()` to determine if the launch is permitted. If you do not specify this function in the Object Editor, sessions can be launched unconditionally.
on_local_player_join_request	Use this callback to determine if the player is able to join the game. It is triggered when the local player tries to join a session. When triggered, the local player's object is linked with the session master's object. This means that the local player and the session master can exchange information using the `OutboundMessage`, `ReceivedMessage`, and other messaging Lua functions. Before this function exits, a call must be made to `GameLaunch.ReturnJoinRequestResult()` to indicate whether or not the local player joined successfully. If the player fails to join, the object is immediately destroyed. You can use this callback to implement additional conditions for restricting player access to a game. For example: • The existence of entitlements • Queries to an external server via `HttpPostData` and `Resource.Request` • Queries to the session master's object about this player's eligibility for the joining the session If you do not use this callback, only the normal join conditions are applied to the session: number of slots, parental control levels, blocking, etc.
on_remove_player	This callback is called when a player is removed from the session.
on_request_full_refresh	This callback is called on the Session Master's object when a player has successfully joined the session. The function synchronizes all relevant session options to the enlisted player. You can use the `OutboundMessage` library for sending messages.
on_session_master_changed	Use this callback to update the state of the object when the Session Master changes. The Session Master can leave a session or elect another member to become Session Master. If you do not provide a function for this callback, the Game Launch session ends if the Session Master leaves. For objects with a Lua Environment component, you must specify this callback to indicate that Session Master Migration is supported.
on_session_setup	This callback is called at different times, depending on the role of the local player. For the Session Master, it is called when session setup begins, otherwise it is called when a player has successfully joined a session. Use this opportunity to initialize any data structures, or other values. When the UI has been set up, the UI displays only when called by the `ShowUI()` function. This function is called from the Game Launching System, when appropriate.

show_ui	This callback triggers the activation of the object's UI. It is called in response to selecting the Options button in the Playerlist menu. When this function is called, any reserved pad inputs are activated.

Game Launch Metadata

The Game Launch object has a number of metadata properties that you can set:



You can set the following properties:

Property	Description
communication_id	The Communication ID of your title. A PlayStation® title can have multiple SKUs, but typically has one Communication ID per title.
title_id	The Title ID of the title you want your object to launch. If you do not set this property correctly, your object will not activate when the user selects the corresponding title from the Game Selection dialog (which is opened when creating a new session via the Menu Screen). If your title has multiple SKUs with different Title IDs, you can enter multiple Title IDs separated by spaces, but make sure that the communication_id you have for each title_id is the same.

See also:

- Custom Lua Scripted Game Launching and the Lua API Reference
- Object Component Properties

Static XML Game Launching

This game launch method allows you to specify player options via an .xml file. It is a fast and easy way to implement menus (for additional functionality, the GameLaunchMenuSys APIs should be used).

> The static XML method of supporting game launching does not guarantee the order menus appear within the file, the developer should take this into account when parsing the file.

Game Launch Options XML Format

This is a generic menu description format allowing the description of game launch options for titles. It consists of 2 sections, <SESSION_OPTIONS> and <PLAYER_OPTIONS>. Each can contain an arbitrary number of menus. Menus can contain any of the item types listed below in any order but one menu within the <SESSION_OPTIONS> section must contain a <MAX_PLAYERS> tag.

Here is an example Game Launch Options XML file:

```
<HOME_GAMELAUNCH_DATA
xmlns="http://home.scedev.net/schema/GameLaunchOptions">
  <SESSION_OPTIONS>
    <MENU>
      <ID>SessionOptionsMenu</ID>
      <NAME_REF>SessionOptions</NAME_REF>

      <MAX_PLAYERS>
        <VALUE_INT>2</VALUE_INT>
        <VALUE_INT>8</VALUE_INT>
        <DEFAULT_INDEX>1</DEFAULT_INDEX>
      </MAX_PLAYERS>

      <!-- insert menu items here -->
    </MENU>

    <!-- more menus can be added here -->

  </SESSION_OPTIONS>

  <PLAYER_OPTIONS>
    <MENU>
      <ID>PlayerOptionsMenu</ID>
      <NAME_REF>PlayerOptions</NAME_REF>

      <!-- insert menu items here -->
    </MENU>

    <!-- more menus can be added here -->

  </PLAYER_OPTIONS>
</HOME_GAMELAUNCH_DATA>
```

Top Level Tags

<SESSION_OPTIONS>

This section contains the session option menus synchronized across all enlisted players, but controlled only by the session master.

<PLAYER_OPTIONS>

This section contains player option menus which are particular to each player. These options can only be changed by the local

player and only appears in the GAMELAUNCHSESSION.XML for that player.

Menu Common Tags

<ID>

ID identifying the menu, used for reference purposes and within the output GAMELAUNCHSESSION.XML file. This ID must be unique within the XML document.

<NAME_REF>

The reference string used to extract the menu's name from the game launching object's localization table. See the Lua function `Object.GetLocalizedText`.

Menu Tags

Menu items usually have a <DEFAULT_xxx> showing the item's initial value, these are optional. Not specifying a default will result in the first/smallest available option described.

<MAX_PLAYERS>

This section specifies the maximum number of players the session master wishes to have in game launching session and must exist somewhere within the <SESSION_OPTIONS> section of the XML document.

```
<MAX_PLAYERS>
  <VALUE_INT>2</VALUE_INT>
  <VALUE_INT>8</VALUE_INT>
  <DEFAULT_INDEX>1</DEFAULT_INDEX>
</MAX_PLAYERS>
```

Valid tags are as follows:

- <VALUE_INT>: Specifies an integer value in the list.
- <AUTO_POPULATE_INT>: Populates the list with values in the range given by the attributes begin and end.
- <DEFAULT_INDEX>: Specifies what initial list index the list should default to. If not specified, the list will default to the first item.

<ITEM_LIST>

There are two ways to specify content for these item types.

1. This is the comprehensive layout supported to allow for extensibility.

   Valid tags are as follows:

   - <TEXT_REF>: The reference string that is used to extract the item's localized name from the game launching object's localization table. See the Lua function `Object.GetLocalizedText`.
   - <TEXT_DIRECT>: This item type is designed to contain text that does not require localization. The contents of this tag will be displayed directly. Useful for items that only contain numerical values.

     For example:

```
<ITEM_LIST>
  <TEXT_REF>Track</TEXT_REF>
  <ITEM>
    <TEXT_REF>TrackA</TEXT_REF>
  </ITEM>
```

```
  <ITEM>
    <TEXT_REF>TrackB</TEXT_REF>
  </ITEM>
  <ITEM>
    <TEXT_REF>TrackC</TEXT_REF>
  </ITEM>

  <DEFAULT_INDEX>1</DEFAULT_INDEX>
  <!-- TrackB -->
</ITEM_LIST>

<ITEM_LIST>
  <TEXT_REF>NumTestInt</TEXT_REF>
  <ITEM>
    <TEXT_DIRECT>100</TEXT_DIRECT>
  </ITEM>
  <ITEM>
    <TEXT_DIRECT>200</TEXT_DIRECT>
  </ITEM>
  <DEFAULT_INDEX>0</DEFAULT_INDEX>
  <!-- 100 -->
</ITEM_LIST>

<ITEM_LIST>
  <TEXT_REF>NumTestFloat</TEXT_REF>
  <ITEM>
    <TEXT_DIRECT>1.0</TEXT_DIRECT>
  </ITEM>
  <ITEM>
    <TEXT_DIRECT>1.2</TEXT_DIRECT>
  </ITEM>
  <DEFAULT_INDEX>0</DEFAULT_INDEX>
```

```
  <!-- 1.0 -->
</ITEM_LIST>
```

2. This layout is shorter and can be used for convenience. It auto-populates the item list with items specified in the range given.

   For example:

```
<ITEM_LIST>
  <TEXT_REF>NumLaps</TEXT_REF>
  <AUTO_POPULATE_INT begin="1" end="10"/>
  <DEFAULT_INDEX>1</DEFAULT_INDEX>
</ITEM_LIST>
```

<ITEM_SLIDER_PERCENT>

This slider type is for integer percentage values. With this type, a min and max value can be specified.

- <DEFAULT_VALUE>: Specifies an optional default value. If not used, the value will default to the smallest valid value.

For example:

```
<ITEM_SLIDER_PERCENT>
  <TEXT_REF>Brightness</TEXT_REF>
  <RANGE begin="0" end="100"/>
  <DEFAULT_VALUE>70</DEFAULT_VALUE>
</ITEM_SLIDER_PERCENT>
```

Custom Lua Scripted Game Launching

This method of game launching is a flexible alternative to static XML game launching. Lua scripted game launching supports the following features:

- Custom UIs: Custom user interface using Lua's OSD API. See the Game Launching sample ui.lua and main.lua scripts.
- XML authoring via Lua: See Exporting GAMELAUNCHSESSION.XML below.

The GameLaunch and GameLaunchExporter Libraries

By adding the Game Launch component, your object has access to the following Lua libraries:

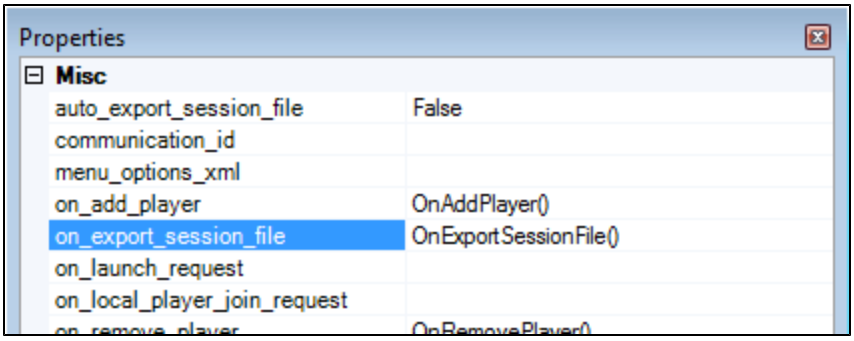
- **GameLaunch**: This library interfaces with the game launching system core.
- **GameLaunchExporter**: This library exports a file called GAMELAUNCHSESSION.XML. This file contains the player list, extra data added by the developer and identifies the session master.

Typically, both libraries are required to support a custom game launching process.

See the Game Launching samples' main.lua scripts.

Exporting a GAMELAUNCHSESSION.XML

To create and export a GAMELAUNCHSESSION.XML file from your Game Launch object, you need to implement the on_export_session_file callback in the properties for the Game Launch component in the Object Editor:



An example of this is shown below:

```
function OnExportSessionFile()

    GameLaunchExporter.BeginSessionExport()

        GameLaunchExporter.BeginSection("SESSION_OPTIONS")

            GameLaunchExporter.BeginSection("SessionOptionsMenu")
            GameLaunchExporter.ExportOption("MAX_PLAYERS", "10")
            GameLaunchExporter.ExportOption("Track", "Grizzly")
            GameLaunchExporter.ExportOption("NumLaps", "2")
            GameLaunchExporter.ExportOption("CatchUp", "On")
            GameLaunchExporter.ExportOption("MirrorMode", "Off")

        GameLaunchExporter.EndSection()
        GameLaunchExporter.EndSection()

    GameLaunchExporter.EndSessionExport()

end
```

This generates output to the following in the GAMELAUNCHSESSION.XML:

```
<HOME_GAMELAUNCH_SESSION>

  <PLAYER_LIST>
    <PLAYER sessionMaster="true">PlayerA</PLAYER>
    <PLAYER>PlayerB</PLAYER>
  </PLAYER_LIST>

  <SESSION_OPTIONS>

    <SessionOptionsMenu>
      <MAX_PLAYERS>10</MAX_PLAYERS>
      <Track>Grizzly</Track>
      <NumLaps>2</NumLaps>
      <CatchUp>On</CatchUp>
      <MirrorMode>Off</MirrorMode>
    </SessionOptionsMenu>

  </SESSION_OPTIONS>
  --<PLAYER_OPTIONS>

    -- Add player options here.

  --</PLAYER_OPTIONS>
</HOME_GAMELAUNCH_SESSION>
```

Bear in mind the following:

- The session master is indicated within <PLAYER_LIST> in the form of a sessionMaster="true" attribute within the <PLAYER> tag. The session can be launched without waiting for the session master.
- All menus that contain session options will be included in addition to the local player's options.
- The session options and player options are specific to each title.
- Tag names are identical to the ones described in GAMELAUNCHOPTIONS.XML.

The static XML method of supporting game launching does not guarantee the order of menus within the file, the developer should take this into account when parsing the file.

Supporting Game Launching in your Title

To add support for game launching to your title's code-base, the title must detect how it was launched.

- If PS Home launched the title, then the title should find a session options XML file that is written to the PlayStation®3's hard disk when the game is launched. This is the mechanism via which PS Home passes game launch parameters to the title being launched.
- If PS Home did not launch the title, it should start up as normal.



The HDK only supports game launching for titles compiled with Cell SDK 2.4.0 and above.

Detecting how your Title was Launched

The Game Exec library in the Cell SDK provides two functions that can detect how a title was launched. These functions are:

```
int cellGameGetBootGameInfo( unsigned int *type,

    char *dirName,

    unsigned int *execdata ) ;

int cellGameGetHomeLaunchOptionPath( char *commonPath, char *personalPath );
```

The following code extract shows how these functions can detect if PS Home launched the running executable:

```
#include <sysutil_game_exec.h>

void CheckHowGameWasLaunched( void )

{

    int ret = CELL_GAME_RET_OK;

    unsigned int type = CELL_GAME_GAMETYPE_SYS;

    char dirName[CELL_GAME_DIRNAME_SIZE] ;

    unsigned int execdata = 0;

    // Find out where this .self was launched from

    ret = cellGameGetBootGameInfo( &type, dirName, &execdata );

    if (ret != CELL_GAME_RET_OK)

    {

        PRINTF("cellGameGetBootGameInfo: Failure.\n ");

        return;

    }

    if (type == CELL_GAME_GAMETYPE_HOME)

    {

        char gamePath[CELL_GAME_HDDGAMEPATH_SIZE];

        char gamePathPersonal[CELL_GAME_HDDGAMEPATH_SIZE];

        char launchOptionsPath[CELL_GAME_HDDGAMEPATH_SIZE + 32];

        PRINTF("Executable was launched from Home\n" );

        PRINTF("Skip normal start-up sequence and load launch options.\n\n" );
```

```
    ret = cellGameGetHomeLaunchOptionPath(gamePath, gamePathPersonal ) ;
    if ( ret != CELL_GAME_RET_OK )
    {

        PRINTF("cellGameGetHomeLaunchOptionPath: Failure.\n ");
        return;

    }

    // Get the name of the XML file that will contain the options
    // the user(s) set while in Home.
    //If using System.WriteGameLaunchXML(), check for a SCRIPTEDGAMELAUNCHSESSION.XML file.
    sprintf( launchOptionsPath, "%s/GAMELAUNCHSESSION.XML", gamePath );

    PRINTF("GameLaunch XML path =\n %s\n\n", launchOptionsPath );
    LoadGameLaunchXML( launchOptionsPath );

}
else
{
PRINTF("Executable was not launched from Home.\n" );
PRINTF("Game should start normally.\n\n" );
}
```

```
}
```

Launching From Mini-games

You can launch an external application from PS Home without using the session management facilities provided by Home's Game Launching objects.

To launch from mini-games:

- Add the Game Launch Properties component to the object in the Object Editor.
- Use the function System.ExecTitle() in the script. Provide a callback to handle launch fails.
- Use the function System.WriteScriptedGameLaunchXML() to pass parameters to the title.

For example:

```
function LaunchTitle()

    local gameOptions = 0

    local titleId = "XXXX12345"

    System.ExecTitle( titleID, gameOptions, "MyLaunchErrorCallback()" )

end

function MyLaunchErrorCallback()

    local errorCode = System.GetExecTitleErrorCode()

    -- Handle error here

end
```

Error codes

You may encounter one of the following error codes:

- 1 - unknown error
- 2 - parental control level restricted

Game Launching Samples

The HDK installer includes samples showing how to implement custom game launching in PS Home. This installer consists of two parts:

- The Game Launch objects that renders a user interface and allows the user to select session options before launching into the title.
- A runtime sample coded in C, which demonstrates how to write a PlayStation®3 title that supports launch sequences that originate from the XMB™ or PS Home.

The Game Launching objects are:

Test Game Launching Object

- Physical location: <HDK_INSTALL_DIR>\Build\objects\00000000-00000000-00000000-00000008
- Sample Package: <HDK_INSTALL_DIR>\Samples\GameLaunchSample\GameLaunchSample.pkg

Game Launch Static and Lua Menus Object

Game Launch object using a combination of Static and Lua menus:

- Physical location: <HDK_INSTALL_DIR>\Build\objects\00000000-00000000-00000000-00000017
- Sample Package: <HDK_INSTALL_DIR>\Samples\GameLaunchSample\GameLaunchSample.MenuSys.pkg

Game Launch With OSD Object

Game Launch object with custom menus build using the OSD libraries:

- Physical location: <HDK_INSTALL_DIR>\Build\objects\00000000-00000000-00000000-00000018
- Sample Package: <HDK_INSTALL_DIR>\Samples\GameLaunchSample\GameLaunchSample.OsdAPI.pkg

Launching the Samples

Running PS Home from the XMB™ in Debug Station mode

Go to the XMB™ and select Game > Debugging Station Launcher > Debugging Station Launcher.

This runs PS Home from XMB™ using the debugging station launcher. This is the mode in which you can test Home game launch objects by launching a sample (and cut-down example of a PlayStation®3 game).

Running the Game Launch Object

When you are running PS Home, you can test the Game Launch object by running it from the Game Launching menu:

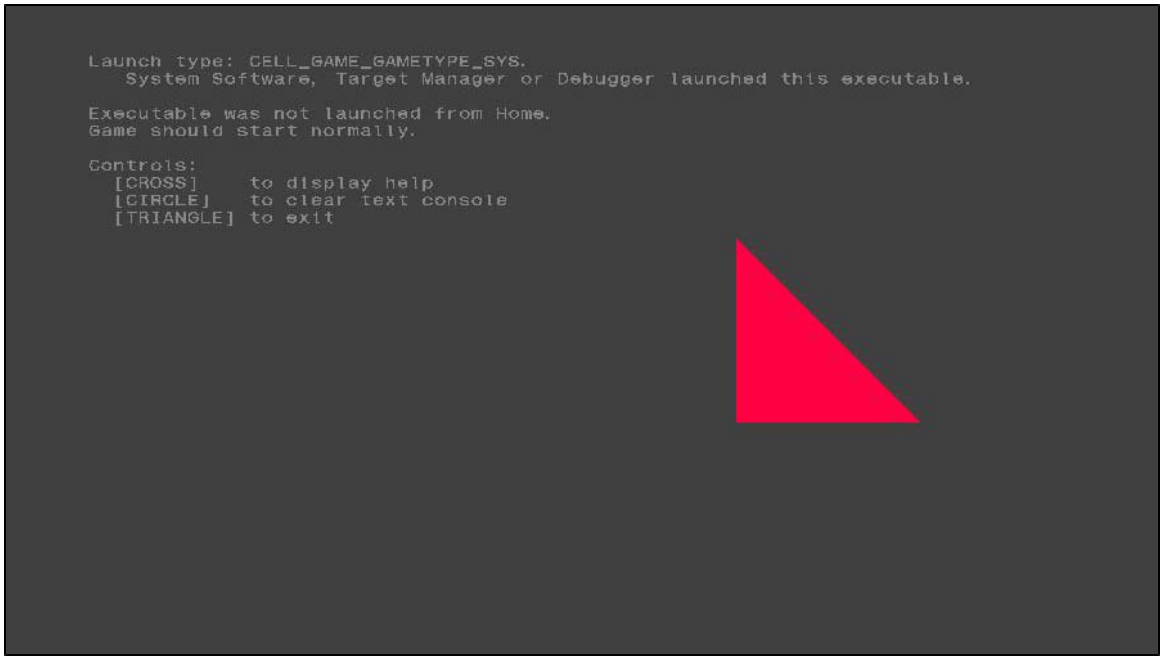
To run the Game Launch object:

1. On the Menu Screen, click the Social icon  to display the Social Menu:



2. Select Game Launching.

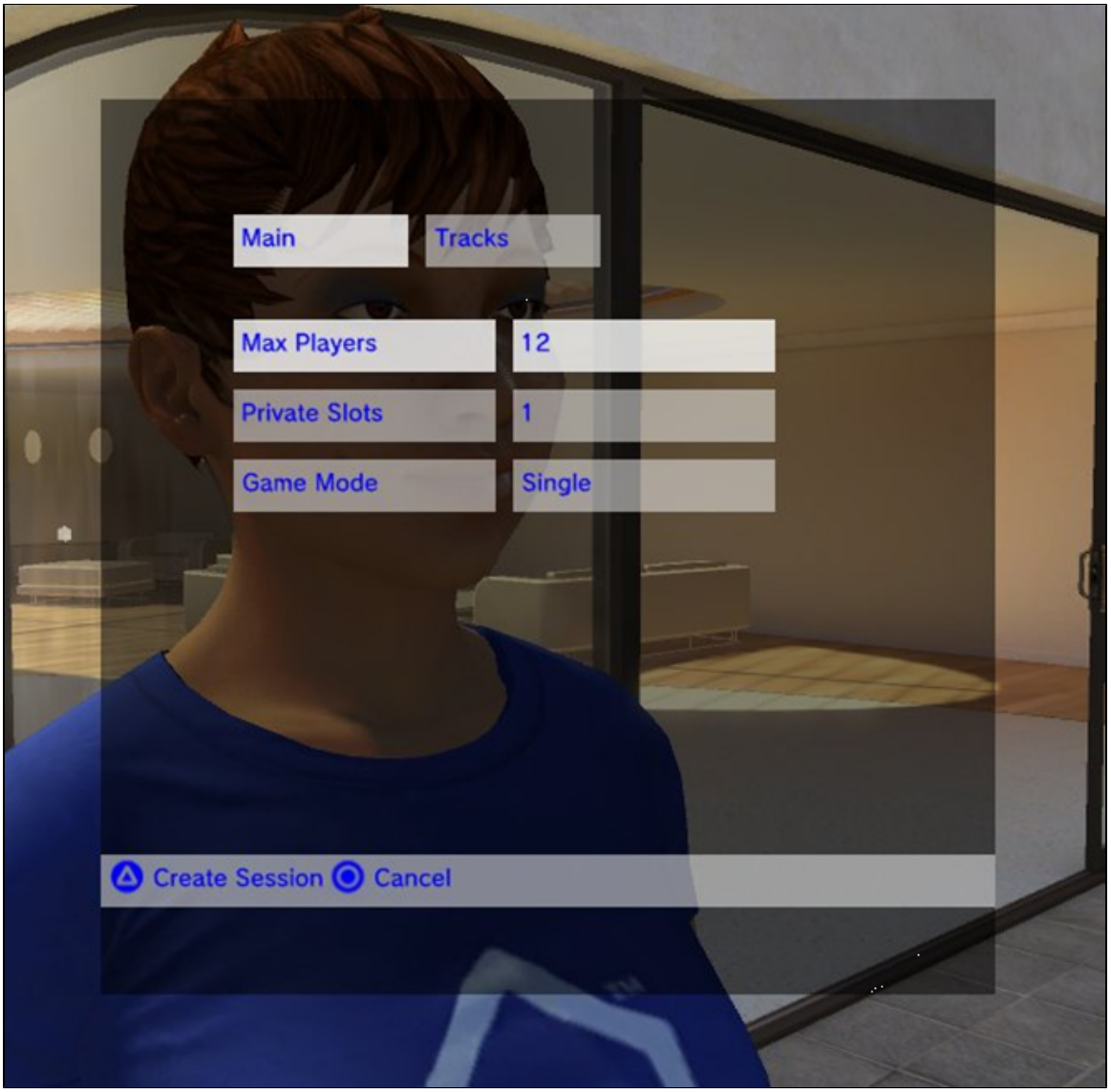
If you run the sample title from the XMB™, Target Manager, or the Debugger, you see the following information:



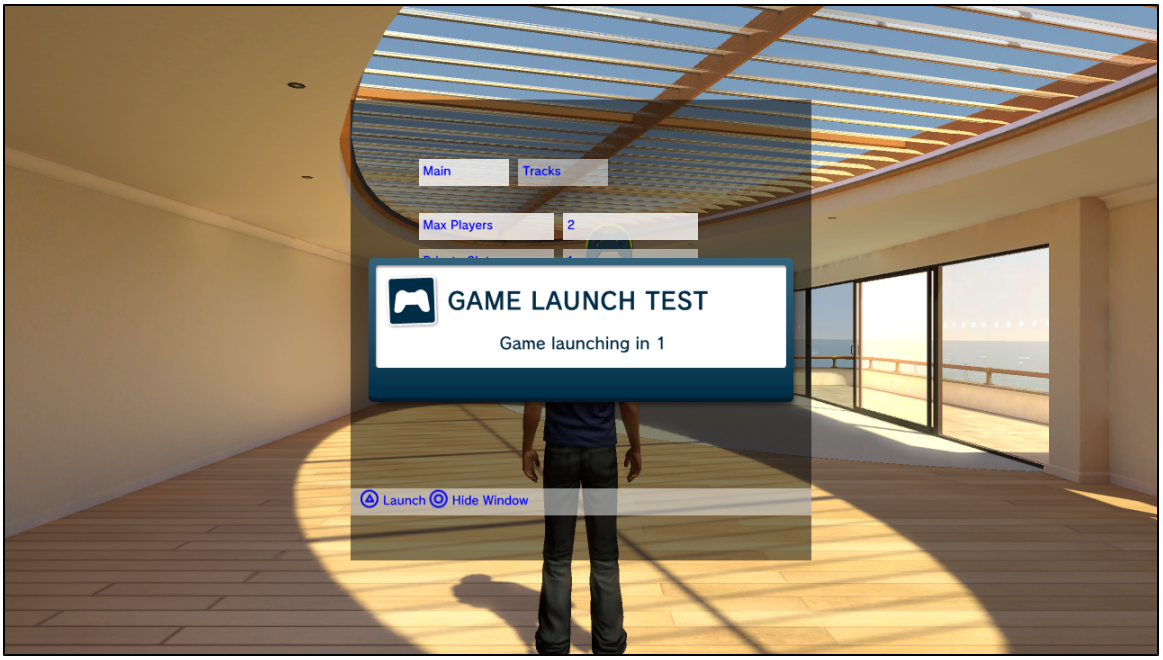
Notice that the title recognizes that it was not launched from PS Home.

Testing the Game Launch Object

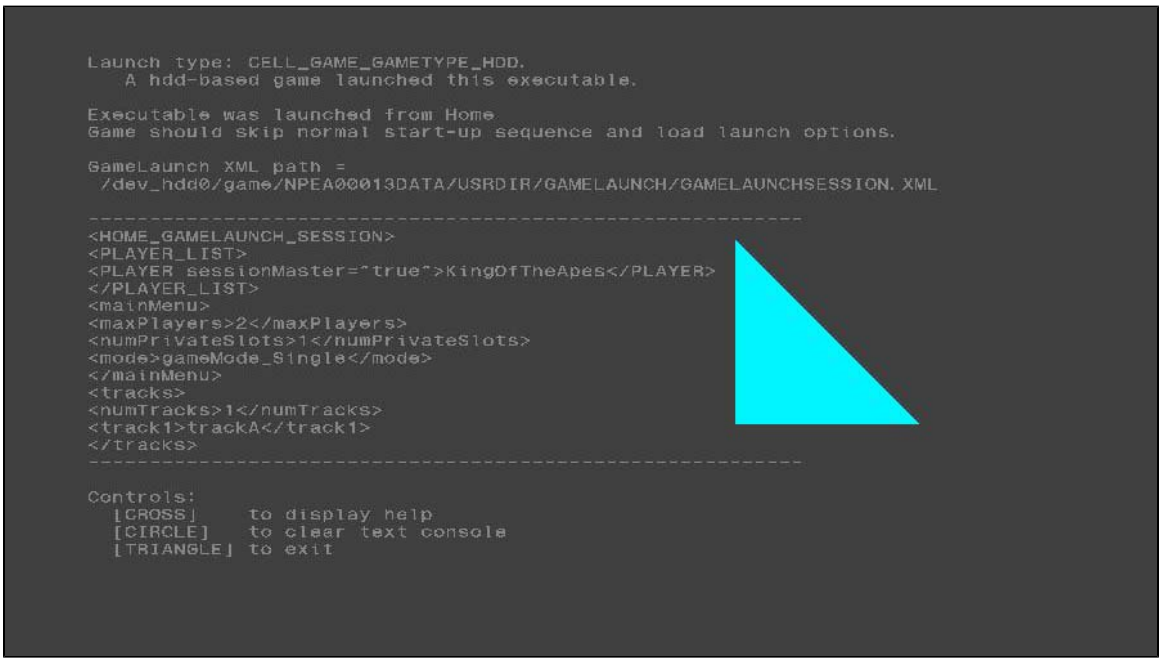
This sample demonstrates Lua scripted game launching. Scripted game launch objects have a Lua component and give you greater control over how and what data is passed to the launched game on execution. You can change the components in the running example to see their effect on the target app:



1. Press [TRIANGLE] to create a session. A message is displayed telling you that the session is created, and some blue text next to an icon depicting a control pad is displayed over your avatar's head.
2. Either wait for other players to join, or continue to the actual launch.
3. On the Launch Options dialog, press [TRIANGLE] to launch the session. PS Home counts down from 10 to 0:

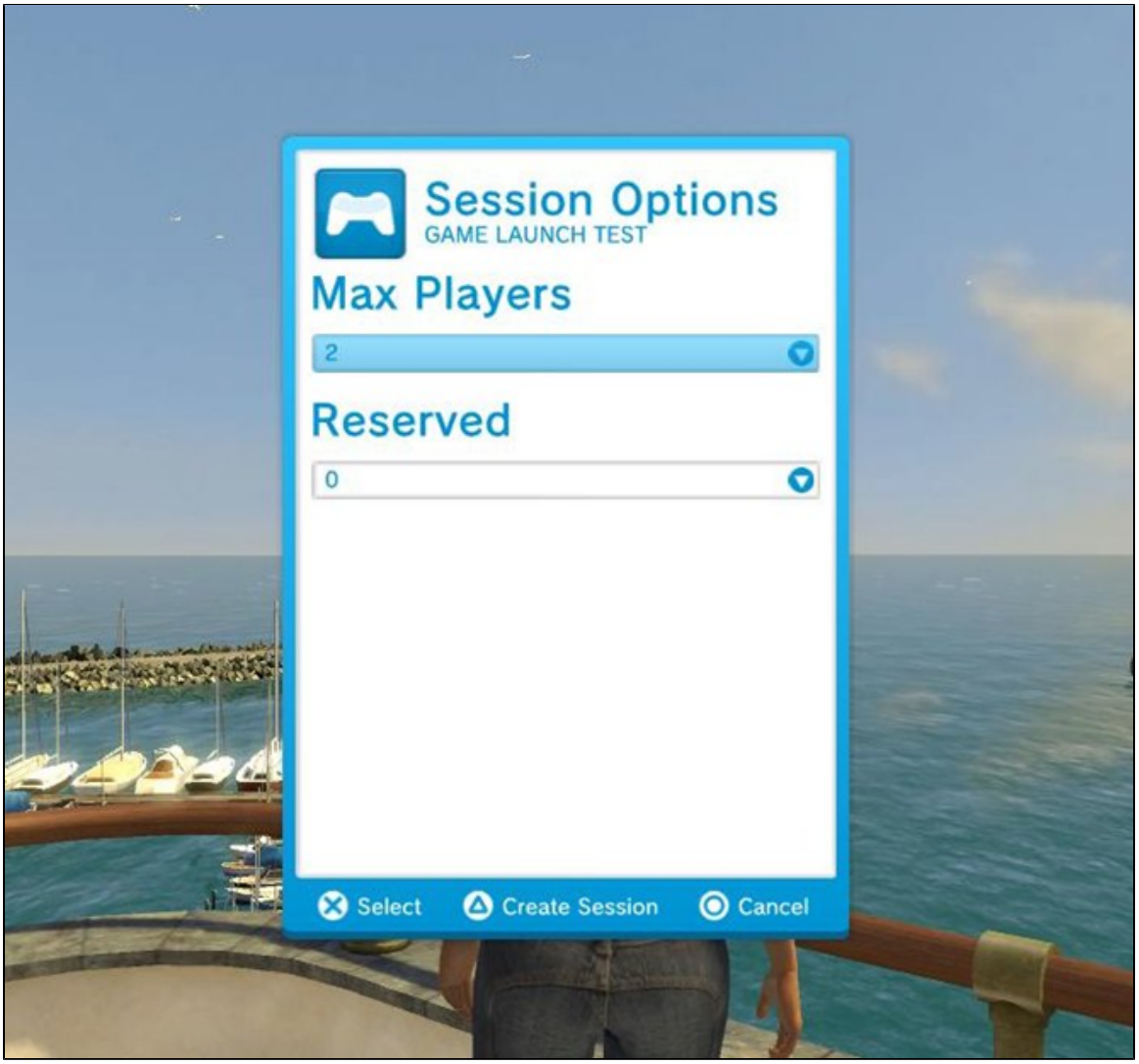


When the counter reaches 0, PS Home calls the Lua script with `OnExportSessionFile()`. This serializes the session options chosen in the menus to the PlayStation®3's hard drive. The `GAME LAUNCH TEST` title is then launched and you should see the following:



Notice that the sample title now recognizes that it was launched by PS Home, and prints the session options to the screen.

Game Launch Test



The Game Launch Object with a Static XML only allows you to set the maximum players and the reserved number. From here you can create the session and then launch the game.

The result is similar to the Lua launch example, but only the max players and reserved number are referenced in the application.

Game Launching FAQs

What is the size limit on game launch objects?

1 megabyte VRAM, 1 megabyte Main and 128 KB Host (these values are correct for 1.60 - for more information, see Memory Limits).

I have more than one Title ID, how do I make my game object work with all of them?

In the Object Editor, your game launch object has a Title ID section in its Metadata. Simply add your Title IDs here, separated by spaces.

Can games with different Title IDs join the same session?

Yes, Title IDs often identify a title's region, the communication ID ensures that the client apps can see each other over the PSN. You can set conditions in your game launch objects to only allow certain regions to communicate with each other if desired.

Where are Communication IDs from and what do they do?

Communication IDs are assigned by SCE and are used by the PSN to identify games online. Where Title IDs can differ between regions for a particular title, a Communication ID is the same for that title across all regions. They may identify a group of games that have differing Title IDs.

Where does the game launch object get the title logo?

Game launch objects use the icon.png of the title for launch. The icon.png is explained in the PS3 Developer Technical Requirements Checklist (TRC).

Can I launch a game without a game launch object?

Yes. From a mini-game's Lua script, you can call `System.ExecTitle()`. This launches the game but it does not provide session management, you have to handle that within your script.

Our uploaded game launch object looks fine, works locally, but doesn't work from our sandbox.

Make sure that only one game launch object for the title is uploaded to your sandbox. If you upload more than one, the client may download the wrong object. When submitting modifications to game launch objects, make sure that you use the same UID as for previous submissions.

Portable Items

A Portable Item is a model and animation which you can deploy in a scene. The object will stay where it is placed until the user removes it or leaves the scene and will react to players in various ways.



Portable Items support the following features:

- Custom model
- Configurable proximity and interaction states
- Custom animations for each different state (idle, proximity and activated)
- Custom rotation options for each state

For a step-by-step guide on creating a new Portable Item see Creating Portable Items.

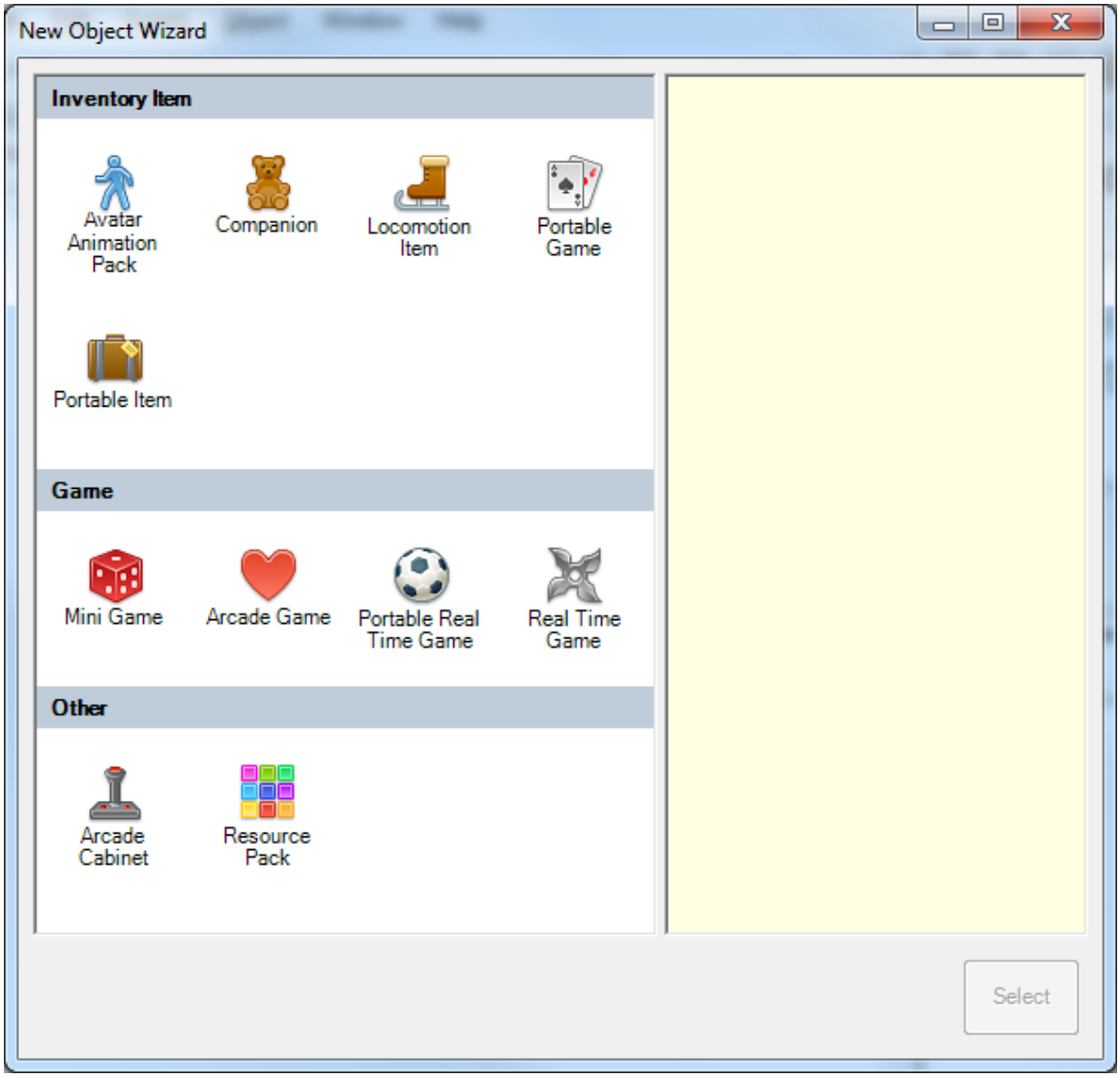
For details on how to configure and customize Portable Item see Configuring Portable Items.

Creating Portable Items

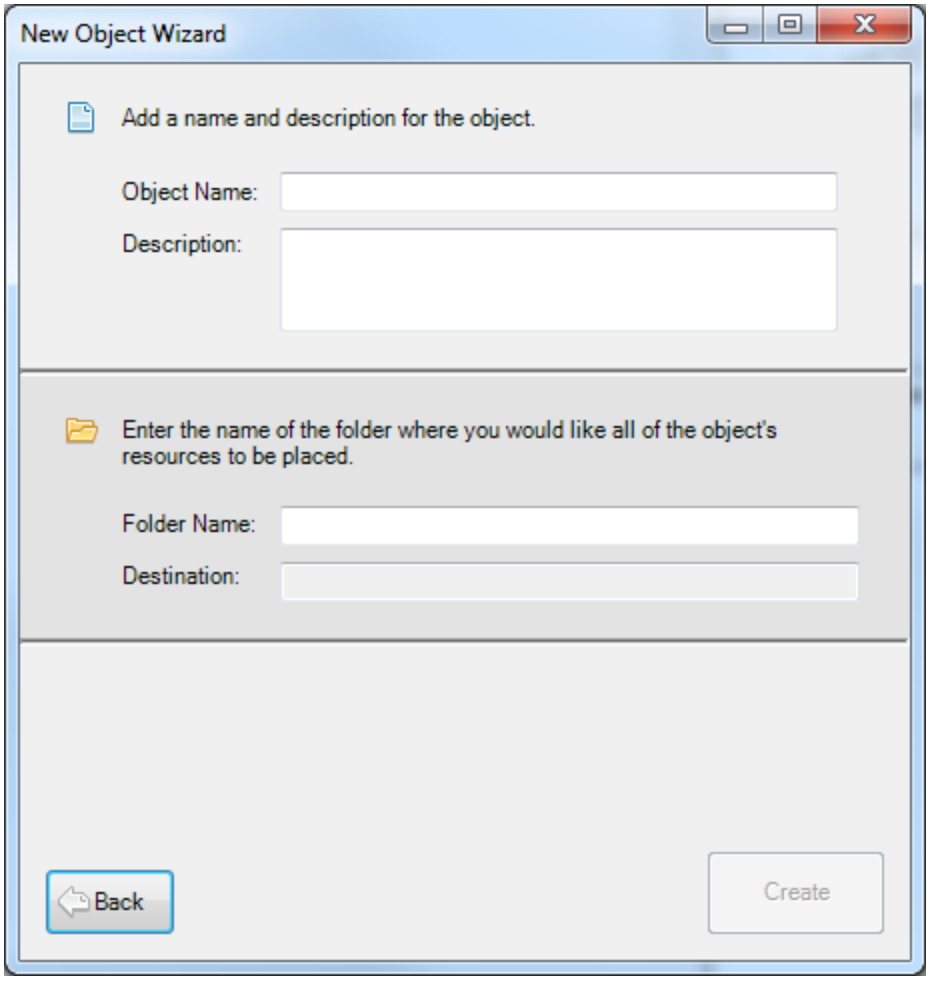
Creating a Portable Item

To create a new Portable Item

1. In the Object Editor, select File > New Object Wizard.
2. In the Object Creation Wizard dialog, select Portable Item and click Select.



1. Enter the name of the Portable Item, and provide a description



1. Click Create to complete the wizard, a new Portable Item is created at: <HDK_INSTALL_DIR>/build/PortableItems/<Portable Resource Folder Name>/.

This folder initially contains all the default template assets that are required for the Portable Item to work (i.e. the model and animation files listed previously). You must replace these files with your own. The folder also includes all the other assets required for the Portable Item to work in PS Home.

Preview the Portable Item in PS Home.

- To launch into the HomeDeveloper.self, see Launching the Home Developer SELF.
- To add items to your inventory in the HomeDeveloper.self, see The DEV DEBUG Menu.

Configuring Portable Items

This is an in-depth guide to configuring Portable Items.

You can configure a Portable Item through the config.xml file, which is referenced in the object's resources.

This page details the customizations that can be made in this configuration.

Config Specs

labelOffset

Specifies how high above the floor the object's name label should appear.

Attrib	Mandatory	Default Value	Notes
x	yes	0	how far from the base of your item your label should be on the 'x' axis

y	yes	1	how far from the base of your item your label should be on the 'y' axis
z	yes	0	how far from the base of your item your label should be on the 'z' axis

local_model

Specifies the model to use for the local portable.

Attrib	Mandatory	Default Value	Notes
name	yes	"SentryMdl"	the name of the resource to load

remote_model

Specifies the model to use for the remote portable - his can be the same as the local, or one with lower detail.

Attrib	Mandatory	Default Value	Notes
name	yes	"SentryMdl"	the name of the resource to load

skeleton

Specifies the skeleton to use when animating your model

Attrib	Mandatory	Default Value	Notes
name	no	"SentrySkn"	the name of the resource to load

proximity_radius

Specifies the radius at which your portable will enter it's 'proximity' state.

Attrib	Mandatory	Default Value	Notes
radius	no	2.5	the radius in metres

idle

This specifies the behaviour your item will exhibit when in an idle state

Attrib	Mandatory	Default Value	Notes
anim	no	IdleAnim	the name of the animation resource to play
loop	no	true	should the animation loop until a new state with an animation is activated (true,false)
face	no	false	should the model face the player during this state (true,false)
text	no	kOffText	the name of a piece of localized text to display in the 'Activate' legend while in this state

proximity

This specifies the behaviour your item will exhibit when the local player is within the 'proximity_radius' specified above

Attrib	Mandatory	Default Value	Notes
anim	no	ProximAnim	the name of the animation resource to play
loop	no	false	should the animation loop until a new state with an animation is activated (true,false)
face	no	true	should the model face the player during this state (true,false)
text	no	kOffText	the name of a piece of localized text to display in the 'Activate' legend while in this state

activated

This specifies the behaviour your item will exhibit when approached and activated by pressing 'x'

Attrib	Mandatory	Default Value	Notes
anim	no	AlertAnim	the name of the animation resource to play
loop	no	false	should the animation loop until a new state with an animation is activated (true,false)
face	no	false	should the model face the player during this state (true,false)
text	no	kOnText	the name of a piece of localized text to display in the 'Activate' legend while in this state

Avatar Interaction Packs



Overview

The Avatar Interaction Pack portable allow interactions to take place between 2 players. An interaction consists of an action or behaviour for both avatars involved in the interaction, which will be started at the same time so the animations can be synchronised. The avatars can be set to equal height, allowing the animations to be vertically aligned for the avatars, making physical contact interactions possible.

Information

An interactions pack can contain 1 or more interactions. When multiple interactions are defined, the portable owner can press SQUARE to bring up a menu to make themselves available for one of the interactions. If there is only 1 interaction in the portable then they are always 'interactable' while they have the portable active.

While they are in an 'interactable' state, the portable owner's status text is changed to indicate that they can be interacted with, and a menu option is added to the owner's target menu for the remote players in the scene. The text for both of these is defined by the developer of the portable.

When a remote player chooses the option to interact with the portable owner, both players are translated to the same position for the interaction to begin. The animations will need to be created with this in mind (the X,Z origin being in the middle of the 2 players).

There is an option in the interaction configuration file to set both players to an equal height (half way between their heights) for the interaction. This will be required all interactions where there is physical contact between the 2 avatars so that the animations are correctly vertically aligned. A female rig scaled up to be the same height as the male rig in Maya is available to make it easier to create animations suitable for when the avatars are set to matching heights in the Home client.

An interaction can consist of either a repertoire action or behaviour for each player. A separate action/behaviour can be specified for each player, or both players can do the same action/behaviour and the option to apply a rotation offset to guest player in the configuration file can be used.

All interactions must be possible between all gender combinations (male &male, male & female, female & female). This means that the owner animation must be exactly the same for both male and female repertoires, likewise for the guest player. For example, if we are creating an interaction where the owner taps the guest on the shoulder, then a 'Give Tap' and 'Receive Tap' action will need to be created for both male and female repertoires and the both actions will need to be animated the same for both genders.

For behaviour interactions, the interaction will be stopped for both players when either player moves or performs a different emote. For action interactions, the interaction will stop when both players have finished the action (so it is advisable that the actions are the same duration for both owner and guest). When the interaction is stopped, both players are unlocked and set back to their position before the interaction began and set back to their original height (if the option to set them to equal height is enabled).

For a step-by-step guide on creating a new Avatar Interaction Pack see Creating Avatar Interaction Packs.

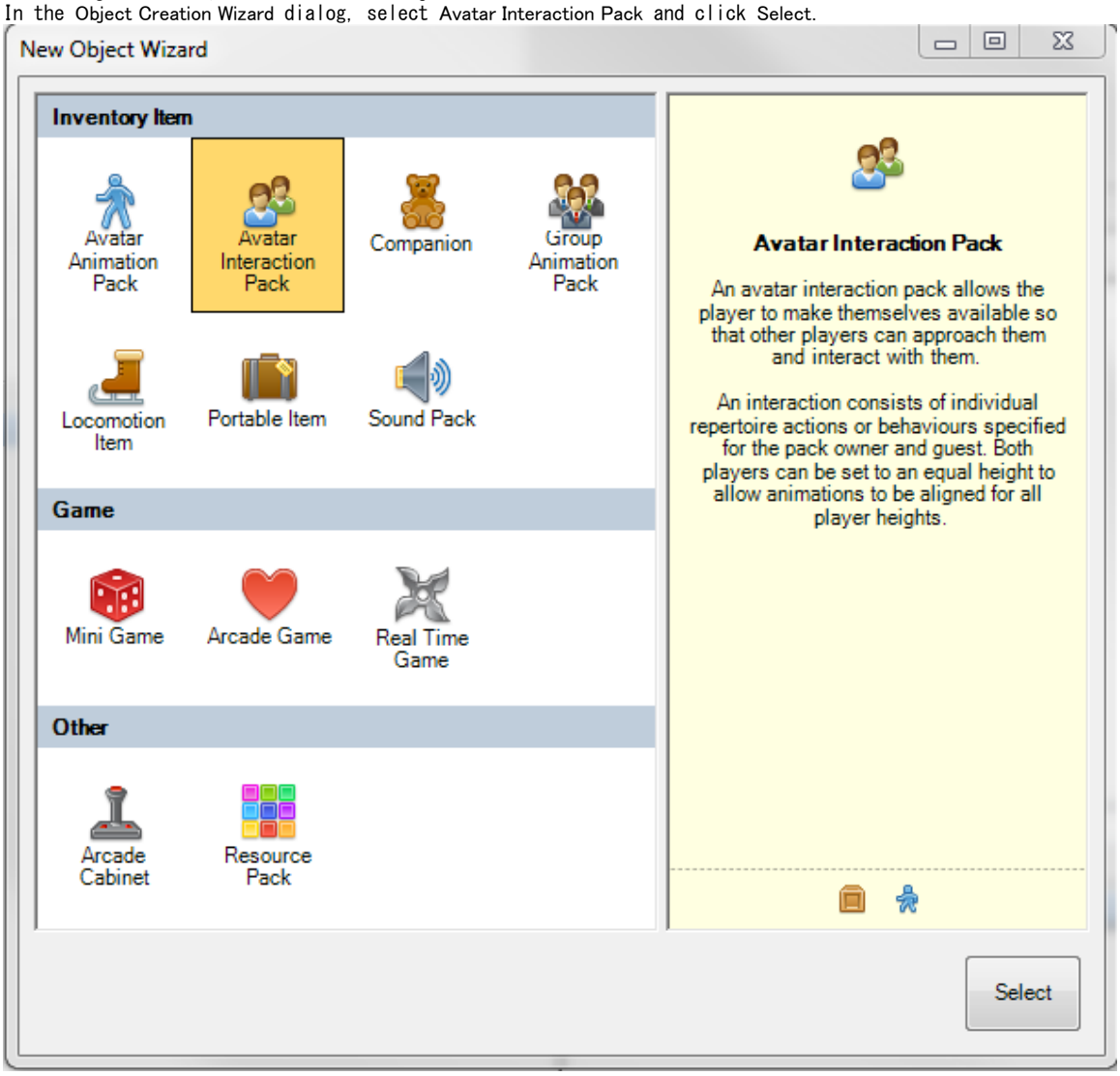
For details on how to configure and customize Avatar Interaction Packs see Configuring Avatar Interaction Packs.

Creating Avatar Interaction Packs

Creating an Avatar Interaction Pack

To create a new Avatar Interaction Pack

1. In the Object Editor, select File > New Object Wizard.
2. In the Object Creation Wizard dialog, select Avatar Interaction Pack and click Select.



3. Enter the name of the Avatar Interaction Pack, and provide a description
4. Click Create to complete the wizard, a new Avatar Interaction Pack is created at: <HDK_INSTALL_DIR>/build/AvatarInteractionPacks/<Resource Folder Name>/.

This folder initially contains all the default template assets that are required for the object to work. You must replace these files with your own. The folder also includes all the other assets required for the object to work in PS Home.

Preview the object in PS Home.

- To launch into the HomeDeveloper.self, see Launching the Home Developer SELF.
- To add items to your inventory in the HomeDeveloper.self, see The DEV DEBUG Menu.

Configuring Avatar Interaction Packs

After creating an interactions portable object there are 3 main steps to making your own interaction. These are:

1. Create animations for the interaction in Maya HDK

For interactions where there will be physical contact between the avatars we need to take into account that the avatars will be adjusted to be the same height in the client. The default female rig in Maya HDK is shorter than the default male rig so we will need to use a special female rig which has been scaled up to be the same height as the male rig. This allows for the animations to be

Note: Creating the animations with the scaled up female rig does not actually affect the height of the rig when the animation is played. Setting the 'set_to_equal_height' attribute to 'true' in the configuration.xml for the interaction causes this to happen. The scaled up rig is just to make it possible to animate the male and female rigs side by side.

Animations can be created with the scaled up female rig in Maya HDK by:

1. Launch Maya HDK 2012, 2013 or 2013.5 (use of the custom rig is not support in Maya 2010)
2. Select File > New Home Avatar Animation
3. Select Female (avatar interactions only)
4. Specify a name for your avatar animation and click Create

To view you guest animation files alongside each other you can create a reference to the other animation by going to File > Create Reference and selecting the other file that you want to view.

More information on creating animations in Maya HDK can be found in at Avatar Animation Tools

> Animation Rotation
>
> When you come to export your animations from Maya, they will probably be facing different directions if you have been viewing the different animations alongside each other. Before exporting, the starting position of the avatar in all animations should be facing the front. This is to ensure that the players will not see their avatar jump to a different rotation when the interaction begins.
>
> If you have only set a rotation on the PelvisRoot node, then it will likely just be a case of setting the Y rotation of PelvisRoot to 0 for all animations. If the rotation has been applied in a child node then you will need to set the Y rotation of PelvisRoot compensate for this, so that avatar is facing the FRONT direction.
>
> The rotation offset between the owner and guest can then be set on the 'ry' attribute of the 'guest_anim' node (detailed below).

2. Set up the repertoires for the interaction in the Repertoire Editor

Once you have created the animations for the interaction they need to be added to the repertoires of the object. See the following documents on adding Repertoire Actions and Repertoire Behaviours to your object

For behaviours created in the Repertoire Editor, it is important that they are set up to allow the user to exit by moving the left analogue stick. There is an example of how to set this up in the 'Looping Behavior' section of the following article: Repertoires - Output Nodes Examples

If you would like to include an intro animation to play once before the looping behaviour then here is an example of how that can be achieved: Repertoire - Example Conditions

The default Avatar Animation Pack also uses this technique similarly to achieve a behaviour with an intro and then a looping animation.

3. Configure the interaction in the configuration.xml resource file

You can configure an Avatar Interactions object through the configuration.xml file, which is referenced in the object's resources. At the base level of the xml file we'll have the following:

```
<xml>
 <interactions>
  <!-- 1 to 5 interactions inside here -->
 </interactions>
</xml>
```

Then inside the 'interactions' element we have between 1 and 5 interaction elements.

Example

Let's have a look at an example of how an interaction is defined:

```
<interaction remote_menu_name="kBallroomDance" status_text="kComeAndDance" safe_zone_radius="1">
 <player_anims anim_type="behaviour" set_to_equal_height="true">
  <owner_anim name="Dance_Lead"/>
  <guest_anim name="Dance_Follow" ry="0"/>
 </player_anims>
</interaction>
```

Here we can see that:

- The interaction uses repertoire behaviours (anim_type="behaviour")
- Noth players will be set to equal height (set_to_equal_height="true")
- The owner's behaviour will be set to 'Dance_Lead' and the guest will be set to 'Dance_Follow', so the male and female Dance_Lead animations will need to be the same, and the male and female Dance_Follow animations will need to be the same.
- The 'ry' property of the guest element is for the Y rotation offset of the guest from the owner, in this case it is set the default 0 as we are using different behaviours for the owner and guest and those animations are already correctly aligned with each other. Depending on the nature of the animation, it can be best to apply no rotation to the player in the animation itself and use the ry value to set the guest offset, this means that the local player (either owner or guest) will not jump to a new position when the animation starts.
- The localisation keys for option added to the target menu of the owner (remote_menu_name) and the status text of the owner while they are interactable (status_text).
- There is also an optional safe zone radius meaning the radius of free space around the players required for the interaction to be allowed (safe_zone_radius).

Config specs

Here are the specifications of each of the different nodes in the avatar interaction pack configuration.xml:

interaction

Specifies a single interaction to include in the portable

attrib	mandatory	default value	notes
owner_menu_name	Yes	none	The localisation key for the name of the interaction in the owner's selection menu (only mandatory if there is more than 1 interaction)
remote_menu_name	Yes	none	The localisation key for the text of the button that appears in the owner's target menu for remote players
status_text	Yes	none	The localisation key for the status text of the owner while they are available to interact
safe_zone_radius	No	0.75	The radius in metres of how much space there needs to be around the owner and guest for the interaction to be allowed to take place.

player_anims

Contains the configuration for the owner and guest animation

attrib	mandatory	default value	notes
anim_type	Yes	none	Either "action" or "behaviour". Whether the animations are set up as actions or behaviour in the Repertoire Editor
set_to_equal_height	Yes	true	Whether both avatars should be set to an equal height, allowing animations with physical contact.

owner_anim

Specifies the repertoire action or behaviour for the owner

attrib	mandatory	default value	notes
name	Yes	none	The name of the action or behaviour for the owner, as it appears in the Repertoire Editor

guest_anim

Specifies the repertoire action or behaviour for the guest player

attrib	mandatory	default value	notes
name	Yes	none	The name of the action or behaviour for the guest player, as it appears in the Repertoire Editor
ry	No	0	The number of degrees that the guest should be rotated in the Y axis from the guest. Generally should only be required if the same action/behaviour is being used for both owner and guest.

Limits and validations

The following restrictions need to be followed for the configuration file:

- ry: -360 to 360
- safe_zone_radius: 0.75 to 3
- interaction: a maximum of 5 interactions nodes are allowed in a single interactions portable
- owner: only 1 per interaction
- guest: only 1 per interaction

Guidelines:

- Emotes should not be included in the repertoires of the interaction portable as they are not needed, and the repertoires are added to the guest player while the interaction is taking place.
- Output nodes must be added to behaviours to allow the players to exit the interaction by moving.
- Action interactions should have animations of the same length for both owner and guest, otherwise the longer animation will not complete
- Ideally animations should not set a different starting rotation for a player. The ry (y rotation offset) attribute of the guest element can be use to set the players to face opposing directions (by setting it to 180), this should mean that for the owner or the joining guest player, they will not see themselves jump to a new position.

Group Animation Packs

Overview

A Group Animation Pack allows 1 or more players to join the portable owner in performing repertoire behaviours in a developer defined formation.

The pack owner puts themselves into a ‘waiting’ state by selecting a specific behaviour from the group animations menu (accessible by pressing SQUARE). This then deploys a model (unique to the group animation pack) to the next available ‘slot’ in the space. Remote players in the scene can target the model and choose to join the animations in the free slot.

The behaviours can be the same for all players or they can be different for each player, controlled by the configuration file of the portable.

Information

A group animation takes place as follows:

1. The owner of the Group Animation Pack activates the portable from their inventory
2. Then the owner can access the list of available group animations by pressing the SQUARE button, when one is selected a space check is performed to make sure that there is a clear path between the owner and where the other slot positions will be, and the slots will be placed at the ground level for their respective positions.
3. When a group animation is selected, the owner is put into a waiting state, where they are doing a developer defined repertoire behaviour, and a model is deploy to the ground in the next available join slot that other players in the scene can see.
4. Other players can join in the group animation by targeting the model and selecting the option to join
5. The joining player is then translated to the free slot position and the group animation is started, with the guest and owner doing the behaviour designated against that slot in the group

6. Other players can join the same way, depending on the amount of guest slots that are specified in the configuration file
7. Guest players can leave the group by simply using the left analog stick to move their avatar, they will also leave if they perform any other emotes. When a player leaves they are translated back to the position they were in when they joined.
8. The owner can stop the group animation at any time by moving their avatar with the left stick, all players will then by unlocked and the model will be removed.

For a step-by-step guide on creating a new Group Animation Pack see Creating Group Animation Packs.

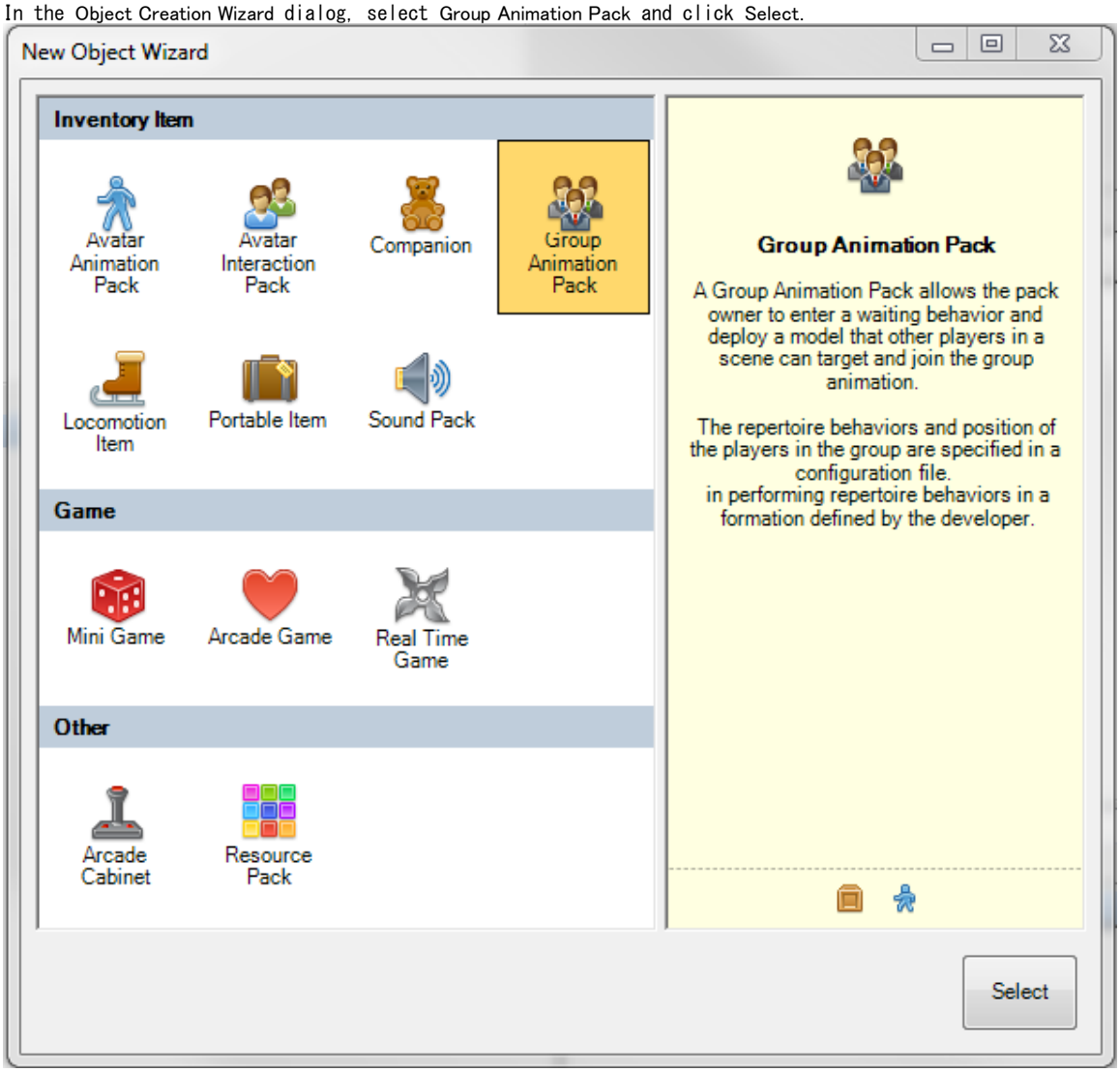
For details on how to configure and customize Group Animation Packs see Configuring Group Animation Packs.

Creating Group Animation Packs

Creating a Group Animation Pack

To create a new Group Animation Pack

1. In the Object Editor, select File > New Object Wizard.
2. In the Object Creation Wizard dialog, select Group Animation Pack and click Select.



3. Enter the name of the Group Animation Pack, and provide a description
4. Click Create to complete the wizard, a new Group Animation Pack is created at: <HDK_INSTALL_DIR>/build/GroupAnimPacks/<Resource Folder Name>/.

This folder initially contains all the default template assets that are required for the object to work. You must replace these files with your own. The folder also includes all the other assets required for the object to work in PS Home.

Preview the object in PS Home.

- To launch into the HomeDeveloper.self, see Launching the Home Developer SELF.
- To add items to your inventory in the HomeDeveloper.self, see The DEV DEBUG Menu.

Configuring Group Animation Packs

After creating a Group Animation Pack portable object there are 3 main steps to making your own pack. These are:

1. Create animations for the interaction in Maya HDK

Animations for the behaviours need to be created in Maya. There will need to be male and female animations created for each behaviour.

More information on creating animations in Maya HDK can be found in at Avatar Animation Tools

2. Set up the repertoires for the interaction in the Repertoire Editor

Once you have created the animations for the group animation pack they need to be added to the repertoires of the object. See the following documents on adding Repertoire Actions and Repertoire Behaviours to your object

For behaviours created in the Repertoire Editor, it is important that they are set up to allow the user to exit by moving the left analogue stick. There is an example of how to set this up in the 'Looping Behavior' section of the following article: Repertoires - Output Nodes Examples

If you would like to include an intro animation to play once before the looping behaviour then here is an example of how that can be achieved: Repertoire - Example Conditions

The default Avatar Animation Pack also uses this technique similarly to achieve a behaviour with an intro and then a looping animation.

3. Configure the Group Animation Pack in the configuration.xml resource file

You can configure an Avatar Interactions object through the configuration.xml file, which is referenced in the object's resources. At the base level of the xml file we'll have the following:

```
<xml>
 <animation_sets>
  <!-- 1 to 5 animation_set elements inside here -->
 </animation_sets>
</xml>
```

Then inside the 'animation_sets' element we have between 1 and 5 animation_set elements.

Example

Let's have a look at an example of how an animation set is defined:

```
<animation_set menu_name="kGroupDance1" waiting_behav="WaitingForDance1"
status_text="kGroupDance1Status" join_title="kJoinGroupDance1">
 <players min_players_to_start="2" min_slots="3" terminate_on_player_leave="false">
  <owner behav_name="Dance1A" />
  <guest ry="0" tx="1" tz="0" behav_name="Dance1B" />
  <guest ry="0" tx="2" tz="0" behav_name="Dance1A" />
  <guest ry="0" tx="3" tz="0" behav_name="Dance1B" />
 </players>
</animation_set>
```

Here we can see that:

- The localisation key for the text in the owner's group animations menu will be "kGroupDance1"
- The behaviour the owner will perform while waiting for remote players to join and start the group animation is "WaitingForDance1"
- The localisation key for the status text of the of the owner while these are free slots in the group animation is "kGroupDance1Status"
- The localisation key for the title text when a remote player targets a free slot is "kJoinGroupDance1"
- The group animation will start when there are 2 players present in total (1 guest)
- There must be at least enough space in the current part of the scene for the first 3 guest slots (the owner and 2 guests)
- The group animation will not terminate when any players leaves the group
- The owner and the player is slot 3 will do the behaviour "Dance1A"
- The guests in slots 2 and 4 will do the behaviour "Dance1B"
- The players will be in a straight line 1 metre apart from the owner's left hand side

Config specs

Here are the specifications of each of the different nodes in the group animation pack configuration.xml:

animation_sets

Acts as a container for all the configuration for the portable (local and remote models and each individual animation set). This has no attributes itself.

local_model

The model deployed to the next free slot in the group animation, viewed by the owner of the pack.

attrib	mandatory	default value	notes
name	Yes	none	The name of the object resource of the model
scale	No	1	The optional scale multiplier that can be applied to the model.

remote_model

The model deployed to the next free slot in the group animation, viewed by all remote players in the scene, the memory limits are lower for remote players so a less detailed model may need to be specified.

attrib	mandatory	default value	notes
name	Yes	none	The name of the object resource of the model
scale	No	1	The optional scale multiplier that can be applied to the model.

animation_set

An animation set defines 1 contains within it all the information about a single animation set (including the behaviour each player will do)

attrib	mandatory	default value	notes
menu_name	Yes	none	The localisation key for the name of the group animation in the owner's selection menu.
waiting_behav	Yes	none	The name of the repertoire behaviour that the owner of the pack should do while waiting for remote players to join.
status_text	Yes	none	The localisation key for the status text of the owner while there are free slots to join the group
join_title	Yes	none	The localisation key for the text that appears when remote players target the model to join the group (the name of the owner will be appended to the start of this)

players

The players element contains configuration relating to the player slots in the group

attrib	mandatory	default value	notes
min_players_to_start	No	2	The number of players required to start the group animation, if a number higher than 2 is used then all players will adopt the waiting behaviour until the minimum is reached.
terminate_on_player_leave	No	false	If the group animation session should be ended whenever any player leaves.
min_slots	No	2	The minimum number of slots there must be space for in order for the owner to enter into the waiting state. Otherwise they will receive a notification that there is not enough space.

owner

Specifies the behaviour that the owner performs when the group behaviour has started

attrib	mandatory	default value	notes
behav_name	Yes	none	The behaviour that the owner should do when the group animation is started.

guest

Specifies a guest slot in the group animation (they should be specified in the correct order, so the first guest element in the animation set will be the second slot of the group, and so on).

attrib	mandatory	default value	notes
behav_name	Yes	none	The behaviour that the owner should do when the group animation is started.
ry	No	0	The Y rotation offset of this guest in relation to the owner
tx	No	0	The X axis translation of this guest from the owner's position
tz	No	0	The Z axis translation of this guest from the owner's position
delay	No	0	Specifies a delay in starting of the behaviour after the owner. Could be used to create a Mexican wave style effect.

Limits and Validations

- guest: There is a maximum of 7 guest slots that can be specified in the configuration file
- min_players_to_start, min_slots: Must be at most the total number of slots
- scale: Must be between 0.1 and 5
- tx, tz: Must be between -5 and 5
- ry: Between -360 and 360
- delay: Must be between 0 and 3 seconds

Sound Packs

Overview

Sound Packs allow the portable owner to trigger sounds that other player near to them in a scene can hear.

The available sounds are specified by the portable developer.

Sound Packs have the following features:

- Allow the pack owner to play sounds that others close to the in a scene can hear.
- Each sound can be linked to a repertoire action or behaviour animation to accompany the sound.
- Sounds can be set to be able to overlap each other or interrupt an sound that is already playing.

For a step-by-step guide on creating a new Sound Pack see Creating Sound Packs.

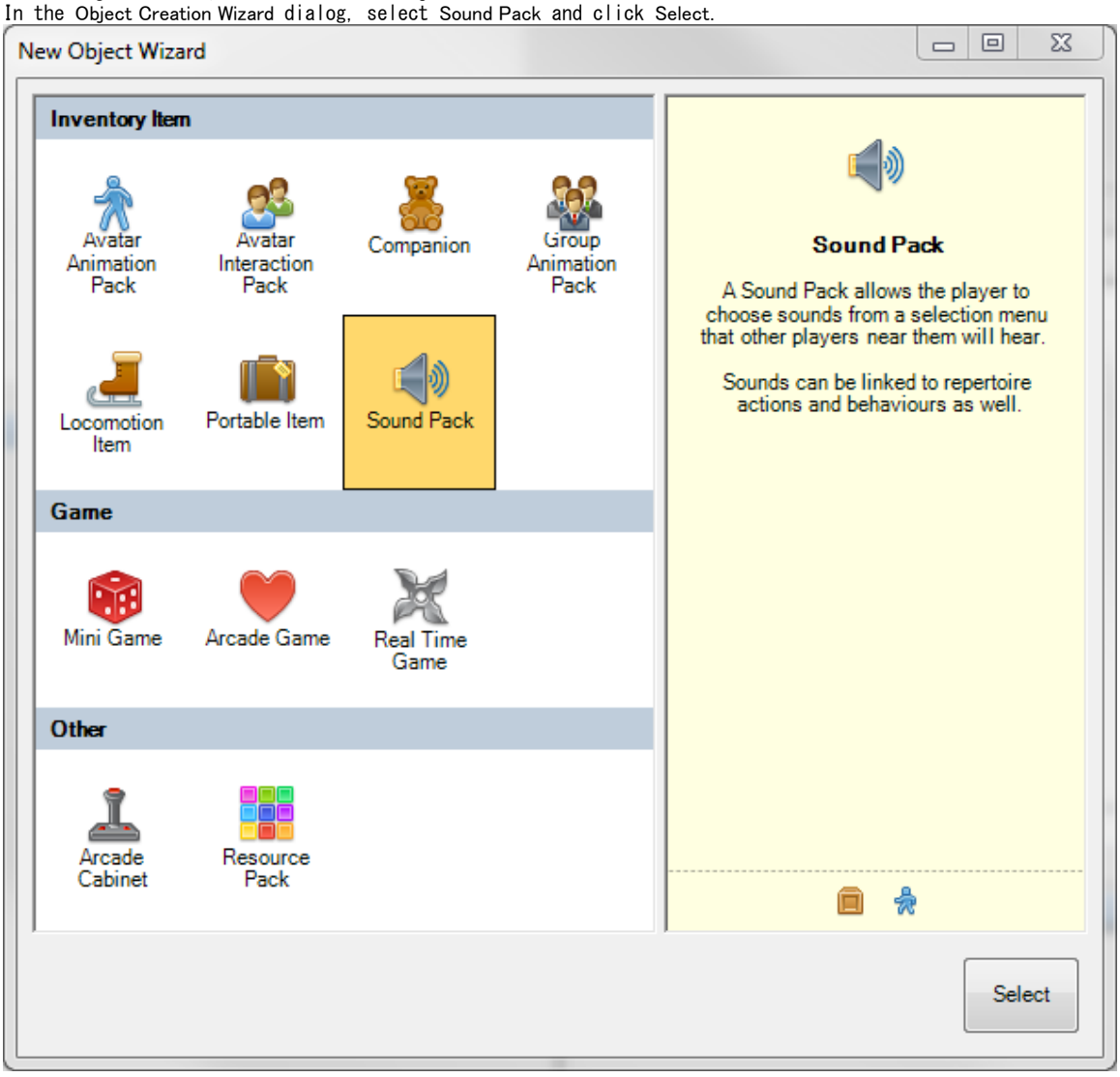
For details on how to configure and customize Sound Packs see Configuring Sound Packs.

Creating Sound Packs

Creating a Sound Pack

To create a new Sound Pack

1. In the Object Editor, select File > New Object Wizard.
2. In the Object Creation Wizard dialog, select Sound Pack and click Select.



3. Enter the name of the Sound Pack, and provide a description
4. Click Create to complete the wizard, a new Sound Pack is created at: <HDK_INSTALL_DIR>/build/SoundPacks/<Resource Folder Name>/.

This folder initially contains all the default template assets that are required for the object to work. You must replace these files with your own. The folder also includes all the other assets required for the object to work in PS Home.

Preview the object in PS Home.

- To launch into the HomeDeveloper.self, see Launching the Home Developer SELF.
- To add items to your inventory in the HomeDeveloper.self, see The DEV DEBUG Menu.

Configuring Sound Packs

You can configure a Sound Pack object through the configuration.xml file, which is referenced in the object's resources.

At the base level of the xml file can be the following:

```
<xml>
	<all>
		<!– configuration data for both genders -->
	</all>
</xml>
```

Or:

```
<xml>
	<male>
		<!–male configuration data -->
	</male>

	<female>
		<!–female configuration data -->
	</female>
</xml>
```

Depending whether you want to have the same sounds available for both genders, or have separate sets for both. Inside the all/male/female elements should be 'sound_resources' and 'sounds' tags.

Example

Here is a simple example for soundbank resources:

```
<sound_resources>
	<sound_resource name="bnk.male" />
	<sound_resource name="bnk.male2" />
</sound_resources>
```

This specifies that the current gender (or both) uses 2 different soundbank resources. These will be loaded in when the sound pack is activated, or the owner changes gender. These names must match the names of the resources in the Object Editor.

```
<sounds>
	<sound sound_name="s1" source="bnk.male" delay="1" anim_name="Act1" anim_type="action" />
	<sound sound_name="s2" source="bnk.male" menu_name="ks2" />
	<sound sound_name="s3" source="bnk.male" volume="0.5" />
	<sound sound_name="s4" source="bnk.male2" />
</sounds>
```

This specifies 4 sounds that can be played by owner in the current gender (or both). Here, the source for each sound must be specified because we are using more than 1 sound_resource. If there is only 1 sound_resource then this attribute can be omitted.

For the sound 's1', a delay has been specified, this will mean that the sound is played 1 second after the owner selects it from the menu. This is to allow the sound to be synchronised with the animation (in this case, the sound matched up with 1 second into the repertoire action animation).

For sound 's2' the menu_name attribute has been specified. This is to specify that the text for the sound in the owner's menu should use the localisation key "ks2". If this attribute is omitted then the sound_name attribute will be used as the localisation key.

Sound 's3' will only be played at half the volume of the source file (this might be necessary to balance the volumes of your sounds equally).

The resources specified in source_resources should be set to defer load and they will be loaded in when the portable is activated or when the player changes gender.

Config spec

Inside the root xml tags, either an 'all' element can be included, or 'male' and 'female'. If 'all' is included then the sounds specified will be available when the owner of the Sound Pack is both male and female. Otherwise, if separate 'male' and 'female' tags are included, then different sounds for each gender can be defined.

Inside male/female/all can be the following elements:

sound_resources

The sound_resources tags have no attributes but serve as a container for the individual sound_resources.

sounds

The sounds tags have no attributes but serve as a container for the individual sounds.

sound_resource

attrib	mandatory	default value	notes
name	Yes	none	The name of the soundbank resource, as it appears in the Object Editor

sound

This specifies an individual sound to be included in the sound pack.

attrib	mandatory	default value	notes

sound_name	Yes	none	The name of the sound in the soundbank to play
source	Yes	none	The name of the soundbank to play the sound from (can be omitted if only 1 soundbank is included in sound_resources)
menu_name	No	The value of sound_name	The localisation key for the name of the sound in the owner's selection menu.
volume	No	1.0	The volume value for the sound between 0 and 1
pitch	No	0	The pitch adjustment of the sound between -10 and 10.
register0	No	0	The value for register 0 for the sound.
register1	No	0	The value for register 1 for the sound.
register2	No	0	The value for register 2 for the sound.
register3	No	0	The value for register 3 for the sound.
delay	No	0	The delay in seconds before playing the sound after it is selected (for synchronising with animations)
allow_overlap	No	false	Whether the sounds are allowed to overlap each other
anim_name	No	none	The name of the action or behaviour to play, as it appears in the Repertoire Editor
anim_type	No	none	Either "action" or "behaviour", whether the value supplied for anim_name is a repertoire action or a behaviour.
stop_anim_at_sound_end	No	false	Whether the action/behaviour should be stopped when the sound finished, generally not required

Animation info:

- If an animation is specified for a sound then it cannot run alongside other sounds (regardless of what the allow_overlap attribute is set to). It will interrupt another playing sound if selected, and it will be interrupted if another sound is chosen.
- For sounds with repertoire behaviours specified, for sounds to be continually looping they must be configured to be self-looping in your audio editor.
- If a player stops the action or behaviour from occurring by moving or choosing another emote then the sound will also stop playing.
- If a sound finishes playing then the action or behaviour will continue, unless the stop_anim_at_sound_end option is set to true.

Scenes

Find out about creating the environments for a scene in one of the art tools, then head over to the Scene Editor where you combine all your assets (as well as embed some scene objects and strategically place screens), and test it out either online or offline before packaging to send it off to the Content Delivery System (CDS) to be published in the live client.

PS Home Spaces tells you what kind of scenes you can create.



See also Scene Information for details on scripting scenes.



PS Home Spaces

For Japanese Live Page
最新の日本語版ページはこちら 「PS Homeのシーン」

For a feature summary for each type of Home space, see Memory Limits for Spaces

Spaces in PS Home can currently be categorized into 4 different scene types (each with different behavior):

- Public spaces
- Personal / Apartment spaces
- Clubhouse spaces
- Game & Video spaces

Each scene type has a different use and different methods for how the user arrives at them.

- Public spaces are much as the name suggests - they are spaces that any player can get to, usually by selecting the space from the navigator or by walking from a connection point in another scene.
- Personal / Apartment spaces are those that are 'owned' by the player. When the player goes to an apartment space, they're going to an instance of the space specific to them. Other users must be invited by its owner to gain entry, and the owner must be present in the space for others to be there. If the space owner leaves the space while invited guests are present, the other users are ejected from the space.
- Clubhouse space instances are connected to a particular club, and only club members can go to the space.

Game Spaces and Video Spaces (16 players) cannot be directly accessed from menus (the XMB™, Navigator or Menu Screen) but must be entered via a Public Space or Clubhouse, using any of the Relocate functions. See Relocating to Instances.
Game Spaces (32 players) can be directly accessed from menus (the XMB™, Navigator or Menu Screen).

See also Memory Limits for Spaces and Launch into PS Home from XMB™ or PS3 Title.

Each PS Home space is one of a number of different scene types, that cater for various use cases and development needs:

Public space scene types

- (Standard) Public Space : A Public Space can be accessed by any user (subject to age restrictions) and can accommodate up to 60 users. Public Spaces are very flexible, supporting most PS Home client features and are well suited to providing a mixture of content. A typical Public Space contains a combination of mini-games, realtime games, arcade games, and screens. So it's a good choice when creating a space for the PS Home community to meet and interact in.
- Public Space (No Video): Like the standard public space, Public Space (No Video) can be accessed by any user (subject to age restrictions) and can accommodate up to 60 users. Unlike the standard public space scene type, it provides as much memory as possible to game content by disabling many non-game features (such as video playback and web browsing).

Personal / Apartment space scene types

- (Standard) Personal Space: A Personal Space belongs to a specific user. Personal Spaces support up to 12 users, including the owner. A key feature of personal spaces is customization; the space owner can place furniture, hang pictures, and use Active Items.
- Personal Video Space: This scene type belongs to a specific user like a Personal Space, but includes support for 720p HD video instead of support for furniture placement and the web browser.
- Personal Space (No Video): This scene type belongs to a specific user like a Personal Space, but disables support for video and provides more memory to the main scene budget.

Clubhouse space scene types

- (Standard) Clubhouse: The Clubhouse is a space open only to members of a club. Each member is able to use Active Items and embedded objects, read the Club Bulletin Board, and access the space even when the space's owner is offline. Only the owner can of the Clubhouse may place furniture.

Game / Video spaces

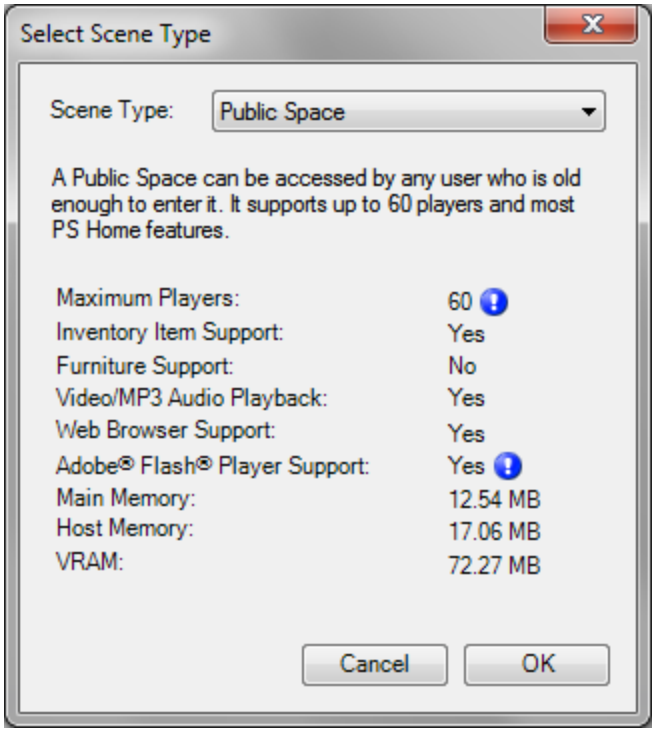
- Game Space (16 Players): This scene type is optimized for games, providing as much memory as possible to game content and disabling many non-game features (such as video playback and web browsing). The maximum capacity of this scene type is 16 players, making it perfectly suited to games that use the RtGame library. This scene type has limitations on how it can be launched.
- Game Space (32 Players): Like Game Space (16 Players), this scene type is optimized for games, but supports up to 32 players. The higher player count means that there is slightly less memory available than in the 16 player space. It is better suited to games using the MiniGame library. This scene type has limitations on how it can be launched.
- Game Space (60 Players): Like Game Space (16 Players) and Game Space (32 Players), this scene type is optimized for games, but supports up to 60 players. The higher player count means that there is slightly less memory available than in the 16 player or 32 player spaces. It is better suited to games using the MiniGame library.
- Video Space: This scene type has additional memory allocated for video playback. This allows AVC/H.264 video playback up to Level 3.1 (HD 720p at 30 frames per second). This scene type supports up to 16 players. When designing Personal Spaces and Clubhouses that use video, be aware that users might want to place Active Items that play video. Create video screens that only play when users elect to watch them. If a Personal Space or Clubhouse is always using a video screen, users will never be able to play videos in active items
- Arcade Space: This scene type is optimized for arcade games, like the Game Spaces it disables many non-game features (such as video playback and web browsing). However, unlike the Game Spaces it still supports inventory items. This scene type supports up to 60 players.

The scene types differ in the following ways:

Feature	Description	Related Documentation
Player capacity	Player capacity ranges from 12 to 60 users.	Memory Limits for Spaces
Access restrictions	Most scene types allow anyone to join at any time unless they are prevented from doing so by age restrictions. Some scene types are invitation-only.	Age Restrictions
Available client features	In some scene types, certain client features (such as video playback) are disabled in order to provide more memory for other types of content.	Media Memory Usage, Video Recorder Guidelines and Tips
Memory limits	Memory limits form part of the technical requirements for publishing content in PS Home.	Space Validations Memory Limits

Changing the Scene Type

In the Scene Editor, chose Edit Scene Type from the Edit menu. Select the desired scene type from the drop-down menu and press OK.



Scene Type Features

The following features are available to (and in some cases required for) certain types of scenes.

Reduced Player Count

The Game Space (16 Players) and Game Space (32 Players) scene types feature a lower player count than the standard 60 players.

Game Spaces and Video Spaces (16 players) cannot be directly accessed from menus (the XMB™, Navigator or Menu Screen) but must be entered via a Public Space or Clubhouse, using any of the Relocate functions. See Relocating to Instances.
Game Spaces (32 players) can be directly accessed from menus (the XMB™, Navigator or Menu Screen).

> LocalPlayer.RelocateToUniqueInstance may require the use of a server-based application to track instance IDs.

Scene Entitlement Object

You use a scene entitlement object to grant a user ownership of the space. The scene entitlement object also includes commerce data and descriptions for your scene. You must create a scene entitlement object for Personal Spaces and Clubhouses. See Designing a Home Space and Clubhouses.

See Scene Entitlement Objects for information on how to create a scene entitlement object.

Scene Flags

Scene flags can turn off settings allowing Favorite, Invite, Go To, Global access and Portables.

You must contact your Regional SCE to use this feature.

See Scene Flags and Settings for Spaces for more details.

Scene Save Data

Save Data enables you to save and load data stored online using the Save Data Service. Save data can also be scene save data for the scene owner, but this method is discouraged.

Scene Save Data is available to scene scripts or embedded objects for Personal Spaces and Clubhouses. Scene Save Data can be used only by the owner of the Personal Space or Clubhouse, and while the owner is present in the space.

For more information, see Save Data and Save Data Service.
For guidelines on using Scene Save Data, see Save Data Guidelines.

Picture Hooks

You use picture hooks to place Picture Frames (the furniture placed on walls) in a scene. You can place picture hooks only in Personal Spaces. See Picture Hooks.

Club Bulletin Board

A Club Bulletin Board provides club members with information. You can place bulletin boards only in Clubhouses. See Club Bulletin Board

Designing a Home Space

When designing a personal space or clubhouse, the memory limits and requirements are the same for both types of space. Public spaces have different memory requirements. For a full list of limits, see Content Requirements.

You create clubhouses using the same tools and workflow as for normal scenes, but you also follow some additional tasks. For more information, see Clubhouses.

For comprehensive documentation on the different elements of scene creation, see Environments.

Using Maya to Create Scenes

Use Maya for the following tasks:

- Creating the scene artwork (geometry), add an ATG material, and apply UV sets
- Adding collision
- Adding light sources

> The default setting for alpha sorting is `true`. You can change this behavior using Lua, by calling the function `Scene.SetGraphicsEngineParam(AlphaSortEnable, false)`.

The example in this section creates a basic personal space (a square room) to demonstrate scene creation and export.

To add sounds, objects, mini-games and other resources, you use the Scene Editor. See:

- Scene Editor
- Furniture and Decorations
- Game Components
- Picture Frames and Wall Decorations

When you have created your scene you can export it using the Export Wizard. See Validating and Exporting Environments.

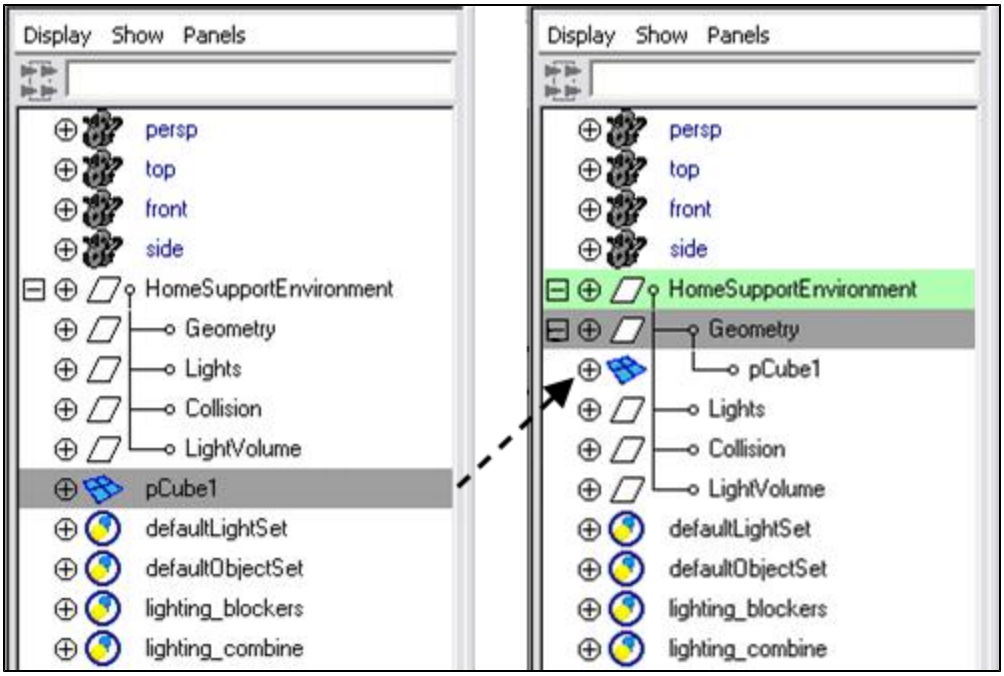
Creating the Scene Geometry

1. In Maya , select File > New Home Environment.
2. Enter a name for the environment and click OK:



   When you create a new home environment, the new project opens with the properties you need to export the space successfully (Geometry, Lights, Collision and LightVolume nodes).

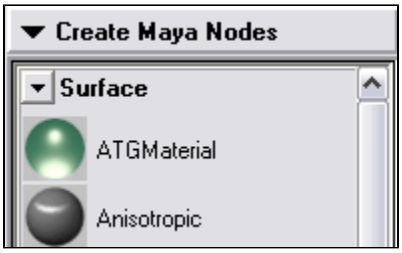
3. Add the geometry. Each geometry shape in your environment must:
   - Be assigned to an ATG Material.
   - Have two UVs on it called 'map1' and 'map2'.
4. As you design your space, move all of your shapes (primitives and AEC objects) into the geometry node:



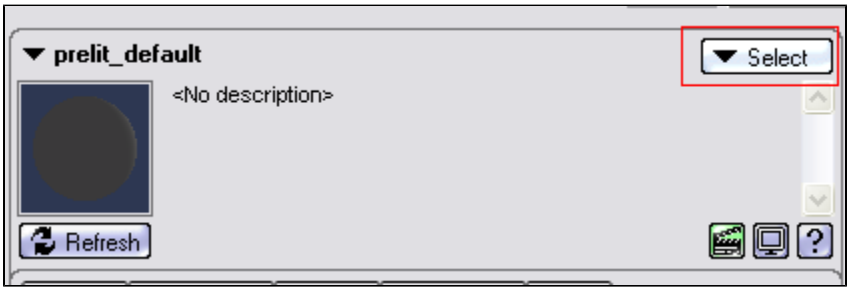
Applying an ATG Material

Applying an ATG Material is no different to applying any standard Maya material:

1. Create an ATGMaterial node (in this view, click on ATGMaterial):



2. With the ATGMaterial selected, change the texture to the one that you want to use by clicking Select:



3. Select the shape or shapes to which you want to assign the material, then right-click on ATGMaterial and select Assign Material To Selection.

Applying UV sets

To apply UV sets:

1. Select Window > UV Texture Editor to open the UV Texture Editor.
2. Select Polygons > Create Empty UV Set.
3. Select Polygons > Rename Current UV Set... to display the Rename Current UV Set Options dialog.
4. Enter map2 in the New UV Set name field and click Rename Current.
5. Select UV Sets > map2 in the UV Texture Editor.
6. Close the UV Texture Editor and select Create UVs > Automatic Mapping.

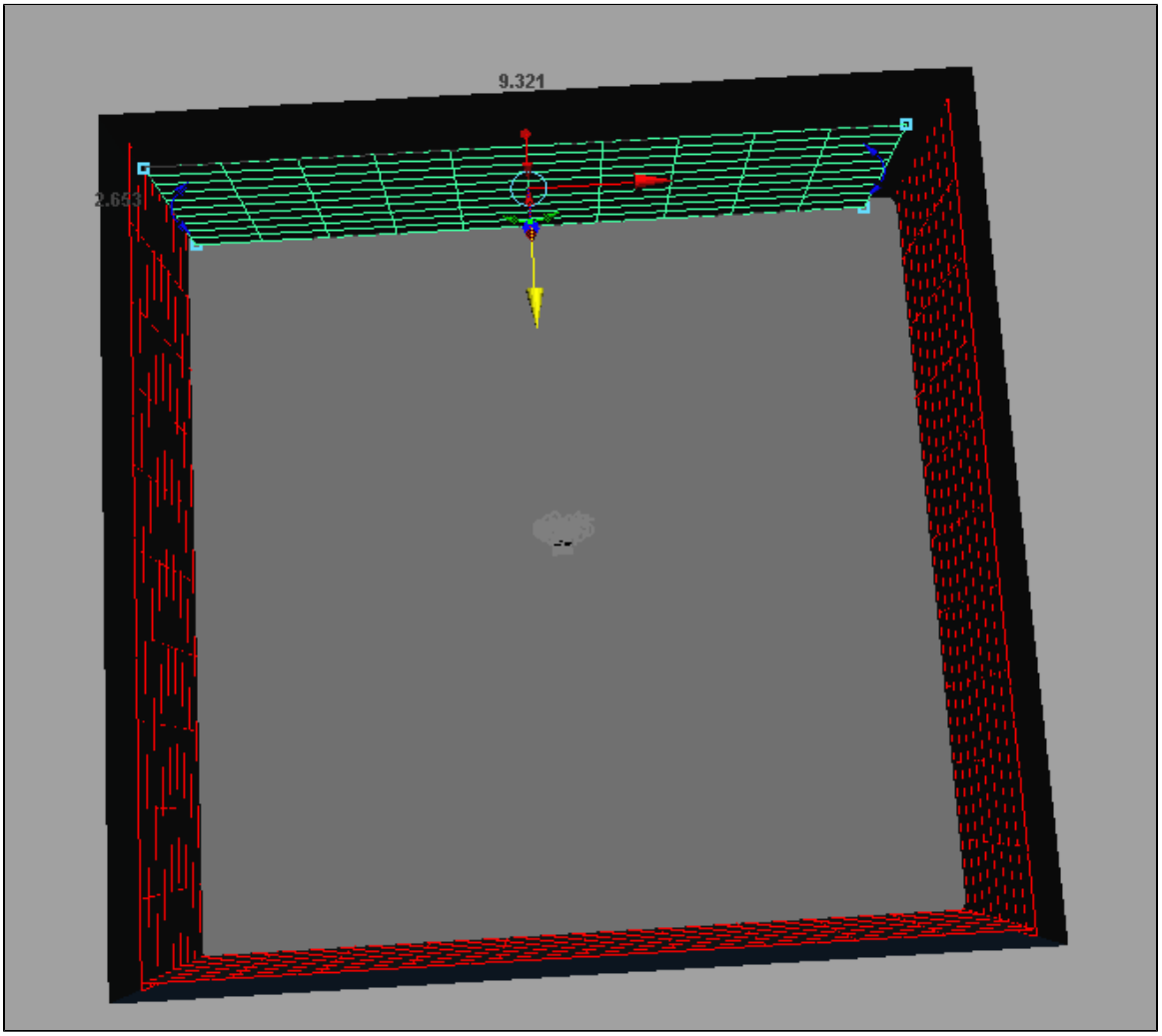
This is a very simple use of UVs, and probably not ideal for your scene. For more information on UVs, see Environment Creation.

Adding Collision

See also Collision in PS Home.

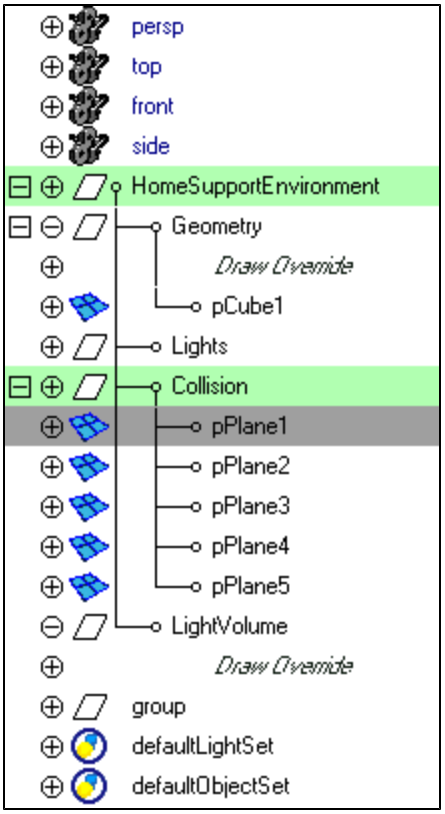
To add collision to your scene:

1. Select Create > Polygon Primitives > Plane.
2. Position the plane shape along the surface against which you want to create collision:



In the image above, the red areas show where plane shapes have already been applied. The green area shows a plane shape being positioned along the fourth wall.

3. Move the collision into the Collision node.



Gaps in Floor Collision

A remote avatar uses a thin ray-cast instead of the collision volume that a local avatar uses. If there are any gaps in the floor of a scene, remote avatars can fall through. The remote avatar eventually returns to the floor in the local avatar's perspective. To prevent this problem, make sure that there are no gaps in the floor collision.

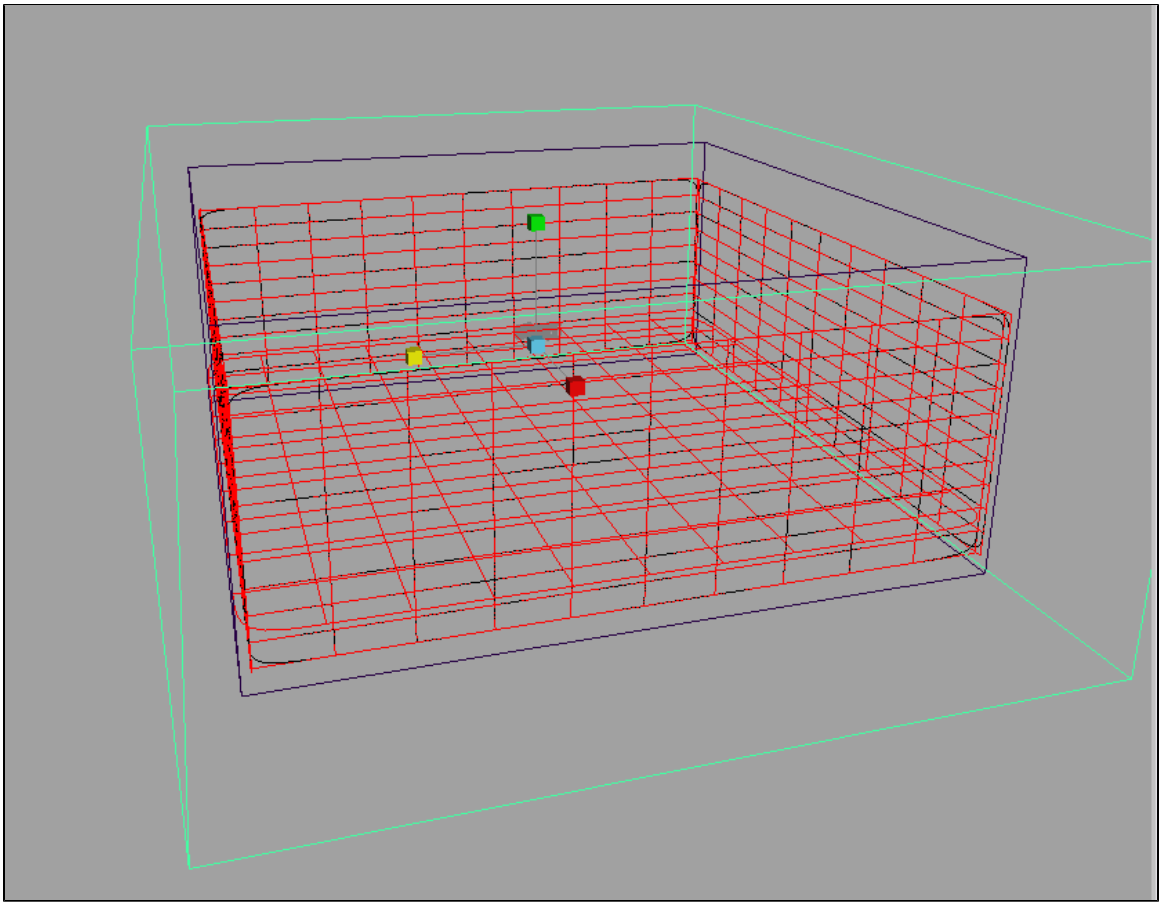
> From the remote user's local perspective, they never fall through the floor, because on their local machine they have a larger collision volume.

Adding Light Sources

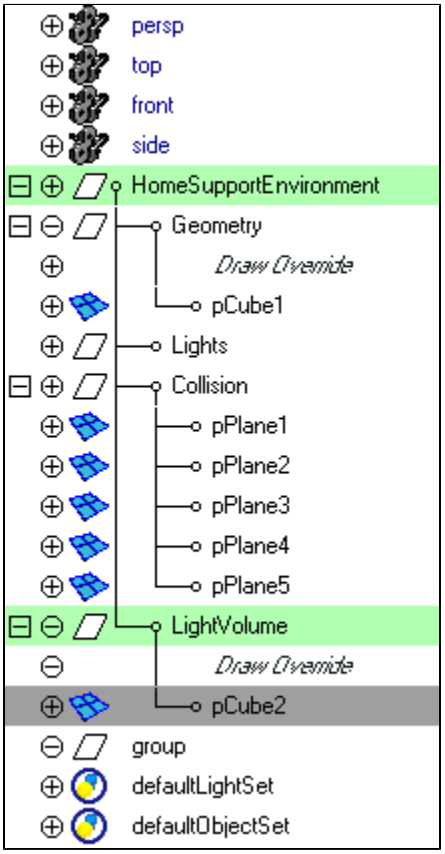
Creating a LightVolume

To create a LightVolume for your scene:

1. Select Create > Polygon Primitives > Cube.
2. Size and position the cube so that it covers every area of space that a user can walk around and on. In the following example, the green cube is the LightVolume cube, surrounding the geometry:



3. Move the cube shape that you are using as LightVolume into the LightVolume node:

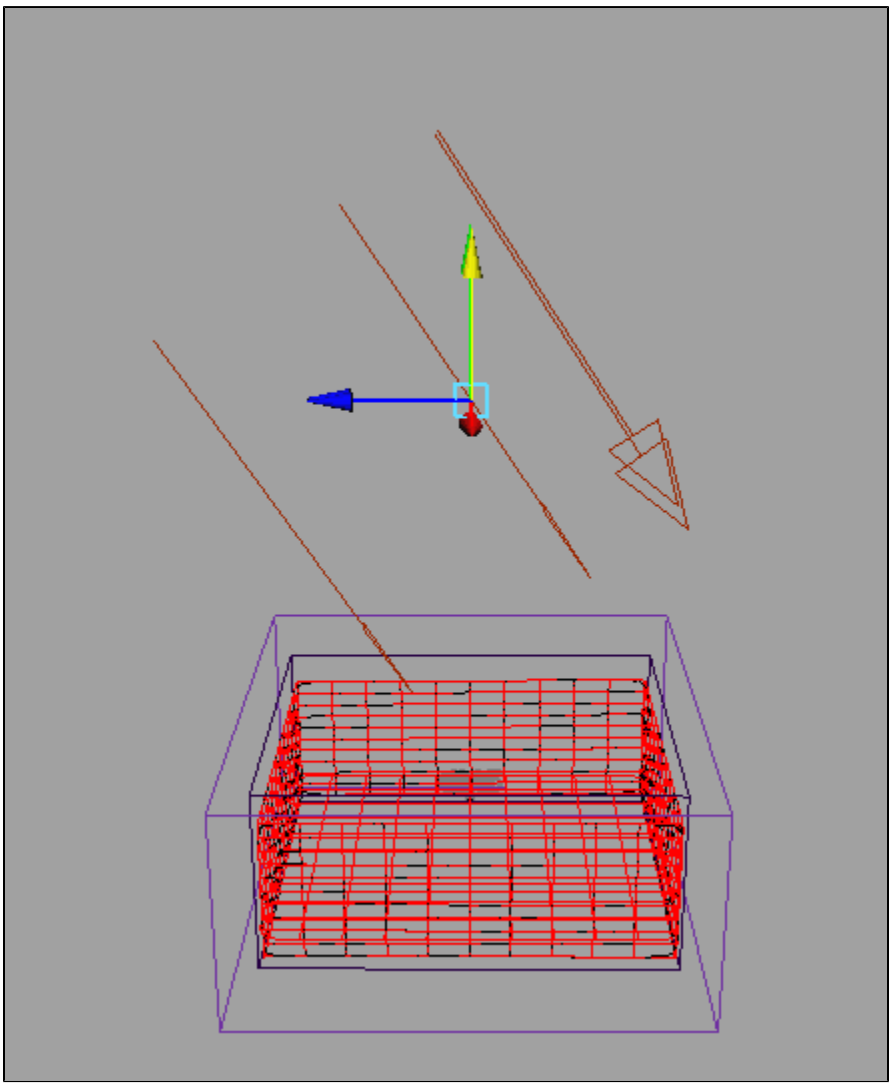


Adding a Sun

All environments must have a sun, whether or not they are completely enclosed. For example, a movie theater scene still has a sun, positioned on the movie screen. The sun defines the direction in which shadows are cast.

There are several ways to add a light source (sun). Here, you use the Home_Env toolbar.

1. Click the Create Sun button  . This creates a sun source sphere that you can position.
2. Position the sun and direction of the sun in the scene.



The image shows the sun and the direction of the light. The sun is the small red sphere above the box. It is intersected by a ray of light, and the blue and yellow arrows.

Dynamic Shadows

You can specify which items cast dynamic shadows by setting the geometry on the items to override default behavior. You can also change the quality of the shadows cast by dynamic lights or the scene's sun.

For more information on dynamic shadows, see Dynamic Shadows in the Environment Lighting.

Clubhouses

A clubhouse is a space associated with a PS Home Club that only clubhouse members can access. A clubhouse is similar to a personal space, with customization options available to the owner, such as placing furniture. However, unlike personal spaces, the space persists after the owner has left and club members can access the space whenever they are online.

The following table lists the key features for a clubhouse:

Feature	Description
Persistence	Clubhouses persist even when the owner is offline. This means that all clubhouse members can enter the scene at any time – the clubhouse is always available to them. It also means that all furniture placement remains permanently in the position that the owner placed it.
Club Bulletin Screen	The clubhouse bulletin screen allows clubhouse users to post messages. Four messages are displayed on the club bulletin screen. You can see any other messages by paging through the old messages. The bulletin screen is available only for scenes that are packaged as clubhouses. See Adding a Club Bulletin Screen.
Customization	The owner can customize the space, such as adding furniture or picture frame. The customizations are permanent. The owner can log out and log in without needing to redecorate the space for each session.
Avatar Limitation	Unlike personal spaces, which can contain a maximum of 12 avatars, up to 32 avatars can enter a clubhouse.

When designing personal spaces and clubhouses that use video, be aware that users might want to place active items. Create video screens that only play when users elect to watch them. If a personal space or clubhouse is always using a video screen, users will never be able to play videos in active items.

See also:

- Creating Clubhouses
- Adding a Club Bulletin Screen
- Testing Clubhouses
- Launching a Clubhouse
- Media

Creating Clubhouses

When creating a club you must first purchase a club slot. If you already have a club and want to set up a new one, you must first disband the existing club you own. You can be the owner of only one club at a time.

You create clubhouses using the same tools and workflow as other types of scene, but in addition, you need to carry out some clubhouse-specific steps:

1. When you have created your environment in Maya, export it as a Clubhouse. See Validating and Exporting Environments.
2. Open the scene using the Scene Editor and add a club bulletin screen. See Adding a Club Bulletin Screen.
3. Set the launch options in the Scene Editor to Clubhouse. See Launching a Scene in PS Home.
4. Discuss with your regional SCE how to make your clubhouse available for commerce, including the pricing or whether it is available for free. See Commerce in PS Home and Types of Commerce.
5. Both personal spaces and clubhouses require a scene entitlement object, which grants a user ownership of the space and includes commerce data and descriptions for your scene. See Scene Entitlement Objects for information on how to create a scene entitlement object. See also Rewards and Commercial Items.
6. Create a Home Club in your test environment so that all the clubhouse features can be used. See Testing Clubhouses.

   Existing PSN Clans or Home Clubs are not available in the test environment.

7. When you package the scene in Scene Editor, select Clubhouse. See Preparing Scenes for Packaging and Packaging Scenes.
8. When you submit the scene to the CDS, select Clubhouse. See Submission Testing and the CDS.

Associating Home Clubs with Clubhouses

When the clubhouse is live in PS Home, any owner of a Home Club can associate your clubhouse with their club. If your clubhouse is a commerce item, they must purchase it first. To associate your clubhouse with their club, the owner must do the following:

1. Select Clubs > My Clubs and select their club.
2. Select Admin Controls > Clubhouse > Change Clubhouse and choose your clubhouse from the list.

Moving Club Information

You cannot move club information from one server to another, or access it from different accounts, without causing errors. If you created a Home Club using a previous version of the HDK (for example, to test clubhouses online), you must disband it and create a new club for a new installation of the HDK. See Disbanding a Home Club.

For information on how to install a new version of the HDK, see Installing the HDK.

See also:

- Designing a Home Space

- Launching a Clubhouse

Adding a Club Bulletin Screen

The Club bulletin screen displays messages posted by clubhouse users. It shows the following information:

- From: The name of the user who posted the message.
- Posted: The date of the message.
- Subject: The subject line heading entered by the user (max. 18 characters).
- Message: The message text entered by the user (max. 64 characters).

When a user in the clubhouse targets the club bulletin screen, the Bulletin Board interaction is available. This interaction allows the user to enter message details and post them to the board. When they post a message, it is added to the first message on the board. A maximum of four messages are displayed at a time. As more messages are added, the first message moves along the queue until it drops off the board. Users can view older messages by paging through them.

The interaction and general functionality is controlled by the Home Client and is the same for all club bulletin screens. You can customize some of the display properties in the Scene Editor.

> You cannot change the font and layout of the message itself. The layout of the board is defined in the Scene Editor.

Creating the Club Bulletin Screen

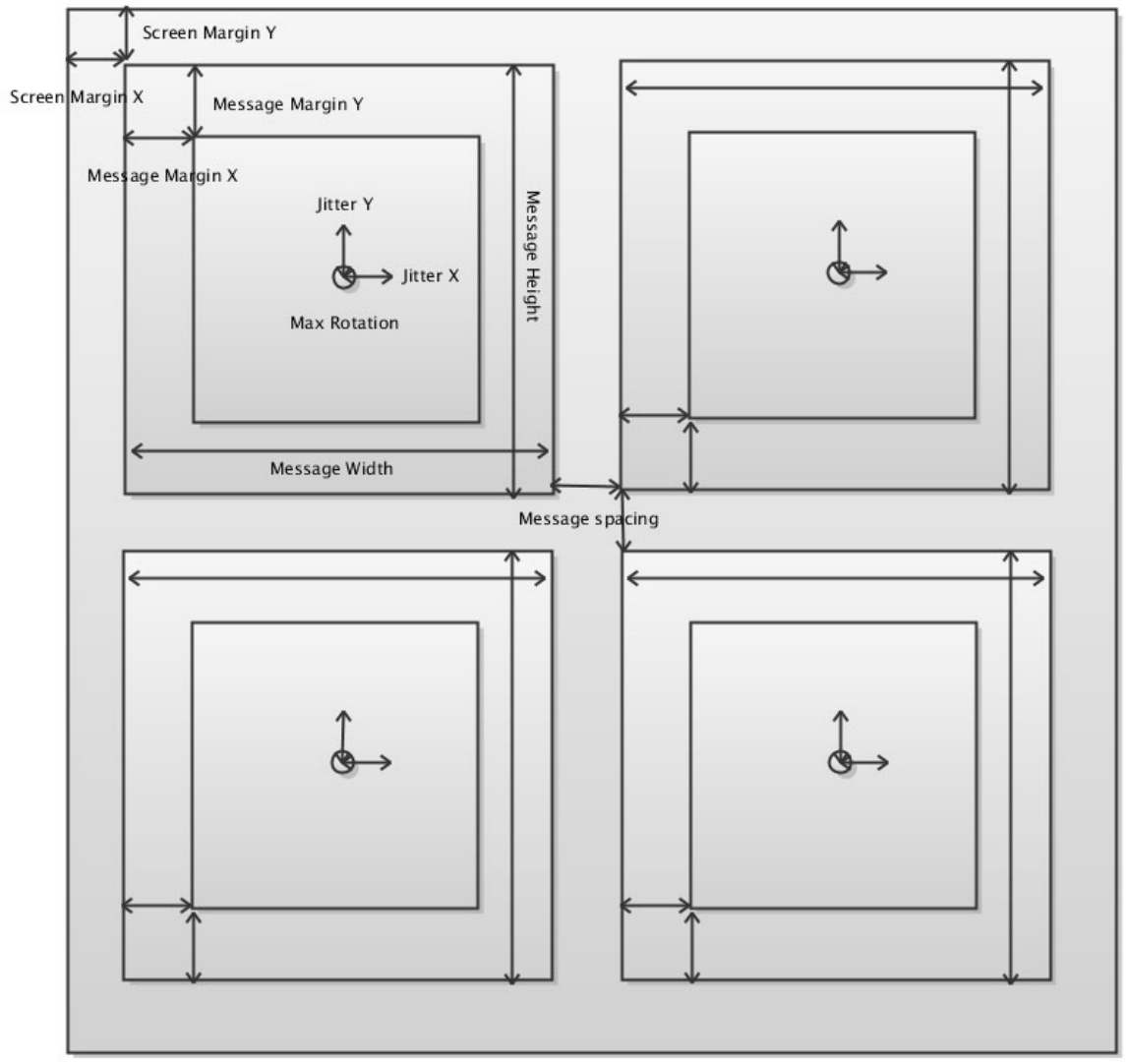
> You can have a maximum of 5 bulletin screens (bulletin boards) in a clubhouse.

To set up the Club Bulletin Screen in your scene:

1. Open the clubhouse scene in the Scene Editor.
2. Drag the Club Bulletin Screen object from the Palette panel to the Game Objects folder in the Project panel, or directly onto the Design View:



3. Edit the properties of the Club Bulletin Screen object or manipulate it using the various tools to place the screen and define its appearance:



For more information on how to edit object properties in the Scene Editor, see Working With Objects in a Scene. For more information on the properties themselves, see the Club Bulletin Properties in Scene, Object and Asset Properties.

Previewing a Club Bulletin Screen

Before previewing your Club Bulletin Screen, ensure that you are a member of a club. If you are not a member of a club, create one from the Social section of the menu. If you have previously created a club using a previous version of the HDK, you will need to disband this club. To disband a club:

1. Use the console command Clans.GetUserClanList to retreive a list of clubs that you belong to.
2. Use the console command Clans.DisbandClan <ID> using the ID displayed from the previous command.

Once you are a member of a club, launch your scene from the Scene Editor, ensuring that the Scene Type is set to clubhouse and that you are launching online with a scene key.

See also:

- Creating Clubhouses
- Testing Clubhouses
- Launching a Clubhouse

Testing Clubhouses

When creating a club you must first purchase a club slot. If you already have a club and want to set up a new one, you must first disband the existing club you own. You can be the owner of only one club at a time.

To make the clubhouse features available when testing your scene using a Test Kit or Development Kit, you must do one of the following:

- Launch the scene online using the Target Manager, then set the clan ID to an existing Home Club.
- Be an owner or member of a Home Club in the test environment. As an owner or member, you can launch the scene online using the Scene Editor. Set the type to Clubhouse.

Setting up a Home Club

To become the owner of a Home Club in your test environment:

1. Launch the Home Client.
2. Select any available scene, and on the Menu Screen, click the Social icon  to display the Social Menu.
3. On the Social Menu, select Clubs > Create Club.
4. Follow the instructions to purchase a Club Slot and then create your club.
5. Enter a Name for the club (maximum 21 characters).
6. Specify a Tag for the test club. This is the abbreviated tag by which the club will be represented (maximum of 8 characters).
7. Click Create and then press [CROSS] to create the club.

> ⚠ All PS Home developers can see test clubs. Make sure that your test club name does not reveal any sensitive information.

The new Home Club is valid only in the environment in which you create it (such as CDEV, CQA, PROD). The club information, including membership and messages posted, is available only in that environment.

When you submit your clubhouse to QA using the Content Delivery System (CDS), all the club information that was used for testing on DEV is no longer available in the new environment. Only members of the club on DEV can access your clubhouse and see the messages.

Joining a Home Club

You can belong to up to five clubs at a time.

To become a member of an existing Home Club in your test environment:

1. Launch the Home Client.
2. Select any available scene.
3. On the Menu Screen, click the Social icon.
4. On the Social Menu, select Clubs > Search Clubs.
5. Enter the name of the club you want to join and then select Start Search.

   You can enter the name or partial name of the club. The search is case-sensitive.

6. Select a club from the list displayed, complete a Subject and Body for the application message, then click Apply to Join.

Your application is sent to the club owner. The club owner must accept your invitation for you to become a member.

Disbanding a Home Club

Only the owner of a club can disband the clubhouse. The owner must be within the clubhouse itself to disband it.

To disband a Home Club in your test environment:

1. Launch the Home Client.
2. Go to the clubhouse for the club you want to disband.
3. On the Menu Screen, click the Social icon.
4. On the Social Menu, select Clubs > My Clubs.
5. Select the club to be disbanded and then click Disband.

Launching a Clubhouse

To launch a clubhouse you must have an existing Home Club. You launch a scene online as a clubhouse using Target Manager.

To launch a clubhouse:

1. Launch the Home Client.
2. Select any available scene.
3. Open the debug console and enter the following command:

```
Clans.GetUserClanList
```

4. The console displays a list of your clubs, together with the Clan ID for each one, for example, `ID=30`.

   The ID is the integer used in the command line parameter in the next step.

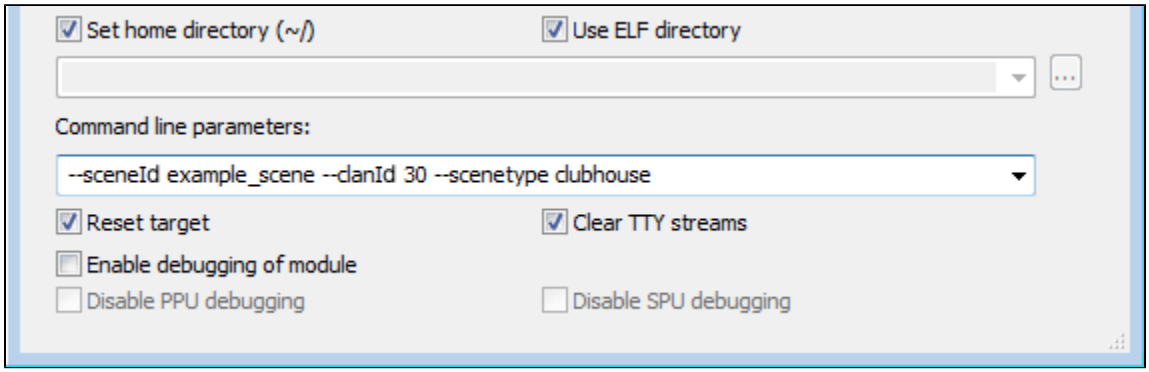
5. Launch the Home Client using the Target Manager and the following command line parameters:

```
--sceneId <scene name>

--clanId <clan ID integer>

--scenetype clubhouse
```

   For more information on tags, see Launching the Home Developer SELF (for example, `<scene name>`).



6. The scene launches with all clubhouse features enabled for the club you specified with the `--clanId` parameter.

Regional and Global Instances

> See Lobby Instances for information on how instances work and Relocating to Instances for methods for relocating to instances.

PS Home operates on a regional model where users are placed in space instances with other users in the same SCE region (based on their PlayStation®Network account). However, users in any region can be invited to enter personal spaces and clubhouses without regional restrictions.

A global space does not group users by SCE region. Users in all supported countries/regions can meet and interact in a public space, game space or video space.

Regional Model

The following table shows the countries/regions in which PS Home is available, grouped by SCE region.

SCE Region	Country
Sony Computer Entertainment Japan (SCEJ)	Japan
Sony Computer Entertainment Asia (SCE Asia)	Hong Kong Indonesia Korea Malaysia Singapore Taiwan Thailand

Sony Computer Entertainment Europe (SCEE)	Australia Austria Belgium Czech Republic Denmark Finland France Germany Greece Ireland Italy Luxembourg Netherlands New Zealand Norway Poland Portugal Russia Saudi Arabia South Africa Spain Sweden Switzerland United Arab Emirates United Kingdom
Sony Computer Entertainment America (SCEA)	Canada United States of America

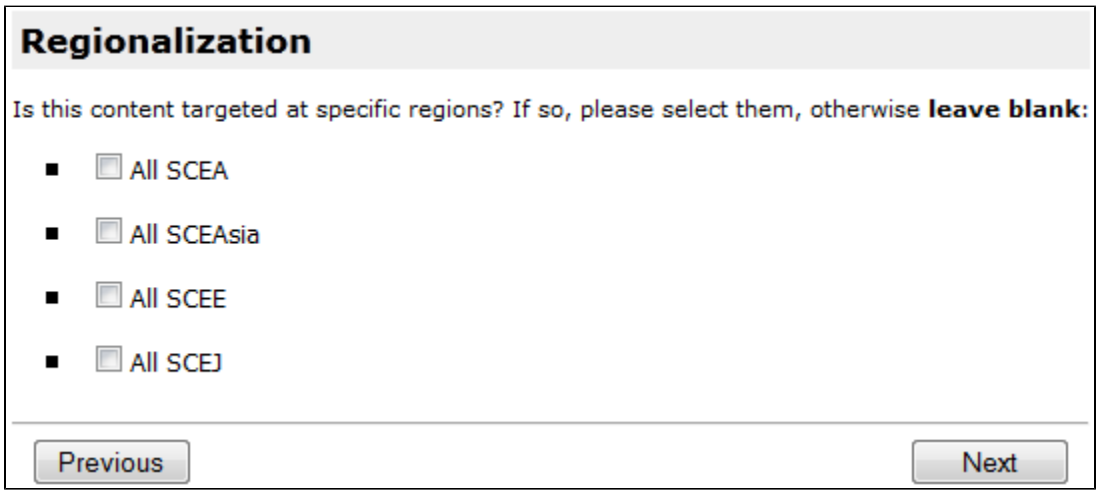
Global Spaces

A Global Space is a space that does not group users by SCE region. This means that all users in all supported countries or regions can meet and interact in a Public Space, Game Space or Video Space. Global spaces must be approved by all four SCE regions before they are released. This is because all the regions will support the space through customer relations, the in-game community, or by physically hosting the space.

If you want to release a space as a Global Space, contact your regional SCE before starting development. They will submit the proposal to the other regions for discussion and will inform you of the result. After you receive approval for the Global Space, your regional SCE will advise you on what you need to do.

Your tasks will include:

- Packaging the space in the Scene Editor as a public space. See Preparing Scenes for Packaging and Packaging Scenes.
- Submitting your space using the Content Delivery System (CDS) as a Public Space (see Submission Testing and the CDS). At the Regionalization step, leave all region boxes blank, as shown below:



- Testing and QA in each region.

All requirements should be the same as public spaces. Your regional SCE will tell you If they are different.

Scene Flags and Settings for Spaces

For Japanese Live Page
最新の日本語版ページはこちら 「シーンでのシーンフラグと設定」

It is possible to request certain settings for spaces - called flags. These can restrict what the spaces allow.

E.g. the following flags can be set to ON/OFF:

- Add To Favorites
- Go To
- Invites
- Global scenes
- Portables

For example

The "Add To Favorites" flag can be set so anyone who is able to get into the space will nnt be able to add it to their favorites.

The "Invites" flag can be set so that users from a different region will not be able to join a friend there and a user already in the scene will not be able to invite anyone, even from their own region.

You can also set a cap on the number of players allowed to enter a space.

⚠ Adjusting the cap on the number of players is not advised. Where possible, it is recommended that you choose an appropriate scene type that limits the number of players (choose a scene type from the drop down menu when uploading your scene). Please see PS Home Spaces for more information.

These flags can only be set by regional SCE.

To change the flag settings, request which flag you want turned on/off from regional SCE, and ensure:

- you include a submission number
- publish the scene to CDEV only and do not request it to QA.

Metadata that has been applied to a particular scene will be applied every time the scene is uploaded to CDEV. If you wish for it to be removed, please contact your regional SCE.

Environments

ℹ These pages are about Environments in general. For specific information on creating character components in Maya, see the Art Tools > Environment Tools section, which includes:

- Maya Lights
- Lighting Set Editor
- Dynamic Reflection Editor
- LOD Groups and LODs
- Atgi Locators
- Creating Particle Locators
- Cloth Simulation for Meshes

You create environments using the Home Art Tools for Maya included in the HDK. Creating an environment in Maya is the first step to creating a space in PS Home. After you have created your environment, you export it and edit it as a scene in the Scene Editor. When your scene is finished, you package the scene for submission to Quality Assurance (QA).

This section describes how to create environments and scenes that are valid and successfully exportable. It also provides help and guidance on how to create scenes, by outlining the creation process and giving specific examples.

See also how to implement:

- Animated models in scenes. See Animation and Effects.
- Clothing and other character components. See Character Components.
- Collision in an environment. See Environment Collision.
- Furniture and other scene components. See Furniture and Decorations.

The HDK provides a shelf of tools and various templates within Maya to help you create content for PS Home. See The Furniture

Shelf and Furniture Menu and The Character Shelf.

Environment Rules and Conventions

For the recommendations and requirements, as well as a full list of all the validations that the tools automatically run on environments (all types of spaces, models and animated models), see Content Requirements.

Naming Conventions

All assets in PS Home follow the same naming conventions. All names are capitalized and use underscores for separation of category. For example, all assets created for 'Brand Name' would be named 'BrandName_AssetType'.

Examples of correct and incorrect naming:

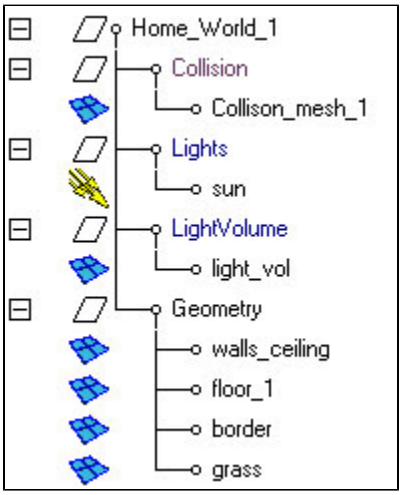
Correct Name Format	Incorrect Name Format
BrandName_DesignType.ma	brandName_designtype.ma
BrandName_DesignType_c.dds	BrandName_designtype_col.dds
BrandName_DesignType_n.dds	brandname_design_type_n.dds

Bear in mind the following:

- Make sure that your naming of folders, scene files, hierarchies, objects and shaders is consistent for debugging and searching tools in the Home pipeline.
- Give objects and shaders within your scene explanatory names, rather leaving the default names like 'Box01' or '07 - Default'.

Hierarchy

The environment scene must follow the appropriate hierarchy:



Each environment must have the following nodes:

Node	Description
Collision	Location for all collision geometry
Lights	Location for sun (light source) and light dome
LightVolume	Location for all Light Volume geometry
Geometry	Location for all scene geometry

Lighting

The environment must have at least one light. All lights must be placed in the Lights group. A light can have any name, and be one of the following types:

- spot
- point
- directional
- area

If the scene is to be sunlit, you must create a sunShape directional light called 'sun'.

To create a sunShape directional light:

1. Click the Create Sun button  on the Home Environment Tool Shelf (Home_Env), to create directional sunlight.
2. Click the Create DomeLight button  , to create diffused light.

> For an explanation of these icons, see The Maya Interface. See also the Introduction to Maya.

Profiling Scenes

Profiling determines how resource-intensive scenes and objects are. Together with the automatic tool validations, profiling enables you to make sure that your content conforms to requirements, runs successfully in the client, and works with all the other content in PS Home.

You have several tools and interfaces available for profiling scenes:

- Use the Debug Console to see your net stats, framerate, mesh count, and other vital statistics, which you should use to ensure that your scene is running within PS Home's requirements.
- Use the `profileEnable 1` console command to see the current GPU and CPU times for entire scenes.
- Use the `enableMemoryStats 1` console command to view the main memory used (split into pools).
- Use the `enableNetStats 1` console command to view the network traffic statistics.

For more information about profiling scenes, see Profiling in the Client.

For more information about content limits and requirements, see Content Requirements.

Environments and Scale

Environment Scale and Dimensions

> For all dimensions in PS Home (including environment dimensions), see Dimensions.

When creating environments, keep in mind the dimensions of all types of content. For example, think about:

- Small spaces - the size of avatars and items that can be placed in personal spaces and clubhouses. Do not make the space too small so avatars cannot move past each other
- Big spaces - the speed at which avatars move. Do not make the space too large so that avatars have to run for a long time to traverse the space
- Seats - must be placed at specific heights to match avatar animations
- Steps - should be at a particular incline
- Surface height - surfaces should be at a height that
- Screens - When making screens, keep avatar height and distance in mind
- Picture frames and hooks - when placing picture hooks, keep picture frame sizes and avatar height and distance in mind

Design spaces where the areas that avatars can move in can be traversed in a reasonable amount of time, or provide alternative methods of moving from one area to another.

For example

If the environment is a series of viewing platforms at the top of the tree canopy in a rainforest, the avatars must be able to move easily between platforms without having to climb down to the forest floor and back up each time. You could create a scripted model that transports avatars direct from one location to another. That way it stays the same space, and is not constrained in size by how far an avatar can walk, but still provides a good experience for players.

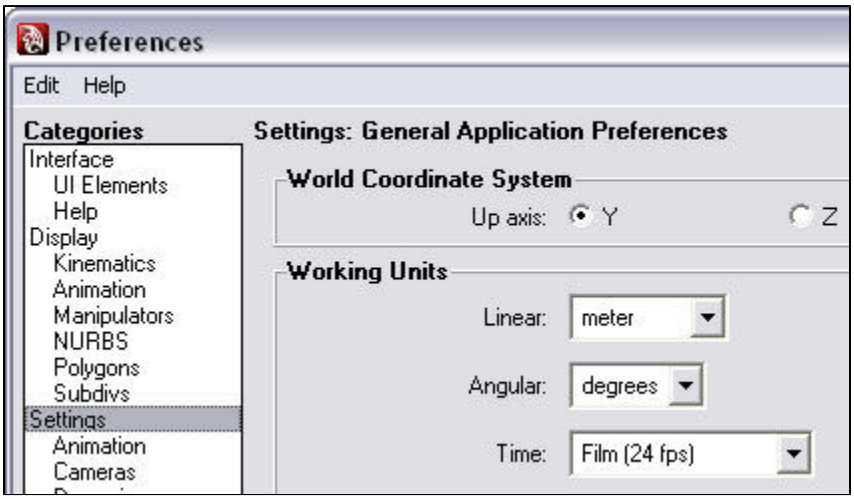
See the section on Dimensions of objects and scenes for a sense of scale in PS Home.

Make sure that you keep to these guidelines so that your worlds have the correct scale and dimensions for PS Home avatars to interact with.

> For a demonstration of what these scales look like in a space, see the Gymnasium sample, available from https://home.scedev.net/projects/samples.

Working Units

Make sure that your Maya Working Units are always set to meters, as shown below:



See also:

- Dimensions for all limits
- PS Home Spaces for information on designing spaces.
- HDK Tools Validations for information on memory limits and other restrictions.

Environment Lighting

This section takes you through the environment lighting process for PS Home. It explains why you use certain effects, as well as how, with the aim of enabling you to produce environments for PS Home, and troubleshoot your own lighting work processes.

How Supernova Works

You do not have to know everything in this section to use the Home tools for environment creation, but it is useful to have a basic knowledge of what happens when you export your work, particularly if you find that your lighting results do not look like you expect them to.

Global Illumination

Supernova is the Home environment lighting render engine. It is a Global Illumination renderer. This section describes Supernova in particular, but the underlying principles are true of many Global Illumination rendering solutions.

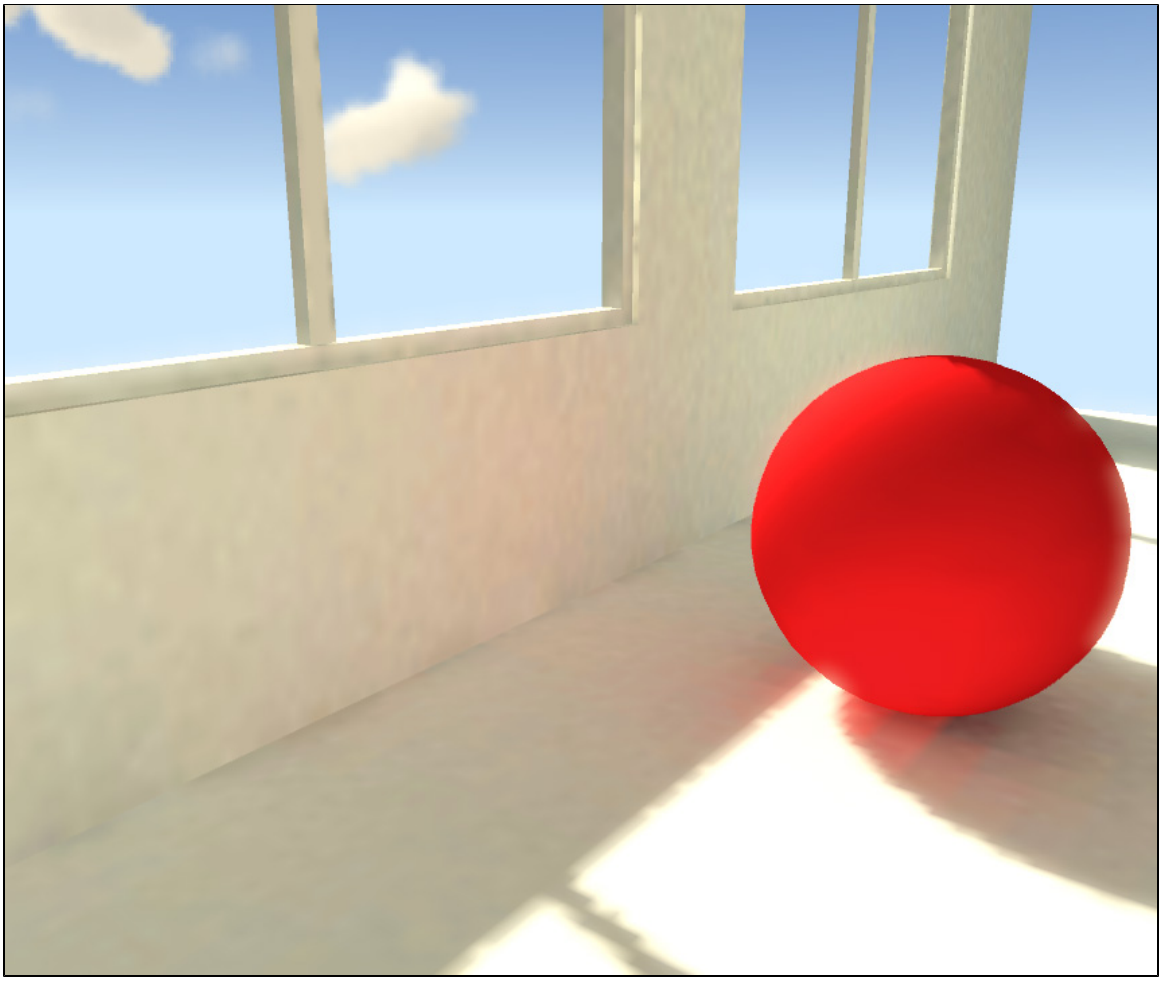
Global Illumination (GI) is a general term that describes a number of lighting algorithms that simulate realistic lighting in 3D scenes. GI attempts to describe not just how objects are illuminated directly by a lightsource, but by the subsequent bouncing of light around a scene. A fundamental concept behind GI is that all surfaces in a given scene can be a lightsource, and should be treated as such when calculating lighting.

The following image shows an example of a section of floor has a bright patch of sunlight falling on it from a skylight:



Notice how the bright patch of light illuminates the area of wall closest to it. The light bounces off the floor, onto the wall. It is particularly noticeable here because the sunlight is intense and the surfaces are white, thus maximizing light reflection. The amount of light that bounces off a given surface depends on a number of factors: light intensity, the surface color, surface angle; but all surfaces reflect light to some degree.

GI also simulates a natural phenomenon known as color bleeding. For example, a brightly lit red ball imparts a slight degree of red to the light that bounces off it. Put it next to a white wall, as in the following image, and you see the effect in action:



These examples illustrate why you should regard all surfaces in a scene as light sources. They all reflect the light they receive to some extend, so they are capable of illuminating other surfaces in the scene. GI renderers calculate the direct lighting in a scene, the subsequent bouncing of light, and then combine the results. These calculations helps to give the final render result a more realistic look.

Note that, for speed and efficiency, Supernova assumes that all surfaces are of a diffuse, non-shiny nature. Diffuse surfaces reflect light in all directions, whereas a shiny surface (such as a chrome kettle in direct sunlight) reflect light in a very distinct way and only in certain directions. Such calculations are referred to as caustic calculations. Calculating shiny reflections takes a disproportionate amount of time for the quality of the eventual results and their visual impact on a scene, so Supernova does not currently support them.

So how does Supernova actually work?

Supernova uses a two-pass solution to light PS Home scenes:

- First pass - Supernova produces a Photon Map and a Radiance Cache
- Second pass - Supernova calculates direct lighting and combines it with the lighting data derived from the Radiance Cachem, using rays.

Photon Mapping and Radiance Cache

Photon Mapping

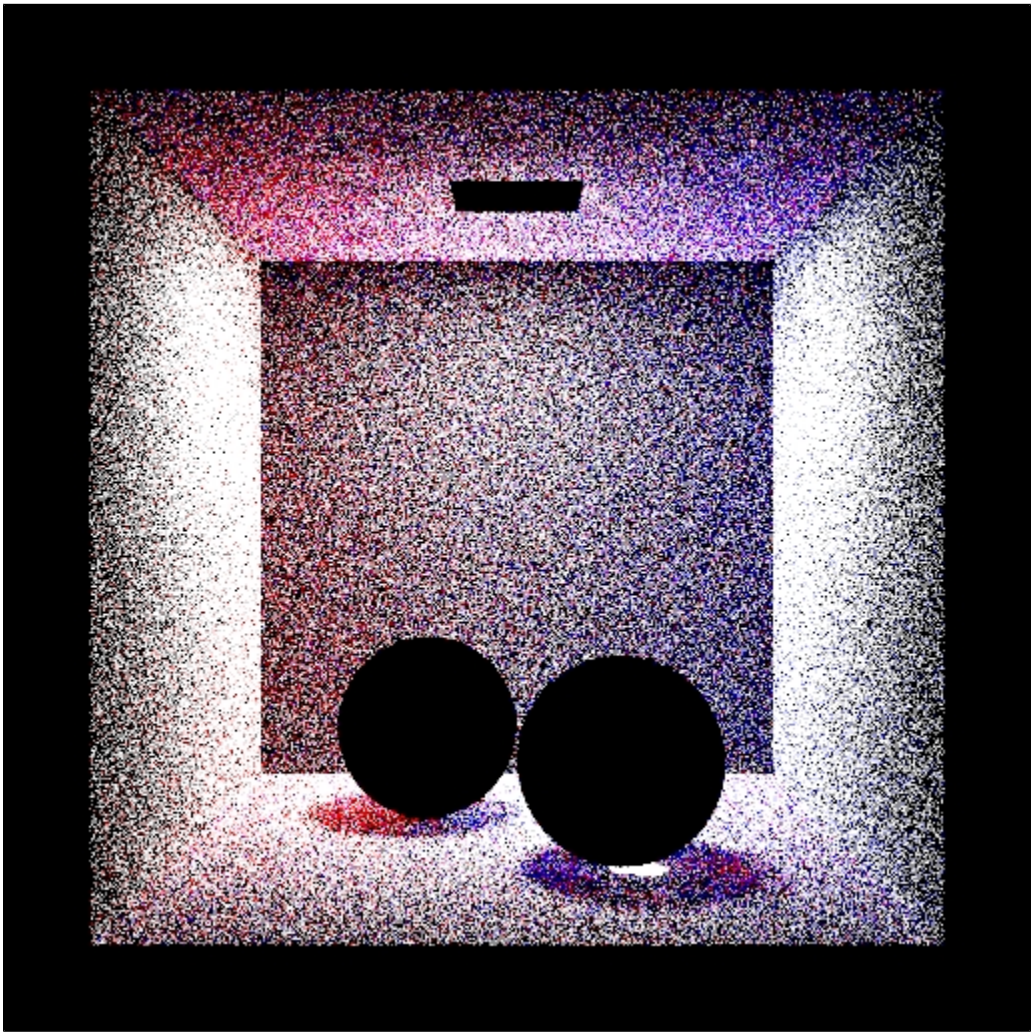
Photon Mapping speeds up the calculation of indirect (or bounced) lighting. In Supernova terms, a photon is a 'packet' of light, fired from a lightsource into a scene. The photon hits a surface, bounces off it, hits another surface, bounces off that one, and so on. Every time a photon hits a surface, information about the intersection is stored in a data structure called a Photon Map. This information consists of:

- Energy of the incoming photon (brightness)
- Color (related to energy)
- Incoming direction
- Photon type (is it direct from the lightsource, or has it already bounced off another surface?)

Photons do not bounce around indefinitely. They lose energy as they bounce and, using an algorithm, some photons are culled when they fall below a certain energy level. Eventually, all are culled when they hit the photon bounce limit, which is set at 9 by

default.

When all the photons have finished bouncing around the scene, Supernova creates a Photon Map - a record of all the photon surface intersections and the data types listed above to go with them. The photon map is a 'rough draft' of the lighting in the scene. Although it is not stored as image data, you can convert it into a viewable state. The following example illustrates that many thousands of intersection points are recorded:



When lighting environments for PS Home, you often use hundreds of thousands of photons to obtain a high quality final result.

This example also shows that there is color to some of the photons. This is because, as mentioned above, the color of a material is taken into account when a photon hits it. In this case, the left box wall is red, and the right blue. The reason the photons hitting those walls are not red or blue is because a photon only carries color information with it after a surface intersection. So the photons are traveling from the white light, hitting the colored walls, and thereafter they carry color information with them that shows up when they next hit a surface.

> In this example, there are three black areas within this photon map. One is the light source, as no photon data is stored from self-lit surface interactions. The other two were shiny balls - the front one glass, and the rear one chrome. No photon data is stored for the balls, as their surfaces would cause the caustic reflections talked about earlier. Such reflections must be calculated in a slightly different way and are therefore separated from the photon mapping process.

Radiance Cache

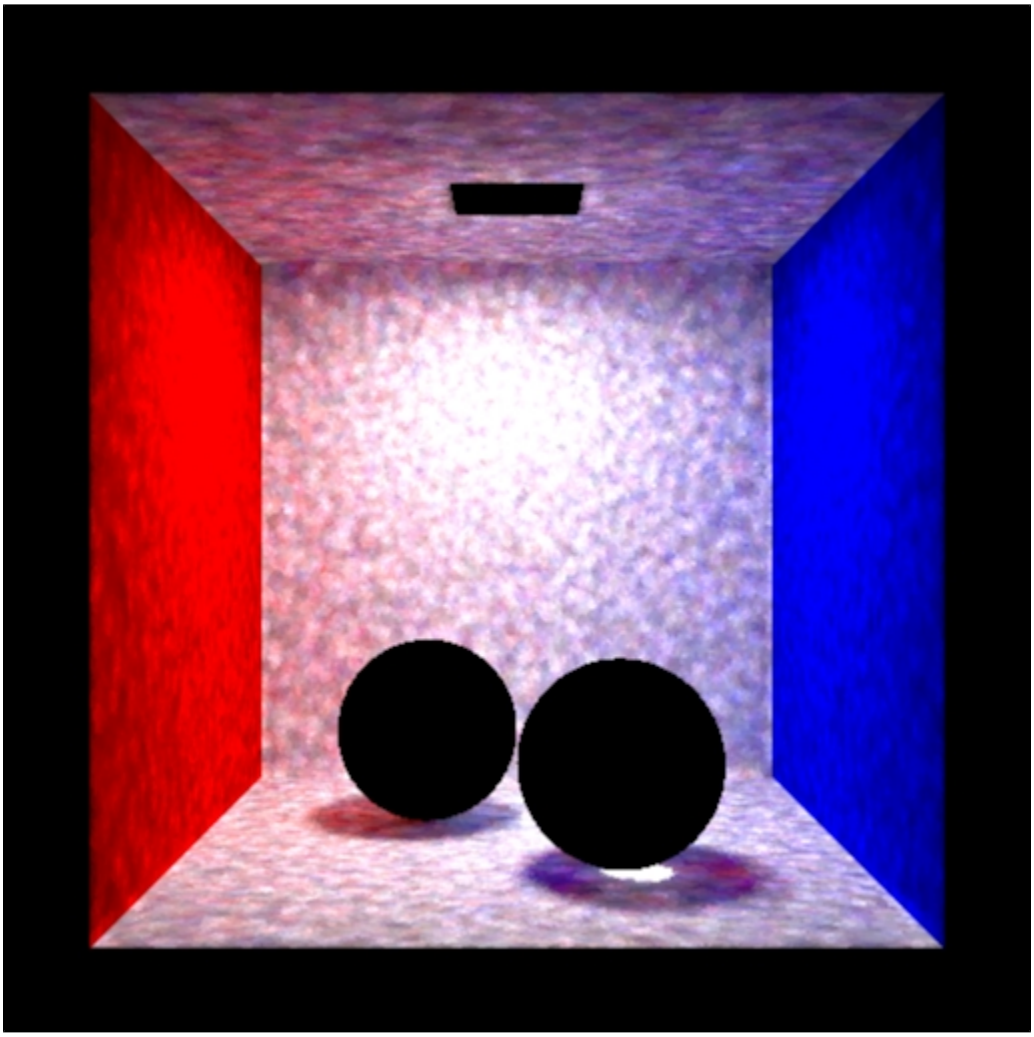
When Supernova has generated the photon map, it then generates radiance estimates. This process turns the photon map data into a form that can be quickly and efficiently interrogated in subsequent steps, speeding up the render process by a significant amount.

Supernova produces radiance estimates as follows:

1. It scans the photon map and selects every fourth photon.
2. It then gathers a random 50 (default value) photons within a hemispherical area around the selected photon, and takes

into account their lighting values. The area is proportional in size to the overall size of the scene.
3. It then 'averages out' the lighting values it encounters in this search to estimate what is called the radiance at this point, or how much light is being reflected from this point. (The process is a more complex than just taking an average, but this is enough detail for an overview.)
4. When Supernova has completed the calculation for every fourth photon in the photon map, it has a new set of data called the Radiance Cache. This cache is a data structure and is not stored in a visual way. However, you can render it to help visualize what is happening behind the scenes.

The following graphic shows that the scene has now been divided into rough areas of radiance. Although this is only a rough image, Supernova has enough data to work with to produce high quality results.



Note how the color bleeding now makes more sense. The Radiance Cache takes into account the surface materials, so you can ow see where the photon mapping was coming from.

Rays

The Photon Map and Radiance Cache provide good approximations of the scene lighting, but to obtain accurate and detailed shadows, Supernova must go into more detail.

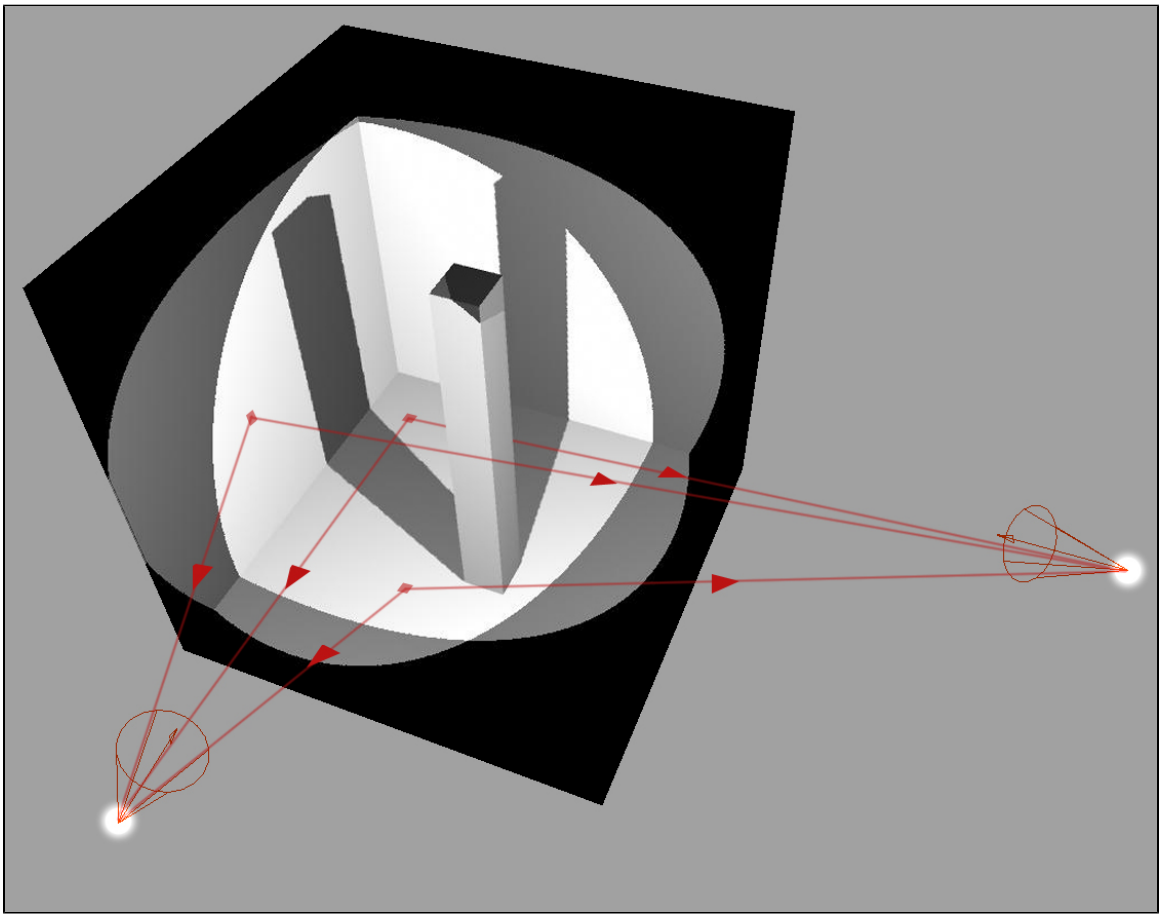
In the second pass, Supernova calculates direct lighting and combines these calculations with the lighting data derived from the Radiance Cache. This data is then rendered out into lightmaps or vertex processed data for use by the game's engine. This process uses rays, which are linear paths shot out from a given point into the scene and which transport intersection data back to their point of origin.

You can use the following types of ray:

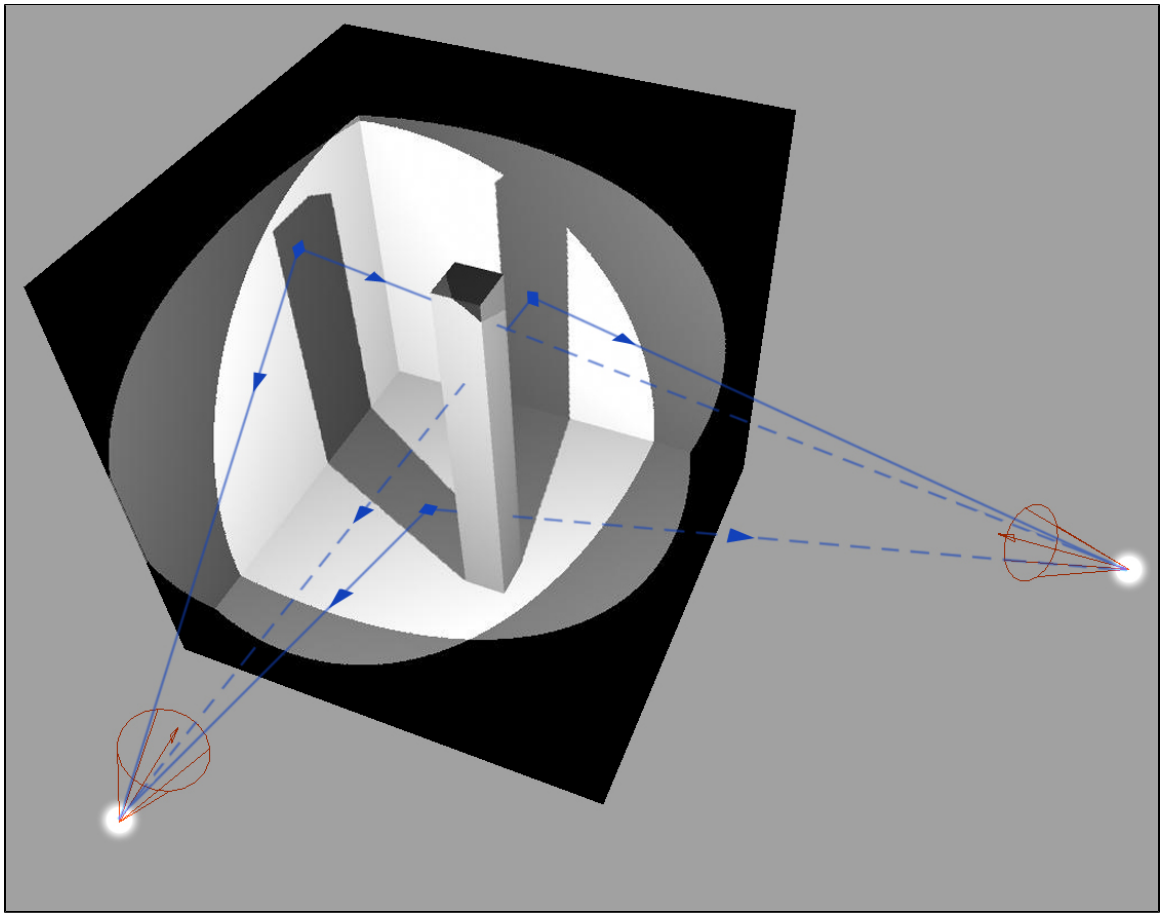
- Shadow Rays
- Final Gather Rays

Shadow Rays

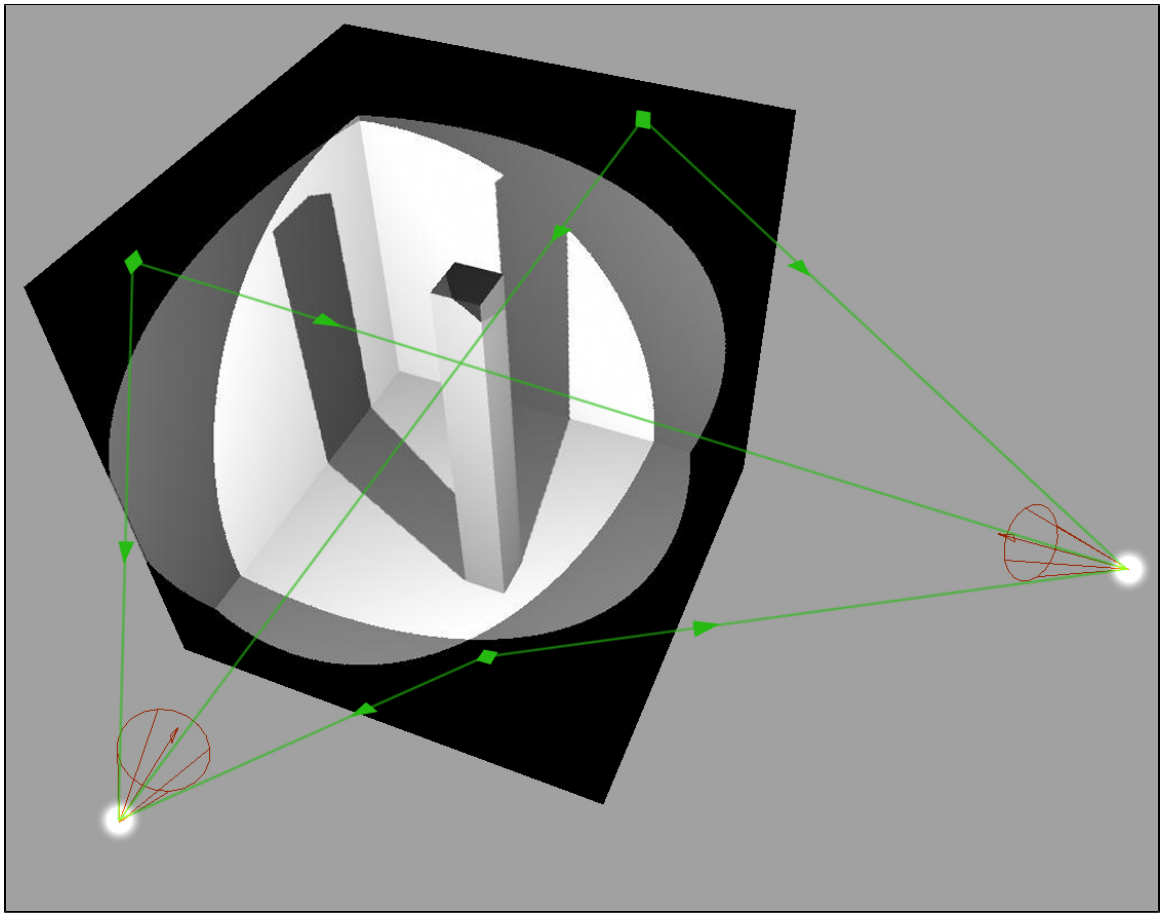
Shadow Rays calculate the shadows in a scene. They are fired from every sample point towards every light source in the scene. The following images show simplified diagrams to illustrate the point. In these diagrams, there is no indirect illumination and all shadows are sharp.



The red rays are being shot from sample points that can see both of the spot lights, so have the brightest illumination.



The blue rays are coming from points that can only see one of the lights - where a shadow ray cannot 'see' a lightsource, there must be a shadow.



The green rays are coming from areas that are outside the influence of either spot light.

For the purposes of Home client, a sample point is either a pixel in a lightmap, or a vertex on a mesh, depending on whether the object has been set to be lightmapped or vertex lit when exported. If a sample point can 'see' a light, the illumination from that light can be calculated, based on distance and angle. If it cannot see the light, it receives no illumination from that light.

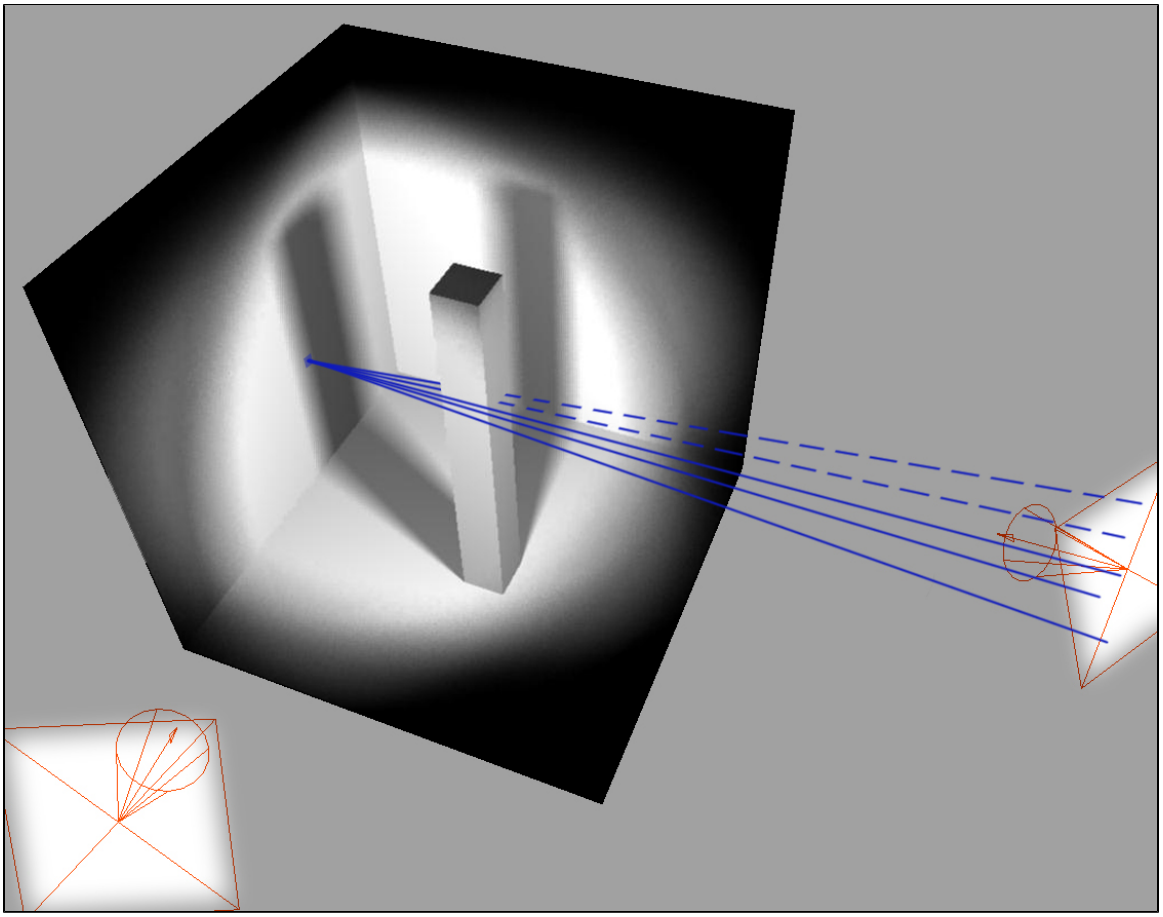
When shadow rays have been fired at all lights in the scene, the results are compiled and the amount of direct illumination at any given point is known.

> ⚠ These calculations are carried out for every lightmap pixel or mesh vertex, so increasing the number of shadow rays can increase the render time for your scene by a noticeable amount.

Supernova uses only a single shadow ray for delta lights (lights with hard edged shadows, as in the shadow ray diagrams above), because sample point can either see a delta light, or it cannot. Delta lights are not physically realistic though, as almost all lights in the real world have area.

Area lights are used extensively when lighting PS Home scenes in order to get more realistic results. For example, you can have actual Maya area lights, a sky dome, an infinite disc for sunlight, soft edged spot lights. Supernova uses multiple shadow rays for area lights, to calculate soft shadows. Because an area light actually has area, it means sample points might be able to see only a certain percentage of that light, leading to soft shadows. This type of lighting requires multiple shadow rays to produce an accurate calculation.

In the following diagram, multiple shadow rays are being fired at a light source. This example uses the spotlights from the previous example, but they have now been given area:



In this example, two of the shadow rays cannot see the area light because of the column in the scene, but three rays can see a percentage of it. In this case, the sample point is neither in full shadow nor in full illumination. The number of shadow rays that can see the light in this graduation from shade to light causes the shadow edge to be 'soft'. Note that this is a very simplistic example. In practice there could be many area lights and hundreds of shadow rays.

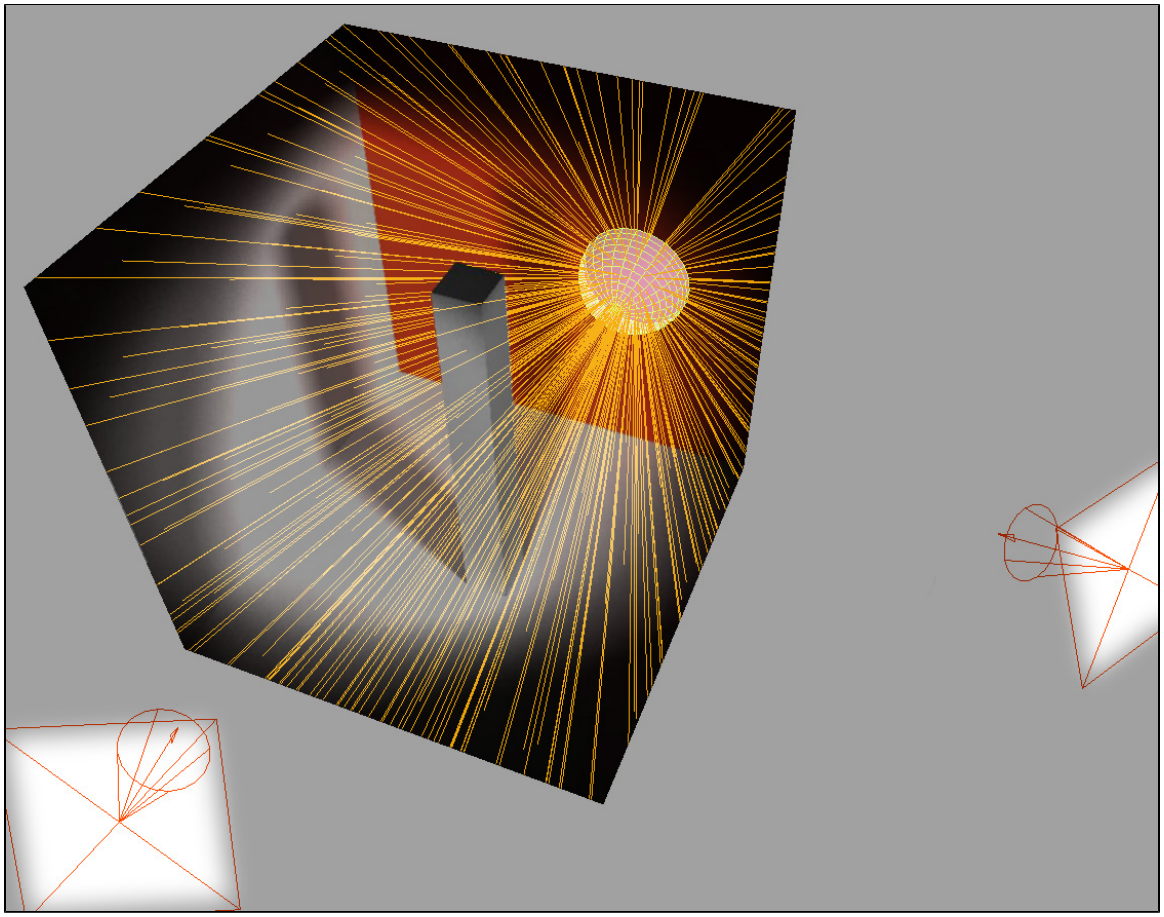
Supernova allocates these shadow rays in a 'weighted' way, meaning that the more illumination a given sample point is judged to be receiving from area lights (measured by the photon map), the more shadow rays are used to calculate soft shadows. Supernova does not fire any shadow rays at an area light that is deemed to have an influence below a certain threshold at a given sample point. This is an optimal way of using shadow rays and speeds up render times.

When Supernova has calculated the direct illumination, it calculates the indirect lighting using Final Gather Rays.

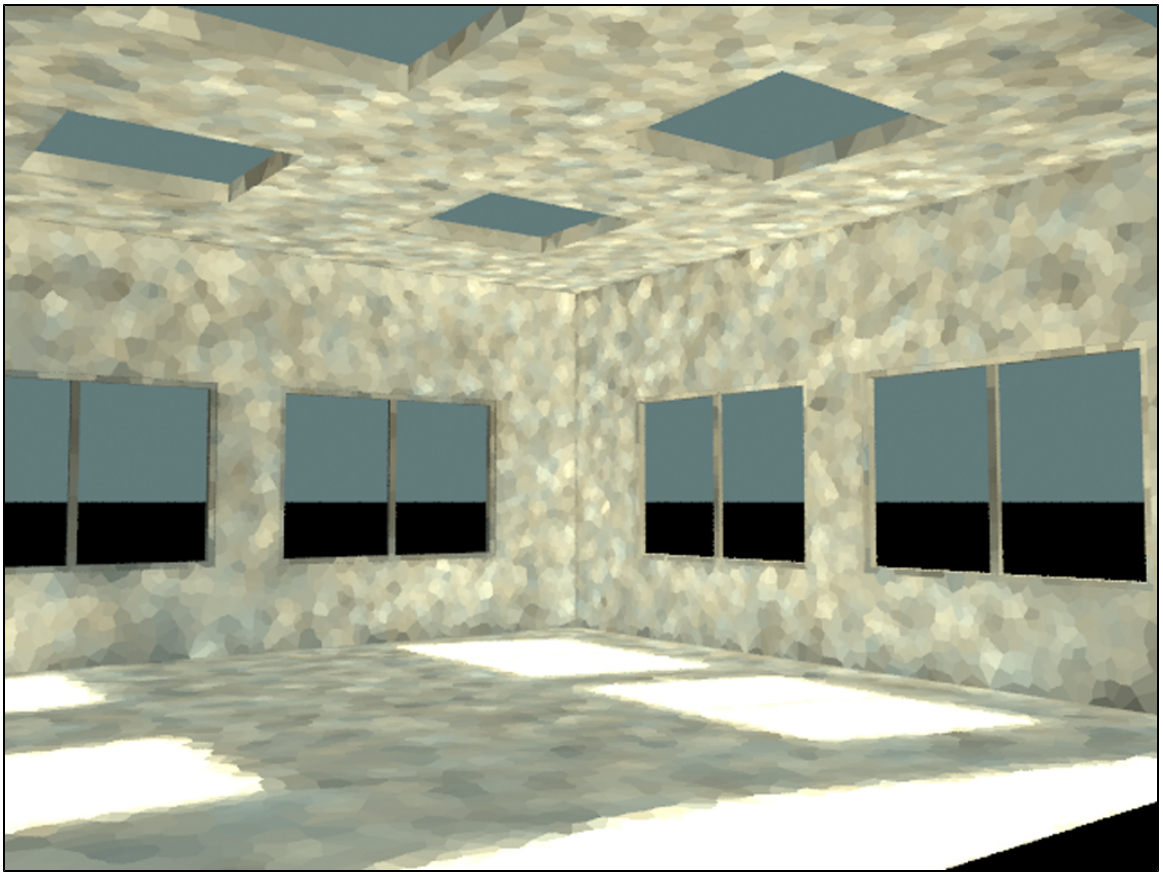
Final Gather Rays

Supernova has already calculated the radiance cache, an efficient data structure, and a rough representation of the scene's indirect lighting generated from the photon map. It now uses the cache to calculate the final indirect lighting.

Final Gather Rays work in a similar way to shadow rays. They are fired from each sample point in the scene (either a lightmap pixel or a mesh vertex), except that they are fired out in a random hemispherical spray, as shown in the following diagram:



When they hit a surface, the rays use the Radiance Cache to calculate the amount of light (radiance) coming from that point. The color of the light is also taken into account so that Supernova can also calculate color bleeding. The following image shows an actual radiance cache visualization from a Home test environment:



Although it looks rough, images can be sampled many times over in many different places for just a single lightmap pixel or mesh vertex. When all the sample values are collated, the result can be extremely accurate.

> ⚠ This step of the lighting process takes up most of the total render time. Be careful when setting the number of final gather rays if you want a quick render.

The reason that this stage takes the longest is that there is no easy optimization for the number of final gather rays that have to be fired out, as is the case with the shadow rays. Shadow rays need only be fired at lightsources, and can safely ignore the ones that do not have much influence. Indirect (bounced) lighting can be everywhere in a scene, so everywhere has to be sampled to get an accurate reading of the incoming indirect lighting.

For example, if you specify your export to use 1024 final gather rays (which is a good number for a decent quality result), Supernova must:

- fire 1024 rays from every pixel or mesh vertex in the scene
- combine all 1024 radiance values it receives back from the Radiance Cache
- combine that with any direct illumination that may be present at that sample point (although direct illumination normally overrides the relatively subtle levels of indirect illumination) , to give you your pixel or mesh vertex lighting value

These calculations are computationally intensive. If you then factor in the potential exponential increase in lightmap pixels for higher resolution renders (a 128 x 128 lightmap with 16,384 pixels at low resolution could become a 1024 x 1024 lightmap with 1,048,576 at high resolution) it becomes apparent why render times will also increase exponentially.

However, modern CPU's can run these sorts of calculatioln efficiently, and Supernova can take advantage of distributed rendering to ease the load further. This, combined with HDK tool functionality that allows selective lighting of scenes, means that despite the heavy CPU workload, lighting PS Home scenes remains a practical task for artists.

When these calculations are completed, the lightmaps are rendered out or the vertex lighting levels are baked into the relevant meshes, and the lighting process is complete.

Dynamic Lights and Deferred Rendering

There is currently no hard limit on the number of dynamic lights you have, as long as they are spot or point lights with shadow casting disabled. However, these lights can be fairly expensive in frame rate terms, depending on their use. Use as few as possible.

PS Home uses deferred rendering for dynamic lights. This means that the amount of work it does for each light depends on the

number of pixels that the light is currently affecting:

- If a dynamic point light is distant, PS Home does very little work.
- If it is illuminating the majority of pixels on a screen, PS Home has a lot of work to do.
- Multiple dynamic lights with overlapping influence, so that several calculations must be carried out per pixel, incurs a greater overhead.

The advantage of using deferred rendering is that the number of triangles being illuminated is irrelevant, so you have no problem with many characters or complex geometry being lit with these lights at the same time.

There are no hard and fast rules, and the amount of processing required depends on the scene. Bear it in mind that if your frame rate starts to suffer, deferred rendering might be a cause.

The total number of meshes in a scene (in the runtime) should not exceed 4096. This includes:

- reflection views
- dynamic shadows
- all objects
- the avatar's Menu Screen

If you add realtime spotlights (with shadow casting enabled) and reflections, the number of rendered meshes can increase considerably. Be careful not to add too many spotlights with shadow casting enabled, or realtime reflections. Alternatively, you can reduce the number of meshes in the scene itself.

Dynamic Shadows

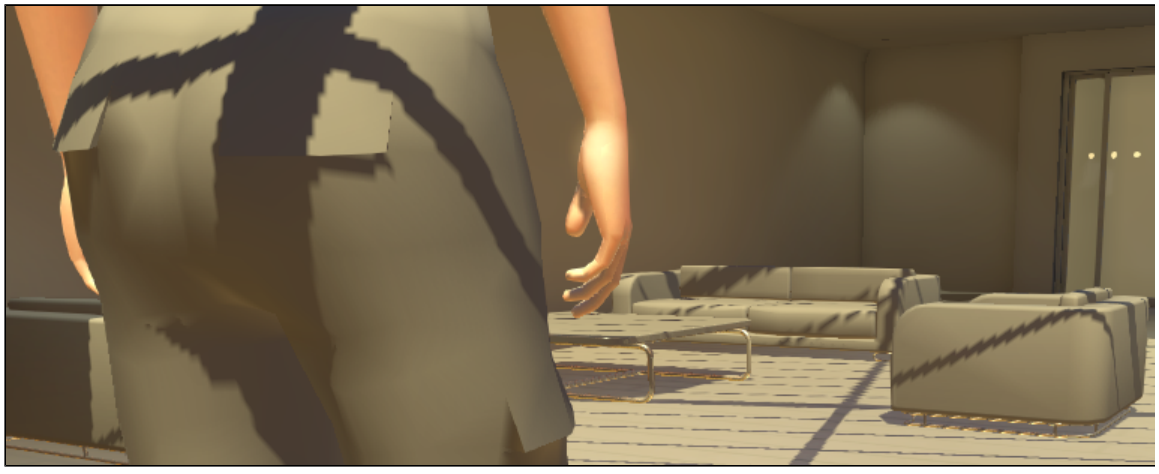
You can control which items cast dynamic shadows by setting the geometry on the items to override default behavior. This means that you can "switch on" dynamic shadow casting for items that normally do not cast shadows (such as semitrans materials), and "switch off" dynamic shadow casting for items that normally do cast shadows (such as opaque or cutout materials).

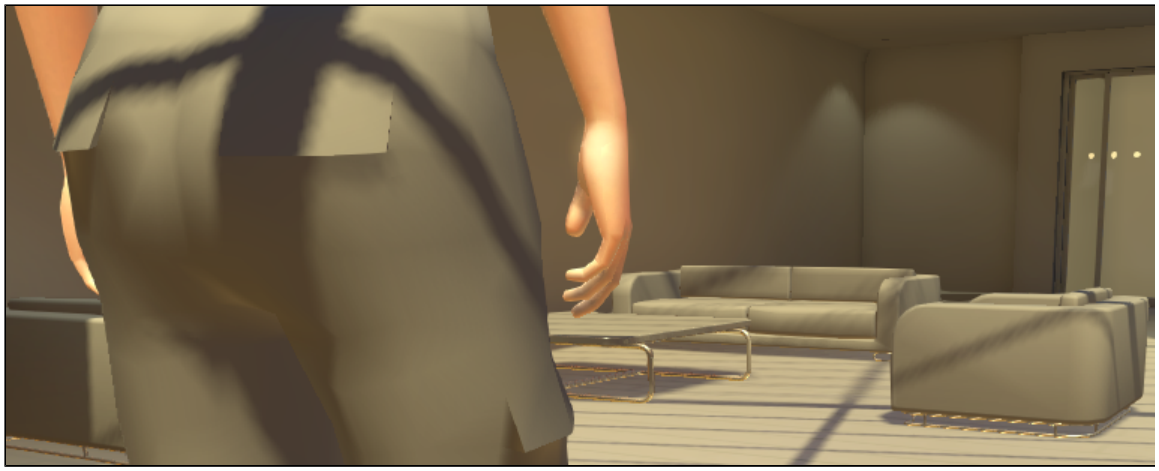
You can change the quality of shadows cast by dynamic lights or the scene's sun:

- For dynamic lights, call the Lua function `Light.SetShadowQuality()` to set the shadow quality.
- For the scene's sun, call`Scene.SetGraphicsEngineParam()` and use one of the `GraphicsEngine.ShadowQuality` values.
  You can set the shadow quality to low, medium, or high. The quality of the shadow specifically refers to the quality of the penumbra region, which is the area that transitions between completely lit and completely shadowed. The higher the shadow quality, the smoother (and less blocky) this area looks.

Note that the higher the shadow quality setting you use, the slower the scene will render.

The shadow quality versus frame rate trade off is scene and camera specific. The following figures show a typical comparison.

Quality Setting	Render Time	Sample
Low	29.02 ms	

Medium	29.64 ms (Low frame rate + 0.62 ms)	
High	31.41 ms (Medium frame rate + 1.77ms)	

Environment Lightmaps

UV Layout

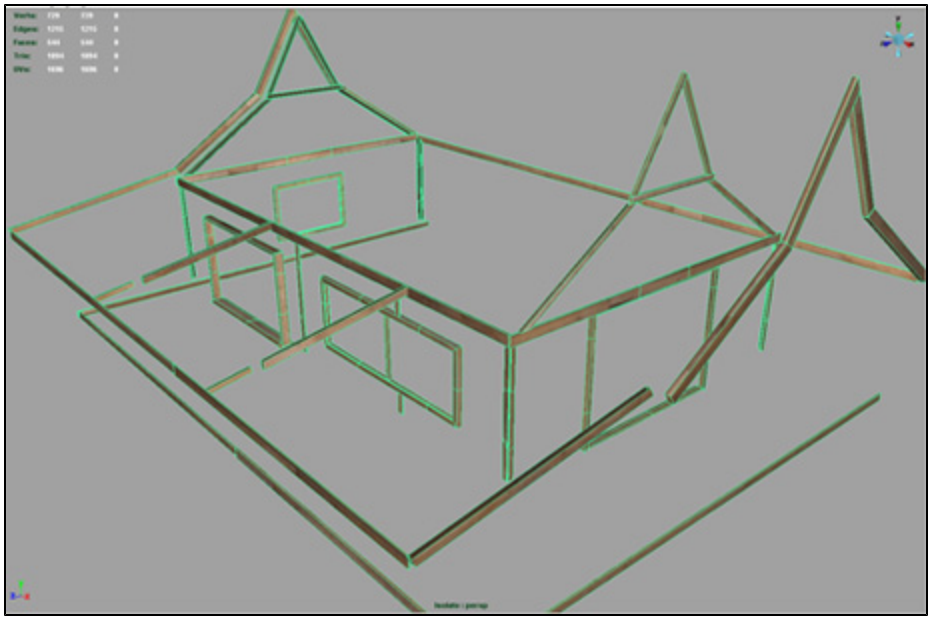
You will need two UV sets: map1 and map2. Make sure your poly objects do not have more than the 2 UV sets.

The default UV set map1 is used for your color maps and can be organized to suit your texturing. The aim is to texture these models with tiling textures.

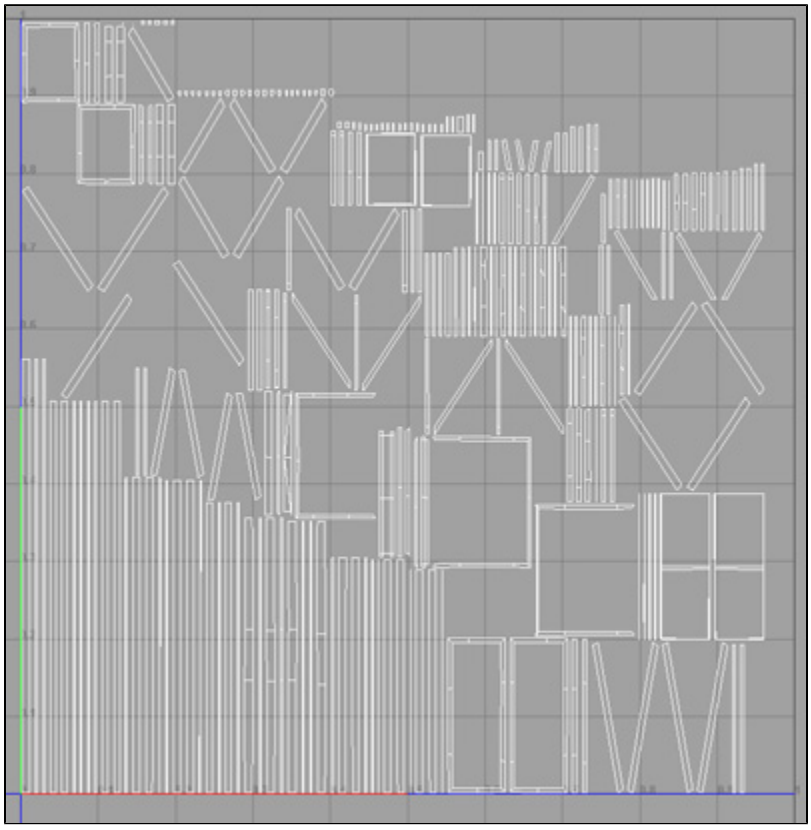
Every poly-object within the model needs a second UV set called map2 which has UV coordinates (all within the 0-1 range), if it is to be lightmapped. It is advisable to copy your color UV set into map2 and use the Layout UV tool. The layout for map2 is not important and simply needs to be present for it to work within the Home engine.

The Automatic Mapping tool in the Create UVs drop-down list can be used to fit the UV sets within the 0-1 range for map2. Automatic mapping works well for objects that are largely rectilinear and that have not been rotated along any axis away from the standard XYZ planes, but may be less effective if this is not the case or if the object in question has a more organic shape.

The following image shows an example mesh:

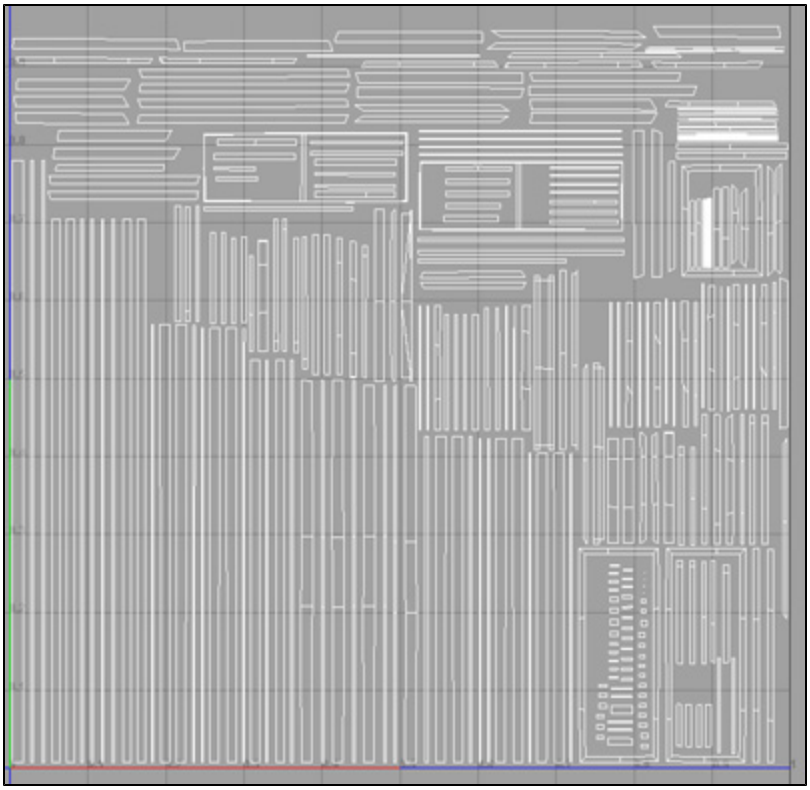


The next image shows the example mesh's initial Automatic Mapping layout:



This is a good example of the Automatic Mapping tool producing a fairly inefficient layout with regard to UV 'packing'.

In the image below you can see a far more efficient layout that an artist has created by hand. Note how they have rotated the diagonal parts to be straight and packed everything tightly together:



It's important to layout map2 UVs carefully to make best use texture memory. The larger the scene, the more important these issues become.

For each mesh in question, the size of the lightmap is calculated according to the average surface area of each of the lightmap polygons. So if you have specified a lightmap texel size of 10 cm, the export pipeline will look at the surface area of every polygon on every mesh and average them out - this figure can then be used to decide what size texture will be needed to allow a lightmap texel to be sized at 10 cm in the PS Home client.

You can modify texel size per mesh by adding to it a custom attribute of float type named texelScale.

This acts like a scale value to the texel size. However, this process assumes that the UV layout is correct and directly proportional to the actual geometry. If it is not, you get the sort of issues described above.

Never change lightmap UVs independently of each other. If there is a UV section that has been scaled independently of other pieces, it will disrupt this calculation and cause the lightmaps to be generated too big or too small.

For example, say you have a lightmap UV layout where a UV section has been scaled right down (because, for instance, it's not visible to the player in PS Home). The exporter will always attempt to create a lightmap that will have the specified texel size in the client. Think of the texel size as the maximum size the exporter will allow a lightmap texel to appear. A small, independently scaled UV section will mean the lightmap will have to be enormous to get the texels of that small section to appear at the specified scale in the client.

Always use a UV layout texture when doing lightmap UVs to ensure they are all in proportion. If you find you are getting smaller lightmaps even when the UVs are correct, it is likely that the geometry in question might have scale-transforms on it that can confuse the polygon surface area calculation described above. Freeze the transforms wherever possible before export.

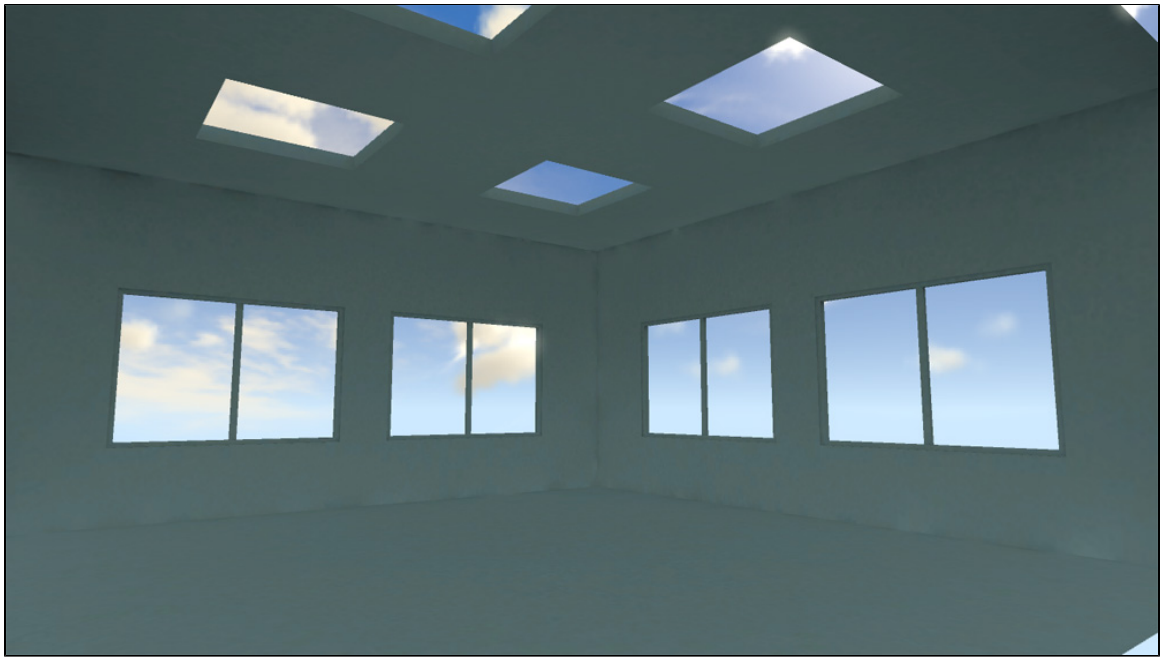
LOM Lighting

This section details the LOM lighting model that can be used when lighting Home environments, one of two available.

This lighting model is for use in scenes that have a single primary light source. Most of the time this will be a daylight scene that has direct sunlight (rather than an overcast day). It could also conceivably be used for other types of scene, for example one with a full moon.

How this lighting model actually works can be a little confusing at first, as it is a combination of dynamic and static lighting. Two main light sources are used in this model, a directional light that will usually be the sun, and a dome light. This is hemispherical by default although it can be turned into a full sphere. This simulates the diffuse illumination that comes from all parts of a daylight sky.

The following images illustrate these two lighting components individually, and then how they combine. Firstly, this is an example scene lit only with the skydome:



The skydome produces a low, even level of lighting all around the scene, as light is arriving from all around with equal brightness. This is how the sky illuminates things in the real world. There are very few surface areas in this scene that cannot 'see' any part of the skydome, only the ceiling and the very upper parts of the walls, but they are nevertheless being illuminated because of the amount of light bouncing around the scene from all directions.

Here's the same scene, but now lit only with the directional (sun) light:



This looks better, but considering these surfaces are white the shadows are too dark and the yellow color of the sun is having too much of an effect. Many surface areas of this scene cannot see the sun light directly, but in the same way as with the skydome, they are being illuminated by the light bouncing around the scene. Without the indirect lighting (see How Supernova Works), then any area that could not directly see the sun would be pitch black.

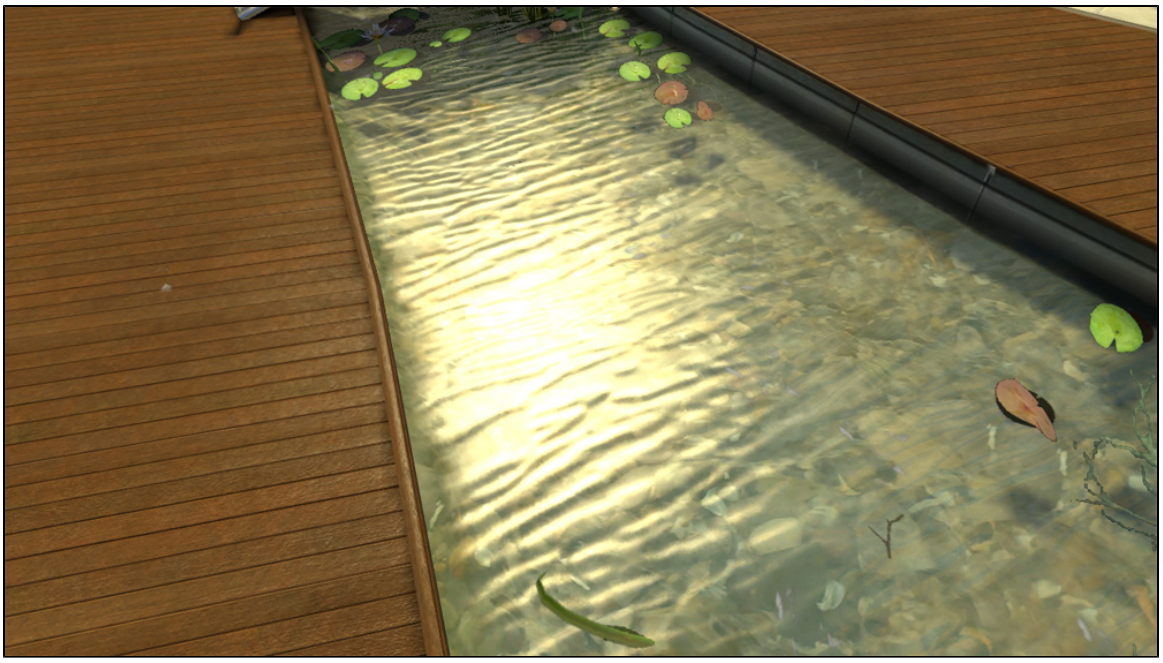
If we combine these two lighting elements, we get this:



The differences are subtle but very noticeable. The shadows have been lightened by the skydome, the color balanced by the slight blue tinge to the sky. This gives a realistic approximation of how this room would look if lit by the sun in the real world.

The most important thing to note about the directional light is that it is dynamic, despite the fact that it is used for generating static lighting. There are two principle reasons for using a dynamic light in a scene with static lighting. Firstly, it provides a physically accurate specular highlight which goes a long way towards making objects and environments look more three dimensional.

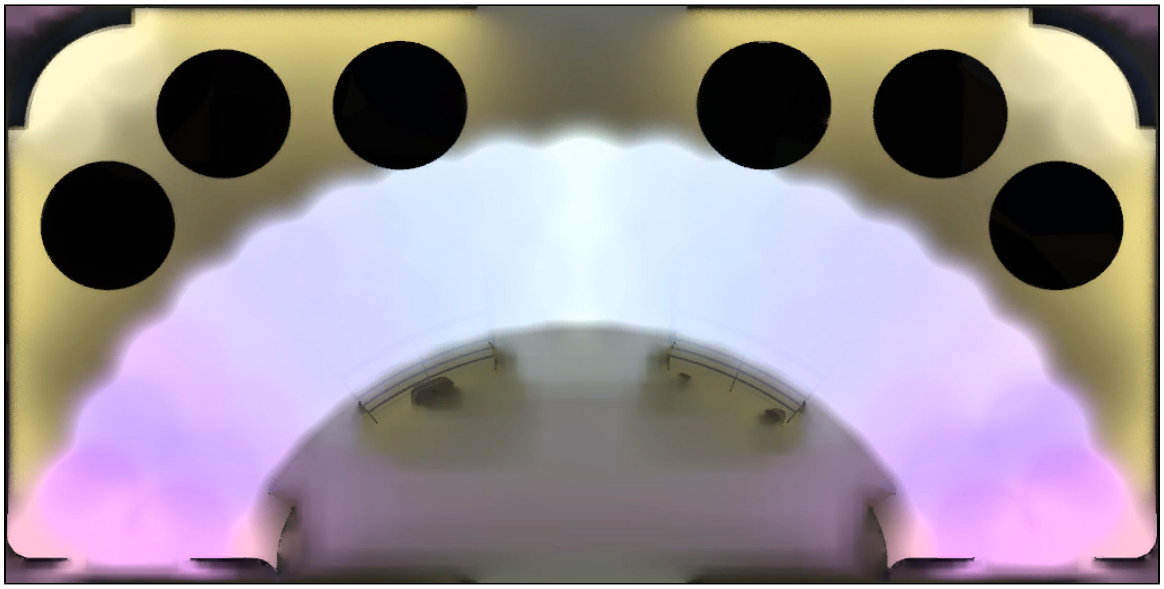
Here's an example of a specular highlight on water:



Secondly, it means that only one lightmap is generated for any given object/surface, as opposed to the three that are generated through the other lighting model, Directional Lightmaps and this saves on VRAM. Only one lightmap is needed because, by using a dynamic light, normal maps will affect the lighting in a physically correct way. The Directional Lightmap model does not use a dynamic light, so needs to generate three lightmaps per surface to get normal maps working correctly.

LOM Lightmaps

A conventional lightmap, also sometimes known as a lightbake, is a texture that's combined at runtime with diffuse, normal and specular surface textures to describe the lighting on a particular surface. Here's an example, an old one that's no longer used in game but a good enough example of a light bake style lightmap, from the Home Game Space floor:

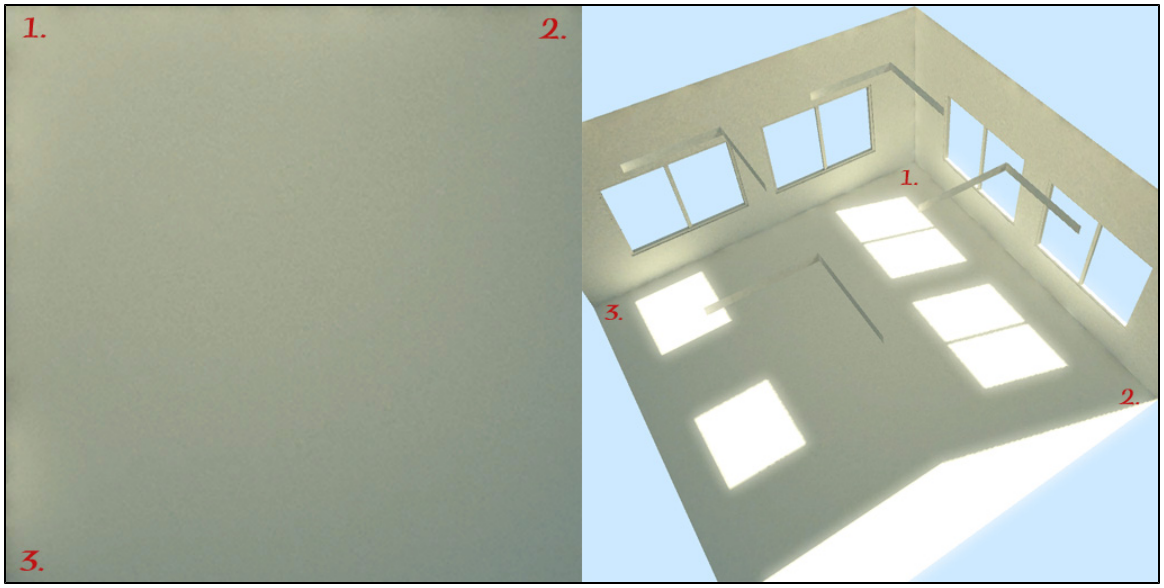


And here's the game space itself to give it context:

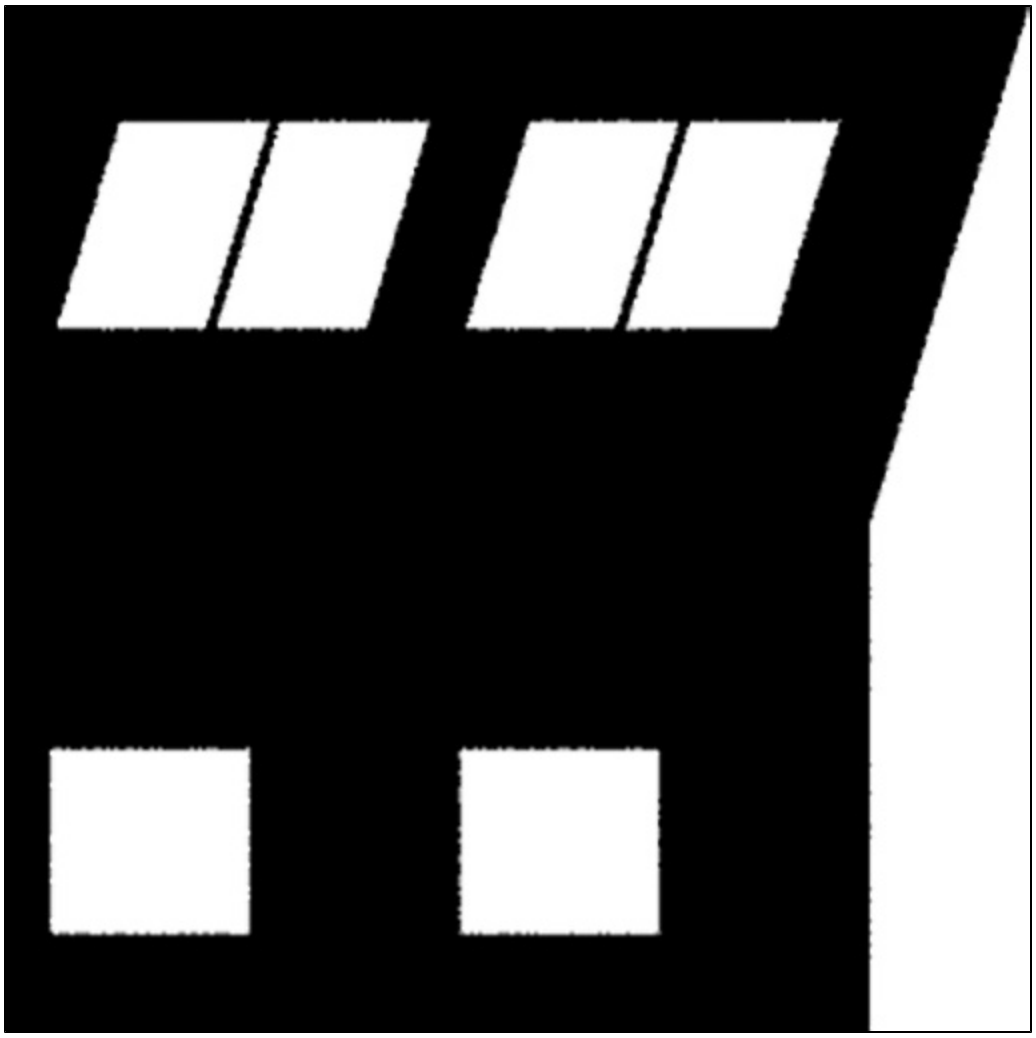


LOM lightmaps are a little different. Instead of containing all lighting data within the RGB channels as in the example above, LOM lightmaps only store indirect lighting data within the RGB channels.

Here are the RGB channels of the floor LOM lightmap from the example scene, along with markers to show the orientation:

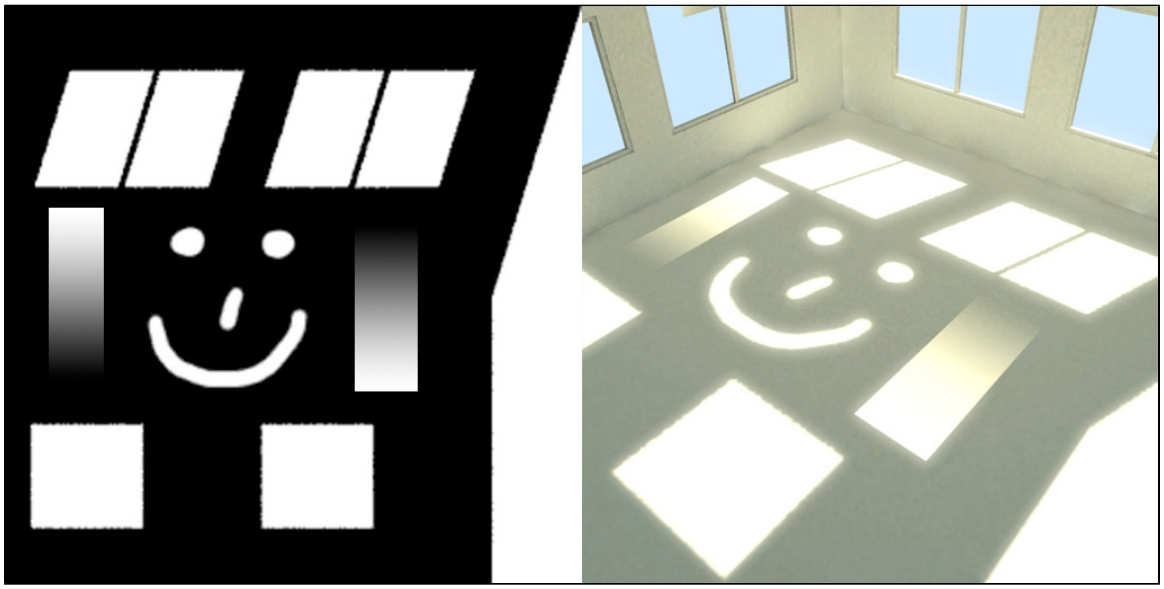


It's clear that there's no direct lighting information visible here, only the indirect. The bounced lighting from the walls is visible as a lightening along the left and top sides of the lightmap. LOM lightmaps also use an alpha channel, so here is the alpha channel from this lightmap:



The direct lighting component is obvious here. LOM lightmaps use the alpha channel as a sort of mask. The dynamic sun light illuminates everything by default, but the alpha channels of the LOM lightmaps generated during the export process merely control where that light is allowed to be rendered, or to put it another way, where the sun light is occluded, hence the name Light Occlusion Maps.

Here's what happens if we draw on the alpha channel in Photoshop to illustrate this:



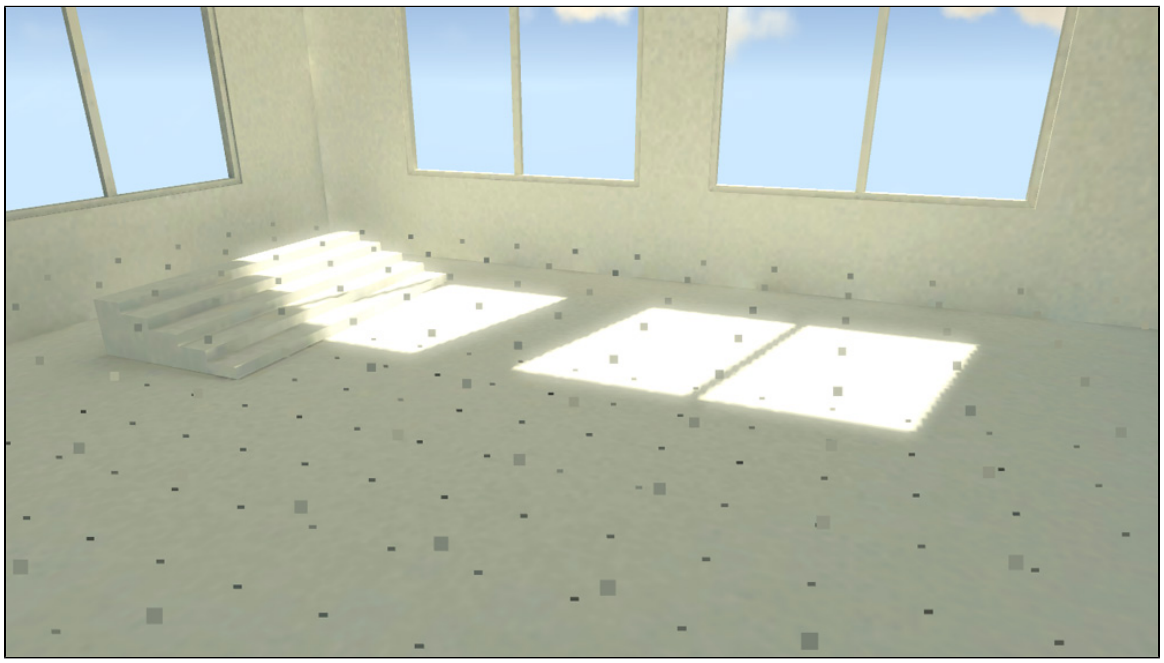
Note how the amount of light can be graduated - it's not the case that the sun is either on or off, as the more oblique an angle a surface has to the sun, the less light it will receive. So, LOM lightmaps contain information not only about indirect lighting but also where the direct lighting is occluded.

However, that's not all the data required. The game also needs to know the direction of the sunlight, its intensity and its color. This data is contained with the scene's Light Probes.

Light Probes

Light Probes in Home are points of lighting data, spaced equally throughout an artist-defined three dimensional volume. The spacing can be set at export but is generally left at the default values.

Here's a rough in-game visualization:



Each point contains data concerning the direction, color and intensity of light at that point in space. With the Directional Lightmap Model, probes are used purely to light dynamic (moving) objects. These include avatars, furniture, mini-game components and so on. As the objects move around the game world they take their lighting from the nearest probe, smoothly interpolating between one probe's embedded lighting values and the next.

Because the probes are created using the same lighting algorithm calculations as the static lighting, they should 'match up' with

their environments and provide convincing lighting for dynamic objects.

With LOM, the probes also dictate the main scene lighting, specifically its direction, color and intensity. The relationship between the LOM lightmaps and the Light Probe data can sometimes be a little confusing.

For instance, this is what happens if we change the color of the sun, change its intensity, move its position and re-export the lightmaps, but without re-exporting the probes:

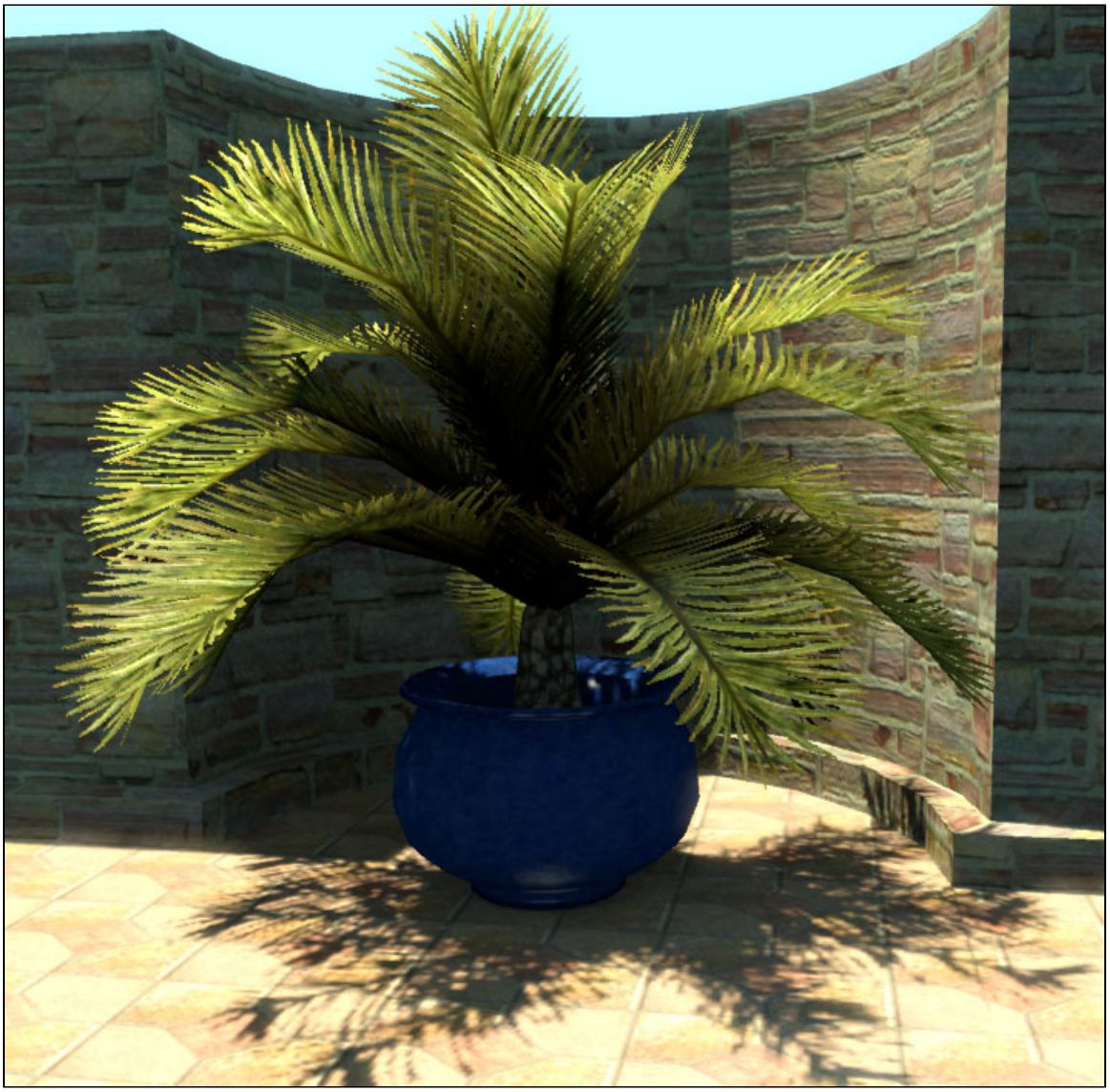


Here the sun has been changed to a pinky red color and shifted around so that it is shining through both sets of windows. The lightmaps have been regenerated and are behaving correctly. The alpha occlusion channel has also been updated to show the new sun position and the indirect lighting is showing a pink hue as it was calculated with the new sun parameters. However, the direct lighting is now incorrect as it's still the original color and direction. Once we have re-exported the light probes as well, we get this:



The direct lighting is now corrected and matches up with the lightmaps. This discrepancy between the lightmaps and light probes will not generally be a problem as, in our experience, changes to a scene's sunlight parameters are rare once the initial lighting has been settled upon by the artist. However, if changes are made to the sun parameters, then the artist will need to re-export the probes as well as the static lighting to get the lighting updated correctly.

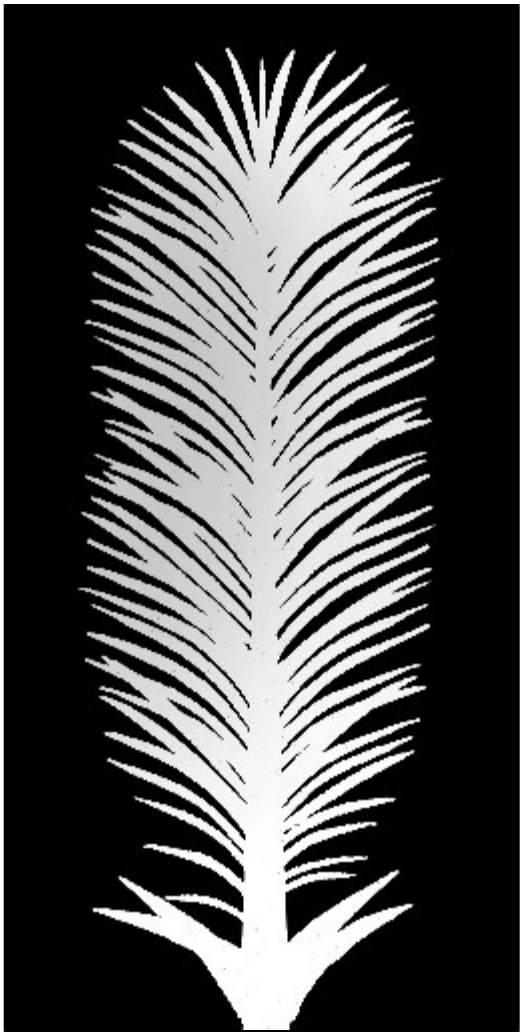
Rendering Shadow Detail



Often the artist will want to render shadows in their lightmaps from texture alpha. A typical example of this is that of foliage. Individual leaves, or in this case palm fronds have to cast shadows from the image alpha otherwise they would simply render the shadows of the quads to which the textures are mapped.

The following section explains how to achieve shadow detail as in the image above.

The first step is to set up the texture correctly. It is the texture's alpha channel that the render engine will reference to generate the occlusion data (shadows) so the texture must contain an alpha channel. In the case of our example the image alpha looks like this:

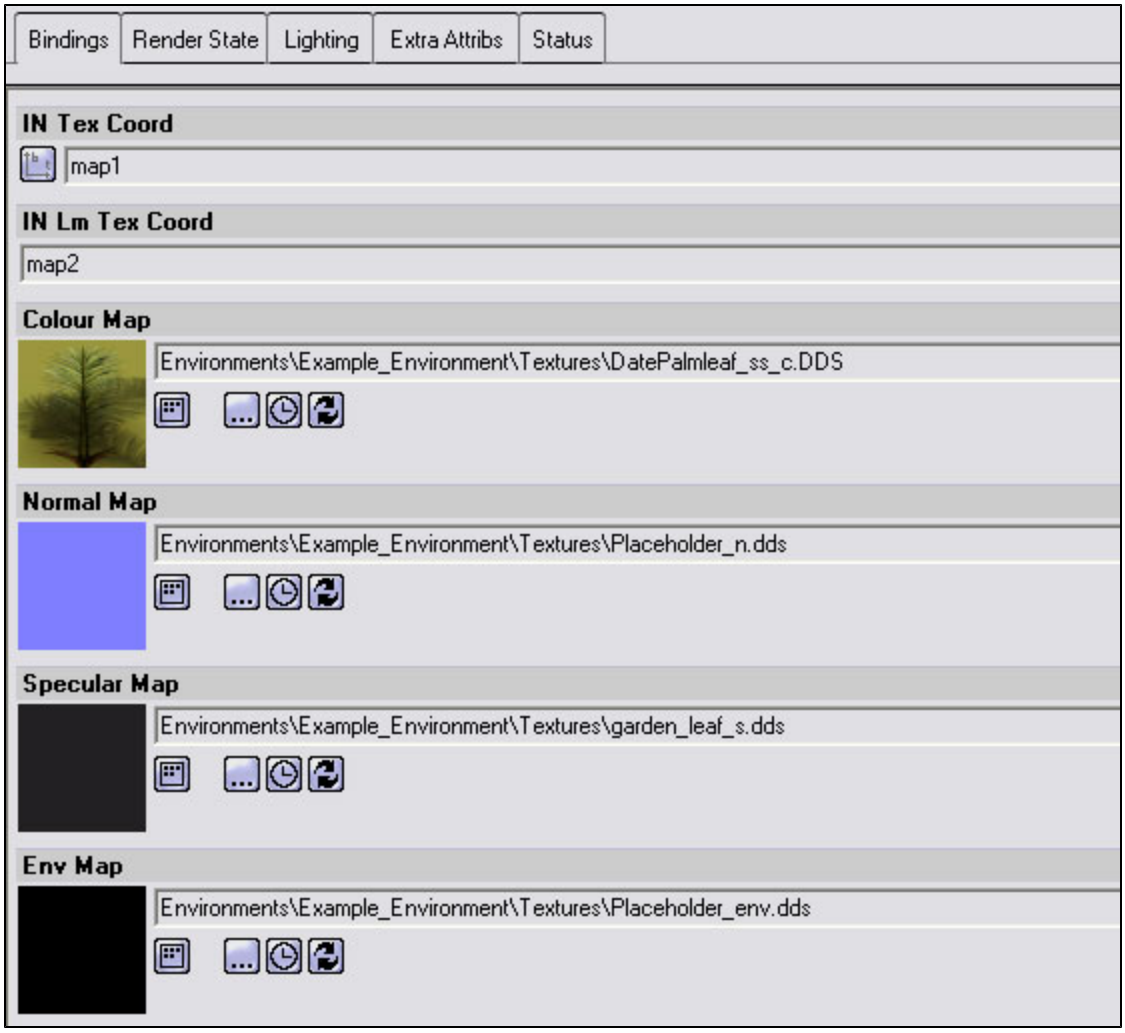
For texture memory (file size) and reliable alpha sorting on the PlayStation®3 one bit alpha (black and white - no shades of grey) is employed and saved in DXT1 ARGB (1 bit Alpha) DDS format. 8 bit alpha can also be used if translucency is required, but bear in mind that this will increase file sizes accordingly and will almost certainly result in alpha sorting artifacts when rendering in-game.

Alpha Sorting

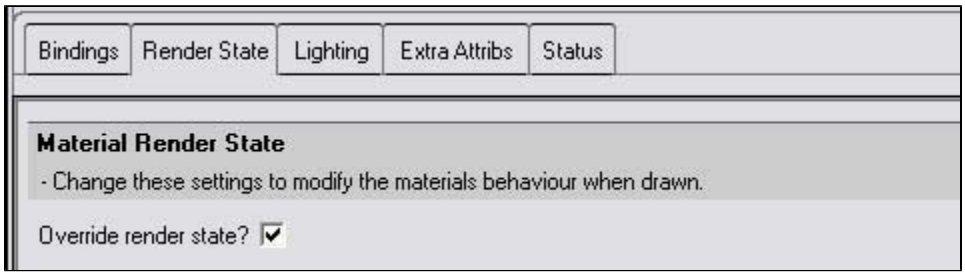
The default setting for alpha sorting is `True`. You can change this behavior using Lua, by calling the function `Scene.SetGraphicsEngineParam(AlphaSortEnable, false)`.

Shader Setup

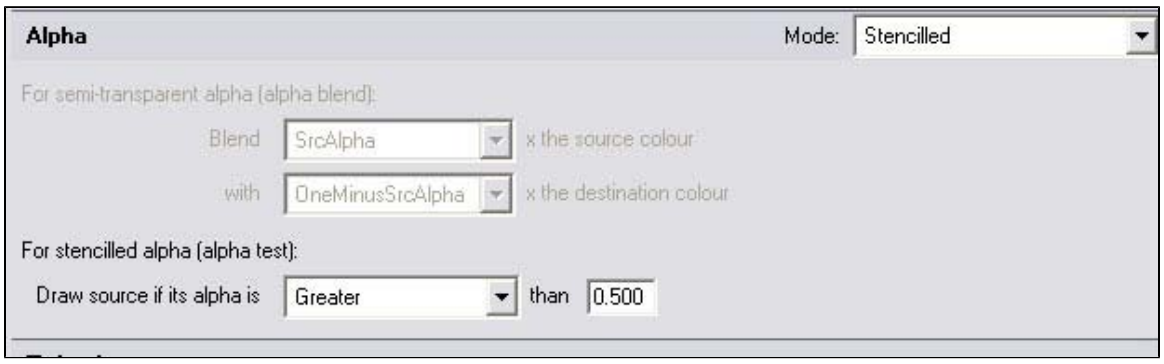
To set up the shader, the Atg Prelit Default material's bindings are set up in the usual way with the Color, Normal, Specular and Environment maps occupying their usual channels:



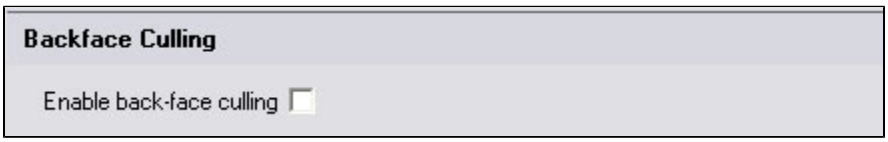
However, in the Render State dialog, we then check the Override render state? box:



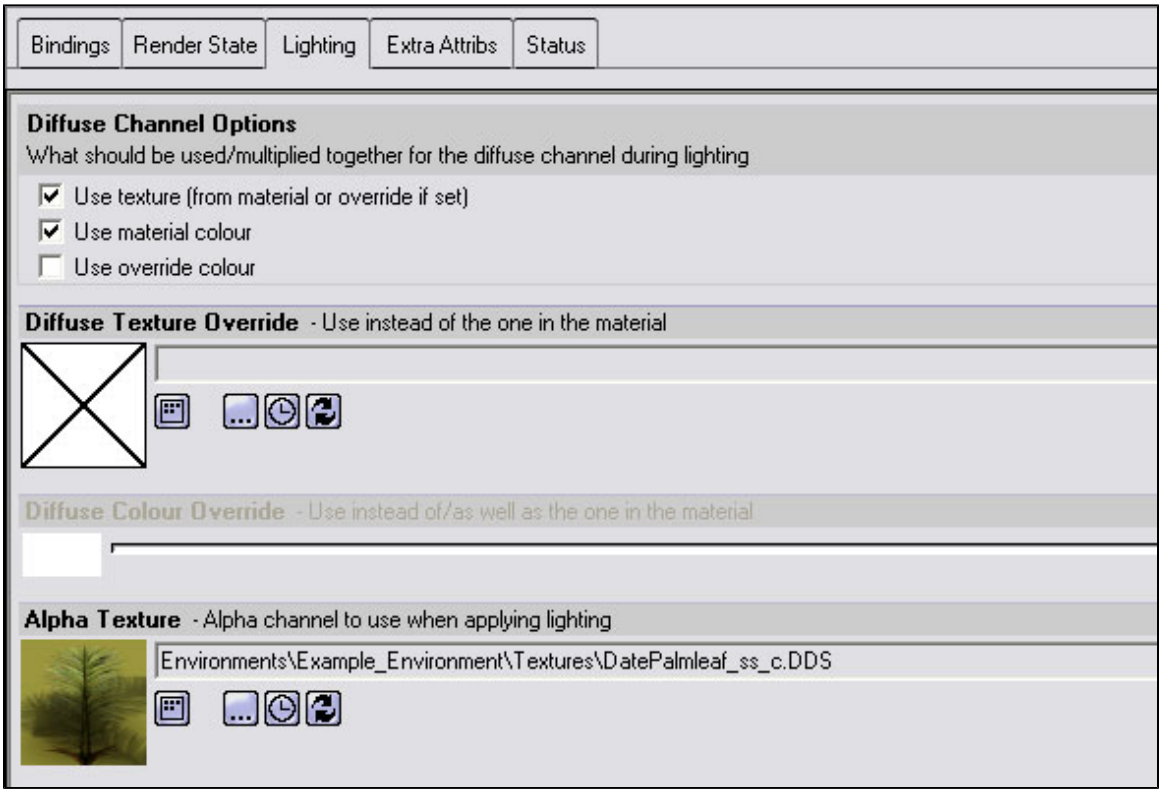
This allows the artist to set the Alpha Mode to Stencilled. The alpha can be filtered if necessary to refine the appearance by tweaking the Render State control settings. For the palm leaves the values below were used, but quite often the defaults are fine:



As it is a requirement, the foliage geometry needs to be rendered double-sided. This can be set by unchecking the Enable back-face culling box:

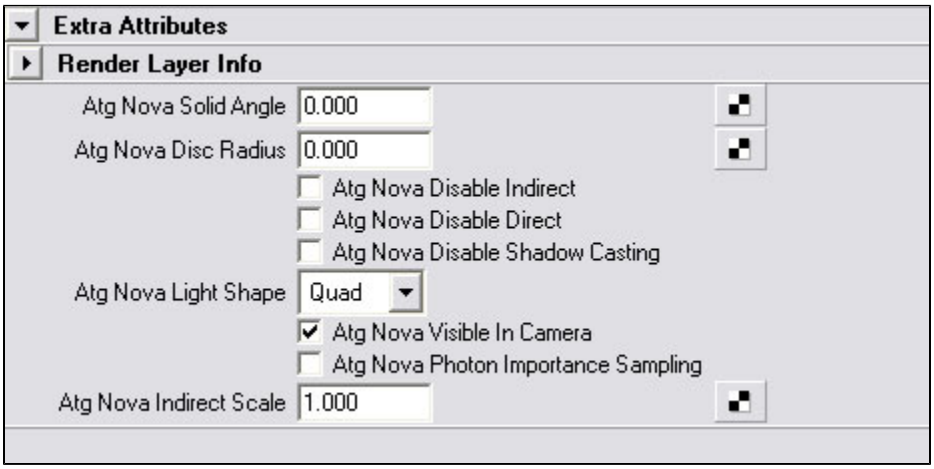


Finally, under the Lighting tab we load the Alpha Texture channel with the color texture. It will be the embedded image alpha that will be used by the render engine to generate the shadow casting of the palm fronds:



Lighting Setup

Next a couple of tweaks to the parameters in the sunlight's attributes are required. With the sunShape node selected, open the Extra Attributes panel. Here you will find specific ATG Nova parameters:

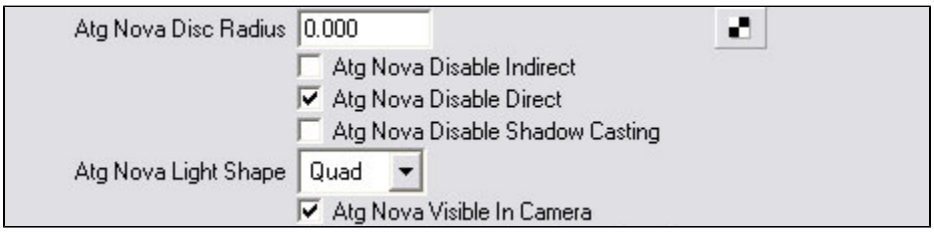


If they are not there, the artist will need to run the `atgAddNovaLightAttributes.mel` script, which can be found in `<HDK_INSTALL_DIR>\dev\External\ATG\AtgTools\Maya\scripts\8.5` (or whatever build of Maya is being used).

The parameter we are interested in is the Atg Nova Solid Angle value. This should be set to 0.500. The higher the value the more

blurred the shadow that is cast becomes. In this case, we need ours to be nice and crisp as if in strong sunlight.

In addition, check the Atg Nova Disable Direct box. Apart from those there is nothing else in the shader that we need to set up.

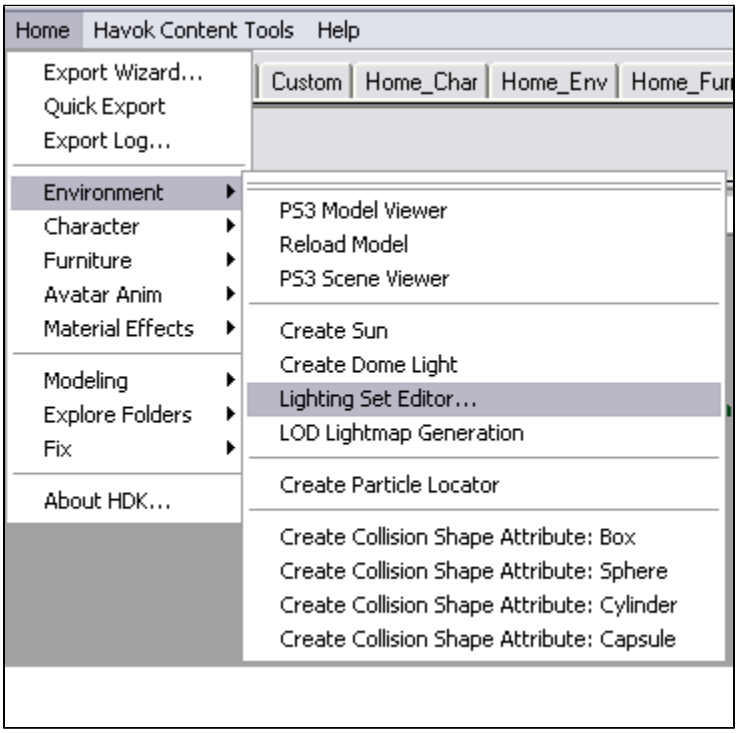


If you do not have this box checked, then direct sunlight will be added twice.

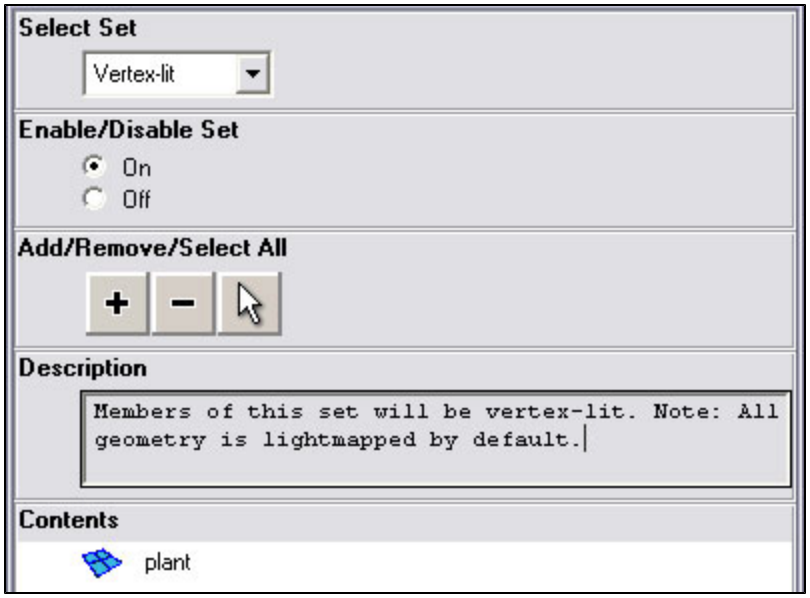
Geometry

Lightmapping the foliage geometry is unnecessary. It would wasteful of VRAM, with little or no gain visually so we will define these objects within Nova to be Vertex lit. We do this using the Home exporter's Lighting Set Editor.

From the Home menu, select Environment > Lightmap Set Editor:



Select Vertex-lit from the Select Set drop-down list and make sure that the set is enabled. Select the objects to be vertex lit in the Outliner (in this case the plant mesh) and use the + button to add it to the set:

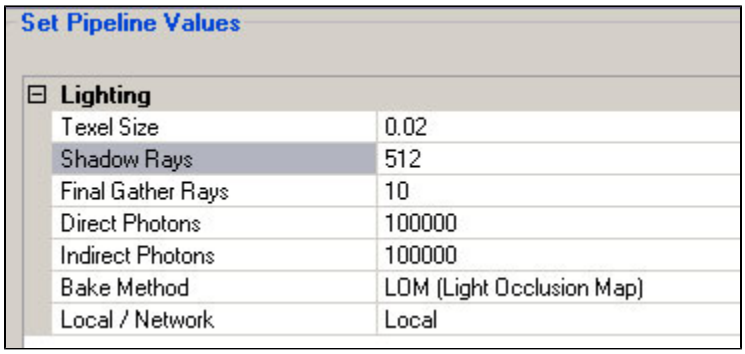


Now when this object is exported it will be excluded from the lightmappping process instead being assigned vertex color values based on the Nova's lighting calculations for the scene.

Export

The main issue to consider when exporting is lightmap resolution. We need enough detail in the lightmap for the shadows to look convincing, while not using any more texture memory than we need to. In the exporter the main factor (although not the only one) that governs lightmap resolution is the Texel Size value. This can be set globally for all objects in the scene in the Export Wizard using the Export Geometry and Static Lighting (Lightmaps) options. In this case we have set the Texel Size to 0.02. As these values are expressed in meters this gives us one Texel for every two centimeters, enough to provide the detail we want in this example.

The other factor that needs to be borne in mind is rendering times. The greater the detail the longer the lightmaps will take to process. With this in mind, and because of the relatively simple nature of the lighting rig in this case, we have lowered the Final Gather Rays value to reflect this:



Rendering

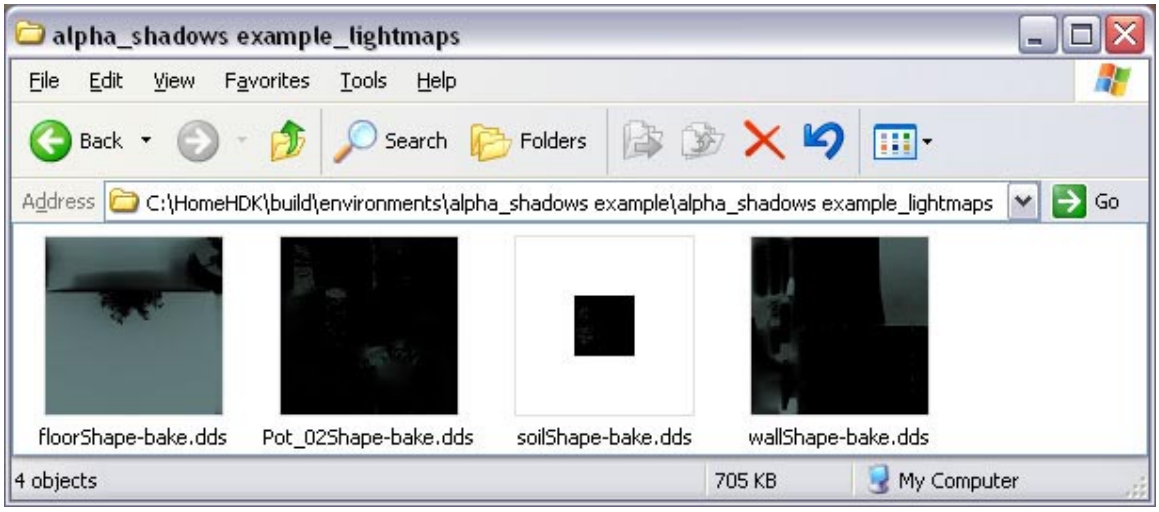
After the scene is exported we can view the results on the PlayStation®3 in the usual way to display the detail rendered into the lightmaps where the sunlight passes through the palm leaves to cast shadows on the surfaces below:



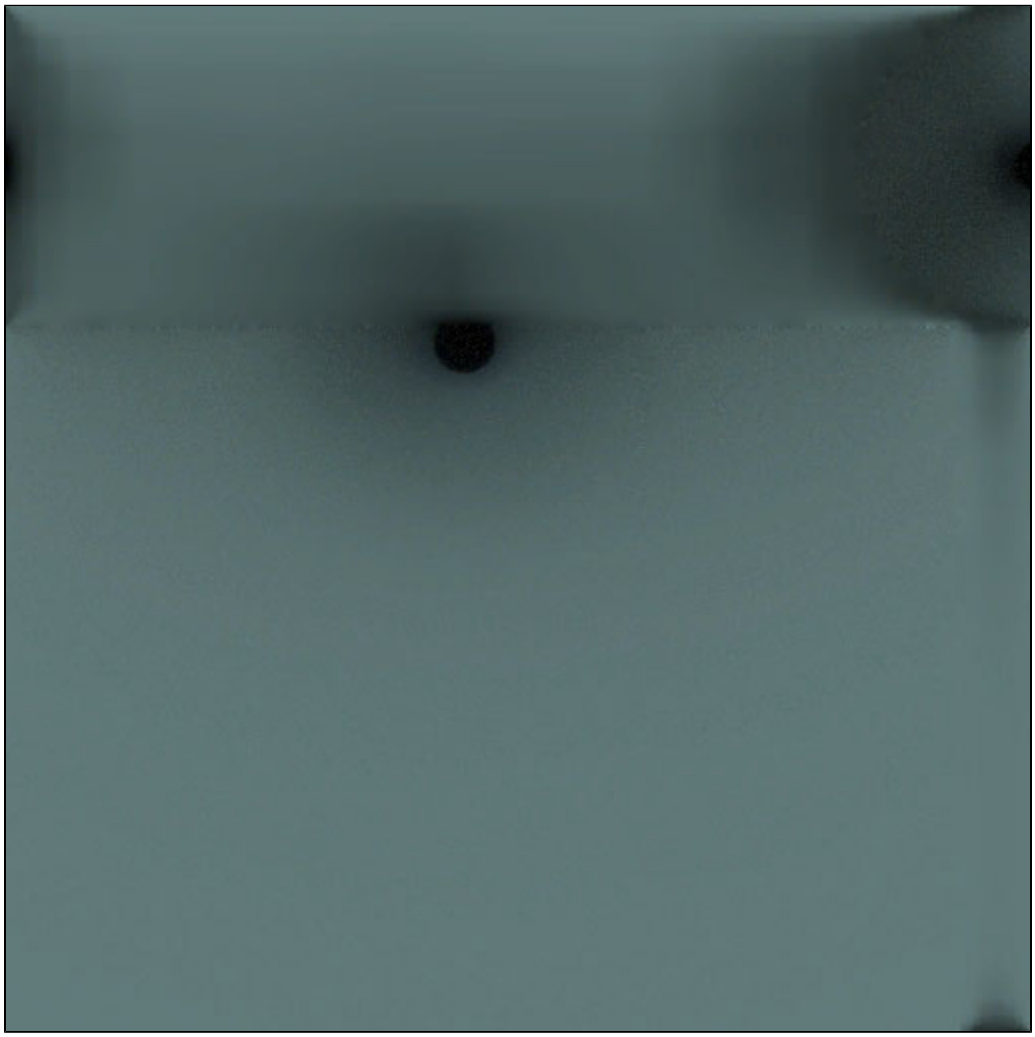
Using the Home Explorer is a quick way to navigate to the Build folder for the scene:



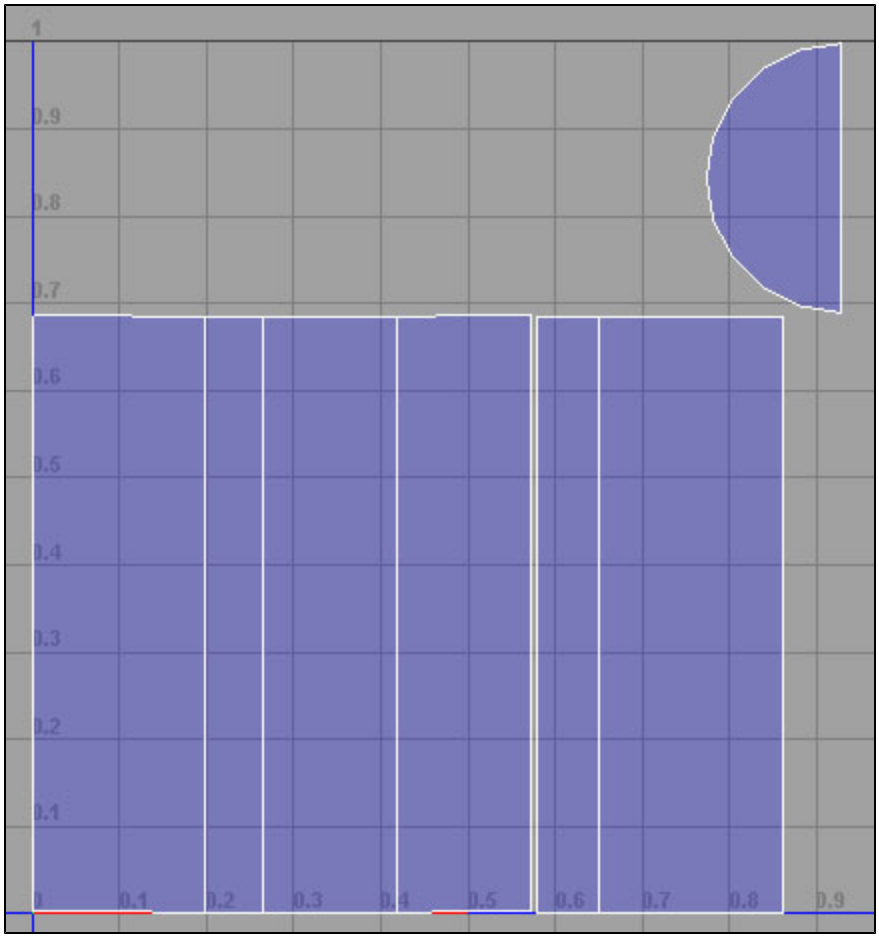
Here can be found the final render of the lightmaps that have been created:



If we take the floorShape-bake.dds and open it up in Photoshop we can examine how the mapping works:



At first this image does not make a lot of sense. It looks nothing like what can be seen rendered on the console. Until that is, we take into account the layout of the UVs for that geometry:



Now it begins to make a little more sense. What we see is the lightmapped color for that geometry as determined by the layout of the UVs. The blue tinge and to a large extent the occlusion is the result of the dome light that is in the scene along with the sunlight. But there is no sign of the detailed shadows.

That is because the shadow information is stored in the alpha channel of the texture:



Viewing the Alpha channel in Photoshop reveals that the shadows (dark maps) are embedded within the image alpha. The lightmaps are using the alpha channel as a sort of mask. The dynamic sun light illuminates everything by default, but the alpha channels of the lightmap that was generated during export controls where the light is allowed to render.

Environment Textures

This section outlines best practice when working with environment textures. For information on the file types, settings and other options for textures, see Maya Textures.

Seamless Tiling Textures

Use seamless tiling textures for large surfaces, such as walls and floors, as this allows you to scale up the UVs and get a high texel resolution. The size of the texture to use depends on how big the surface is, and how much of it will be seen at any given time on screen. How much is visible is the primary concern.

For example, if you have a large marble floor that stretches throughout an entire scene, but is masked by a lot of other objects, boundaries, overlays and height splits, you might be able to use a smaller texture (such as 512 x 512, rather than 1024 x 1024), because the visual repetition is not particularly noticeable. Textures of 1024 x 1024 are usually enough even for areas that fill large expanses of the screen.

> Textures can be any combination of pixels, provided that they are to the power of 2. For example, 1024 x 64.

Seamless tiling textures can lead to visual repetition, so use alpha'd overlays where necessary. Also, use edging (where a long thin texture of 1024 x 64 might be used) to help make boundaries between large expanses of tiling textures more appealing. See the SCEE Home Square's floor for a good example of this technique.

Seamless tiling textures also helps to keep the UV basis of a surface consistent. Home client uses the UV basis (orientation) to generate 'fake' specular highlights that help to give the surface a more 3D appearance. If you have a flat floor made up of two

faces (divided down the middle) and you create a texture to fit one side, then mirror it on the other (e.g. if you have a symmetrical logo on the floor) - the specular highlight will not follow seamlessly over the divide because the UV basis has been flipped. It is better to tile the entire floor with a single tiling texture, then add the logo as an overlay.

Unique Textures

'Unique' textures are textures that are drawn directly according to UV layout. Only create unique textures for small objects. You would create a computer terminal, for example, using textures in the traditional manner: build the geometry, lay out the UVs, save out the UV layout texture, and then paint over in Photoshop. The incoherence in the 'fake' specular highlights will not be noticeable on an object like this.

Shaders

The default Shader provides Environment Map settings, plus control of Transparency, Reflection, Glow, Normal Map and Specular. You can achieve many additional effects using ATG Materials. The ATG Material is a custom Home-specific shader in Maya. ATG Materials are added to the standard Maya setup when you install the HDK.

The PS Home custom ATG Shaders are:

- default
- emissive
- glass
- hair
- prelit_default
- prelit_glass
- skin
- water

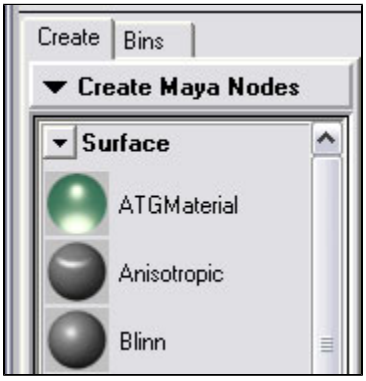
For a brief description of these material types and their behaviors, see Home ATG Materials.

Shader Setup

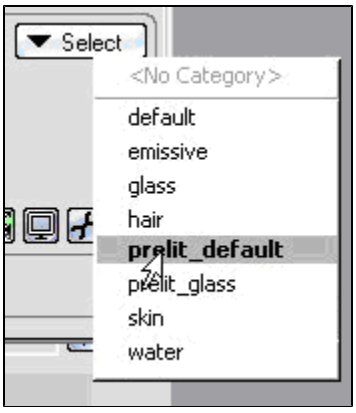
This section shows the basic steps of shader setup using the most important material, prelit_default.

To create a custom Home Shader:

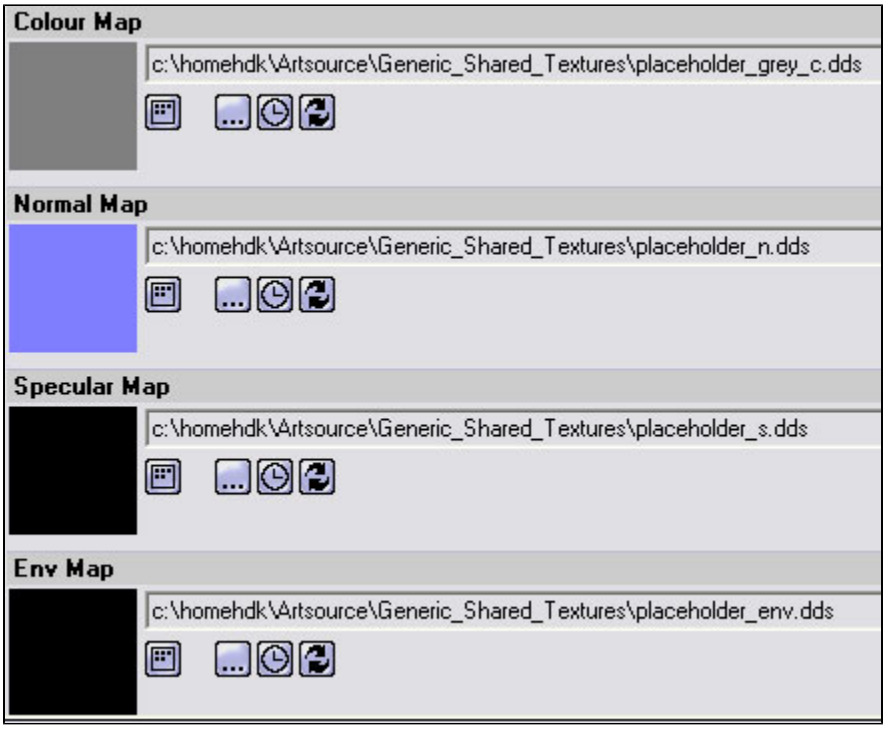
1. In the Hypershade in the Create Maya Surface Nodes list, click ATGMaterial:



2. Open the Attribute Editor, and select the prelit_default material type from the Select drop-down list:



The ATG Shader is automatically configured with four mapping channels, as shown here:



3. Each Map has a default placeholder texture. Replace these as required with your own texture maps. Leave the placeholders in any unused Map. As you do so, the Maya viewports display the maps on your geometry in the usual way.

You can set the normal map and specular map bindings for ATG materials to a filename that includes the word 'placeholder'. This setting turns off the material effect (normal mapping or specularity) for that material. The result is exactly the same, visually, as if you had assigned an identity map: all-blue for normal maps, or all-black for specular maps.

Note that:

- The placeholder textures provided for PS Home are identity textures. This is a rendering optimization, and causes the Home client to render the materials with disabled material effects faster.
- If you create a non-identity texture with the word 'placeholder' in it, the effect of your texture may be ignored.

Blend Shaders

Blend shaders have many uses, the most common being for terrain or any large areas of tiling that need to be broken up. It is also useful for adding dirt or wear-and-tear to objects.

Blend shaders use more memory than default or prelit default shaders. To help offset this cost, use them on larger areas where the creation of multiple default shaders with bespoke textures would be more expensive. The main benefit of this shader type is its ability to tile simple textures while disguising the tiling effect.

To create a blend shader in Maya:

1. Create two sets of textures (two Color, two Normal and two Specular). See Creating Textures for the Blend Shader.
2. Create a new blend or prelit_blend shader and set the maps in the appropriate slot. See Setting up the Blend Shader.
3. Assign the shader to the geometry.
4. Assign a color set to the shader. See Assigning a Color Set to a Shader.
5. Paint the vertex color information onto the mesh. See Painting Vertex Color onto the Model.

Creating Textures for the Blend Shader

Create two sets of textures (two Color, two Normal and two Specular). These maps are ordered into two sets of three each containing one of the map types.

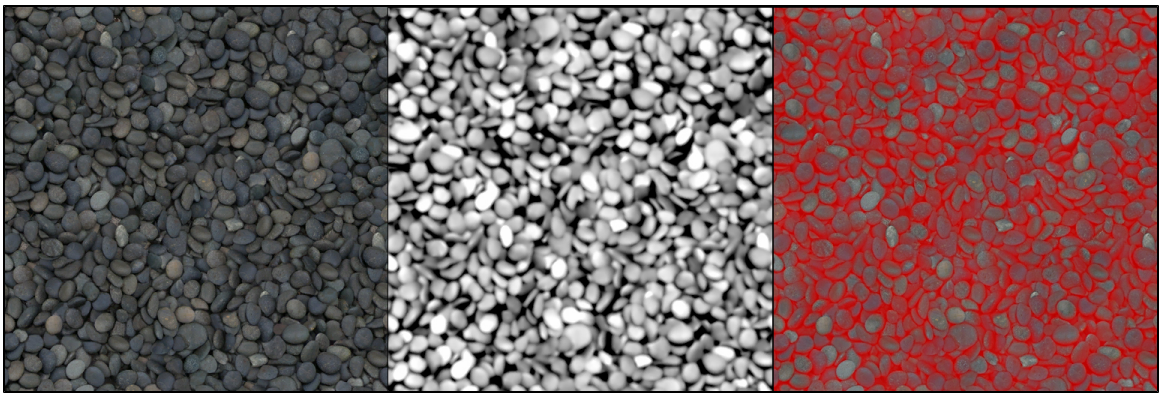
If a map is used in one of the sets it must be used in both. So for example, if a specular map is used in the first set there must also be a specular map in the second. The default maps that are created when the shader is initially selected are not valid - they are optimized on export.

Alpha channel and height map

Create the textures as normal except for the first color map. This map must have an alpha channel saved as DXT5. The alpha channel of this map is used along with the vertex color to control how and where blending will take place within the shader.

The alpha map used is usually a height map, although any map can be used depending on the way you wish to blend the textures. The main reason a height map is often used is that it begins the blend at the low points of a normal map giving the effect of building up from within cracks and crevices.

Below is an example of a color map. From left to right: the color (RGB) channel, the alpha channel (Height Map) and finally the alpha viewed as a mask.



This is the only map that needs a special set-up on its alpha channel. The other maps are created as normal.

When creating texture for the blend shader keep in mind that there are additional attributes added to the shader that allow you to adjust tiling on the second set of textures. This means that the two sets of texture maps do not have to be at the same resolution (see Painting Vertex Color Information).

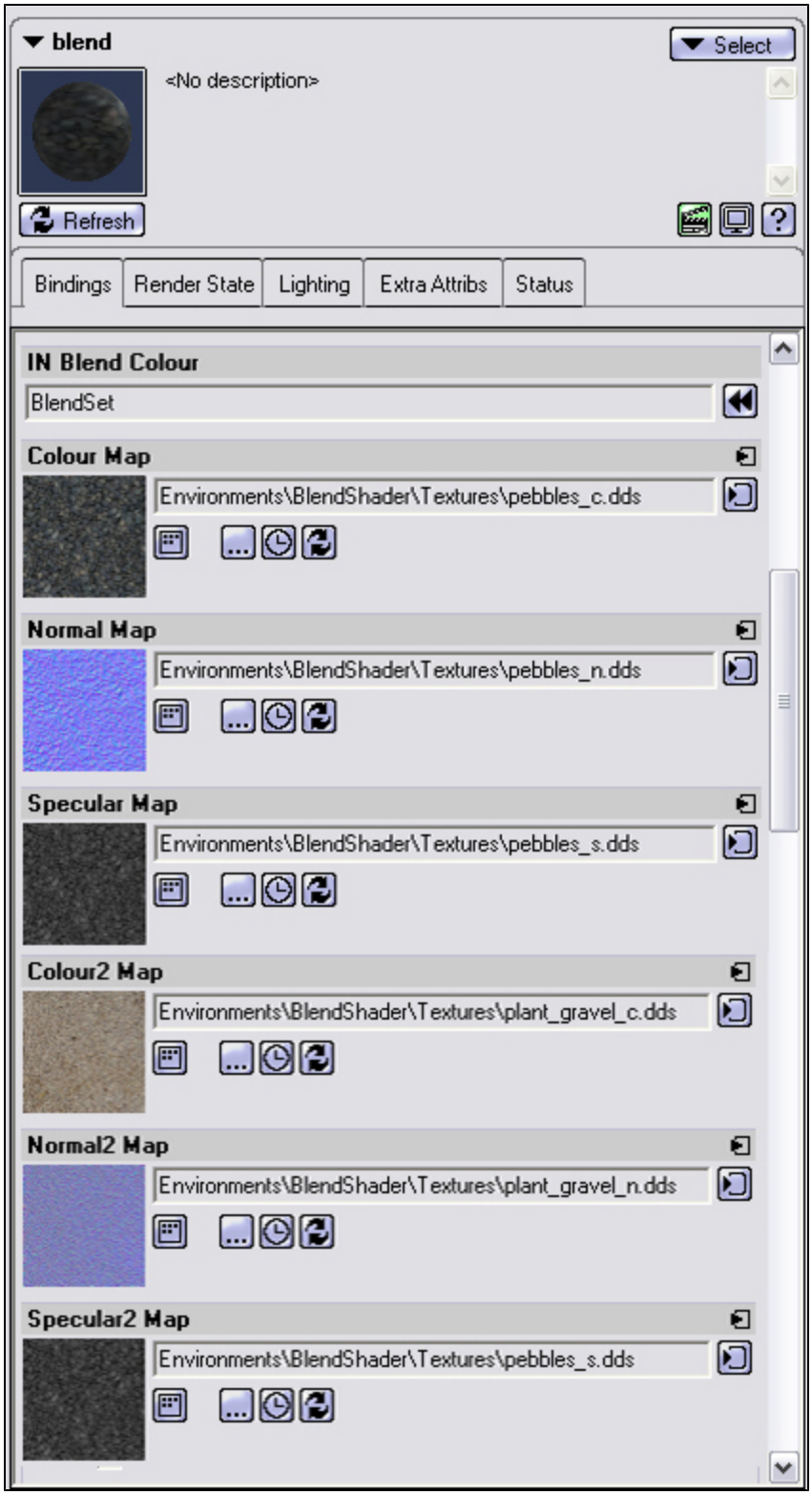
Setting up the Blend Shader

There are two types of blend shader in Maya:

- blend: A shader for blending together two textures using a heightmap.
- prelit_blend: Similar to a blend shader, but it is prelit with either assigned lightmaps or baked vertex lighting.

For more information on blend shaders and other material types, see Home ATG Materials.

The image below shows the blend shader setup and uses the example texture from Creating Textures for the Blend Shader:



You can see that there are six maps in two sets (the second set has the number 2 appended after the map type within the name).

> The first color map has the texture with the alpha height map assigned to it.

In this example, the two sets of textures depict pebbles in the first set and sand in the second. The sand texture is of a lower resolution than the pebbles.

The resolution of the first color map determines the resolution of its alpha channel and so also the height map used for blending.

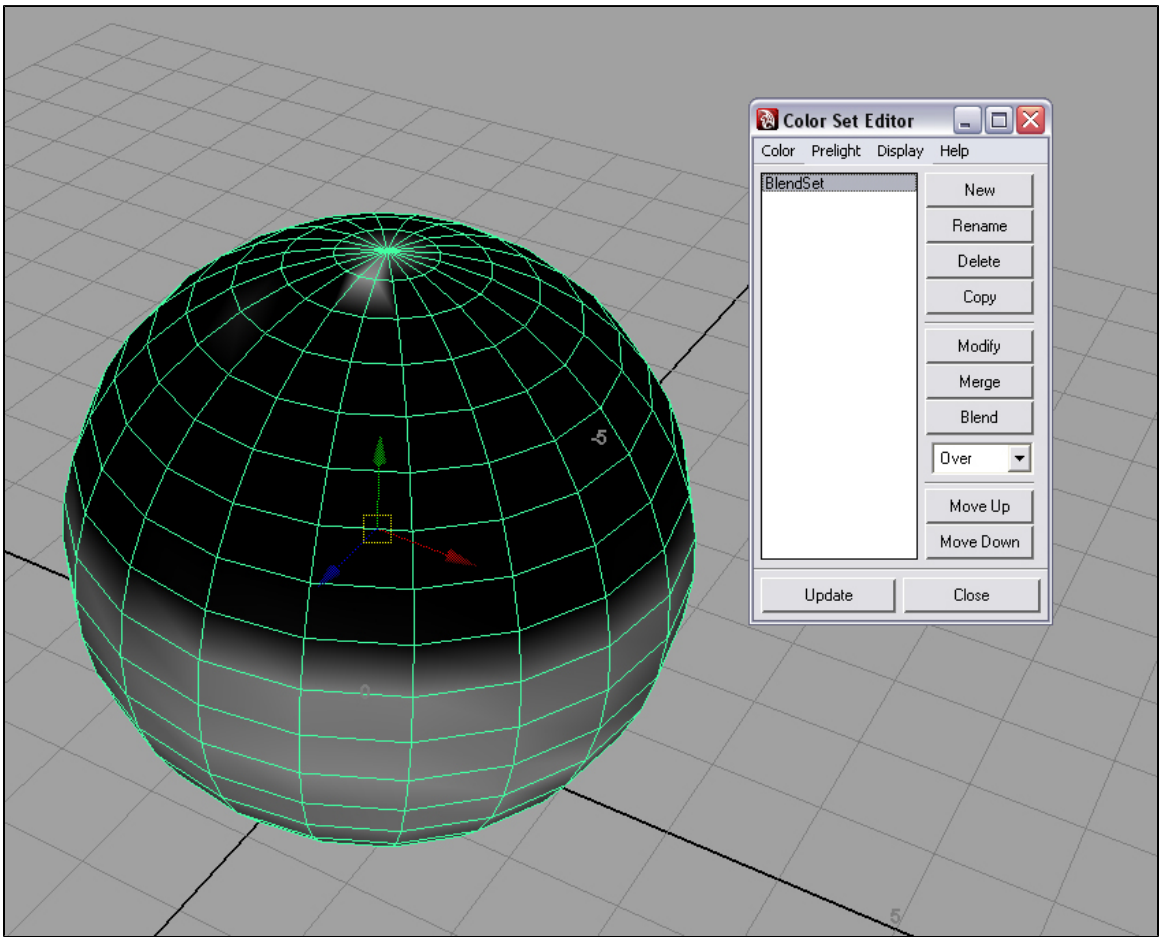
Assigning a Color Set to a Shader

After you have assigned all the maps in Maya, assign a color set called BlendSet to the shader.

To assign a color set to the shader:

1. Select the mesh in Maya.
2. Select Color > Color Set Editor under the polygon menu set.

   The Color Set Editor allows you to add a color set to the mesh:



   In the example above, a new color set called BlendSet has been added.

We recommend the color set is named BlendSet.

3. Select the BlendSet color set from the drop-down list of all the available color sets on the selected mesh (displayed at the top of the blend shader setup options):



Please note:

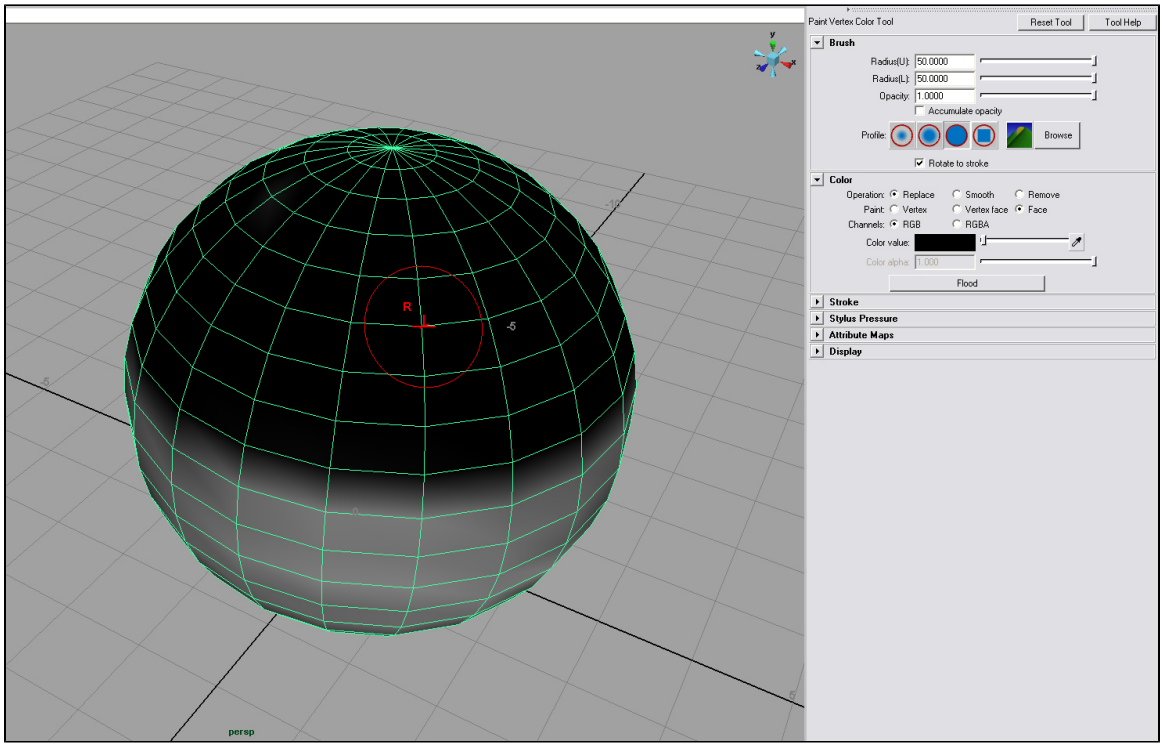
- Only one color set can be selected for each blend shader, however it is possible to assign this shader to more than one mesh.
- If the shader is used on multiple meshes, each mesh must have a color set with the same name for the shader to work.

This allows you to paint vertex color onto the model using the Paint Vertex Color Tool. See Painting Vertex Color onto the Model.

Painting Vertex Color onto the Model

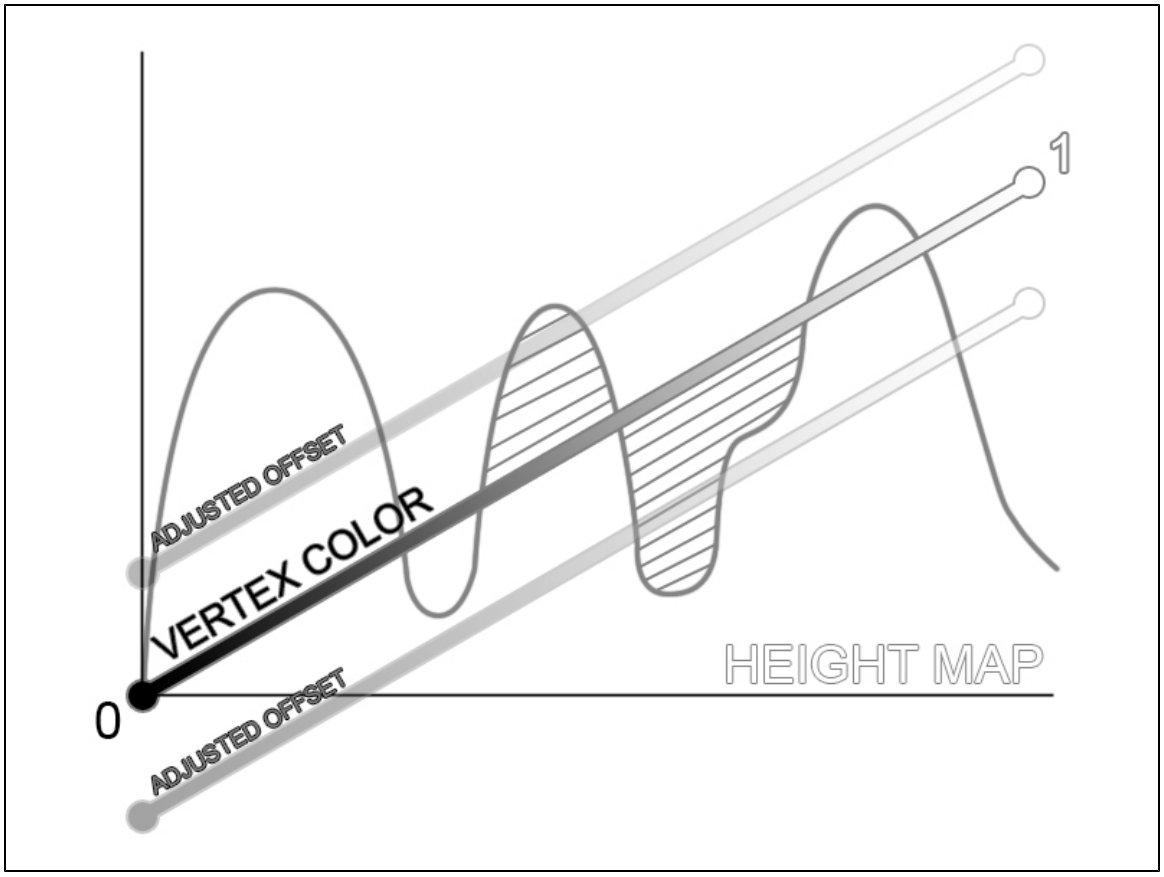
After you have added a color set to the mesh, you can paint vertex color information into the vertices using the Paint Vertex Color Tool (select Color > Paint Vertex Color Tool in Maya).

The image below shows an example of a sphere that has vertex color applied to it:

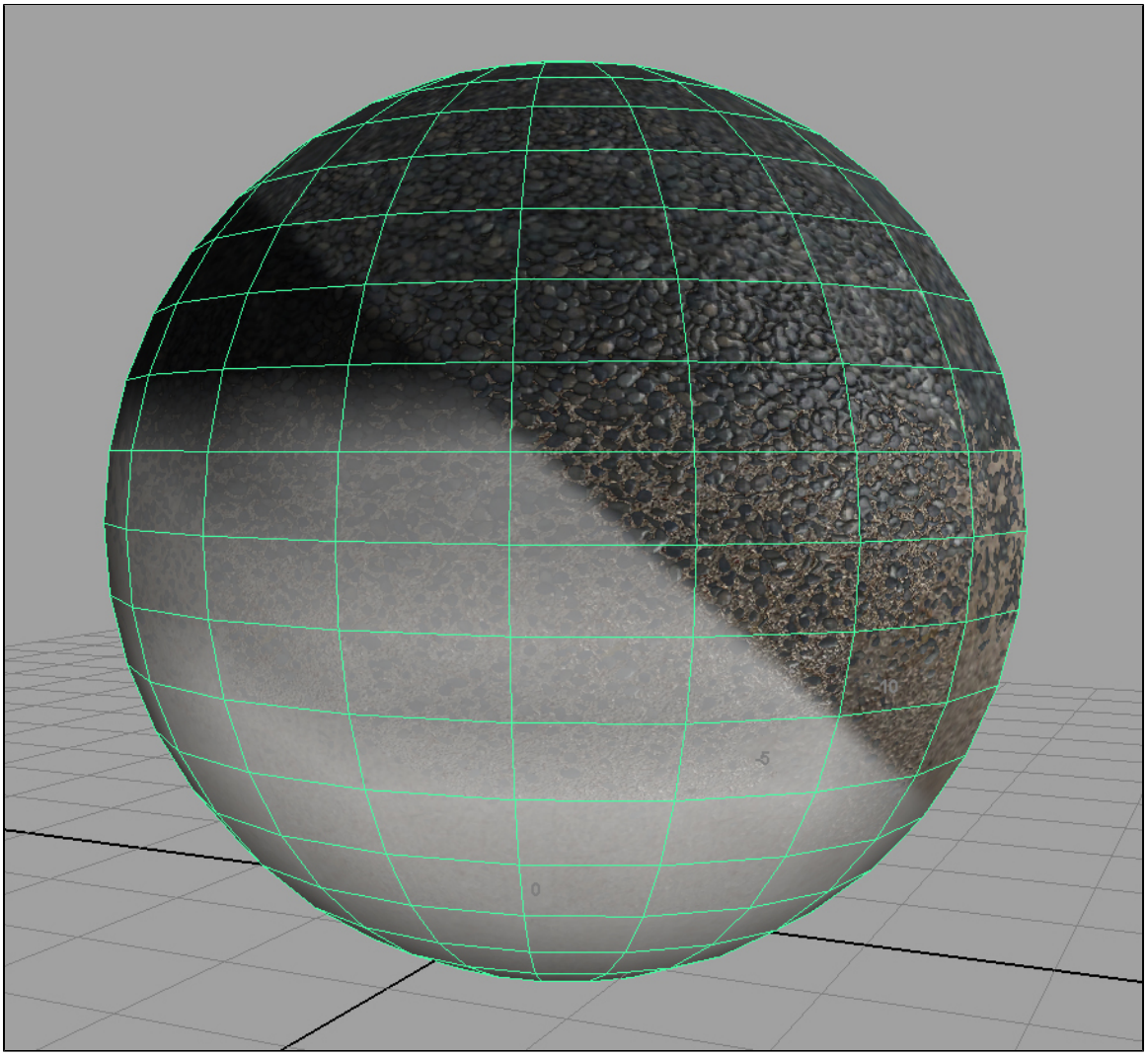


When painting for the blend shader, use greyscale values between 0 - 1 only.

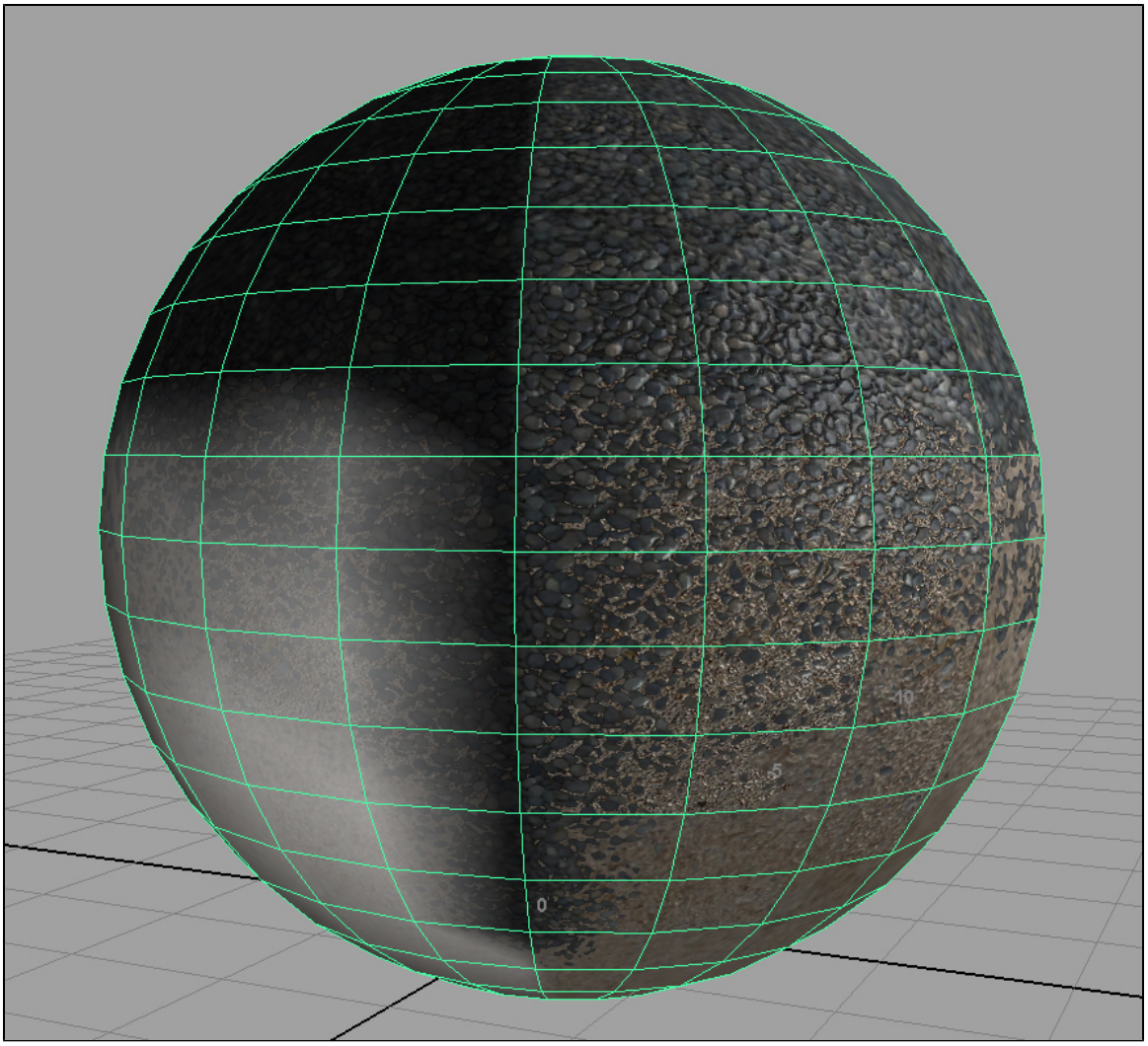
The graph below shows the how blending is handled between the vertex color and the height map. The vertex color applied to the vertices is represented in a simple 0 - 1 line, the wave represents an example height map in the alpha channel. The texture is blended where the vertex color and the color of the height map meet.



Here is an example using the shader and vertex painting from above:



The sphere has a uniform gradient going from black at the top to white at the bottom. This creates a smooth blend from pebbles to sand. Editing the vertex color changes areas where the blend takes place as well as the intensity of the blend.



The above image shows where black vertex color has been added to the sphere in a line down the center. The area of black now has dense pebbles in a line following the vertex color.

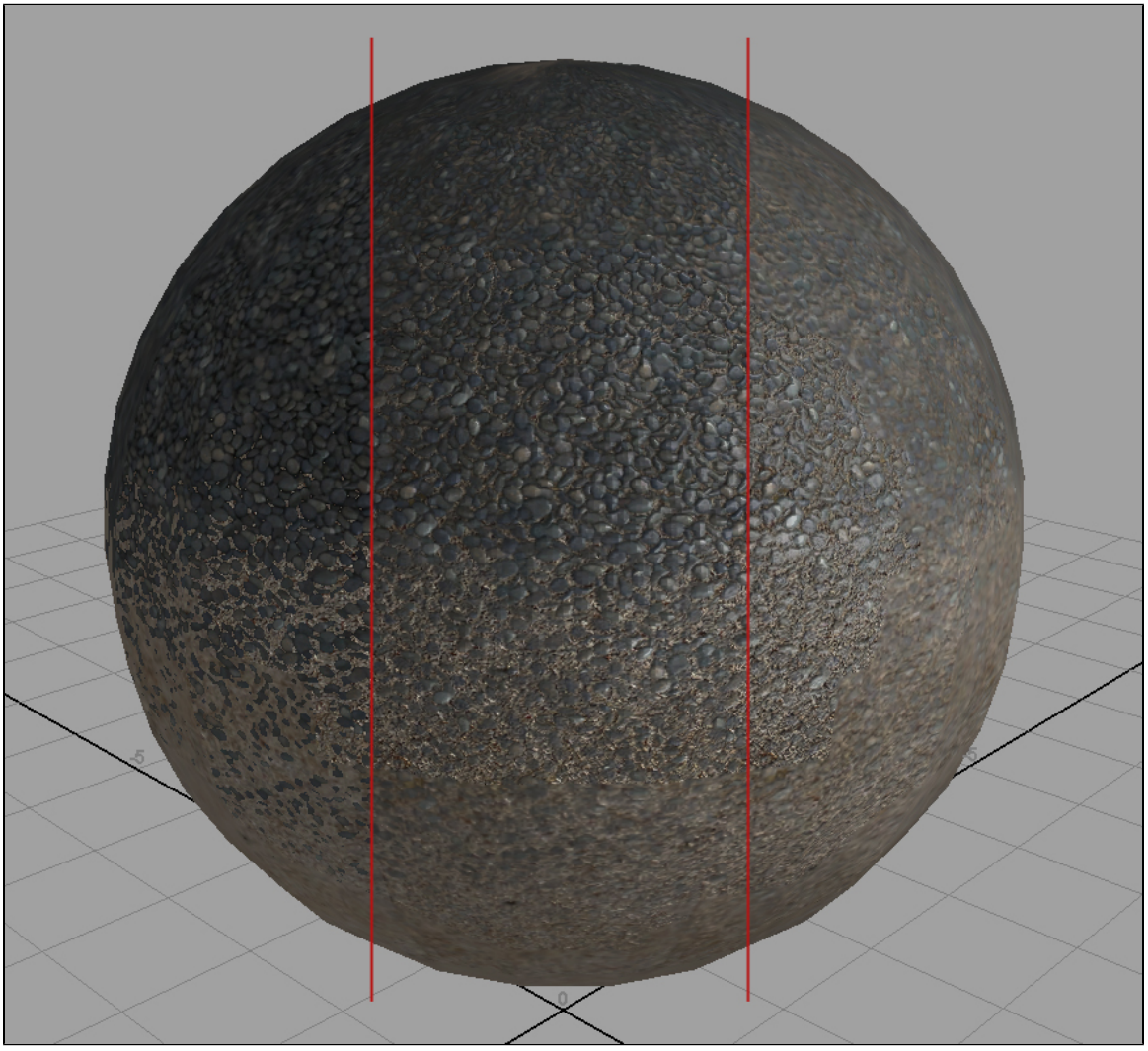
Blend Shader Settings

Blend Offset and Blend Ramp

The two values under Tweaks are used to affect how the materials are blended:

- blendOffset: Offsets the vertex color values - moving them all towards either black or white. This has the effect of shifting which areas are blending and by how much. Pushing the offset all the way in either direction causes the material to display only one of the textures.
- blend ramp: Effects the smoothness of the blend in areas where blending is taking place. The image below shows 3 settings for the blend ramp. The left side of the globe has the blend ramp set to 0. This creates a harsh transition with no

smoothing. The middle is set to 0.5 and the right side to 1. This section displays very smooth blending.

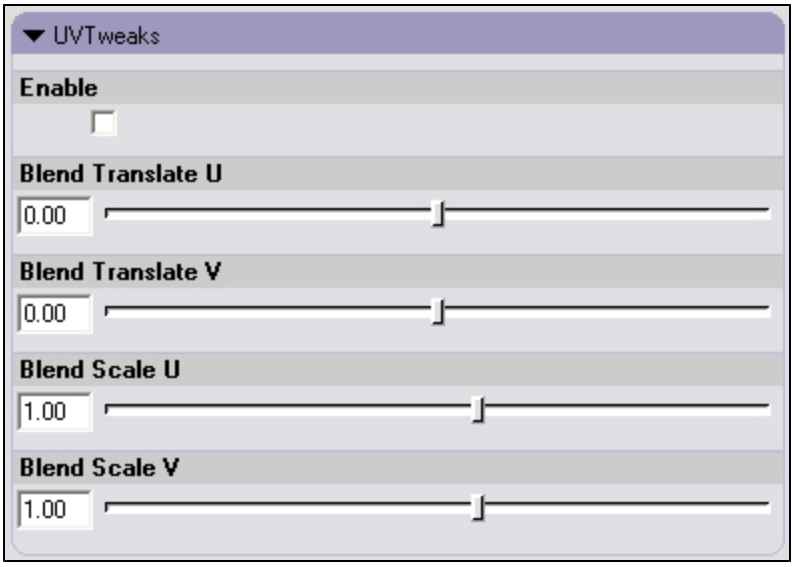


It may be necessary to edit both these setting to achieve certain looks as they will affect each other to a certain extent. The height map quality and color range also affects the final look.

Both of these values are editable using the Material Lua API.

UV Tweaks

The UVTweaks panel allows the adjustment of the UVs for the second set of textures. The basic settings here can change the scale in U and V to create greater tiling and change the translation in U and V. These values work as an offset or multiplier based on the current UVs:



The main use for this is to allow the second set of textures to be a lower resolution than the main set of texture maps. In the example above, the sand texture could be made to tile several times by increasing the scale allowing the texture to be of a lesser resolution at source but still appear the same after the tiling has taken place.

Material Lua API

Use the Material.SetVector function to set the blend offset and blend ramp values. The vector to be set is blendParams this is passed a vector4 with the first two values the blend offset and blend ramp. The other two components of the vector4 are not used.

Below is an example of the Lua used to set the blendParams:

```
Material.SetVector(<MATERIAL>, "blendParams", Vector4.Create(<BlendOffset>,< BlendRamp>0,0,0))
```

It is also possible to set the texture maps in the same way as any other material. To define the first set of three textures, the standard API commands will work, the second set has specific names. Below is an example of this.

```
<MATERIAL>.SetTexture(<MATERIAL>, "colour2Map", <TextureResource>)
<MATERIAL>.SetTexture(<MATERIAL>, "specular2Map", <TextureResource)
<MATERIAL>.SetTexture(<MATERIAL>, "normal2Map", <TextureResource)
```

Where colour2Map, specular2Map, normal2Map are the defined names for the second set of colour, specular and normal maps for blend shaders.

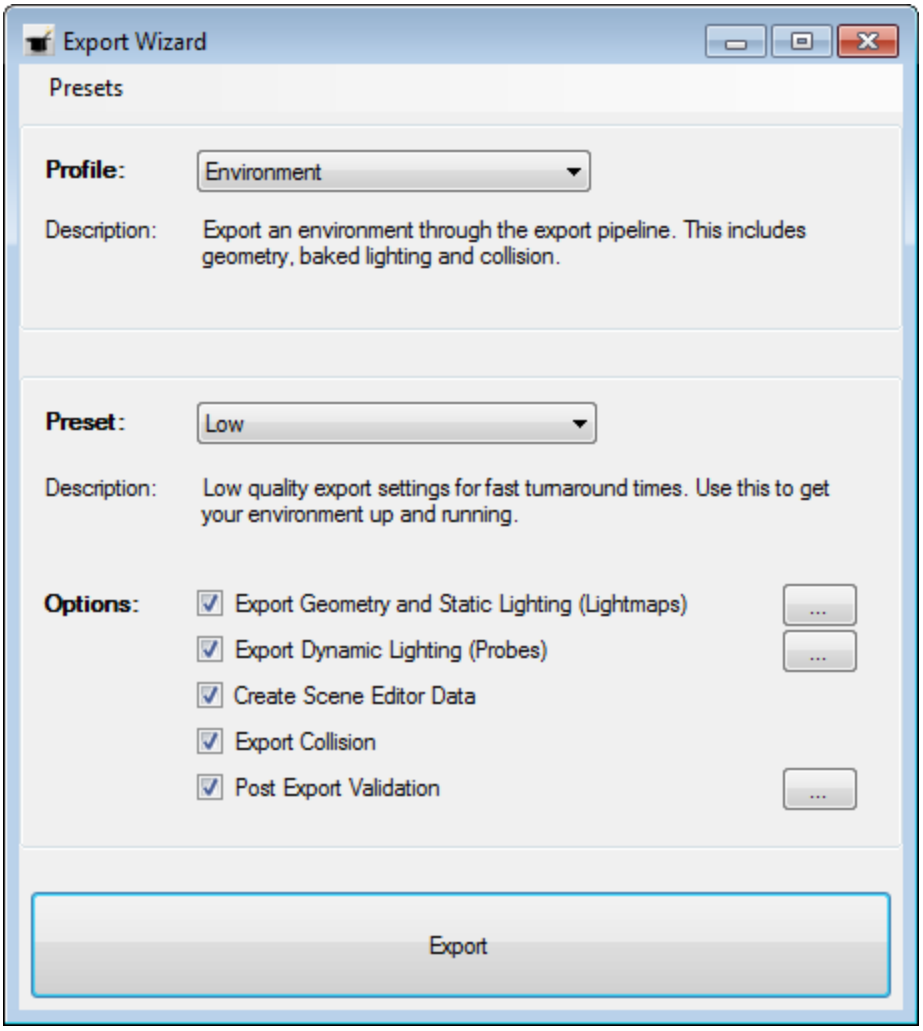
Validating and Exporting Environments

To export an environment:

1. Launch the Export Wizard from the Home drop-down menu or by clicking  .

> The Quick Export option (also available from the Home Tool shelf or the Home menu) exports the environment using the last settings used.

The Export Wizard dialog is displayed:



2. Select Environment from the Profile drop-down list.

   The project data is exported to an intermediate folder in <HDK_INSTALL_DIR>\Intermediate which is automatically created with the same name as the Maya file when the first export takes place. This contains a number of files required by PS Home to build the final environment and an export log file that contains useful data about the export.

3. Select the export quality from the Preset drop-down list. This gives the basic export quality options:

   - Low: This is the default option. Use this when developing and debugging your environment artwork. It configures the exporter options to give the fastest export time.
   - Medium: You can use this when developing your environments lighting. It improves the definition of the lightmapping and probe lighting; however, it has an impact on the export times. The impact depends on the complexity of your scene.
   - High: Use this when you are getting close to the final output for the lighting solution. This has a substantial impact on export times but depends on the complexity of the geometry and the lighting and the specification of your hardware. The render quality settings can be set higher still, although you would only want to employ these to get high quality results.

4. For custom presets, check or uncheck the export Options boxes, as required.

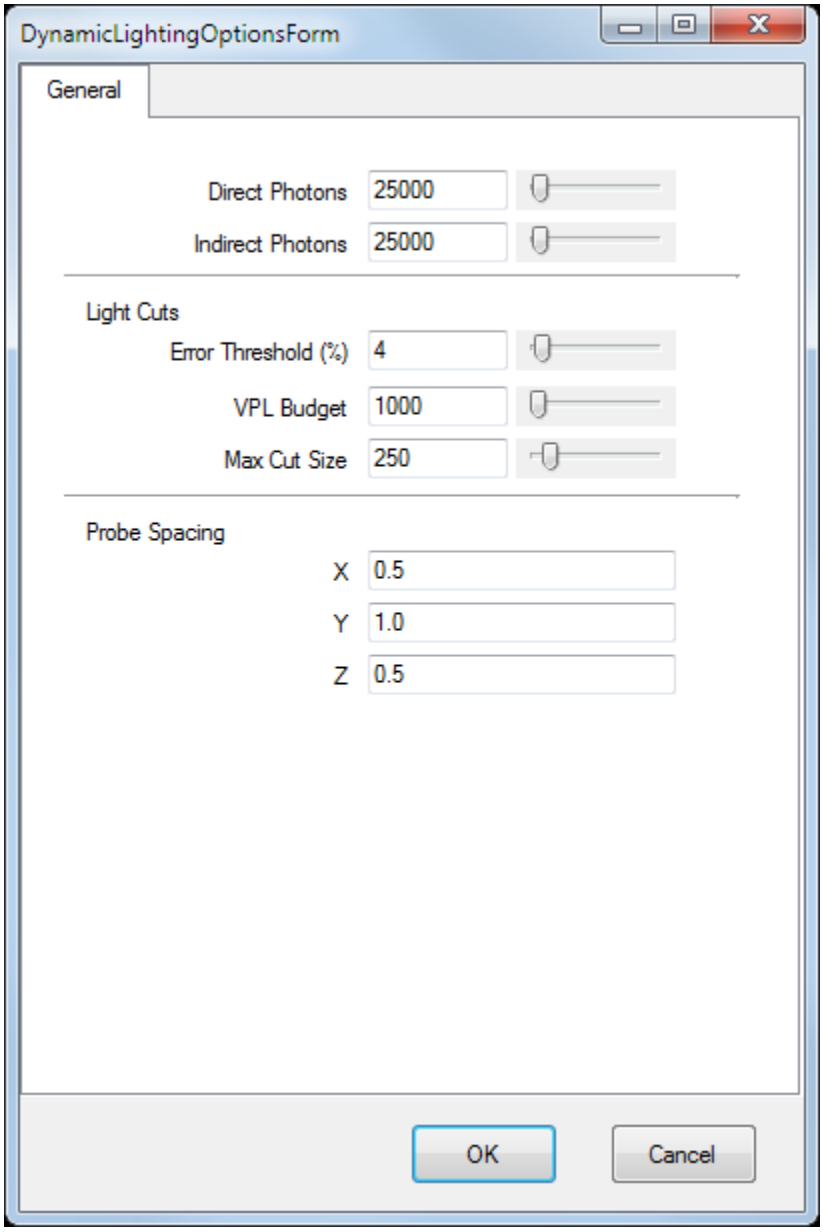
   All the boxes are checked by default. However, you can choose to export only certain parts of an environment, like collision geometry by unchecking the boxes next to the options you don't require. This saves the exporter having to re-export the environment geometry or re-calculate the Lightmaps and probes.

   You can also change the export parameters by clicking the  buttons for further options.

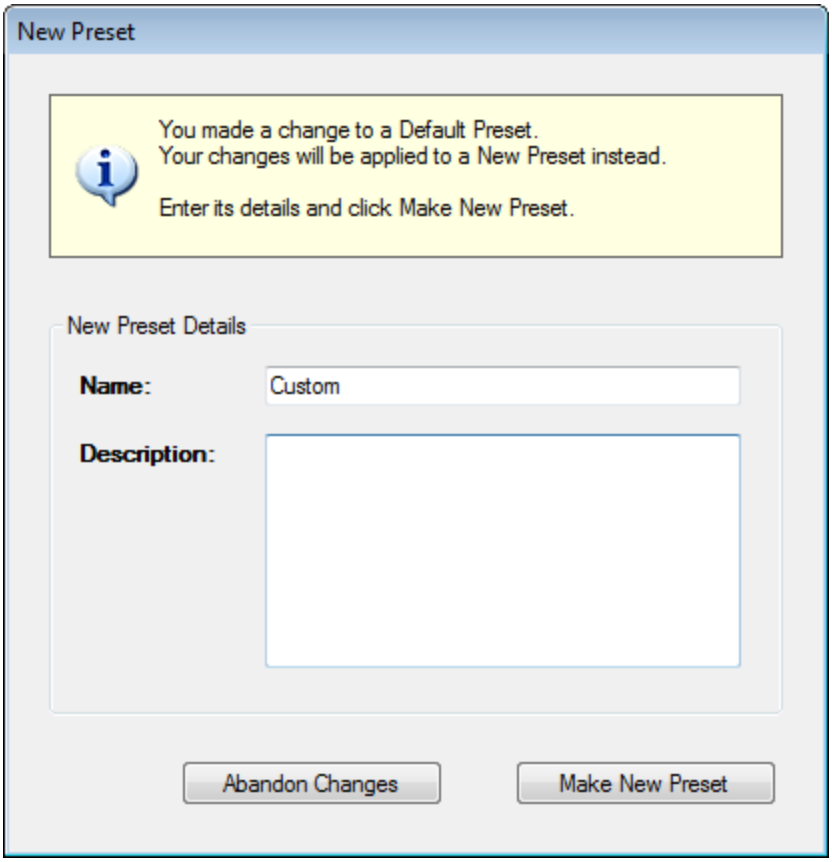
   For example, to change the probe density, click  next to the Export Dynamic Lighting (Probes) box:



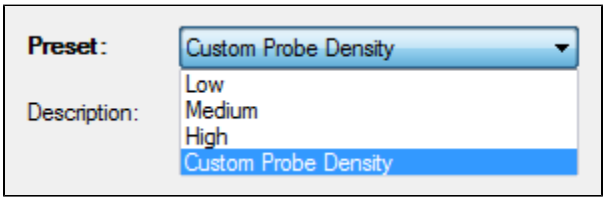
The DynamicLightingOptionsForm dialog is displayed and you can then modify the dynamic lighting parameters:



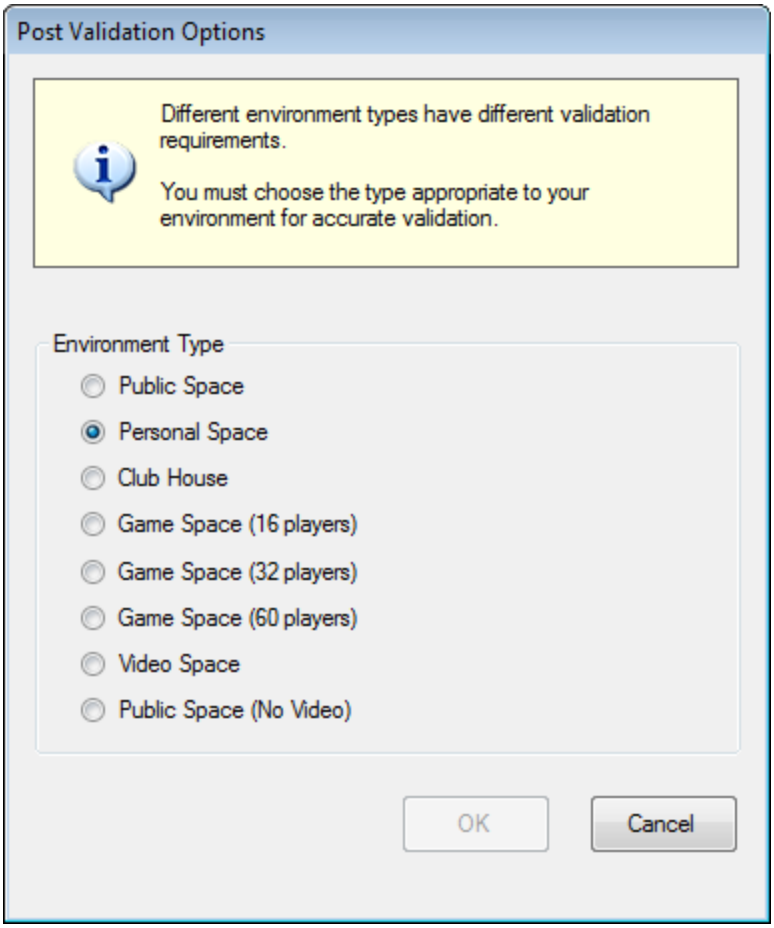
If you have, for example, changed the Probe Spacing X field from 0.5 to 0.25, the New Preset dialog is displayed so you can describe your new preset:



Any further parameters you change are then saved to this custom preset and the custom preset is then added to the Export Wizard's Preset drop-down list:



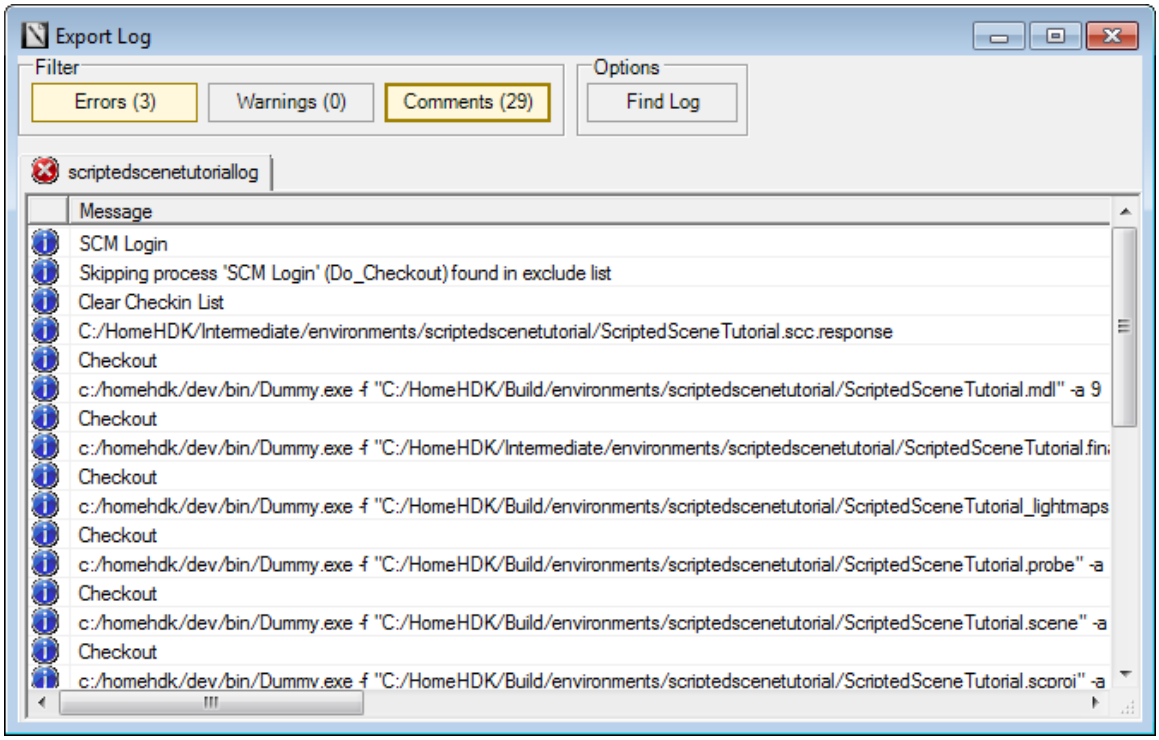
5. To select a scene type, click  beside Post Export Validation. The Post Validation Options dialog is displayed:



You can change the scene type later when packaging the scene in the Scene Editor for submission to the Content Delivery System (CDS). See PS Home Spaces for an explanation of each scene type.

If the Post Export Validation box is checked, the Export Wizard will validate your content to make sure it complies with PS Home requirements. For more information on what is validated, see HDK Tools Validations. Only uncheck this box if you are sure you won't be packaging the content for submission to PS Home (for example, if you are just testing the export to the Scene Editor or Object Editor).

6. Click Export to start the export of the data.
7. At the end of the export process, the Export Log dialog displays the results of the export and lists any errors, warnings and comments, for example:



If there are no errors, select Next to move onto the next step. If the log reports errors, you must correct them before continuing.

If your export is successful, "Export Complete" is displayed.

The export process creates a Home model file (.mdl) that contains the geometry and a matching Havok file (.hkx) that contains the collision data. A probe file (.probe) is also exported with the realtime lighting data and a scene project file (.scproj) that references all the other files to create the final Home environment.

When validation and export are successfully completed, you can edit the scene or object/component created in Maya (if necessary) and package it in the appropriate editor. Use the Scene Editor for scenes and the Object Editor for objects or components. The scene or object/component must successfully export and be packaged by the associated editor before you can submit it for quality assurance.

Imogen – PS Home Distributed Lighting Exporter

The lighting export stage is the most time-consuming part of an environment export. The Imogen patch expedites the distribution of workload (jobs) between multiple machines. You can share the export process between multiple machines by setting up the artists' machines as clients, and configuring other machines on the network to function as slaves and one as a coordinator of the job allocation.

This section outlines how to use Imogen with the HDK to distribute workload between multiple machines.

Imogen Requirements

To run Imogen, you require the correct version of the Imogen client, Java and VCRedist.

Imogen Client

Client and Coordinator

You require the latest version of the HDK.

Slave

You need to have the latest HDK installed or the Imogen patch.

If you are installing the slave on a computer that does not have the HDK installed, you need to download the Imogen patch from https://home.scedev.net/projects/hdk.

If you are installing a slave on a computer that does not have the HDK, the patch will create the path <HDK_INSTALL_DIR>/dev/External/ (for example, C:/HomeHDK/dev/External/).

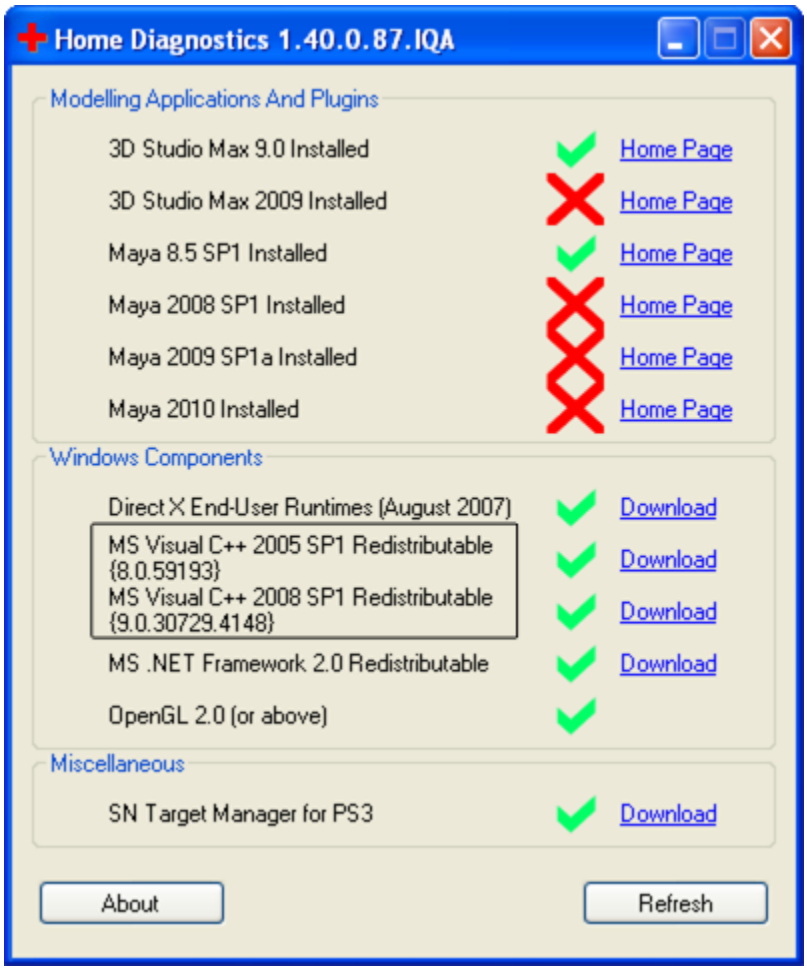
Java JRE 1.5/5.0 or higher (if using Imogen)

Imogen is a Java application, so make sure you have a 1.5/5.0 or later JRE installed on your system. Check whether you have the correct installation or download the latest from http://java.com.

Microsoft Visual C++ 2005 SP1 Redistributable Package (x86)

The vcredist_SP1_x86.exe and vcredist_x86.exe are applications that install the latest version of Microsoft Visual C++ runtime files and operating system components.

Machines that already have the HDK installed (such as Client or Coordinator machines) can use the HDK Diagnostic Tool to check whether the correct version of the VCRedist is installed, or click the Download link to get it.



For machines that do not have the HDK installed (for instance, Slave machines), ensure that the VCRedist (specifically the vcredist_SP1_x86.exe and vcredist_x86.exe) are downloaded and installed from:

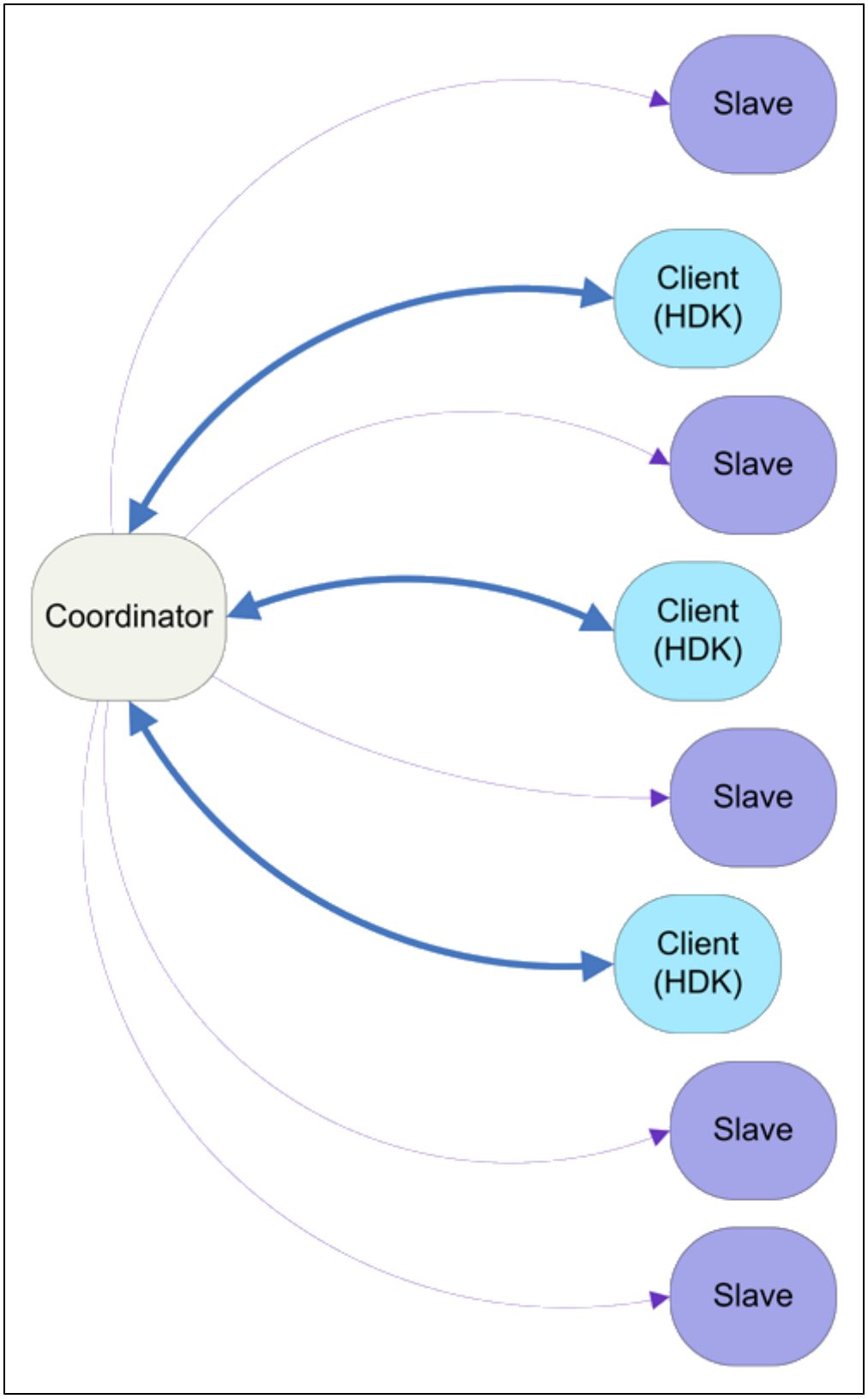
http://www.microsoft.com/downloads/details.aspx?FamilyID=200b2fd9-ae1a-4a14-984d-389c36f85647&DisplayLang=en

http://www.microsoft.com/downloads/details.aspx?familyid=9b2da534-3e03-4391-8a4d-074b9f2bc1bf&displaylang=en

Imogen Overview

Each machine in the distributed lighting effort will be one of three types:

- Those with the HDK already installed can function as clients.
- Further machines as slaves.
- One machine as a coordinator.



One coordinator controls the export jobs, receiving requests for distributed exporting from the clients (with HDK) and assigning these jobs to the available slaves (without HDK) and the local client. The local client, in this context, is the machine that exported the job.

When a client exports a lighting job to the coordinator, the client too works on the job (this is called the local client). The coordinator does not assign the job to other clients.

Details of each machine type are as follows:

Type	Task	Quantity	Installation Needs
Slave	The lighting export workload is distributed between slave machines	Several	Imogen vcredist_SP1_x86.exe vcredits_x86.exe
Client	Executes a lighting export request (handled by the HDK) to the Coordinator. Works on local lighting workload alongside slave machines.	One per artist	Imogen Latest HDK
Coordinator	Receives lighting export requests from Clients (handled by the HDK) and distributes the workload to a pool of slaves and connected clients	One	Imogen Latest HDK

Installing the Imogen Client

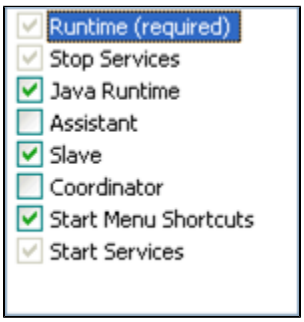
Follow these installation instructions on the machines which will act as a client:

1. Run:

   `<HDK_INSTALL_DIR>/dev/External/Imogen-Installer_xxxxxx.exe`

   > This file name is subject to change.

2. Check the Java Runtime, Slave and Start Menu Shortcuts boxes:

3. Set these Control Panel settings (for more information, see Displaying the Imogen Control Panel):

   - Builder > General > Window > true
   - Builder > Distribution > Coordinators > <Coordinator PC address>
   - Slave > General > Coordinator > <Coordinator PC address>

4. Run the patch file:

   `HomeHDK_Distributed_Light_Processing_1.60`

   > This file name generally changes with each release.

   On the Select Destination Location part, choose the location of your existing installed HDK.

Installing the Imogen Slaves

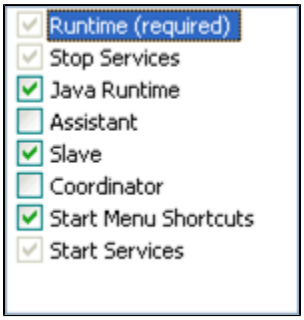
Follow these installation instructions on the machines which will act as a Slave:

1. Run

   `<HDK_INSTALL_DIR>/dev/External/Imogen-Installer_2_5_0_RC12.exe`

   > This file name is subject to change.

2. Check the Slave and Start Menu Shortcuts boxes:



3. Set these Control Panel settings (For more information, see Displaying the Imogen Control Panel):
    - Slave – Coordinator – <Coordinator PC address>
4. Run `vcredist_SP1_x86.exe` and `vcredist_x86.exe`

> The installers should complete without any notification.

The vcredist_SP1_x86.exe is an application that installs the latest version of Microsoft Visual C++ runtime files and operating system components. The files are included with Visual Studio 6.0 SP 4, or can be downloaded from:

http://download.microsoft.com/download/vc60pro/update/1/w9xnt4/en-us/vc6redistsetup_enu.exe

http://www.microsoft.com/downloads/details.aspx?familyid=9b2da534-3e03-4391-8a4d-074b9f2bc1bf&displaylang=en

For details on what this installation contains, see:

http://support.microsoft.com/kb/259403

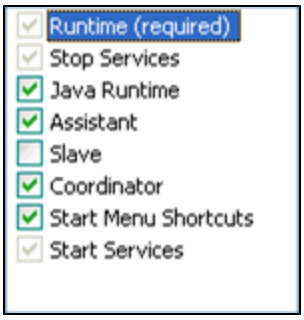
Installing the Imogen Coordinator

Follow these installation instructions on the machines which will act as a Coordinator:

1. Run `<HDK_INSTALL_DIR>/dev/External/Imogen-Installer_2_5_0_RC12.exe`

> This file name is subject to change, and usually does with each HDK release.

2. Check the Coordinator and Start Menu Shortcuts boxes:



3. Run the Coordinator by selecting Start Menu > Programs > Imogen > Coordinator.

Displaying the Imogen Control Panel

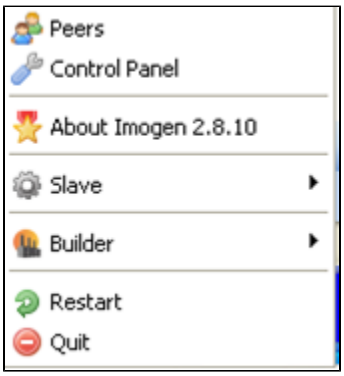
Correct installation requires configuration of Imogen from its default settings.

The necessary settings are described in the install section for each Imogen type (slave, client, coordinator).

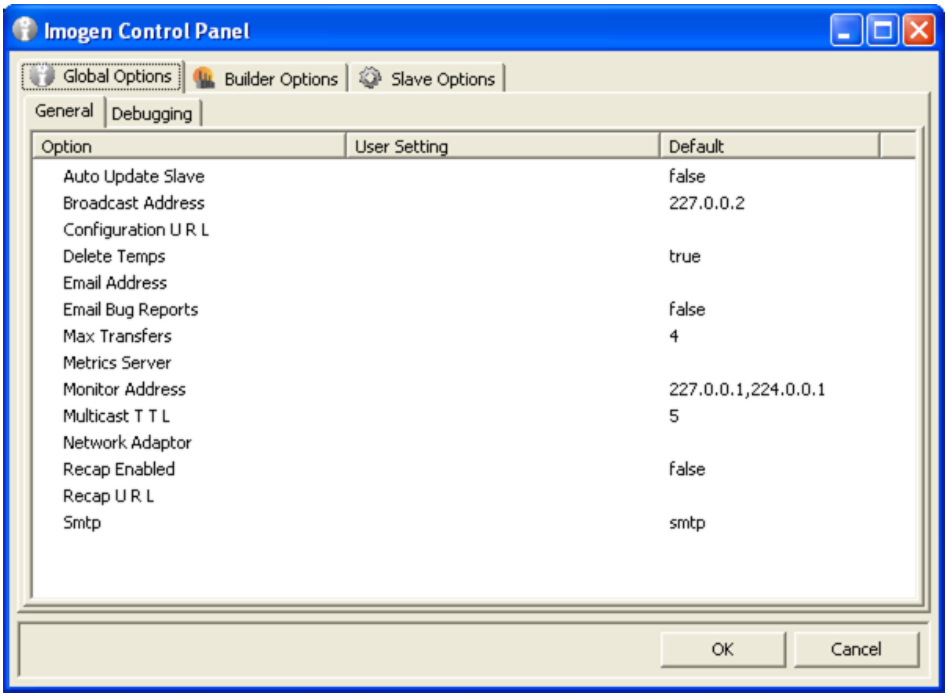
1. Look for the  icon in the system tray.



2. Right click the icon and select Control Panel from the pop-up menu displayed:



The Imogen Control Panel has multiple categories that accessed by the top tabs (Global Options, Builder Options, Slave Options):

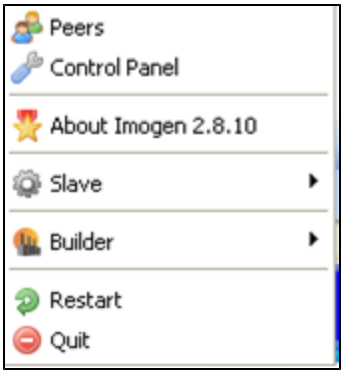


Displaying the Imogen Peers Dialog

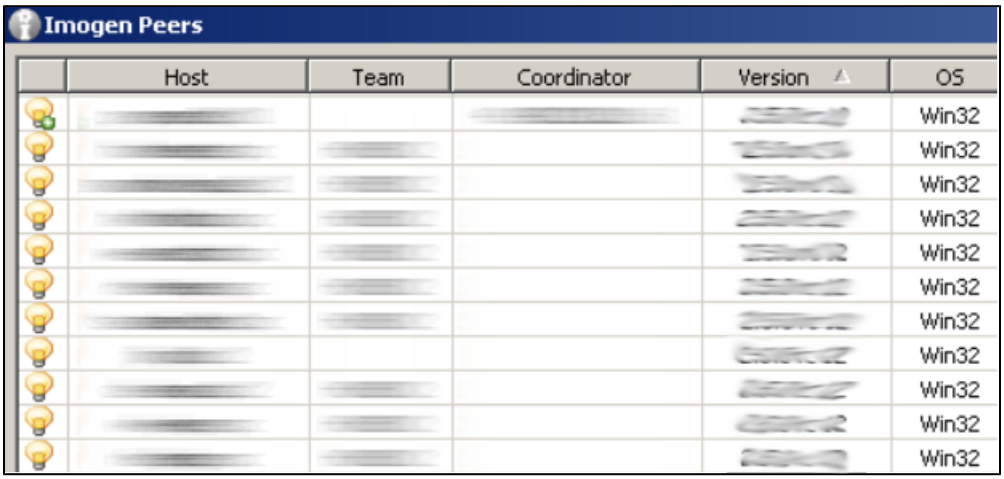
1. Look for the Imogen  icon in the system tray.



2. Right-click on the icon and select Peers from the pop-up menu displayed:



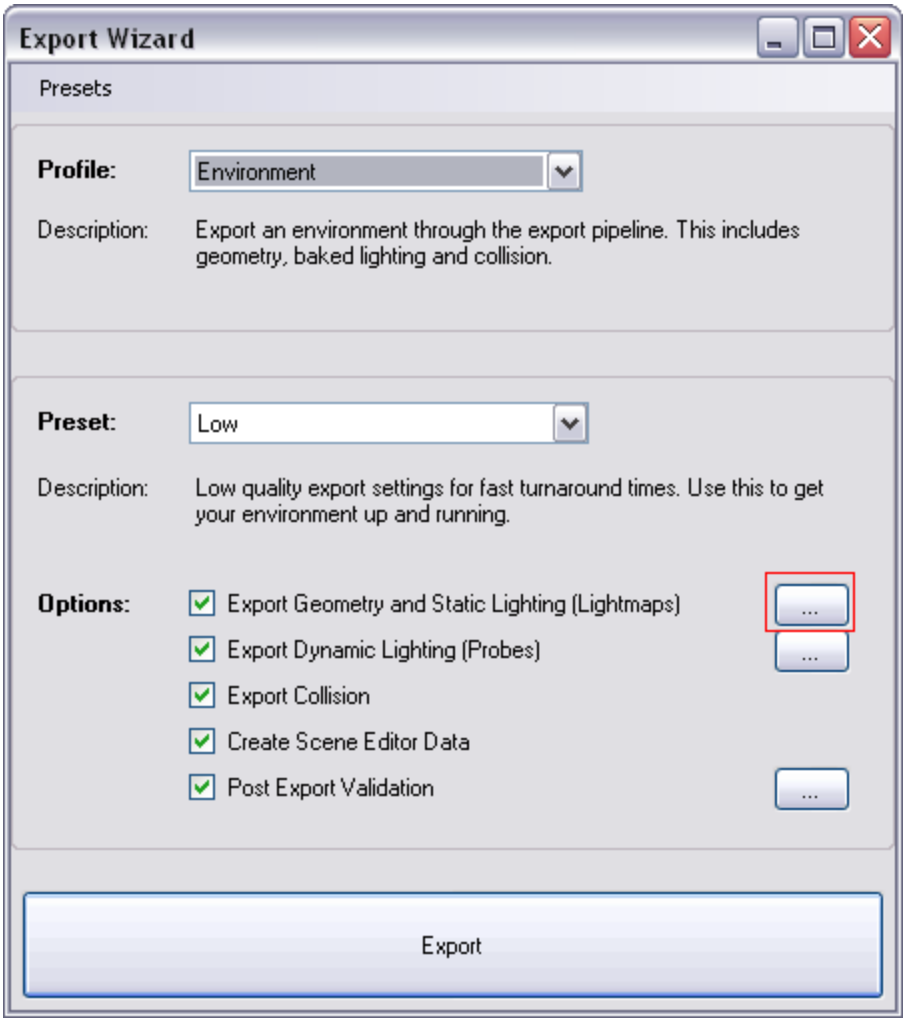
The Imogen Peers dialog offers useful information on available Slaves and their connectivity:



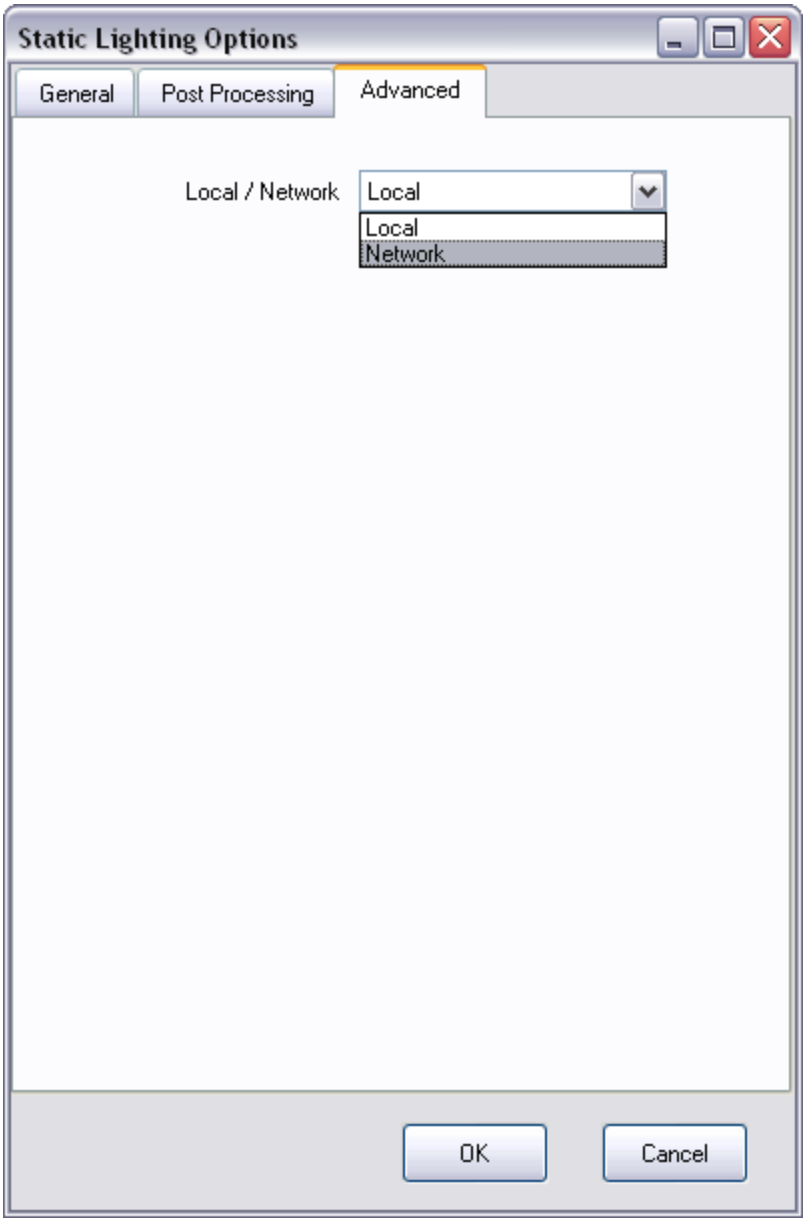
Validating and Exporting for Imogen Distributed Lighting Exporter

In Maya you can access the Export settings only when you run the Export Wizard:

1. Select Home > Export Wizard to display the Export Wizard dialog:



2. Click on the ... button next to Export Geometry and Static Lighting (Lightmaps) and select the Advanced tab.
3. Select Network from the Local / Network drop-down list:

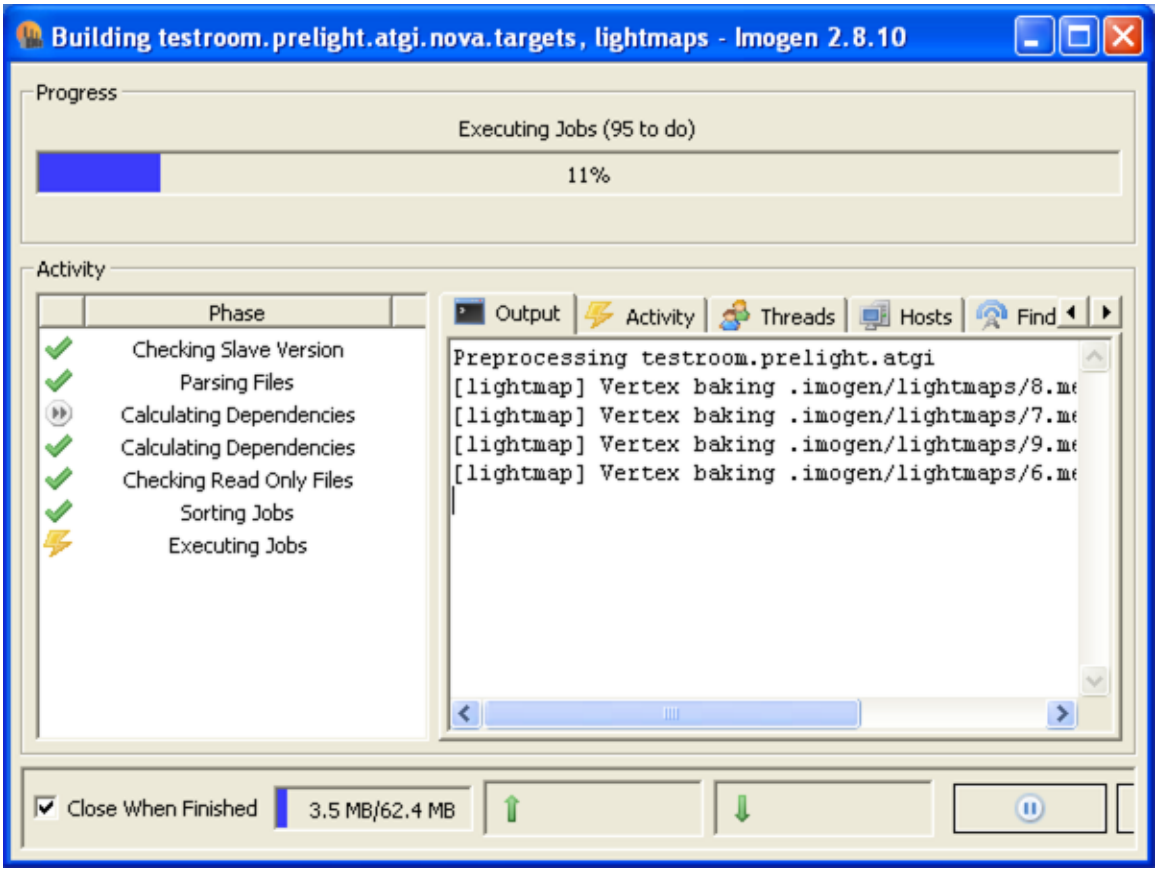


Selecting Network will enable Imogen distribution for both Static and Dynamic lighting exports.

You can set the Imogen Builder window to appear during the lighting phase of the export.

In the Imogen Control Panel, set Builder > General > Window to true.

It gives you the opportunity to cancel the export and to view distribution information should the need to debug network problems arise.



Notes and Troubleshooting

Note	Imogen uses TCP ports 8111-8117 Make sure they are not blocked by your local firewall.
Slaves are not responding	Imogen can provide information during the export process on why a slave is not being used. In the Control Panel, set the Build > Distribution >Slave Excuses option to true. This information is provided in the Window which pops up during export (not the Control Panel). In the Finder tab, you should see responses like "jblogsxppc: Ignoring request: Not idle" or similar.
Slaves do not appear on the peers dialog	This may indicate some network problem. If the machines are on the same subnet then it may be that Imogen has attached itself to the wrong network controller - you can manually set this using the Network Adaptor. If they are on a different subnet, then you should change to a different Monitor Address and Broadcast Address in the Control Panel. If you change these to 227.0.0.1 and 227.0.0.2 respectively, the switches should pass the packets on correctly. The Coordinator only allocates Slaves to Clients. It does not forward all communication. The Slaves must be able to directly connect to the IP address of the Client, making it unlikely to work across different subnets.
Use Slaves Only	It is possible to use only Slaves to perform a job. In the Imogen control panel set Builder > Distribution > Prefer Remote to true.
Choosing Slaves	You cannot choose which Slave machines the Coordinator will assign certain jobs to.

Scene Editor

For Japanese Live Page
最新の日本語版ページはこちら 「Scene Editor - JP」

The Scene Editor is an HDK tool for adding assets to scenes created in Maya, and placing them in the scene as objects, triggers and scripts. You use the Scene Editor to turn the artwork you designed in Maya into an interactive space. Maya exports a scene as a .scproj file to the <HOME_INSTALL_DIR>\build\environments folder. You can open and edit the file in the Scene Editor. Using the Scene Editor interface, you can take all the elements created and open them on the PlayStation®3 development kit or test kit so that you can view your work.

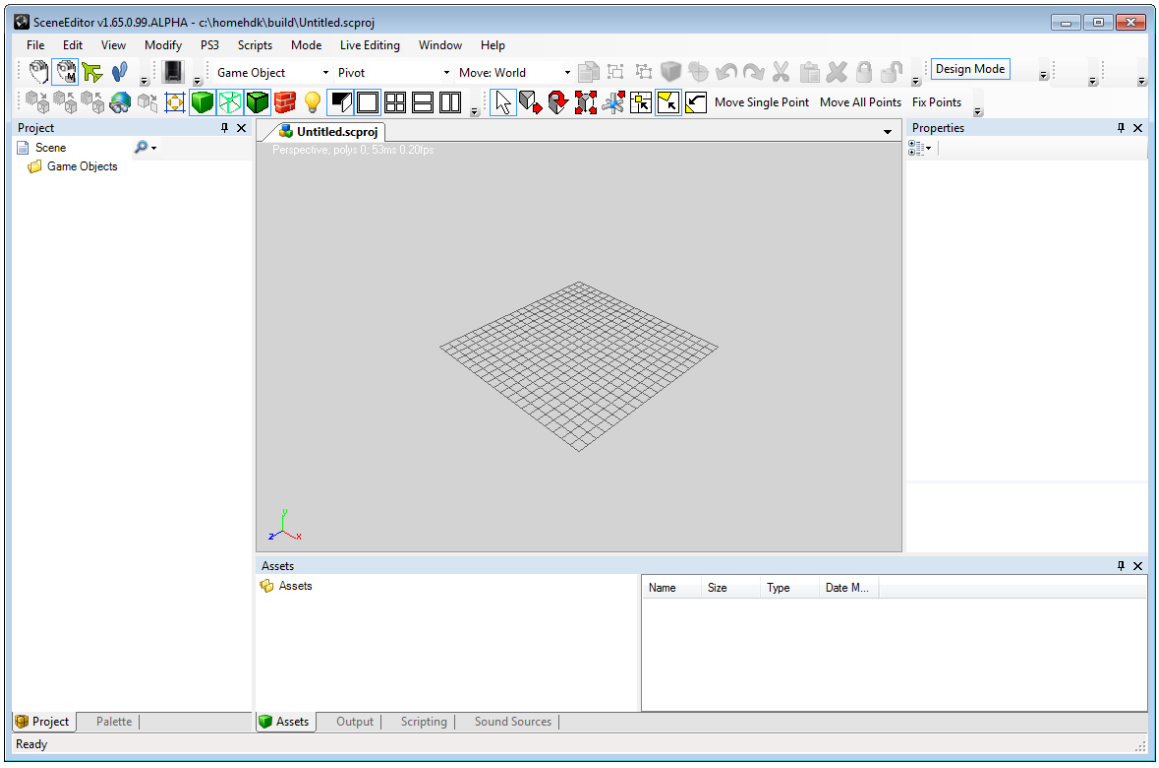
You can use the Scene Editor for many tasks:

- Add and position objects and assets (such as seat locators, picture hooks, Mini-games) in a scene. See Creating a Seat, Adding Picture Hooks and Mini-games.
- Position the avatar spawn point. See Game Object Types.
- Connect scenes to each other, through Dynamic Spawn Points. See Dynamic Spawning.
- Add sounds (ambient, point). See Adding Sounds to a Scene.
- Add particle effects. See ATG Particle Effects Tool.
- Add screens, to add movement and a central point of interest for users. See Introduction to Screens.
- Add event triggers. See Game Object Types.
- Compile your Lua scripts while developing and testing content. See Compiling Scripts When Packaging.

This section describes the Scene Editor and how to use it to place objects and assets in scenes. It also covers scene editing and packaging. The result of a successful packaging of a valid scene in the Scene Editor is an archive that contains the scene information. You submit this file to Quality Assurance (QA) prior to publishing.

Scene Editor Interface

When the Scene Editor first opens, the following interface is displayed:

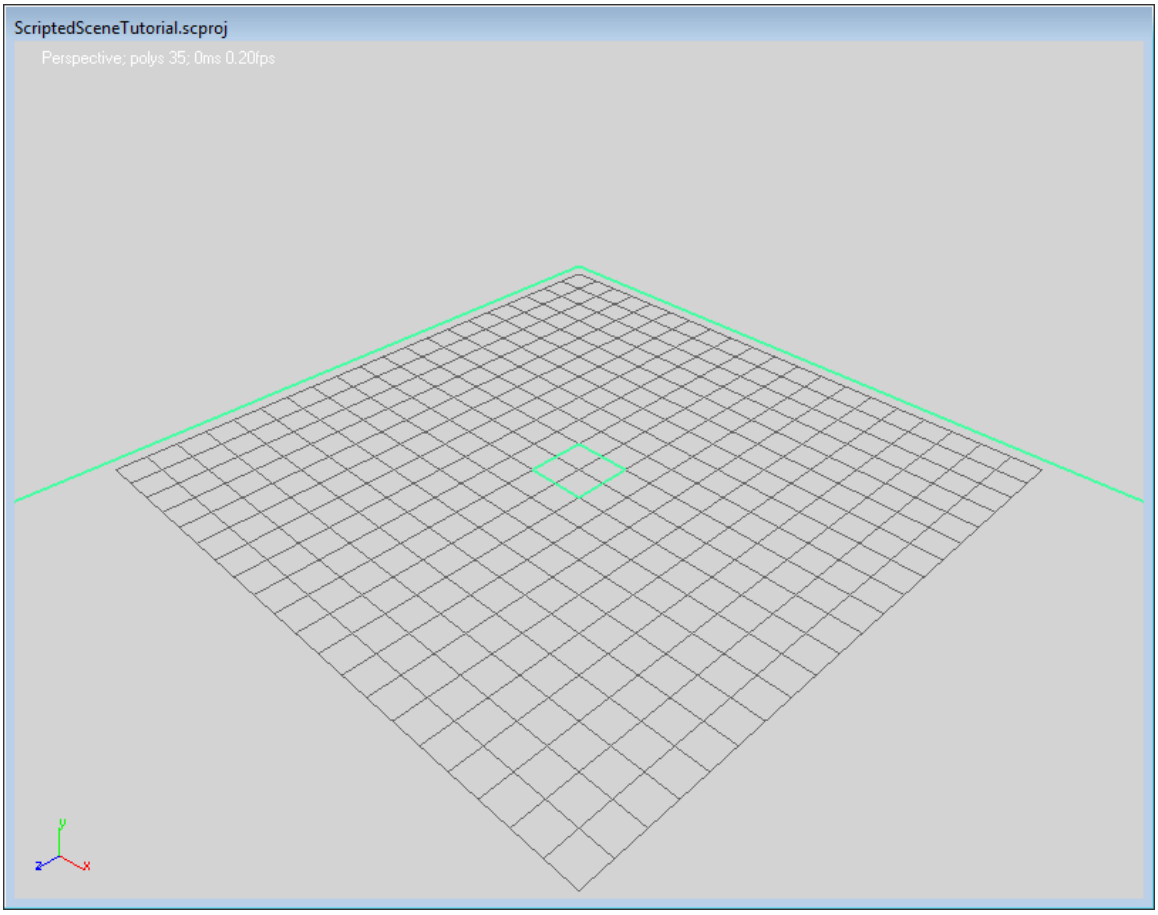


The Scene Editor utilizes the following elements:

- The Design View
- Navigation
- Toolbar Buttons
- Editor Panels

Scene Editor Design View

The Design View displays the currently visible game objects in the scene:



Scene Editor Navigation

The drop-down lists in the toolbar determine how the Design View pivots, rotates and zooms:

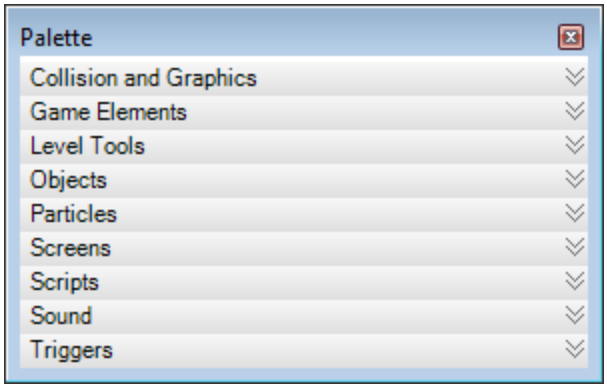


The following options are available:

Drop-down list options	Description
Game Object Any Object Point Transform Node	Use this drop-down list to decide what can be selected in the Design View. You can select game objects, any object in the work area, points, or transform nodes.
Pivot Origin Bottom Center	Use this drop-down list to select the snap behavior. When you choose to have an object snap, you can choose whether the object will snap to its pivot point, to its origin, or to its bottom center. For more information on snapping, see Manipulator Options.
Move:World Move:Local	Use this drop-down list to determine whether when moving an object, the object moves relative to the world orientation or its own orientation.

Scene Editor Panels

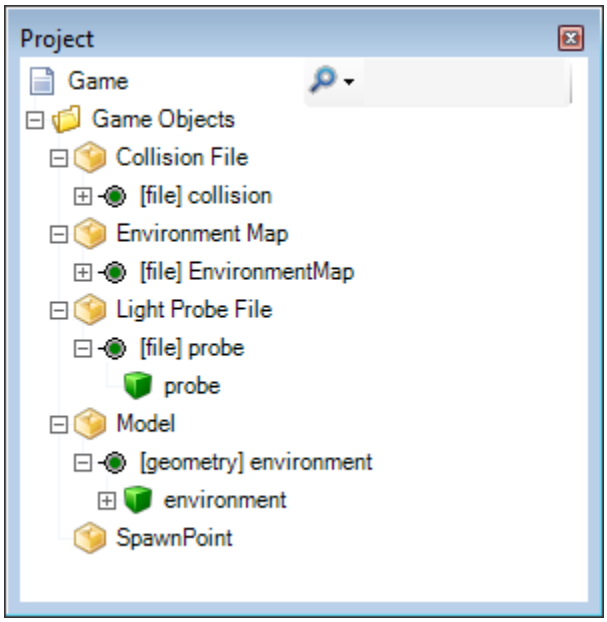
Palette



The Palette panel contains the list of available objects. To add an object to the scene, drag the object from the Palette panel into the Design View or into the hierarchy in the Project panel.

For more information about objects and their properties, see Game Object Types.

Project

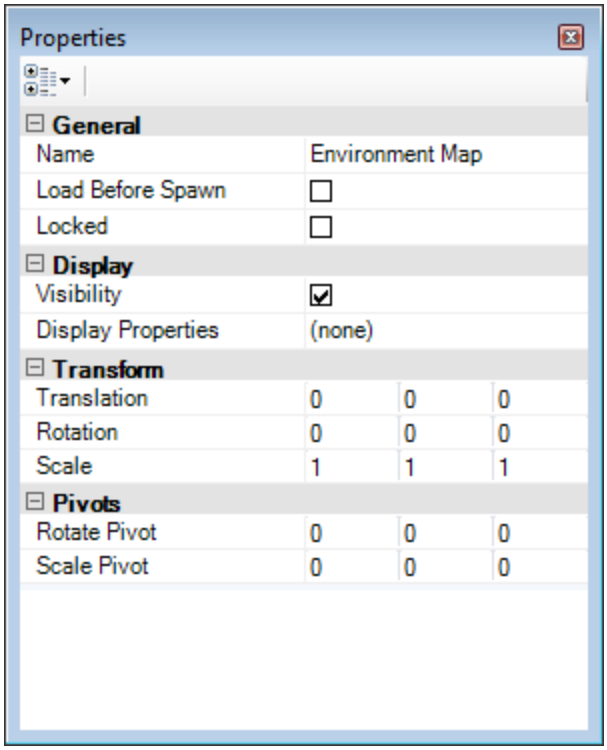


The Project panel shows the hierarchy of game objects in the scene. Game objects are the building blocks of the scene. Every element stored in the scene file is a game object. A valid scene file must have a model file for geometry, a collision file for collision and a spawn region to spawn the player into. Game objects can be added from the Palette panel. The Scene Settings object is not on the palette, but is created automatically when age rating information is added to a scene by using the Edit Scene Age Rating dialog (available from from Edit > Edit Age Ratings).

For more information on spawn points and age ratings, see Preparing Scenes for Packaging.

The Game tree is a hierarchy, which ensures that if Translate, Rotate or Scale is applied to a parent, it applies to all its children as well. This is especially useful when wishing to reposition groups of objects.

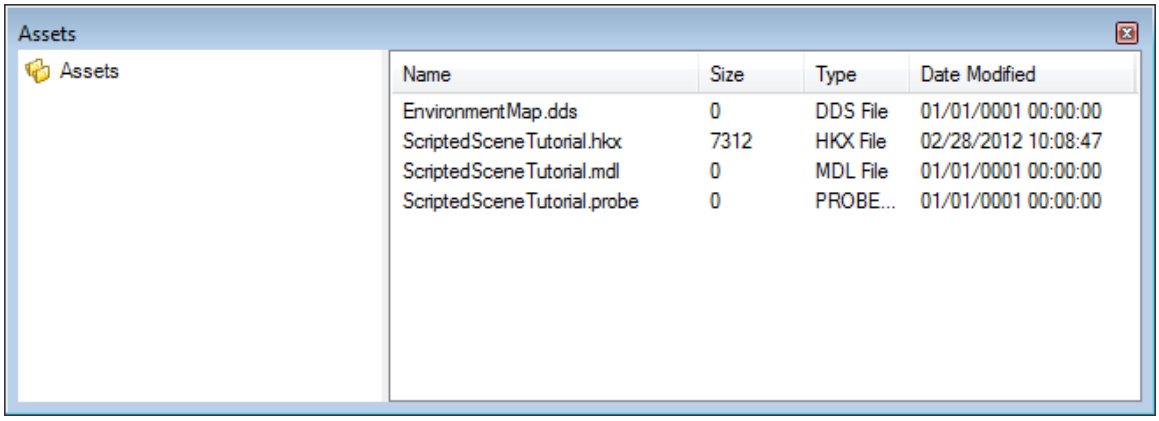
Properties



The Properties panel displays all the properties related to the currently selected game object. Game objects share a set of properties (General, Display, Transform and Pivots). Properties specific to the type of object selected are available under Misc.

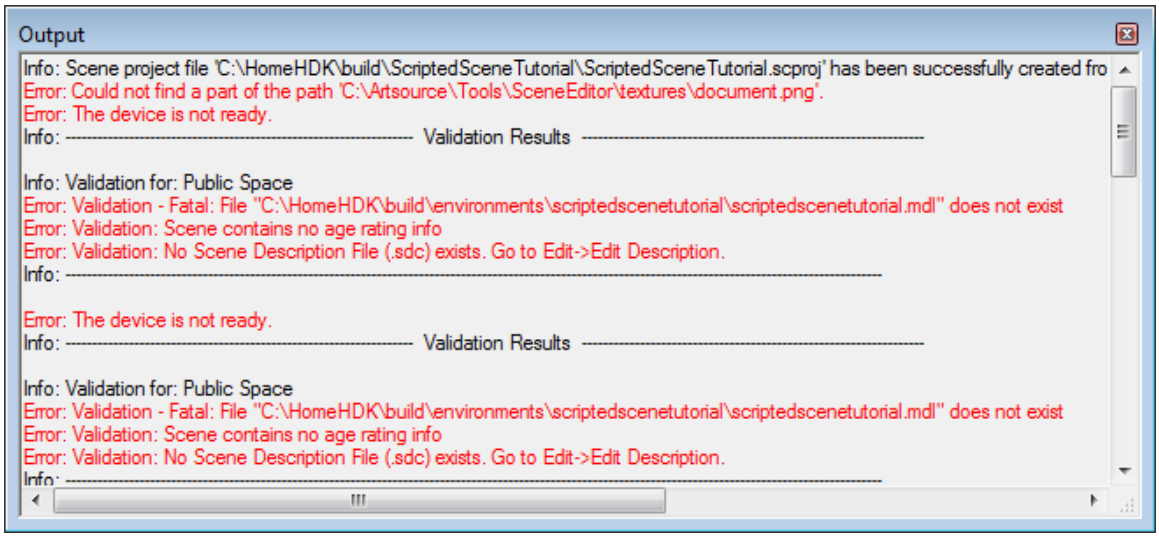
For more information on specific game object properties, see Scene, Object and Asset Properties.

Assets



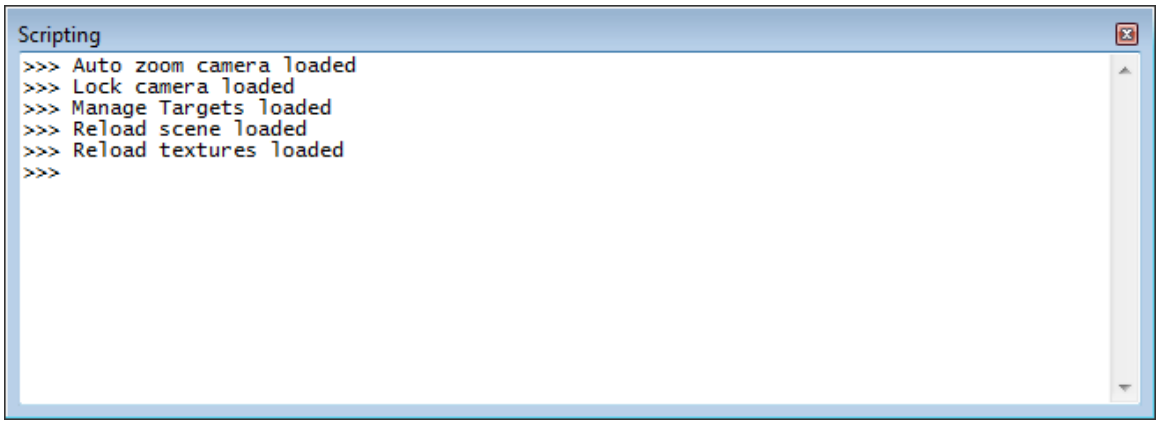
The Assets panel is where external assets such as geometry, sound, music, scripts and text files are displayed. Assets are only added to the scene when referenced in an object in the Project panel. For example, if you add a .mdl file to the Assets panel, it will not be visible in the scene until you add it to a model object in the Project panel.

Output



The Output panel displays error messages.

Scripting



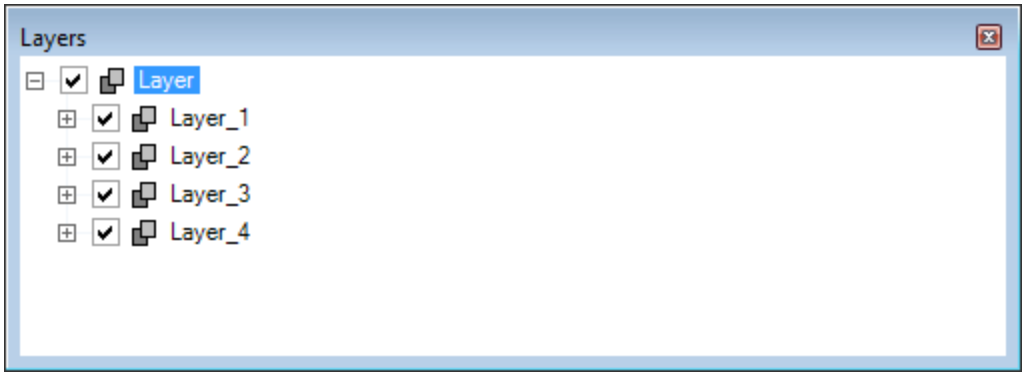
You can use the Scripting panel to:

- Enter HDK API functions
- Load HDK API scripts (i.e. IronPython scripts)
- Enter Dev Debug Console Commands

To enter console commands, you must first connect the Scene Editor to the Target Manager. For more information on how to connect to the Target Manager, see Launching a Scene in PS Home.

See also Running Scripts.

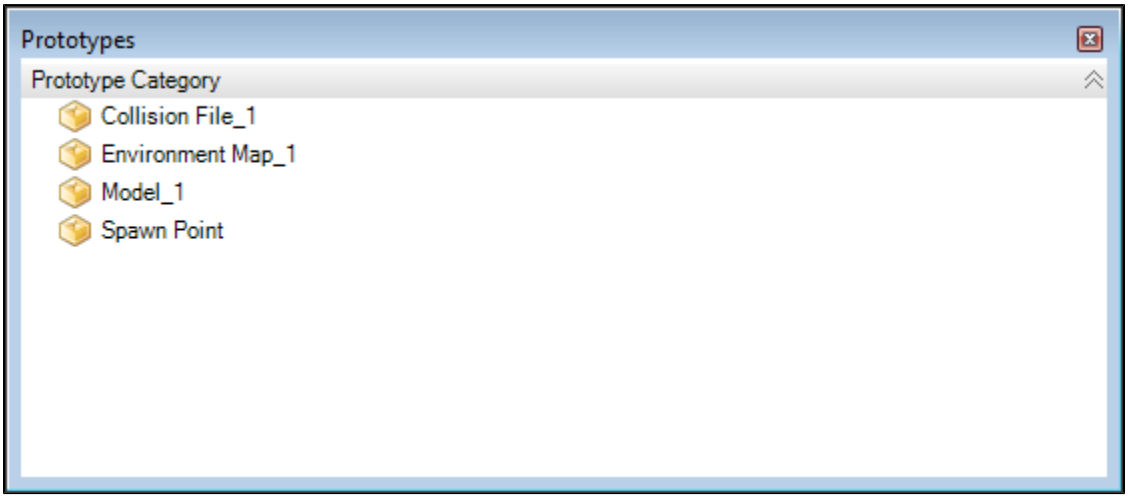
Layers



The Layers panel can be used to switch on or off rendering for arbitrary groups of objects, similar to Photoshop layers. To add a

layer right click in the Layers panel and select Add > Layer. To add objects to the layer simply drag them from the hierarchy in the Project panel into the correct layer.

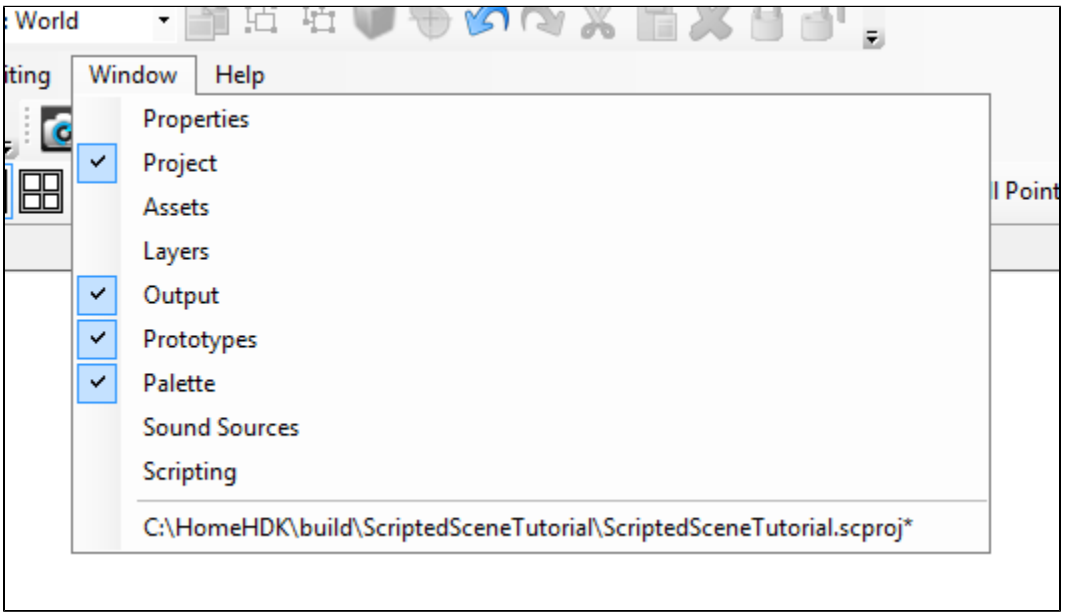
Prototype



The Prototype panel is for editing properties on objects before they are added to the scene. You can add objects to the Prototype panel and edit their properties, but they are not added to the scene until they are dragged from the Prototype panel into the Design View or into the Project panel. It is useful for creating multiple instances of similar items.

Showing, Hiding and Moving Panels

You can show and hide panels by using the Window menu:

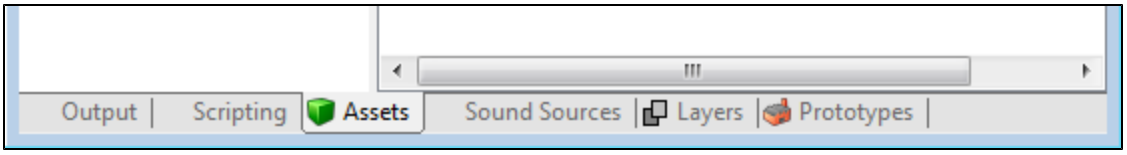


If the name of the panel has a check next it, it's displayed in the Scene Editor. Uncheck the name of the panel to hide it.

You can attach panels to the main Scene Editor interface, have them as standalone windows or as tabs in the same dialog:

- To detach a panel and have it as a standalone window, place your cursor over the title of the panel and drag it out, away from the dialog it's attached to. Alternatively, double-click on the panel's title.
- To attach a panel, drag it over the dialog that you want to attach it to. Alternatively, double-click on the panel's title.

If you attach a panel to another panel, you can view the panels using the tabs:



Scene Editor Toolbar Buttons

File Options



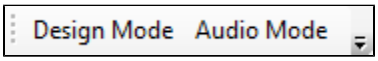
Button	Name	Description
	Save	Saves the scene project file (.scproj).
	New Scene Project	Creates a new scene project file.
	Open	Opens an existing scene project file.
	Package Scene	Packages the scene.

Movement Control Options



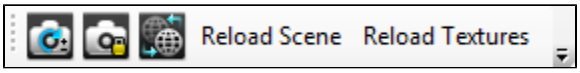
Button	Name	Description
	Arcball	If this button is selected, the following movement options are available: • Zoom: Use the mouse wheel or Alt + the right mouse button. • Free rotation: Use Alt + the left mouse button. • Dragging: Use Alt + the middle mouse button.
	Maya style Trackball	If this button is selected, the following movement options are available: • Zoom: Use the mouse wheel or Alt + the right mouse button. • Rotation (with Up axis maintained): Use Alt + the left mouse button. • Dragging: Use Alt + the middle mouse button.
	Fly	If this button is selected, the following movement options are available: • Ascend/descend/left/right: Use the W, A, S and D keys (adjust the speed using the mouse wheel). • Panning: Use the middle mouse button. • Adjust height: Use Alt + the middle mouse button.
	Walk	If this button is selected, the following movement options are available: • Left, right, forwards or backwards (on current plane): Use the W, A, S and D keys (adjust the speed using the mouse wheel). • Panning: Use the middle mouse button. • Adjust height: Use Alt + the middle mouse button.

Display Mode Options



Button	Name	Description
Design Mode	Design Mode	Displays the full details of the audio objects.
Audio Mode	Audio Mode	Displays the placeholders, as well as emitters. For more information about the Audio Mode, see Introduction to Audio.

Live Editing Options



Button	Name	Description
	Auto Zoom Camera	When you select an object in the Scene Editor, the PS Home camera automatically zooms in on that object. For example, if you select a scene object in the Scene Editor and the PS Home camera will zoom in on that scene object.
	Lock Camera	Locks the PS Home camera to copy the view in the Scene Editor (automatically connect to the Target Manager if not connected).
	Manage Targets	Allows you to choose which target the Scene Editor is connected to. It lists all target PS3s available in the Target Manager. The Scene Editor can only be connected to one target at a time, so if the Scene Editor is already connected to a target, connecting to another target will disconnect the first target.
Reload Scene	Reload Scene	Refreshes the screen loaded in the client without restarting. It also updates the .scene file to reflect changes made in the Scene Editor.
Reload Textures	Reload Textures	Refreshes textures loaded in the client. Only textures that have not changed in total data size since the creation of the texture are loaded. If the reload process is unsuccessful, the original texture is unchanged and a console warning message is displayed.

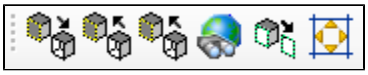
Editing Options



Button	Name	Description
	Copy	Copies the selection onto the clipboard.
	Group	Groups the selection.
	Ungroup	Ungroups the selection.
	Existing Asset	Adds an existing asset.
	Center Pivot	Returns the object's pivot point back to its center.
	Undo	Undoes the last change.

	Redo	Redoes the last edit.
	Cut	Cuts the selection and places it on the clipboard.
	Paste	Pastes the clipboard contents into the scene.
	Delete	Deletes the selection.
	Lock	Locks the selection.
	Unlock	Unlocks the selection.

Show/Hide Pan/Zoom Options



Button	Name	Description
	Hide	Hides the current selection.
	Show Hidden	Shows the last hidden selection.
	Show	Shows the current selection.
	Show All	Shows everything.
	Isolate	Isolates the selection.
	Fit In Active View	Pans and zooms to the center selection.

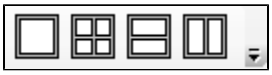
Render Options



Button	Name	Description
	Smooth	Applies smooth shading.
	Wireframe	Applies wireframe rendering.
	Outlined	Applies smooth shading with a wireframe outline.

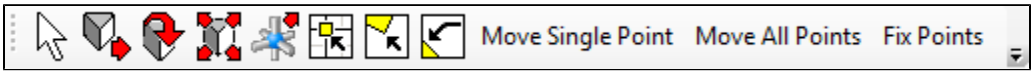
	Textured	Applies textured rendering.
	Lighting	Applies lighting.
	BackFace	Applies backface rendering.

Layout Options



Button	Name	Description
	Single View	Switches the Design View to the single view.
	Quad View	Switches the Design View to the quad view.
	Dual Horizontal View	Switches the Design View to the dual horizontal view.
	Dual Vertical View	Switches the Design View to the dual vertical view.

Manipulator Options



Button	Name	Description
	Select	Activates the Selection Manipulator.
	Move	Activates the Move Manipulator.
	Rotate	Activates the Rotation Manipulator.
	Scale	Activates the Scale Manipulator.
	Move Pivot	Activates the Move-Pivot Manipulator.
	Snap To Grid Point	Snaps to the grid point, rather than just the grid plane. Alternatively, hold down the Ctrl key.
	Snap To Vertex	Snaps to another object's vertex, instead of just its surface. Alternatively, hold down the Shift key.
	Rotate On Snap	When snapping to another object, rotates the object to be aligned to the target surface.

Move Single Point	Move Single Point	Activates the Move Single Point Manipulator.
Move All Points	Move All Points	Activates the Move All Points Manipulator.
Fix Points	Fix Points	Fixes the select object's points to form a valid shape.

Launch Options

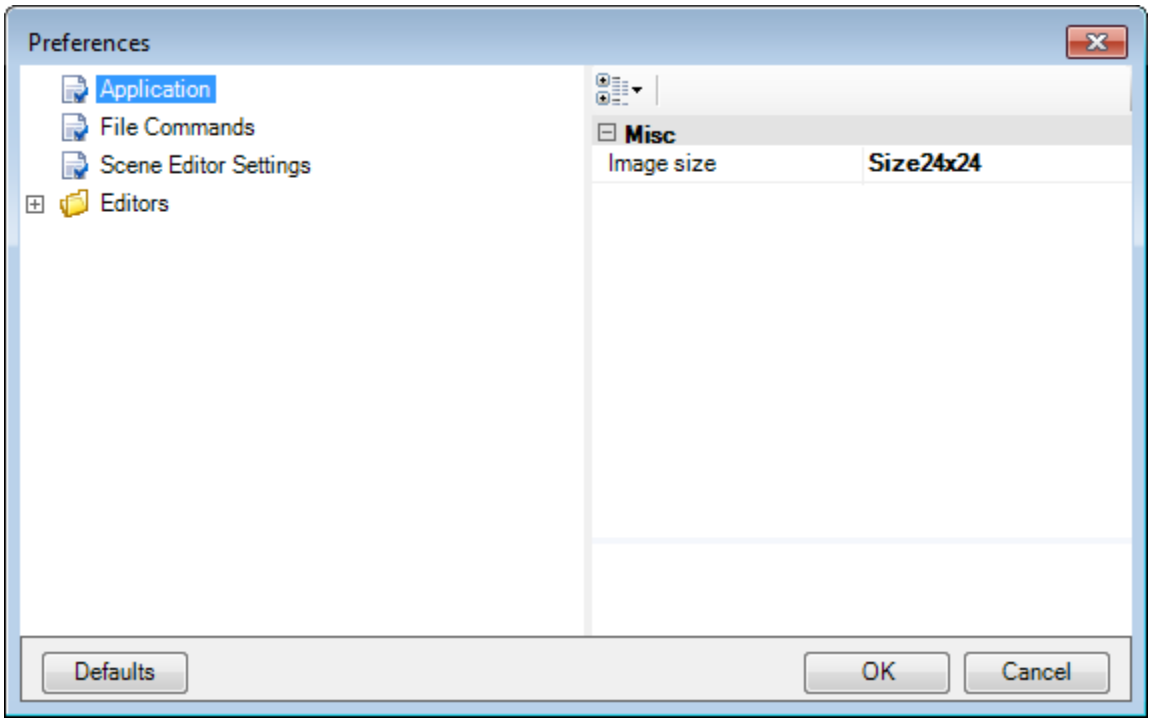


Button	Name	Description
	Launch Home	Allows you to quickly launch a scene in PS Home from the Scene Editor.

Setting Scene Editor Preferences

To set your Scene Editor's preferences:

1. Select Edit > Preferences to display the Preferences dialog:



2. Update your preferences as required, by clicking on the area on the left side of the Preferences dialog, then making the changes needed on the right side of the dialog.

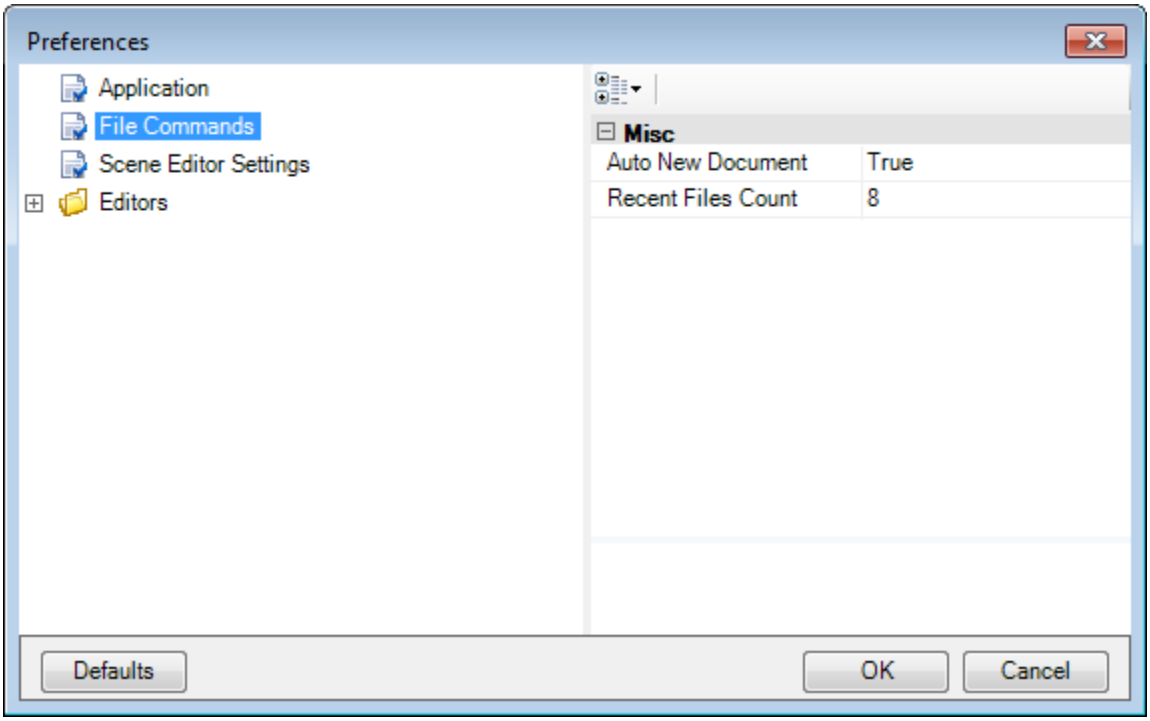
> If at any point you want to return any changed values to the installation defaults, click the Defaults button.

You can change the following aspects of the Scene Editor:

- Application

  Select how big you want the Scene Editor's toolbars, displayed at the top of the Scene Editor, to be from the Image size drop-down list (see the image above).

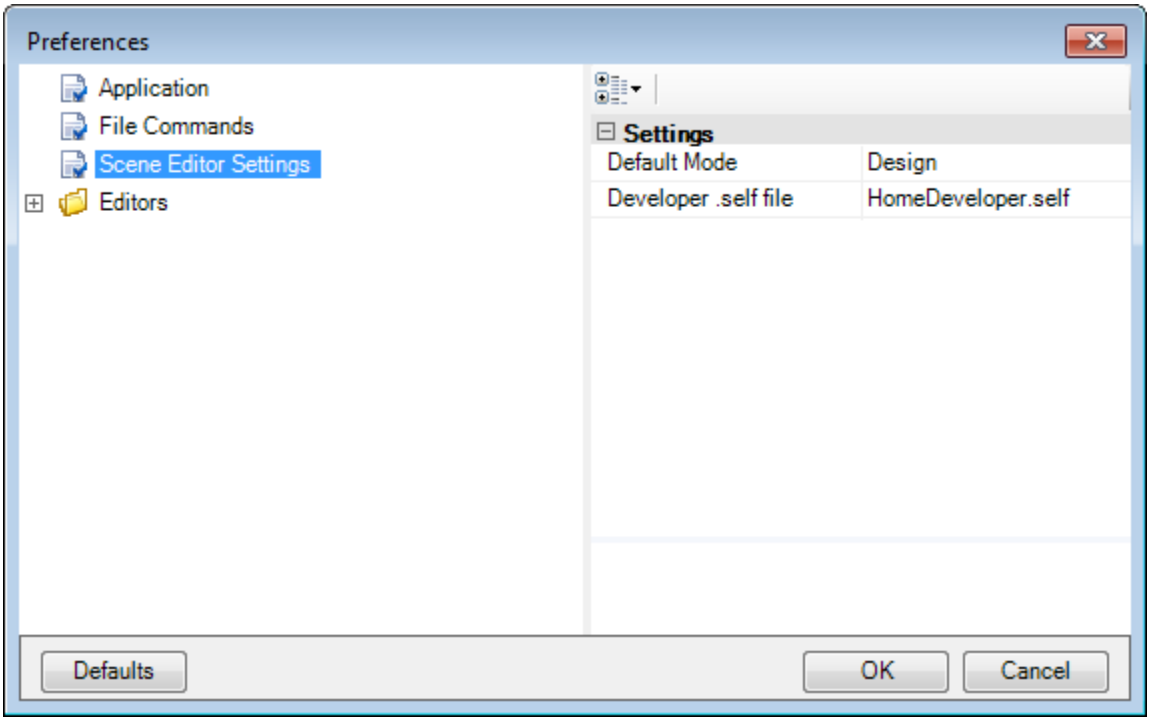
- File Commands



To change the File Commands preferences:

a. If you want the Scene Editor to automatically create a new .scproj file if an existing file hasn't been opened, select True from the Auto New Document drop-down list. If you don't want a new file created, select False.
b. To change the number of recent file names saved by the Scene Editor, enter the number required in the Recent Files Count field.

- Scene Editor Settings



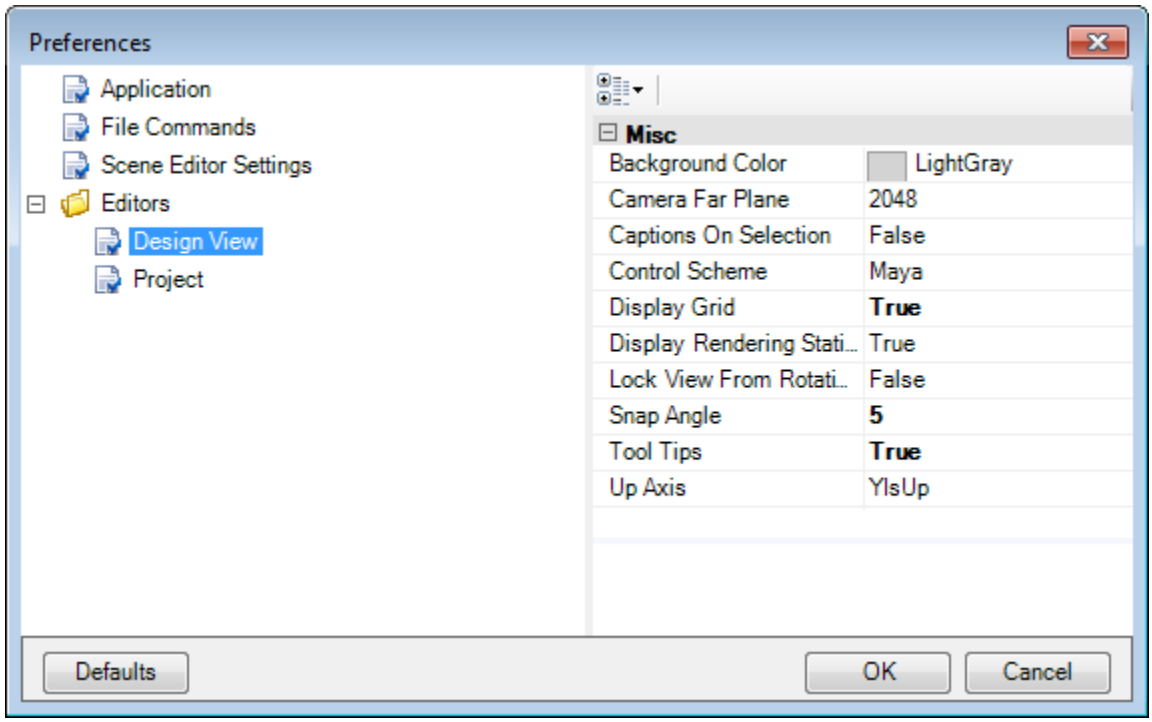
To change the Scene Editor Settings preferences:

a. To set the Scene Editor to open in Design Mode so the full details of the audio objects are not displayed, select Design Mode from the Default Mode drop-down list. To open in Audio Mode so the placeholders and emitters are displayed, select Audio Mode.

   If you close the Scene Editor in a mode, that mode becomes the default when it is next opened. For more information about the Audio Mode, see Adding Sounds to a Scene.

b. To specify the executable file to be launched when the Launch button is clicked, enter the file name in the Developer .self file field.

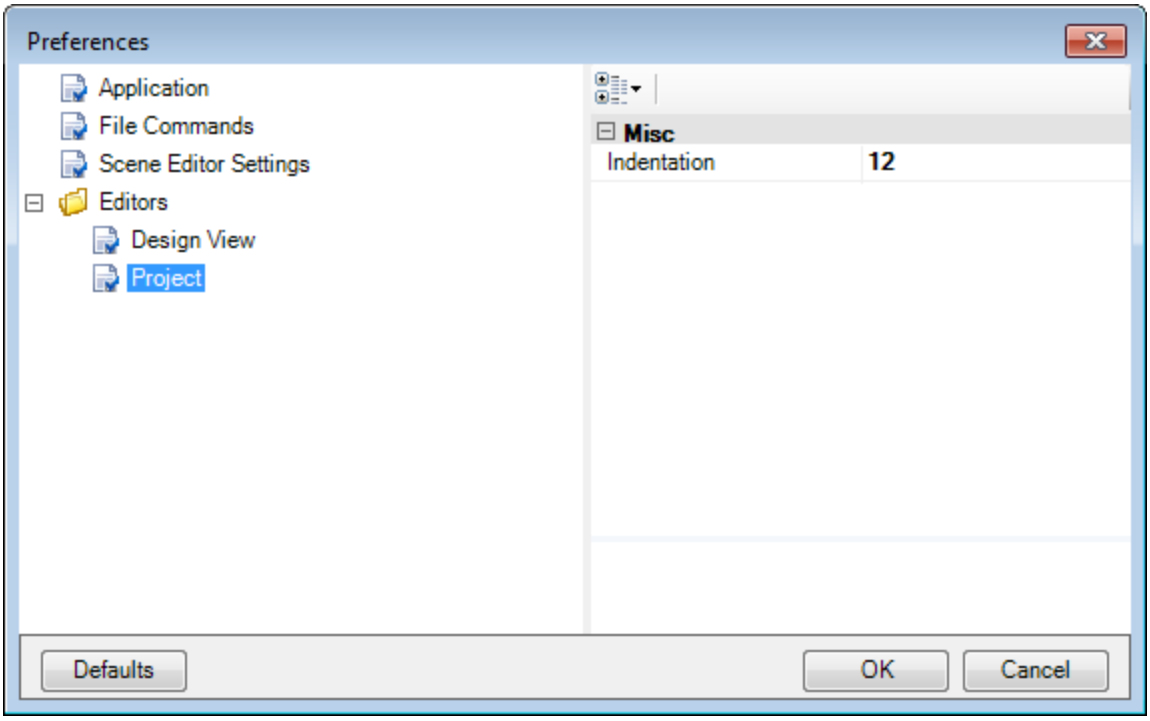
- Design View



To change the Design View preferences:

a. To change the color of the Design View's background display, select the color you want to use from the Background Color drop-down list.
b. To set the distance from the camera that the scene will disappear from view, enter the distance in the Camera Far Plane field.
c. To display captions on selected objects in the Design View, select True from the Captions On Selection drop-down list. To hide the captions, select False.
d. To set the mouse and keyboard actions that control camera and selection behavior, select the preferred scheme from the Control Scheme drop-down list.
e. To display the grid, select True from the Display Grid drop-down list. To hide the grid, select False.
f. To display the rendering statistics for Scene Editor, select True from the Display Rendering Statistics drop-down list. To hide the statistics, select False.
g. To prevent you from rotating the camera while in Top, Side or Front View, select True from the Lock View From Rotating drop-down list. To be able to rotate the camera, select False.
h. To set the angle increment that the geometry will rotate when using the Rotation Manipulator, enter the angle in the Snap Angle field (e.g. enter 5 for every rotation to be 5 degrees). Enter 0 to turn it off.
i. To display tool tips, select True from the Tool Tips drop-down list. To hide the tool tips, select False.
j. To set the Up Axis to the Y Axis, select YIsUp from the Up Axis drop-down list. To set it the Z Axis, select ZIsUp.

- Project



Enter the number of pixels to indent for each level of the Project hierarchy display in the Indentation field.

3. Click OK to save your changes.

Opening Scenes in the Scene Editor

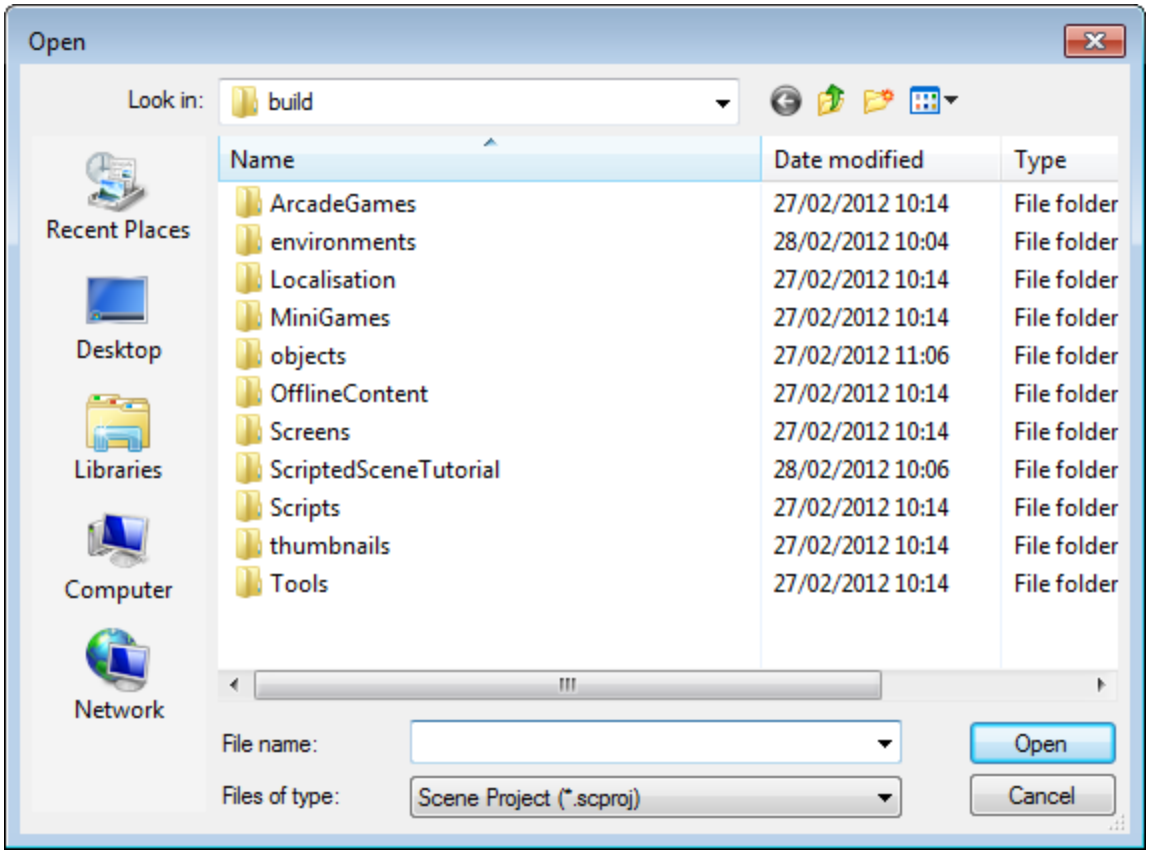
You can open scenes made using DCC tools (Maya) in the Scene Editor.

From HDK 1.60, the Scene Editor started using a different file format for scene files, scene project (.scproj). When you use the Scene Editor to open a scene file (.scene) file created prior to HDK 1.60, you are prompted to convert the file to a scene project file. The Home Client continues to use .scene files.

> ⚠ The only program you can use to edit scene files for PS Home is the Scene Editor.

To open a scene in the Scene Editor:

1. Click the Open button  on the toolbar, or select File > Open to display the Open dialog:



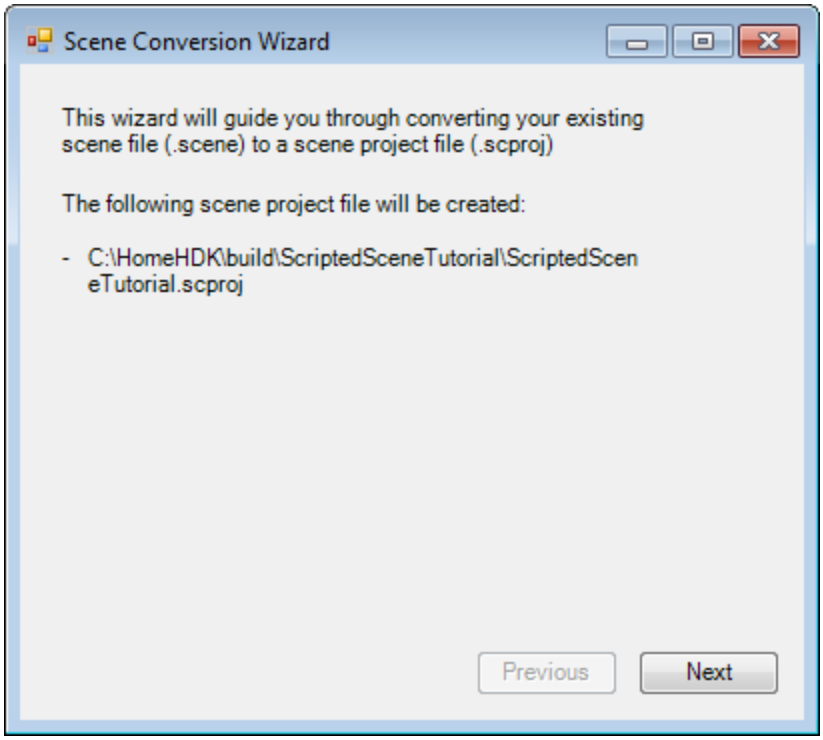
2. If you want to open scene file created prior to HDK 1.60, select Scene (.scene) from the Files of type drop-down list.

   For scene files created in HDK 1.60 or later, the correct file type, Scene Project (.scproj) should already be selected by default.

3. Browse to and select the file required, then click Open.
4. If you selected to open a .scproj file, the file opens in the Scene Editor.

> If the file was from Maya, a set of intermediate ATGI format files are created. The minimum intermediate files needed by the Scene Editor are the .hkx, .hkx, .xml, and the .atgi files.

If you selected to open a .scene file, the Scene Conversion Wizard is displayed to convert your existing .scene file to a .scproj file:



To convert the .scene file to a .scproj file:

a. Click Next.
b. If you want to create a backup of your scene, check the Create Backup box.
c. Click Convert to convert the file. When the conversion is complete, the file opens in the Scene Editor.

If you need to add different scene files together to make a bigger environment in the Scene Editor, import the model, collision and probe files as assets and attach them to their equivalent objects (found under Collision and Graphics in the Palette panel).

You can now start adding game objects and assets in your scene. You need the following panels open in the Scene Editor to edit a scene:

- Properties
- Palette
- Assets
- Project

For more information on how to open these panels in the Scene Editor, see Scene Editor Interface.

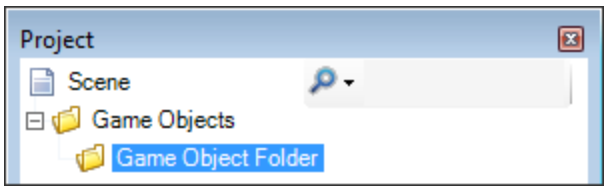
Adding Objects to a Scene

Most objects can be added to a scene using the process described below. However, the process for adding seats and picture hooks is slightly different. For information on adding these objects, see Creating a Seat and Adding Picture Hooks.

To add objects to a scene in the Scene Editor:

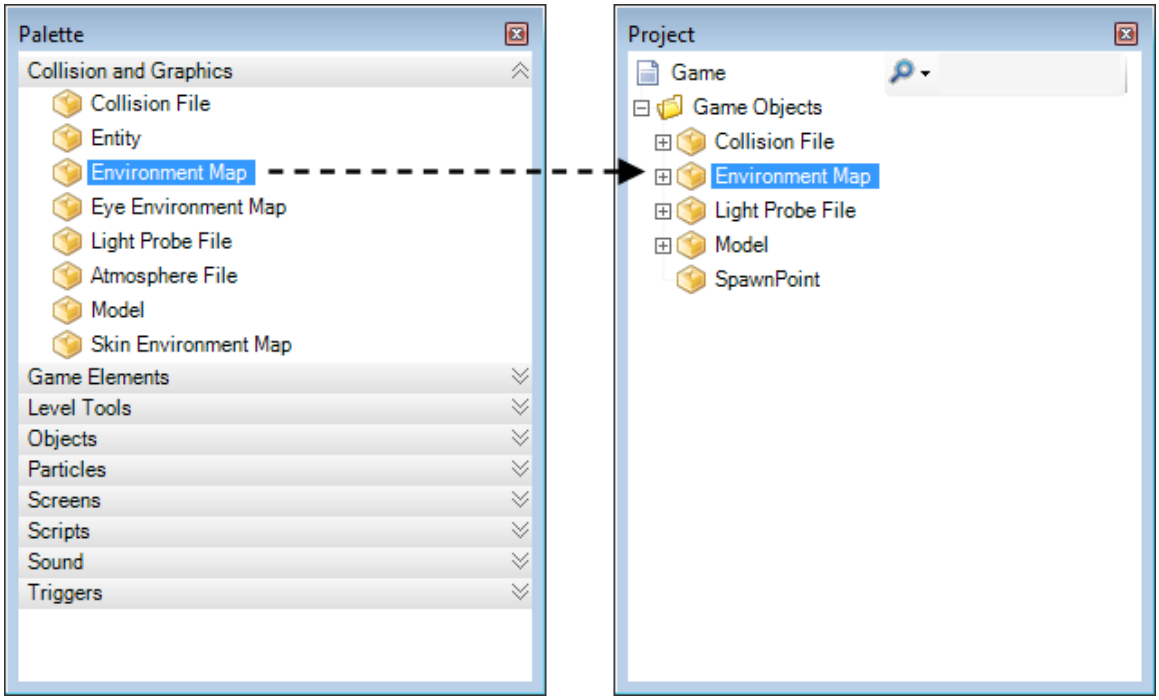
1. If you are planning to add more than one object to your scene, you can add a folder for each one. Right click on the Game Objects folder at the top of the Project panel and select Add > Game Object Folder from the menu displayed.

   A new Game Object Folder folder is added under the top level Game Objects folder:



You can rename the folder by clicking on it and entering the new name.

2. Select the object you want to add to your scene in the Palette panel.
3. Drag the object from the Palette panel to the Game Objects folder in the Project panel, or directly onto the Design View. The object is added under the Game Objects folder:



Alternatively, right click on the Game Objects folder in the Project panel and select Add, then the name of the object that you want from the menu displayed.

Once an object has been added to the scene, you should determine whether the object needs an asset to package successfully, and if it does, add that asset to the object. Not all objects require asset files. For example, the Spawn Point is automatically created when you export your scene from Maya, and need only to be positioned in your scene. For more information about how to add assets to an object, see Adding Assets to a Scene.

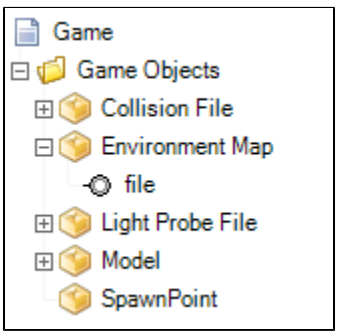
For more information about how to manage objects within a scene, see Working With Objects in a Scene.

Adding Assets to a Scene

Once an object has been added to the scene, you should whether object needs an asset to package successfully, and if it does, add that asset to the object. Not all objects require asset files. For example, the Spawn Point is automatically created when you export your scene from Maya, and need only to be positioned in your scene.

Checking whether an Object needs an Asset

Click ⊞ next to the object you've added in the Project panel to see if the object needs an asset to package successfully. If an object needs an asset, you see a file entry under the object in the hierarchy:

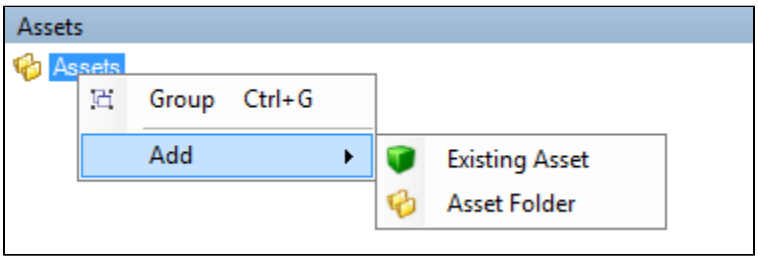


For more information on how to add objects to a scene, see Adding Objects to a Scene.

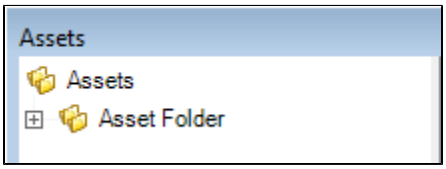
Adding Asset Folders

If you want to need multiple assets for your scene, you can create asset folders in the Assets Panel to more easily manage the assets.

To add a folder, right click over the Assets folder at the top of the Assets panel and select Add > Asset Folder from the menu displayed:



A new Asset Folder folder is added under the top level Assets folder:

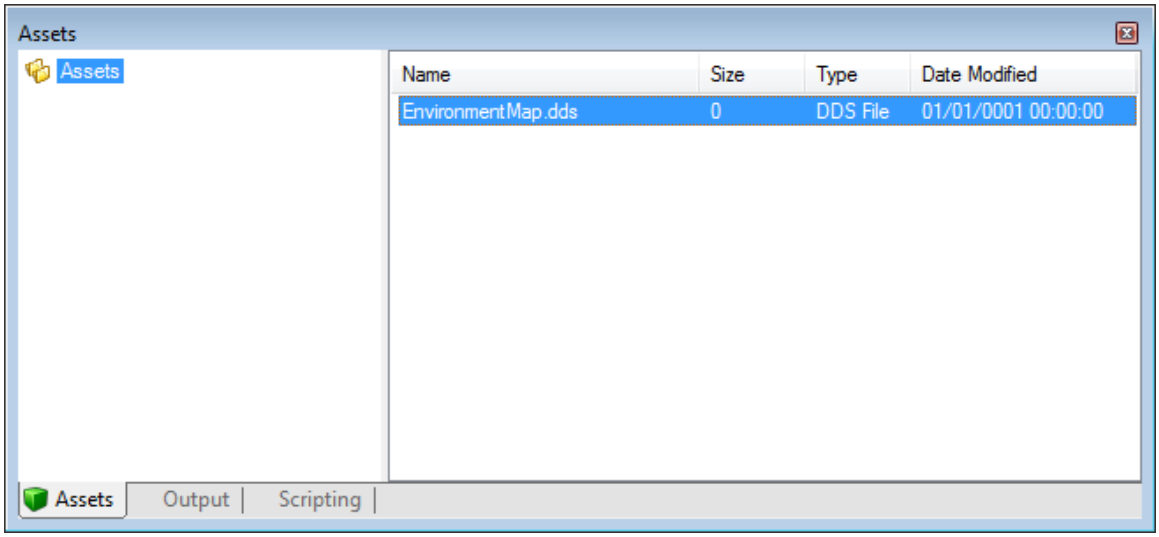


You can rename the folder by clicking on it and entering the new name.

Adding Assets to an Object

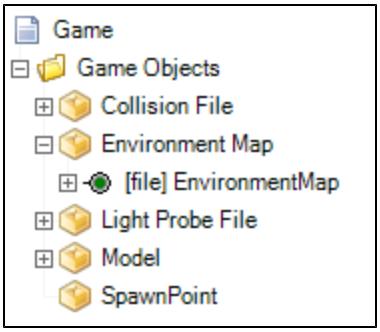
To add an asset to an object:

1. Right click on the assets folder that you want to add the asset to on the left side of the Assets panel, and select Add > Existing Asset to display the Open dialog.
2. Browse to and select the file required, then click Open. The asset is added to the right side of the Assets panel:



The asset file type depends on the kind of object you add to your scene. For example, for an Environment Map, you must add a .dds file, for a soundbank, you must add a .bnk file.

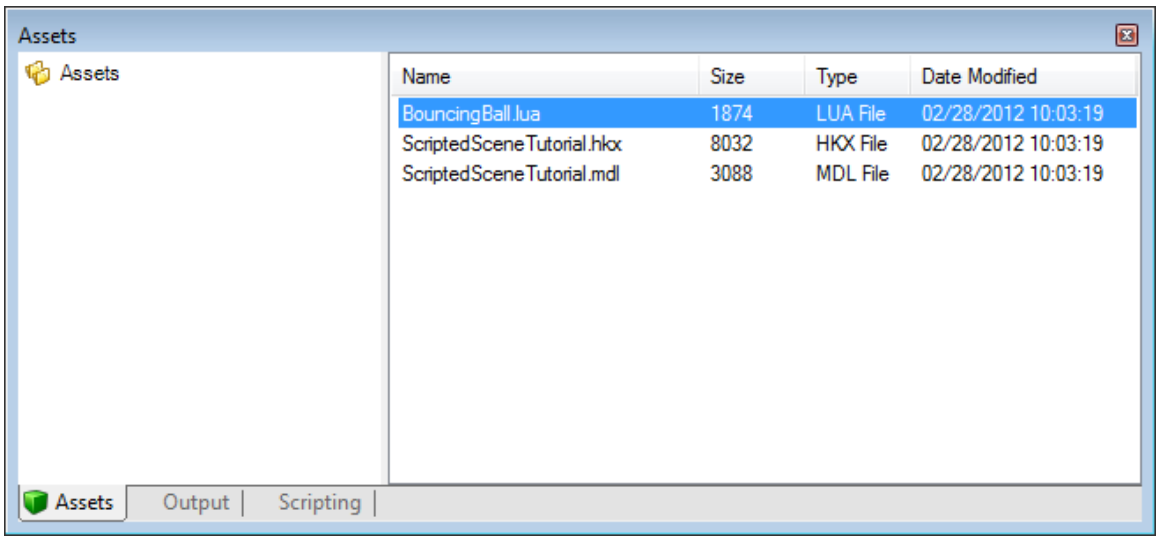
3. Apply the asset by dragging it from the Assets panel onto the file entry under the object in the Project panel. When the asset has been applied, the file entry turns green:



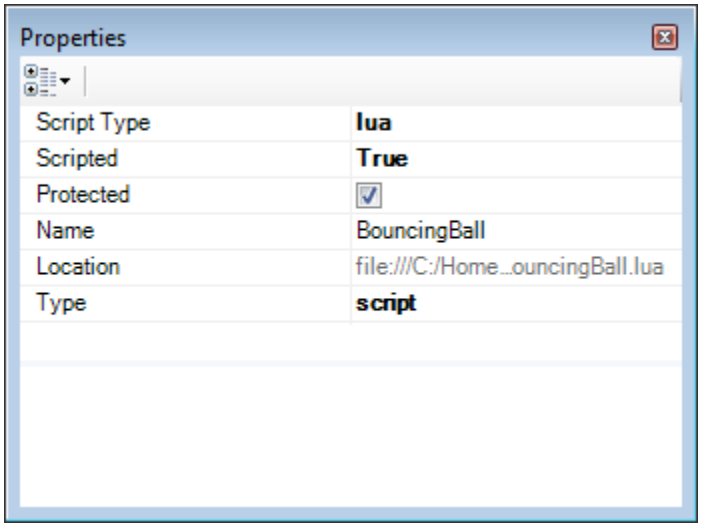
Lua Script Assets

For scenes created after HDK 1.3.5, Lua scripts that added to your scene as assets are automatically protected and the script type is set to lua so that it packages successfully. However, for scenes created before HDK 1.3.5 you need to manually change each Lua asset to ensure the asset's protected and has the right script type.

To change Lua script assets created before HDK 1.3.5, select the Lua script asset in the Assets panel:



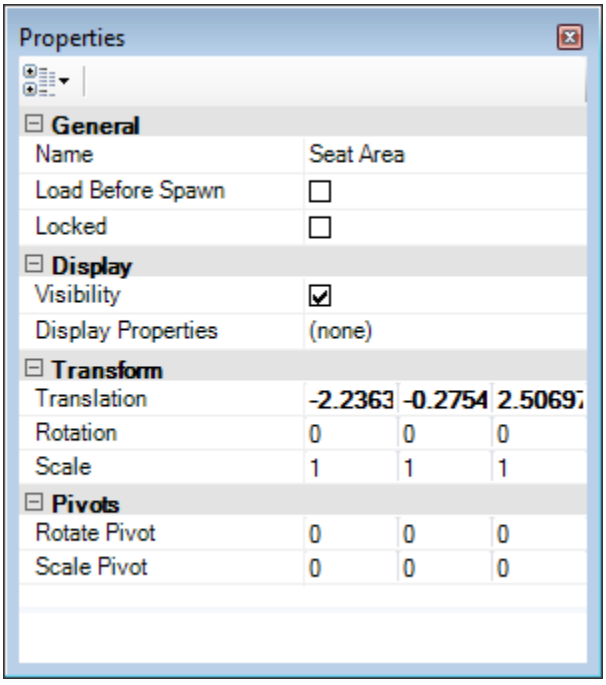
In the Properties panel, ensure that the Protected box is checked and that lua is selected from the Script Type drop-down list:



Working With Objects in a Scene

Editing an Object's Properties

To edit an object in your scene in the Scene Editor, select the object in the Design View or in the Project panel. The properties for the selected object are automatically displayed in the Properties panel, for example:



Edit the object's properties in the Properties panel as required. All of the objects share common properties (those properties grouped under General, Display, Transform and Pivot). These are also object-specific properties that depend on the object selected. For more information on the object properties available, see Scene, Object and Asset Properties.

Moving an Object in a Scene

When you select an object, you can use the manipulators to move it around a scene, rotate it, or stretch it. You can select which manipulator to use by clicking on a button on the toolbar or by pressing the appropriate shortcut key:

-  or E key: Use the three manipulation directional compass points to drag the object along the axes.
-  or M key: Use the rotation manipulators to rotate the object into position.
-  of R key: Use the three scale manipulators to resize the object.
-  : Use the pivot manipulators to change the object's pivot point.

You can get more precise placement by editing the object's Transform and Pivots properties (see above).

Removing an Object from a Scene

To remove an object, select the object in the Project panel and do one of the following:

- Click the Delete button  on the toolbar.
- Select Edit > Delete.
- Right click on the object in the Project panel and select Delete from the menu displayed.

The object is removed.

> Any asset added to the deleted object remains in the scene unless you remove the asset as well.

Game Object Types

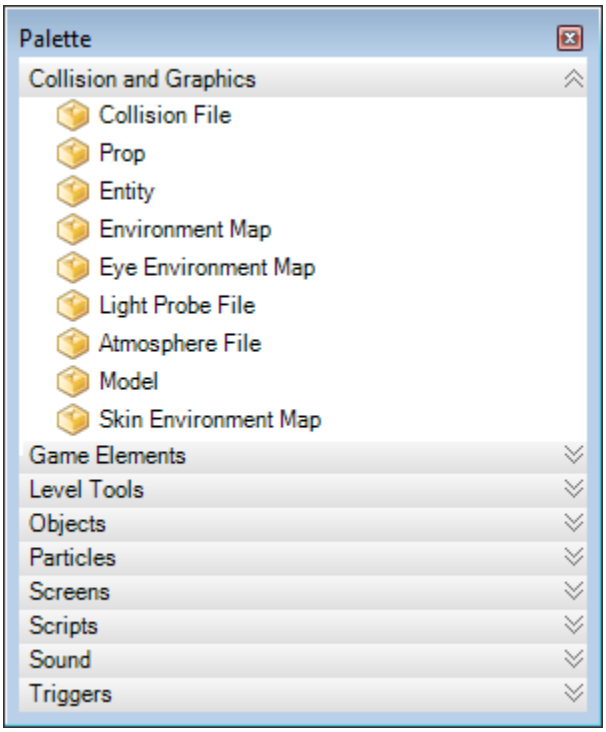
For Japanese Live Page
最新の日本語版ページはこちら 「ゲームオブジェクトタイプ」

The following game object types are available from the Palette panel:

- Collision and Graphics
- Game Elements
- Level Tools
- Objects
- Particles
- Screens
- Scripts
- Sound
- Triggers

For more information about game object properties, see Scene, Object and Asset Properties.

Collision and Graphics



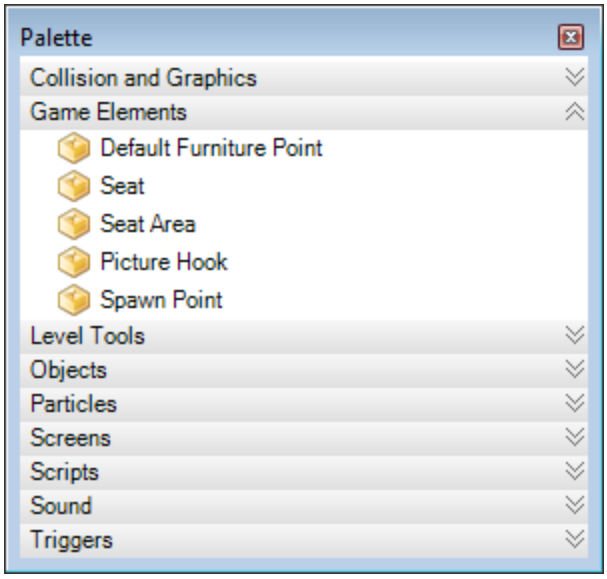
The following collision and graphics objects are available:

Object	Description	Samples	Related Documentation
Collision File	Adds the collision information for the scene from the Havok™ (.hkx) file. The collision file is created when you export from Maya. It contains everything in the Collision node in Maya. ⚠ You can only have one collision file per scene.	All of the samples that have a .scene/.scproj file (e.g. HTS Aeronautilus, Arcade Tutorial, etc).	Collision in PS Home Environments Validating and Exporting in Maya

Prop	Enables you to add a model and its associated collision to your scene. This is useful for placing recurring models which are scattered about your scene, such as trees. The prop object contains two file slots in the Project panel, geometry and collision, which can reference a .mdl and .hkx asset.	N/A	N/A
Entity	Enables you to add animated models to your scene. The entity object contains three file slots in the Project panel, animation, geometry and skeleton, to which you can add an .ani, .mdl and .skn asset respectively.	Tools > Simple Animation	Joint Animation
Environment Map	Map of the scene that will be referenced by shaders that reflect the scene. ⚠ A custom Environment Map is required for every scene.	All of the samples that have a .scene/.scproj file (e.g. HTS Aeronautilus, Arcade Tutorial, etc).	PS Home Spaces Scene Editor Maya Preparing Scenes for Packaging
Eye Environment Map	Map of the environment that is reflected in avatars' eyes.	N/A	N/A
Light Probe File	Adds the light probe information for the scene created in the DCC Tools.	All of the samples that have a .scene/.scproj file (e.g. HTS Aeronautilus, Arcade Tutorial, etc).	Environments Lightmaps - Best Practice Dynamic Lighting Creating and Exporting Lighting and Light Volumes
Atmosphere File	Enables you to adds the .atmos file to your scene, giving you dynamic sky.	HTS Aeronautilus	Sky Design Sky Design Quick Start Tutorial
Model	Loads model files (added as assets) to the scene. The model file (.mdl) is created when you export from Maya. It contains everything in the Geometry node in Maya. You can add more than one environment made in the DCC Tools to a single scene by adding more model objects.	All of the samples that have a .scene/.scproj file (e.g. HTS Aeronautilus, Arcade Tutorial, etc).	Environments
Skin Environment Map	Adds the skin environment map.	N/A	N/A

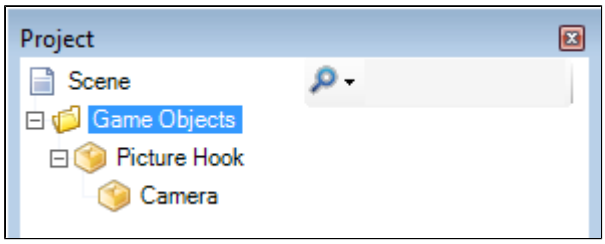
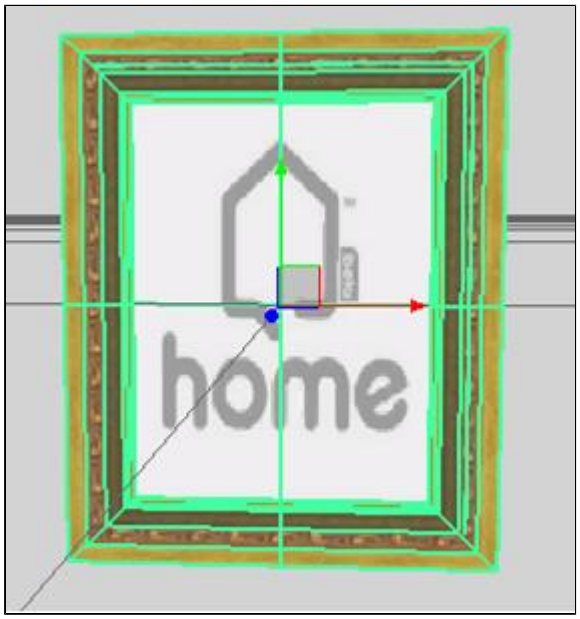
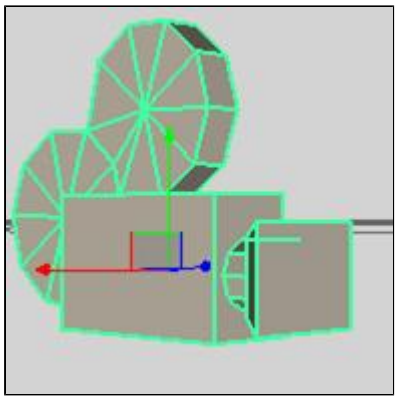
ℹ The Sky Texture object has been removed from HDK 1.3 onwards. For more information on how to create sky animations, see Sky Design.

Game Elements



The following game elements are available:

Object	Description	Samples	Related Documentation
Default Furniture Point	Sets a point in personal spaces and clubhouses where furniture will appear when a user places the furniture. Several of these can be added to your personal space and the one closest to the user when they bring up the Furniture menu will be used to spawn the furniture item.	N/A	N/A
Seat	Marks a feature of the scene as a place where avatars can sit, and provides a character reference in the seated position to ensure geometry is realistic (must be placed within a parent seat area). The seated figure in the Design View matches where the user will sit in your environment. You can drag the seat around until it matches your seat geometry. ⚠ A seat object can only be placed in a seat area object. You can have multiple seat objects within one seat area object.	HTS Aeronautilus Tools > Gymnasium	Collision in PS Home Environments Seating and Collision Adding Objects to a Scene Environments and Scale Dimensions General Design Guidelines
Seat Area	The parent node for each collection of seats. Add seat objects to set positions where avatars can sit.	HTS Aeronautilus Tools > Gymnasium	Seating and Collision Adding Objects to a Scene

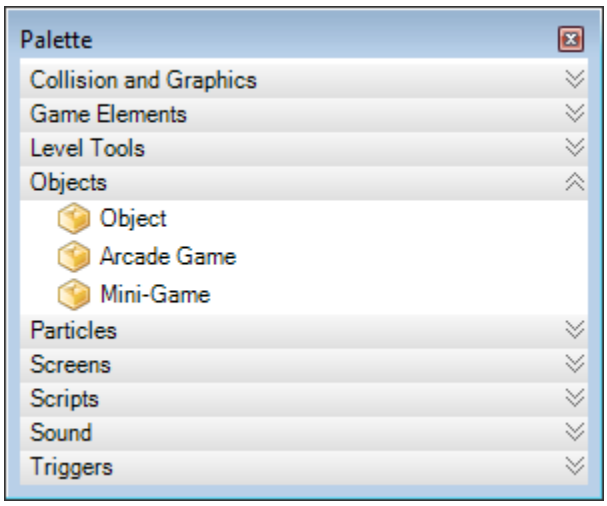
Picture Hook	Adds a point on which a picture frame can be hung in a personal space. It consists of 2 components, the hook position and a camera:  The hook position is shown by a dummy picture frame in the Design View. Move this to where you want users to be able to hang pictures:  The camera represents where the game camera will switch to when you are hanging a frame on the hook in-game: 	N/A	Picture Frames and Wall Decorations
Spawn Point	The point or area into which avatars are spawned when entering a space. You can have multiple spawn points in one space and you can set the origin space name so that avatars from particular spaces spawn to specific points. You can specify a specific spawn point from the Navigator by adding an attribute to the Navigator file.	All of the samples that have a .scene/.scproj file (e.g. HTS Aeronautilus, Arcade Tutorial, etc).	Dynamic Spawning Adding Objects to a Scene Deploying an Arcade Game in a Scene



The following level tools are available:

Object	Description	Samples	Related Documentation
Game Object Group	Creates a parent folder under which game objects can be grouped (child instances will inherit parent group properties). This is an easy way to organize game objects. You can also name each game object group. For example, if you had several mini-games in one space, you could place all of them in the a game object group called 'Mini-games'. Any properties that you set for the game object group will also be set for the child objects in it. For example, if Locked is set to True in the game object group, every object in it will also be locked.	N/A	N/A

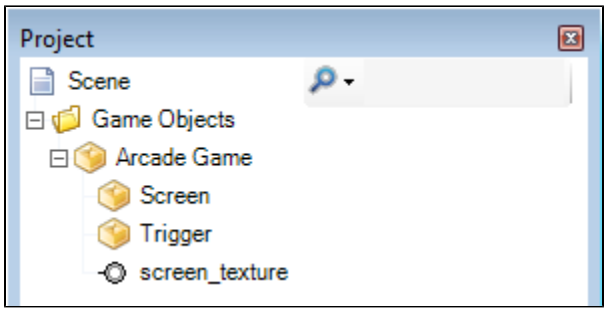
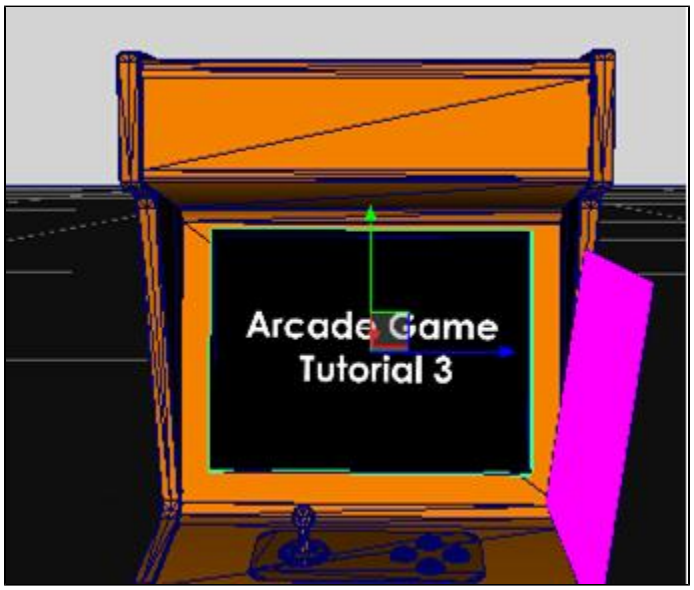
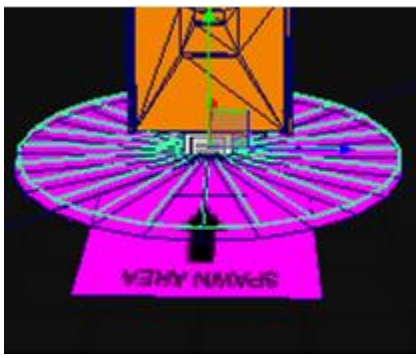
Objects



The following objects are available:

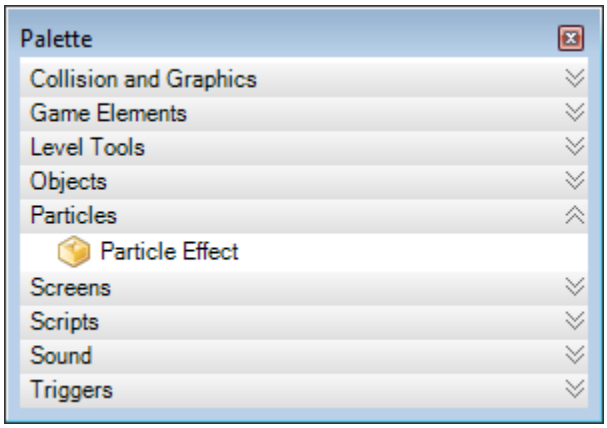
Object	Description	Samples	Related Documentation

Object	Enables you to add any object that has a Furniture component or Scene Object component, while maintaining the full functionality of the object. For example, you can add a chair, mini-game, realtime game, or arcade game. This allows you to have multiple instances of a single object, and if they have a Lua component, you can have instances of each object. For example, if you have a poker mini-game, you can have three instances of that single Mini-game: one with a 5 pound ante, one with a 10 pound ante and one with a 20 pound ante. You can specify that objects download with a scene when a user downloads the scene (as opposed to downloading when the user enters the scene by default. See Downloading Objects with Scenes.  Scene objects are not allowed in personal spaces or clubhouses.  Only objects with a Furniture or Scene Object component can be added as an embedded object. The Furniture component is automatically added to objects exported as furniture in Maya. The Scene Object component needs to be added to objects from the Object Editor.	N/A	PS Home Spaces Embed Objects into Scenes Object Component Properties.

Arcade Game	Marks a screen as an arcade game, references the arcade game script and creates a trigger area that will prompt the user to play the arcade game when the avatar is in that area. You can also define the image displayed on the arcade game screen when not in use. This object has three daughter components, Screen, Trigger, and screen_texture:  The screen component moves the screen on which the arcade game appears:  The screen component of an arcade game has no special properties. The trigger component moves and alters the trigger radius for the arcade game:  The trigger component of an arcade game has no special properties. The screen_texture component adds a texture (.dds) that appears when the arcade game is not being played. The screen_texture component of an arcade game has no properties.	N/A	Arcade Games Deploying an Arcade Game in a Scene Embed Objects into Scenes Object Component Properties

Mini-Game	Marks an area as a mini-game. It references the mini-game script and creates a customizable trigger area that prompts the user to participate in the mini-game when the avatar is in that area. ⚠ Adding a mini-game through the MiniGame component does not allow you to set instance parameters. You must add the mini-game as an embedded object to gain this functionality. This is also the preferred method.	Games > Simon Games > Tic Tac Toe HTS Aeronautilus > Telegraph Game Tools > Gymnasium > ExampleRoomPopUp	Active Items Deploying an Arcade Game in Furniture Embed Objects into Scenes Mini-games Object Component Properties Scripted Entity Animation Repertoire Editor

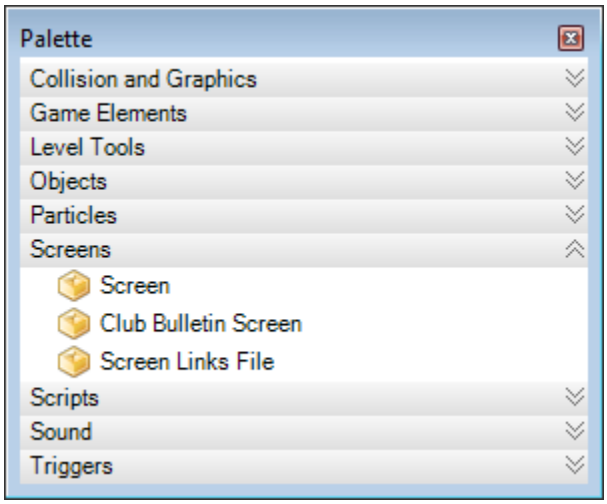
Particles



The following particles are available:

Object	Description	Samples	Related Documentation
Particle Effect	Allows you to add particle effects created in the ATG Particle Effects Tool (installed with the HDK). You must first add the particle effect as an asset to the scene.	HTS Aeronautilus Screens Example	ATG Particle Effects Tool Creating Particle Locators

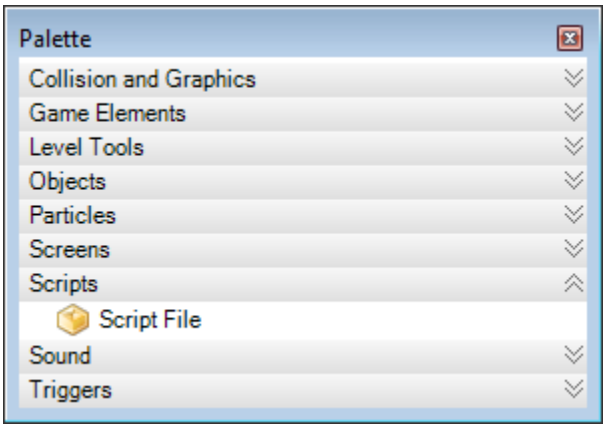
Screens



The following screens are available:

Object	Description	Samples	Related Documentation
Screen	Adds a planar screen to your scene. You can adjust the size of the screen using the manipulators (scale manipulator, Move All Points). The properties allow you to change the sound and define what content from the screen links file the screen will show or play.	Screens Example	Creating Screens
Club Bulletin Screen	The clubhouse bulletin screen allows clubhouse users to post messages, four of which are displayed on the club bulletin screen with the remainder available by scrolling through the old messages. It is only available for scenes that are packaged as clubhouses. The properties for the club bulletin screen determine how the screen is set up.	N/A	PS Home Spaces
Screen Links File	References an XML file that links the screen area in the scene to the content that will be displayed on the screen. ⚠ You can only have one screen links file per scene.	Screens Example	Creating Screens

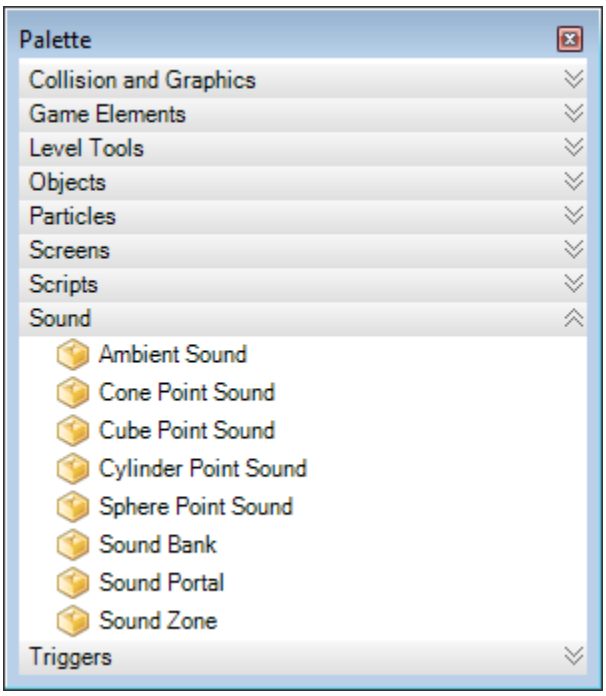
Scripts



The following script objects are available:

Object	Description	Samples	Related Documentation
Script File	This is the scene script. This script starts running as soon as the user enters the scene. You must first add the .lua or .luac script as an asset, before you can attach it to the Script File object. ⛔ Scenes must contain only one scene script file. ✅ Use scene objects with scripts instead of scene scripts. Scene objects are easier to manage, reuse, and update. To run more than one script in a scene, you must use scene object scripts. You can run other scripts referenced in your scene script file, but you cannot have more than one scene script file, because only the last one will run. Scene script startup occurs when the scene loads, so if there are many script processes, the scene takes some time to load. To avoid slow loading, divide the script and use separate scripts in embedded objects.	Screens Example HTS Aeronautilus Rewards Tutorial	Scene Scripting

Sound

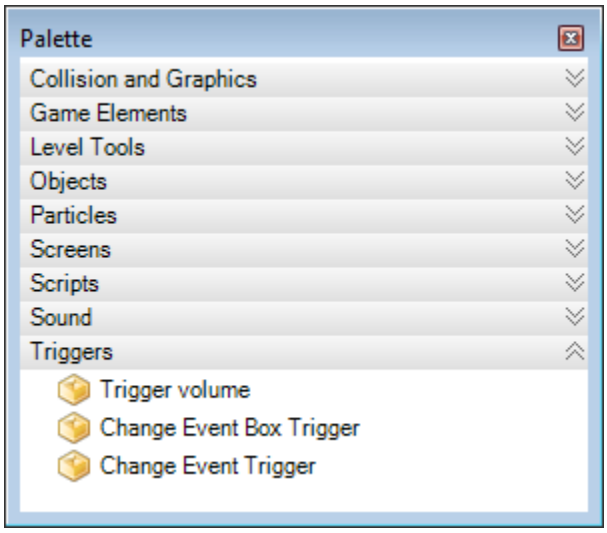


The following sound objects are available:

Object	Description	Samples	Related Documentation
Ambient Sound	This is a single big sound zone per scene that emits omnipresent ambient sound at a constant volume.	HTS Aeronautilus Screens Example	Working with Sound Objects
Cone Point Sound	Emits sound in a cone shape in a specific location (with fall off zone).	HTS Aeronautilus Screens Example	Working with Sound Objects
Cube Point Sound	Emits sound in a cube shape in a specific location (with fall off zone).	HTS Aeronautilus Screens Example	Working with Sound Objects
Cylinder Point Sound	Emits sound in a cylinder shape in a specific location (with fall off zone).	HTS Aeronautilus Screens Example	Working with Sound Objects
Sphere Point Sound	Emits sound in a sphere shape in a specific location (with fall off zone).	HTS Aeronautilus Screens Example	Working with Sound Objects
Sound Bank	A collection of sounds created in the SCREAM tool and added to the scene as a .bnk asset. It has no unique properties.	HTS Aeronautilus Screens Example	Working with Sound Objects

Sound Portal	Place it in a specific location in a scene to provide a smooth transition between two Sound Zones. It has no unique properties.	HTS Aeronautilus Screens Example	Working with Sound Objects
Sound Zone	Zones that inherit ambient sound, and can contain specific point sounds as well as specify reverb settings.	HTS Aeronautilus Screens Example	Working with Sound Objects

Triggers



The following trigger objects are available:

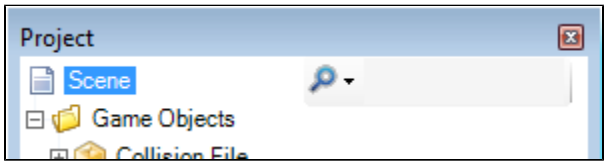
Object	Description	Samples	Related Documentation
Trigger Volume	Creates a volume which triggers a callback in lua whenever it is entered/ exited by a player, entity or item of furniture.	N/A	N/A
Change Event Box Trigger	Creates a cube event trigger area that changes the conditions of the environment for a user (e.g. exit the space and enter an adjacent space).	N/A	Dynamic Spawning
Change Event Trigger	Creates a flat event trigger area that changes the conditions of the environment for a user (e.g. trigger an animated model).	N/A	Dynamic Spawning

Scene, Object and Asset Properties

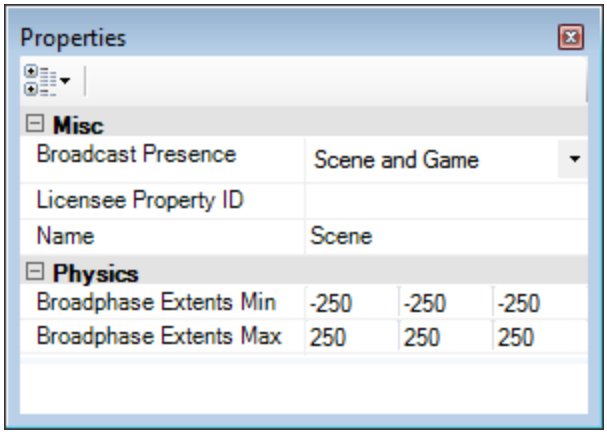
For Japanese Live Page
最新の日本語版ページはこちら 「シーン、オブジェクト、アセットのプロパティ」

Game/Scene Properties

The properties for the scene as a whole become available when you select Scene at the top of the Project panel:



Depending on the scene that you are editing, the project may be named Game rather than Scene.

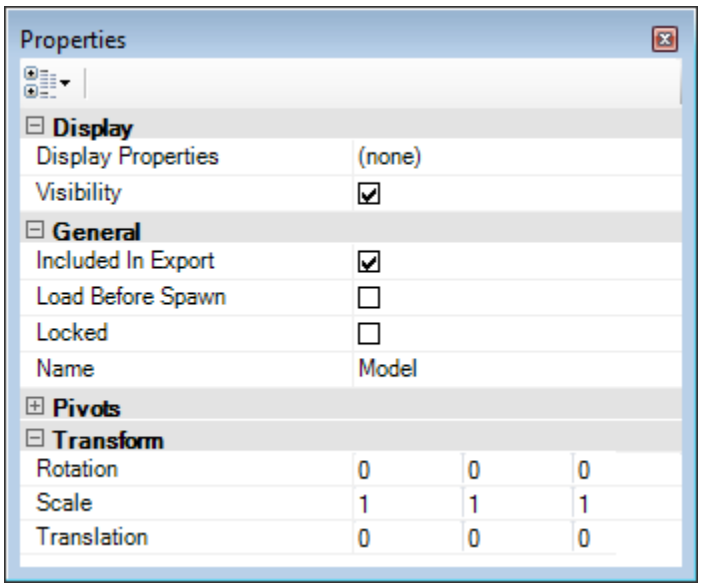


The following properties are available:

Property	Description
Broadcast Presence	The level of presence to broadcast to the XMB for the player's friends. This can be set to: • None - presence is not broadcast • Scene - presence within the scene is broadcast • Scene and Game - presence within the scene and any game in the scene is broadcast (Note: the Is Allowed Presence Broadcast flag in the game object's Mini Game component must also be set to true)
Licensee Property ID	A custom ID used to restrict the clothing which can be worn in the scene.
Name	The name of the scene/game.
Broadphase Extents Min	The point defining the minimum extent of the Havok broadphase.
Broadphase Extents Max	The point defining the maximum extent of the Havok broadphase.

Game Object Properties

Common Object Properties

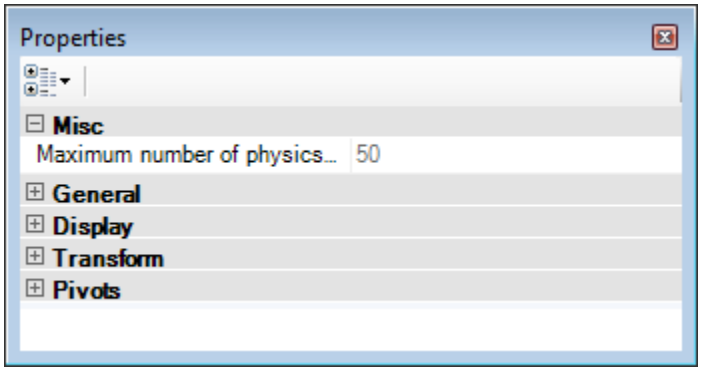


The following properties are common to all objects in the Scene Editor

Property	Description

Name	The name of the object.
Load Before Spawn	If this box is checked, the object needs to load in the scene before the user spawns. ⚠ Use this with caution. If large assets or many assets have this property in a scene, or if the user's connection is slow, it increases the amount of time it takes for a user to relocate to the scene.
Locked	Check this box to lock the object from edits.
Included In Export	Uncheck this box to stop the object from being exported when you package the scene. This allows you to have objects which are purely for reference and won't be included in the final scene. Objects which will not be included in the exported scene are shown in grey italics in the Project panel.
Visibility	Check this box to make the object visible in the Design View. Uncheck this box to make the object invisible.
Display Properties	Displays a small dialog where you can define three values: • Box: Check this box to make a box appear around the object. • Caption: Check this box to make a caption showing the name of the object appear. • Pivot: Check this box to show the pivot on the object.
Translation	Apply transforms on collision geometry before exporting to the Scene Editor. Translation, Rotation and Scale properties within the Scene Editor have no affect on the collision mesh.
Rotation	Apply transforms on collision geometry before exporting to the Scene Editor. Translation, Rotation and Scale properties within the Scene Editor have no affect on the collision mesh.
Scale	Apply transforms on collision geometry before exporting to the Scene Editor. Translation, Rotation and Scale properties within the Scene Editor have no affect on the collision mesh.

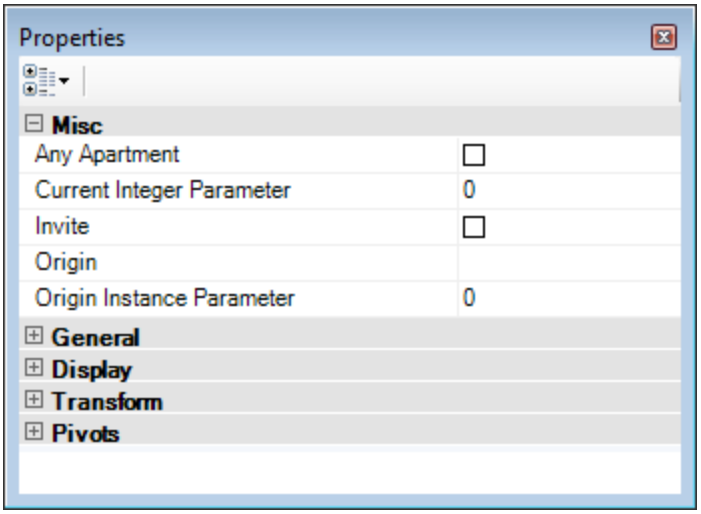
Collision File Properties



The following additional properties are available with collision file objects:

Property	Description
Maximum number of physics objects	Indicates how many items users can place in a personal space or clubhouse. (It has no effect on public spaces). This value currently cannot be edited, and is set to 50. ⚠ If you want to disable users placing objects in their personal space (i.e. set the number to 0), please discuss this with your Home Account Manager beforehand.

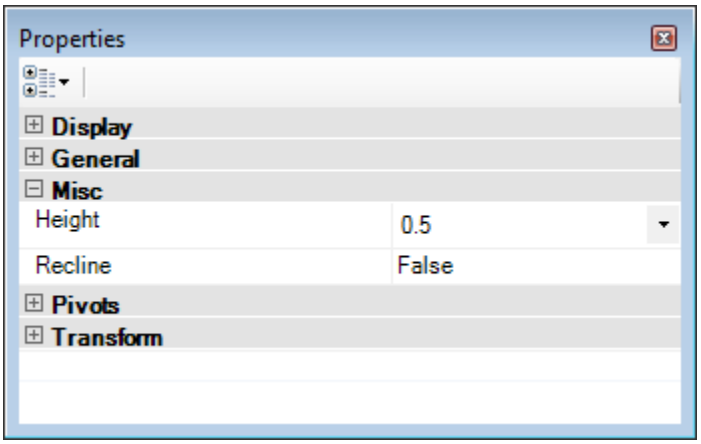
Spawn Point Properties



The following additional properties are available for spawn points:

Property	Description
Any Apartment	When set will cause any user coming from a personal space to spawn at this point.
Current Integer Parameter	Defines where a user enters in a scene. See [HDKdraft:Dynamic Spawning] for further details.
Invite	When set will cause any user which was invited to the space to spawn at this point.
Origin	You can choose to have users spawn at this spawn point if they are coming from a specific scene. Enter the scene name here.
Origin Instance Parameter	Defines which instance of a scene you enter when you pass through that spawn point. This enables you to display different content in geometrically matching scenes. See [HDKdraft:Dynamic Spawning] for further details.

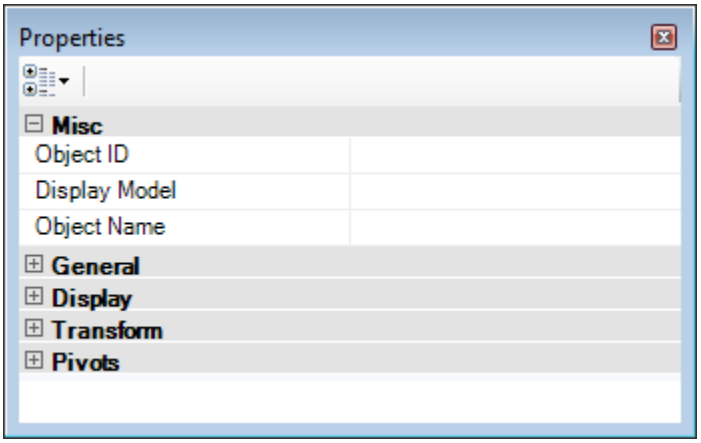
Seat Properties



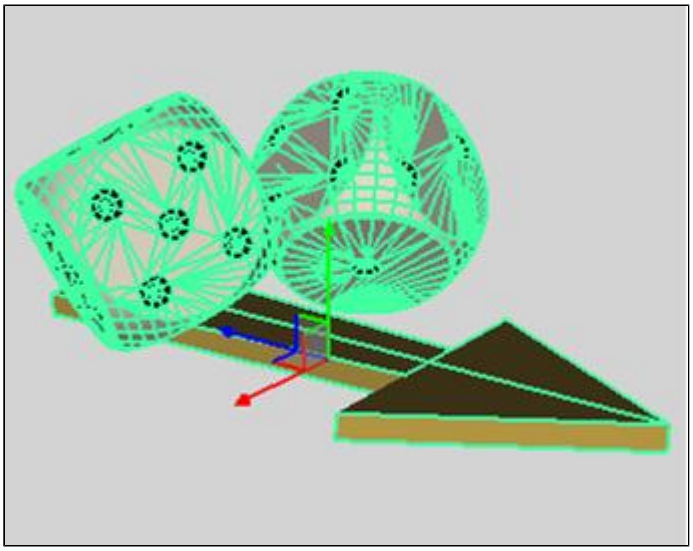
The following additional properties are available for seats:

Property	Description
Height	The height of the seat. This can be either: • 0.35: Low • 0.5: Mid • 0.75: High
Recline	Whether the player can lay back in the seat. This option is not available for high seats.

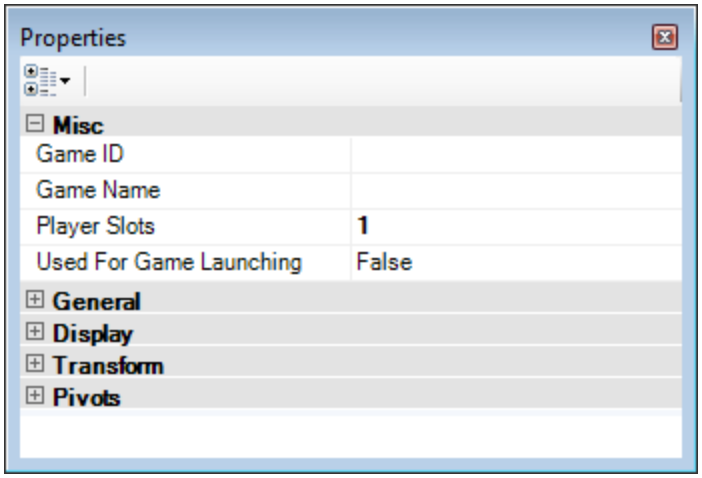
Object Properties



The following additional properties are available with objects:

Property	Description
Object ID	Allows you to select valid objects from your <HDK_INSTALL_DIR>\build folder.
Display Model	Allows you to select the model (.mdl) resource to show. You can choose either the default .mdl, or the object's .mdl. The following image shows the default .mdl: 
Object Name	When you select an object through the Object ID property, the object's name appears here.

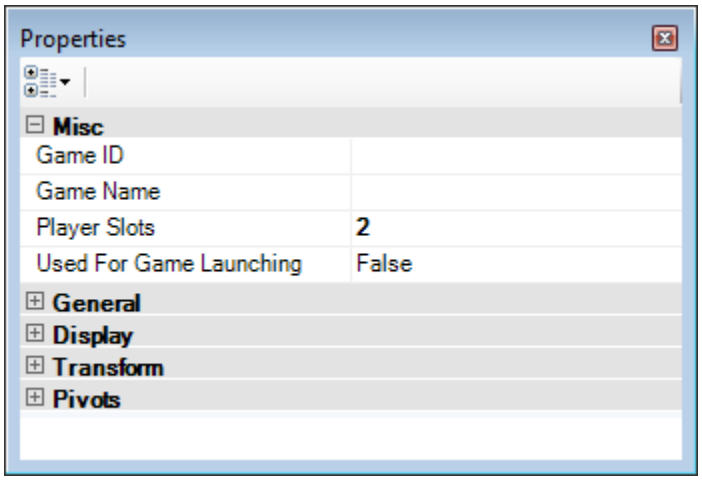
Arcade Games Properties



The following additional properties are available with arcade game objects:

Property	Description
Game ID	The object ID for the arcade game. Allows you to choose from all of the available arcade games in your <HDK_INSTALL_DIR>\build folder.
Game Name	The name of the arcade game. The name automatically appears when you select the Game ID.
Player Slots	Arcade games only support single-player mode. You must always set this property to 1.
Used For Game Launching	Allows you to conform to the rule that only one game launching session can be in progress at one time. When you select True, PS Home displays appropriate pop-ups to the user, informing them that they must cancel their current game launching instance before they can use the object.

Mini-Game Properties

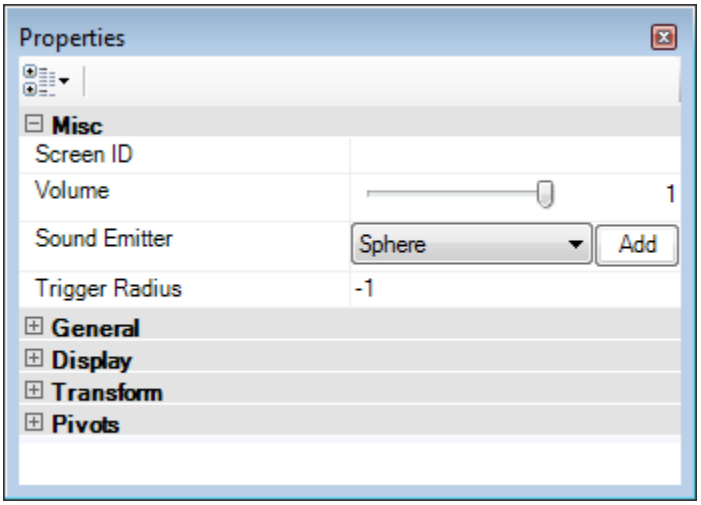


The following additional properties are available with mini-game objects:

Property	Description
Game ID	The object ID for the mini-game. Allows you to choose from all of the available mini-games in your <HDK_INSTALL_DIR>\build folder.
Game Name	The name of the mini-game. The name automatically appears when you select the Game ID.

Player Slots	The number of users who can join the mini-game at one time. This value is ignored for sessionless mini-games. For more information on sessionless mini-games, see Mini-games. ⚠ If the session size is defined for the mini-game, it must match the Player Slots value. You set session size within the XML of the Network component and add it using Object Editor. For more information, see Active Items.
Used For Game Launching	Allows you to conform to the rule that only one game launching session can be in progress at one time. When you select True, PS Home displays appropriate pop-ups to the user, informing them that they must cancel their current game launching instance before they can use the object.

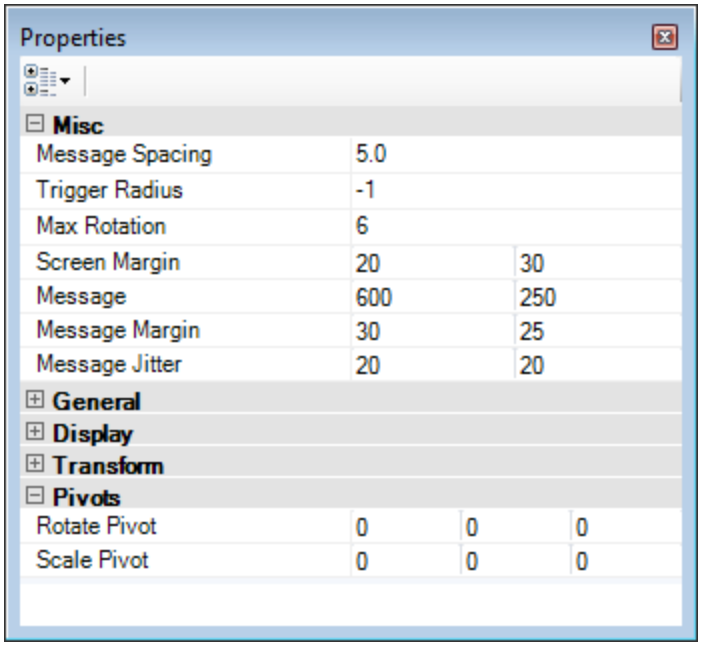
Screen Properties



The following additional properties are available with screen objects:

Property	Description
Screen ID	The ID of the screen. This ID is listed in a screen links file.
Volume	A slider that controls the sound's volume. Drag the slider, or click on the number to its right and enter a value between 0 and 1.
Sound Emitter	Adds additional sound emitters to the screen. A default sphere emitter will be used if no emitters are added.
Trigger Radius	The radius of the trigger that activates the screen at which users will see the target prompt.

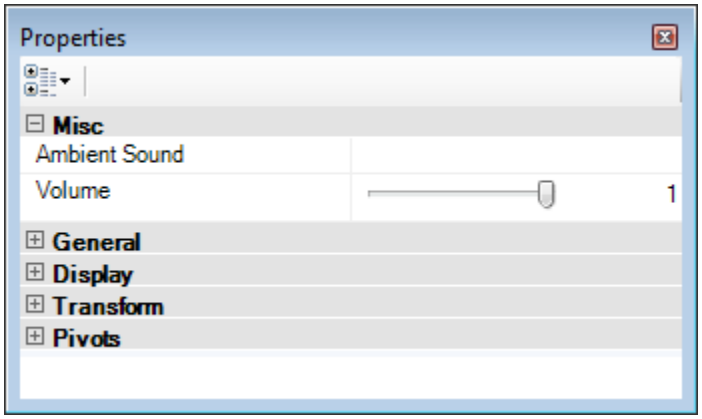
Club Bulletin Properties



The following additional properties are available with club bulletin objects:

Property	Description
Message Spacing	The space between each message (in pixels).
Trigger Radius	The radius of the trigger that activates the club bulletin screen interaction. For more information about the targeting system in PS Home, see Screens.
Max Rotation	Rotates the messages around the pivot point. Each message is rotated by a different random amount determined by the value entered (in degrees). For example, if you enter 20, each message is rotated by a random value between -20 and 20 degrees.
Screen Margin	The space between the screen margin and the message margin (in pixels).
Message	The width and height of the messages (in pixels).
Message Margin	The space between the message and the screen margin (in pixels).
Message Jitter	Moves the pivot point for each message along the Y and X axis. Max Rotation is applied to each message using this pivot point.

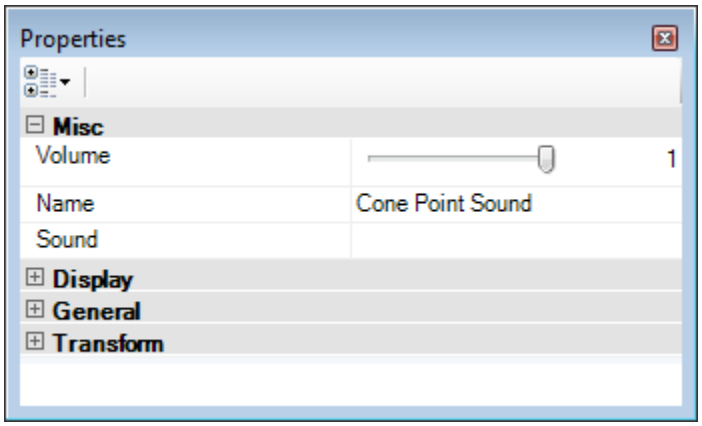
Ambient Sound Properties



The following additional properties are available with ambient sound objects:

Property	Description
Ambient Sound	The sound asset. To add it, drag the sound file from the Sound Bank into this field or enter the audio name as it appears in the Sound Bank. This field is blank by default.
Volume	A slider that controls the sound's volume. Drag the slider, or click on the number to its right and enter a value between 0 and 1.

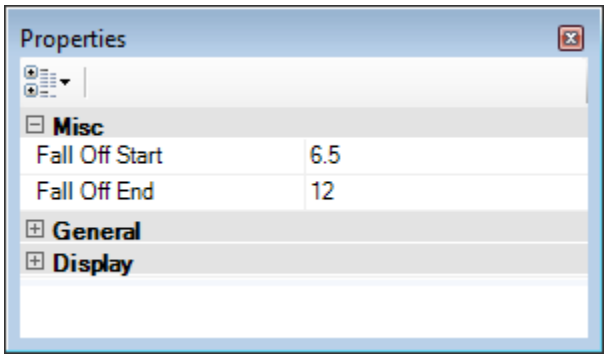
Point Sound Properties



The following additional properties are available with point sound objects:

Property	Description
Sound	The sound asset. To add it, drag the sound file from the Sound Bank into this field or enter the audio name as it appears in the Sound Bank. This field is blank by default.
Name	The name of the point sound object. This value cannot be changed.
Volume	A slider that controls the sound's volume. Drag the slider, or click on the number to its right and enter a value between 0 and 1.

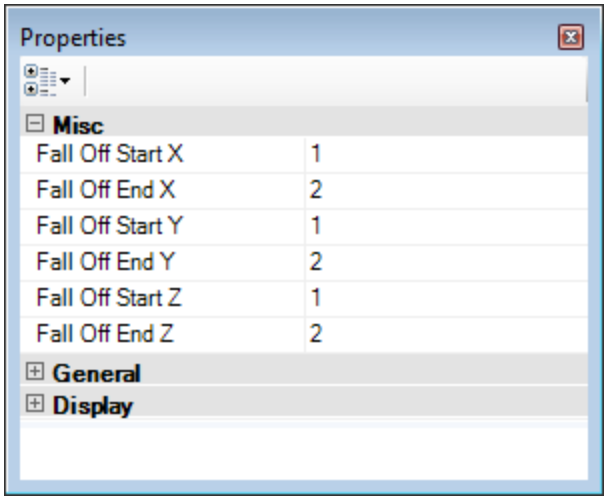
Sphere Point Sound Emitter Properties



The following additional properties are available with sphere point sound emitter objects:

The sound zone work area represents the position of the sound, followed by the Fall Off areas. The emitter fall off start and end distances can be adjusted by entering new values in Fall Off Start and Fall Off End.

Cube Point Sound Emitter Properties



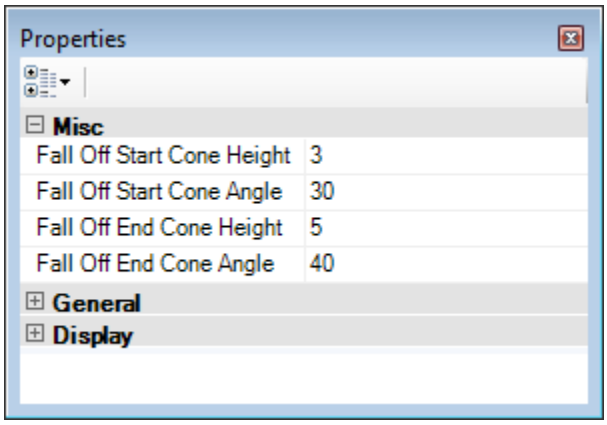
The following additional properties are available with cube point sound emitter objects:

The sound zone work area represents the position of the sound, followed by the Fall Off areas. You can adjust the emitter fall off start and end distances by entering new values in Fall Off Start and Fall Off End.

The Cube has a Fall Off Start and Fall Off End parameter for each axis.

> ⚠ Cube emitters are currently very expensive to process, and having more than 2 in a scene may result in framerate and audio quality issues.

Cone Point Sound Emitter Properties



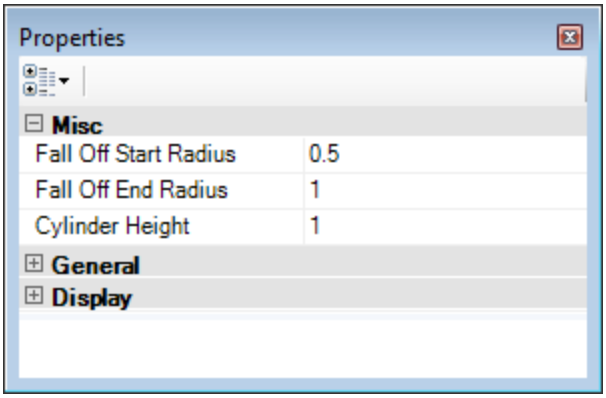
The following additional properties are available with cone point sound emitter objects:

The Name, Visibility and Locked properties are described in Ambient Sound Properties.

The shape represents the position of the sound, followed by the Fall Off areas. You can adjust the emitter fall off start and end distances by entering new values in Fall Off Start and Fall Off End.

The Cone has a Fall Off Start and Fall Off End parameter for the Cone Height and the Cone Angle. The angle setting is in degrees.

Cylinder Point Sound Emitter Properties



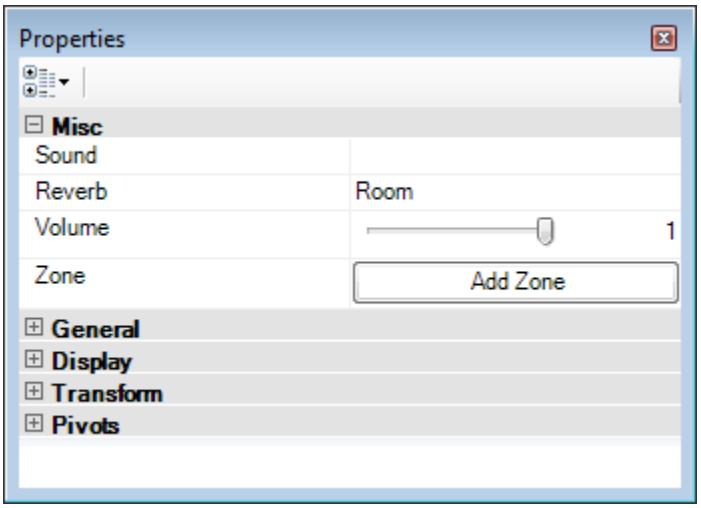
The following additional properties are available with cylinder point sound emitter objects:

The Name, Visibility and Locked properties are described in Ambient Sound Properties.

The shape represents the position of the sound, followed by the Fall Off areas. You can adjust the emitter fall off start and end distances by entering new values in Fall Off Start and Fall Off End.

The Cylinder has a Fall Off Start and Fall Off End parameter for the Radius of the cylinder. You can also configure the height.

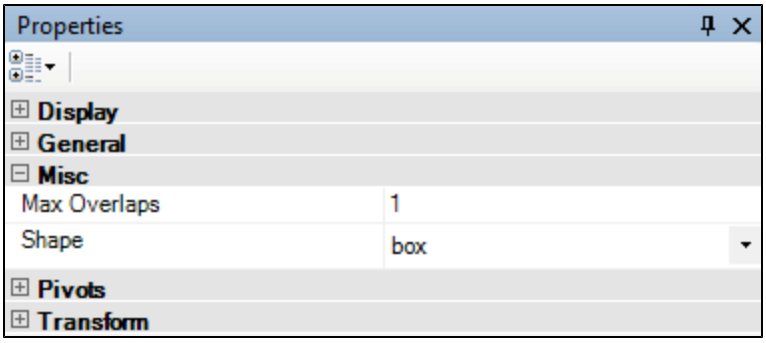
Sound Zone Properties



The following additional properties are available with point sound objects:

Property	Description
Sound	The sound asset. To add it, drag the sound file from the Sound Bank into this field or enter the audio name as it appears in the Sound Bank. This field is blank by default.
Reverb	Provides options to reflect the environment the sound zone is modeling: • Room • Hall • Outside
Volume	A slider that controls the sound's volume. Drag the slider, or click on the number to its right and enter a value between 0 and 1.
Zone	A button that allows you add child instances of the sound zone. All zones within the parent sound zone inherit the parent properties. You can manually adjust the child zones' properties (except for the sound asset).

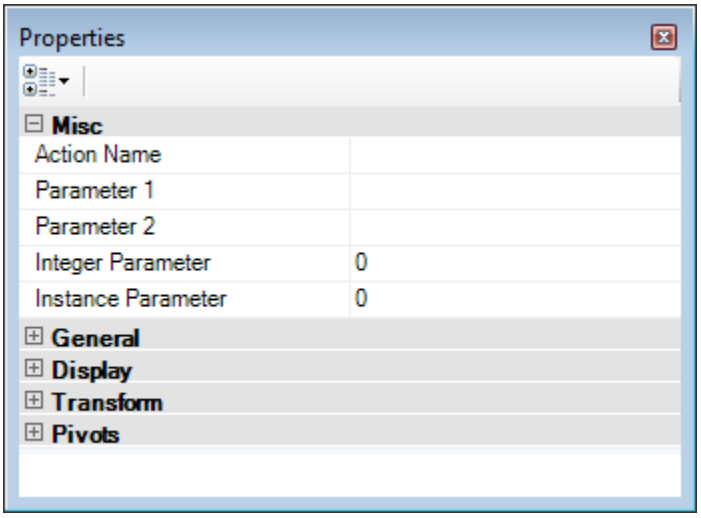
Trigger Volume Properties



The following additional properties are available with trigger volume objects:

Property	Description
Max Overlaps	The maximum number of simultaneous overlaps which can be detected. If more players, entities or furniture than this are currently within the trigger volume the lua callback will not be triggered.
Shape	The shape of the trigger volume. This can be either: • Box • Sphere • Cylinder

Trigger Properties

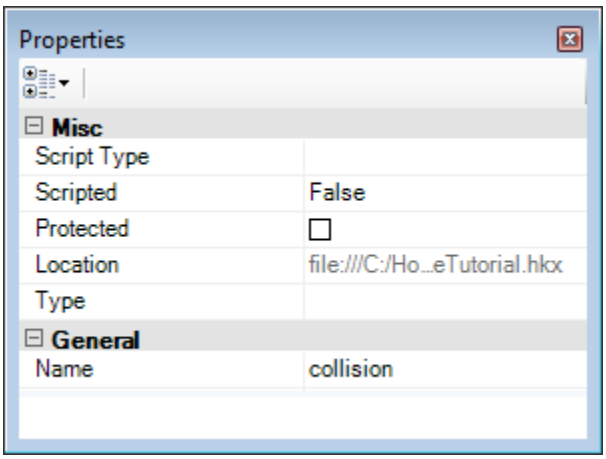


The following additional properties are available with trigger objects:

Property	Description
Action Name	Choose from one of three events: • ChangeLobby: This option makes avatars automatically spawn at the Instance Parameter specified when they enter the trigger radius. • ChangeSceneMessage: This option asks the avatar to confirm that they want to enter the Instance Parameter specified when they enter the trigger radius. • CommercePoint: This opens a commerce point (for more information, see Creating a Commerce Point).
Parameter 1	When you select ChangeLobby or ChangeSceneMessage from the Action Name drop-down list, enter the name of the scene that you want to transition to here. When you select CommercePoint from the Action Name drop-down list, enter the path of the Commerce Point XML file here.
Parameter 2	Not used.

Integer Parameter	Allows you to choose where in a scene a user will enter. This is related to, but different from a regular spawn point.
Instance Parameter	Defines what instance of a scene you will enter when you pass through that trigger point. This allows you to display different content in geometrically matching scenes.

Asset Properties



The following additional properties when you select an asset in the Project panel:

Property	Description
Script Type	Allows you to select the type of resource, as used by the scripts.
Scripted	Determines whether the resource can be accessed by scripts. Select True to allow access or False to deny access.
Protected	Check this box if you want to protect the asset when it is packaged. This setting is required for content submitted for SCE QA. This box is checked by default for Lua scripts.
Location	The filename and path of the asset.
Type	The content type of the asset.
Name	The unique name of the asset.

Downloading Objects with Scenes

For Japanese Live Page
最新の日本語版ページはこちら 「シーンとオブジェクトのダウンロード」

By default, objects in a scene begin to download when the user enters the scene. Users cannot see or interact with an object (such as a mini-game) until the object has finished downloading. This means that users might have to wait several minutes after joining a scene before they can play in it. The delay is most noticeable for very large objects.

To improve the user experience, you can specify that when a user downloads a space, they also download certain objects. You can select up to 8 objects to pre-download.

Pre-Download Object Guidelines

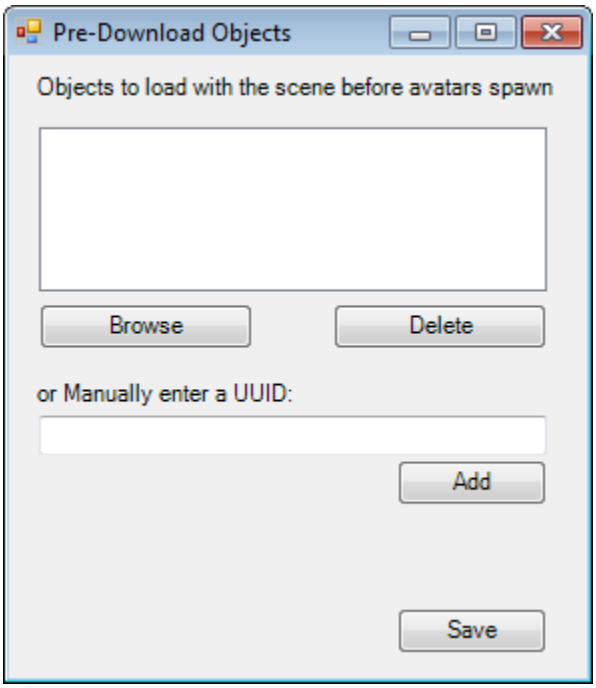
- The scene should not pre-download more than 100 MB of objects. If you want to pre-download more than 100 MB of objects, contact your regional SCE. (This restriction is primarily aimed at maintaining a balanced HDD caching behaviour. See notes below).
- It is not guaranteed that the objects will always be downloaded before the user is relocated to your scene. (PS Home cannot check whether the objects are up to date on every relocation, as it would add cost and delays for the user).
- You may also want to use the 'Load Before Spawn' feature for scene objects. Setting this flag extends the scene loading/relocation process but ensures that the specified objects are loaded (and downloaded if necessary) before the user is able to view or interact with the scene.

- Don't package pre-download objects with the scene.
- All pre-download objects must be published in the live environment before the scene is published.

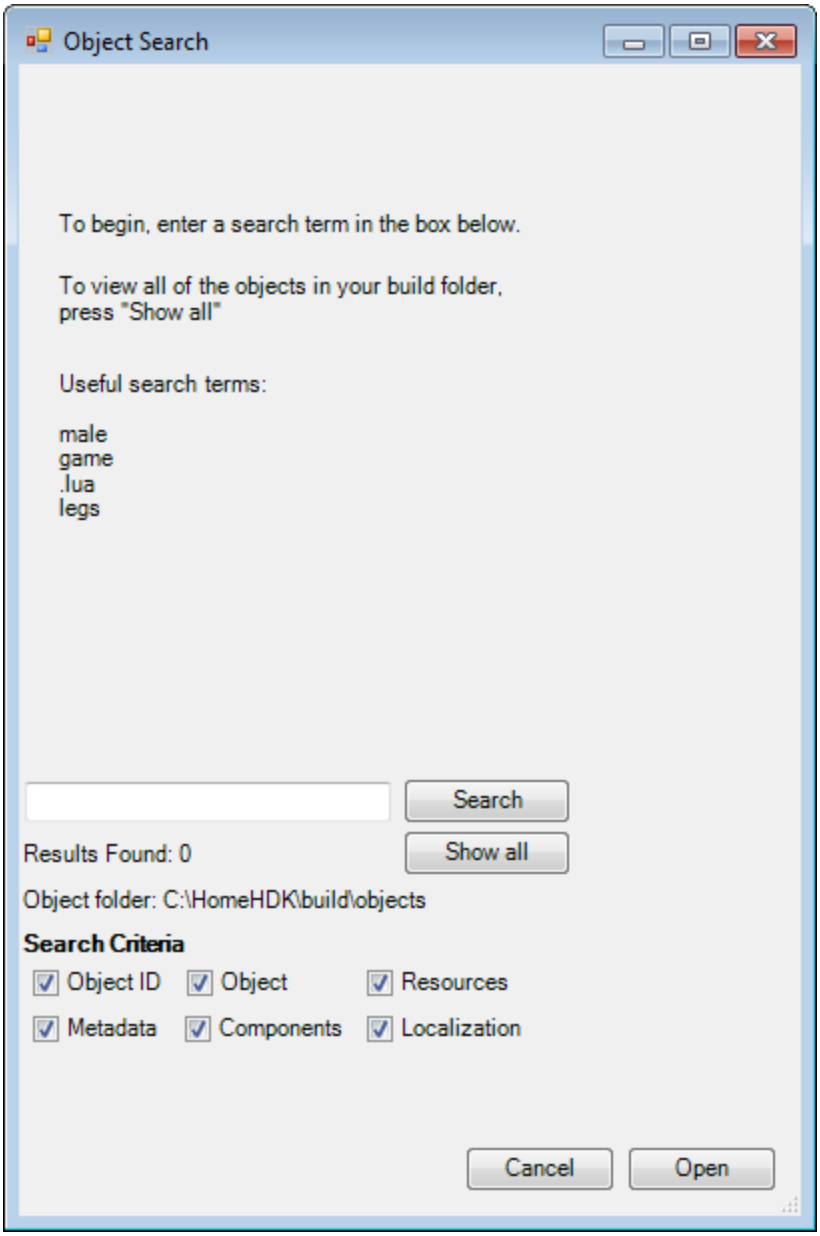
Setting up Pre-Download Objects

To select the objects for pre-downloading with a scene:

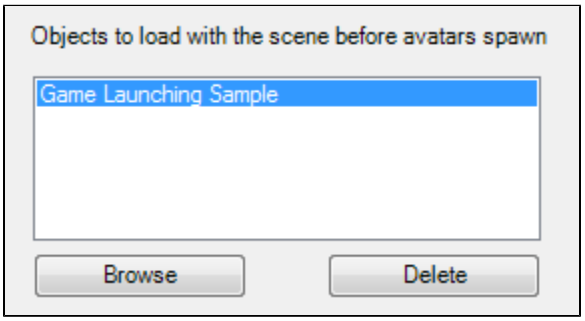
1. Open the scene in the Scene Editor.
2. Select Edit > Edit Pre-Download Objects List to display the Pre-Download Objects dialog:



3. To add objects to the list, do one of the following:
    - To search for the objects in your `build\objects` folder, click Browse to display the Object Search dialog:



Use the search criteria to find the objects you want to pre-download or click Show all to display all of the objects available in your `build\objects` folder. Once an object you want to add is displayed, select it and click Open. The selected object is listed in the Pre-Download Objects dialog:



- If the object is not in your build folder, enter the object's UUID in the field provided and click Add.

4. When you have finished adding objects, click Save.
5. Save your scene and package it as normal. For more information about how to package a scene, see Preparing Scenes for Packaging and Packaging Scenes.

Notes on Home HDD Cache Behaviour

PS Home uses a portion of the PlayStation®3 HDD for its cache. By default this is the minimum 3 GB, and only a portion of this cache is reserved for objects.

As users move between scenes and as remote users enter scenes, the user's cache fills up. Scenes that require a lot of assets (such as scenes that pre-download a lot of objects, or dynamically load Resource Packs), large objects in a scene, or objects that dynamically load large or many resources from Resource Packs can quickly consume the cache.

When the cache is full and PS Home needs to download more content, it pushes older content out to make room for the new downloads.

The more large objects the user has to download, the more often they will have to re-download other content, increasing the consumption of the user's bandwidth. (Some users may be on restricted connections and will not appreciate this).

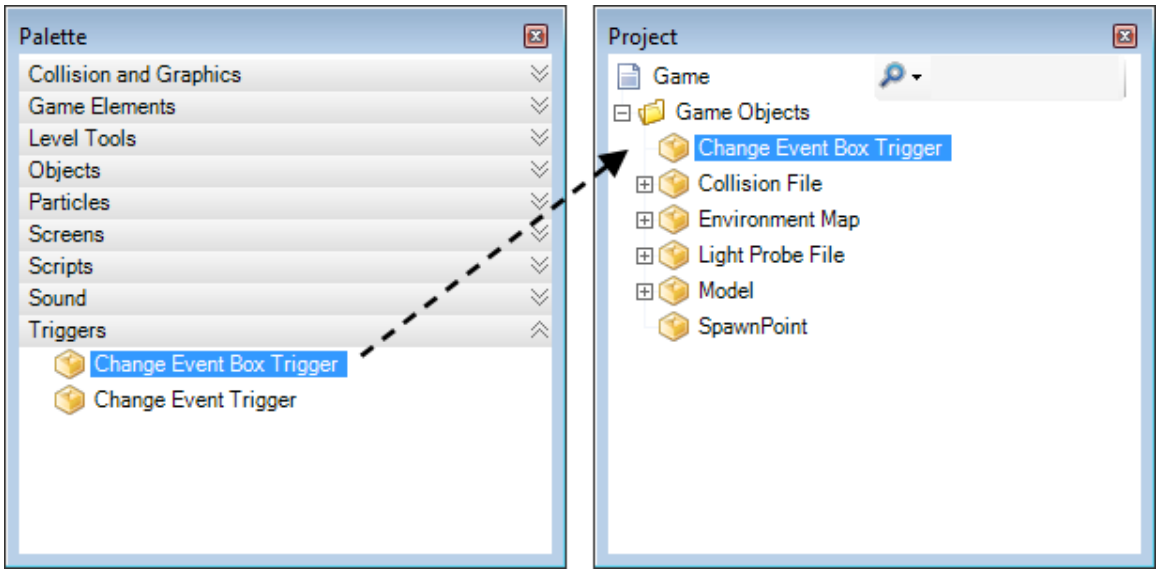
So it is beneficial to the user (and hence the developer) to keep scene and objects as small as possible.

Dynamic Spawning

> For Japanese Live Page
> 最新の日本語版ページはこちら 「動的スポーン」

You can define dynamic content and actions within a scene using the Change Event Triggers in the Scene Editor. The dynamic content and actions are based on how an avatar enters a scene, or from where the avatar enters the scene.

In the Scene Editor, add either the Change Event Box Trigger or the Change Event Trigger from the Palette panel to the Project Panel:



In the Properties panel, set the required scene parameters for dynamic spawning:

- Action Name
- Integer Parameter
- Instance Parameter

Action Name

This property is a drop-down list that provides the following options:

- ChangeLobby: This option makes avatars automatically spawn at the instance parameter specified when they enter the trigger radius.
- ChangeSceneMessage: This option asks the avatar to confirm that they want to enter the instance parameter specified when they enter the trigger radius.
- CommercePoint: This opens a commerce point (for more information, see Commerce).

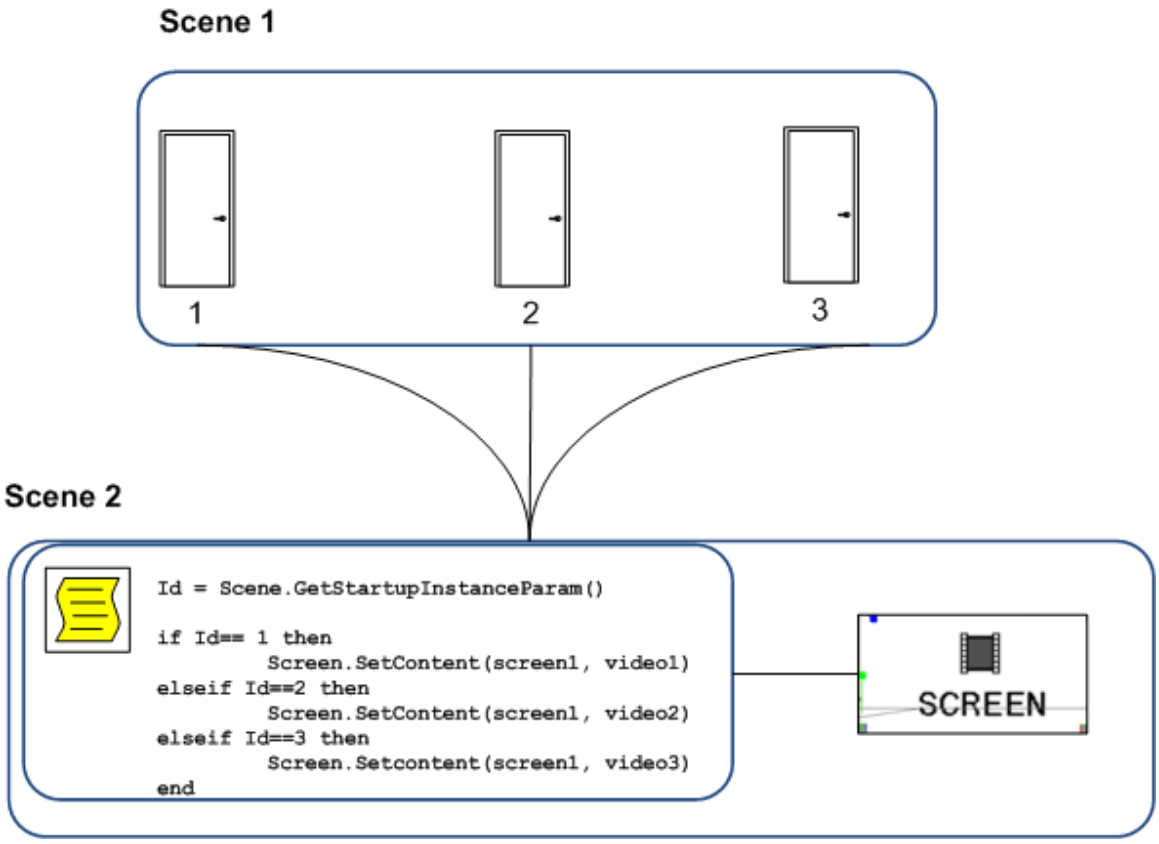
Instance Parameter

This property allows you to define which instance of a scene you enter when you pass through that trigger point. This allows you to display different content in geometrically matching scenes.

You can create between 1 and 65,535 different instances using instance parameters.

For example, in the Home Cinema you have ten auditoriums, each showing a different trailer. Instead of creating 10 identical scenes (with ten unique names and scene keys) you have just one. When you pass through a doorway into the first auditorium, the trigger sets the Instance Parameter to 1. When the scene loads, it checks the instance parameter via a script and plays the correct trailer. When you leave, the Home Cinema scene checks the 'Origin Instance Parameter' and spawns you at the appropriate point (that is, the one you left). It does the same for the second door, but sets the Instance Parameter to 2, and so on.

Because these are separate 'instances' of the same scene, users see only other users in their instance.



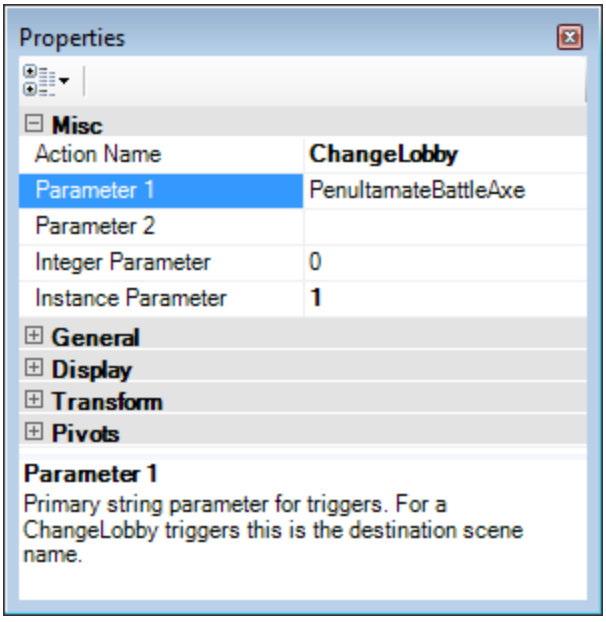
Setting the Instance Parameter

To set the Instance Parameter:

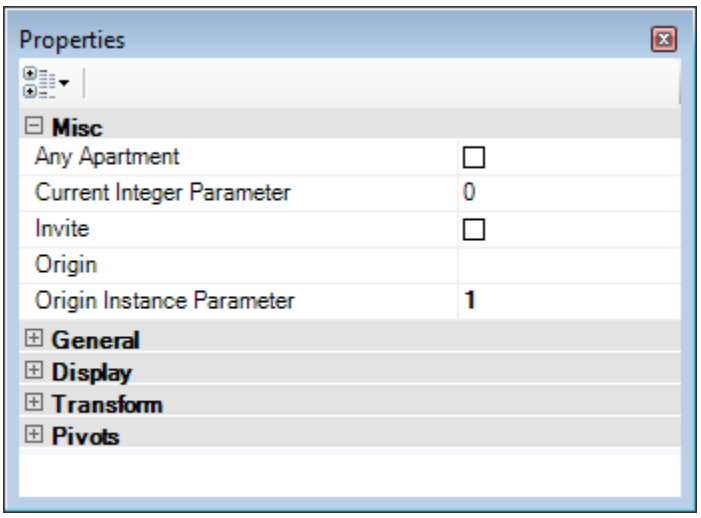
1. Add Change Event Trigger or Change Event Box Trigger to the hierarchy in the Project panel.
2. In the Properties panel, set the Instance Parameter property.

   In the Home Cinema example above, there are 10 auditoriums, so you would create 10 Change Event Triggers or Change Event Box Triggers, and number each Instance Parameter property from 1 to 10.

3. In Parameter 1, add the name of the scene that you want to transition to. (Enter the name as it appears in the scene list.)



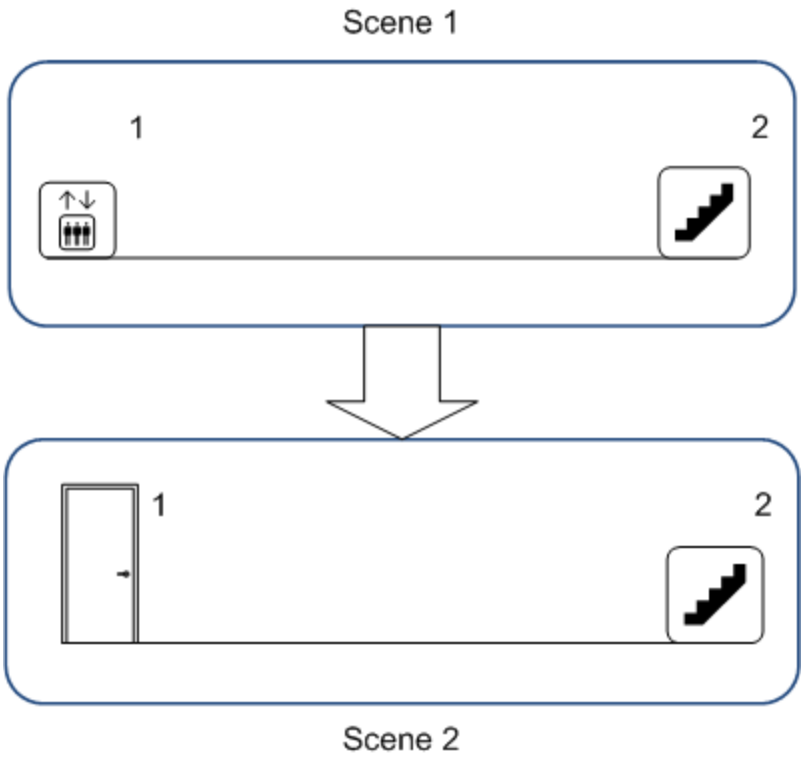
4. Add the LUA function `Scene.GetStartupInstanceParam()` to your script. For more information on how to use the function, see the Lua API Reference.
5. Click on the Spawn Point in the Project panel.
6. In the Properties panel for the Spawn Point, set the Origin Instance Parameter field to the same number as was set for the Instance Parameter field:



Integer Parameter

The integer parameter, like the instance parameter, provides a method to transfer a parameter between scenes. Unlike the Instance Parameter, the integer parameter does not affect the way new network instances are created - it is just a number that the script in the destination scene can query and react to.

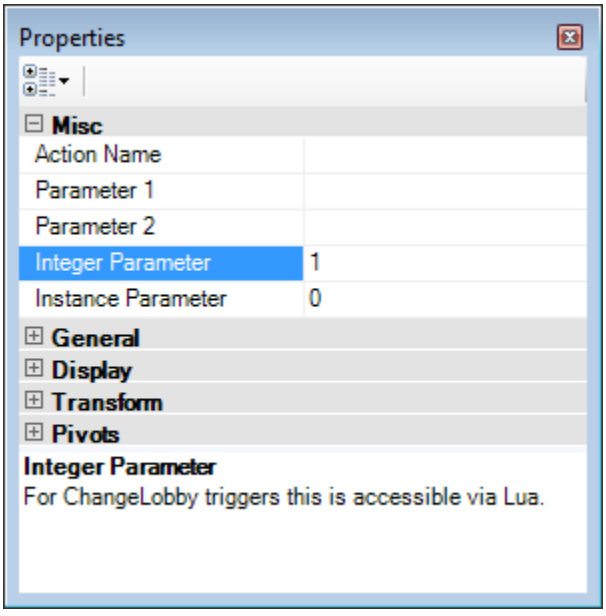
For example, you can use this property to select where in a scene a user enters :
You have two large scenes. One is 'upstairs' from the other, and the scenes have several entrances and exits which link the two. Exit 1 is an elevator, and Exit 2 is a staircase. If a user takes the elevator to go to the 'downstairs' scene, you can have them appear in front of the elevator in the downstairs scene when they enter the scene. To enable exiting the Upstairs Scene using Exit 1 (elevator), you set the Integer Parameter of the Upstairs Scene to 1 using the Integer Parameter property of the Change Event Trigger or the Change Event Box Trigger object. To enable entering the Downstairs Scene using a spawn point at entrance/exit 1 (elevator) you set the Current Integer Parameter property to 1.



Setting the Integer Parameter

1. With the first scene open, add a Change Event Trigger or Change Event Box Trigger object to the hierarchy in the Project panel.
2. In the Properties panel, set the Integer Parameter property.

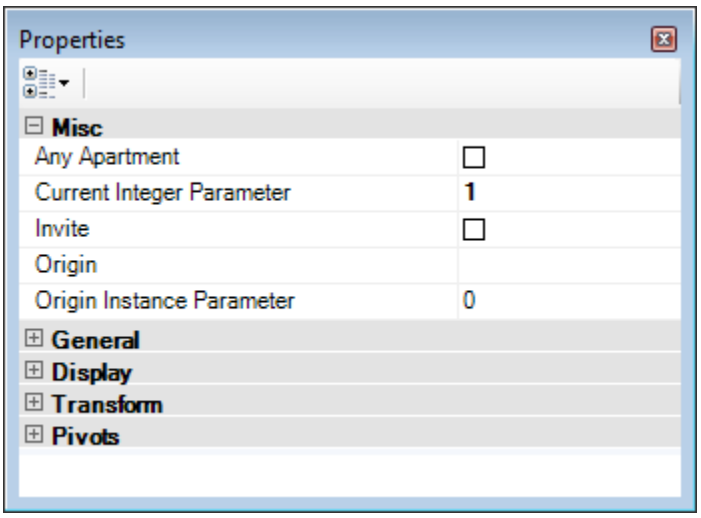
   In the example above, the upstairs elevator is given the parameter 1, and the upstairs staircase the parameter 2.



3. Open the second scene, and add a Spawn Point to the hierarchy in the Project panel.
4. In the Properties panel for the Spawn Point, set the Current Integer Parameter field to the same number as was set for the Integer Parameter field.

   In the example above, the Spawn Point at the elevator entrance in the downstairs scene has the Current Integer Parameter

property set to 1.



The important difference is that the Origin Instance Parameter property gets the scene instance parameter of the scene that you left. The Current Integer Parameter property gets the scene integer parameter of the scene you're currently in.

To get the Integer Parameter of the scene you're in, you need to use `Scene.GetstartupInstanceParam()` in your script.

Specifying the Spawn Point from the Navigator

You can query the actions using Spawn Points in the Scene Editor and through the Scene API in LUA.

You can specify a specific spawn point from the Navigator by adding an attribute to the Navigator file.

For example, if you add a parameter to the `LocalPlayer.Relocate()` function:

```
LocalPlayer.Relocate( string sceneName, number instanceParam = 0, number valueParam = 0, string
SpawnSource = 0 )
```

You can add an attribute to the Navigator configuration XML:

```
<DESTINATION SPAWNFROM="Marketplace">Home Square</DESTINATION>
```

In both cases, the additional string overrides the standard behavior and is treated as the previous location for spawn point selection. The spawn points in scenes contain a reference to the SceneIDName that the user came from. In the current example, even if the navigator is being used in the 'Game Space', the Home Square spawn point selected would be that of the marketplace. This means that the user would appear standing outside the entrance to the marketplace, rather than outside the bowling alley.

Relocating

See the Relocating to Instances page for information on relocating, including using the Relocate library for relocating to unique instances, group instances or using Instance parameters.

Targeting System

Local avatars can target:

- Other avatars (remote avatars)
- Screens (video or poster) except screens that map to environment geometry (unpredictable shapes can't be zoomed in on in a reliable or predictable way)
- Mini-games, Realtime games and arcade games
- Furniture, picture frames and actives
- Inventory items (portables) placed by themselves
- Trigger points in an environment such as seats, animated models (if they're set to be targetable), exits (if they're set to prompt confirmation to relocate)

Targeting in PS Home

When the local avatar approaches the object (or area) that can be targeted, a pop-up appears inviting them to interact.



The user has three choices:

1. Walk away and the pop-up disappears or press [DECLINE] to exit the pop-up.
2. Press [ACCEPT] to interact with the target.
3. Use the [LEFT] or [RIGHT] directional buttons to retarget. For example: If the user wishes to target an object that happens to have a group of other avatars around it, trying to target that object may initiate the profile of one of the nearby avatars instead. Pressing the [LEFT] and [RIGHT] directional buttons on the controller will cycle to the next targetable object (or avatar) and the correct one can then be chosen.

> Weighted system for Targeting from HDK v 1.3.5 From HDK v 1.3.5 onwards, the focus target is chosen by using the facing direction of the local avatar. The target most in-line with this facing direction is chosen, but this is weighted in favor of more important target objects, e.g. Mini-games have a higher weighting than remote avatars.

Trigger Radius (Proximity Radius) for Targeting

If something can be targeted, it has a proximity radius around it that determines at which distance from the target the local avatar will get the pop-up inviting them to interact.

In PS Home, a default trigger radius is set automatically. The trigger radius distance depends on the type of thing being targeted.

For instance, remote avatars may be targeted from 3.25m, Mini-games from 1m and posters from 5m. These values are sometimes changed in the system if it is found that typical interaction with something calls for a different default value than specified.

There are some exceptions to this - furniture and screens:

- Furniture: Furniture placement and targeting operates in a separate mode. Once furniture has been placed, the local avatar can retarget it from within a certain radius (an xyz position from the avatar which defaults to a certain distance, but which can be set in the individual objects if required). All furniture is targetable on an the xyz position, except lights and picture hooks (which ignore the avatar's y position). Another exception is seats, that cannot be targeted for sitting through collision. If you have a scene that requires furniture to be targetable irrespective of the avatar's y position, contact regional SCE to disable this.
- Screens: Local avatars can only target screens if they face them and if the screen is visible (even if they are within the proximity radius). Also, local avatars cannot target a screen if there is any collision geometry between the avatar and the screen. This is because a collision ray is fired from the local avatar's head to the center of the screen.

Setting the Trigger Position

You can retrieve and set a screen's trigger position by calling the functions `Screen.GetTriggerPosition()` and `Screen.SetTriggerPosition()`. By default, the trigger position will be at the screen's center.

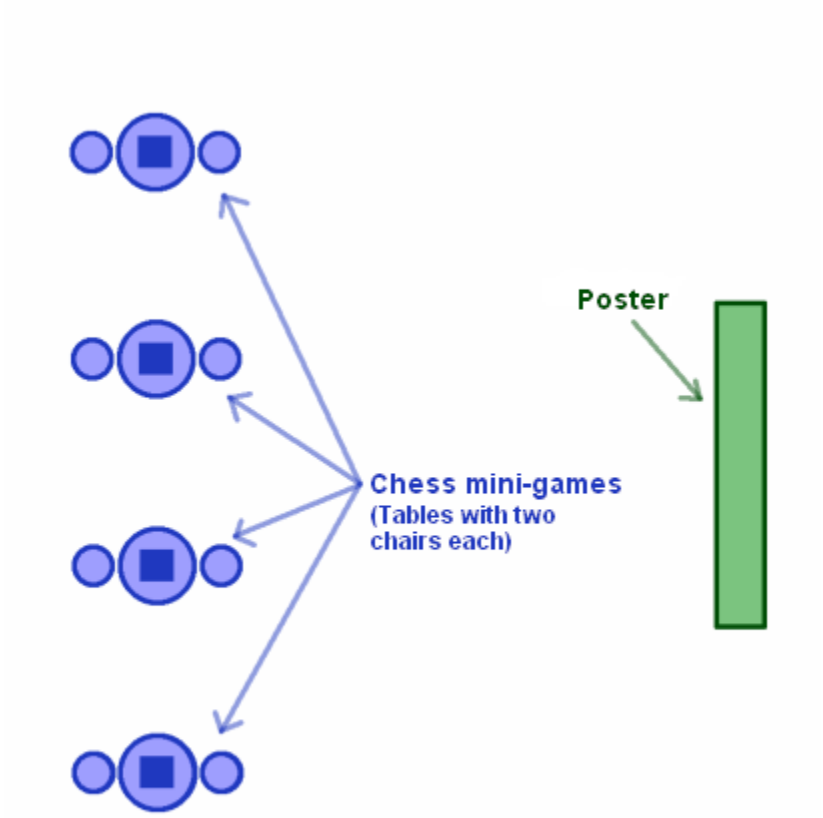
Setting the Trigger Radius

The general rule is that everything in PS Home has a specific default trigger radius, and everything can be targeted by the local avatar from the default distance.

There is a default proximity trigger radius for targeting things, set automatically in PS Home at different distances depending on the type of object. As a rule, the default will work in most situations, but you can also set the proximity radius manually. There are several things to keep in mind when you do this.

The proximity radius should be set proportionately. To make targeting seem realistic and intuitive, the proximity radius will depend on the type of thing being targeted. So if it is a chess game you wish to join or observe, the distance from the game will be greater than if you want to target a seat. The method for manually setting the proximity radius depends on the type of object as well, but mostly it is done automatically.

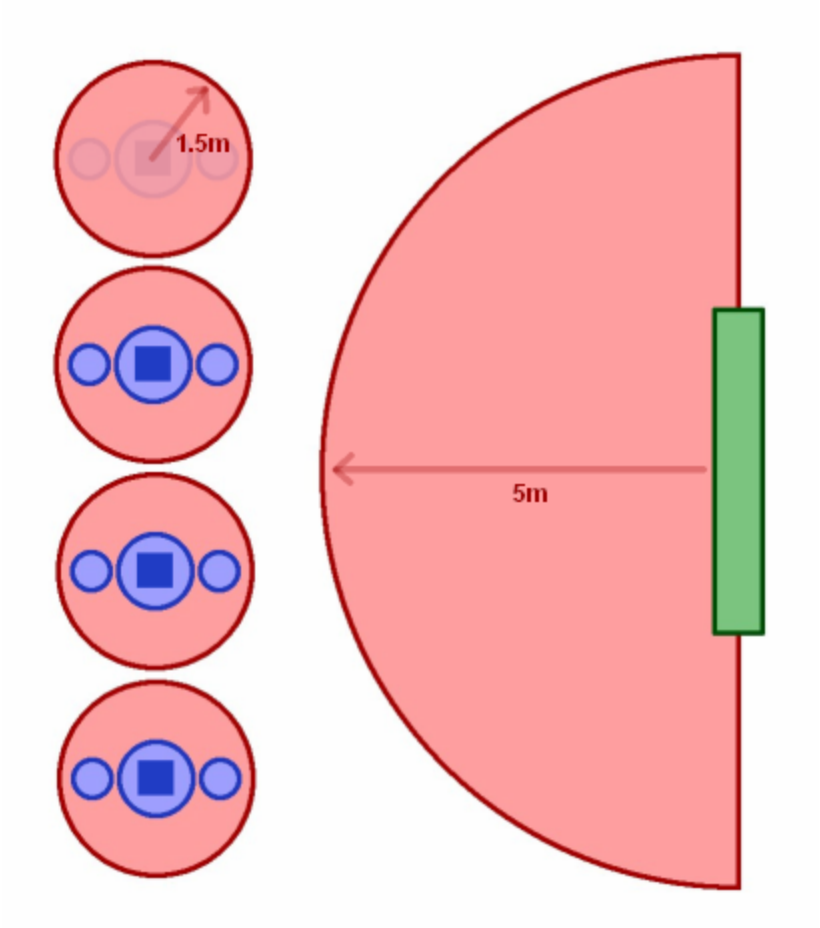
An example of this would be an environment with a few chess mini-games and a poster:



For the chess mini-games, it is likely you would want a trigger radius which encompasses the chess table and the seats, so that you can join the table from any direction as soon as an avatar moves close to it or collides with it. On deciding how far out the trigger radius extends from this encompassing radius, you'll need to bear in mind that there are other chess games close by. Although the targeting system is intended to be as intuitive as possible on deciding which game has targeting preference, the trigger radii should not overlap.

For a poster, the trigger radius needs to be large, so that it can be viewed in full screen with ease, but not so large that it gets selected by accident. So in this case it is best to extend the trigger radius out as far as possible, without overlapping the chess games.

Taking into account typical sizes of tables, chairs and posters, and the collision radius of the avatar, the trigger radius distances may look something like this:



If the trigger radius is not specified by the developer, the system will assign appropriate default values depending on the identity of the item in question. For example, the default target radius for a screen will usually be larger than for a mini game. However it's advised that a trigger radius be specified to ensure the developer has confirmed what they consider a reasonable value.

Changing the Trigger Radius

For screens added to a scene in the Scene Editor, the trigger radius can be changed within the screen's properties.

> If the Targetable component is added to the object in the Object Editor and the trigger_radius property is set, this overrides the trigger radius value set in the Scene Editor. For more information on this component, see Object Component Properties.

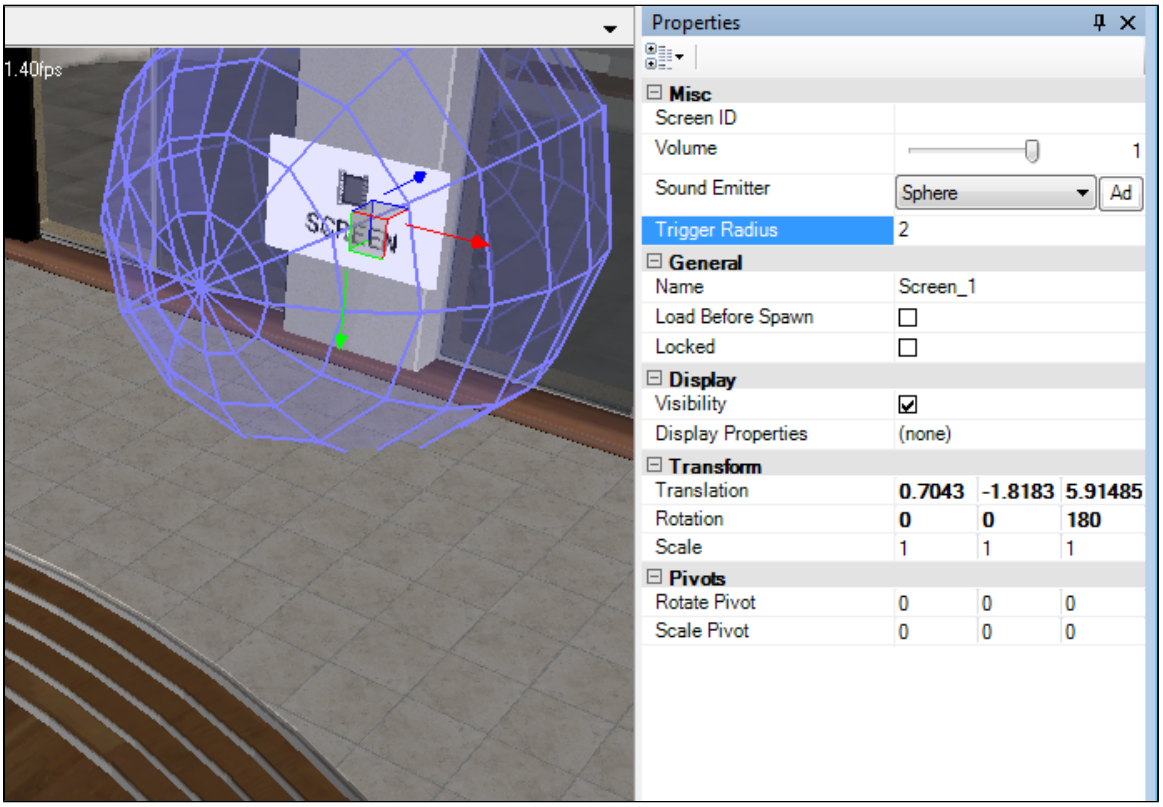
For screens created via Lua, the trigger radius can be changed using the function `Screen.SetTriggerRadius()`. For Lua screens, see the Lua API Reference.

The trigger radius for the following objects can be edited in Scene Editor:

- Screens
- Club Bulletin Screens
- Scene objects
- Arcade Games
- Mini-games
- Spawn Point
- Change Event Trigger
- Change Event Box Trigger

> Only the trigger radius for the objects listed above can be edited in Scene Editor.

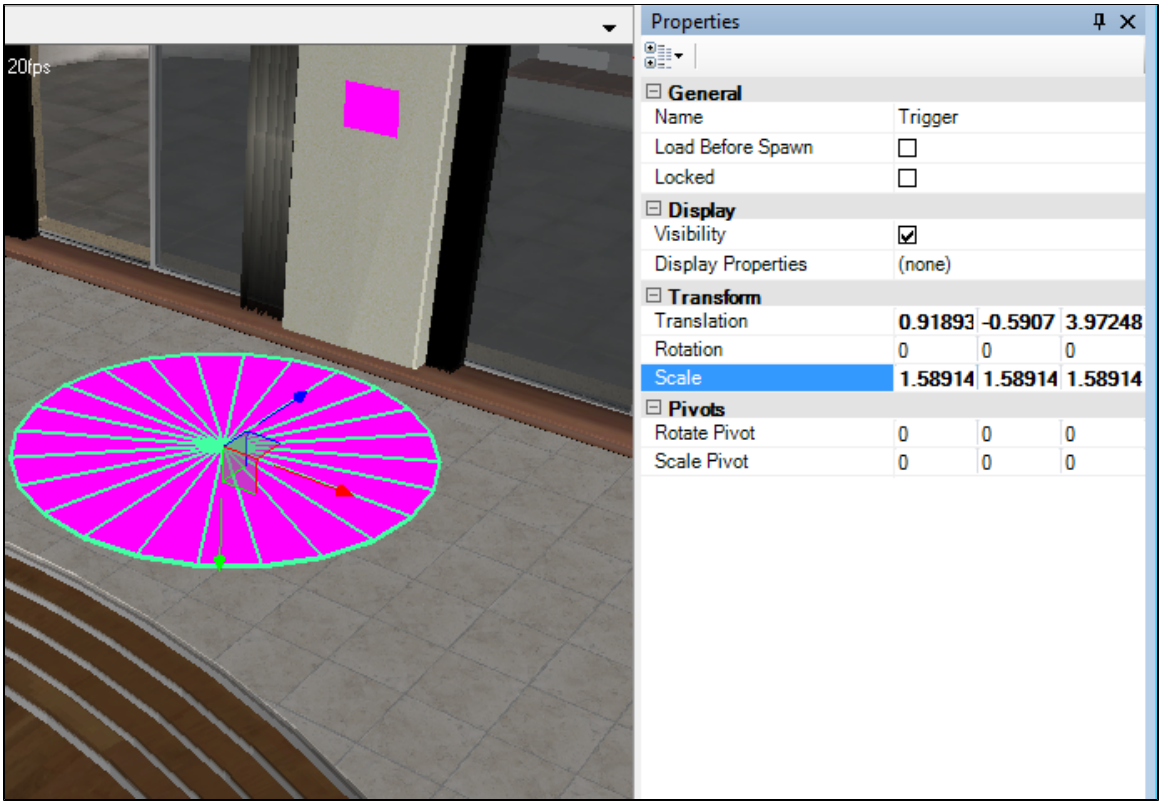
To change the trigger radius for a screen in Scene Editor, select the screen object and edit the Trigger Radius field in the Properties panel. This is measured in meters:



Arcade Games

To change the trigger radius for an arcade game in Scene Editor, select the trigger for the Arcade Game object and edit the Scale fields in the Properties panel. This is measured in meters.

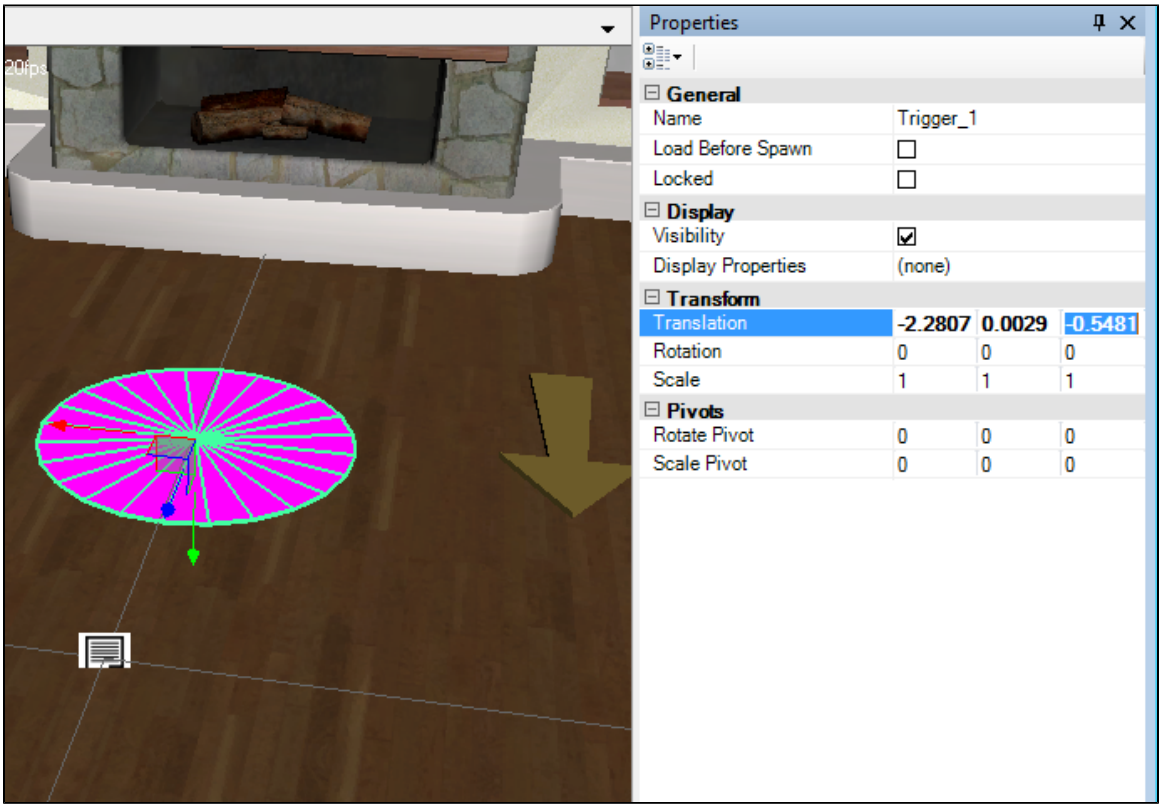
Alternatively, click  on the toolbar, then click and drag to increase or decrease the radius of the trigger.



Mini-games

To change the trigger radius for a mini-game in the Scene Editor, select the trigger for the mini-game object and and edit the Scale fields in the Properties panel. This is measured in meters.

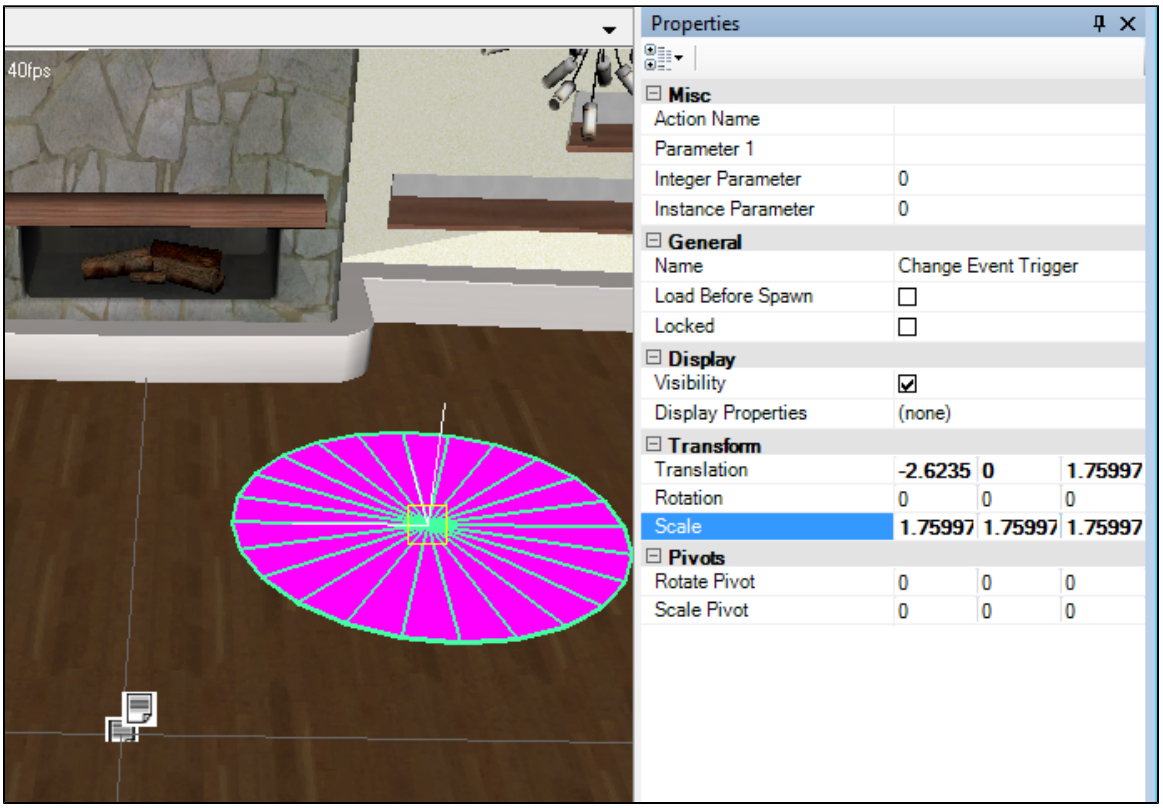
Alternatively, click  on the toolbar, then click and drag to increase or decrease the radius of the trigger.



Change Event Trigger and Change Event Box Trigger

To change the trigger radius for a change event or change event box in the Scene Editor, select the Change Event Trigger or Change Event Box Trigger object and edit the Scale fields in the Properties panel. This is measured in meters.

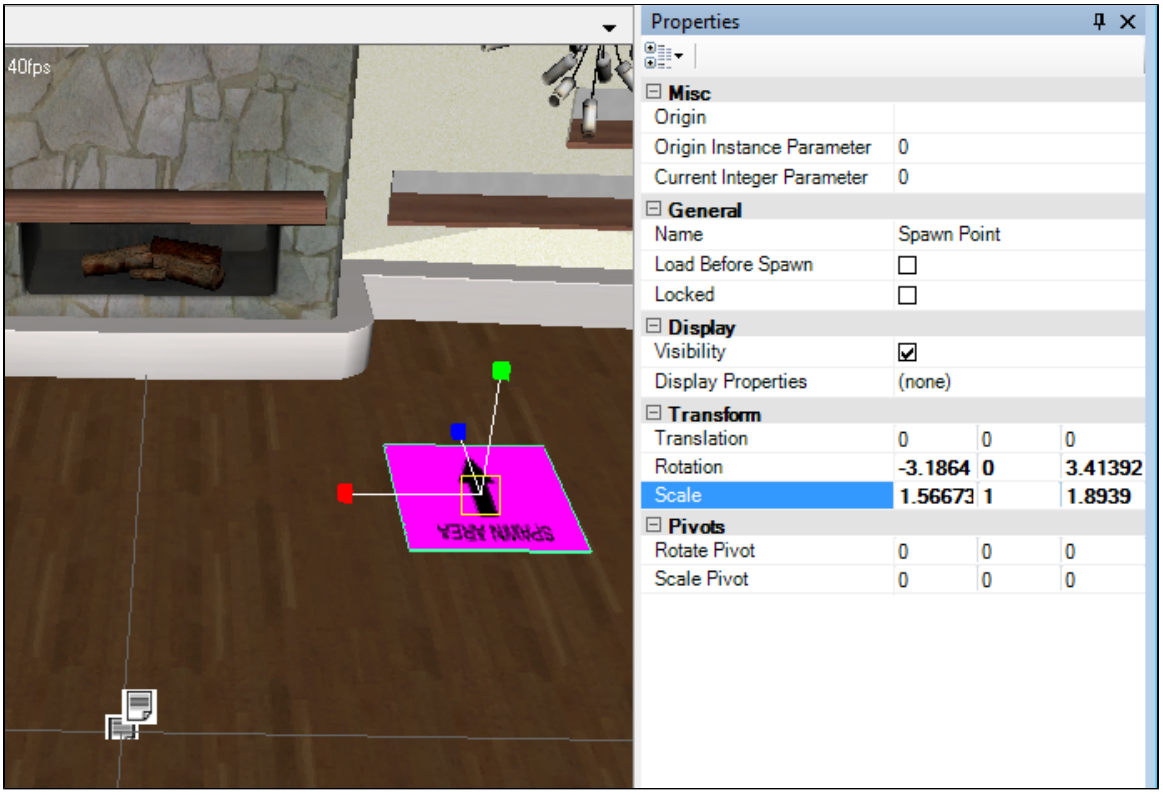
Alternatively, click  on the toolbar, then click and drag to increase or decrease the radius of the trigger.



Spawn Point

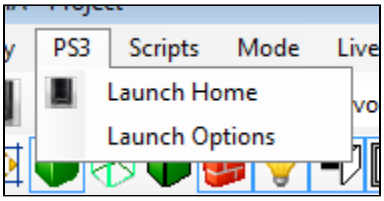
To change the trigger radius for a spawn point in the Scene Editor, select the spawn point and edit the Scale fields in the Properties panel. This is measured in meters.

Alternatively, click  on the toolbar, then click and drag to increase or decrease the radius of the trigger.



Launching a Scene in PS Home

To quickly launch a scene from the Scene Editor, click the Launch Home button  , or select PS3 > Launch Home:



A .scene file is generated based on the content currently loaded in the Scene Editor, and the client launches the generated scene file.

> Remember the following:
>
> - If your .scene file is subject to source control, ensure it's checked out before you launch the scene, as the .scene file is updated when you launch the scene. If you don't remove the read-only flag from the .scene file, an error message will be displayed.
> - The HomeDeveloper.self file specifies the file executable that will be launched when you click the Launch Home button. If the file is incorrectly specified, the launch will fail.
> - Click the Reload Scene button to refresh the loaded screen without restarting. The button also updates the .scene file to reflect any changes made.

For more information on setting up and launching the Home Developer client, see The Home Client.

Connecting the Scene Editor to the Target Manager

You can connect the Scene Editor to the Target Manager to control the camera in the client through the Scene Editor.

To connect the Scene Editor to the Target Manager, connect to a Test or Devkit from the Target Manager, and then launch the Scene Editor. Now you're connected.

Alternatively:

1. Open the Scene Editor.
2. Select Live Editing > Manage Targets.
3. Select the target you want to connect to from the list displayed.
4. Click the Connect button.

> You can also connect the Scene Editor to a target through the Target Manager while the Scene Editor is open, but only if the Scene Editor is not connected to a target already.

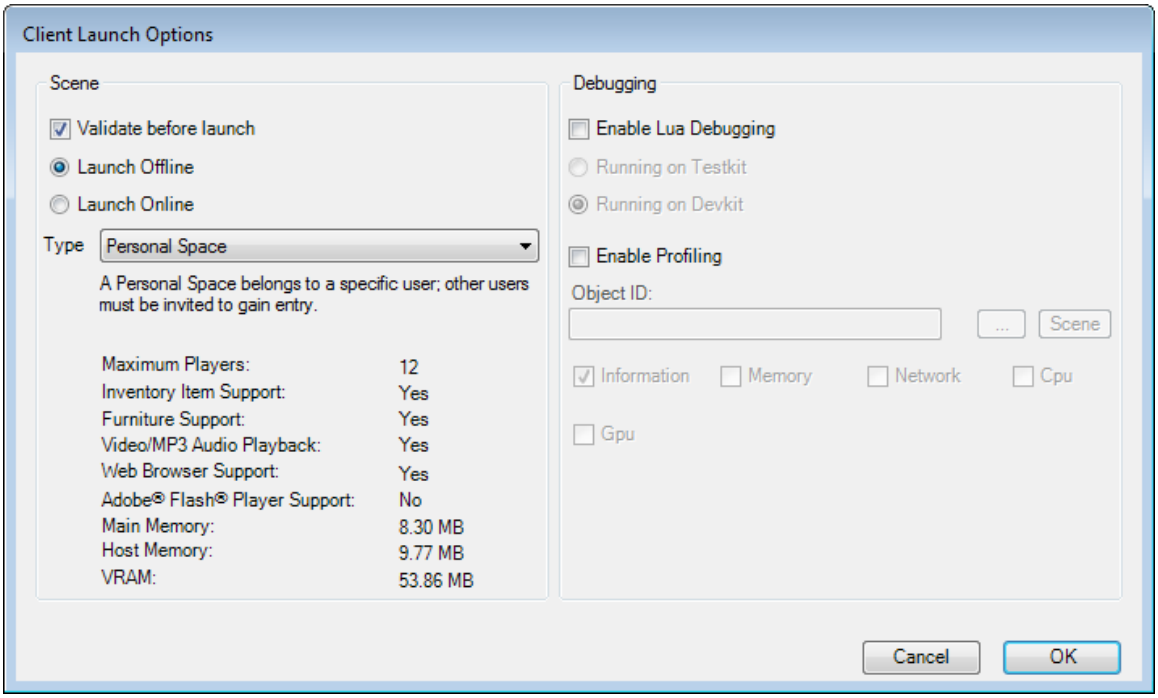
When you're connected to the Target Manager, the Scene Editor icon in the Icon Tray changes to  . You can issue Dev Debug console commands through the Scene Editor's Scripting window.

Changing the Launch Options

The Scene Editor's PS3 menu also contains Launch Options, which you can can to specify how scenes are launched.

To change the launch options:

1. Select PS3 > Launch Home. The Client Launch Options dialog is displayed:



2. If you want to validate your scene before it is launched, check the Validate before Launch box.
3. If you want to test your scene on a local PlayStation®3 console, select Launch offline. Only your avatar will be available in the scene.

   Alternatively, if you want to give scene access to more than one user, select Launch Online (each user needs a separate PlayStation®3 console). This utilizes the developer build online across a network, where available. For more information about launching your scene online, see Launching Online below.

> For testing online, you will need a scene key (see below).

> Clubhouses cannot be run offline with the full functionality. Launch the scene online.

4. Select the type of scene that you want to launch from the Type drop-down list. Your selection changes the scene type when launching from Scene Editor and determines what functionality is available within that scene. The following options are available:

   - Personal Space: A personal space with access to customization options, such as placing furniture. A personal space only exists while the owner of the space is present and other users can only enter by invitation from the owner.

Once the owner exits the space, the instance of that space ends and any users still occupying it are returned to their default space (usually their own personal space). Personal spaces support up 12 users, including the owner.

- Public Space: A shared public space can accommodate up to 60 users. Anyone can enter a public space (unless the space has an Access Object ID associated with it, see Step 6 below).
- Clubhouse: A space associated with a PS Home Club that only clubhouse members can access (up to 32 users). In many ways, the clubhouse is the same as a personal space, with customization options such as the placing of furniture. However, unlike personal spaces, the space persists even when the owner has left and club members can access the space whenever they are online.
- Game Space (16 Players): This Scene type is optimized for games, providing as much memory as possible to game content and disabling many non-game features (such as video playback and web browsing). The maximum capacity of this Scene type is 16 players, making it perfectly suited to games which utilize the `RtGame` library.
- Game Space (32 Players): This Scene type supports up to 32 players. The higher player count means that there is slightly less memory available than in the 16 player space and is better suited to games using the `Minigame` library.
- Game Space (60 Players): This Scene type supports up to 60 players. The higher player count means that there is slightly less memory available than in the 16 player and 32 player spaces and is better suited to games using the `Minigame` library.
- Public Space (No Video): Like a public space, this space can accommodate up to 60 users and anyone can enter the space (unless the space has an Access Object associated with it). Unlike a public space, it provides as much memory as possible by disabling many non-game features (such as video playback and web browsing).
- Video Space: This Scene type has additional memory allocated for video playback. This allows AVC/H.264 video playback up to Level 3.1 (HD 720p at 30 frames per second). This Scene type supports up to 16 players.

> You cannot embed furniture objects in personal spaces and Clubhouses.

The scene type for publishing is defined within the Content Delivery System (CDS) when submitting the content. For more information on submitting content, see the CDS Online Help.

For more information on the differences between each scene type, see PS Home Spaces.

5. To use the SCE Lua Editor and Debugger tool (SLED) when launching the scene, check the Enable Lua Debugging box. You can then select whether to run the debugger on a Testkit or Devkit. This sets the appropriate port and protocol settings for the kit.

> You need to enable debugging support in your script. For more information on using SLED, see Debugging Lua Script.

6. To activate the profile GUI for scenes and objects, check the Enable Profiling box (for more information on the profiling tools available, see Profiling in the Client). If this box is checked, you can do the following:

    a. Enter an object ID in the Object ID field to specify that the scene requires an access object in the user's inventory for them to enter the scene.

    Normally, a public space is available to all users in a specific region (so a public space that is submitted to the SCEA region is available to all users in the SCEA region). When a public space is submitted to the CDS, you have the option to restrict access to it by adding an Access Object ID to the submission details. If an Access Object ID is added, only users with that object in their inventory can access the scene. Users who don't have the access object cannot be invited into the scene. The Access Object can be any object in PS Home and the object can be any type of object. Nothing needs to be done to the object to make it an Access Object.

> This only changes the access restrictions for the scene launched from the Scene Editor. You need to re-enter the Access Object ID when submitting the scene to the CDS to restrict access to the scene when it is live.

    b. Select the sidebar panels that are initially visible once the scene is launched by checking the box next to the name of each panel required.

7. Click OK to save the changes made.

Scene Keys

If you're developing and testing your scene online, you need a scene key. Every scene needs a scene key, which also allows anyone who launches online with the same key to see and interact with others in that scene. The scene key is generated by the Home Support Team on request. To request a scene key, contact Home Support.

To test online with a scene key but restrict access to the scene, use the console command `enablePrivateLobbies 1` (in the Target Manager or the debug menu) so that every time you enter a public space, the space is made inaccessible to other users,

unless they are specifically invited, functioning as if it were a personal space (e.g. an apartment). This is useful if you have many users logging into a space and you are changing network features like mini-games. You can isolate your new mini-game so that it doesn't break the old ones the other users have access to.

Launching Your Scene Online for Multiple Users

To launch your scene online for multiple users:

1. Open the Scene Editor and select PS3 > Launch Options to display the Client Launch Options dialog.
2. Ensure that the following launch options are set (you cannot launch online without them):
    a. Select Launch Online.
    b. Select the scene type. For more information on launching the different scene types, see PS Home Spaces.
    c. Enter the scene key.
3. Ensure that your development or test kit is set up so that Scene Editor can launch the scene online and it's on a network. For more information, see Configuring the Network.
4. Click the Launch Home button  to load the scene online using the networked developer build. You can now access the scene using your PlayStation®3 console.
5. To allow multiple users into your scene, you need to set up additional PlayStation®3 consoles on your network to load the client and access the scene.

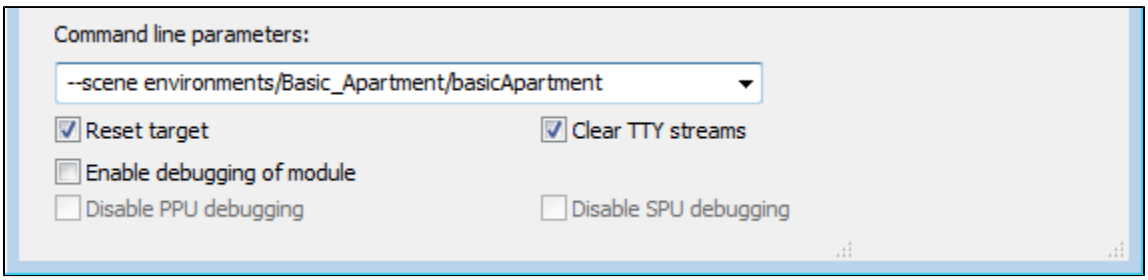
> The client only opens scenes that are included within the LocalSceneList.xml file in the \build\environments folder of the HDK installation. This means that each client you want to have access to your scene needs access to the scene list. The easiest way to do this is to load the client by connecting to every PlayStation®3 console using the Target Manager from your PC. This way you can control the clients from one place (for example, making it easier to reset them), and every client uses the same files from your local HDK installation. The local scene list is updated every time you launch a scene using the Scene Editor and the scene information added. Because you launch all the clients using the scene list on your PC, it guarantees that the scene is available for each client.

6. Connect to each additional PlayStation®3 console from your PC using the Target Manager, by entering the following command line parameter:

```
--sceneid <filename>
```

Where <filename> is the name of your scene without the file extension (.scene).

For example, a scene called 'BasicApartment.scene' would be entered in the Load and Run Executable dialog of the Target Manager like this:



This loads the scene on each PlayStation®3 console and allows a user on each console to access your scene (provided access isn't restricted by an access object or the maximum number of users in the scene, etc.).

For more information on the restrictions that may apply, see PS Home Spaces.

> The scene ID entered in the Target Manager is defined by the entry in the local scene list, which is generated from the filename of your scene. When submitting your scene to the CDS, you enter a scene name with your submission. Unlike the local scene list used by the Scene Editor, the unique ID for your scene in the CDS is the scene name, not the filename.

Preparing Scenes for Packaging

To successfully package a scene in the Scene Editor, your scene must have at least these items:

- Scene Identifiers
- Thumbnails
- Age Rating Information
- An Environment Map

Resources such as particle effects, animated models and Lua scripts that have been added are also packaged with the scene. However, objects (such as mini-games and arcade games) referenced by the scene must be packaged separately using the Object Editor, then submitted along with the scene. For more information, see Preparing Objects for Packaging.

> Successful packaging does not guarantee that your scene passes Quality Assurance (QA). For QA requirements, see the Technical Requirements Checklist (TRC) for scenes, and the scene limits in HDK Tools Validations and Memory Limits.

Scene Identifiers

A scene has the following identifiers:

Name	Description
Scene Project Filename	The name of the .scproj file that contains the scene that the HDK tools use to edit the scene. You can choose any filename, because the only place this name appears is in the scene list. For example, yourfilename.scproj.
Scene Filename	The name of the .scene file that contains the scene. The file is stored in the .sdc scene description file within your scene package. The PS Home client uses this file to load the scene. This name is referenced in the scene list, but PS Home users cannot see it.
Display Name	A descriptive name for the scene. The display name, description and thumbnails are visible to everyone in PS Home. For more information on how to add the display name, see Adding a Display Name, Description, and Thumbnails below.
Scene Name	A unique name that you assign to the scene when you submit it for publication. When you submit your scene package online, you are asked for the scene name. This name is a unique ID and does not have to match the name of your .scene file or the descriptive name in the .sdc file. When you submit the scene to the Content Delivery System (CDS), the site automatically checks whether the name is available. PS Home users cannot see this name. To request that your scene name is assigned to your scene, submit a support ticket.

> The scene project filename (.scproj) and the scene filename (.scene) cannot have spaces in them.

Thumbnails

When you complete your scene, make sure that you have named and described the scene and assigned the correct thumbnails before packaging.

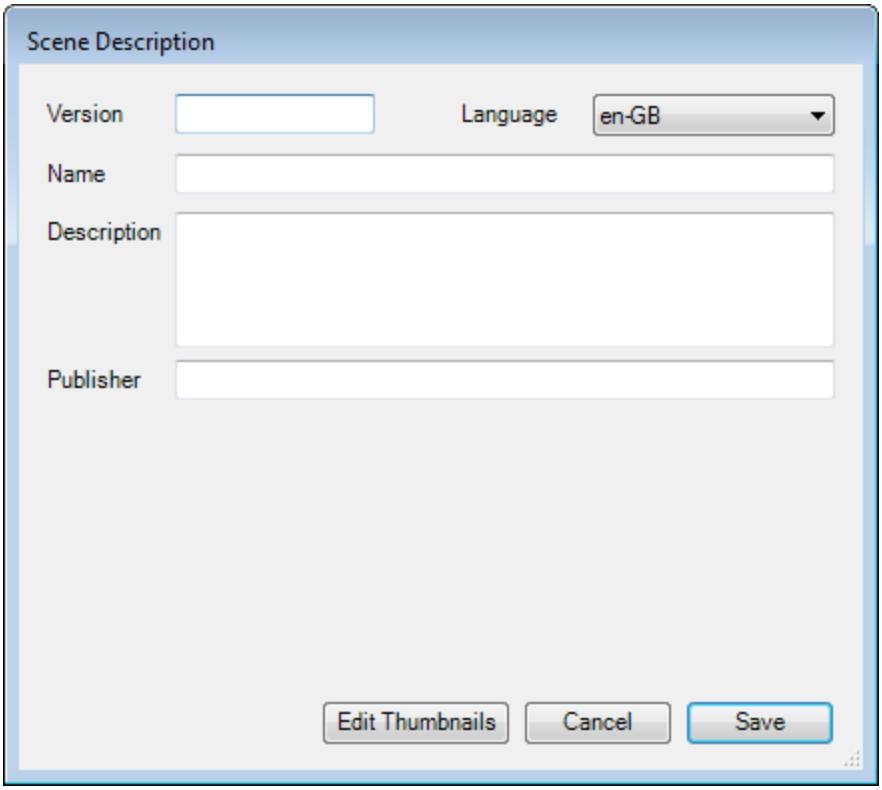
Every scene must have a large and a small thumbnail. Small thumbnails appear in the menus in PS Home. Large thumbnails appear in the PlayStation®Store and are displayed on the relocation screens for scenes.

For more information on creating object thumbnails, see Thumbnails.

Adding a Display Name, Description, and Thumbnails

To add a display name, description and thumbnails:

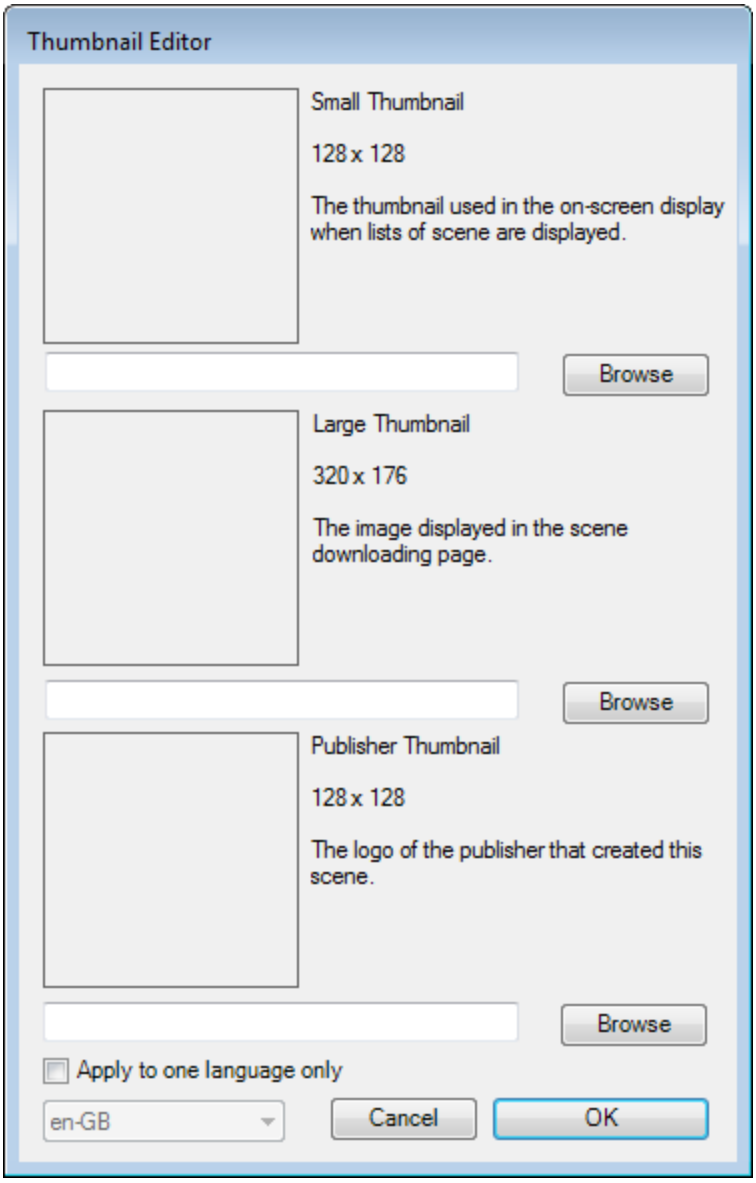
1. Select Edit > Edit Description to display the Scene Description dialog:



2. Enter the following (this information is visible to users in PS Home's Download menu):

    a. Enter the version number for your scene in the Version field.
    b. Select the language that you want the information to be localized in from the Language drop-down list.
    c. Enter the name of the scene in the Name field.
    d. Enter a brief description of the scene in the Description field.

3. To add thumbnails for your scene, click Edit Thumbnails to display the Thumbnail Editor dialog:



4. Browse and select the required thumbnail images.

   The information is stored in an .sdc file, and the thumbnail images are included when you package the scene. The .sdc file must contain the following items:

   - A scene display name (this is not the scene name given to your scene by your SCE Regional Manager)
   - A description of the scene
   - A small thumbnail (128 x 128)
   - A large thumbnail (320 x 176)

   > The Publisher Thumbnail is optional.

5. If you want to enable regional variation and ensure the object appears differently in different regions, check the Apply to one language only box and select the language from the drop-down list displayed (you will need to repeat this step for each region you want to localize to).
6. Click OK to save and close the Thumbnail Editor dialog, then Save to save and close the Scene Description dialog.
7. Repeat the above steps for each language in which the scene is translated.

Age Rating Information

To make sure that your content works as expected, set the following age restrictions:

- Age rating (if required)

- Parental control level (mandatory)
- Minimum age (mandatory)

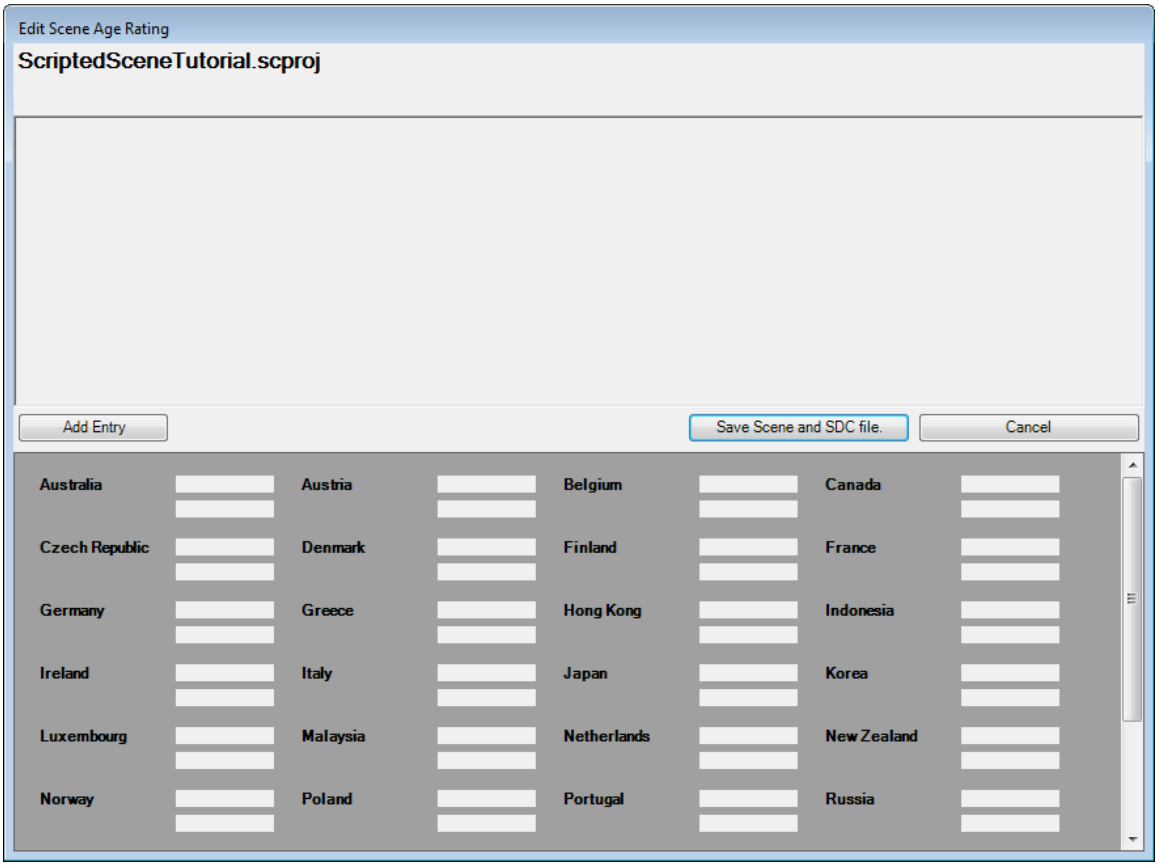
The Home Developer Build must be connected to PSN to validate the age of the user, even in offline mode. If there is no connection to the PlayStation Network (PSN), the default age of 0 is used. To test content without a connection to PSN therefore, you must set the minimum age of entry to 0. You can set the Minimum Age to 0 only for testing. You cannot use this setting for published content.

> Restricting content is a requirement. For information on which settings to use and where to obtain official guidance, see Age Restrictions and contact your Regional Account Manager.

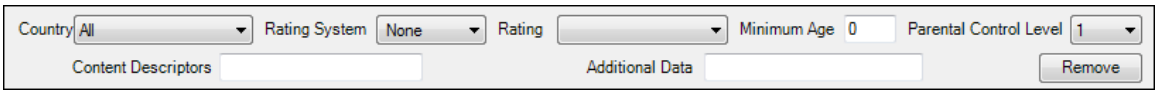
Setting Age Restrictions

To set age restriction information:

1. Select Edit > Edit Age Ratings. The Edit Scene Age Rating dialog is displayed:



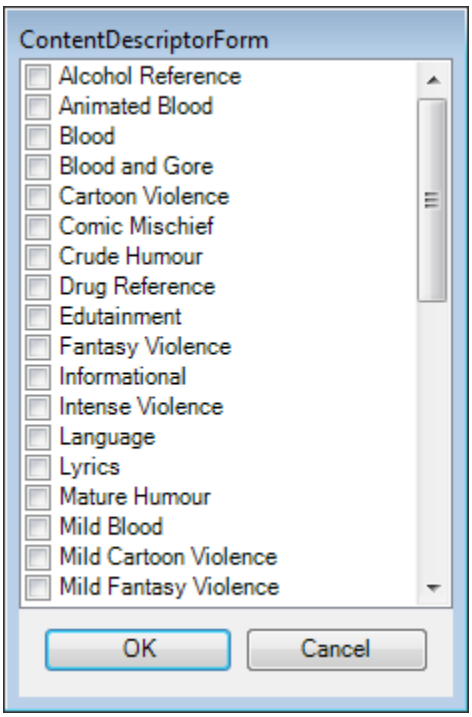
2. Click Add Entry to create a new entry:



> The Additional Data field is not currently used. Leave this field blank.

3. Select the country in which the restriction applies from the Country drop-down list.
4. Select the appropriate rating system from the country selected from the Rating System drop-down list.
5. Select the rating level required from the Rating drop-down list.
6. Enter the minimum age for which the content is appropriate in the Minimum Age field. This setting must match the Rating level set.
7. For regions where age rating systems require descriptions of the contents, click the Content Descriptions field to display

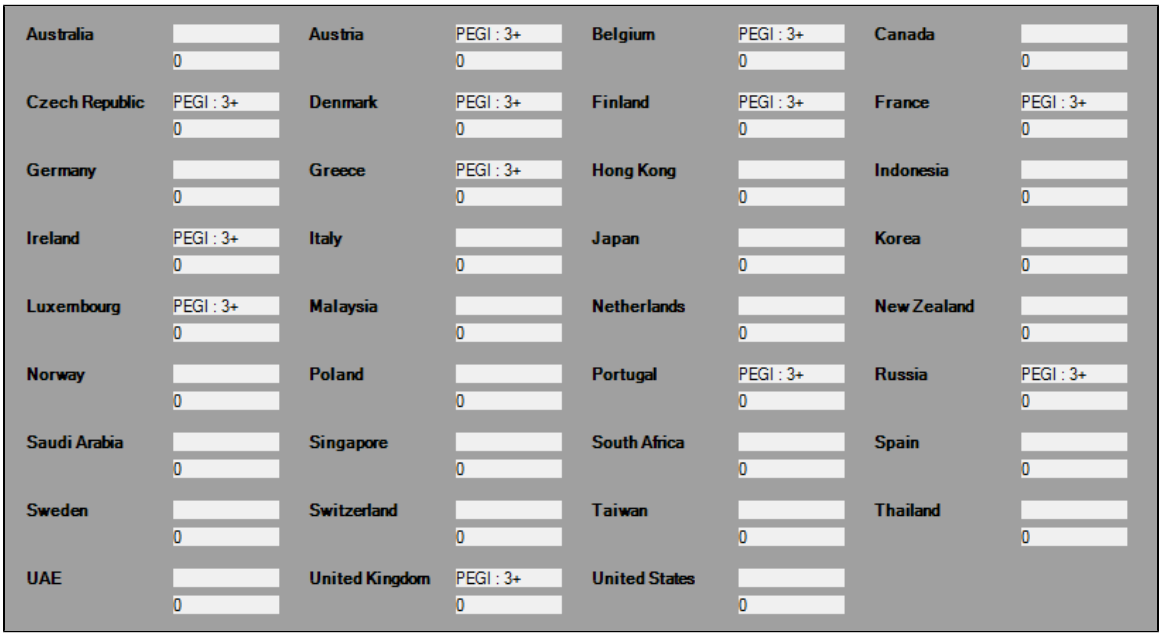
the Content Descriptor Form dialog:



Check the boxes next to the descriptors that apply to your scene, then click OK. Each rating system has its own set of terms for describing the content of certain ratings. For example, the ESRB descriptors include such descriptors as: 'Alcohol Reference', 'Blood and Gore', 'Mature Humour'.

> The maximum number of descriptors you can set is 9. If you set more than this, the descriptors will not display in PS Home.

8. For each country affected, click Add Entry and repeat steps 3 to 7. A summary of all of the restrictions that you're adding are automatically displayed in the bottom half of the Edit Scene Age Rating dialog, for example:



9. When you've add all of of the required age restrictions, click Save Scene and SDC file to save the scene and the .sdc file.

Environment Map

Every packaged scene must contain a custom environment map for that scene to make sure that any shaders that require environment map reflections reflect the scene in question and not the default environment map reflection.

1. In PS Home, position the free camera to a point in the scene from where you would like the origin of the environment cube map to be generated. You can toggle to and from the free camera mode by clicking the right control stick.
2. When in position, execute the console command `envShot 512` (where 512 is the height of the cube map). This command generates a 3072 x 512 TGA format file in your `build` folder called env.tga. It has six views in each of six directions, laid out as shown in the following example:

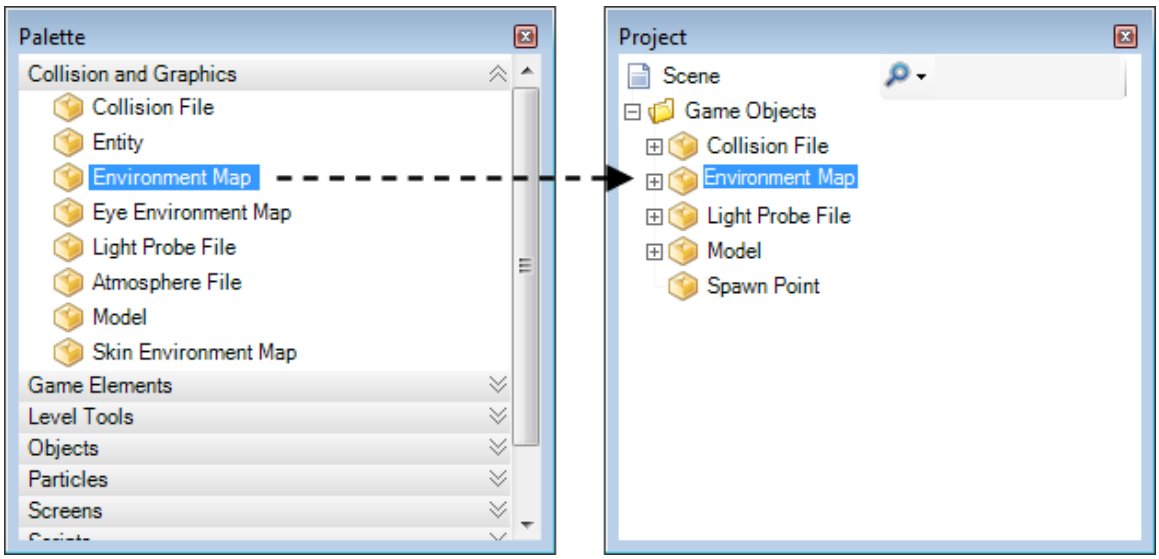


3. Convert this file to .dds format and save it to the `build\environments\your scene` folder as EnvironmentMap.dds. You can now use the Scene Editor to add it to your environment. For more information about valid .dds, see Maya Textures.

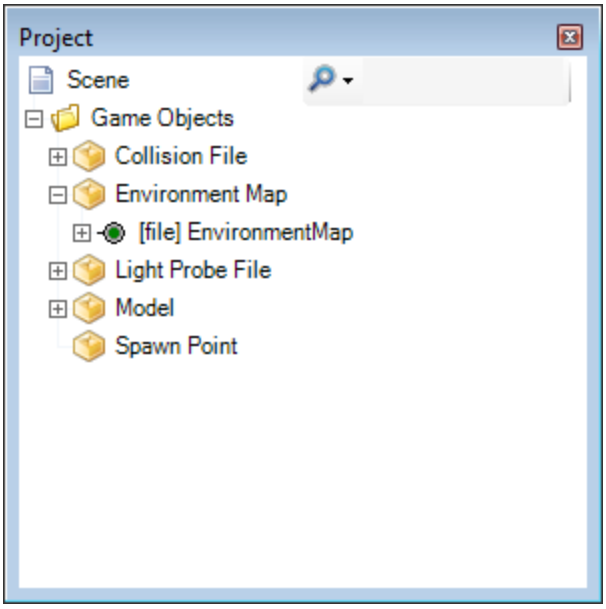
4. Open your .scproj file in the Scene Editor and add the environment map texture as an asset:



5. Drag the Environment Map object from the Palette panel to the Game Objects folder in the Project panel, or directly onto the Design View. The object is added under the Game Objects folder:



Add the EnvironmentMap.dds file to the Assets panel and then drag it onto the file entry under the Environment Map object in the Project panel. When the asset has been applied, the file entry turns green:



6. Save the .scproj file. When the environment is live, any environment mapped surface has the correct reflection.

Profiling Content

When testing your content, you should profile your scenes to determine how resource-intensive the scenes and objects are. Use the following console commands to profile your scene:

- `profileEnable 1`
- `enableMemoryStats 1`
- `enableNetStats 1`
- `ProfileGui.Enable 1`

For more information on profiling, see Profiling in the Client.

Packaging Scenes

You must package your scene into an optimized archive format before submitting it for Quality Assurance (QA).

Packaging Requirements

Make sure that:

- You have completed the preparation for packaging and that the scene is finalized before finally packaging it. See Preparing Scenes for Packaging.
- You do not modify the scene content, file name or localization text after you have packaged the scene. If you do, the content will fail QA.
- The scene does not include any unused files. Packaged scenes cannot contain unused files.
- Scene packages are within the maximum size limit. Packages submitted for publication through the Content Delivery System (CDS) must be less than 100 MB. Keep the file size of packaged scenes as small as possible.

Packaging Scenes

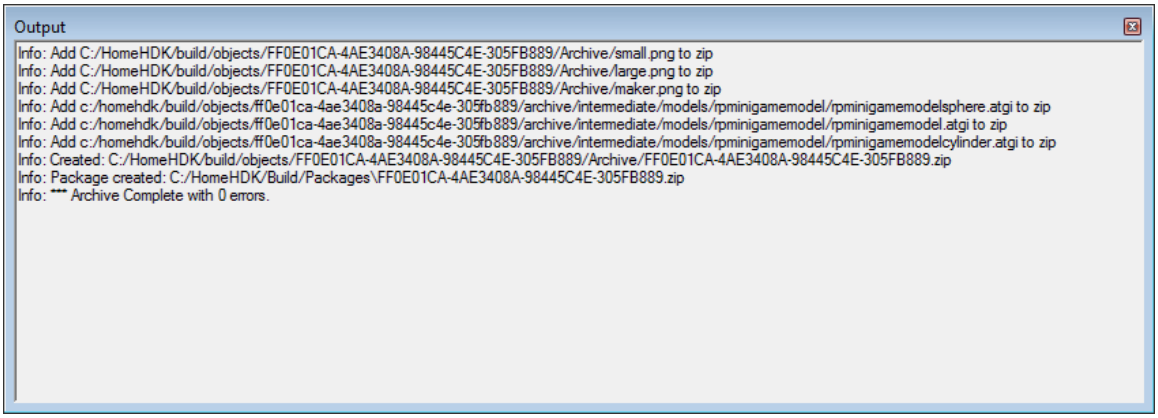
> From HDK 1.2 onwards, the Scene Editor and the Object Editor automatically optimize file compression during packaging. If you have scenes that were created with versions older than HDK 1.2, open them in the Scene Editor and repackage them using the packaging process described here.

To package a scene:

1. In the Scene Editor, select the scene you want to package.
2. Click the Package Scene button  on the toolbar, or select File > Package Scene.
3. Check the Output panel to see whether or not packaging was successful.

Automatic validation is carried out during packaging. For information on what is validated, see HDK Tools Validations.

If the packaging process is successful, the Output panel shows that there are no errors or warnings:



Successful Packaging

The packaging process creates the following items:

- A subdirectory called Packages, in the build directory.
- A .zip file in the Packages directory, which contains the packaged scene.

For example, if you package the following file:

<HDK_INSTALL_DIR>\build\environments\myenvironment\myenvironment.scene

The scene packaging process generates the following .zip file:

<HDK_INSTALL_DIR>\build\Packages\myenvironment.zip

Scripted Content

With more than 70 libraries, The Lua API in PS Home gives you control over the systems, resources, and network communication in the client. This section provides information about the general use of Lua in PS Home, optimization and garbage collection, debugging and resources.



Lua Scripting

Lua is an open source, industry standard, scripting language. It is free software that you can use for both academic and commercial purposes. It has no royalties or GNU-like "copyleft" restrictions. Its licenses are compatible with GPL. Lua is not in the public domain and PUC-Rio keeps its copyright.

Lua licensing details are included in the HDK Installation. Lua documentation is available at http://www.lua.org/docs.html.

The HDK supports the implementing of dynamic behavior within the Home runtime using Lua. Over 800 Home-specific functions are exposed to the developer. These functions are grouped into related sets, called libraries.

You can use Lua to do many things, such as:

- Author interactive 3D experiences like multiplayer networked games.
- Write 2D sprite-based arcade games.
- Control aspects of Home's runtime through a command console.
- Customize Home's game launching process.

Content exported using the current HDK does not support Lua API functions from previous releases that have been deprecated in the current HDK or an earlier release. Only the functions listed in current HDK's Lua API Reference are supported.

Scene Scripting

Although this section deals with scene scripts, if possible use embedded objects rather than adding scripted behavior to your scene. We recommend using embedded objects because, if you update the script on an embedded object, you need only re-submit the object to the Content Delivery System (CDS) for publishing. The scene still undergoes Quality Assurance (QA) with the updated embedded object.

If you use a scene script, you need to repackage and resubmit the scene and any other embedded object to the CDS, which all undergo the standard submission and QA processes.

You can write scene scripts in Lua. Scene scripts run as soon as the associated environment loads and are typically used to control dynamic, ambient behavior. For example, you can implement planes flying overheard, or a search light that follows avatars using this mechanism.

Writing Scene Scripts

The main scene script for an environment must have the following structure:

```
-- Load required HDK LUA Libraries here. e.g.:
LoadLibrary( "Entity" )
LoadLibrary( "Scene" )

-- Load other user scripts required by the main script. e.g.:
Resource.Run( "LightUtils" )

-- Global initialization here...
lightEntity = Scene.FindEntity( "floodlight" )

-- Update function. Spelling of function name is important...
function Update()

      -- Per frame update code here...
      LightTrackAvatar(lightEntity)

end
```

The Scene library supports the querying of the associated environment for certain scene elements, such as named entities and screens listed in the Scene Editor.

Only objects owned by the scene are granted permission to use the Scene library, such as:

- Embedded objects (including embedded active items)
- Mini-Game objects



Note that:

- Arcade games cannot access the Scene library.
- Access to the Scene library requires additional testing and implementation, because it is possible for multiple scripts to make changes to scene parameters (e.g. using the Atmos API to change the sky, or various graphics engine settings). This means that, before submission to the CDS, a new script that uses the Scene library must be tested against all the other scripts that use the Scene library in that scene (and in every scene to which the script is to be added).

Locking a Player

With scene scripts you can lock a player's controller, using the function Scene.LockPadControls(). This is useful for situations like an introduction camera flyby, retrieving a player's records from a web-server, or a triggered event that you want a player to watch. Be careful that the player is not confused by the fact that their controller is locked, and pay careful attention to the restrictions for using this function.

Bear in mind the following:

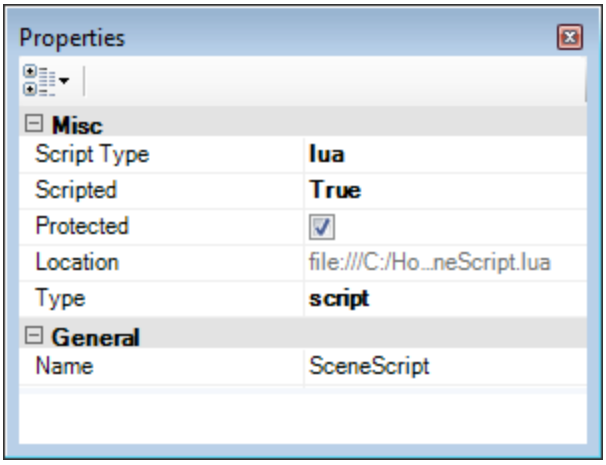
- Always have an obvious reason for the player being locked, like an introductory sequence, or a message telling the player to wait. Do not just have the avatar visible as normal with locked controls and no indication as to why.
- There should be a short or no delay between locking a user's controls and playing the event (e.g. an intro sequence), or showing the display.
- Unlock a player's controls once the event is finished.
- If it is not obvious why the player's controls are locked, inform the player.

Adding a Scene Script to your Scene

Scene scripts are unlike all other scripts in PS Home in that they are not resources that are loaded into the Object Editor and assigned to an object. Instead, the scene script is loaded as an asset in the Scene Editor. You can add as many script assets as you like to your scene in Scene Editor, but only one must define the Update function that will be called by Home every frame.

To add the script as an asset to the scene in the Scene Editor:

1. Right click on the Assets folder on the left side of the Assets panel and select Add > Existing Asset.
2. Browse to and select your script file and click Open.
3. Select the script asset in the Project panel. For a Lua script, the Script Type drop-down list in the Properties panel is set to lua (or luac for compiled Lua scrips) and the Scripted field is set to True:



For other assets that your script needs to access, make sure that the Script Type drop-down list is set to the corresponding value and the Scripted field is set to True. The script types are as follows:

Script Type	Description
lua	A Lua script
luac	A compiled Lua script
file	A data script (typically an XML file)
model	An MDL file (generated by the HDK exporter)
anim	An ANM animation file (generated by the HDK exporter)
skeleton	An SKN skeleton file (generated by the HDK exporter)
collision	An HKX collision file (generated by the HDK exporter)
texture	A DDS texture file
particle	An EFX particle effect file (generated by the ATG Particle Effects Tool)
soundbank	A BNK SCREAM sound bank
soundstream	An MP3 sound stream

Spelling and capitalization are important in Lua scripts. Some functions may cease to work or cause a crash if they are inaccurate.

The Protected property of Lua scripts is checked by default, to improve the security of the content.

Object Scripting

An object is an item that can be downloaded by the Home client by the end user. This item can be static, like an item of furniture. Or it can be dynamic, like a multi-player networked mini-game, or a 2D arcade game. Dynamic objects are made possible by incorporating Lua scripts into them.

An object consists of a set of components where each component adds some functionality to that object. You can add and remove these components using the Object Editor. Adding components allow your script to access Home Lua libraries using the LoadLibrary function.

The list of components and the Lua libraries they provide are as follows:

Object Component	Accompanying Home Lua Libraries

Active Item	Active
Arcade Game	ArcadeGames
Camera	Camera
Entity	Entity
Game Launch	GameLaunch GameLaunchExporter GameLaunchMenuSys GameLaunchMenu GameLaunchButton GameLaunchSlider GameLaunchListItem
Light	Light
Lua Environment	All global functions (no LoadLibrary call necessary) AsyncCommand AsyncContext Base64 BasicGenX BigInteger Collision Debug Environment GameInfo Hash HttpPostData LocalPlayer Material Matrix44 MediaLibrary MemoryContainer Object Osk Person Quaterion Raycast Resource Rewards SaveData Sound SoundBank SoundStream System Texture Time User Vector4 Xml
MiniGame	MiniGame
Network	NetPropertyBag OutboundMessage ReceivedMessage
OSD	Osd OsdBasicChip OsdBasicLegend OsdBasicMenu OsdBasicPopup OsdBasicSingleGraphic OsdBasicText OsdObject OsdSelectionParent OsdSimpleMenu OsdTextLines OsdTextPanel
Pad	Pad

Renderer	Renderer Sprite SpriteAnim
Repertoire	LocalPlayer Person
Reward	Rewards
Screen	Screen

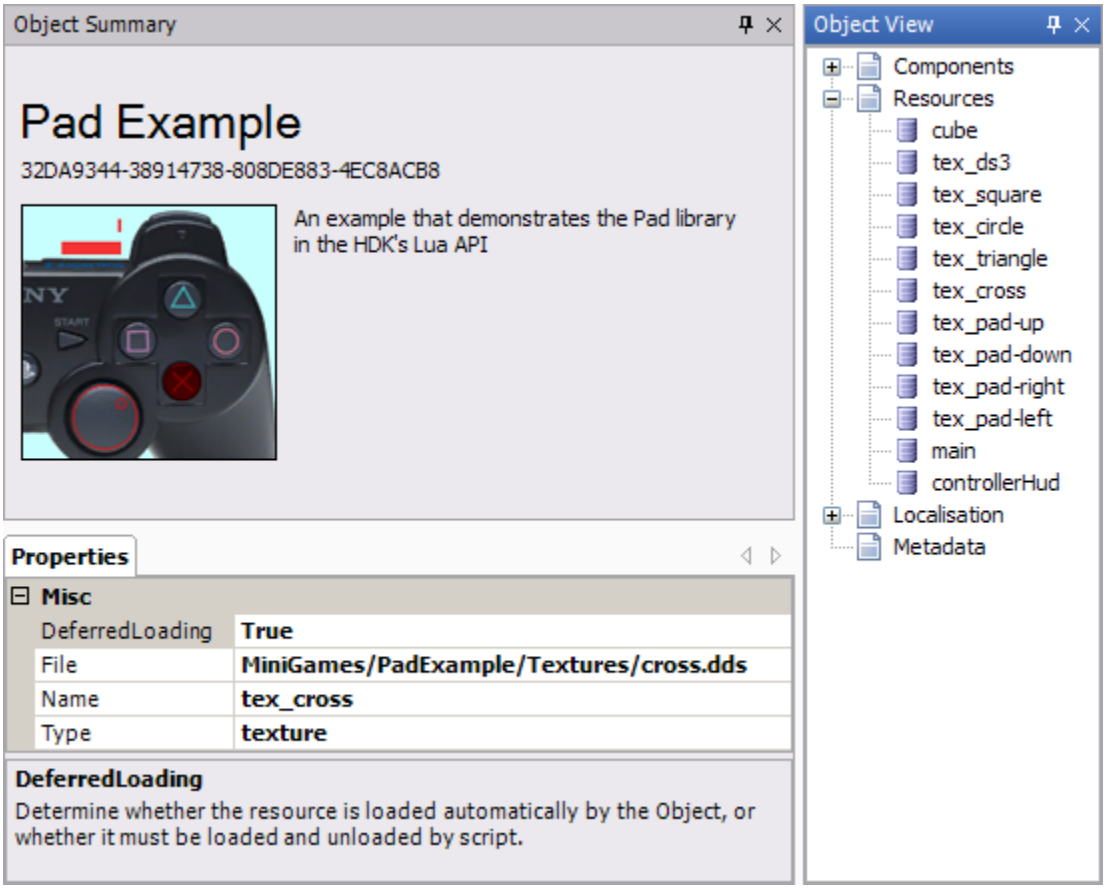
Lua Resources

A resource is a piece of data which can be read from a network source or the PlayStation®3 hard drive. The Lua API exposes a number of libraries for manipulating resources. You can:

- Read resources from the PlayStation®3 hard drive.
- Download resources over HTTP/HTTPS.
- Perform GET/POST requests over HTTP/HTTPS to interface with server-side scripts.

Manually Loading Resources

By default, resources added to an object are loaded into memory automatically when the object is initialized. The deferred loading feature allows resources to be loaded and released from memory on-demand, allowing memory to be conserved for other purposes. To prevent a resource from being loaded automatically, set the DeferredLoading field to True in the Properties panel for the resource in the Object Editor.



To load a deferred resource manually:

1. Locate the resource using `Resource.Find( Resource Name )`.
2. Use `Resource.StartLoading( resource )` to begin loading the resource.
3. Query if the resource has finished loading using `Resource.IsLoading( resource )` and/or `Resource.IsLoaded( resource )`.

You cannot use deferred loading (i.e. their resources cannot have deferred loading set to True) for the following:

- Light probes
- Custom avatar animation files (.ani)

Scripting Collision

For details on collision scripting and debugging, see:
Debugging Collision Items
Lua Collision Functions

Using Loaded Resources

Functions that take a resource as an argument, such `Resource.Run()`, `Entity.SetModel()`, `MemoryContainer.Create()` and `Xml.SetData()` can accept a loaded resource immediately. For functions that do not take a resource as an argument, you can prepare the loaded resource by calling `Resource.GetData( resource )`.

This function currently supports:

- Textures
- Images (jpg/png)
- Repertoires
- Sound Banks
- Sound Streams
- MP3 Audio

  You can use `Resource.RequestTexture()` to control the compression of a resource by setting the resource type to `Image`.

We suggest using `LoadLibrary("LocalPlayer")` for Repertoires. The `LocalPlayer` library will load all repertoire resources automatically, and will be able to distinguish between male/female. For more information on repertoires, see Avatar Animation.

Unloading Resources

When you no longer require a loaded resource, you release it from memory using `Resource.Release( resource )`. This applies to both resources with deferred loading enabled and resources that were loaded automatically.
For objects such as active items that users can place in a scene and then remove, any resource that you load is unloaded automatically by the client when it is removed from the scene.

Be careful when unloading resources that have multiple instances of the object, for example, mini-game and embedded objects. Unloading a resource for one instance makes that resource unavailable to all instances.

Requesting Resources over HTTP

It is sometimes useful for your script to be able to download files from a server. For example, downloaded XML files may contain configuration properties for a Mini-game, the top 20 players in a high score table, or any other data that you would like to generated dynamically.

In order to add support for resource handling in your script, you must first include the `Resource` library:

```
LoadLibrary( "Resource" )
```

Download requests for resources can be done in the following way:

```
function FetchXmlResource()
   url = "http://www.example.com/highscores.xml"
   xmlResource = Resource.Request(url, "file")
   while Resource.IsLoading(xmlResource) do
      print("Loading resource...")
      coroutine.yield()
   end
   if Resource.IsLoaded(xmlResource) then
      print("Resource loaded.")
   else
      print("Resource not loaded.")
   end
end

coLoad = coroutine.create(FetchXmlResource)
coroutine.resume(coLoad)

function Update()
   if coroutine.status(coLoad) == "suspended" then
      coroutine.resume(coLoad)
   end
end
```

As you can see, the code snippet above uses coroutines to facilitate the download. This is not compulsory, and you could load the resource in the main thread of the script, but coroutines are used here as they are arguably more elegant.

In addition to plain text files, textures and soundstreams can also be requested over HTTP/HTTPS. To do this, simply replace `file` with `texture` or `soundstream` when calling `Resource.Request()`.

Requesting Resources over HTTPS

In addition to HTTP download of resources, you can also use HTTP over Secure Socket Layer, or HTTPS. You may want to use this protocol if the information you want to transmit is sensitive. The only thing you need to do to enable HTTPS support in your script is to:

1. Obtain an SSL certificate.
2. Install the certificate on the web server from where your script sends and/or receives data.

List of SSL certificates supported by PS Home

The following certificates are supported by the current Home client.

Certificate
Omni Baltimore CyberTrust CA
Verisign Class 1 Public Primary CA G2
Verisign Class 1 Public Primary CA G3
Verisign Class 1 Public Primary CA
Verisign Class 2 Public Primary CA G2
Verisign Class 2 Public Primary CA G3
Verisign Class 2 Public Primary CA
Verisign Class 3 Public Primary CA G2
Verisign Class 3 Public Primary CA G3
Verisign Class 3 Public Primary CA G5
Verisign Class 3 Public Primary CA
Verisign Class 4 Public Primary CA G2
Verisign Class 4 Public Primary CA G3
EnTrust.net Secure Server CA (CPS)

GeoTrust Equifax Secure CA
GeoTrust Equifax Secure eBusiness CA-1
GeoTrust Global CA
Omni Globalsign Root CA
Omni GTE CyberTrust Global Root CA
RSA Security Root CA 1024 (Valicert Class 3 CA)
RSA Security Root CA 2048 V3
Thawte PremiumServer CA
Thawte Server CA
Valicert Class 2 CA
AAA Certificate Services CA
AddTrust External Root CA
UTN USERFirst Hardware CA
GeoTrust Equifax Secure Global eBusiness CA-1
DigiCert High Assurance EV Root CA
DigiCert Assured ID Root CA
DigiCert Global Root CA
Omni GTE CyberTrust Root CA
Verisign RSA Secure Server CA
Verisign Time Stamping Authority CA

Additionally, the following certificates will be supported from client release 1.82 onwards:

Certificate
Cybertrust Global Root
Entrust.net Certification Authority (2048)
Entrust Root Certification Authority
GlobalSign Root CA - R2
Go Daddy Class 2 CA
Starfield Class 2 CA
Starfield Services Root CA
thawte Primary Root CA
ValiCert Class 1 Policy VA
Security Communication RootCA1

Wild Cards and Domain Names

The HTTPS library that Home uses handles wild card characters for domains in a specific way. To demonstrate, we will use the example of the fictitious https://www.publisher.com domain.

You can use a wildcard in the SSL certificate issued to the domain:

e.g. *.publisher.com

This will result in all subdomains being included:

e.g. title.publisher.com and news.publisher.com

But will not include a generic truncated domain without any characters where the wildcard expects them

e.g. not https://publisher.com

The solution is to ensure you always set up the server to ensure you have subdomains, so there are no conflicts if wildcards are used,

e.g. use game.publisher.com or www.publisher.com/game, rather than https://publisher.com/game.

Parsing an XML Resource

Once you have a handle to a valid XML resource, you can parse it using Lua. The following code example demonstrates how you can print all of the elements in an XML resource:

```
function DumpXmlElementChildren( xmlParser, depth )
   while Xml.FindNextElement( xmlParser, nil ) do
      local tabs = "\t"
      for tab = 2, depth do tabs = tabs .. "\t"end
      print( tabs .. "<" .. Xml.GetElementName( xmlParser ) .. ">" )
      Xml.IntoElement( xmlParser )
      DumpXmlElementChildren( xmlParser, depth + 1 )
      Xml.OutOfElement( xmlParser )
      print( tabs .. "</" .. Xml.GetElementName( xmlParser ) .. ">" )
   end
end

function DumpXml( xmlParser )
   print( "<" .. Xml.GetElementName( xmlParser ) .. ">" )
   Xml.IntoElement( xmlParser )
      DumpXmlElementChildren( xmlParser, 1 )
   Xml.OutOfElement( xmlParser )
   print( </" .. Xml.GetElementName( xmlParser ) .. ">" )
end

xmlParser = Xml.Create()
Xml.SetData( xmlParser, xmlResource )
DumpXml( xmlParser )
```

As well as parsing the element tags, you can also parse element values and attributes. Notice that these values can be returned as specific types. This is more optimal, and means that type conversions are performed internally in C++ code rather than Lua code.

Server-Side Programming

We have seen how it is possible to download XML over HTTP and then parse it. However, it may be the case that the content of the downloaded XML needs to be dynamic. One solution to this is to write a server-side script which generates and returns XML in response to a Resource.Request call. For the purposes of this documentation, PHP will be used as an example, but you are free to use whatever server-side programming technology you like.
Instead of requesting an XML file from the server, you could request a PHP file which generates XML:

```
url = "http://www.example.com/generatexml.php"
xmlResource = Resource.Request(url, "file")
```

One use case is to request dynamic HSML (Home Screen Markup Language) from the server. In this case, generatexml.php might look as follows:

```
<XML>
   <PAGE>
      <RECT X="0" Y="0" W="1280" H="720" col="#C0C0C0" />
      <RECT X="20" Y="20" W="1240" H="60" col="#70000000" />
      <TEXT X="430" Y="20" col="#FFFFFF" size="4">HSML SCREEN</TEXT>
      <RECT X="20" Y="100" W="1240" H="600" col="#70000000" />
      <TEXT X="120" Y="290" col="#FFFFFF" size="4">
         Time and date reported by server:
      </TEXT>
      <TEXT X="140" Y="360" col="#FF0000" size="5">
         <?php echo date("M jS Y, G:i:s", time()); ?>
      <TEXT>
   </PAGE>
</XML>
```

Notice the PHP statement in the file. This statement will execute when the resource request is satisfied by the server, and the server time and date will be inserted into the file. Every time the resource is requested, the server dynamically generates a new and different HSML file.

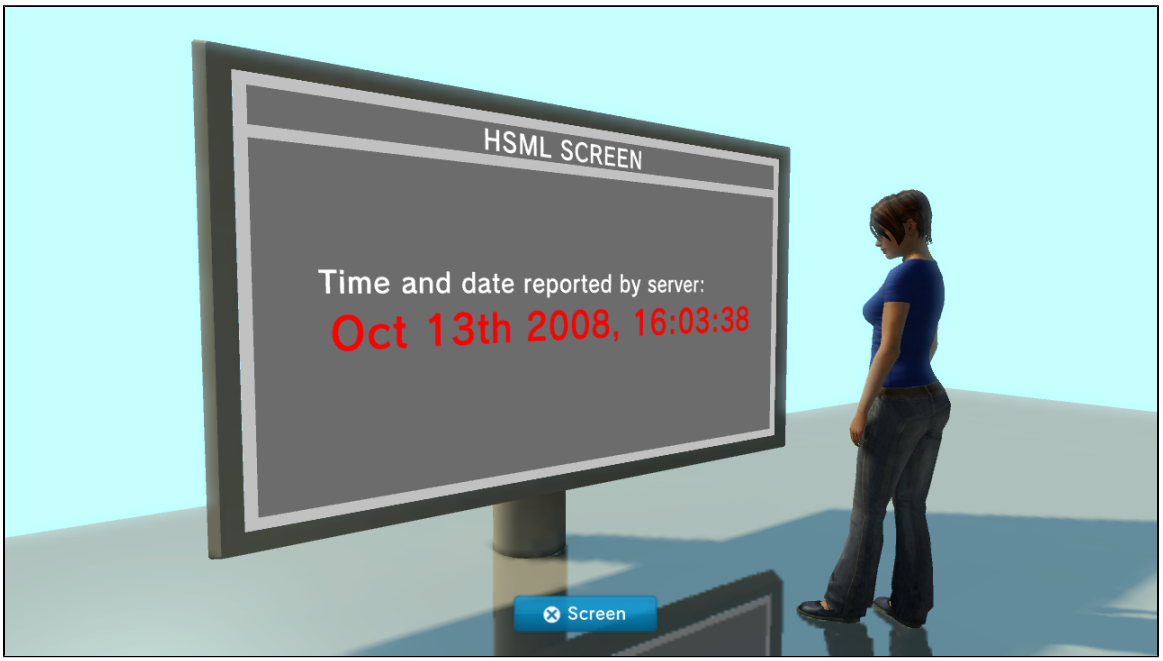
Once downloaded, the resource simply needs to be applied to a screen in the scene using code similar to the following:

```
if Resource.IsLoaded(xmlResource) then

      Screen.SetData(Scene.FindScreen("MyScreen"), xmlResource)

end
```

This should look something like the following:



Object Memory Controls

You can divide a portion of your scene's memory into defined memory slots using embedded objects. These memory slots are defined in an embedded object's object.xml and once initialized are accessed via Lua through the `MemoryContainer`, `Resource` and `ResourcePack` libraries. This feature is very useful for scenes that use deferred loading, giving you more direct control over memory allocation and helping you avoid problems with memory fragmentation.

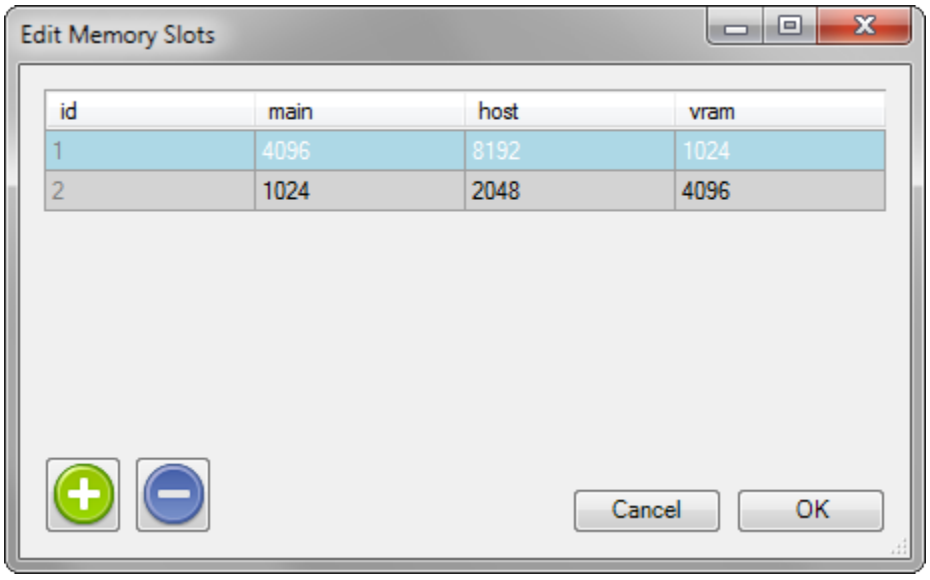
> Several embedded objects in a scene can use memory slots. Memory slots are exclusive to the object that created the slot, and can only be used by that object.

Developers can create and edit the memory slots using the Edit Memory Slots dialog in the Object Editor.

Creating Memory Slots

To create memory slots:

1. Add a Memory Slots component to your object.
2. Select Objects > Edit Memory Slots to edit the memory slots. This will display the following dialog.



3. Use the and buttons to add and remove memory slots. To edit the amount of main memory, host memory and vram which each slot reserves click on the corresponding cell and type the desired value.

> Memory values are specified in bytes and should be 16 byte aligned.

> The id values are sequential, starting from 1. They cannot be edited.

4. When you have finished editing the memory slots click the OK button to confirm your changes.

Bear in mind the following:

- Memory slots can only be used by embedded objects.
- Memory slots can only allocate a scene's memory that is unused at the embedded object's initialization.
- Several embedded objects in a scene can use memory slots, however the maximum total amount of memory slots between all objects is 12.
- Objects cannot share memory slots (objects can only use the memory slots that they create).

The client initializes an object's memory slots before running any scripts:

- If an object.xml specifies more than the maximum number of allowed slots, the object will fail to load.
- If an object.xml allocates more memory than is available in the object, the object will fail to load.

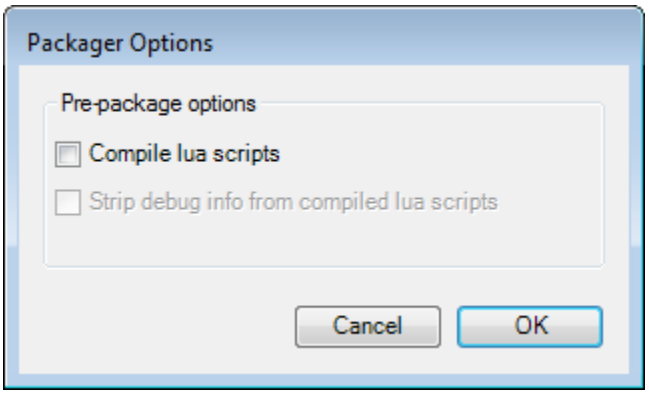
Compiling Scripts When Packaging

Running the Compiler from the Object Editor or Scene Editor

Use the Home Lua Compiler to translate your Lua scripts into binary files for faster loading times and smaller file sizes. You can compile scripts automatically within the packaging process (see Compiling Scripts in the Object Editor or Scene Editor) or manually for testing purposes before packaging.

Pre-compiling your script does not mean faster execution. Lua chunks are always compiled into bytecodes before being executed.

To compile when packaging your content, select File > Packager Options in the Object Editor or the Scene Editor to display the Packager Options dialog:



Check the Compile lua scripts box to set the option for the current file (only). When you package your content, all Lua script resources for the content are compiled and the resource links changed to point at the compiled files.

Check the Strip debug info from compiled lua scripts box to apply the -s option for the compiler every time it is run when packaging. For more information on the -s option, see Running the Compiler from the Command Line below.

If your content includes Lua scripts that are already compiled, the packager recompiles them anyway. This ensures the compiled version is up to date with the source script.

If the content includes Lua scripts that are not compiled, only the content resources in the package are updated to point at compiled files - the original content is not updated. This means that your offline content is still pointing at uncompiled Lua scripts.

The package always includes the uncompiled Lua scripts for debugging purposes.

If there are compiled scripts for your content but you do not check the Compile lua scripts box in the Packager Options dialog, then the packager does not include the compiled scripts. The resources within the packaged content are changed to point at the original source files.

Running the Compiler from the Command Line

The compiler can be run from the command line to compile specific files or load files for syntax checking. If the default options are used, then the compiler writes an output file luac.out containing the bytecodes for all source files given. You can mix text files of Lua source scripts with binary files containing pre-compiled chunks. This allows you to create a custom process that includes compiling your scripts, e.g. by creating a batch file.

You can use the compiled Lua script can for testing, but when the object or scene is packaged, recompile the Lua script using the Packager Options dialog (see above).

You can run the compiler from <HDK_INSTALL_DIR>\dev\bin\HomeLuaC.exe.

Use the following syntax:

```
HomeLuaC [options] [filenames]
```

You can use the following options:

Option	Description
-	The standard input is used as a source file.
- -	The end of options (all remaining arguments are treated as source files even if they start with "-".
- l	Produce a listing of the compiled bytecode for Lua's virtual machine. If no files are given, then the compiler loads luac.out and lists its contents.

-o <filename>	Output to the specified filename, instead of the default luac.out. It is possible to define the output file with a source file because all files are loaded before the output file is written. Therefore, be careful not to overwrite your source files this way. Only Lua script files with the extension .lua and Lua compiled files with the extension .luac are supported in PS Home. Both files must be in the same location.
-p	Load files but do not generate an output file. Used mainly for syntax checking and for testing pre-compiled chunks. Corrupted files are likely to generate errors when loaded. Lua always performs a thorough integrity test on pre-compiled chunks. Bytecode that passes this test is completely safe, meaning it will not break the interpreter. If no files are given, then the compiler loads luac.out and tests its contents. No messages are displayed if the file passes the integrity test.
-s	Strip excess information out before writing the output file. This saves some space in very large chunks, but if errors occur when running a stripped chunk, then the error messages may not contain the full information they usually do. For instance, line numbers and names of local variables are lost. Use this only as a last resort to reduce the file size, as error information is lost.
-v	Show version information.

Add options with a space in between, for example:

```
HomeLuaC.exe -l -o Active.lua
```

Profiling Lua Scripts

◆英語側にはる

> For Japanese Live Page
> 最新の日本語版ページはこちら 「Luaスクリプトのプロファイリング」

Scripted objects, such as mini-games, realtime games, and arcade games, have dynamic behavior and therefore cannot be validated in the tools. You must profile them in the runtime using the Object Profiler. You use the Object Profiler to select all objects in your scene (including scripted objects), view their rendering rates individually, and view the whole scene's rendering.

To profile Lua scripts, use the Object Profiler from the Debug Console command. You can run the Object Profiler in online or offline mode. For further information on profiling, see Profiling in the Client.

Loading the Object Profiler

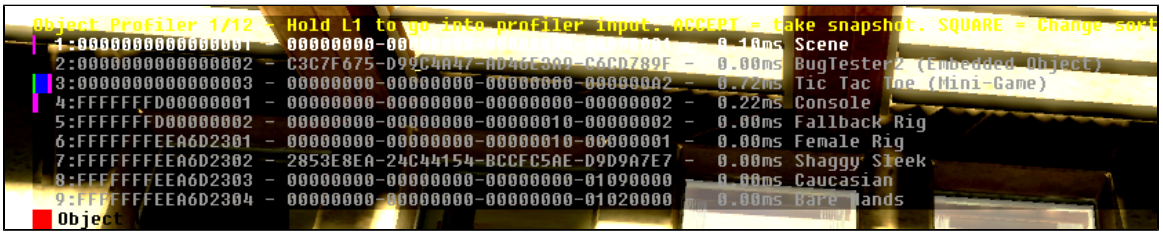
To open the Object Profiler:

1. On your controller, press SELECT.
2. Select DEV DEBUG > Open Console and then type objectProfiler 1.

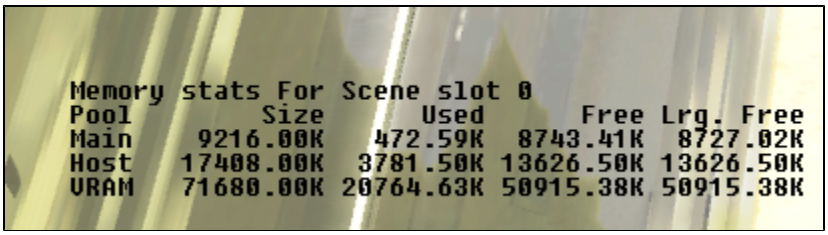


The Object Profiler generates the following information and displays it on-screen:

- A list of the components in your scene (for example, the scene itself, objects, Mini-games, arcade games, character components, furniture), as shown below:



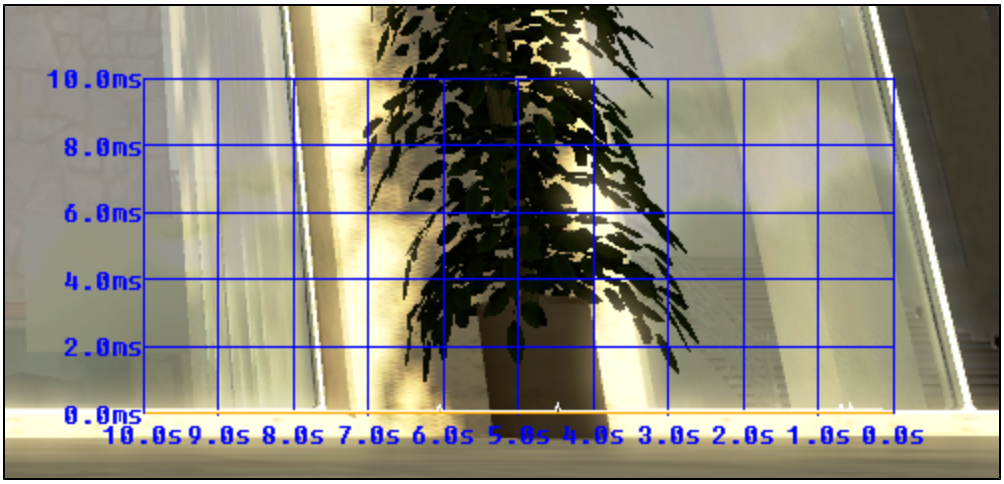
- A table showing memory statistics:



  The statistics show the total memory allowed for the scene, the amount used, and the amount remaining.

> ⚠ The total size displayed is not the amount available for you to use. Always refer to Memory Limits for the maximum memory limits. The amounts reported as used are accurate. Check them against the limits listed for that content type in General Design Guidelines.

- A graph that shows the rendering in ms (milliseconds) for the object you selected for profiling:



The X-axis is time in seconds, and the Y-axis is frame rendering in ms.

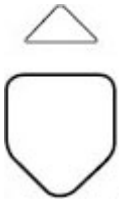
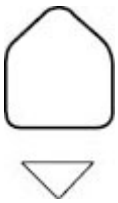
Object Profiler Commands

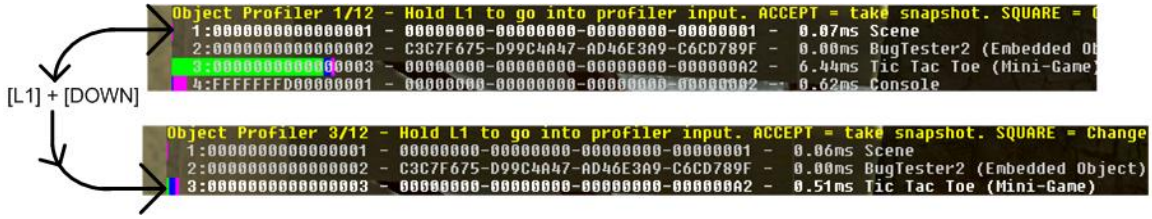
You can use the following commands:

Command	Valid Parameter(s)	Default Value(s)	Description
objectProfiler	false, 0, off, true, 1, on	False	Enable/disable the Object Profiler.
To Customize the Memory Stats Table			
objectProfiler.MemStartx	0 to 1280	32	Horizontally sets the position of the Memory Stats Table, in pixels.
objectProfiler.MemStartY	0 to 720	200	Vertically sets the position of the Memory Stats Table, in pixels.
To Customize the Graph			
objectProfiler.VertIntervalMs	0 to MAXFLOAT	2.0	Set the profile graph's vertical axis spacing (in milliseconds).
objectProfiler.HorzIntervalMs	0 to MAXFLOAT	1.0	Sets the profile graph's horizontal axis spacing (in milliseconds).
objectProfiler.GraphStartY	0 to 720	200	The position, in pixels, for the top of the Y-axis.
objectProfiler.GraphEndY	0 to 720	32	The position, in pixels, for the bottom of the Y-axis.
objectProfiler.GraphStartX	0 to 1280	750	The position, in pixels, for the left-edge of the X-axis.
objectProfiler.GraphEndX	0 to 1280	1125	The position, in pixels, for the right-edge of the X-axis.
objectProfiler.ObjectListMaxLineCount	0 to MAXINT	9	The number of objects currently running in the scene to be displayed in the on-screen list (if the number of objects is greater than this number, the list scrolls).
objectProfiler.RangeMs	0 to MAXFLOAT	10.0	The maximum value of the vertical axis of the profile graph. Altering this scales the traces vertically.
objectProfiler.GraphColourA	0.0 to 1.0	1.0	This allows you to change the alpha in your graph lines.
objectProfiler.GraphColourB	0.0 to 1.0	0.4	This allows you to change the blue color in your graph lines.
objectProfiler.GraphColourG	0.0 to 1.0	0.4	This allows you to change the green color in your graph lines.
objectProfiler.GraphColourR	0.0 to 1.0	0.6	This allows you to change the red color in your graph lines.

Controlling the Object Profiler

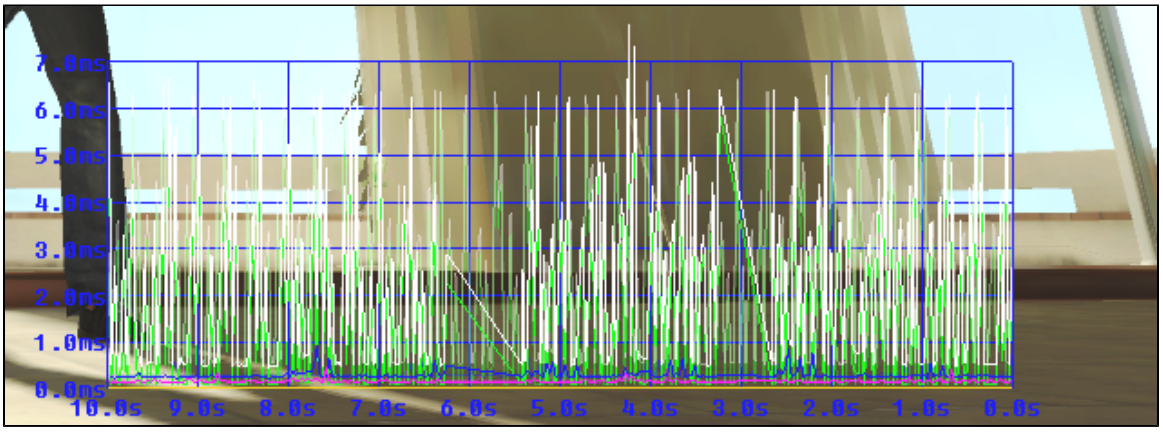
With the Object Profiler enabled, you can control certain functions by pressing and holding L1 on the connected controller and then pressing one of the following buttons at the same time:

- L1 +  or L1 +  to select the previous/next object in the list to profile:



- L1 + ACCEPT: Burn a static snapshot of the profile trace into the graph. Taking a new snapshot clears the last one taken. To completely remove a snapshot from the graph, reload HomeDeveloper.self.

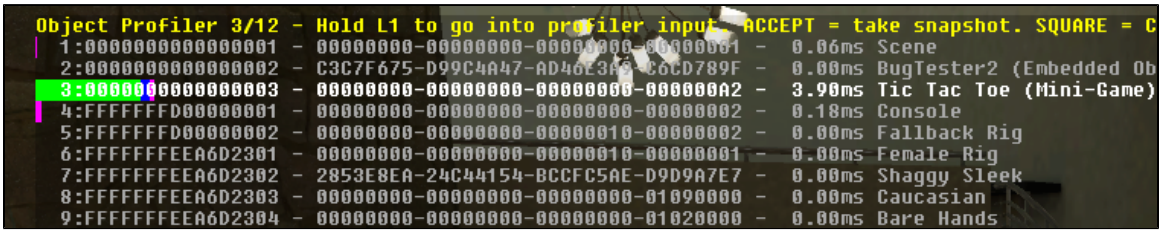
  The following graph shows the live graph, in white, and the static snapshot in green:



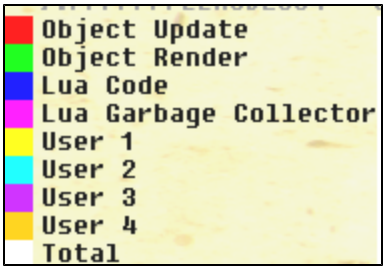
- L1 + SQUARE: Select the next sort mode for the object list. The list can be sorted on instance ID, object ID, object name or total profile time.

Object Profiler Traces

When you open the Interactive Object Profiler, the list of components on the left shows a colored bar (the trace bar) for those items that require a lot of rendering time:



The trace bar shows in realtime those objects consuming the most resources, and thus time, to render. The following list of scene components explains what each of the colors means:



- Object Update: Every object is built from components (in the Object Editor). Internally, PS Home calls an Update function for each object, for each component that it contains. An expensive Object Update trace probably indicates that your script is spawning too many Sprite or Entity instances.
- Object Render: The Object Render shows the cost of executing the Render function of each component in an object. An expensive Object Render trace probably indicates that your script is trying to submit too many primitives to a Renderer instance (lines, rectangles, circles, sprites or text).
- Lua Code: The Lua Code trace shows the total amount of time that the Lua interpreter spends executing code in your object. This time includes any time that Lua functions spend in native code. For example, the Lua API function `Vector4.Create` has a native C++ implementation that is callable from Lua. The execution time of this native code called from Lua functions is included in the total.
- Lua Garbage Collector: The amount of time per frame spent garbage collecting allocations made by your object's script. The Lua garbage collector is run once per frame.
- User 1 to 4: 4 user defined traces. Specific parts of an object's script can be profiled by wrapping them with calls to `Environment.DebugProfileBegin` and `Environment.DebugProfileEnd`. These functions take an integer value of 1 to 4, which adds the execution time of the delimited code block to the corresponding trace in the profile graph.

Optimization Tips

- Where possible, allocate scratch Lua API objects. Lua garbage collection is expensive, so avoid allocating objects on a per frame basis. If possible, make allocations in the global execution space instead. For example, if your script's `OnUpdate` function requires a Vector4 instance, call Vector4. Create and assign the result to a global variable, rather than to a variable local to the update function.
- Allocate HDK Lua API objects sparingly.
- For objects that need to send and receive messages (using the Network component), try to send the least amount of information to allow your object to function as required. For example, in a chess game object, every time a player makes a move, you could send the entire board state, or just the move that was made. The latter is more efficient. It takes less code to build and decode the message, and less data needs to be sent over the network.

Scripting Tips

LuaSrcDiet to Reduce Lua Source File Size

To reduce this size of your Lua source files, you can use LuaSrcDiet from Luaforge.net. The program removes unnecessary whitespace and comments, optimizes constant tokens, and renames local variables to shorter names.

LuaSrcDiet can be found at http://luaforge.net/projects/luasrcdiet/.

Releasing Lua Source to Save Memory

If you have a large Lua source code and want to regain memory after loading the code, call the following:

```
local myCode = Resource.Find( 'game.lua' )

Resource.Release( myCode )
```

> ⚠ If you have multiple instances of games where the games are in the same memory slot, write your code to release the Lua memory only after each instance of the game has loaded.

Synchronizing Animation in Scenes

To create animation, you script it into a scene. You can create animation that appears for all users in a scene, for example you

can have a Sword of Eternal Conduction floating in the air above the Stone of Sacred Sanctity, by using mini-games that users cannot join.

Mini-games run on all users' systems regardless if someone has joined the game or not.

Bear in mind the following:

- Place the trigger region far away from the areas that users can access.
- You can send messages to users not in the game by not specifying a user ID in the send message.
- To establish an owner, use the welcome messages. When the mini-game asks a user in the lobby to send a welcome message, it perceives that user as the owner.

Debugging Lua Script using SLED

Lua's `print` function is very useful, but if you need a more powerful debugging tool, use the SCE Lua Editor and Debugger (SLED) tool. The HDK currently ships with SLED v 4.0.2. SLED can be found in `dev\External\SLED`.

SLED provides the following features:

- Lua script management (including creation, editing and compiling). See Creating and Managing Lua Scripts.
- Project creation and management. See Creating and Managing Projects.
- Script execution control. See Controlling Script Execution.
- Breakpoints. See Setting Breakpoints.

> Some tools may prompt you to install updates to more recent releases, such as SLED v 4.0.2. Keep in mind that the HDK supports only the version packaged with the HDK or indicated in the Installing the HDK. If you update the software for these tools to a version other than the version provided or specified, the HDK may not work as expected and content created may not pass Quality Assurance (QA).

To debug or profile your scripts using SLED, you need to:

- Use the Target Manager to connect to a target machine. See Launching the .self in Debug Mode.
- Launch, on the target machine, the .self associated with your Lua scripts. See Launching the .self in Debug Mode.
- Configure SLED to connect to the target machine. See Launching the .self in Debug Mode.
- Set up a SLED project and add your script files to it. See Creating and Managing Projects.
- Prepare your scripts for debugging. See Preparing Scripts for Debugging.
- Connect SLED to the target machine. See Launching the .self in Debug Mode.

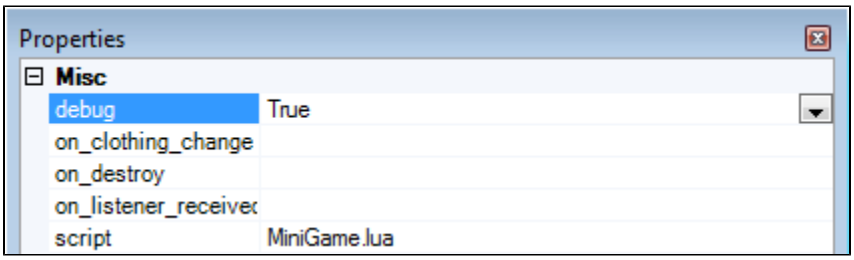
> Detailed user guides for SLED are available in PDF format within your HDK install directory under `dev/External/SLED/Doc`.

Debug Library and Public Library

All `Debug` functions provided in the HDK Lua API are for debug purposes only. You cannot include them in content when submitting it for publishing in the live environment.

Using the Debug Property

To use `Debug` Lua functions, you must set the Lua Environment component's `debug` property to `True` in the Object Editor:



This setting also gives scripts access to debug functions from other libraries (such as `Object.DebugRun()`).

Setting this property to `False` disables the `Debug` library and any other debug function.

Functions Available in Debug Mode

The following functions are available in debug mode:

Function	Debug Mode	Public Mode	Deprecated
assert	Y (must set the Lua Environment compoent's debug property to True)	N	
collectgarbage	Y	Y	
coroutine.create	Y	Y	
coroutine.resume	Y	Y	
coroutine.running	Y	Y	
coroutine.status	Y	Y	
coroutine.wrap	Y	Y	
coroutine.yield	Y	Y	
debug.debug	Y (must set the Lua Environment compoent's debug property to True)	N	
debug.getfenv	Y (must set the Lua Environment compoent's debug property to True)	N	
debug.gethook	Y (must set the Lua Environment compoent's debug property to True)	N	
debug.getinfo	Y (must set the Lua Environment compoent's debug property to True)	N	
debug.getlocal	Y (must set the Lua Environment compoent's debug property to True)	N	
debug.getmetatable	Y (must set the Lua Environment compoent's debug property to True)	N	
debug.getregistry	Y (must set the Lua Environment compoent's debug property to True)	N	
debug.getupvalue	Y (must set the Lua Environment compoent's debug property to True)	N	
debug.setfenv	Y (must set the Lua Environment compoent's debug property to True)	N	
debug.sethook	Y (must set the Lua Environment compoent's debug property to True)	N	
debug.setlocal	Y (must set the Lua Environment compoent's debug property to True)	N	
debug.setmetatable	Y (must set the Lua Environment compoent's debug property to True)	N	
debug.setupvalue	Y (must set the Lua Environment compoent's debug property to True)	N	
debug.traceback	Y (must set the Lua Environment compoent's debug property to True)	N	
dofile			Y
error	Y (must set the Lua Environment compoent's debug property to True)	N	
gcinfo	Y	Y	
getfenv	Y	Y	
getmetatable	Y	Y	
io.close			Y
io.flush			Y
io.input			Y
io.lines			Y
io.open			Y
io.output			Y
io.popen			Y
io.read			Y
io.type			Y
io.write			Y
ipairs	Y	Y	

load	Y	Y	
loadfile			Y
loadstring			Y
math.abs	Y	Y	
math.acos	Y	Y	
math.asin	Y	Y	
math.atan	Y	Y	
math.atan2	Y	Y	
math.ceil	Y	Y	
math.cos	Y	Y	
math.cosh	Y	Y	
math.deg	Y	Y	
math.exp	Y	Y	
math.floor	Y	Y	
math.fmod	Y	Y	
math.frexp	Y	Y	
math.ldexp	Y	Y	
math.log	Y	Y	
math.log10	Y	Y	
math.max	Y	Y	
math.min	Y	Y	
math.mod	Y	Y	
math.modf	Y	Y	
math.pow	Y	Y	
math.rad	Y	Y	
math.random	Y	Y	
math.randomseed	Y	Y	
math.sin	Y	Y	
math.sinh	Y	Y	
math.sqrt	Y	Y	
math.tan	Y	Y	
math.tanh	Y	Y	
module			Y
newproxy	Y	Y	
next	Y	Y	
os.clock			Y
os.date			Y
os.difftime			Y
os.difftime			Y

os.exit			Y
os.remove			Y
os.rename			Y
os.setlocale			Y
os.time			Y
package.loaded			Y
package.loadlib			Y
package.seeall			Y
pairs	Y	Y	
pcall	Y	Y	
print	Y	Y	
rawequal	Y	Y	
rawget	Y	Y	
rawset	Y	Y	
require			Y
select	Y	Y	
setfenv	Y	Y	
setmetatablev	Y	Y	
string.byte	Y	Y	
string.char	Y	Y	
string.dump	Y	Y	
string.find	Y	Y	
string.format	Y	Y	
string.gfind	Y	Y	
string.gmatch	Y	Y	
string.gsub	Y	Y	
string.len	Y	Y	
string.lower	Y	Y	
string.match	Y	Y	
string.rep	Y	Y	
string.reverse	Y	Y	
string.sub	Y	Y	
string.upper	Y	Y	
table.concat	Y	Y	
table.foreach	Y	Y	
table.foreachi	Y	Y	
table.getn	Y	Y	
table.insert	Y	Y	
table.maxn	Y	Y	

table.remove	Y	Y	
table.setn	Y	Y	
table.sort	Y	Y	
tonumber	Y	Y	
tostring	Y	Y	
type	Y	Y	
unpack	Y	Y	
xpcall	Y	Y	



Remove Debug functions from scripts before packaging and submitting content to the Content Delivery System (CDS) for publishing to the live environment. Using Debug functions in a submitted script will generate Lua errors in QA which will fail the content.

Launching the .self in Debug Mode

To enable the Home runtime ability to communicate with SLED over the network, you must launch HomeDeveloper.self from the Target Manager, and specify the following command line parameter:

```
--luadebugger <BUILD_FOLDER> --luaprotocol <deci3|tcp> --luaport <PORT>
```

The luaprotocol switch specifies the protocol for the target to which you are connecting in SLED, and the luaport switch specifies the port number for that target. Test Kits support the TCP protocol and Reference Tools support both DECI3 and TCP. If you do not specify the luaprotocol switch, it defaults to deci3. If you do not specify the luaport switch, it defaults to 8530.

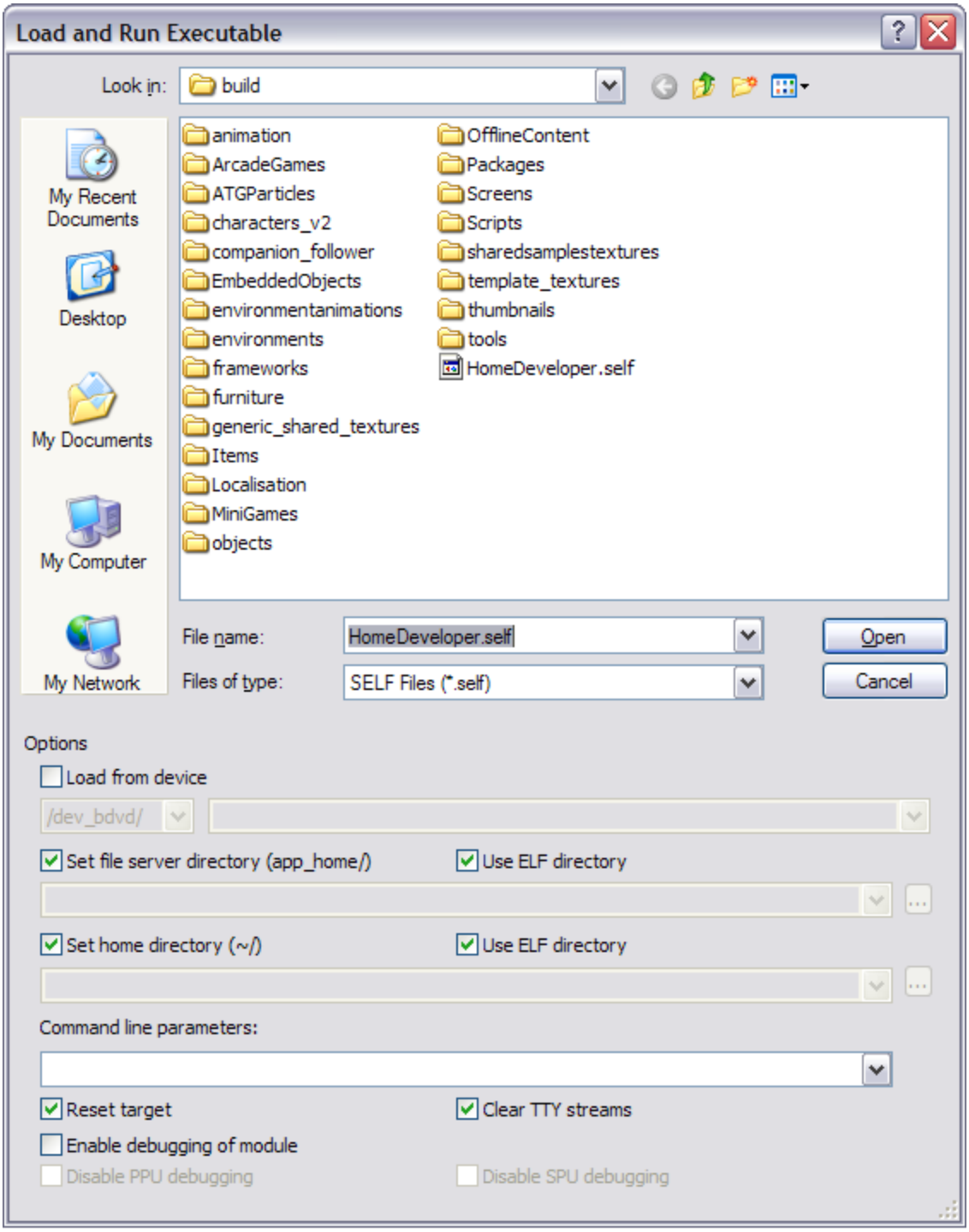
For example, launching with object number 00000000-00000000-00000000-000000a3:

```
--game 00000000-00000000-00000000-000000a3 --luadebugger C:\HomeHDK\build
      --luaprotocol tcp --luaport 11111
```

Adding your Target to SLED

To add your target to SLED:

1. Open the Target Manager and connect to your target by selecting your target and either clicking on the Connect button  on the toolbar or selecting Target > Connect.
2. Select Target > Load and Run Executable.
3. Set the options as follows:



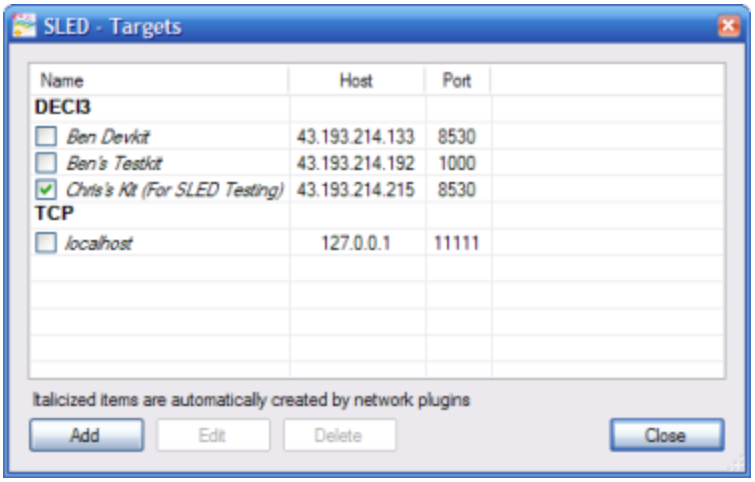
Specify the Command line parameters as follows:

```
--luadebugger [HDK build root] --sceneId [Name of scene in LocalScenelist.xml]
```

For example:

```
--luadebugger C:\HomeHDK_165\build --sceneId maya2012env
```

4. Ensure that you are in the correct build folder, and then launch HomeDeveloper.self.
5. When the client is loaded, open SLED.
6. Disconnect from the Target Manager.
7. In SLED, select Target > Manage Targets and ensure that your target is displayed:



8. Select the target you require and click Close.

Connecting and Disconnecting the Target

To connect to the target, click on the Connect button  on the toolbar or select Debug > Connect.

To disconnect from the target, click on the Disconnect button on the toolbar or select Debug > Disconnect.

Connection Issues

If you're experiencing any issues, check the following:

- If you connect to your target with DECI, you will need to ensure you first disconnect from the target in the Target Manager, then connect from SLED.
- If connecting with TCP, ensure that you supply the correct IP address for the test kit. If in doubt, check the IP address in the Network Settings section of the XMB.
- Due to a limitation in SLED, core dump targets in your Target Manager may cause issues with SLED when connecting to a target. If this occurs, remove the core dump target and retry. This issue should be fixed in a future release.

Creating and Managing Projects

A project is a group of Lua script files that, collectively, control an element of game logic and flow. Such an element may be small, like a mini-game, or large, like an entire game.

> SLED no longer sets up a project for you automatically. To debug your scripts, you must set up a SLED project and add your script files to it.

Project files reside in the project directory that you select, and are the heart of a project; they define its scope and characteristics. The SLED project file stores names and locations of the files comprising the project, with the following information stored in a temporary user settings file:

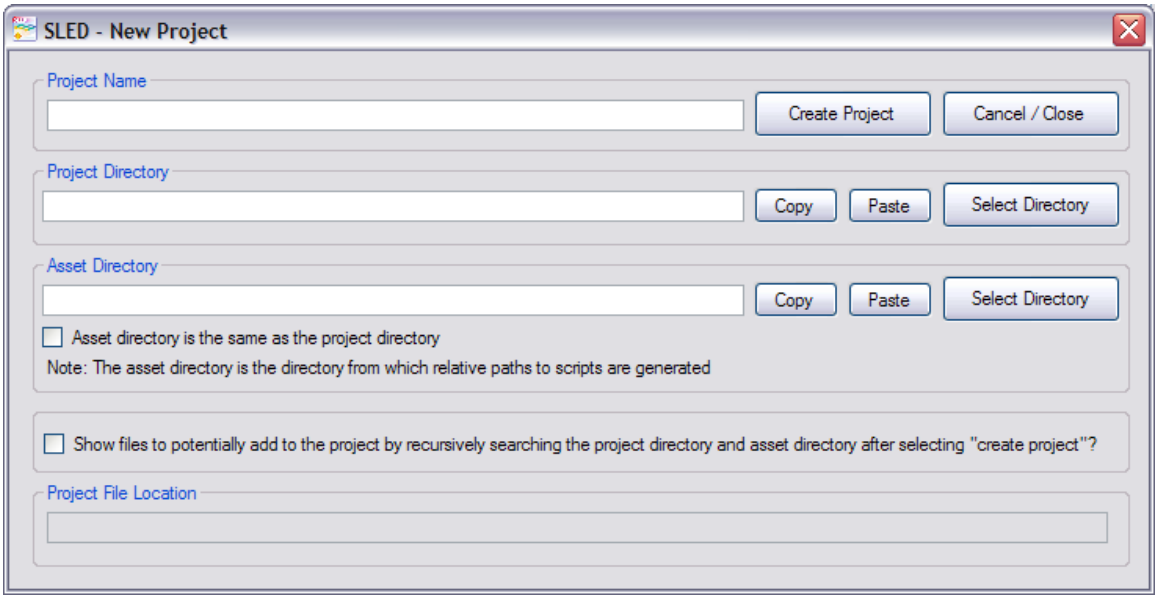
- Functions used in the project
- Breakpoint locations and conditions

The default project file extension is .spf. The SLED user settings file extension is .sus.

A project's Lua script files may reside anywhere. While the Lua script files do not have to reside in the Asset directory, their paths must match up between the target machine and SLED to enable breakpoints to be hit.

Creating a New Project

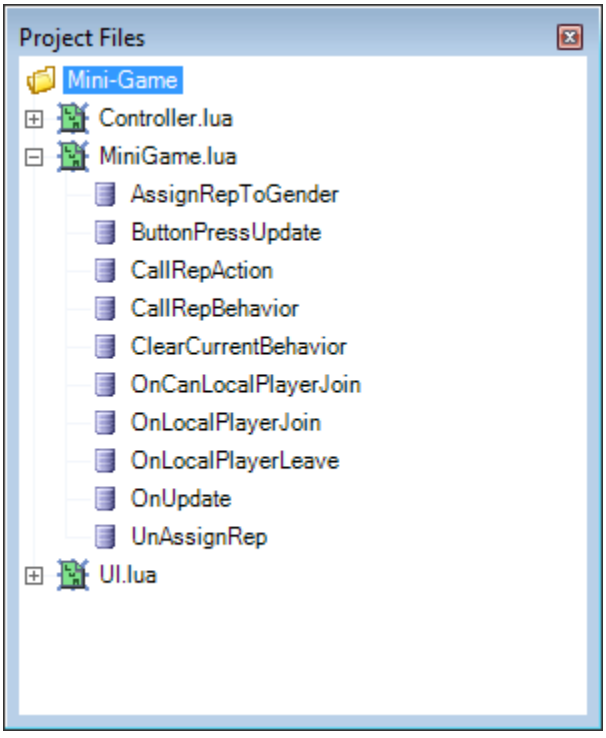
1. Open SLED.
2. Select File > Project > New. The New Project dialog opens:



3. Enter a descriptive name for the project in the Project Name field.
4. Enter the directory in which the SLED project file is saved in the Project Directory field. Click Select Directory to browse to the correct location. You may find it helpful to set this to your build directory, but it can be located anywhere.
5. Enter the build directory passed to the client using --luadebugger in the Asset Directory field. Click Select Directory to browse to the correct location. The directory must match your file serving directory as SLED uses it to locate the script files loaded by the client.
6. Click Create Project to create the project file.

Opening an existing Project

Select File > Project > Open and select the project required. The files contained within the project are displayed in the Project Files panel:



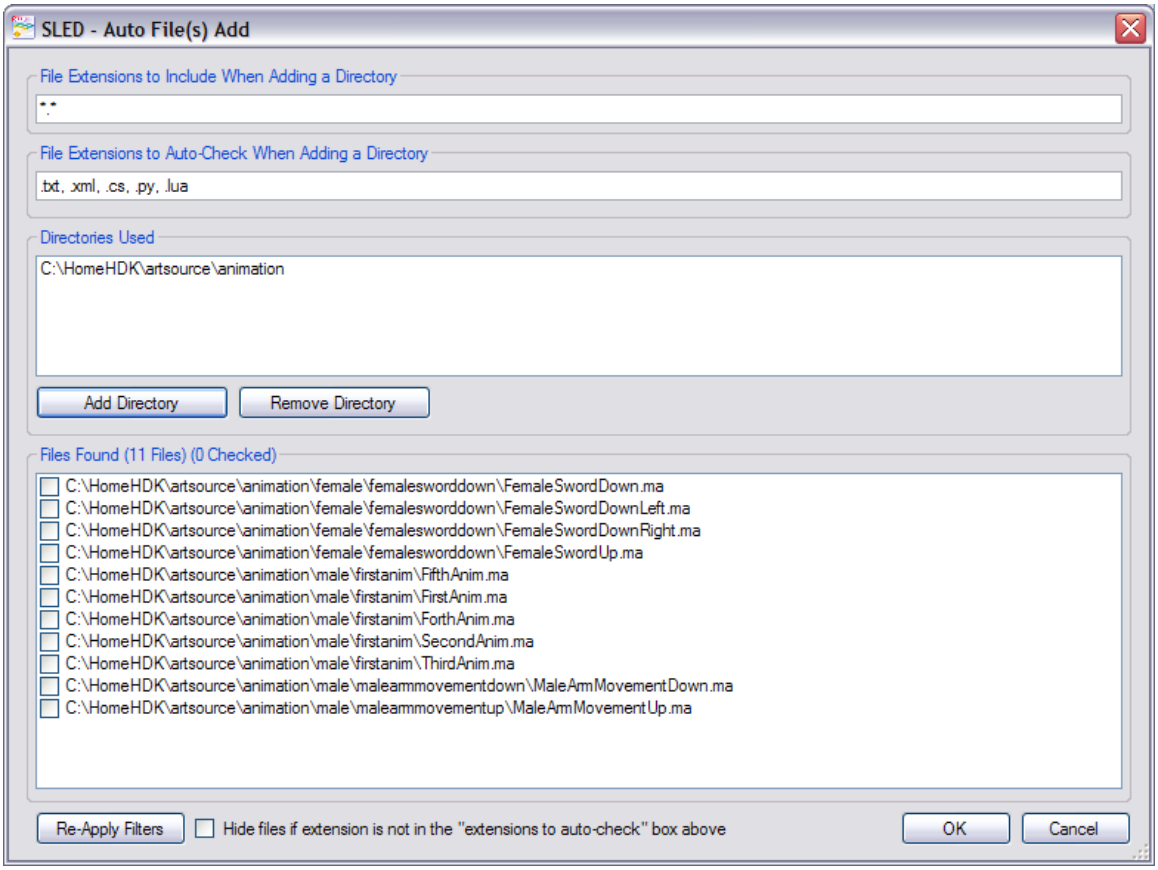
If the Project Files panel is not displayed in SLED, select Window > Project Files.

To display script file within a project, double-click on the name of the script in the Project Files panel.

Adding Existing Scripts to a Project

To add existing files to a project:

1. Ensure that the project that you want to add script files to is open in SLED.
2. Right-click on the top level project folder in the Project Files panel and select Add Files... from the pop-up menu displayed. The Auto File(s) Add dialog opens:



3. Click Add Directory to add directories for searching. SLED automatically searches through the added directories and displays the files found in the Files File box. Those files with the extensions entered in the File Extensions to Auto-Check When Adding a Directory field are checked by default.
4. To select a file to add, select the box next to it.
5. When you have finished selecting the files required for the project, click OK.

After you have added files to your project and the object is running in the client, you can breakpoint and step through your scripts in SLED.

Adding New Scripts to the Project

To add a new Lua script file to a project:

1. Ensure that the project is open in SLED.
2. Open the script file in a script editing window.
3. Select File > Project > Add File.

Removing Scripts from a Project

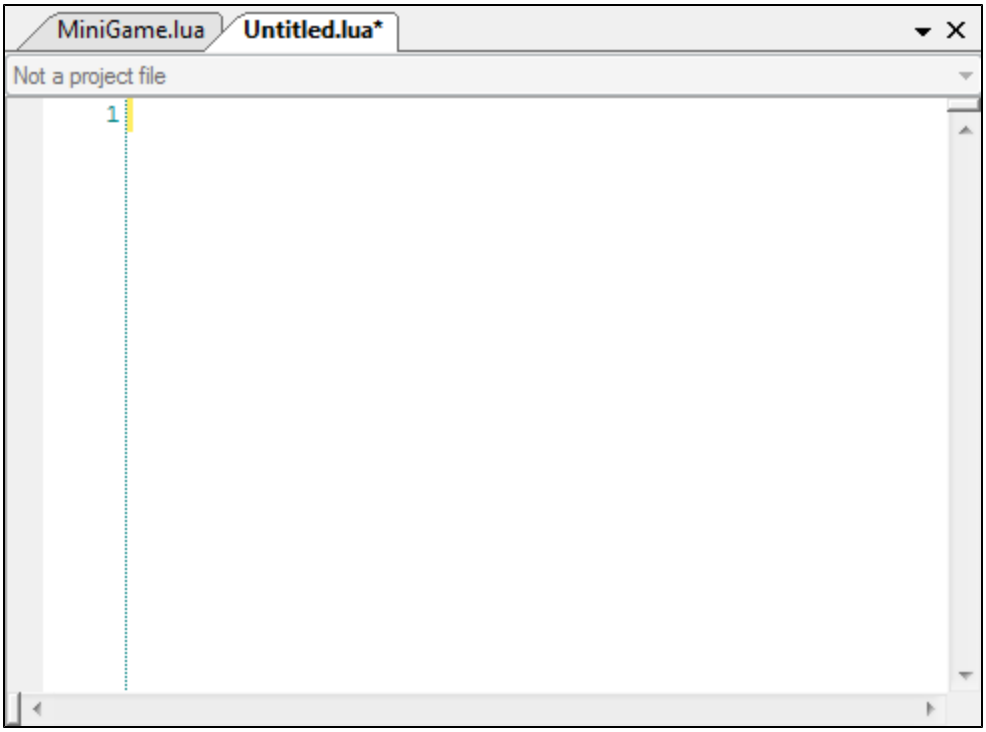
To remove a script from a project, right-click on the name of the script in the Project Files panel and select Remove from the pop-menu displayed. The script is removed from the project but is still available in the location to which it was saved.

Creating and Managing Lua Scripts

SLED allows you to create Lua scripts from scratch or to editing existing scripts. It also has a built-in Lua compiler, allowing you to compile Lua scripts from within SLED for the supported platforms.

Creating and Editing Script Files

To create a Lua script, select File > New > Lua. A new script editing window is displayed:



To edit a Lua script, double-click on its name in the Project Files panel. The script will open in a new script editing window.

As you type, SLED parses and highlights your syntax, making your script more easily readable. SLED also displays an asterisk next to the name of your script file when the content of the file has changed since the file was last saved.

SLED also checks your syntax each time you save your script file and displays any errors in the Syntax Errors panel. If the Syntax Errors panel isn't currently displayed, select Window > Syntax Errors. Double-clicking on an error in the Syntax Errors panel takes you to the source of the error in the script in a script editing window.

To save a file you have created or edited, do one of the following:

- To save the script file, click the Save button on the toolbar or select File > Save.
- To save the script file under a new file, click the Save As button on the toolbar or select File > Save As.
- To save all of the currently open script files, click the Save All button on the toolbar or select File > Save All.

For more information about adding new Lua scripts to a project (or removing existing scripts from a project), see Creating and Managing Projects.

> To debug your scripts, you must set up a SLED project and add your script files to it. You must also enable debugger support in your script. See Preparing Scripts for Debugging.

Compiling Script Files

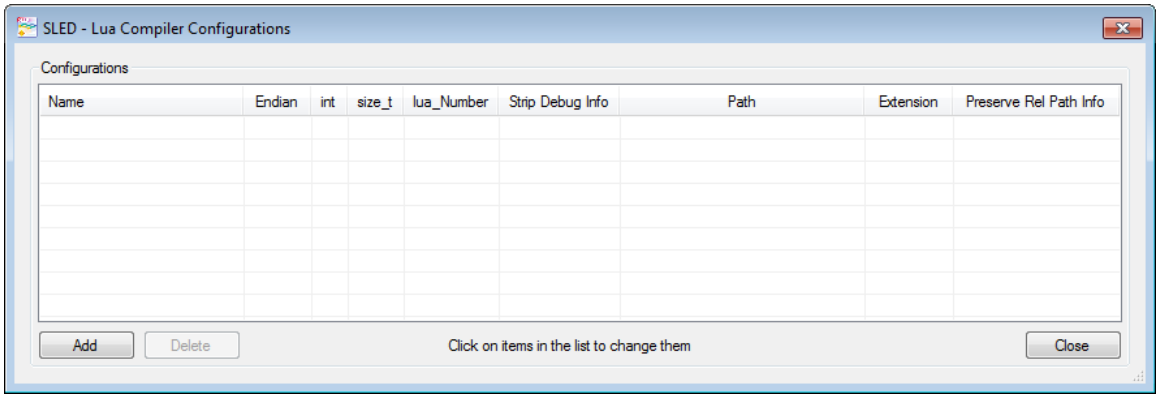
SLED includes a built-in Lua compiler, so you can compile Lua scripts from within SLED. You can also adjust the size (in bytes) of data types such as `int`, `size_t`, and `lua_Number`. You can have multiple compiler configurations for a project and the built-in compiler performs validation as well.

> You must add at least one compiler configuration to be able to compile your Lua scripts (see below).

To compile Lua scripts, open the project containing the scripts and either click the Compile button on the toolbar or select Lua > Compile. You can view the compilation output in the Output panel (select Window > Output if the Output panel isn't displayed).

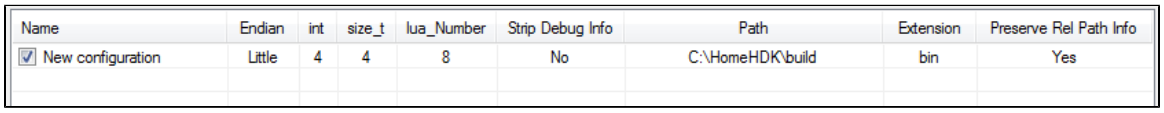
Adding a New Compiler Configuration

1. If you have not yet created a compiler configuration, either click the Compile button on the toolbar or select Lua > Compile. A message is displayed, prompting you to set up a compiler configuration. When you click OK, the Lua Compiler Configurations dialog is displayed:



If you have created a compiler configuration before, select Lua > Compiler Settings to display the Lua Compiler Configurations dialog.

2. Click Add. A default Lua compiler configuration is added:



3. To change any value, click the field to select it, then enter or select the desired value. Enter new values for fields such as Name and Extension, and select from allowed values for fields such as Endian, int, size_t, lua_Number, and Step Debug Info. When you click Path, a dialog appears that allows you to browse to the desired path in which to place compiled Lua script output.
4. When you are satisfied with the configuration, click Close. SLED saves the configuration.

Managing Existing Compiler Configurations

1. Select Lua > Compiler Settings to display the saved configurations in the Lua Compiler Configurations dialog.
2. Check the box next to the name of the configuration that you're interested in.
3. Do one of the following:

    - To change the value of any configuration fields, click the field to select it, then enter or select the desired value.
    - To add a new configuration, click Add and follow the steps in Adding a Compiler Configuration.
    - To delete a configuration, click on the row containing the configuration to select it, then click Delete.

4. When you are finished, click Close to save your changes.

Preparing Scripts for Debugging

To enable the debugger support in your script, you must make some minor edits to the code:

1. Load the Debug library. Add the following line to the top of your script:

```
LoadLibrary("Debug") -- Required to access Debug.* functions
```

2. Register the script with the debugger. Add the following lines below the point where you load your libraries:

```
Debug.DebuggerRegister()Debug.DebuggerSetupProject()
```

An example Lua script may look something like this:

```
LoadLibrary("Debug") -- Required to access Debug.* funcs

print("======= STARTING DEBUGGER TEST =======")
Debug.DebuggerRegister()      -- Called 1st
Debug.DebuggerSetupProject() -- Called 2nd

-- Your code and stuff to debug hereon
function Main_Loop()
      print("Press F9 on this in SLED to set breakpoint.")
end
```

⚠ The order in which you call the two Debug functions is important.

For more information on the Debug library, see Debug Library and Public Library.

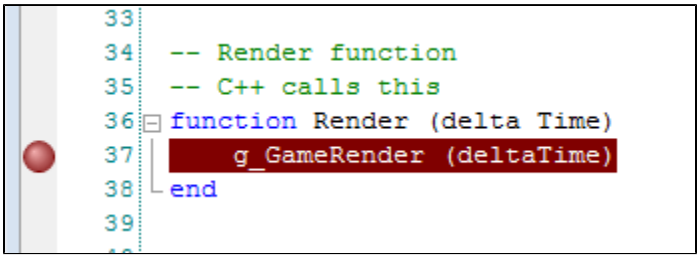
Setting Breakpoints

You can set a breakpoint to pause script execution on a certain line. Setting a breakpoint assures that execution always pauses on that line. You can also specify a condition so that execution pauses only if the condition is met.

When you connect to the target machine, the maximum number of breakpoints that you can set will be displayed in the Output panel. You cannot set more than the maximum number of breakpoints. The executable on the target machine hits the breakpoints only if SLED is connected to the target machine.

Setting Breakpoints

To set a breakpoint, point the cursor at the gray-shaded, vertical margin (the breakpoint bar) on the left side of the script editing window. Align the cursor with the line on which to set the breakpoint, and click once. A red sphere appears where you've clicked, indicating that a breakpoint now exists at that line:



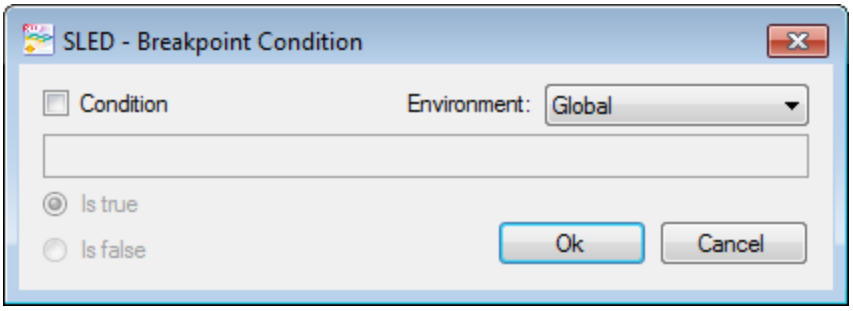
You can toggle the breakpoint on or off by either clicking on the breakpoint icon, or by placing your cursor on the same line as the breakpoint and selecting Debug > Toggle Breakpoint.

When a breakpoint halts script execution, SLED opens the script file and indicates the line where the breakpoint occurred. Some breakpoints will not be hit, depending on where you place them. Those breakpoints
are colored differently when SLED connects to the target machine. For example, a breakpoint on a comment line will not be hit.

Setting a Breakpoint with Conditions

To set a breakpoint so that it halts script execution only if certain conditions are met:

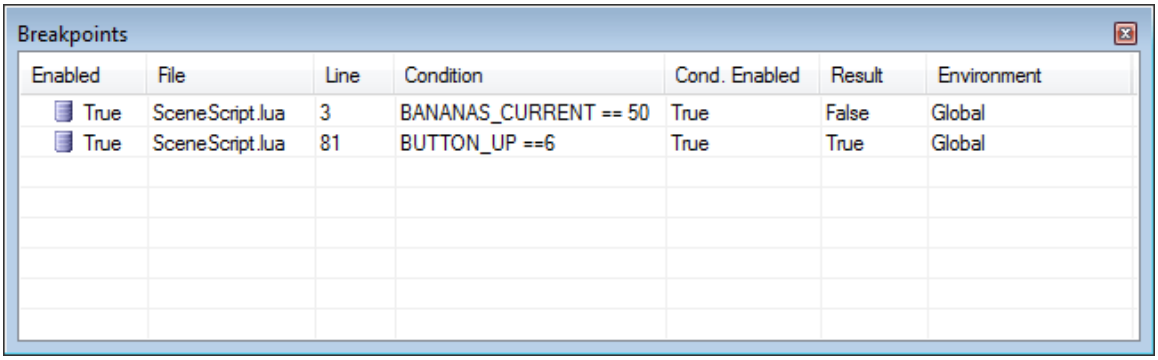
1. Set the breakpoint (see above).
2. Right-click on the breakpoint and select Condition from the pop-up menu displayed. The Breakpoint Condition dialog is displayed:



3. Check the Condition box.
4. Enter a valid Lua expression to evaluate. The syntax is highlighted as you enter the Lua expression.
5. Click OK. The condition is set.

Managing Breakpoints

The Breakpoint panel displays all of the currently set breakpoints and their attributes. If the Breakpoint panel isn't already displayed in SLED, select Window > Breakpoints:



Right-click on any breakpoint row in this panel to display a pop-menu that allows you to remove or disable the breakpoint, or to modify or disable any condition associated with the breakpoint.

Disabling a Breakpoint

You can leave a breakpoint in place but inactive by doing one of the following:

- Right-click on the breakpoint icon , and select Disable Breakpoint from the pop-up menu displayed.
- Display the Breakpoints panel (see above), right-click on the row representing the breakpoint you want to disable and select Disable from the pop-up menu displayed.

Removing Breakpoints

To remove breakpoints, do one of the following:

- Click on the the breakpoint icon .
- Right-click over the breakpoint icon and select Remove Breakpoint from the pop-up menu displayed
- Right-click over the row representing the breakpoint that you want to remove in the Breakpoints panel (see above) and select Remove from the pop-up menu displayed.
- To remove all the breakpoints applied to the script, right-click on any of the breakpoint icons , and select Remove All Breakpoint from the pop-up menu dispalyed.
- To remove more than one breakpoint at the same time, select the rows representing the breakpoints that you want to remove in the Breakpoints panel, right-click and select Remove All Selected from the pop-up menu dispalyed.

Controlling Script Execution

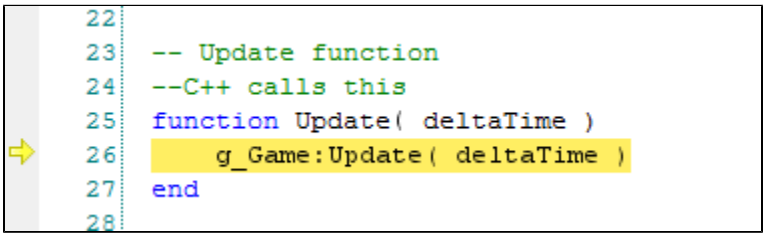
SLED provides an array of powerful and flexible tools to help you debug your scripts. You can pause and resume script execution, and tell SLED to continue executing the script in certain increments with the execution paused. You can also set breakpoints.

Pausing and Resuming

SLED allows you to pause and resume script execution.

To pause script execution, click the Stop debugging button on the toolbar or select Debug > Stop.

When you pause execution, SLED opens the script file in which execution halted, and highlights the line where the pause occurred:



To resume script execution, click the Start debugging button on the toolbar or select Debug > Start.

Stepping

With execution paused, you can tell SLED to continue executing the script in certain increments:

- Click the Step Into button on the toolbar or select Debug > Step Into advances script execution by one line each time you select it. SLED follows the script wherever it goes, entering functions when they are called, and returning from them when they end.
- Click the Step Over button on the toolbar or select Debug > Step Over tells SLED to advance script execution by one line each time you select it. Instead of following the script when it calls a function, SLED skips over the function without executing it.
- Click the Step Out button on the toolbar or select Debug > Step Out tells SLED to finish executing the current function, return from it, and pause execution.

Monitoring Variables

SLED provides the ability to monitor runtime values of local, upvalue, and global variables. It also allows you to set watches on variables of particular interest to you.

> Script execution must be paused for SLED to provide runtime information about variables.

Viewing Variables

You can view the values of variables by doing the following:

- To view local variables, select Window > Locals to display the Locals panel.
- To view upvalue variables, select Window > Upvalues to display the Upvalues panel.
- To view global variables, select Window > Globals to display the Globals panel.

Setting Watches on Variables

You can set a watch on variables of particular interest to you. Right-click on the variable name in the Locals, Upvalues or Globals panel and select Add Watch from the pop-up menu displayed, or to go to the variable in the code.

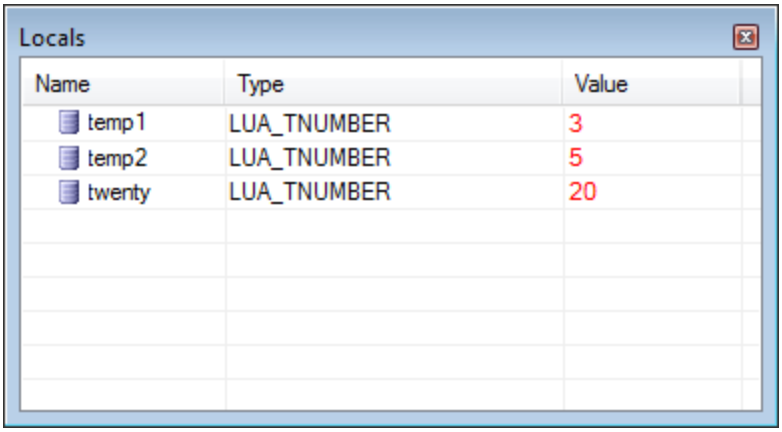
When you set a watch on a variable, it is added to the Watch List panel. The panel does not display the scope for each variable, but you can right-click on the variable to Remove Watch(es). The appropriate scope will appear for the variable you select.

Modifying Variables and Scripts

While script execution is paused, you can change variable values or even the script itself. You can see the effects of your changes immediately upon resuming execution. You do not need to restart the executable.

Modifying Variables

Pause script execution, then modify any value displayed in red in the variable panels, for example:



Double-click on the existing value and enter your change, then press the Enter key. When execution resumes, the script uses the new value.

Modifying Scripts

To modify a script during debugging:

1. Pause script execution.
2. Make the changes required in the script editing window.
3. Resume script execution.
4. When SLED asks if you wish to reload the updated file on the target machine, click Yes. SLED displays a message in the TTY panel that indicates whether or not the target machine successfully reloaded the script.

> ⚠ When you tell SLED to reload the updated file, it automatically saves the changes that you made to the script. Make sure that you have a backup or some other way to revert your script to its previous state.

Debugging using SLED – Example

This example shows how to set up a SLED project for a scene that contains a mini-game, using the following steps:

1. Add your scene to the scene list XML file
2. Add your target to SLED
3. Create a new SLED project
4. View and debug your Lua script file

Prerequisites

To set up SLED using this example, you require the following:

- A Lua mini-game
- A scene that contains the mini-game

1. Add your Scene to the Scene List XML File

To add your scene to the scene list XML file:

1. Open your scene mini-game .lua file and add the following commands:

```
LoadLibrary( "Debug" )
Debug.DebuggerRegister()
Debug.DebuggerSetupProject()
```

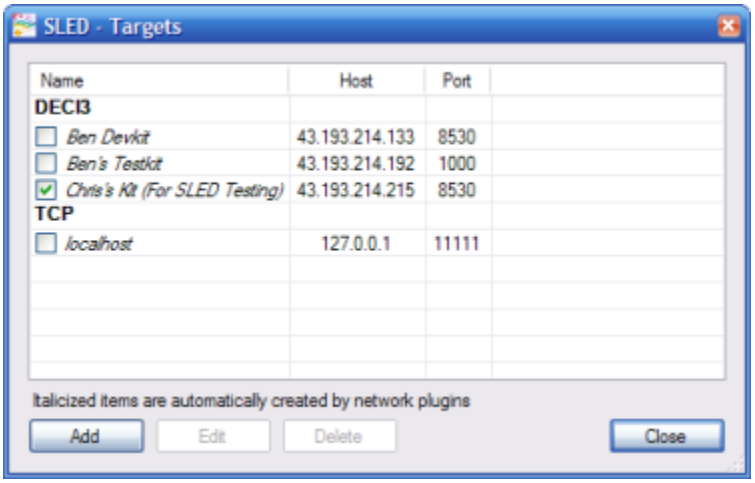
2. In the Object Editor, set the mini-game object's Lua Environment component's debug property to True, then save the file.
3. Launch the scene online so that the scene is added to LocalScenelist.xml, as shown in the following example:

```
<SCENE Name="maya2012env" Type="PublicLobby" SceneID="6713d426-e89a-0834-82f1-d33c85c0f677"
config="environments/maya2012env/maya2012env.scene" />
```

LocalScenelist.xml is located in C:\HomeHDK\build\environments.

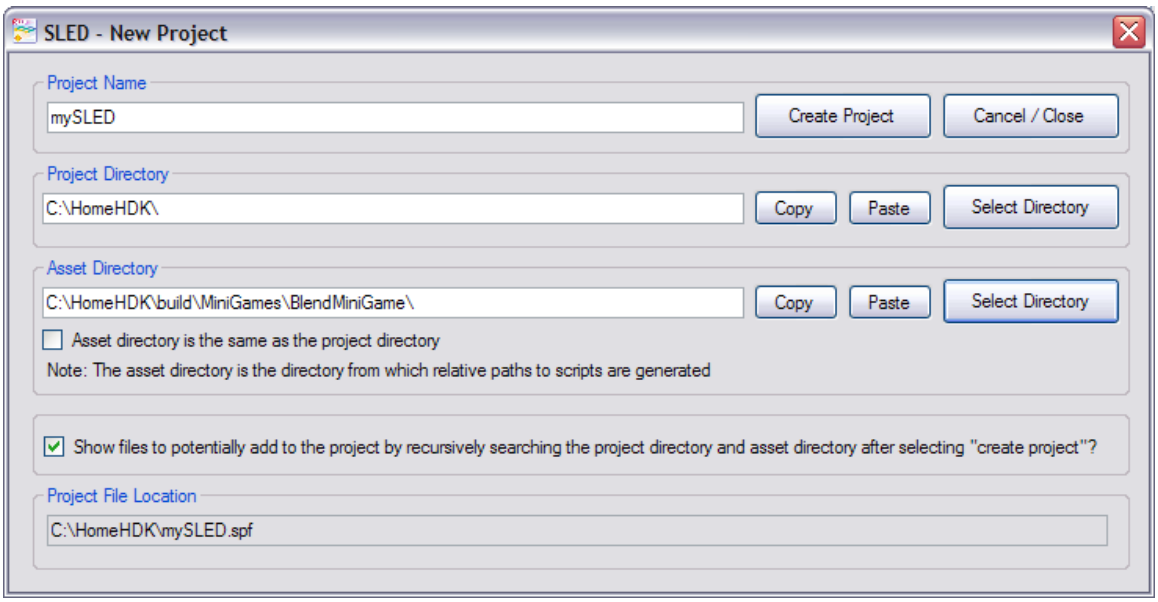
2. Add your Target to SLED

Add your target to SLED, as described in Launching the .self in Debug Mode, remembering to select the target you require in SLED in the Targets dialog:

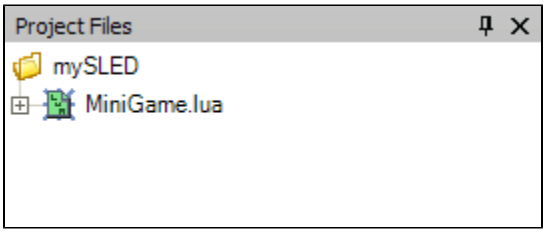


3. Create a New SLED Project

1. Create a new SLED project as described in Setting up a SLED Project, ensuring that the Asset Directory field is set to your mini-game's build folder, as shown in the following example:



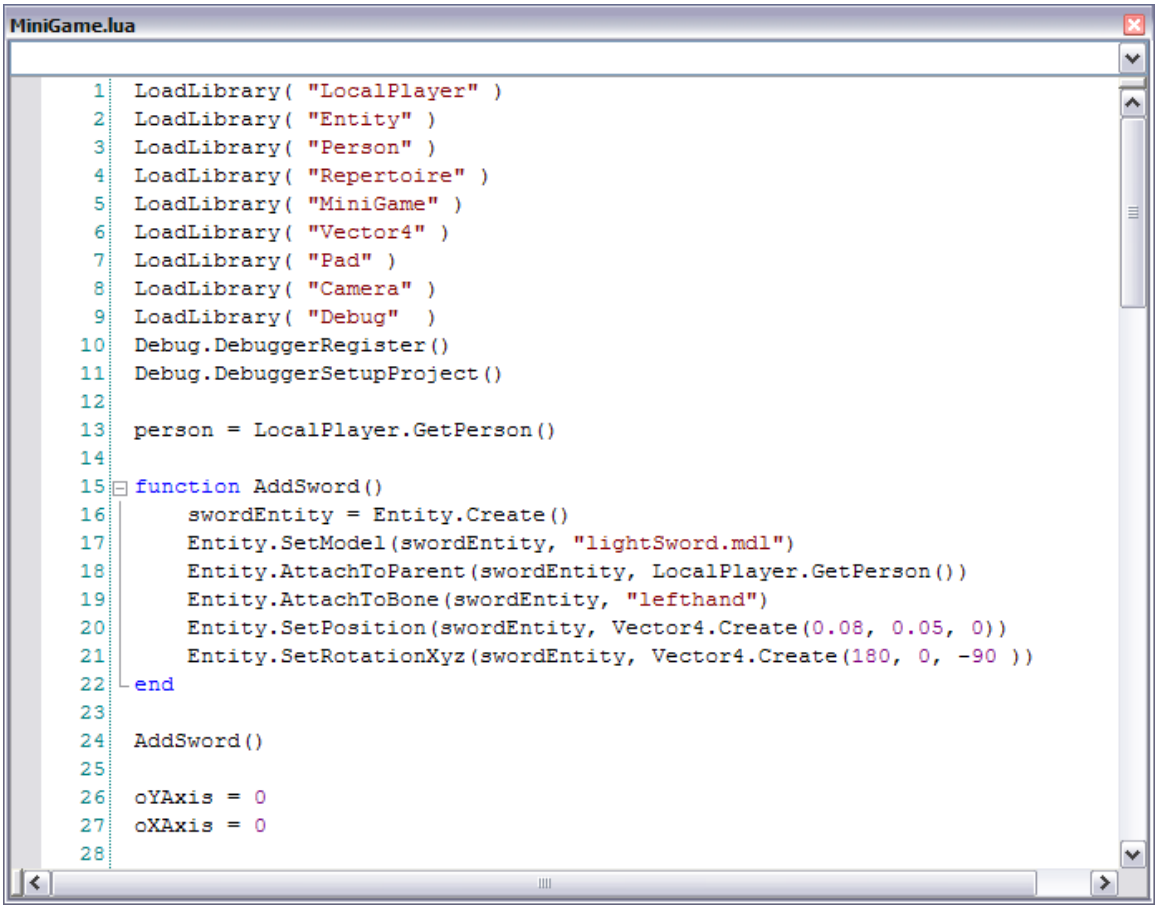
2. Click Create Project. Your MiniGame.lua project file is displayed in the Project Files panel:



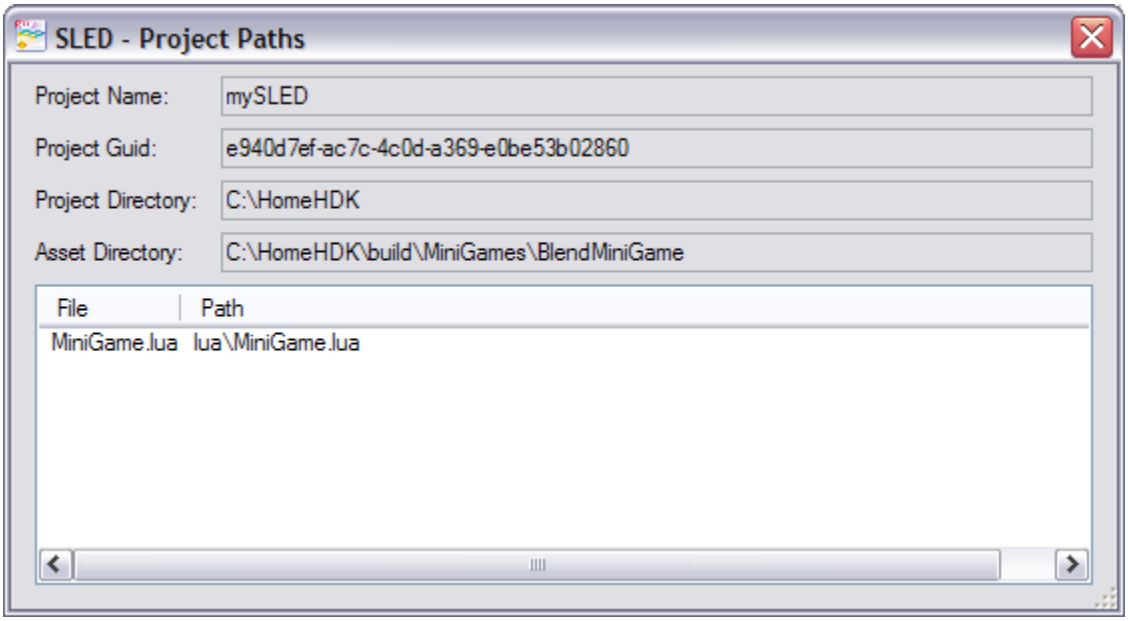
4. View and Debug your Lua Script File

To view and debug your Lua script file:

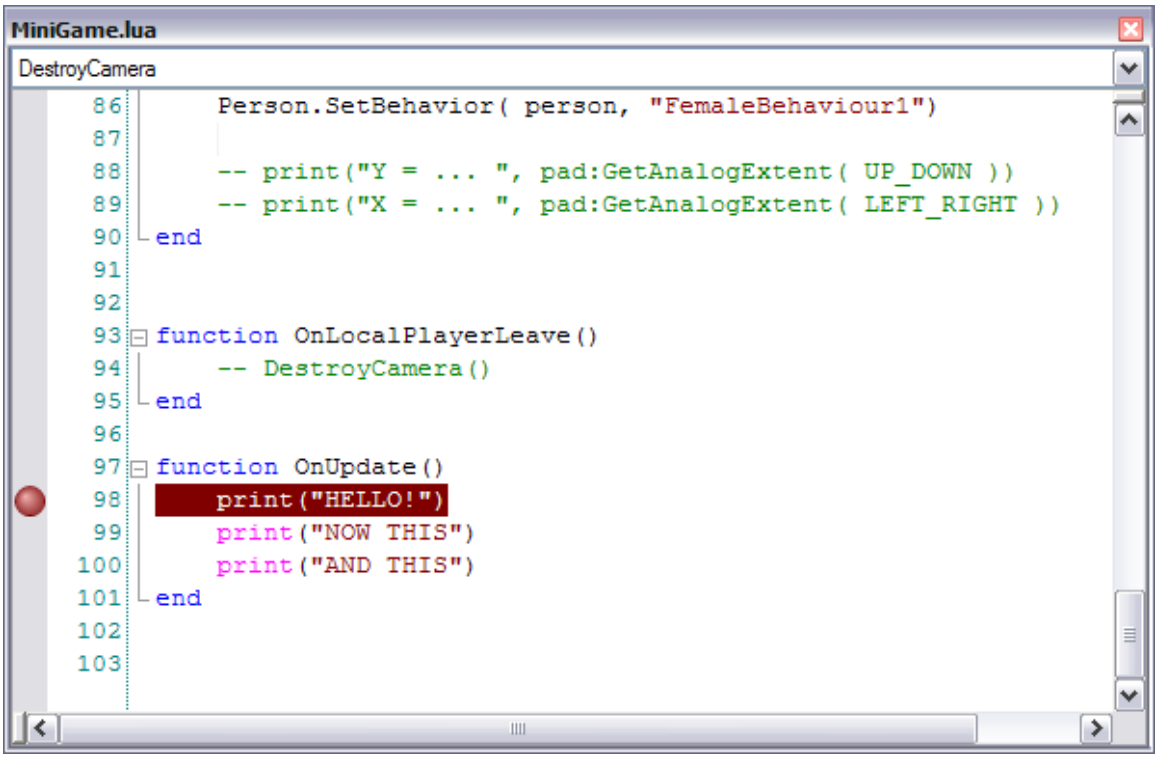
1. In the Project Files panel, double-click the MiniGame.lua file. The Lua script is displayed in a script editing window:



2. Right click on the name of the project in the Project Files panel and select View Project paths. The Project Paths dialog is displayed:



3. Verify that the path and asset directory match the actual location of the Lua script file.
4. Add a breakpoint to the following line, as described in Setting Breakpoints:



5. Select Debug > Connect. You'll see the breakpoint when the script is run.

> The game also stops running.

6. Use debug commands to step over the lines after the breakpoint, as described in Stepping.
7. Disconnect from the target, as described in Connecting and Disconnecting the Target.

Lua Garbage Collection

This section provides information on how Garbage Collection is used in PS Home and in Lua script. This section also includes:

- Tips on getting better memory and CPU performance within PS Home through script and using the Environment Library
- Tips on writing scripts that do not require garbage collection and creating efficient objects and multiple objects in a scene

For more information on garbage collection in Lua, see:

- http://lua-users.org/wiki/GarbageCollectionTutorial
- http://lua-users.org/wiki/OptimisingGarbageCollection
- http://www.lua.org/manual/5.1/manual.html#pdf-collectgarbage

For more information on the Object Profiler, see Profiling Lua Scripts.

What is Garbage?

Garbage consists of Lua allocations that are no longer referenced in the script. Garbage can be generated by redefining variables through shallow copying, in loops, as functions.

The for loop below generates a lot of garbage:	The for loop below generates no garbage:

```
for i = 1, 1000 do

      local vec = i
      print ( vec )

end
```

In the example above, the variable vec is no longer referenced as it goes out of scope in each iteration. When the loop finishes, there will be 999 values that vec cannot access, and one (1000) that vec can.
All of this garbage gets marked for collection, and collected when the garbage collector is called.

```
local vec = 0
for i = 1, 1000 do

      vec = i
      print ( vec )

end
```

In the example above, the variable vec is created before the loop and redefined in each iteration, generating no garbage.



If you do not call garbage collection in a loop, the memory cannot be freed until after the loop has completed and the garbage collector is invoked. Be wary of consuming your memory resources by garbage in loops.

The function below will also generates garbage:

```
function Initialize()

      print( 'Initialized' )

end
Initialize()
```

When the Initialize() function is called, it prints 'Initialized', and then does nothing. Memory was allocated to the function, so while it serves no further purpose after printing, it still takes up that memory until the garbage collector is called.

As with the for loop above, the way to prevent the build-up of garbage (that requires collection) is by destroying it:

```
Initialize = nil
```

Garbage Collection in PS Home

PS Home is designed with garbage collection built-in. By default, the client explicitly calls the garbage collector during:

- Script Initialization: After the Lua script in the Lua environment (e.g. a mini-game) has been loaded and executed for the first time, a full garbage collection is performed. This reduces the maximum memory usage an environment may reserve by inserting a garbage collection between initialization of Lua content and its first update.

  This is only done once for each Lua environment and cannot be controlled by the script.

- Frame Garbage Collection: A garbage collection is performed each frame after the Lua callbacks have run and threads have been updated. This defaults to a full garbage collection to keep the frame rate steady and the memory usage to a minimum. The Environment library can control frame garbage collection.

  Environment.SetFrameGcStep: The frame garbage collection can be changed to a stepped garbage collection using Environment.SetFrameGcStep with a specified step size.

  Environment.EnableFrameGc: Garbage collection can be turned off completely using Environment.EnableFrameGc.

- Callback Garbage Collection: There are two types of Lua callbacks in PS Home, per-frame callbacks and per-call callbacks:

  - Per-frame callbacks are callbacks that occur every frame when active. Examples include those found on the MiniGame component (on_update, on_local_player_update_gameplay, on_can_local_player_join).

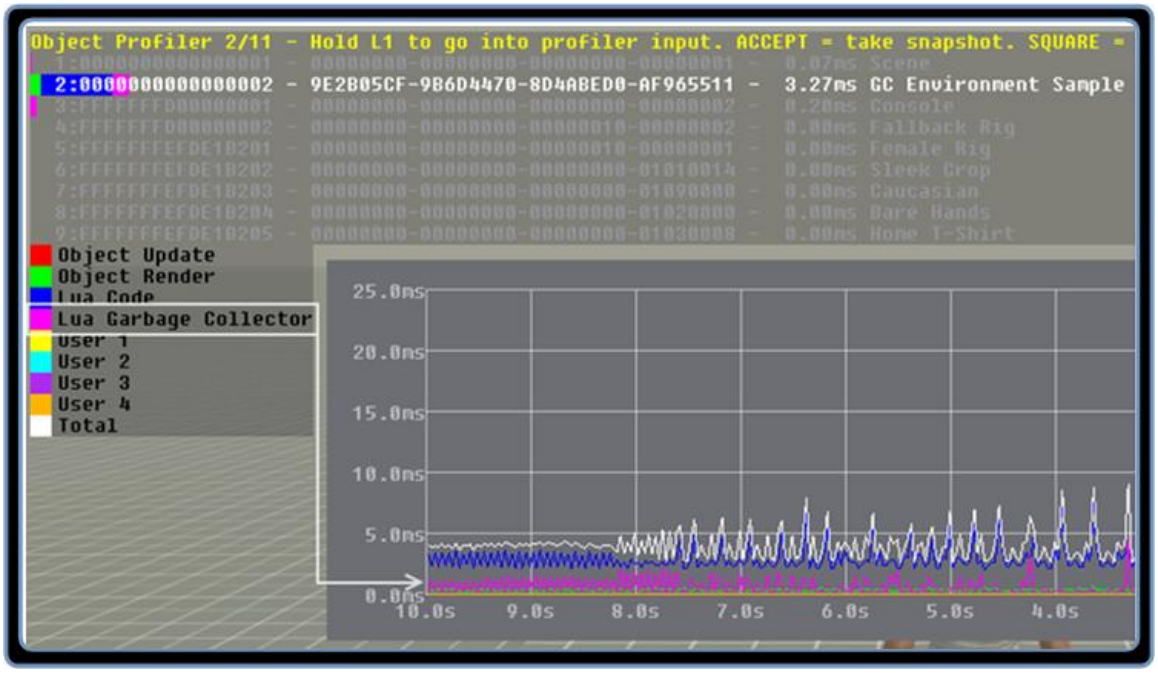
  - Per-call callbacks can be called multiple times per frame. If these callbacks generate garbage, then garbage can pile-up (a large amount of memory may can be allocated), causing out of memory problems before the frame garbage collection is called. For example, on_add_player can generate garbage; since it is called each time a player enters a scene, it can result in a lot of garbage if many people join the scene at once.

    PS Home counts the number of per-call callbacks, and will invoke the callback garbage collector if they exceed a

specific quantity (by default a garbage collection is invoked for every 10 callbacks). This is likely to occur for on_add_player on the MiniGame component, on_remote_player_join on the RealTime Game component, and on_message_received on the Network component.

The counter resets after each frame garbage collection. Only one automatic per-call callback garbage collection can be invoked per frame. The default threshold for automatic garbage collection is 10 per-call callbacks in a frame. This threshold can be controlled using `Environment.SetCallbackGcStep` and `Environment.SetCallbackGcInterval`, or turned off completely.

These explicit garbage collection calls from PS Home appear separately in the Object Profiler as part of the 'Lua Garbage Collector' profile.



Garbage Collection Outside of PS Home

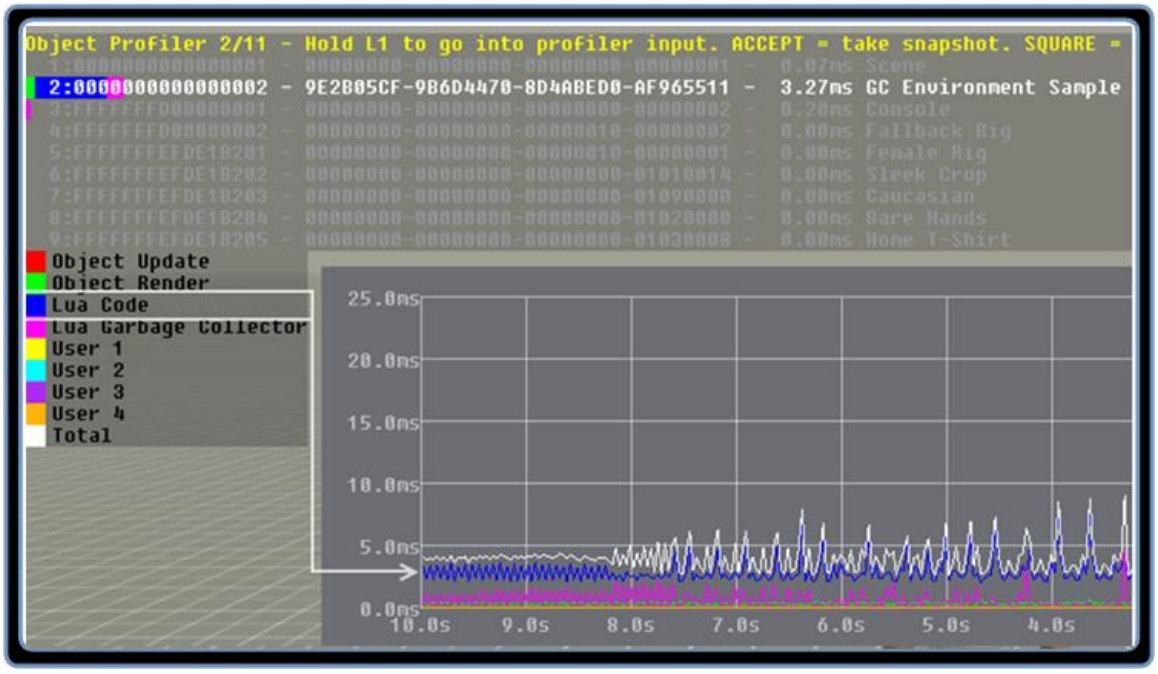
Lua can also call garbage collection outside of PS Home (i.e. within Lua) in the following circumstances:

- In Script: Scripts can invoke a garbage collection when desired by calling the standard Lua function `collectgarbage`. Scripts could, for instance, handle garbage collection completely. You can disable the frame and callback-based garbage collection, and then call `collectgarbage` when appropriate. For more information on using this function, see the Lua 5.1 reference manual.

- Lua Virtual Machine: While the virtual machine runs the script, it checks if the Lua memory usage has passed the Lua environment's garbage collection threshold. When it has, it invokes garbage collection while your script is running. For more information, see luaC_checkGC in the standard Lua documentation.

  You can switch this off by calling `collectgarbage( "stop" )`, which does not actually stop the garbage collection, but sets the threshold to an extremely large number. The CPU time spent performing either of these types of garbage

collection is visible in the Object Profiler as part of the 'Lua Code' profile.



Memory Allocation in PS Home's Lua Virtual Machine

Platforms supporting the Lua language can optionally override the behavior in the standard Lua Virtual Machine (VM). The Lua VM in PS Home takes advantage of this to provide speed increases in allocating small amounts of memory.

- If Lua requires 1-20 bytes, it is allocated from the 20 byte pool.
- If Lua requires 21-32 bytes, it is allocated from the 32 byte pool.
- If Lua requires more than 32 bytes, it performs a standard allocation from the Lua environment's memory pool (e.g. the scene pool).

The 20 byte and 32 byte pools were added to reduce the time taken to allocate memory, as well as reduce the cost of the memory header and alignment for small allocations.

When allocating memory from either pool, the following process occurs:

- When an allocation is requested, the Lua environment checks for free items in a free list.
- If the free list is empty, it allocates a pool for 128 items, adds them to the free list, and then returns one of the items.
- When Lua has finished with the memory, it returns that one item back to the free list.

The benefits to this method are extremely fast memory allocation and a reduction in overall memory usage, but at a cost of returning memory blocks only when script is destroyed (such as changing a scene). Garbage collection only returns the unused memory to each pool's free list, rather than freeing the memory and making it available to the Lua environment memory pool.

Other Home-specific types of objects that use the pool system (namely Vector4, Matrix44, Quaternion) are managed on a per-environment basis.

The Renderer object from `Renderer.Create()` also has pools for specific types of render items: text, rects, and sprites. If the Renderer is garbage collected, it will also return all of the memory used by these pools.

> ⚠ The memory pools belonging to the Lua environment will continue to increase if you create garbage but do not run garbage collection.

> ℹ When you run garbage collection you do not return to your original memory footprint, but instead return memory blocks to free lists. The memory footprint is only recreated when the script is destroyed.

Managing Garbage Collection

Disabling Garbage Collection

You can disable garbage collection dynamically or manually using the standard Lua command `collectgarbage`. You may want to disable garbage collection for the following reasons:

- Collecting garbage after initialization, then switching off garbage collection.
- Only collecting garbage if the user is playing a game.
- Interleaving garbage collection between Lua environments to reduce overall CPU time in the scene.
- Only collecting garbage on non-update callbacks, or at particular events.

To disable all garbage collection, call the following commands within your Lua content:

```
Environment.EnableCallbackGc( false )
Environment.EnableFrameGc( false )
collectgarbage( "stop" )
```

Setting Up Callbacks

By default, callbacks in PS Home are specified as a string. The string is sent to the interpreter (e.g. `OnUpdate()`) and executed. This process requires compilation and uses memory that requires garbage collection. A new format has been introduced where the callback must be a function and prefixed with an asterisk (*) .

`OnUpdate()`, for example, will now be `*OnUpdate()`, avoiding any memory allocation when called.

PS Home content still runs normally without the asterisk. We recommend using it to optimize performance.

Writing Code That Doesn't Allocate Memory

Pass Arguments or Avoid Functions that Allocate Memory

Lua tables, functions, and userdata allocate memory. PS Home's Lua API uses Lua userdata to create their instances. For example:

```
Vector4.Create()

Time.Create()

Matrix44.Create()

Person.FindInInstance( strnig personId )
```

If you want to write code that does not allocate any memory, you cannot use any API function that creates userdata (e.g. Vector4), in your update loop. If you do use a function that creates userdata, you will require garbage collection at some point. You can, however, create your userdata beforehand, and pass that information as arguments into the functions. Many of the Home Lua APIs have changed to allow userdata to be passed into them, such as the two syntaxes of `Entity.GetPosition()`:

```
--create the Vector4 (i.e. userdata)

Vector4 EntityGetPostition( Entity entity )

--Entity.GetPosition takes and stores the results in the Vector4, instead of creating a new Vector4.

Entity.GetPostition( Entity entity, Vector4 outVec)
```

The new syntax, allowing arguments to be passed into functions, is slowly being implemented across the Home Lua API. You may find that some functions currently do not support this syntax, which makes writing code that does not generate garbage impossible.

Functions that return a string, number, or nil do generate garbage or require garbage collection. For example:

```
for i=1, 100 do


        local t = "fred"

        t= "july"


    end
```

Create a Fixed Size Pool

If you require dynamic creation and destruction of userdata created through the Home Lua API, you can avoid dynamically allocating memory by using a fixed size pool. The example below show a very simple code that creates a pool for the function `CreateEntity()`, creates and entity, and then destroys it. The code is very simple, missing elements of a robust codebase, like debug code.

```
function InitEntityPool()

    g_entityPool = { }
    for i = 1, 100 do

        table.insert( g_entityPool, Entity.Create() )

    end

end

function CreateEntity()

    local size = #g_entityPool
    if size == 0 then

        return nil

    end

    local e = g_entityPool( #g_entityPool )
    table.remove( g_entityPool)
    return e

end

function DestroyEntity( e )

    e:SetVisible( false )
    table.insert( g_entityPool, e )

end
```

Reducing Globals

If you have garbage collection enabled, you can reduce the mark and sweep time of the garbage collection by taking the following steps:

- Divide Global Data Into Lua Resources: One way to reduce global data is to remove all data that is not constantly needed, and place it in smaller resource files for use when appropriate.

  For example, you could keep a game level's initialization data, level layout, power-ups, and enemy location in one table, but the garbage collection on this constant data would take longer (possibly significantly longer depending on the table size) for the garbage collector to mark and sweep through the table. Instead, split each level, enemy information, etc into separate Lua resources, run the resources when needed, and set them to `nil` when you have parsed the data.

- Store Data in a Memory Container: If you require access to the data in your general update loop, you can store the data as a binary file resource, and access it through a Memory Container. The Memory Container, no matter its contents or size, is seen by the garbage collector as one item. The disadvantage to storing data in a Memory Container, though, is that the functions used to access a Memory Container's data are slower than those that simply access a Lua table. This can lead to a situation where the time you saved running the garbage collector by placing data in Memory Containers is less than the

code time for it. If you find using Memory Containers to be a drain on time, you can try caching the API function calls, and using the cached versions instead. For example:

```
LoadLibrary( "MemoryContainer" )

function update()

    local total = 0

    for i =1, 1000 do

        total = total + MemoryContainer.GetInt8( mc, i )

    end

end
```

can be optimized to:

```
LoadLibary( "MemoryContainer" )

MemoryContainer _GetInt8 = MemoryContainer.GetInt8

function update()

    local total = 0

    for i = 1, 1000 do

        total = total + MemoryContainer_GetInt8( mc, i )

    end

end
```

Checking Memory Leaks

Use the Debug Console profiling options to check for memory leaks. See Profiling in the Client.

The easiest way to check for memory leaks is to isolate the content in a scene, and enable the Object Profiler. For example, to test a mini-game, create a scene with just the mini-game in it.

With the Object Profiler open, select the object, and watch the memory stats. Ideally, the memory stats should not change at all:



You can also check the garbage collection on the object in the CPU Profiler panel:



If you have a memory leak, isolate the location of the leak by commenting out all of the code and adding a bit at a time until you find the leak.

Another way to discover memory leaks is to manage memory allocations using Lua's debug library. You can place the following code inside a Lua script to check for any increase in memory usage after each line of code within a Lua environment:

```
function PrintMemoryAllocation(event, lineNumber)
        local usedMemory = collectgarbage("count")

        if (usedMemory > previousUsedMemory) then

            if (debug.getinfo(3, "u")) then

                local functionInfo = debug.getinfo(2, "nS")
                local functionName = functionInfo.name
                local functionFile = functionInfo.source

                allocatedBytes = (usedMemory - previousUsedMemory) * 1024

                if functionName ~= nil then

                    print(string.format("[Debug] (%s: %d) %d bytes allocated in %s()", functionFile,
lineNumber, allocatedBytes, functionName))

                else

                    print(string.format("[Debug] (%s: %d) %d bytes allocated in callback",
functionFile, lineNumber, allocatedBytes))

                end

            end

        end

        previousUsedMemory = usedMemory;

        collectgarbage("collect")

    end
    debug.sethook(PrintMemoryAllocation, "l")
```



Use this code for debugging errors in Lua content only; do not use it when the script is submitted for use in the live environment.



To use Debug Lua functions, you must set the Lua Environment component's debug property to True in the Object Editor.

Lua Features

PS Home's Lua features are too numerous to list, so feel free to browse through features described in this section.

See also:

- Arcade Games
- Mini-games
- Active Items
- Camera Obstruction on Seated Avatars
- Chat System in PS Home
- Collision Layers and Filters
- Collision UserID
- Handling User Input
- Maximum Number of Players in Mini-Games
- Tips for Scripting Collision



Object Information

Overview

All games, items and other content in PS Home, including minigames, furniture items, clothing, inventory items and companions, are generically called objects. Objects have a unique ID called an object ID which is used to identify that object in the object catalog, and an instance ID which is unique per object instance, so multiple instances of the same object can be differentiated from each other. Objects are created using the Object Editor, which is part of the HDK.

Because objects represent so many different types of individual items, PS Home uses a component system, which allows objects to only use the functionality they require.

For example
Furniture items have a furniture component, which gives them the functionality to be placed in a scene using the furniture system. Objects can have as many or few components as they like depending on what sort of functionality they wish to make use of. Some components require other components (or preclude the use of other components) to operate, and this is enforced in the Object Editor, which is used to create objects and add components.

Object Library

You can query information about an object in Lua using the Object library, which is available to all objects. Generally you will be most interested in the current object, i.e. the one running the script. You can get a handle to this object by calling `Object.GetMe()`. You have limited access to objects which are not the current object in certain situations, the most common of

which is probably when using Listener data (see below). The general categories of Object functions are listed below.

Object Function Categories

Querying object details

You can query object details such as the object ID, instance ID, object name, the object owner if there is one, initial position/rotation and so on.

Listener data

Using the Scene library (see Scene Information, and the Scene library in the Lua API Reference documentation), you can send data directly to other objects in the scene, who are known as listeners. Any type of Lua data can be sent to these objects, which will receive a callback with the data when it is sent.

> Note that sending listener data is synchronous, so it will arrive immediately within the same frame. Generally you will obtain a list of listener objects from the Scene library, and then use the Object library to send listener data to the appropriate functions. See `Object.SendListenerData` for more details.

Instance parameters

Objects can have instance parameters assigned to them in the Scene Editor, which allows instances of the same object to customize themselves based on these parameters.
For example, if you have a pool table minigame object and you wish to have 6 tables in your scene, 3 of them blue and 3 of them red, you can create one pool table object and have it read the instance parameters assigned to it to determine which color it should be.

Text localization

PS Home has a built-in text localization system, which allows you to retrieve text stored in the object localized to the correct locale for the current user. The functions `Object.GetLocalizedText` and `Object.HasLocalizedText` are used for this purpose.

Targeting

For objects with a targeting component, a number of functions are available to modify the targeting behavior of PS Home. You can modify the position and text of the targeting label and enable and disable targeting if necessary.

> This is not an exhaustive list, there are several other functions also available in the object library. For more detailed information on these or the above functions, consult the Lua API Reference documentation.

Scene Information

Overview

All spaces in PS Home are known as scenes. Scenes are 3D environments in which the user's avatar is able to walk around and interact with objects and other avatars. Scenes are classified into different types, which gives them differing levels of functionality. It is important to classify your scene as the right type - based on the kind of scene you wish to create (Personal Space, Public Space, etc. - see PS Home Spaces for a full list of scene types). Scenes are created in the Scene Editor, which is part of the HDK.

Scene Library

Objects can query information about the current scene by using the Scene library. The scene itself is represented by a "virtual object" which owns the scene Lua script if there is one, and can be retrieved using the function `Scene.GetVirtualObject`.

> Only objects permanently located within a scene are able to use the Scene library. This includes minigames and real-time games, embedded objects, and the virtual scene object itself. All qualifying objects in the scene may use the Scene library, and it is up to the developer to ensure that objects do not try to apply conflicting settings to the scene.

There are a number of functions in the Scene library, which can be broadly grouped into the categories described below.

Scene Function Categories

Finding scene items

It is possible to find items in the scene by name and return a handle to them, so that scripts can update or otherwise manipulate or query these items. The items which can be located currently are entities and screens.

Scene parameters

Scripts and objects can request that parameters be passed to a scene during the relocation process, and these parameters can be obtained from the Scene library. These parameters may be used to identify the origin of a user, or to customize a scene depending on how the user was relocated to that scene.

Listener objects

Objects in a scene can register themselves as listener objects. The scene will maintain a list of these objects, and allow scripts to query the list and send data to these objects at any time using the Object library. For more information, see Object Information.

Visual customization

The Scene library allows a script to customize various aspects of how the screen is lit and rendered, including light probes, sunlight direction and colour, shadows, colour balance, saturation, contrast, glow/bloom and environment maps.

Save data

Certain scene types (personal spaces/apartments and clubhouses) allow certain users to save data with the scene, which is available for all users in the scene to load. This can be used to allow the apartment owner or club owner to customize aspects of the scene and have these changes be loaded for all other users in the scene. The format or content of this save data is up to the scene, no restriction other than the maximum size of the data is made by the client.

> This is not an exhaustive list, there are several other functions also available in the scene library. For more detailed information on these or the above functions, consult the Lua API Reference documentation.

SceneInfo Library and Async Command

The SceneInfo library is a more limited library accessible to all objects. It allows a script to identify the current scene and retrieve its scene ID and name, and also the unique instance ID of the current scene. This may be useful if tracking the progress of a user through a series of scenes or identifying where to send a user to next, amongst other things.

Additionally, there is an async command available to query information for a specific scene, which need not be the current scene. PS Home maintains a database of all scenes, and can query this database to provide details about a specific scene given the scene ID. This is also available to any object type. The details available include the scene description, thumbnail URL, and author information.

For more information on both of these see the Lua API Reference.

System Information

Overview

The PS Home client has a number of system settings and the System library provides access to this functionality (functionality which is not executable by custom content created by developers).

Most core functionality is included in this group, such as the web browser, game launching, the chat log and chat system, commerce points, and so on. The System library allows scripts to have a certain degree of control of this functionality where it is beneficial, and also to query the state of certain system features to allow objects to determine if their functionality may be affected and adjust their behavior if necessary.

System Library

The System library is accessible to all objects. It provides a number of functions relating to built-in PS Home functionality and features which can be used by objects if necessary, and which can be grouped into the following categories.

System Function Categories

Game launching

Game launching allows the PS Home client to exit and execute another game title, and then return when that title is exited. The System library allows a script to trigger this process automatically on behalf of a user, and to detect if the user has initiated their own game launching session.

Web browser

The PlayStation® 3 contains a built-in web browser, which can be requested by a script and directed to a specified URL. Note that the web browser requires additional memory and may not be available in some scene types.
Also, some web browser functionality may not be available in some scene types - e.g. Adobe® Flash® Player is not available in the web browser in Personal Spaces or Clubhouses, see the technote https://home.scedev.net/technotes/view/46/1.

NP messages

The library allows a script to send a PSN message directly to one or more users' inbox. This can be read by the users outside of PS Home.

System settings

Various system settings that may be of use to a script are available, including whether the look Y axis is inverted, the screen dimensions, the SCE territory that a particular region belongs to, the chat log state (minimized, maximized or closed), the current user's router NAT type and so on.

Commerce points

The System library is used to open a commerce point, and can optionally direct a user to a specified node in the commerce point.

Video upload

PS Home allows a script to upload a video file recorded by a user in PS Home to YouTube® for public viewing. The upload can be initiated and the progress queried using functionality provided by the Scene library. The resulting URL of the video is provided if the upload succeeds.

There are a number of other miscellaneous functions also located in the System library, please refer to the Lua API Reference for more details.

Person Information

Person Information - Accessors that Get and Set the Player State

The Person library in the Lua API gives you access to the state of the player.

> For example:
>
> - What they are doing
> - What they look like
> - Where they are focusing
> - How they appear to others
> - Their collision or velocity
> - Where their friends are in the instance
> - Whether they are locked in place
> - What their label says

You can get information about a player state. You can also set the player state.

The individual Lua functions in the Person library can be separated into roughly three groups: General, Status and Appearance.

General

Some functions in the general group:

Person.IsCollision	Tells you if the persons collision is enabled
Person.IsVisible	Tells you if the person is visible
Person.IsValid	Tells you if the handle to the person is valid
Person.IsLoaded	Tells you if the person's avatar has loaded and is ready

These functions are commonly used in Mini-games to ensure the player's state is appropriate for interactions with other users.

Status

Players can set their status. Statuses can influence facial animations (moods) and the Player Label status text.
These functions allow developers to get the status the users have set, or to set them.

`Person.GetAvatarStatusType` - This returns the actual status type the user has set.

`Person.GetMood` - Returns the mood (the facial animation).

Status, Moods & Animations

AvatarStatusType	MoodType	Icon
Neutral	Neutral	none
WantToChat	Neutral	none
WantToPlayGame	Neutral	none
Busy	Neutral	Inactive
Happy	Happy	Happy
Sad	Sad	Sad
Angry	Angry	Angry
Confused	Confused	Confused
IsLookingForHelp	Confused	Confused

 For related information, see Repertoire Condition Rules

Appearance

Players can customize their avatar by changing elements such as gender, body size, facial structure, eye color, hair style, makeup, etc, along with what clothing they wear.
Some of these customizations can be queried by the Person library functions.

`Person.GetRigComponentTypes`	Find out what clothing types are available for the person.
`Person.GetRigComponentObjectId`	Find out which clothing items the person is using for a given clothing type - e.g. what hat the person is wearing.
`Person.GetRigObjectId`	Find out which rig the person is using.
`Person.GetRigField`	Query the rig for information. Currently the only field that is exposed is "sex". `GetRigObjectId` could also be used to determine the sex.
`Person.GetHeight`	Query the avatar body height (in meters).
`Person.GeWidth`	Query the avatar body width - returns a factor of the avatar width between -1.0 (thin), 0 (default), and 1.0 (large).

 See the `Person` library in the Lua API for more details.

Broadcasting Person Functions

Changes in avatar gender, clothing, animation (e.g. dancing or walking), movement, and orientation, are automatically broadcast to all other users in the scene.

To broadcast any other avatar settings, use the Lua API to send the information manually.

Debug Person Information

The following are console commands for debugging using the Person library, which can be useful for testing a variety of character-related content such as repertoires:

`Person.Dbg <person name>`	Prints the current debug person if no parameter is specified. If a person is specified, sets that person as the current person (if that person exists).
`Person.Dbg.ShowStats`	Lists clothing details, current animation network (and state), and animation register values.

Person.Dbg.ShowAnimRegisters	Lists the animation register values.
Person.Dbg.ListRepertoires	Lists the repertoires currently added to the person.
Person.Dbg.SetDebugAnimNetwork	Adds the specified animation network name (action or behaviour) to the list of animation networks that will output debug info. Every time a debug network changes state (or starts) then all the results of the conditions for the states are output, listing the current possible states, and the selected state.
Person.Dbg.ListDebugAnimNetworks	Lists the current animation networks that are marked for debug output.
Person.Dbg.ClearDebugAnimNetwork	Removes the specified network (action or behaviour) from the list of debug networks.
Person.Dbg.ClearAllDebugAnimNetworks	Removes all debug networks.

Retrieving Object Metadata

You can retrieve an object's name and description in your Lua script using the AsyncCommands library. The ObjectRetrieveMetadata functions will return the localized name and description for the region that you (or the user) are in.

To retrieve an object's metadata:

1. Create the AsyncContext and create the AsyncCommand using the AsyncContext.

```
context = AsyncContext.Create( AsyncContexts.ObjectCatalog )
if ( context == nil ) then

    print( "Unable to create object catalog context!" )
    return
    end

metadataCmd = AsyncCommand.Create( AsyncCommands.ObjectRetrieveMetaData )
```

2. Execute the AsyncCommand.

```
if ( metadataCmd ~= nil ) then

    if ( not metadataCmd:Execute( context, "ABCDEF01-ABCDEF01-ABCDEF01-ABCDEF01" ) ) then

        print( "Metadata query execute failed" ) -- script can retry here if necessary

    end

end
```

3. Retrieve the results of the AsyncCommand.

```
if (metadataCommand:IsFinished() and metadataCommand:Succeeded()) then

    local r = metadataCommand:GetResults()

    print("Object name:", r.name)

    print("Object description:", r.description)

end
```

Lobby Instances

What is an Instance?

An instance is a kind of copy.

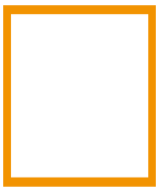
For example

Picture a space - let's say a Circus Tent space. The maximum number of people in this space is 8.

Once 8 people have entered that space - what happens now? If there was no instancing, that would be it - nobody else but these 8 people could access the Circus Tent.

With instancing, each time a space reaches capacity in terms of users, another copy of the space is spawned and that copy of the space starts filling up with the new group of users (until capacity is reached again, and then the process repeats).

So we start with one instance of the Circus Tent space



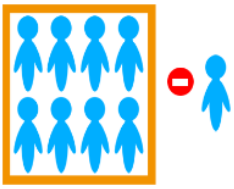
Then people start entering the space



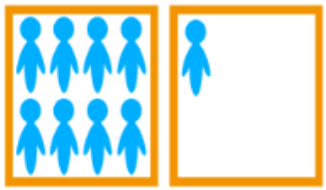
Eventually the Circus Tent Space capacity of 8 is reached



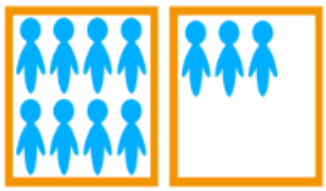
Now no further people can enter this space



Until a new instance (a copy of the Circus Tent space) has been created



for the next group to populate / and everybody trying to enter the Circus Tent space will be funneled to the instance of that space that isn't full to capacity



And each time one instance of the Circus Tent space gets full, another instance spawns and starts being populated



PS Home instances scenes automatically. If you wish to prevent automatic instancing, use the `LocalPlayer.RelocateToUniqueInstance` function (you can read about it in Relocating to Instances).

In the example, each instance of the Circus Tent space is identical, except for the population. Each instance has a collection of people who can see each other and see the same things happening in the space. Each instance is often referred to as a "lobby".

But instances can also vary, as explained in the next section.

Variations in Instances

Instances are a kind of copy - and in some cases each instance is exactly identical.

But instances are not just about making sure no spaces ever get "full" by automatically creating new ones to populate. Instances can also have variations.

The instances have parameters that can be set differently, making each instance have slightly different values and behavior. This is determined in the destination scene's script and takes effect when players relocate to the destination scene.

Instance Parameters

The `InstanceParam` function allows you to set instance parameters for a scene. Instance Parameters are used to create variations by changing the behavior in the instances of a scene.

When a player is relocated to a new scene (either by using a relocate region placed in another scene, or by the `LocalPlayer.Relocate` function), an optional 'instance' parameter can be passed.

The Instance Parameter can be queried by the destination scene's scene script to change the behavior in this new instance of the scene.

For example

You might have a Leisure Lounge scene with a Game Room scene connected to it.

When players choose to move from the Leisure Lounge to the Game Room, the Leisure Lounge scene's script could direct players based on their regions.

The script could send all European users to Game Room 1 instances and all US users to Game Room 2 instances. European users would all appear in one set of instances of the Game Room, US users would all appear in another set of instances of the Game Room.

The Game Room script could then use the Instance Parameter function to determine if the current instance is a European or US one and change behavior appropriately.

For more information and how to set instance parameters, see Dynamic Spawning > Instance Parameters

Related pages:

Relocating to Instances
Managing Groups
Regional and Global Instances

Relocating to Instances

For Japanese Live Page
最新の日本語版ページはこちら 「インスタンスへの移動」

Relocating uses the LocalPlayer Library:

Function name	What the function does
LocalPlayer.Relocate	Use **LocalPlayer.Relocate** to relocate users to a specific scene, locating and creating new instances as required.
• InstanceParam	With LocalPlayer.Relocate you can use the Instance Parameter (**InstanceParam**) argument to manage space instances and direct users to particular space instances based on their gameplay needs.
LocalPlayer.RelocateToUniqueInstance	Use the **LocalPlayer.RelocateToUniqueInstance** to manage unique instances of spaces. When using this function, the destination instance is determined by the instance ID you provide. Only 1 instance with the specified scene name and ID will exist. It can be used in conjuction with LocalPlayer.GetHomeAccountID to send the user to 'their own personal instance'.
LocalPlayer.RelocateToGroupInstance	Use the **LocalPlayer.RelocateToGroupInstance** to relocate group members into the same instance. This function keeps players in a generic holding medius lobby and creates a unique DME instance based on the uniqueID they provide.

You may use Relocate to relocate to any scene of any player size. RelocateToUniqueInstance or RelocateToGroupInstance should be used when you want direct control over how scene instances are managed together with your own server-side code.

The relocation process is asynchronous, the user will continue to be considered present in the scene (and joined to a minigame, if applicable) until the scene transition occurs.
Therefore, expect that your script will continue to execute for some time after this call until the relocation has occurred.

Lobbies created using the standard Relocate method and InstanceParam are not unique. If a lobby becomes full, new instances are created automatically by the system. Use the LocalPlayer.RelocateToUniqueInstance() function if you want to ensure that only one instance of a particular scene/UID exists.

Relocate

The Relocate function relocates the local player to the specified scene.

Setting **InstanceParam**

You can set instance parameters with the `InstanceParam` argument.

Setting `InstanceParam` on `Relocate` does not create a single unique instance. New instances of the space are created if:

- a space created using the standard `Relocate` method and `InstanceParam` is full, or
- multiple users attempt to join a new `InstanceParam` space simultaneously

Use

Use `InstanceParam` to create variations by changing the behavior in the instances of a scene. See Lobby Instances > Variations in Instances for more information.

For example - in your source script, you may check for the player's language setting, then send players with language 1 to scenes with instance param 1, and players with language 2 to the same scene but with instance param 2. In the destination scene, the script would check the passed-in instance param and use the appropriate localisation set.

For more information and how to set instance parameters, see Dynamic Spawning > Instance Parameters

Relocate to Unique Instance

`RelocateToUniqueInstance` is a way to turn off PS Home's automatic instancing for a particular space and allows you to create a single unique instance of a space.

For example

Individual instances for each player's area in a persistent game.

Perhaps your game allows each player in Home to build their own castle (with the state of the castle stored using the savedata API). Players would congregate in your public lobby area, before jumping a unique instance of their own castle by calling the `RelocateToUniqueInstance` function with the player's HomeAccountId as the instance parameter.

If the unique instance space is full, the user is returned to their previous location. However, the previous location may be a different instance from the one they left when trying to access the unique instance space. If they were participating in a mini-game when they were relocated to the unique instance space and it is full or access is denied, they are no longer participating in the mini-game that performed the relocation.

See Dynamic Spawning

Relocate to Group Instance

Relocates the local player to a group instance of the specified scene.

`RelocateToGroupInstance` works like `RelocateToUniqueInstance`, except that the 'uniqueID' for the destination instance is derived from the unique group ID that the player is joined to, rather than defined by the script.

RelocateToGroupInstance works with any group – regardless of how it was formed
RelocateToGroupInstance doesn't care how the group is created, whether by the API or by users in the menu screen.

`RelocateToGroupInstance`, combined with the `Group` Library, allows you to control instances in an optimal way without having to implement your own back-end to track IDs.

For example

- User joins a minigame in a space.
- The minigame handles the 'matchmaking', putting the user into a group with up to 7 other players using the `Group` Library.
- When everyone is ready, the minigame calls `RelocateToGroupInstance` for each of the 8 local players.
- The players are relocated to the instance of the destination space along with other members of that group, where they'd participate in the main game.

There can be multiple group instances, (e.g. 4 members of a group could be redirected to a unique group instance of SceneA, while the other 4 may be relocated to a unique group instance of SceneB), but there can only ever be 1 group instance for a particular scene.

Calling the `RelocateToGroupInstance` will only relocate the local player. Each group member in turn would be expected to call that at the appropriate time.

The group instance will be unique for the destination space.

> **For example**
>
> Say we have group 101, and one member relocates to a groupInstance for sceneID 9999. We basically create a network lobby for 101-9999. If, at some point later, another member of the same group is relocated to a group instance for sceneID 9999, they go to lobby 101-9999. (Extending that, its therefore perfectly acceptable to have group instances coexisting for different scenes - for instance 101-7654 for sceneID 7654).

It is worth thinking of this in the context of the way a lot of PS Home games work - they use the space and a 60 player minigame as the lobby area, and join the users into smaller groups for the gameplay. With the extended Group Library, you can use the same type of lobby area to form a group of X players in the game session. Once everyone is ready, they're relocated to the new space for the main game bit. (The new space could be a game16 scenetype to get extra memory for gameplay and you could use a scene-transition object to make the whole transition appear seamless).

Related pages:
Lobby Instances
Dynamic Spawning
Managing Groups
Regional and Global Instances

Managing Groups

Group Library

'Groups' in Home are temporary collections of up to 8 players. Players in a group have a custom chat channel (text and voice) that remains available even if the players are in different scenes within Home. Using Lua functions, instances of scenes accessible only to the group can be created. Players can create groups themselves using menu screen options or - if the player has not created a group themselves - they can be created through Lua.

Group Library functions allow you to query the status and details of the player's group if they are in one, and form or join a group if they are not.

Function name	
`Group.Create`	Creates and autojoins the local user to the group, if not already in a group.
`Group.GetId`	Returns the users current group ID into the supplied BigInt, or returns false if not in a group.
`Group.GetMemberCount`	Gets the number of members of the current user's group. If the current user does not have a group, this function will raise an error.
`Group.GetMemberList`	Get the list of members of the current user's group. If the current user does not have a group, this function will raise an error.
`Group.IsInGroup`	Returns true if the current member is in a group.
`Group.Join`	Joins the local user to the specified group, if not already in a group.
`Group.Leave`	Leaves the current group, but only if the user was joined to this group by the current script. That is, `Leave` cannot be used to remove the user from a group they have entered themselves or been entered into by another script, in which case it will return false.

> The create, join and leave operations take a certain amount of time to complete. Notification of the result will take place via a callback function.

To use the `Group` library you need to add the group component. This component will specify one callback function "event_callback", that will be of the form:

```
function OnGroupEvent(GroupEvent type, number resultCode)
  end
```

Enum values:

```
GroupEvent.CreateSuccess
GroupEvent.CreateFail
GroupEvent.JoinSuccess
GroupEvent.JoinFailGroupEvent.Leave
```

An Example of Using the **Group** Library

A typical use of the Group library is to use it to form a group that can be relocated together with the LocalPlayer.RelocateToGroupInstance function - as described in Relocating to Instances. RelocateToGroupInstance allows developers to place players in unique instances without a scalability overhead on PS Home servers.

The general procedure would be :

- Players enter public lobby area.
- Your lobby script/minigame allows the user to form or join a group with other players (if they are not already in one).
- Once ready, all players in the group are relocated to a group instance to play a game.
- When complete they are relocated back to the main lobby to start another game.

To create a 'seamless' user experience, you might also consider using a SceneTransition object to hide the relocation process. (Putting up a 'Game Loading' screen or other info while the relocate is in progress).



Note:

The recommended method of relocating groups is to use the Group library and the LocalPlayer.RelocateToGroupInstance function (described in Relocating to Instances).

The GroupDoor library method is still supported, but not encouraged.

Related pages:
Lobby Instances
Dynamic Spawning
Relocating to Instances

Using the GroupDoor Library



Note: This is not the recommended method for relocating groups
The recommended method of relocating groups is to use the Group library (see Managing Groups) and the LocalPlayer.RelocateToGroupInstance function (described in ?Relocating to Instances).

The GroupDoor library method is still supported, but not encouraged.

The GroupDoor library allows instancing in PS Home. The feature creates an instance of a scene exclusively for group members, where they are guaranteed to play a Mini-game or Realtime game together.



In short: The GroupDoor sends group members to an instance of a Group Scene



.

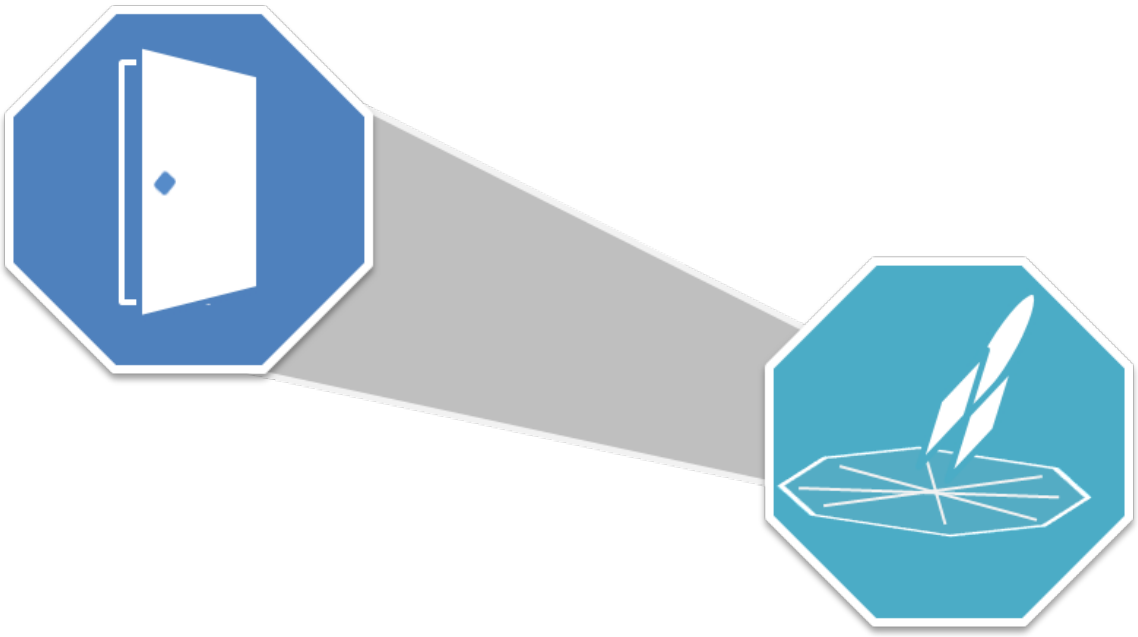
How It Works

A member of the group enters the GroupDoor, creating a group instance of the linked scene (henceforth referred to as the Group Scene). The other members of the group join the scene by either:

- Entering the GroupDoor
- An invitation from a group member to join that Group Scene
- Choosing from their Menu Screen to go to group members

While in the group instance, members join the Mini-game or Realtime game in the scene. The group members are guaranteed to play the game together, without the possibility of the game becoming full from non-group members.

The Group Scene persists as long as at least one member remains in the instance. If a user leaves the group while in a Group Scene they are automatically relocated to their Personal Space.



Prerequisites

Scene name - The unique name of the scene that the GroupDoor links to. This name is given with the Scene Key, and must be requested from the Home Support Team on https://home.scedev.net.

Two Scenes - One scene that has the GroupDoor, and one scene that the GroupDoor links to.

How to Use GroupDoor



Script

1. Create a GroupDoor by calling the function `GroupDoor.Create()`.
2. Check that `GroupDoor.Create()` does not return nil.
3. Relocate the user to the Group Scene by calling `GroupDoor.Enter()`.



Object

4. In the Object Editor, select the object and add a Group Door component.
5. Finish and package the object as normal.
6. In the Scene Editor:



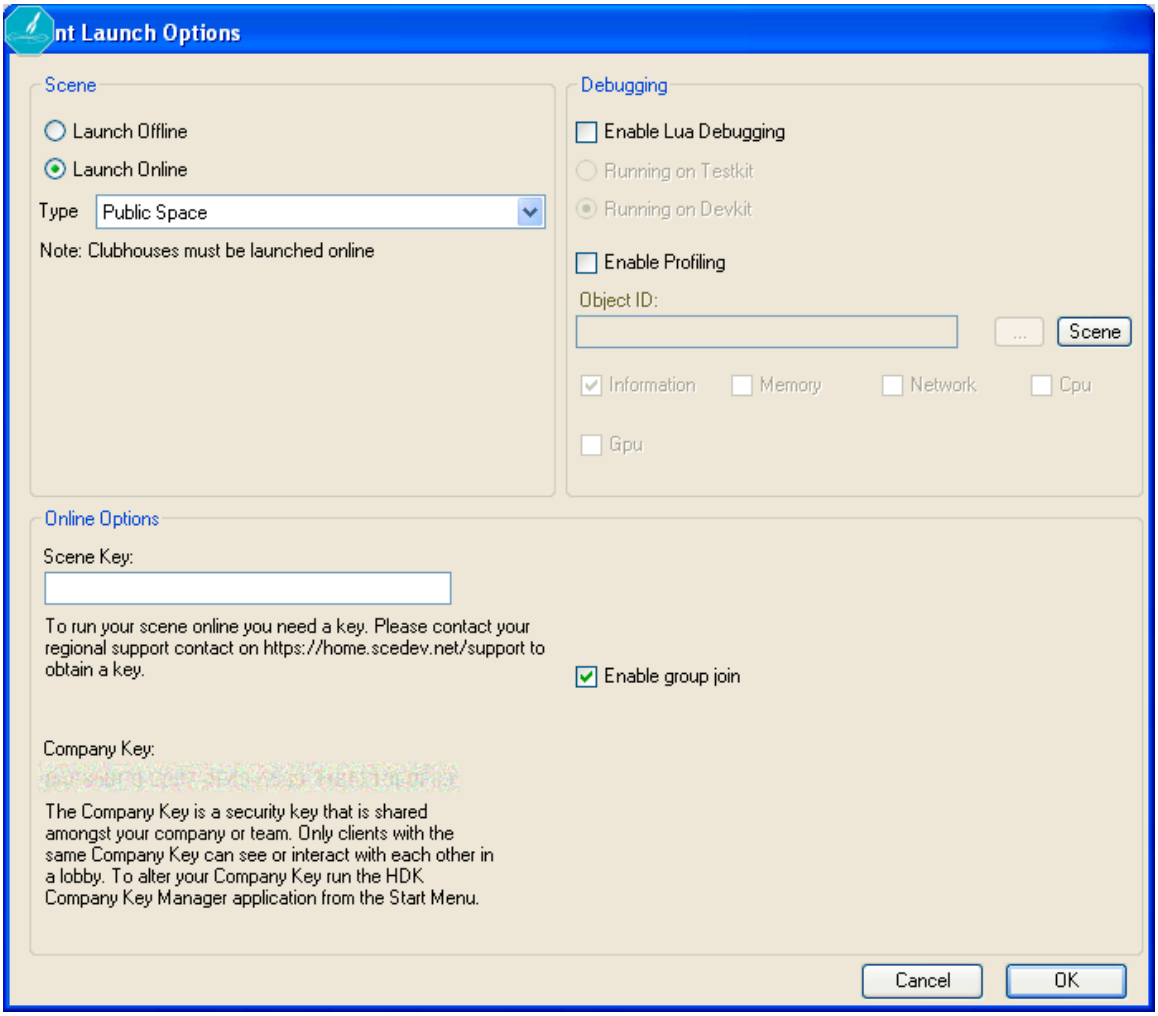
a. Embed the GroupDoor object in the GroupDoor scene. Create and package this scene as normal.



b. In the Group Scene, create and package the scene as normal.

When ready to test the GroupDoor functionality online and push it to the live client select PS3 > Launch Options.

The following dialog opens:



c. Select Launch Online.
d. Change Type to Public Space.
e. Select Enable group join.

7. In the Content Delivery System (CDS), when using GroupDoor, there are two scenes: one that contains the GroupDoor, and one that the GroupDoor links to:



a. GroupDoor Scene: When submitting the scene with the GroupDoor, there are no special steps. Submit this scene as normal.

⚠ The GroupDoor does not work until the Group Scene is also submitted and flagged as a Group Scene by your regional SCE.



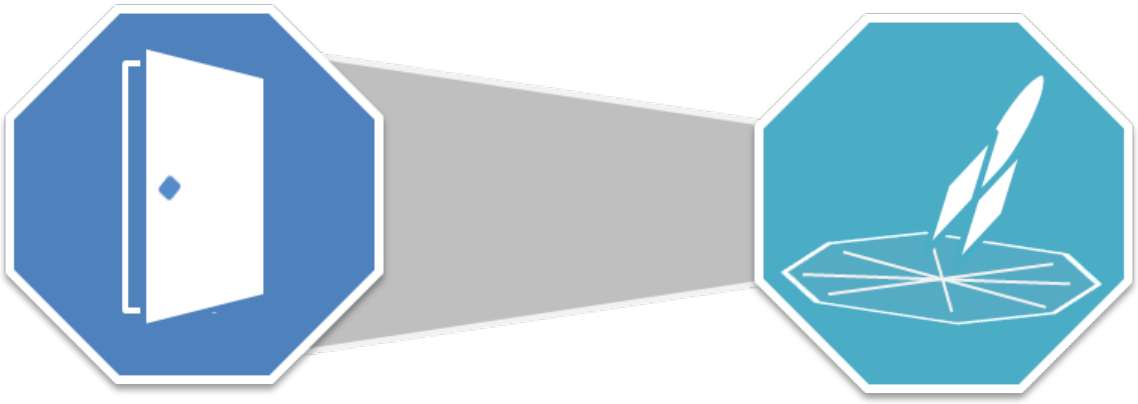
b. Group Scene: When submitting the scene that the GroupDoor links to, there are no special steps in the CDS. However, you must tell your regional SCE that this is a Group Scene. The regional SCE will mark this scene as a Group Scene. The GroupDoor cannot be tested until this scene is marked in the CDS as a Group Scene.

Where to Use GroupDoor

The GroupDoor can be placed in a:

- Mini-game script
- Realtime game script

Though, GroupDoor is best suited in a Mini-game, since the useful callbacks and trigger are already in place.



Place the GroupDoor object in one scene. The GroupDoor can only link to a separate scene.

Custom Messages

By default, if a user is unable to enter a GroupDoor the client displays a message. To turn this off and use custom OSD and custom font to present the error message:

1. When creating the GroupDoor, set isSimpleDoor to `false`. (i.e. `GroupDoor.Create( sceneName, false )`)
2. Whenever `GroupDoor.Create()` is called, if it returns `nil` call `GroupDoor.GetFailedEnterMessage()`, and present this message to the user.

Guidelines

- The GroupDoor must link to a separate scene.
- GroupDoor cannot be used in Personal Spaces.

- The Group Scene must be a Public Space.
- The term "GroupDoor" is developer-facing terminology and should not be presented to users. When using this feature, please describe it in more user-friendly terms, for example: "Play this game in a Group Space", "Play this game with your group", or "To play this game with your group, go through this door".

Groups

- GroupDoor is limited to groups of at least 2.
- Groups are limited to 8 users. Keep this in mind when designing games for groups.
- A single group may have multiple concurrent group instances of different group spaces (up to a maximum of 1 per group member), but may not have multiple group instances of the same space.
- If a user who is not a member of a group attempts to use a GroupDoor they will see a message saying that they cannot enter. If a user is in a group that contains no other users, they will also see a similar message.
  There is currently no way to query the following:
- The number of users in a group.
- The ID of each user in a group.
- If a user is in a group.

CDS

- To test a GroupDoor, your regional SCE must flag the Group Scene as a 'Group Scene'.

Using the Clubs Library

The Clubs Lua library is designed to expose read-only information about a player and the clubs they are a part of (their membership). This information can be used by scripted content in PS Home, all centered around the club membership of a player.

The Clubs library is a series of back-end Lua functions that enable developers gain information concerning the clubs a player is in. As a general Lua library, you can use it in any PS Home content that support Lua scripts (e.g. a scene object in a clubhouse scene). By making this information available, you can design your content to react to specific details relating to clubs. For example, the public space for a game such as SOCOM© can include an "Officer's Club" that allows access only to subleaders or leaders of clubs that have a certain number of members.

You can use the functions to find out more about which groups a player belongs to. You can also use them in conjunction with other libraries (such as User and Person) to find out whether other members of the clubs the player is a member of, are online. This information tells you which players have a connection, and probably have similar interests, so that you can design activities and communication around these groups.

You can use the Clubs library functions wherever Lua content can be executed, including:

- Mini-games
- Realtime games
- Actives
- Scenes (e.g. a clubhouse, personal space or public space)

The only requirement is that the player must be a full member (that is, a member, subleader or leader) of at least one club, for the functions to have any effect.

The Clubs library is largely an independent library that has no direct dependencies on other functions or resources. However, it does have some links with the User and Person libraries due to the exposing of such objects when retrieving information on a club member. For example:

- A script can retrieve the User object for a club member using Clubs.GetMemberList, then use User.IsInHome() to see whether that person is currently in PS Home.
- Similarly, the names of players allow you to use the Clubs library in conjunction with the AsyncCommand library, to utilize NP functionality.

Restrictions

To use the Clubs library, the PS Home client must be operating in online mode. If you are working in offline mode, the Clubs library functions report that the player does not belong to any clubs, irrespective of whether or not they do.

The player must be a full member of a club to acquire complete information about it (e.g.players invited into a club cannot view the club's member list).

Due to the restrictions placed on the Clubs system within PS Home, content should be designed around these points:

- A player can be associated with a maximum of 5 clubs.
- A club can contain a maximum of 32 members.
- Club data is not refreshed within 5 minutes of a previous refresh.

Use Cases

At present, the primary use of the Clubs library is to retrieve information about the clubs the local player belongs to.

Basic Reporting

The Clubs library can report basic information about player's clubs, as well as information on other members in each of these clubs. For example:

```
<CODE>

-- A typical function, printing the member list for each of the local player's clubs to the console.

function PrintClubMembers()

   local clubCount = Clubs.GetClubCount()

   if clubCount == 0 then

      print("You are not part of any clubs.")

      return

   end

   local clubs = Clubs.GetClubList()

   for clubIndex,club in pairs(clubs) do

      print(club.name)

      local memberCount = Clubs.GetMemberCount(clubIndex)

      if memberCount > 0 then

         local memberList = Clubs.GetMemberList(clubIndex)

         for memberIndex,member in pairs(clubMembers) do

            print(string.format("%d %s %d/%d", memberIndex, member.name, member.status, member.rank))

         end

      end

      print("-----")

   end

end

</CODE>
```

Reporting Club Members in a Player Instance

You want to show not only all the members in the club, but also which of them are in the same scene instance as the player (e.g. in the clubhouse, in a game lobby).

In this case, the two lists must be in separate sets:

- the club member list
- the list of club members in the same scene as the player

If you want the two sets of information two together, you must combine the two lists in the Lua script, because the flag denoting who is in the same scene is not provided in GetMemberList(). For example:

```
<CODE>

-- A more complex version of the typical function, this time also reporting whether the member is in
the same scene instance.

function PrintClubMembersWithOnlineStatus()

   local clubCount = Clubs.GetClubCount()

   if clubCount == 0 then

      return

   end

   local clubs = Clubs.GetClubList()

   for clubIndex,club in pairs(clubs) do

      print(club.name)

      local memberCount = Clubs.GetMemberCount(clubIndex)

      if memberCount > 0 then

         local memberList = Clubs.GetMemberList(clubIndex)

         local instanceMemberList = Clubs.GetInstanceMemberList(clubIndex)

         for index,member in pairs(memberList) do

            -- The index numbers from GetInstanceMemberList correspond to those from GetMemberList. If
there's an member of the same index in both, then that member is in the instance.

            if instanceMemberList[index] then

               member.isHere = true

            else

               member.isHere = false

            end

         end

         for memberIndex,member in pairs(clubMembers) do

            print(string.format("%d %s %d/%d \-\- %s", memberIndex, member.name, member.status,
member.rank, member.isHere))

         end

      end

      print("-----")

   end

end

</CODE>
```

Reporting Exclusive Clubs

The Clubs library also lends itself to more exotic use cases. For example, a military-type game (such as SOCOM©) may feature a public space as well as another scene that acts as an 'Officer's Club'. This club can be entered only by leaders and subleaders of large clubs. You can place a 'door' in the public space, acting as the only way in and out of the Officer's Club. To handle

the checks, you can place an auto-join mini-game in front of the door, which checks the players' credentials as they move in front of it. For example:

```
<CODE>

local MINIMUM_CLUB_SIZE = 16

function OnLocalPlayerJoin()

   local allowEntry = false

   local clubCount = Clubs.GetMemberCount()

   if clubCount > 0 then

      local clubList = Clubs.GetClubList()

      for index,club in pairs(clubList) then

         local playerIsOfficer = (club.role == ClubRole.Leader) or (club.role == ClubRole.Subleader)

         if (playerIsOfficer == true) and club.memberCount >= MINIMUM_CLUB_SIZE then)

            allowEntry = true

            break

         end

      end

   end

   if allowEntry then

      LocalPlayer.Relocate("Officers_Club")

   else

      -- Calls a typical user function that would tell the player that only officers of larger clubs
are allowed in.

      ShowWarningToPlayer()

   end

end

</CODE>
```

Clubs Library Design

The primary use of the Clubs library is to retrieve information about the clubs the local player is in. If you want only the number of clubs or club members, use the Clubs.GetClubCount/GetMemberCount functions instead of GetClubList/GetMemberList, which count the number of entries.

You should also design content to handle scenarios where use of the Clubs library is restricted, including:

- The player is not a full member of any clubs.
- The player has been invited to join a club but has not accepted the invitation.
- The player has requested to join a club but the club leader has not accepted.

More exotic uses may involve using the Clubs library as a miniature social network within PS Home. For example:

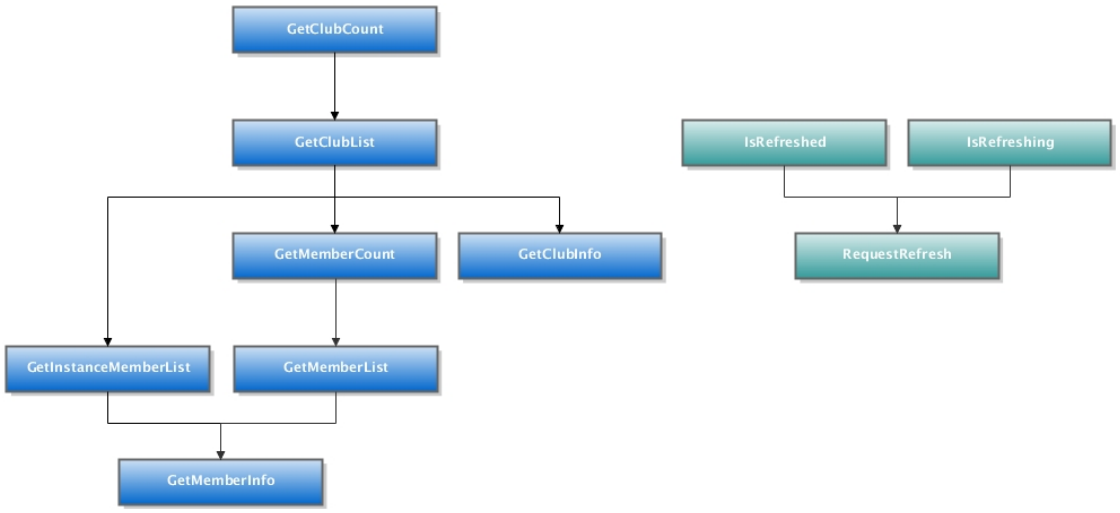
- When the script has queried the clubs the player belongs to, it can then retrieve the members lists for each club and combine them into a list of players that the player shares membership with in at least one club. You might use this information for things such as sending Game Launch invites to specific members of the various clubs.

- Similarly, because some Clubs functions provides a User or Person object for representing club members, any functionality in these libraries can be theoretically invoked by the Clubs library (e.g. inviting a club member to the

player's location such as the clubhouse or a public space).

Because queries require network traffic to the PSN servers, do not design club data to be refreshed automatically by scripts. Refreshes should be performed only by actions invoked by the player (for example, a player browsing a club's member list for people to invite into a Game Launching session). Where possible, cache Club information.

Content that allows players to invoke a refresh directly should contain safeguards to prevent the player from refreshing too often. For example, you could include a button that invokes `RequestRefresh` to check that the last time data was refreshed at least 5 minutes ago.



Using the Clubs Library

To use the `Clubs` library:

1. Include a Lua script into content that is capable of running it (e.g. a mini-game). Include the normal requirements depending on the type of content (e.g. an `Update()` loop for scripts).
2. Load the `Clubs` library `LoadLibrary("Clubs")`.
3. Run the content in the client:

```
LoadLibrary("Clubs")
```

Getting the Number of Clubs a Player Belongs to

Call `Clubs.GetClubCount()`. The return value will be the number of clubs the player is in:

```
local numClubs = Clubs.GetClubCount()
```

Getting the Club List for a Player

1. Call `Clubs.GetClubCount()` to determine how many clubs the player is in. If the return value is zero, then the player is not a member of any clubs.
2. Call `Clubs.GetClubList()` to retrieve a table, containing basic information about each club:

```
if Clubs.GetClubCount() > 0 then

   local clubList = Clubs.GetClubList()

   for index,club in ipairs(clubList) do

      print(string.format("#%d is ID %s, '%s' with tag '%s' and %d full members", index,
club.id, club.name, club.tag, club.memberCount))

   end

end
```

Getting Additional Information for a Club (via the Index)

1. Retrieve the club's list using `Clubs.GetClubList()`.
2. Call `Clubs.GetClubInfo()` with the index of the club to retrieve a table containing further information:

```
if Clubs.GetClubCount() > 0 then

   local clubList = Clubs.GetClubList()

   for index,_ in ipairs(clubList) do

      local clubInfo = Clubs.GetClubInfo(index)

      print(string.format("#%d is '%s', created %s with the profile: %s", index, clubInfo.name,
Time.GetDate(clubInfo.dateCreated), clubInfo.description))

   end

end
```

Getting Additional Information for a Club (via Club ID)

1. Retrieve the ID of the club (e.g. by caching the id parameter of `Clubs.GetClubList()`). The player must be Invited or a full member of this club.
2. Call `Clubs.GetClubInfo()` with the ID to retrieve a table containing further information about the club:

```
local clubInfo = Clubs.GetClubInfo(storedClubId)

print(string.format("ID %s is '%s', created %s with the profile: %s", storedClubId,
clubInfo.name,
Time.GetDate(clubInfo.dateCreated), clubInfo.description))
```

Getting the Member List for a Club (via Index)

1. Retrieve the club's list using `Clubs.GetClubList()`.
2. Call `Clubs.GetMemberList()` with the index of the club to retrieve a table containing the members of the club:

```
if Clubs.GetClubCount() > 0 then

   local clubList = Clubs.GetClubList()

   for clubIndex,_ in ipairs(clubList) do

      if Clubs.GetMemberCount(clubIndex) > 0 then

         local memberList = Clubs.GetMemberList(clubIndex)

         for memberIndex,member in ipairs(memberList) do

            print(string.format("#%d is %s", memberIndex, member.user:GetName())

         end

      end

   end

end
```

The memberCount attribute returned in Clubs.GetClubList() and Clubs.GetClubInfo() should not be used for iterating through the member list, since the value will exclude invited members. Use ipairs() for iterating through member lists instead.

Getting the Member List for a Club (via Club ID)

1. Retrieve the ID of the club (e.g. by caching the id parameter of Clubs.GetClubList()). The player must be a full member of this club.
2. Call Clubs.GetMemberList() with the ID to retrieve a table containing the members of the club:

```
if Clubs.GetMemberCount(storedClubId) > 0 then

   local memberList = Clubs.GetMemberList(storedClubId)

   for memberIndex,member in ipairs(memberList) do

      print(string.format("#%d is %s", memberIndex, member.user:GetName())

   end

end
```

The memberCount attribute returned in Clubs.GetClubList() and Clubs.GetClubInfo() should not be used for iterating through the member list, since the value will exclude Invited members. Use ipairs() for iterating through member lists instead.

Getting the List of Members in the Same Scene Instance (via Index)

1. Retrieve the club's list using Clubs.GetClubList().
2. Call Clubs.GetInstanceMemberList() with the index of the club to retrieve a table containing the club members inside the same instance. The player must be a full member of the club:

```
if Clubs.GetClubCount() > 0 then

   local clubList = Clubs.GetClubList()

   for clubIndex,_ in ipairs(clubList) do

      if Clubs.GetMemberCount(clubIndex) > 0 then

         local memberList = Clubs.GetInstanceMemberList(clubIndex)

         for _,member in ipairs(memberList) do

            print(string.format("%s is in the instance", member.user:GetName())

         end

      end

   end

end
```

Getting the List of Members in the Same Scene Instance (via Club ID)

1. Retrieve the ID of the club (e.g. by caching the id parameter of `Clubs.GetClubList()`). The player must be a full member of this club.
2. Call `Clubs.GetInstanceMemberList()` with the ID to retrieve a table containing the club members inside the same instance.

```
if Clubs.GetMemberCount(storedClubId) > 0 then

   local memberList = Clubs.GetInstanceMemberList(storedClubId)

   for memberIndex,member in ipairs(memberList) do

      print(string.format("%s is in the instance", member.user:GetName())

   end

end
```

Refreshing Club Information

1. Call `Clubs.IsRefreshed()` to determine whether the existing club data is up-to-date.
2. If the club data is not up to date, call `Clubs.RequestRefresh()`. This must not be called automatically but instead invoked by the player (e.g. pressing R1).
3. Call `Clubs.IsRefreshing()` to determine whether the refresh is still in progress.
4. Update all relevant objects (e.g. OSD objects) once `Clubs.IsRefreshing()` returns false and `Clubs.IsRefreshed()` returns true:

```
function OnUpdate()

   if Clubs.IsRefreshed() then

      UpdateInterface()

   elseif not Clubs.IsRefreshing() then

      if Pad.WasJustPressed(Pad.GetPad(1), REFRESH_CLUBS)

         Clubs.RequestRefresh()

      end

   end

end
```

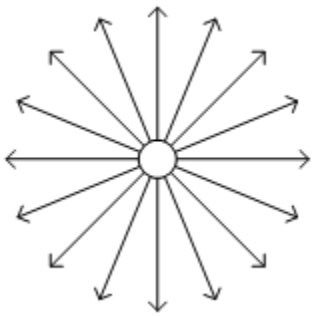
Adding Dynamic Lighting

You can use Lua to script and add dynamic lighting to your scene. You can use the following types of lighting:

- Point lights
- Spot lights

Point Lights

Point lights emit light in all directions, similar to a bare light bulb suspended in the air.



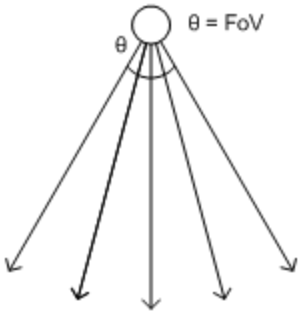
To create a point light, first define and configure a new entity, then create a new light, then assign the light to the entity.

```
entity = Entity.create()
-- Entity configuration options (e.g. SetPosition) here
pointLight = Light.Create( "point" )
-- Light configuration options (e.g. SetColor) here
Entity.SetLight( entity, pointLight )
```

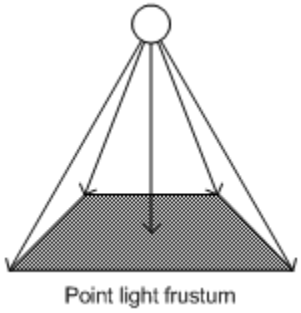
After you apply a light to an entity, you can manipulate it using the `Entity` functions in the Lua API (position, rotation, etc).

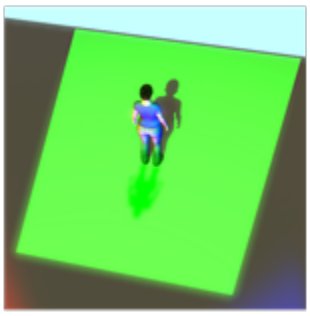
Spot Lights

Spot lights shine light in a given direction (along the +z axis), creating a four-sided pyramid of light over an area. The area covered by a spot light is determined by setting a Field of View (FoV) parameter, the wider the angle, the greater the area covered by the light.



On a flat plane, the resultant area covered by a spot light will be square.





To create a spot light, first define and configure a new entity, then create a new light, then assign the light to the entity.

```
entity = Entity.create()
-- Entity configuration options (e.g. position) here
spotLight = Light.Create( "spot" )
-- Light configuration options (e.g. color) here
Entity.SetLight( entity, spotLight )
```

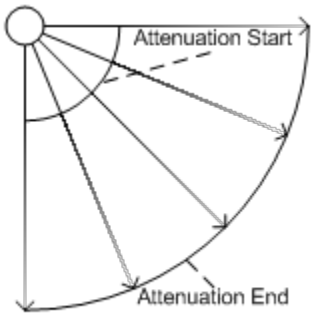
Once a light is applied to an entity, it can then be manipulated by all of the `Entity` functions in the Lua API (position, rotation etc).

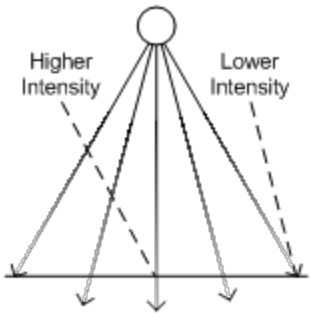
Configuring lights

Attenuation

Light attenuation affects the way in which the intensity of a light diminishes as the distance from a light increases. There are three parameters for configuring light attenuation:

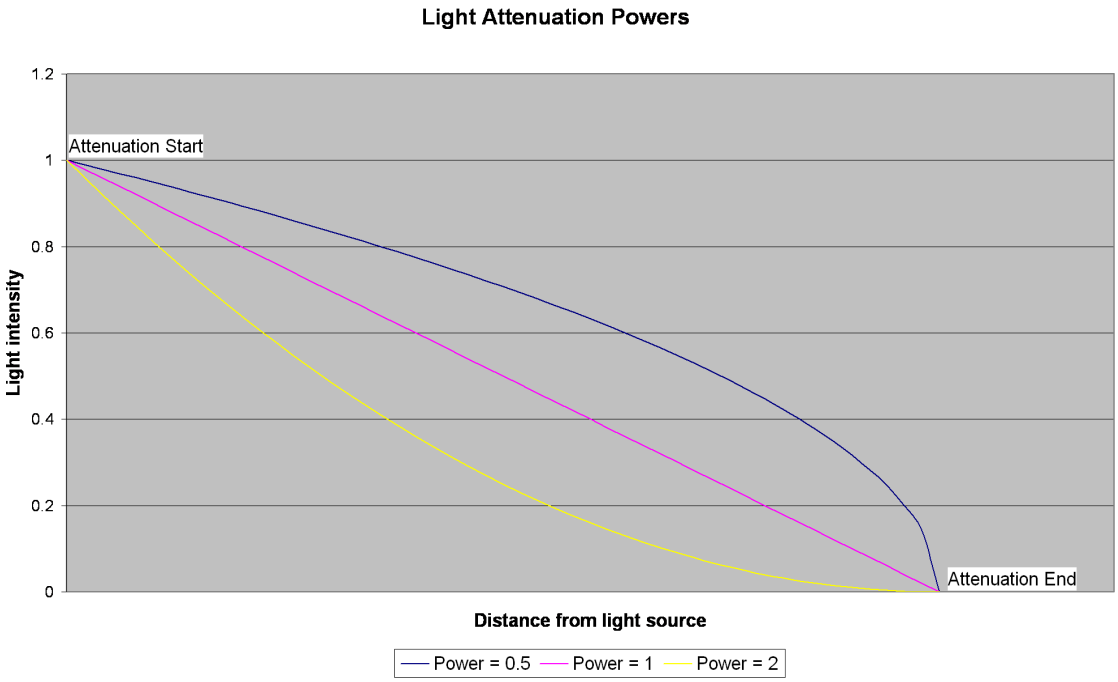
- Attenuation Start: Defines the distance at which attenuation begins to act on a light. For example, if this value is set to 1, objects within 1 meter of the light will be lit with full intensity.
- Attenuation End: This function defines the distance at which attenuation reduces a light's intensity to zero. For example, if this value is set to 5, objects more than 5 meters from the light source will not be lit by the light.







- Attenuation Power: Light attenuation power affects the gradient of light intensity created as distance from the light source increases. A light attenuation power of 1 will result in a linear gradient, reducing the power below 1 results in the light intensity staying higher over more of the gradient, but dropping off more quickly at the end, and a power greater than one results in a more gradual drop off, creating a 'softer' effect.



Light masks

Light masks allow you to control which areas covered by a light are illuminated. By default, the area illuminated by a spot light will cover a square area, but with the use of a light mask, the area can be any shape or pattern. A light mask is a grayscale image that is applied to a light; areas of the mask that are black will not be illuminated by the light once the mask is applied.

As spot lights emit light in only one direction, the light mask required is a simple 2D image.



Point lights, however, emit light in all directions and so a cube map is required in order to declare a light mask for each axis (both positive and negative).



When creating a cube map for you scene, it is worth noting which square of the map relates to which direction in your scene; these directions are as follows:



To apply a light mask to a light, first declare the image you wish to use for your light mask, and then apply it to your light:

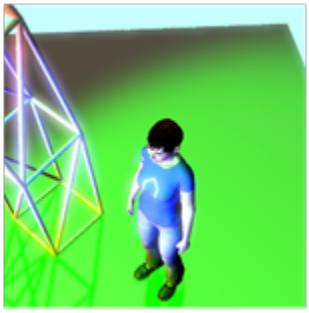
```
mask = Texture.Find( “myMask” )
spotLight = Light.Create( "spot" )
Light.SetMask( spotlight, mask )
Entity.SetLight( entity, pointLight )
```

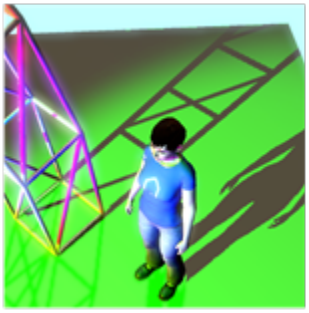
Light Color

The function `Light.SetColor()` allows you to set the color and the brightness of your light. The function takes a Vector4 as an argument containing RGB values for the color you wish to create (the W component of the Vector4 is ignored). These values are not clamped at 1.0 and higher values will result in brighter lights (lower values will also result in dimmer lights).

Spot Light Shadows

The function `Light.SetSpotShadow()` controls whether a spot light casts a shadow when its light is obstructed by a model. When enabled, all models in a scene will cast shadows by the spot light. Note that shadows are only cast by models added to the scene and not by geometry that is part of the environment and that this feature is only available for spot lights.





Spot Light Field of View

The function `Light.SetSpotFov()` declares the angle of a spot light's field of view. The wider the angle, the wider the area the spot light will cover. The default value is 45 degrees. This function cannot be applied to point lights as they always emit light in all directions. Note that this feature is only applicable to spot lights.

Animating Lights

Through an `Update()` function in a Lua script, you can change the parameters of lights frame-by-frame, producing animation. Any parameters can be changed to produce different effects, including:

- Cycle RGB color values to change a light's color over time
- Increase/Decrease RGB color values to produce a pulsating light effect
- Change the position/rotation values of lights

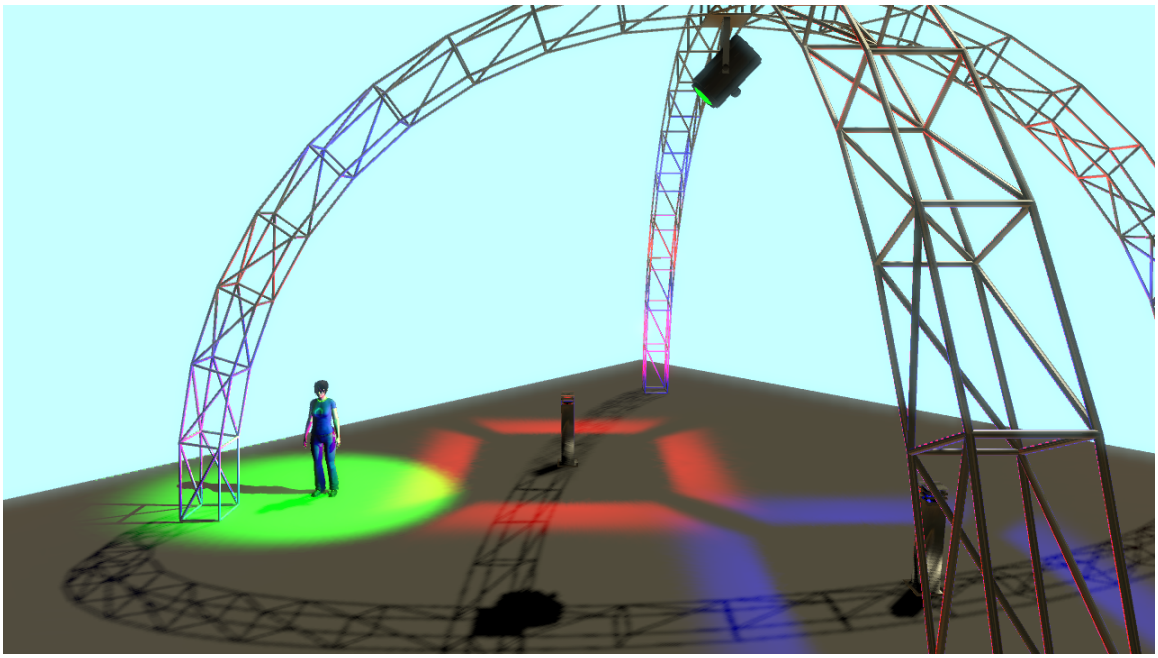
For example, the following function will produce a pulsating green light, gradually alternating the green component of the light between 4 and 1.5:

```
-- Values used to aid pulsating lights
intensity = 1
sign = 1

function Update()
   -- Adjust sign for +ve or -ve
   if (intensity >= 4) then
      sign = -1
   end
   if (intensity <= 1.5) then
      sign = 1
   end
   intensity = intensity + (0.02 * sign) -- Adjust intensity up or down
   pointLight:SetColor(Vector4.Create(0, intensity, 0)) -- Set new color
end
```

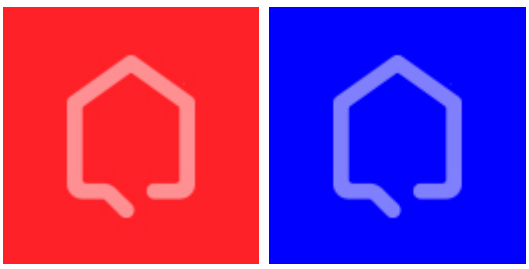
The Light Example

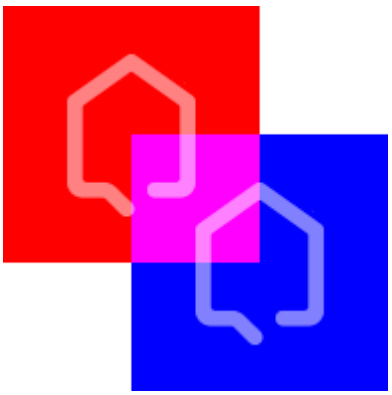
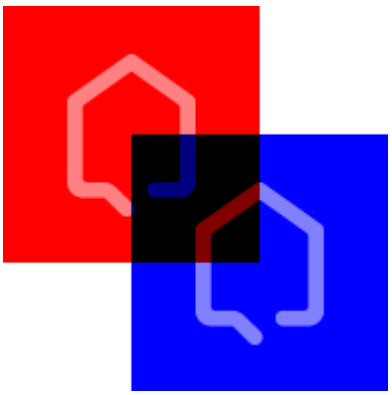
An example scene, LightTutorial is available that demonstrates all of these features through the included Lua script, SceneScript.lua. The scene features a green spot light suspended from a metalwork truss that tracks the player as they move around the scene. In addition to the spot light, two point lights (one red, one blue) are included, as the player moved towards these lights, the green color from the spot light will mix with the red and blue lights to produce yellow and cyan light respectively. The point lights are scripted so their intensity increases and decreases, producing a pulsating effect, and example light masks are used on all lights.



2D Blending

The Lua `BlendMode2d` API is an extension of the `Renderer` library that allows you to blend overlapping 2D elements. This increases the options you have on how elements, such as sprites and text, are rendered on Lua screens and in a user's main view. 2D blending can only be used wherever elements can be drawn through a Renderer object; the blend modes are restricted to entities accessed through the `Renderer` API (e.g. video screens cannot access 2D blending). The following blend modes are available:



Blend Mode	Description	Example
Blend	This is the default blend mode. For a given pixel, the RBGA of that pixel is combined with the RBGA of the pixel behind it to produce a blended result. The degree of blending is based on the front pixel's transparency.	
Add	This mode adds the RBGA values of two elements.	
Multiply	This mode multiplies the RGBA values of two elements together.	
XOR	This mode applies an exclusive OR (XOR) comparison to each RBGA component.	

Rendering an Element in a Different Blend Mode

`Renderer.SetBlendmode2d` requires a renderer and an element to render (e.g. quad, sprite, text).

To render an element in a different blend mode:

1. Create a renderer as normal inside of Lua content (e.g. a mini-game).
2. Call `Renderer.SetBlendmode2d()` during the `on_render` loop.
3. Draw any elements you wish to render. Each element will be blended with the pixels behind it.
4. To use a different blending mode within the same frame, call `Renderer.SetBlendmode2d()` with the new blending mode.

For example:

```
renderer = Renderer.Create()

exampleSprite = Sprite.Create()


function OnUpdate()


    Renderer.SetBlendmode2d(renderer, Blendmode2d.Multiply)

    Renderer.DrawSprite( renderer, exampleSprite)


end
```

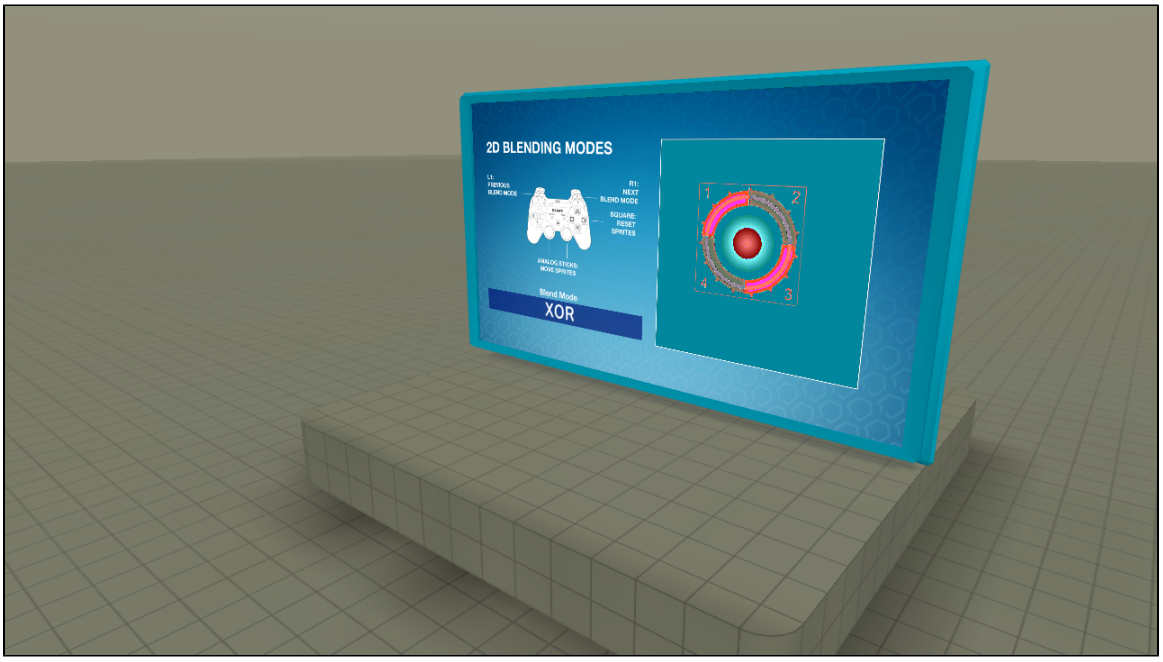
Loading the 2D Blending Sample

The 2D Blending sample demonstrates `Renderer.SetBlendmode2d`, and the modes that can be set.

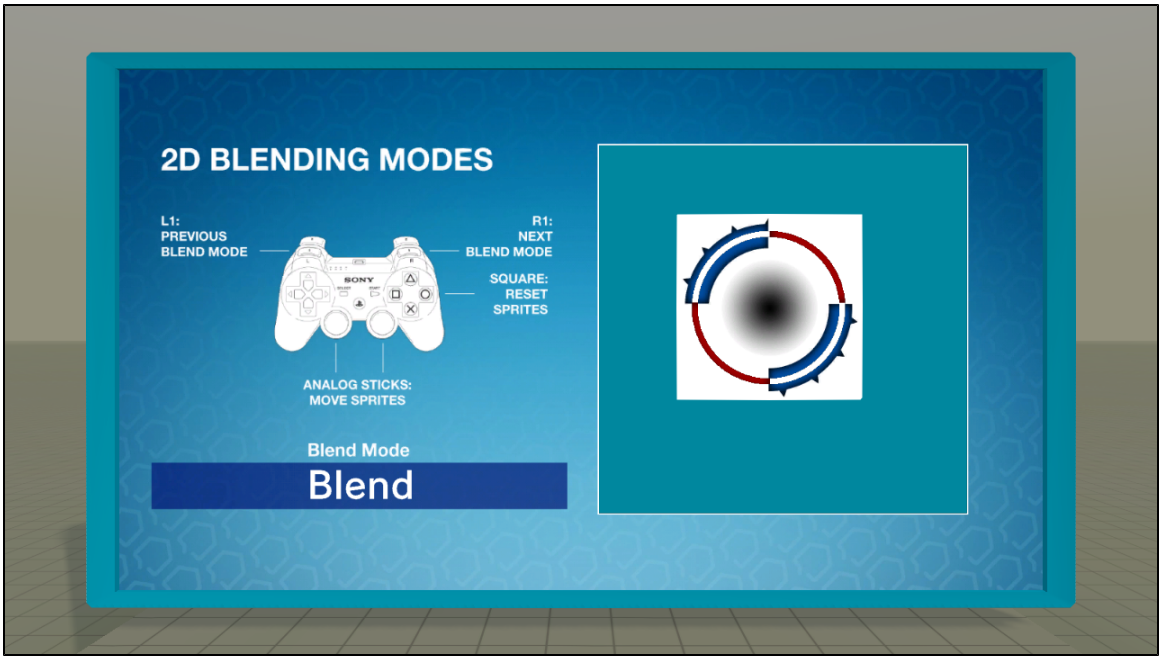
The sample has been designed so that its primary subject (i.e. the 2D Blending Modes) is kept within the main.lua script. The call to `Renderer.SetBlendMode2d` is wrapped inside the `OnRender()` function, with the rest of the script highlighting other actions required to use the feature.

To load the sample:

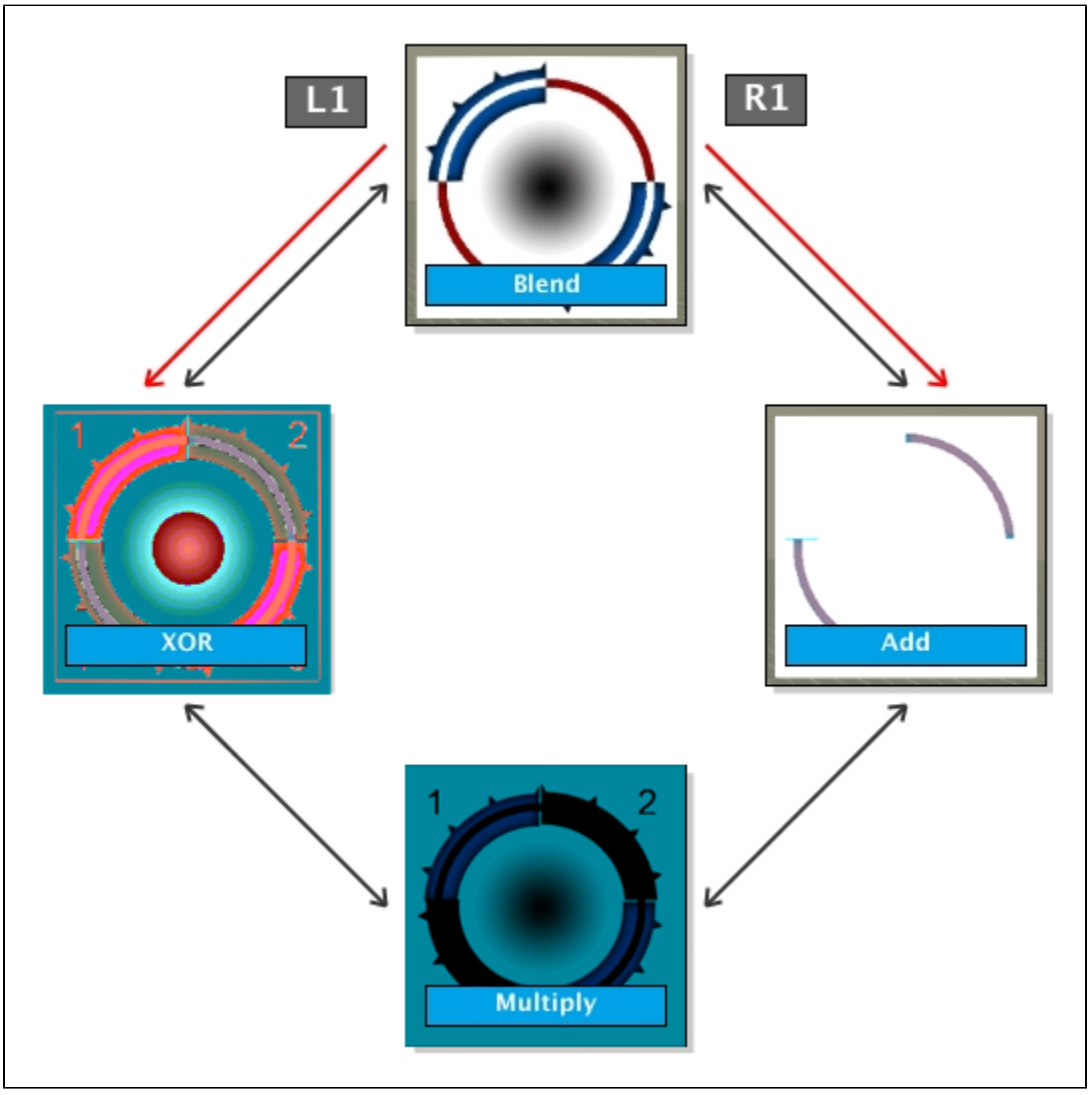
1. Download the 2D Blending sample from https://home.scedev.net/projects/samples.
2. In the Scene Editor, open MinigameTutorial.scene. You are prompted to update it to a .scproj file.
3. Select Lua Game in the Project panel.
4. In the Properties panel, set the Game ID field to the 2D Blending sample.
5. Click the Launch Home button  on the toolbar. When you join the scene, a screen appears:



The screen has two blendable sprites that demonstrates the various blend modes. Use the analogue sticks on the controller to move the sprites around and press square to reset the sprite:



The player can switch between the various blend modes using the L1/R1 buttons. As the blend modes are cycled, the sprites in the window render the effect.



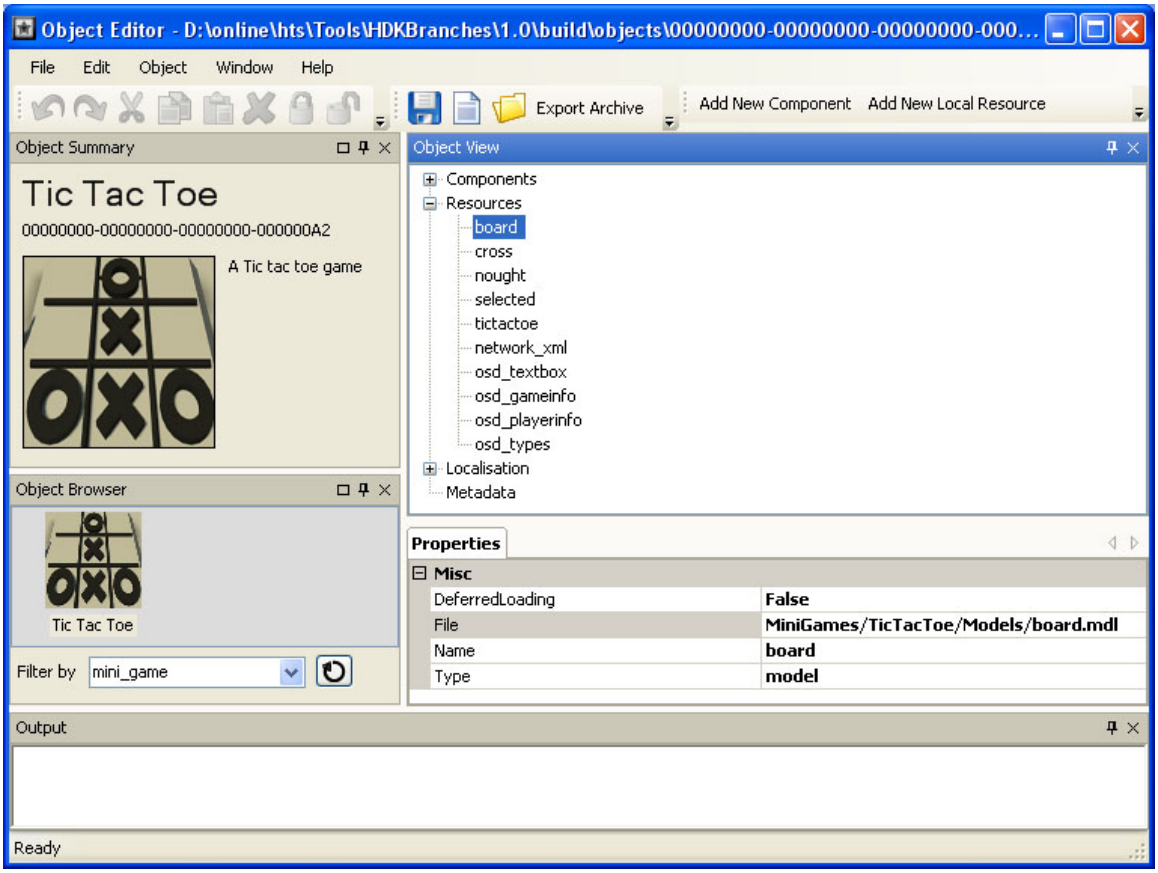
Blending Guidelines

- `BlendMode2d` can only be used with Renderer objects.
- To use the `Renderer.SetBlendmode2d` on a 2D screen, you must link the renderer to the screen.
- You cannot use `Renderer.SetBlendmode2d` to modify the rendering of 3D entities directly.
- Do not switch blend modes quickly, or allow users to switch blend modes quickly. Switching blend modes can cause a flashing effect and may induce photosensitive epilepsy.

Rendering 3D Graphics

When the TicTacToe game was prototyped, the actual logic of the game was tested using an ASCII rendering of the 'board' which was output to the console and TTY (see the `BoardPrint` function). However, in all likelihood, you are going to need to graphically represent your mini-game (or realtime game) within a lobby in some way.

Every Mini-game and Realtime Game object maintains a list of resources, which represent all graphical and audio assets referenced by the game. This allows the script to look up individual assets by a named key. For the TicTacToe game, the resource list is as follows:



The item list under the Resources node in the Object View panel makes four 3D model assets available to the game: 'board', 'cross', 'nought' and 'selected'. The board is the game grid and the model named 'selected' is used to represent the player's currently selected grid square. Let's examine the script code which loads and initializes the models representing the tokens that a player can place down on the board when a legal square has been selected:

```
-- Load in the 3D geometry for the game and add it to the scene
for row = 1,boardWidth do
   crosses[row] = {}
   noughts[row] = {}
   for col = 1,boardWidth do
      crosses[row][col] = Entity.Create();
      crosses[row][col]:SetVisible( false )
      crosses[row][col]:SetModel("cross")

      noughts[row][col] = Entity.Create();
      noughts[row][col]:SetVisible( false )
      noughts[row][col]:SetModel("nought")
   end
end
```

Each token (nought or cross) is retrieved from a dictionary of objects via its name 'key' as specified in the Object Editor. Then, for each board square, the script creates an instance of the nought and cross models. It is important to realize here that an entity simply references a model, they do not make copies. In other words, there is only a small memory overhead in creating an entity.

```
crosses[row][col]:SetVisible( true )
crosses[row][col]:SetRotationXyz(Vector4.Create(0, 45, 0))
crosses[row][col]:SetPosition(ptrpos)
```

When the models are first cached, they are set to invisible. When it is time to render them, their visibility needs to be set to true and their position and rotation need to be set. The position of a model will typically be some offset of the location of the mini-game or realtime game which can be obtained as follows:

```
theGame = Object.GetMe()
gamePos = theGame:GetInitialPosition()
```

This game is slightly wasteful in that it creates two player counters per grid square. It could have been coded such that a new player counter is created only when a move is taken, but the current method used in the script is there for simplicity.

Lastly, note that a rotation around the Y axis is applied to the 'cross' marker. This is because the geometry was modeled as a '+' instead of an 'X', so we require a rotation at runtime.

Customizing Displays

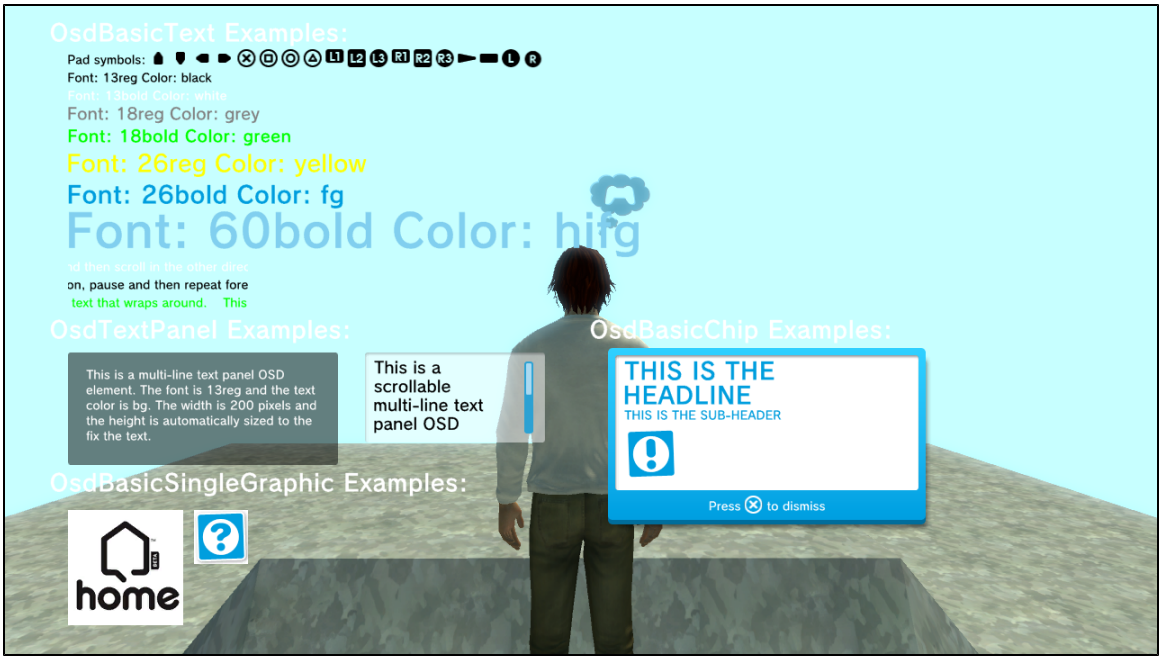
PS Home has a distinctive user interface with a consistent style. When writing a mini-game or realtime game script, you can access a family of Lua libraries to help you to render user interface elements in Home's style. These elements can be text, images or dialogs (referred to as 'chips').

The OSD API Sample

The HDK includes a sample that shows to create the available OSD element types. The sample comes in the form of a skeleton mini-game. When the local player avatar joins the mini-game, the screen will fill with a demonstration of the different elements types.

To run the sample:

1. Download the OSD API sample from https://home.scedev.net/projects/samples.
2. In the Scene Editor, open MinigameTutorial.scene. You are prompted to update it to a .scproj file.
3. Select Lua Game in the Project panel.
4. In the Properties panel, set the Game ID field to the OSD Example.
5. Click the Launch Home button  on the toolbar. When you join the scene, a screen appears:



For more information on OSD, see On Screen Display (OSD) Lua Library.

Setting the Rendered Text Font

There are several options for changing the look of rendered text, including text sizes, making text bold and custom typefaces.

The following font-related Lua functions have already been included in earlier versions of the HDK (in the Renderer Library):

- Renderer.SetFontHorzAlignment() Set rendered text to the left, center or right of the renderer area
- Renderer.SetFontScale() Scale text up and down in both the x and y axes independently
- Renderer.SetFontVertAlignment() Align text with the top, center or bottom of the renderer area

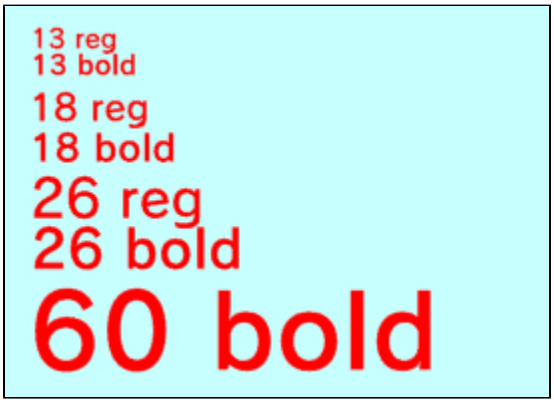
These original APIs are used to set the standard rendered text to almost any size and position - unfortunately scaling up too far results in pixelation and poor quality display.
The following API were added in HDK 1.3.2:

- Renderer.Setfont() Set rendered text to a choice of pre-defined sizes, with the option to set to bold.

The accepted arguments are:

- 13reg (default for Renderer)
- 13bold
- 18reg
- 18bold
- 26reg
- 26bold
- 60bold

These are shown below.



Custom Fonts

The Font library was added in HDK 1.50 in order to allow the use of custom fonts in scripted content. This library allows you to load a TTF font file in the form of a resource attached to your object, and use this font for rendering text using the function Renderer.SetFont. The font can be created at a custom size and/or slant if required.

For full details please refer to the Lua API Reference.

Best Practice

Larger fonts require a lot more memory than smaller fonts, so it is best practice to scale the smaller fonts up as far as you can. At the point the scaling no-longer displays properly, switch to the next font size up and start scaling again until you reach your desired size.

Using the OSK Library

Use the OSK (On-Screen Keyboard) Library to acquire user text input and save it as a string for use elsewhere in your script. You can open an instance of the OSK with a prompt for specific input from the user (name, age, password or a text-adventure style question for example). This can be useful in many ways, including the ability to add password protection (when coupled with Save Data).
There are four OSK functions:

- `Osk.Close()` - Closes any instances of the OSK opened by this script. If the OSK was opened by another script then it has no effect.
- `Osk.GetText()` - Obtains the text entered into the last instance of the OSK and returns it as a string. If the user canceled the OSK it returns `nil`.
- `Osk.IsAvailable()` - Checks to see if it's possible to open an instance of the OSK (only one can be open at a time).
- `Osk.Open()`- Opens a new instance of the OSK and allows the developer to set parameters (e.g. limiting the amount of characters, allowing only alphanumeric characters, etc).

> See `Osk.Open()` for a list and description of all parameters.

Running the Example

To run the OSK example:

1. Open MinigameTutorial.scproj in the Scene Editor.
2. Select the Mini-Game object.
3. In the Properties panel, browse and select the OSKExample in the Game ID field.
4. Click Launch Home to run the scene.

   The first OSK instance appears.

5. Enter up to the maximum of 511 characters.

   The text is displayed on the screen in the top left corner. This display uses the new `Renderer.SetFont()` function to set the rendered text to bold.

6. After entering your text the second OSK instance appears - this demonstrates `Password Mode`.

   Anything entered via the OSK while the password argument is set to `true` is displayed on the screen as asterisks. The entered text is again displayed on-screen via the `Renderer` Library.

   The third OSK is set to `Latin Only` - this only accepts standard Latin lettering and punctuation (ASCII characters 32-126). Accented letters are not allowed and neither are Asian characters.

   The last instance allows line breaks in the text input. This activates the `Line Break` button on the OSK itself and allows those with a physical keyboard attached to their console to use the ALT+ENTER keys to create a line break.

Using the XMB™ Media Lua Library

The `MediaLibrary` Lua library contains functions for searching, accessing, and writing to the media stored on the user's PlayStation®3. You can access pictures, music and videos using this library. You can also export pictures from PS Home into the user's photo library.

> Photos are the only media type that you can display using the media library search functions.

Searching for Media

When you start a new search, the MediaLibrary search functions must first be initialized by calling MediaLibrary.StartSearch(). This function takes no arguments and returns no values; it just allocates resources for the impending search. After initialization, you can apply a number of different search options, the most important of which you set by MediaLibrary.SetResultsToInclude(). This function declares which attributes of the files you are searching for should be retrieved.

Search options include:

- file: Returns a string containing the media's file path.
- thumbnail: Returns a string containing the media's thumbnail file path.
- title: Returns a string containing the title (XMB™ caption) of the media.
- width: Returns the width of the image (if applicable).
- height: Returns the height of the image (if applicable).

You can include more than one of these attributes in the search. You need one function call for each attribute requested.

When searching for images or image thumbnails, you must specify that these files should be prepared by display by the PlayStation®3. Use these functions:

- MediaLibrary.SetPrepareSearchFiles(true) for images
- MediaLibrary.SetPrepareSearchThumbnails(true) for image thumbnails

If you do not set these options and the images have not previously been prepared, the images are not displayed and are replaced with a 'Not Found' icon.

When you have configured the search, start the search using MediaLibrary.Search(). This function takes three arguments:

- PositionType: Specifies where to begin the search.
    - Use first or last to specify the beginning or the end of the library.
    - Use next or prev to pick up after or before the position where the previous search finished.
- MediaType: Determines what sort of file to search for. Choose video, music or photo.
- MaxCount: The maximum number of results to return in the search.

The search is asynchronous, so to determine when the search is complete, use the function MediaLibrary.IsSearchFinished(). When the search is complete, you can retrieve the results as a table by calling MediaLibrary.GetSearchResults(). Typical usage for these functions is as follows:

```
while MediaLibrary.IsSearchFinished() == false do
   coroutine.yield()
end
results = MediaLibrary.GetSearchResults()
```

When the search is complete, call MediaLibary.StopSearch() to clear any resources that were allocated to the search. To perform a new search you must then call MediaLibary.StartSearch() again.

Retrieving Data From a Search

Search results are stored as an indexed table, indexed from position 1 (the Lua standard). Results are labeled in the table using the same names as were used as arguments when calling the MediaLibrary.SetResultsToInclude() function. For example, results[6].width would be the width of the sixth file returned in a photo search.

A search returns the path of a file and not the file itself, so you must request files in order to use them. To do this, use the Resource library. For example:

```
myRes = Resource.Request( results[1].file, "texture" )
while ( Resource.IsLoading( myRes ) ) do
   coroutine.yield()
end
myTex = myRes:GetData()
mySprite = Sprite.Create()
mySprite:SetTexture( myTex )
```

Loading resources is asynchronous, so run loading in a co-routine that yields until the resource has finished loading.

⚠ The size limit on imported images 512x512px. Images larger than this are resampled to 512x512px.

Exporting Files from PS Home to the PlayStation®3

You can export images from PS Home to the user's PlayStation®3 media library, which can then be accessed through the XMB™. To use this facility, call `MediaLibrary.ExportImage()` or `MediaLibrary.Exportvideo()`, providing the filename of the file to export, as a string argument. This function is asynchronous, so run it as a co-routine.

The following sample code shows how to take a screenshot:

```
myFile = Screenshot.Create()

capturing = Screenshot.Capture(shot, false, false, false, false, false)

if (capture and not myFile:IsInProgress() and myFile:Succeeded() ) then
      MediaLibrary.ExportImage( myFile, "Home Screenshot" )

      while MediaLibrary.IsExporting() == true do
            coroutine.yield()
      end
end
```

The simplest use for this facility is to save screenshots and videos of PS Home back to the user's PlayStation®3. For example, if you combine this functionality with the Camera library to take a picture from a user-specified position, results in a simple camera that the user can use to take photos of scenes within PS Home.

You can set screens in the scene not to appear in a screenshot or video. Screens with the protected property set to `false` display. Those with the property set to `true` do not. To change a screen's protected property, call `Screen.SetProtected()` in your script. You can call this function for both Lua screens, and screens added to a scene through the Scene Editor.

Confirming the Image Export Success

Using the callback, you can confirm the screenshot has successfully been moved to the XMB™.

See the `MediaLibrary.ExportImage` function in the Lua API for more details.

The MediaLibrary Example

An example mini-game is available that demonstrates all the `Media Library` functions. In this example, users can select a photo from a range of thumbnails and then draw over it using a range of simple drawing tools. When finished, the user can then export a copy of their drawing back to their PlayStation®3's hard drive.

Controls for MediaLibrary Example

The controls differ, depending on whether or not the media browser is displayed.

Controls when the media browser is displayed:

- D-Pad Left/Right: Browse photos
- Cross: Select photo
- Triangle: Hide media browser

Controls when the media browser is not displayed:

- Left Analogue Stick: Move paintbrush
- Triangle: Show media browser
- Circle: Undo
- Cross: Draw
- Square: Toggle between drawing lines and rectangles
- L1/R1: Cycle colors
- R3: Export picture

For this example, the user must have at least one picture stored in their PlayStation®3's photo library.



PS Home Camera

A camera represents a viewpoint in a PS Home scene. It is defined by a view frustum, which is a 6-sided pyramidal shape which lies between two parallel planes. The shape of the frustum is controlled by:

- A near and a far clip plane. You can modify these planes to tune the resolution of the Z-buffer to avoid clipping of 3D objects and avoid 'Z-fighting' artifacts.
- A vertical field of view.
- An aspect ratio. The horizontal field of view is deduced from this value and the vertical FOV.

In your Lua scripts, you can:

- Query the default Home camera's properties.
- Create your own cameras and override the default Home camera.
- Attach a camera to an entity.

Querying a Camera

You can query a camera created by a script for various information about the camera at that point in time, regardless of whether it is the currently active camera. For a full list of the information that you can retrieve, see the Lua API Reference.

The example function below shows the type of information that can be queried by:

- Any camera in a scene (`GetCameraInfo()`)
- Only the active camera (`GetActiveCameraInfo()`)

```
function GetCameraInfo(targetCamera)

    -- Retrieve the 3D position of the camera (Vector4)
    local position = Camera.GetPosition(targetCamera)

    -- Retrieve the 3D position of look-at point (Vector4)
    local lookAt = Camera.GetLookAtPosition(targetCamera)

    -- Viewport field of view (in degrees)
    local vpFOV = Camera.GetViewPortFOV(targetCamera)
```

```
    -- Scalar distance between camera position and near clip plane (world units)
    local vpFar = Camera.GetViewPortZFar(targetCamera)

    -- Scalar distance between camera position and far clip plane (world units)
    local vpNear = Camera.GetViewPortZNear(targetCamera)

    -- Whether the camera is currently active (boolean)
    local isActive = Camera.IsActivated(targetCamera)

end

function GetActiveCameraInfo()

    -- Same as functions in GetCameraInfo() above, but no argument is given, making it retrieve
the default/active camera.
    local position = Camera.GetPosition()
    local lookAt = Camera.GetLookAtPosition()
    local vpFOV = Camera.GetViewPortFOV()
    local vpFar = Camera.GetViewPortZFar()
    local vpNear = Camera.GetViewPortZNear()

    -- Current projection matrix (Matrix44)
    local projMat = Matrix44.Create()
    Camera.GetProjectionMatrix(projMat)

    -- Current view matrix (Matrix44)
    local viewMat = Matrix44.Create()
    Camera.GetViewMatrix(viewMat)

    -- Inverse of current view matrix (Matrix44)
    local invViewMat = Matrix44.Create()
    Camera.GetInverseViewMatrix(invViewMat)
```

```
end
```

Retrieving Matrix Information

Use the Camera library to retrieve the inverse view matrix and the projection matrix on the active camera only. The Home client does not store matrix information for each camera, and can only retrieve matrix information on the currently active camera.

Overriding Home's Camera

When activating the TicTacToe game, the camera swoops to a position above the position of the mini-game and looks down at the center of the board. You can manipulate the camera at any point in your script. However, the TicTacToe game commandeers the camera when the local player joins the game and will relinquish it when the local player leaves. The code for this is as follows:

```
function CameraInitialize()
   -------------------
   -- Camera set up --
   -------------------
   camera = Camera.Create()
   gamePos = theGame:GetInitialPosition()
   camPos = Vector4.Create(gamePos:X(), gamePos:Y()+3, gamePos:Z()+3)
   camera:SetPosition(camPos)
   camera:SetLookAtPosition(gamePos)
   camera:Activate(2.0)
end

function CameraTerminate()

   ---------------------------------
   -- Deactivate the game's camera --
   ---------------------------------
   if (camera:IsActivated()) then
      camera:Deactivate(2.0)
   end
end
```

After you have specified the camera's position and target (note that the up vector for the camera is implicitly set to be the positive Y axis in world space), the camera is activated. The parameter passed to camera:Activate specifies the time taken to transition from the standard Home camera position/orientation to that of the mini-game. During the transition, the position of the camera is linearly interpolated and a spherical linear interpolation (SLERP) is performed on the orientation. This gives a pleasing switch between the two viewpoints.

On leaving the game, the local player's camera is 'deactivated' (providing the camera is already activated). This essentially performs the camera transition in reverse and restores the Home camera to its state before the mini-game began.

The Camera API also provides for the modification of the camera's projection properties: near and far clip planes (in meters), vertical field of view (in degrees) and viewport aspect ratio (width divided by height).

Attaching Cameras to Entities

You can control cameras more finely by attaching them to entities using the function Camera.AttachToParent. In such cases, the camera's world transform is inherited from that of the entity, allowing the camera to move as the entity would. This allows more complex movement, such as banking around the z-axis or being subject to Havok physics (such as a camera attached to an entity in freefall).

> While the current camera's world transform is modified by this method, the view and projection transforms remain unaffected.

For example:

```
function OnLocalPlayerJoin()

-- Creates a camera & entity

    g_entity = Entity.Create()

    g_camera = Camera.Create()

    -- Sets the entity a short distance above the origin

    Entity.SetPosition(g_entity, Vector4.Create(0, 5, 0))

    -- Attachs the camera to the entity.

    Camera.AttachToParent(g_camera, g_entity)

    -- Activates the attached camera. At this point, the player's view should be above the origin.

    Camera.Activate(g_camera)

end

function OnLocalPlayerLeave()

    -- Deactivates the attached camera, returning the player's view to normal.

    Camera.Deactivate(g_camera)

    -- Deattaches the camera from the entity.

    Camera.AttachToParent(g_camera, nil)

end
```

You can attach a camera to only one entity at a time. Successive calls to `Camera.AttachToParent()` for the same camera result in the latest call superseding all previous calls.

Due to the interpolation used when transitioning between cameras, we recommend avoiding specifying transition times when activating or deactivating cameras attached to entities.

Adding Network Communication

PS Home objects have access to the following mechanisms for sharing and passing state between clients over a network:

- Outbound Messages
- Welcome Messages
- Network Property Bags

To add networking support to your object, you must add the Network component to your object in the Object Editor. This component allows the object to send messages to and receive messages from other Home clients. It also supports a construct called the network property bag (`netpropertybag`) which transparently replicates shared state over a network.

For more information about the Network component properties available for editing in the Object Editor, see Network Component.

Outbound Messages

Sending Outbound Messages

The messaging system in PS Home is intended to for low bandwidth, potentially high latency connections. Create network messages using the `OutboundMessage` library; it contains functions to create, modify, and send messages to a single user or all of the users in a scene. The system sends a message (i.e. data) to other users through the PS Home severs in less than 1 second. Although the rate is typically, and more likely, faster, work under the assumption that a message takes 1 second to be delivered.

Turn-Based As Opposed to Realtime

The Network component's messaging system is designed to support turn-based games rather than realtime, low latency games (like a typical action game). For realtime games, see Realtime Games.

Message Size

The maximum message size you can send is approximately 1 KB. This is because PS Home is demanding from a network utilization point of view and script writers are encouraged to be conservative with the amount of data that they broadcast per frame.

When creating a new message, you need to know the size of the data (in bytes) you want to send. You set this size when you create the message in Lua using `OutboundMessage.Create()`. The size is based on the type of message you are sending (for example, `int32`, `float`, `memoryContainer`, etc).



To see how to calculate the size of a message, take a look at the example code in To Whom is the Message Addresed? below.

Message Data

Once the message has been created, you can write different data types into it. Lua supports numbers and strings but an `OutboundMessage` allows you more control over how data is written to the message's buffer. Instead of simply writing a number, you can choose to write a floating point value or an 8, 16 or 32 bit integer. Floats and 32-bit integers are 4 bytes, 16 bit integers are 2 bytes, and 8-bit integers are 1 byte. If you know that the number you want to send can be represented in 8 bits (-128 to 127), then sending this as an Int8 instead of an Int32, say, will save 3 bytes.



When sending strings, remember to account for a terminating character when calculating the size of the message. This can be worked out as follows:

```
size = string.len( mymessage ) + 1
```

To Whom is the Message Addressed?

When sending a message, you have some control over the receiver:

- Messages can be sent to all avatars in the same instance of a scene as the sender.
- Messages can be sent to specific avatars in the same instance of a scene as the sender.

The following example code shows how to send the various supported message data types:

```
-- Some information to broadcast
PERSONAL_DETAILS = 0
name   = "Joe Bloggs"
height = 1.85
age    = 33
weight = 280
id     = 12345678

local type = PERSONAL_DETAILS
-- Calculate the size of the message based on the
-- information we want to send
local size = 1 + 2 + 4 + 4 + string.len( name ) + 1

local message = OutboundMessage.Create( type, size )
OutboundMessage.WriteString( message, name )
OutboundMessage.WriteFloat( message, height )
OutboundMessage.WriteInt8( message, age )
OutboundMessage.WriteInt16( message, weight )
OutboundMessage.WriteInt32( message, id )


-- Send the message to all avatars in the sender's instance
OutboundMessage.Send( message )
```

If the message needs to be sent to a specific avatar, the send code would be:

```
OutboundMessage.Send( message, Person.GetId( otherPlayer ) )
```

Sending Messages Limits

The maximum message size of the NetworkComponent_Message is 1KB.

Sending Messages Best Practices



Do not send write your code so that it sends messages every frame.



Send messages/information only to those users who need it. For example, in a mini-game only send messages to other users playing the mini-game, and not everyone in the scene.



Instead of sending messages across the network, use network property bags when you can. See Network Property Bags below.

Receiving Outbound Messages

To receive network messages, you must define a function in your object's Network component, which has a property called on_received_message. Assume you set this to OnReceivedMessage. This function is effectively a callback that gets called by the Home runtime whenever a message needs to be sent to the client running the script. If you were to misspell this function and the runtime tried to inform your script that a message had been received, then the Lua interpreter will generate an error to the console and TTY.

The matching receive code for the message send described in the previous section is as follows:

```
function OnReceivedMessage()

   local message = ReceivedMessage.GetEventMessage()
   local type    = ReceivedMessage.GetType( message )

   if ( type == PERSONAL_DETAILS) then
      name   = ReceivedMessage.ReadString( message )
      height = ReceivedMessage.ReadFloat( message )
      age    = ReceivedMessage.ReadInt8( message )
      weight = ReceivedMessage.ReadInt16( message )
      id     = ReceivedMessage.ReadInt32( message )
   else
      -- Handle other message types
   end

end
```

Mini-game Welcome Message System

Welcome messages inform users entering a scene of the state of a mini-game in progress. For example, two users are playing a Chess game, when a third user enters the scene. The current game state must be sent to the new arrival so that they know the positions of the Chess pieces and can watch the game.

For more information about mini-game welcome messages, see Mini-game Design.

Realtime Games

Realtime games can use the Network component, Outbound Messages and Welcome Messages. These messages are not sent down any of the Realtime game's peer-to-peer pipelines. This feature is enabled to allow some communication with other users in the scene not playing the Realtime game.

For example, in a scene that contains a First Person Shooter Realtime game, a scoreboard could be in the general area of the scene. Users not in the Realtime game can see the current score of the Realtime game, which is updated using Outbound Messages.

For more information about realtime games, see Realtime Games.

Network Property Bags

A Network Property Bag (netpropertybag) is a mechanism for replicating data across multiple network clients. This can greatly simplify the sharing and synchronization of data between clients, without having to explicitly send messages. When new clients run the script, the contents of everyone else's property bags are replicated to the new joiner without any need for the script to take any action.

Property bags are created on a single client and replicated to all other clients in the same scene instance. Only the property bag owner can modify the contents of the bag. The implication is that your script will need to designate one person in the instance as the bag manager/owner, and all other clients as subscribers to the bag. Subscribers can read the contents of the bag but not write to it.

The property bags used by an object are defined in an XML file resource. With this file, you are able to specify types and update schedules to get the most efficient use out of the network. When a bag is defined, it is assigned a type id which is a hash of the name so that long strings to identify bag types don't need to be sent across the network.

As Lua does not define many native data types, the native side networking layer needs to be given a few hints as to how it should handle certain data values when sending them across the network. The network definitions XML file lets you define up front the property bags your script will use and how often each field within the bag will be updated.

An example property bag definition XML file is as follows:

```
<NETWORK_DEFS>

    <BAGS maxBags= "64" >

        <BAG name="player">

            <FIELD>

                <NAME>name</NAME>

                <TYPE>string</TYPE>

                <ARRAYSIZE>64</ARRAYSIZE>

                <MAXFREQUENCY>0</MAXFREQUENCY>

            </FIELD>

            <FIELD>

                <NAME>score</NAME>

                <TYPE>int32</TYPE>

                <MAXFREQUENCY>500</MAXFREQUENCY>

            </FIELD>

            <FIELD>

                <NAME>weaponAmmo</NAME>

                <TYPE>int32</TYPE>

                <ARRAYSIZE>4</ARRAYSIZE>

                <MAXFREQUENCY>500</MAXFREQUENCY>

            </FIELD>

        </BAG>

    </BAGS>

</NETWORK_DEFS>
```

Within the BAGS section, you are able to define multiple bags which can each have multiple fields. You must define the maxBags attribute which defines the maximum number of bags you expect a script to see created either locally or remotely. For example, a game that supports 8 players, each player creating 2 bags, must support a maximum of 16 bags even though there may only be a couple of different bags defined in the XML.

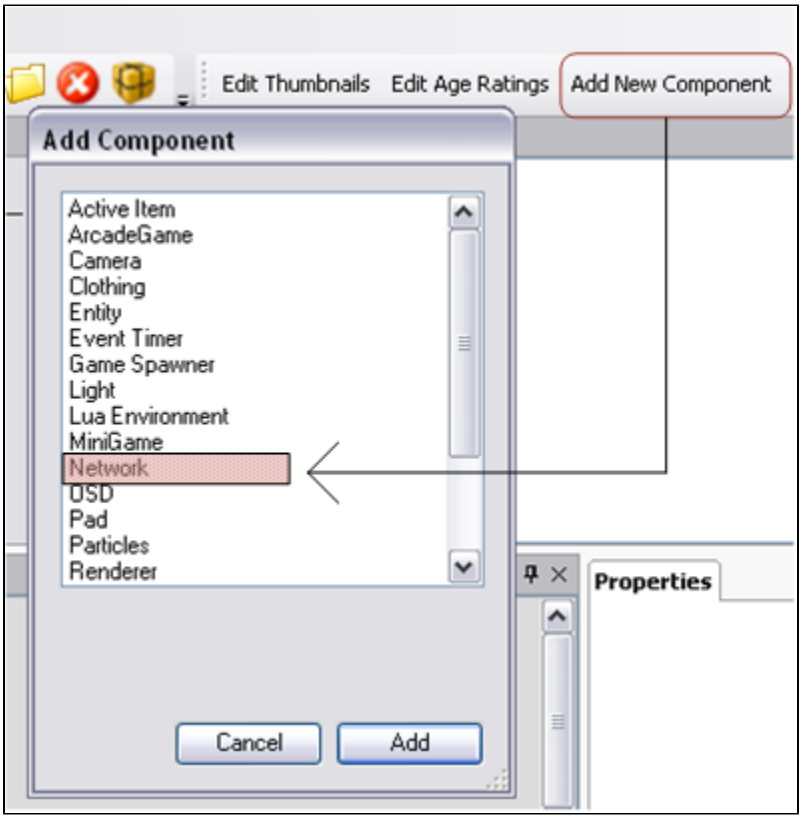
A field is made up of the following sections:

1. NAME: The name of the field that is used in the script.
2. TYPE: The data type of the field (int8, int16, int32, float32 and string).
3. ARRAYSIZE: For strings, you need to specify the maximum size they will take. The ARRAYSIZE is in bytes.
4. MAXFREQUENCY: How often field updates will be sent out in milliseconds. For example, the player might update their

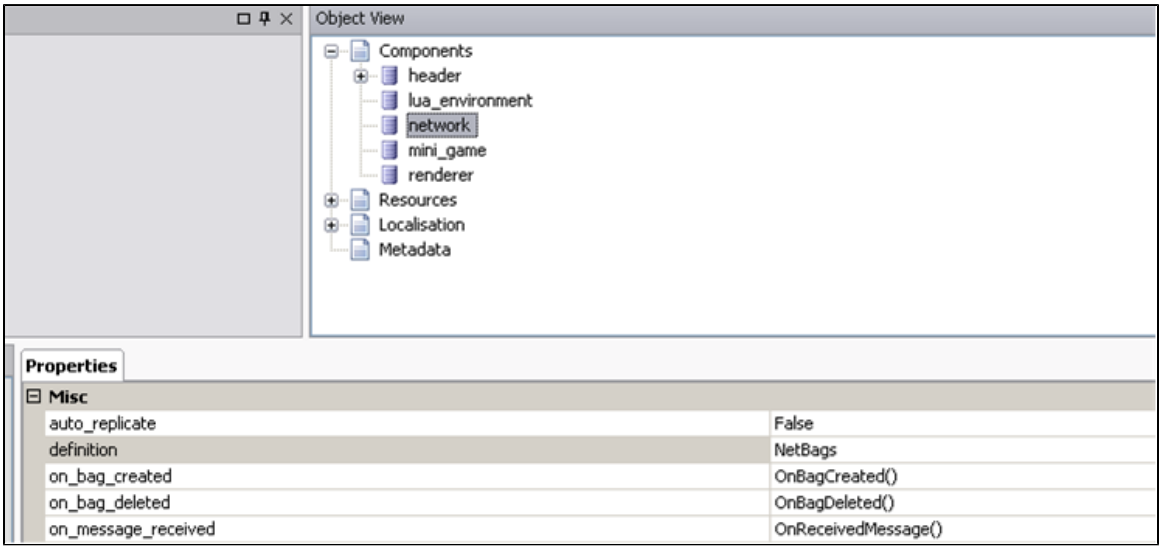
position in the bag every frame but we do not want that to be sent out every frame. If the update frequency is set to 500ms, the field will get sent out at most once every half second. If the frequency is set to 0, then the field will only be sent out when the bag is initially created. This is intended for data that could be read-only such as player names or account IDs.

Adding Netproperty Bag to an Object

1. Create a netproperty bag definition .xml file. Name it NetBags.xml.
2. Open the Object Editor, click Add New Local Resource and add the NetBags.xml.
3. Click Add New Component and in the Add Component dialog, select Network:



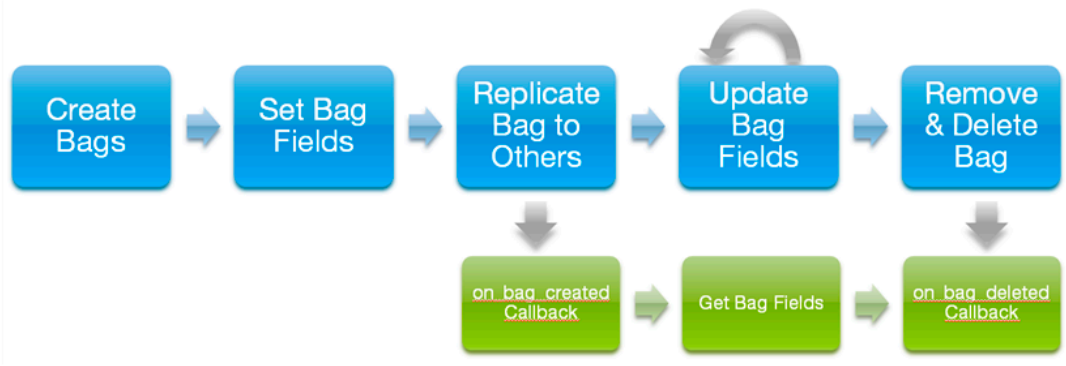
4. In the Object View panel, select the Network component to view its properties in the Properties panel. Change the definition field to NetBags:



5. Use the netproperty bag in the object's Lua script. See Scripting and Replicating Netproperty Bags below.

Scripting and Replicating Netproperty Bags

While bags are defined by XML, the actual replication must be performed by Lua script. The replication process follows the diagram below:



Create a netproperty bag using one of the definitions in the Netproperty Bag XML. The user creating the netproperty bag is also the owner. The owner cannot be changed.

To create a new bag in script:

1. Call `NetPropertyBag.Create()`, using the name of the desired definition.
2. Call `NetPropertyBag.SetField()` to specify the values of each <field> in the bag.
3. Call `NetPropertyBag.ReplicateToNet()` to replicate the bag to other users in the scene.
4. Update the bag's fields when necessary by calling `NetPropertyBag.SetField()`.
5. Once the bag is no longer needed remove it by calling `NetPropertyBag.DeleteFromNet()`.

For example:

```
-- This function creates a Netproperty Bag and replicates it to other users.
function CreatePlayerBag()

    -- 1. Create the bag.
    g_localPlayerBag = NetPropertyBag.Create( "player" )

    -- 2. Set the values of each field.
    NetPropertyBag.SetField( g_localPlayerBag, "name", LocalPlayer.GetPerson() : GetName() )
    NetPropertyBag.SetField( g_localPlayerBag, "score", 0 )
    NetPropertyBag.SetField( g_localPlayerBag, "weaponAmmo", {100, 50, 8, 4 } )

    -- 3. Replicate the bag to other users.
    NetPropertyBag.ReplicateToNet( g_localPlayerBag )

end

-- 4. This function updates the "score" field of a Netproperty bag.
function AwardPoints( extraPoints )

    local score = NetPropertyBag.GetField( g_localPlayerBag, "score" )
    score = score + extraPoints
    NetPropertyBag.SetField( g_localPlayerBag, "score", score )

end

--5. Delete Netproperty Bag when player leaves mini-games and null all references to the bag.
function OnLocalPlayerLeave()

    if NetPropertyBag.IsOwnedLocally( g_localPlayerBag) then

        NetPropertyBag.DeleteFromNet(g_localPlayerBag)

        g_localPlayerBag = nil

    end

end
```

Deleting the Netproperty Bag and Nulling References

In the example above, the netproperty bag is deleted and all of its references are set to nil in the OnLocalPlayerLeave()

function.

The netproperty bag is automatically deleted by the client when a user leaves the scene, but as a best practice, delete the bag when a user quits a Mini-game.

Also set any reference to that player's networkproperty bag to nil so that the client does not attempt any unnecessary calculations or tries to unnecessary messages.

Network Component Callbacks for the Netproperty Bag

There are two callbacks in the Network component for netproperty bags:

- on_bag_created()
- on_bag_deleted()

When either callback is called, identify the bag by calling NetPropertyBag.GetEventBag(), and identify the type by calling NetpropertyBag.GetTypeId() within the callback function.

For example:

```
function OnBagCreated()

    local eventBag = NetPropertyBag.GetEventBag()
    if NetPropertyBag.GetTypeId( eventBag ) =="player" then

        local player=

            {

                name = NetPropertyBag.GetField( g_localPlayerBag, "name"),
                score = NetPropertyBag.GetField( g_localPlayerBag, "score" ),
                ammo = NetPropertyBag.GetField( g_localPlayerBag, "weaponAmmo"),

            }
            OnPlayerEntering( player )

    end

end
function OnBagDeleted()

    local eventBag = NetPropertyBag.GetEventBag()
    if NetPropertyBag.GetTypeId( eventBag ) == "player" then

        local playerName= NetPropertyBag.GetField( g_localPlayerBag, "name" )
        OnPlayerLeaving( playerName )

    end

end
```

DME Limits

For Japanese Live Page
最新の日本語版ページはこちら 「DME制限」

Distributed Memory Engine Limits

These network limits ensure optimal network use and optimal bandwidth use with regard to the overall design of any scripted system. These limits are required for a scene containing 45 active people (that is, people who are chatting normally and using the scene's available facilities and running the scene's scripts).

Distributed Memory Engine (DME) refers to the PS Home network traffic, not to the HTTP or PSN network traffic.

DME Send bandwidth for a user playing a mini-game:

- maximum: 2 KB/sec
- recommended: 1 KB/sec

DME Receive bandwidth for any single player at any one time

- no more than 4 KB/sec

Occasional spikes are unavoidable and are allowed as long as they are not too frequent and sustained.

To check the network values, use the console command `enableNetStats 1`or via the Profile GUI `ProfileGui.Enable 1`.
The maximum size of an outbound message sent using the OutboundMessage Lua library is 1KB.

Also see Information Transmission limits and policies in General Design Guidelines and Network Platform Restrictions.

Additional Guides on Scripting

The following Lua features have comprehensive documentation on their use:

- Avatar Animation
- Scripted Entity Animation
- Dynamic Lighting
- Network Platform (NP)
- On Screen Display (OSD)
- The DUALSHOCK®3 Controller
- Realtime Games
- Video Recorder

Intersection Testing

The Lua API provides a number of functions to perform intersection/overlap testing in both 2D and 3D for various geometric primitives. These functions are highly optimized and should outperform any similar general purpose routines written in Lua.

You may find these functions useful for collision detection, AI, visibility testing, camera control, or other gameplay requirements.

Basics

The library supports testing between all* combinations of the following primitives:

- 2D
    - Ray (half-line)
    - Line segment
    - Line
    - Circle
    - Axis-aligned bounding box (AABB)
- 3D
    - Ray (half-line)
    - Line segment
    - Line
    - Sphere

- AABB
- Plane

* In 3D the various line types (ray, segment, line) cannot be intersected with each other, with the exception of computing the closest distance of approach between two lines.

All intersection combinations will indicate whether or not an overlap has occurred. Generally, most intersection combinations will also optionally provide further detail on the intersection if there is one, including an intersection distance for line primitives, and penetration distance for circles, spheres, boxes and planes. Usually it is slightly more expensive to compute the additional intersection data, so if you do not require it, do not request it. Exceptions to this are noted in the Lua API Reference documentation. An intersection distance for a line primitive is given in terms of that primitive's direction vector, so for rays and lines it will be an absolute distance along the ray or line from the origin or line point, and for segments it will be between 0 and 1 and represent a point on the segment.

All primitives are specifed with core types (Vector4, number) rather than API-defined types like Sphere or Line, so that you are not forced to use API types and can retrofit these routines into your scripts if desired.

See the section below for definitions of the supported primitives and how they are defined when using the API. All API functions are of the form

```
<return values> Intersect.<Primitive><Primitive><Dimension>(Primitive1Params, Primitive2Params, ...)
```

For example:

```
Intersect.RayLine2d
Intersect.AabbAabb3d
Intersect.SegmentPlane3d
```

For more details, see the Lua API Reference documentation.

Primitive specifications

In all 2D functions, the Z and W components of Vector4 values are ignored or unspecified.

> ⚠ Most semantic requirements (unit-length vectors, normals and so on) are NOT verified by the Lua API, so you should be careful to ensure you comply with them. This is to avoid performing expensive checks on these requirements in the routines.

Ray

A ray (or half-line) is specified by an origin point and a unit-length direction vector. The ray is considered to start from the origin and continue infinitely along the specified direction. Mathematically it may be defined as:

r = o + t*d

where r is the ray, o is the origin point, d the direction vector, and t a scalar parameter in the range 0 to +infinity.

Intersect API

2D

- number ox, number oy OR Vector4 origin - X and Y coordinates of the origin point.
- number dx, number dy OR Vector4 dir - a unit-length (normalized) 2D direction vector (specified as X and Y components).

3D

- Vector4 origin - X, Y, Z coordinates of the origin point.
- Vector4 dir - a unit-length (normalized) 3D direction vector

Segment

A line segment is conceptually the finite line between a start point and an end point. In the Lua API it is specified by the start point and the vector from the start point to the end point - therefore, if the line segment is from p1 to p2, then the start point is p1 and the segment vector is (p2 - p1). This vector will be called the segment vector in the Lua API. Mathematically it may be defined as:

s = p + t*v

where s is the segment, p is the start point, v the segment vector, and t a scalar parameter in the range 0 to 1.

⚠ Note that the segment vector need not be normalized, the length of the vector defines the length of the segment.

Intersect API

2D

- number ssx, number ssy OR Vector4 segStart - X and Y coordinates of the start point.
- number svx, number svy OR Vector4 segVec - the segment vector (see description above).

3D

- Vector4 segStart - X, Y, Z coordinates of the start point.
- Vector4 segVec - the segment vector (see description above).

Line

A line is specified by a point on the line and a unit-length direction vector. The line is considered to extend infinitely away from the point in both directions. Mathematically it may be defined as:

l = p + t*d

where l is the line, p is a point on the line, d the direction vector, and t a scalar parameter in the range -infinity to +infinity. Note that the same line can be specified in an infinite number of ways depending on the point p that is chosen, and the direction vector can be negated and still represent the same line, although these will change the intersection distance values returned where appropriate.

Intersect API

2D

- number px, number py OR Vector4 point - X and Y coordinates of a point on the line.
- number dx, number dy OR Vector4 dir - a unit-length (normalized) 2D direction vector (specified as X and Y components).

3D

- Vector4 point - X, Y, Z coordinates of a point on the line.
- Vector4 dir - a unit-length (normalized) 3D direction vector.

Circle/Sphere

A circle (2D) or sphere (3D) is specified by a center point and a radius, which must be positive. Mathematically it may be defined as the set of points that are exactly "radius" distant from the center point.

Intersect API

2D

- number cx, number cy OR Vector4 center - X and Y coordinates of the circle center point.
- number r - the radius of the circle.

3D

- Vector4 center - X, Y, Z coordinates of the sphere center point.
- number r - the radius of the circle.

AABB

An axis-aligned bounding box (AABB) is defined by the minimum and maximum extents on each canonical axis in both 2D and 3D. The X, Y (and in 3D, Z) components of the minimum and maximum extent vectors specify the minimum and maximum extent of the box on the respective axis. An equivalent specification that is also used is to specify the box using the center point and the half-extents on each access. The Intersect library does not use this formulation, but you can swap between them with the following formulae:

Min and max extents => Center and half extents

center = (min + max) * 0.5
half extents = (max - min) * 0.5 OR max - center

Center and half extents => Min and Max extents

min = center - half extents
max = center + half extents

Intersect API

2D

- number minx, number miny OR Vector4 min - minimum extents on the X and Y axes.
- number maxx, number maxy OR Vector4 max - maximum extents on the X and Y axes.

3D

- Vector4 min - minimum extents on the X, Y, Z axes.
- Vector4 max - maximum extents on the X, Y, Z axes.

Plane

A plane in this context is a flat, 2D surface embedded in 3D space, which extends infinitely and thereby divides 3D space in half. It is defined by its normal, and the distance of the plane from the origin, and all points on the plane satisfy the following equation:

$$n \cdot p - d = 0$$

where n is the plane normal, . represents the dot product operator, p is a point and d is the distance of the plane to the origin. The side of the plane which the normal points "out of" is called the positive half-space, and the other side is called the negative half-space. You can specify whether you wish a plane to be considered as one- or two-sided when performing an intersection test. If it is one-sided, only impacts that hit the "front" of the plane (i.e. coming from the positive half-space) are considered, if two-sided, all impacts are considered.

Intersect API

3D

- Vector4 plane - the X, Y, Z components represent the normalized (unit-length) plane normal. The W component represents the distance from the origin (d in the equation above).

HDK API Scripting

The HDK API is a custom PS Home API based on IronPython. It enables you to customize certain tools within the HDK. The HDK API is available for the Object Editor, Scene Editor and Repertoire Editor. You can use the API to:

- Enter console commands through the Object Editor, Scene Editor and Repertoire Editor.
- Create custom UI elements in the Object Editor, Scene Editor and Repertoire Editor.
- Run scripts through the Object Editor, Scene Editor and Repertoire Editor to run processes like batch updating of content or running pipelines.

You can use scripts to simplify processes, create buttons and improve your development.

This section covers how to:

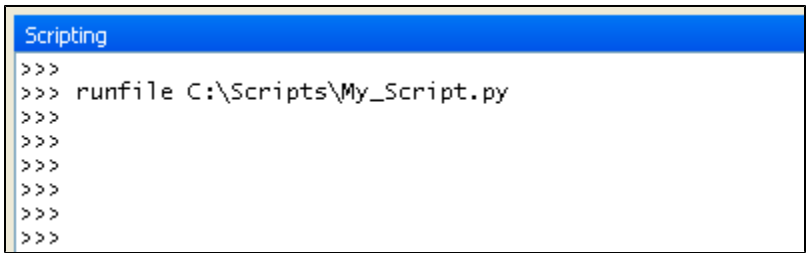
- Use scripts and load your IronPython scripts into the Scene Editor.
- Create custom UI elements.
- Enter commands into the Object Editor, Scene Editor and Repertoire Editor console and create menu items.

See the HDK API Reference for the list of available functions.

Running Scripts

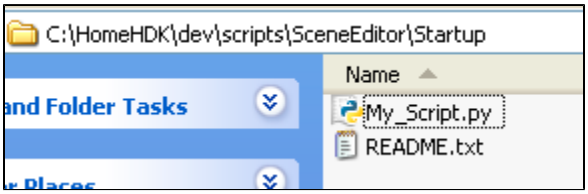
You can run a script using the Scene Editor/Object Editor's Scripting panel or using the Scene Editor/Object Editor's Startup folder.

To run a specific script using the Scripting panel, enter `runfile <FULL_FILE_PATH>`. For example, `runfile C:\Scripts\My_Script.py`



To run a script using the Scene Editor's Startup folder, place the script in the `<HDK_INSTALL_DIR>/dev/scripts/SceneEditor/Startup` folder and it will automatically run when the Scene Editor starts up.

For the Object Editor, the path for startup scripts is <HDK_INSTALL_DIR>/dev/scripts/ObjectEditor/Startup.



Help Commands in the Scripting Panel

In the Scripting panel, you can also retrieve information on available functions by entering the following commands:

Command	Description
help	Retrieves a list of all available classes.
help <class>	Where <class> is the name of the class - retrieves a list of all functions in a class.
help <function>	Where <function> is the name of a function in a class - retrieves the details on a function.

Importing Python Scripts

To use one script within another, you use a mechanism in Python called 'modules'. For more information on modules, see http://docs.python.org/tutorial/modules.html.

Using one script within another

Enter import <my_script> at the beginning of your script, where <my_script> is the name of the script that you want to import.

For example:

```
import my_script
```

This command looks for a script named my_script.py, and then imports it as a module.

Importing Several Scripts

To import several scripts:

1. Place all the scripts that you want to import into one folder.
2. Create an empty script and save it as _init_.py.
3. Place _init_.py in the folder created in step 1.

The folder is now set up.

- To load all the scripts from that folder, enter from <folder_name> import *.

  Where <folder_name> is the name of the folder created in step 1.

- To load only one script from that folder, enter from <folder_name> import <file_name>.

  Where <folder_name> is the name of the folder created in step 1, and <file_name> is the name of the Python script.

Using Functions and Classes from Imported Scripts

When you have imported a script, you can use its classes and functions within it by calling <imported script name>.<className()>

For example, a script named my_script.py contains the class ClassName(). To create an instance of the class from the imported script:

```
myClass = my_script.ClassName()
```

Script Paths

When you import scripts Python searches a number of locations on your computer automatically. You can hasten the search and guarantee that Python will find an included script by telling it where to look.

To tell Python where to look enter the following in the main script before calling `import <my_script>`:

```
import sys

sys.path.append(<script_path>)
```

Where `<script_path>` is the folder location of the scripts you want to import.

> Minimize the number of paths that you add in this way. Where possible, it is better to import scripts relative from one common path, to avoid potential issues with performance and script name conflicts.

Custom UI Elements

Creating New Menus

To create a new menu, enter the following in a script:

```
exampleMenu = MenuManager.CreateMenu("Example")
```

This creates a new menu called 'Example'. If a menu already exists with this name then `CreateMenu` will throw an exception.

> Menus will not appear in the Scene Editor/Object Editor until they contain a menu item (see Creating Menu Items).

Accessing Existing Menus

If you want to access a menu which you have already created enter the following in a script:

```
exampleMenu = MenuManager.GetMenu("Example")
```

This tries to get a previously created menu called 'Example'. If the menu does not exist then `GetMenu` returns `None`.

Multiple Scripts

If you have multiple scripts creating and adding to one menu we advise you check that the menu already exists before trying to recreate it. The code below shows how to check if a menu already exists:

```
exampleMenu = MenuManager.GetMenu("Example")

if exampleMenu == None:

    exampleMenu = MenuManager.CreateMenu("Example")
```

> Menus will not appear in the Scene Editor/Object Editor until they contain a menu item (see Creating Menu Items).

Creating Menu Items

To add a menu item to a menu you must first create the menu item itself:

```
exampleItem = MenuManager.CreateMenuItem("Example Item")
```

This creates a new menu item called 'Example Item'.

To add this menu item to an existing menu:

```
exampleMenu.AddMenuItem(exampleItem)
```

This adds the menu item to to the 'Example' menu.

The sections below describe how to customize menu items and attach functions that are called when the menu item is pressed. These customization steps must be performed before the menu item is added to a menu through AddMenuItem.

Menu Item Properties

Custom properties can be set on the menu item object returned by CreateMenuItem.

These properties include:

- IconPath: The path to an image which the menu item should display.
- Description: A description of the menu item which is displayed in its tools tip.
- Shortcut: A keyboard shortcut which when pressed will execute the menu item.
- Toggable: Whether the menu item should have an on and off state.
- HasToolbarButton: Whether the menu item should have a button on the Scene Editor toolbar.

Attaching Functions to Menu Items

By default, when clicked a menu item will do nothing. To make a menu item actually perform a function you must use the Execute event of the menu item object returned by CreateMenuItem:

```
exampleItem.Execute += self.ExampleExecute
```

This causes a function, ExampleExecute(), to be called when ever the menu item 'exampleItem' is selected.

Any function set to be called by the menu item's Execute must have the sender and args parameters in their signature as shown below:

```
def ExampleExecute(self, sender, args)
```

Enabling/Disabling Menu Items

By default all menu items are enabled and can be clicked at all times. To change a menu item to be disabled you must use the CanExecute event:

```
exampleItem.CanExecute += self.ExampleCanExecute
```

The CanExecute function will return either True or False to set the current enabled or disabled state, respectively, of the menu item.

```
def ExampleCanExecute(self):

    return False
```

In the example above the menu item will always be disabled because the CanExecute function returns False.

This function will be called throughout the running of the Scene Editor, so you can dynamically update the enabled state of the menu item.



Setting a menu item to be disabled will also disable any button and keyboard shortcuts which you have set.

Examples

The Live Editing buttons (Manage Targets, Lock Camera, and Auto Zoom Camera) were created using the HDK API. Custom Menu Example shows and describes the manage_targets.py script that creates the Manage Targets menu and button.

These scripts can also be found here: <HDK_INSTALL_DIR>\dev\scripts\Internal\SceneEditor\Startup.

Custom Menu Example

The example below shows the manage_targets.py script. This script creates the Live Editing button Manage Targets. The original script and other Live Editing scripts are located at <HDK_INSTALL_DIR>\dev\scripts\Internal\SceneEditor\Startup.

```
import clr

clr.AddReference("System.Windows.Forms")

clr.AddReference("System.Drawing")

from System.Windows.Forms import Keys

from System.Drawing import Bitmap

from HdkApi.Core.Communication import TargetManager

from HdkApi.Core.Utilities import PathUtilities

from HdkApi.SceneEditor.UserInterface import MenuManager, Menu, MenuItem

class ManageTargets():


    def __init__(self):

        # Retrieves the menu "Live Editing". If the menu does not exist, it will return None.

        liveEditMenu = MenuManager.GetMenu("Live Editing")



        # If "Live Editing" Menu did not exist, creates it.

        if liveEditMenu == None:


            liveEditMenu = MenuManager.CreateMenu("Live Editing")


        # Creates the menu 'Manage Targets,' and gives it a description, an icon, places it in the
toolbar, and gives it a function (ManageTargetsExecute) to call when the icon is pressed.

        manageTargetsItem = MenuManager.CreateMenuItem("Manage Targets")

        manageTargetsItem.Description = "Select which PS3 the editor is connected to"

        manageTargetsItem.IconPath = PathUtilities.HdkInstallRoot +
"/dev/Scripts/Internal/SceneEditor/Icons/ManageConnections.png"

        manageTargetsItem.HasToolbarButton = True

        manageTargetsItem.Execute += self.ManageTargetsExecute



        # Adds the manageTargetsItem to the "Live Editing" menu.

        liveEditMenu.AddMenuItem(manageTargetsItem)



        TargetManager.Initialize()
```

```
    # Defines the function ManageTargetsExecute, which is called whenever the Manage Targets icon is
pressed.

    def ManageTargetsExecute(self, sender, args):


        TargetManager.DisplayTargetsForm(None)
```

```
ManageTargets()

print("Manage Targets loaded")
```

HDK API Guidelines

Loading and Importing Scripts

Minimize the number of paths that you add in the beginning of a script that tells Python in which folders to look for script files. Instead, where possible, import scripts relative from one common path. This prevents potential issues with performance and script name conflicts.

Menu Items

Setting a menu item to be disabled also disables any button and keyboard shortcuts which you have set.

HDK API Executable

HdkApi.exe is an executable file that allows many HDK API features to be executed from the Windows® Command Prompt. This allows custom pipelines/batch processing to be performed without the need for the main HDK tools to be running. The executable is located in <HDK Installation Directory>¥dev¥bin on 32-bit Windows® editions and <HDK Installation Directory>¥dev¥bin64 on 64-bit Windows® editions.



Some functionality, such as UI-based APIs, are restricted to the runtimes of the tools that they can be found in. These features are therefore not available via HdkApi.exe.

HdkApi.exe includes features such as context sensitive help, run file and a fully functioning IronPython rep loop to give the tool a look and feel consistent with other HDK tools. You may wish to perform simple tasks from the console in the rep loop (runfile is still an option) or pass scripts and arguments to the HdkApi.exe and allow it to run once (particularly useful in pipeline).

Console Arguments

Arg	Description	Example
-s	Script to run	-s "d:¥blah.py" or -s "d:¥blah.py,d:¥blah1.py"
-h	Help	Display arg help
-v	Verbose	Show output

To allow more flexibility, any option provided that is not already restricted will be set as an environment variable and will be accessible in your scripts, highlighted below:

HdkApi.exe -myArg "myValue"

```
import os

def main():
    if "myArg" in os.environ:
        print(os.environ.get("myArg"))

main()
```

Reploop keywords

Keyword	Description
-help	Print overview of HdkApi
-help *classname*	Print help for specific class

-help *methodname*	Print help for method of class context
-exit	Exit reploop
-runfile	Run file from harddrive

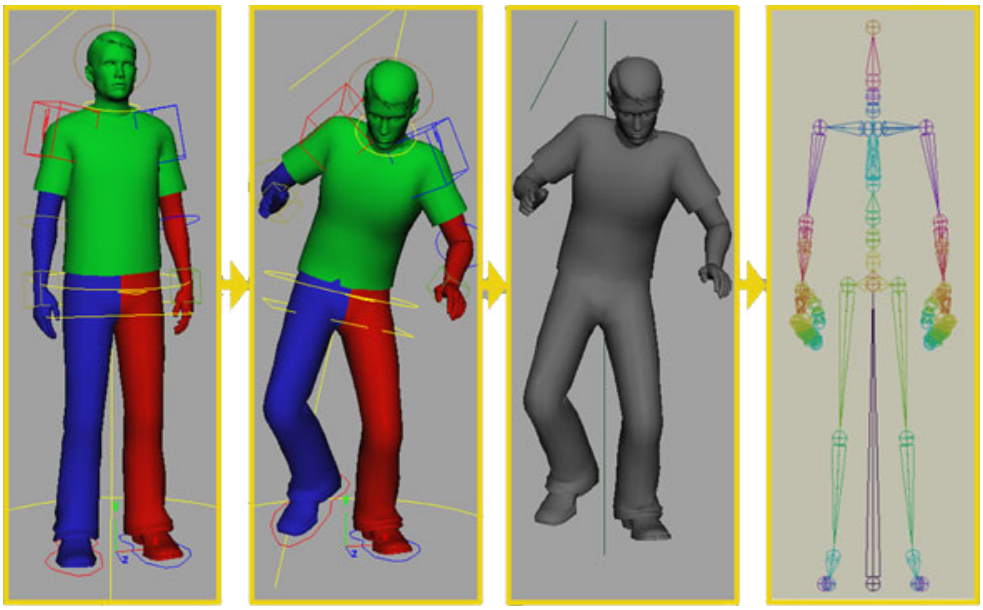
Animation and Effects

Animation

There are two basic types of animation In PS Home:

- Avatar Animation: By using the Repertoire Editor, you can add custom avatar animations exported from the HDK Art tools to FullBodySuits and mini-games.
- Other Animation and Animation Effects: Animated models and animated shaders allow you to make all kinds of things really come alive, from moving models in a space or game to textures on clothing and furniture.

The following image shows the basic animation creation workflow:



From HDK 1.3 onwards, the Maya tool chain provides a pipeline to enable developers to export custom avatar animation. You can use these animation repertoires only within realtime games, mini-games and FullBodySuits.

See:

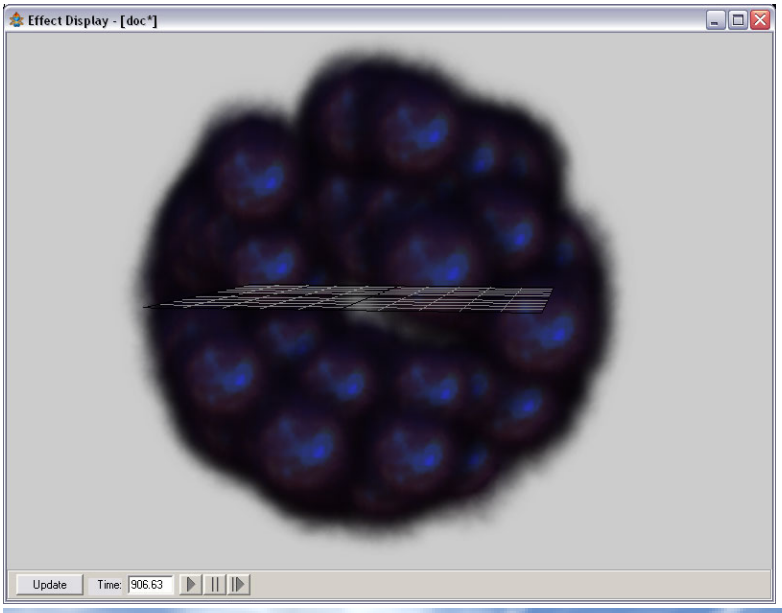
- Avatar Animation Puppet Rig and Avatar Animation Puppet Poser to learn about creating the DCC tools that allow you to create .ani animation files.
- Repertoire Emotes to learn how to create custom emotes in the Repertoire Editor using .ani files.

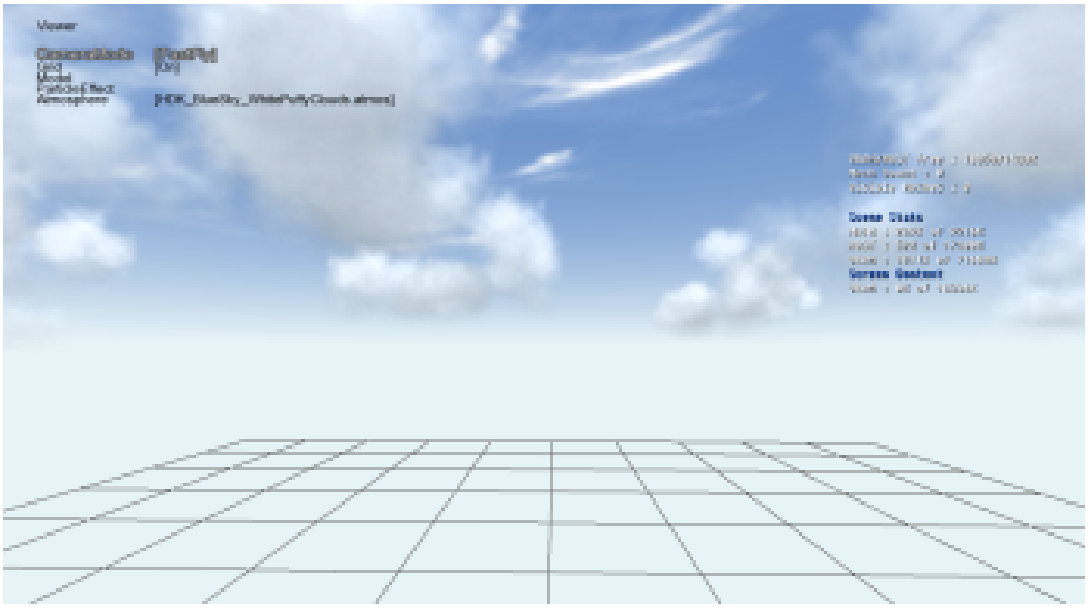
Animating Entities with Lua Script

You can script animating behavior for an entity (see Scripted Entity Animation), animate geometry in Maya, then export it as part of the scene, or as entities that scripts can dynamically load and control.

Cloth support is not included in this section.

Effects





Particles

Use the ATG Particle Effects Tool to create particle effects for scenes or scripted objects (except furniture and clothing). This tool lets you to create effects by placing a number of 'nodes' into an 'effect graph' and tying together attributes on each node, to represent the data flowing between the nodes.

Sky Design

In addition to the other animation options, you can also control the way the sky looks and changes using the Home Sky Tool. This is more of an effect or simulation than an animation, and it allows you to make a more dynamic sky or even simulate day/night cycles. See Sky Design.

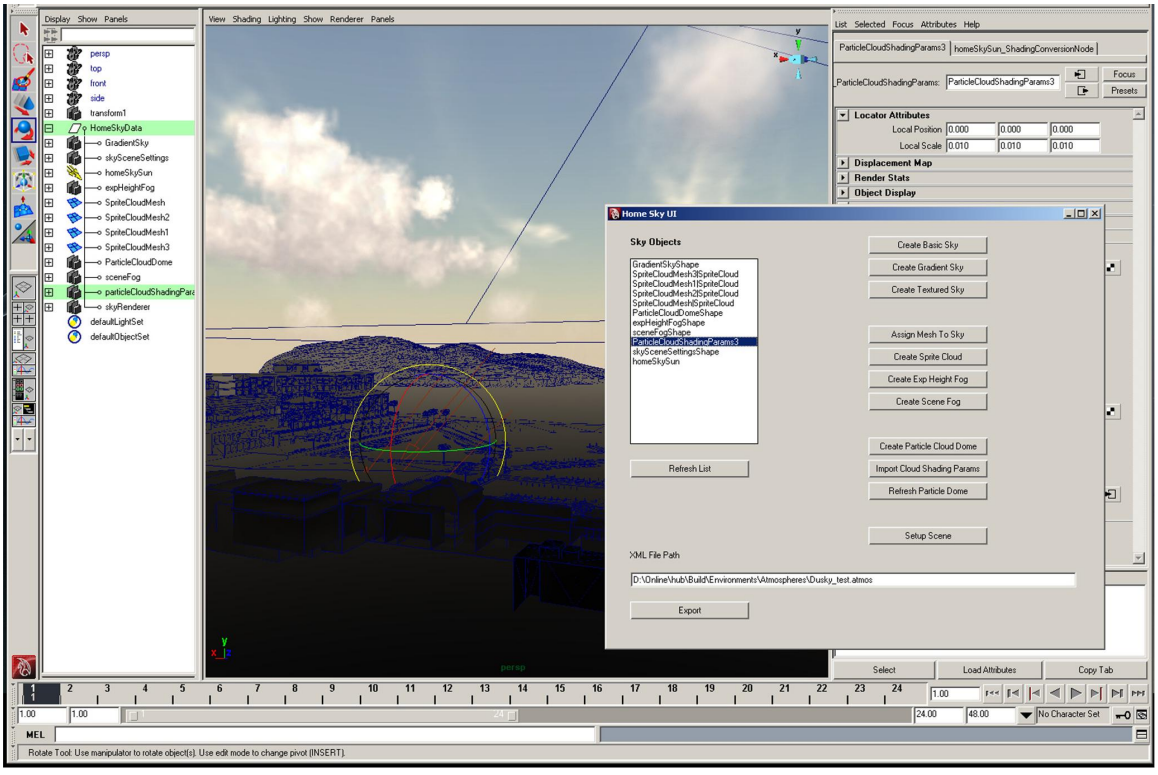
Sky Design

The Home Sky Tool provides a way to create a procedural animated sky background texture for your Home scene. This background texture is rendered behind all other geometry and cannot interact with scene geometry at all. The tool also allows for defining fog within Home scenes.



The Home Sky Tool is a plug-in for Maya. Sky (or atmosphere) scenes created in Maya are not related to your main Home scene geometry in any way. Sky Maya scenes are their own entity and export their own data directly to the <HDK_INSTALL_DIR>.

> For a short step-by-step process of creating a simple sky for your scenes, see the Sky Design Quick Start Tutorial.



Opening the Home Sky Tool

In Maya, the Home Sky Tool is accessed via two buttons on the Home_Env tool shelf. The left button launches the Home Sky UI. The right launches the Cloud Modeler UI.



Sky Files

All working Maya files should be in the <HDK_INSTALL_DIR>\artsource folder, and all referenced .dds textures and final exported data should live in the <HDK_INSTALL_DIR>\build folder.

Sky scenes export to a single XML file with extension .atmos. This file is linked to your main Home scene via a resource connection in the Scene Editor.

Console Commands

There are several commands that you can use while profiling your sky scene in the Home Developer Self. For more information, see Debug Console Command Reference.

Sky Design Workflow

This section outlines the Sky Tool modes.

Cloud Modeling

Before you can build a sky for a scene, you need some clouds to populate the sky with. You do this with the Cloud Modeler UI.

The Cloud Modeler UI allows us to model a set of Cumulus-style fluffy clouds. These are known as "particle clouds". Groups of spheres are used to model the overall cloud shape, and particle sprites with special shading properties are used to flesh out the volume of the cloud. Once a set of clouds have been made, they are saved to an XML file in the <HDK_INSTALL_DIR>\artsource folder.

This pre-modeled set will be used later for populating skies in the Home Sky UI.

Cloud library XML files, and Cloud Modeling Maya scenes should live near each other in the <HDK_INSTALL_DIR>\artsource folder, but should not be bound to any particular scene. This allows the clouds to be used by many different scenes, for example:

```
artsource\Environments\Atmospheres\CloudLibs\CloudLib1.ma
artsource\Environments\Atmospheres\CloudLibs\CloudLib1.xml
```

Sky Modeling

This is used to model the actual sky for a specific Home scene. Sky scenes are started from clean new Maya scenes.

With the Home Sky UI open, clicking the Setup Scene button initializes the Maya scene, ready to accommodate sky elements. An AtgiLocator node is created, referencing the destination scene's intermediate .atgi file. The AtgiLocator shows a wireframe representation of the scene within Maya to aid with sky modeling.

Once set up, you can model the sky. You can create a particle cloud dome which will reference a cloud library .xml file as created by the Cloud Modeling UI.

Sky modeling Maya files are encouraged to live alongside the main Home scene Maya file. For example, for a scene called:

```
artSource\Environments\MyScene\MyScene.ma
```

There would also be a sky scene called:

```
artSource\Environments\MyScene\MyScene_Sky.ma
```

Other Requirements

For correct exporting, all textures referenced by the sky nodes must be .dds files and live in the <HDK_INSTALL_DIR>\build folder.

The Home Sky UI is used for final scene exporting. Sky scenes export to a single .xml file with extension .atmos. The sky scenes should export to the same location as the Home scene project (.scproj) file. If a particle cloud dome has been used, then another file with extension .cdata will also get exported into the same location, for example:

```
build\Environments\MyScene\MyScene.atmos
build\Environments\MyScene\MyScene.cdata
```

Once exported, the user can link the .atmos file to their scene using the Scene Editor.

The Maya viewport must be in wireframe mode when making use of the Home Sky Tool for the sky to render correctly.

Opaque HDR textures as used by Gradient Sky, Textured Sky and Sprite Mesh, should have 4 color channels. If HDR information is not needed, leave the alpha channel white.

Example Sky Scene Files

This section uses example files located within the <HDK_INSTALL_DIR> folder.

All of the resources in <HDK_INSTALL_DIR>\arsource\Environments\Atmospheres that this section cites are included with the HDK by default, and do not need to be downloaded separately.

The only resource cited in this document that needs to be downloaded separately is the HTS Aeronautilus. The HTS Aeronautilus sample can be downloaded from https://home.scedev.net/projects/samples.

The following sample files are available for download:

- Sample Maya scene files: artsource\Environments\Atmospheres\ExampleMayaScenes
- Sample Cloud Lib files: artsource\Environments\Atmospheres\CloudLibs

  This is the output from the example cloud modeling scene. You can reference these in initial sky scenes to get up and running quickly.
- Source Textures: artsource\Environments\Atmospheres

  These are .tga source images for the textures used in the Sky Tool.
- Final example .atmos exports: build\Environments\Atmospheres

  These are some final .atmos exports. They can be viewed using the HDK browser, or linked into your scene via the Scene Editor.
- Final Example Textures: build\Environments\Atmospheres\Textures

  These are example final .dds textures. Some are referenced by the example scenes.
- HTS Aeronautilus: This download includes all of the resources to load and export the scene in Maya and the Scene Editor. It also includes all of the asset files you need to load the HTS Aeronautilus' sky recources.

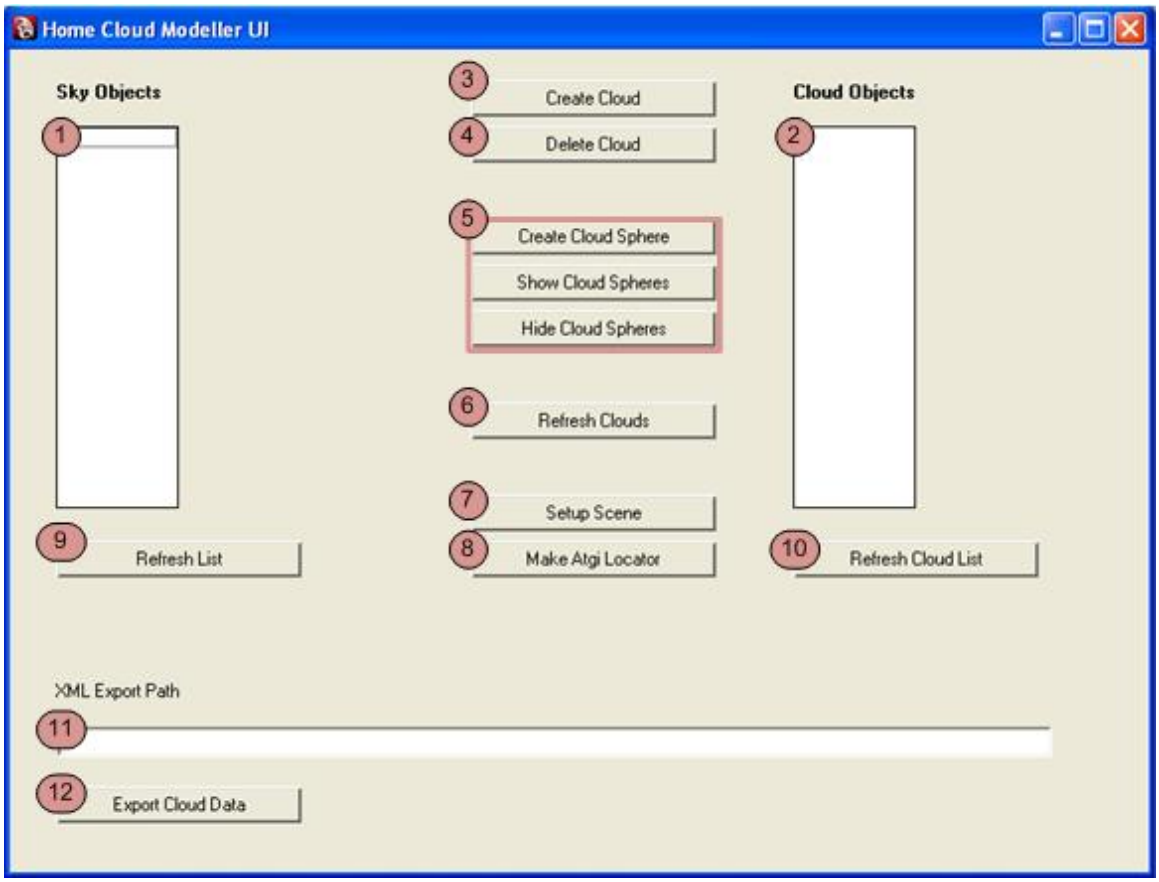
  <HDK_INSTALL_DIR>\artsource\Environments\HTS_Spaces\HTS_Airship\HTS_Airship_Sky.ma
  <HDK_INSTALL_DIR>\build\environments\hts_spaces\hts_airship\hts_Airship.atmos
  <HDK_INSTALL_DIR>\build\environments\hts_spaces\hts_airship\hts_Airship.mdl

Home Cloud Modeler UI

The Home Cloud Modeler is a mode of the Sky Tool which allows for modeling a set of Cumulous-style fluffy clouds. These clouds are known as Particle Clouds because they're made up of many small shaded particle sprites.

To launch the Home Cloud Modeller UI, click  on the Home_Env tool shelf.

The Home Cloud Modeller User Interface



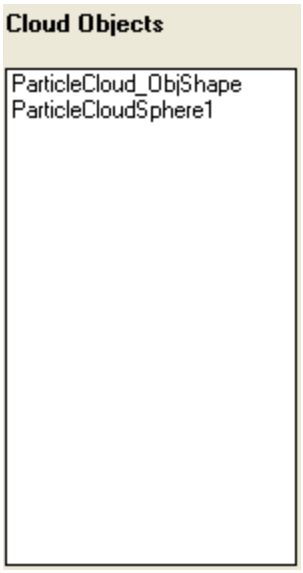
1. Sky Objects List



This is a list of the sky objects that you have created. The Sky Objects list is empty until you click Create Cloud, after which the Maya sky nodes appear. Selecting any item on this list will also select it in the Maya Outliner.

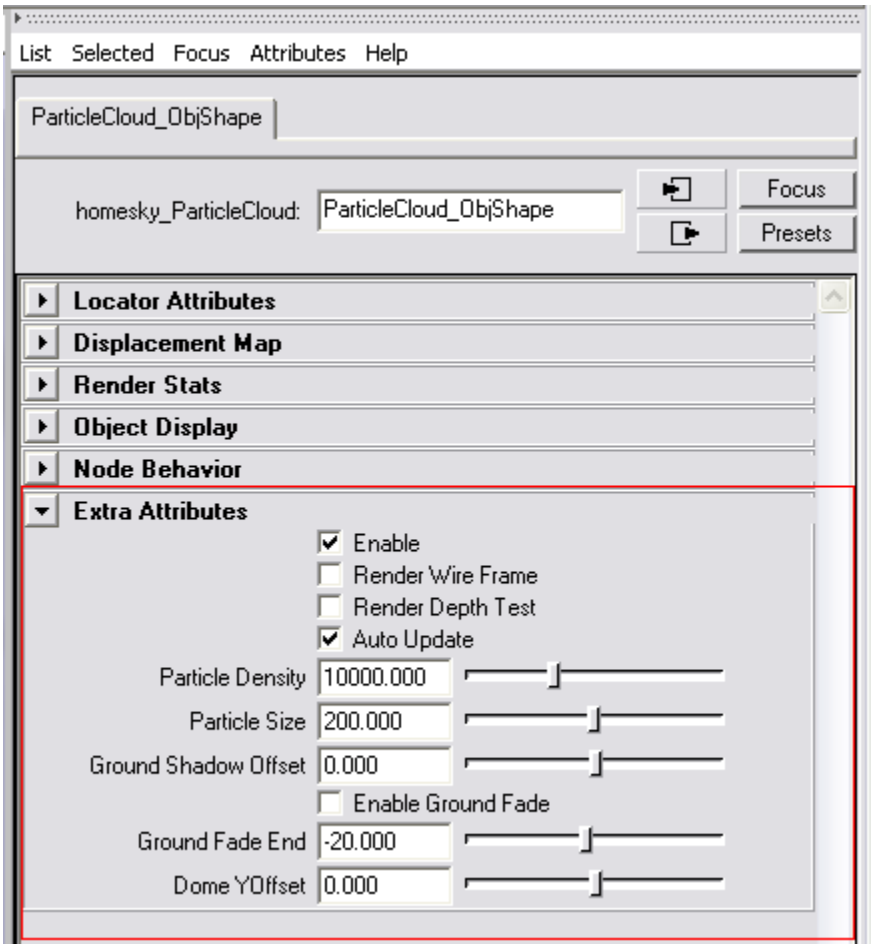
> Although the nodes are listed in the Maya Outliner, selecting from the Sky Objects list is easier to use as it also shows the appropriate node attributes up in the Attribute Editor.

2. Cloud Objects List



When a particle cloud is selected in the Sky Objects list, the components making up the cloud are shown in the Cloud Objects list.

The first item in the Cloud Objects list is the ParticleCloud locator itself, which contains all of the geometry settings (for example, density, particle size) for the cloud. You can see the geometry settings in Attribute Editor > Extra Attributes.



The remaining items in the Cloud Objects list are the spheres that make up the overall cloud shape.

3. Create Cloud



Clicking this button will create a single ParticleCloud node and a ParticleCloudShadingParams node in the Sky Objects box (and in the Maya Outliner).

ParticleCloud

The ParticleCloud node controls the population and scaling of the particle sprites for the cloud. Each ParticleCloud node has two default objects:

- Particle Cloud ObjShape: Use to tweak the particle density and particle size parameters.
- Particle Cloud Sphere1: Used to define the overall shape of the cloud. You can scale (stretch, rotate) and position the cloud with this node.

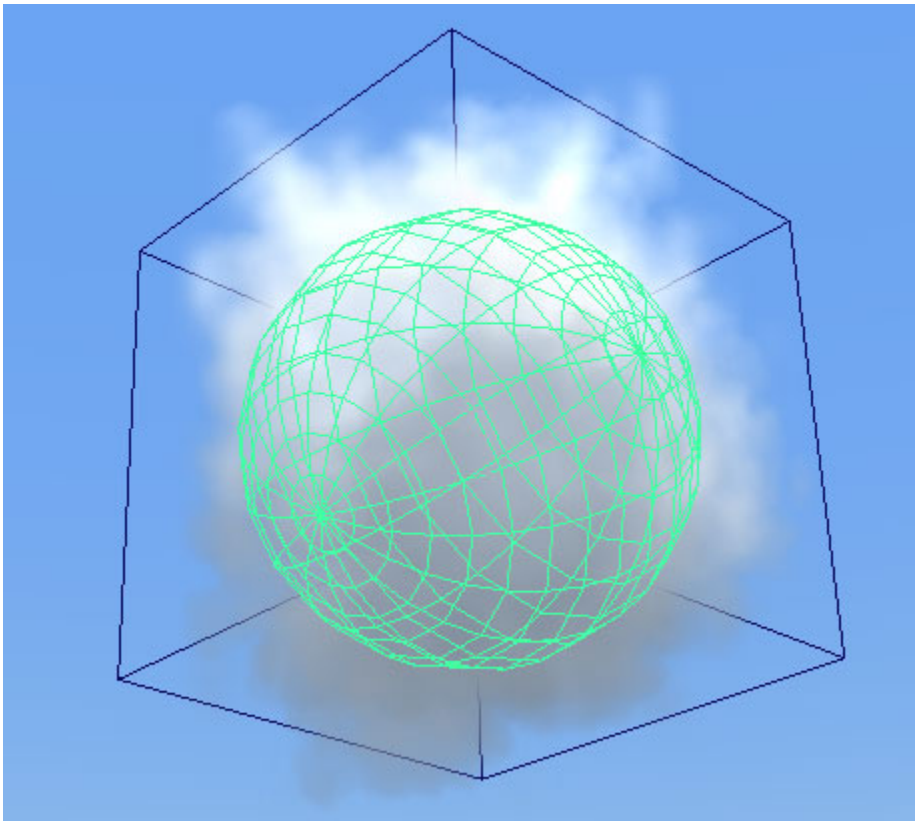
> The fewer sprites within a cloud, the faster the whole sky will render in PS Home. Remember to keep the number of sprites as small as possible to create an efficient final sky. It is often possible to simply increase the particle size rather than increase the number of particles when a cloud needs more volume. Between 100-200 is an approximate good number of particles per cloud.

particleCloudShadingParams

The shading of the clouds is defined by the particleCloudShadingParams node. Clouds do not make use of Maya materials at all. All shading parameters and texture paths are defined on this node. All clouds share the same single shading parameters node.

If the particle clouds do not display the correct texture, the link to the texture to be used in the tool will need to be assigned. The Sprite texture location should be:

<HDK_INSTALL_DIR>\build\environments\Atmospheres\CloudParticleTextures\cloud_particle.dds



4. Delete Cloud



Clicking this button will delete the currently selected cloud, along with its cloud spheres.

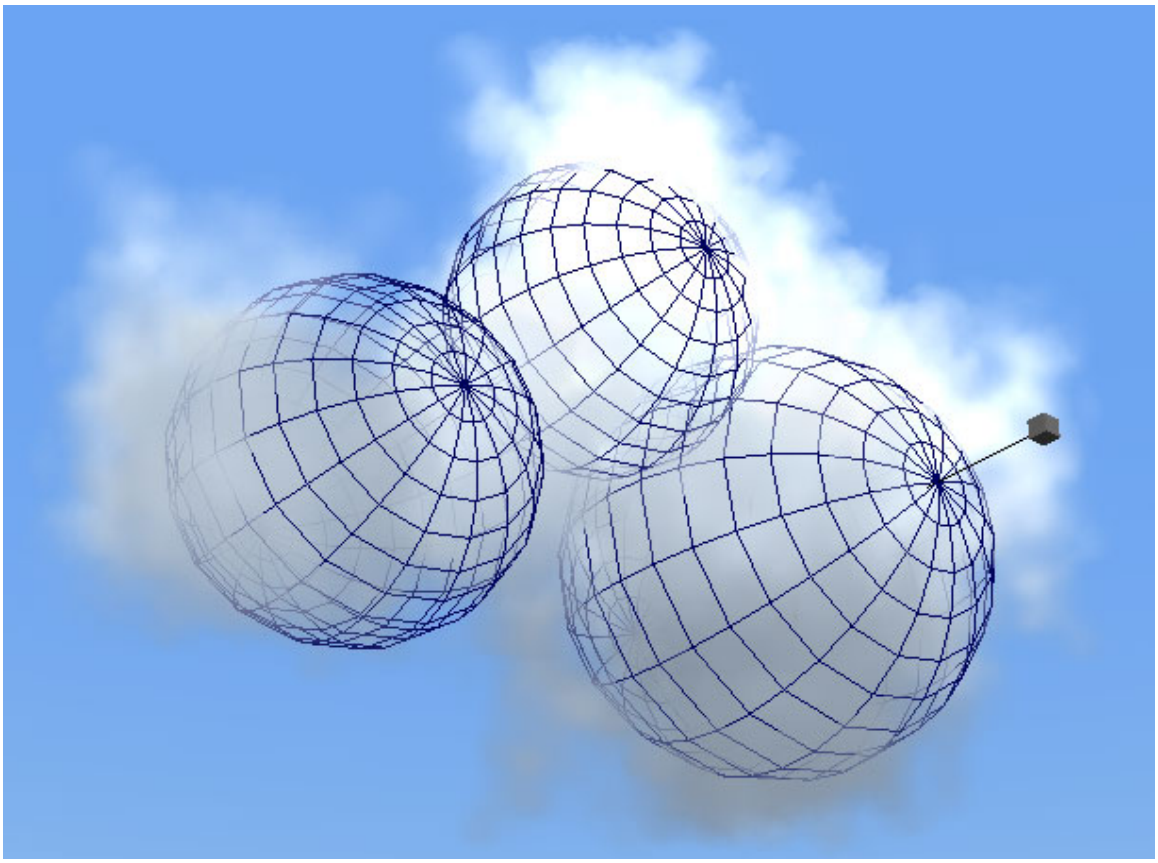
5. Cloud Sphere Buttons



These buttons allow for creation and visibility toggle of cloud spheres.

Create Cloud Sphere

This will create a new cloud sphere and parent it under the currently selected cloud.

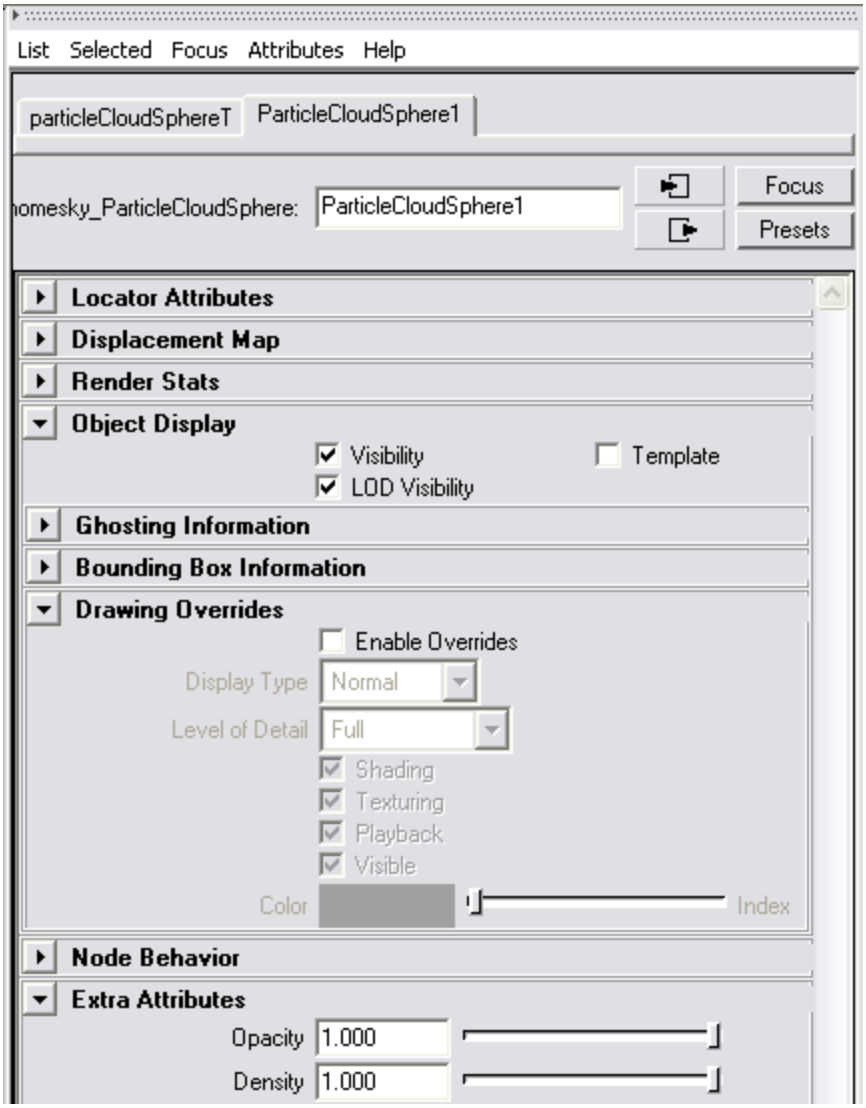


Show Cloud Spheres and Hide Cloud Spheres

Sometimes the cloud spheres can clutter the screen making the shading of the cloud hard to make out. These buttons allow for quick hiding and showing of the cloud spheres.



Cloud Spheres also contain some individual shading parameters such as opacity and density which will affect the particles contained within. You can adjust these settings by selecting the Cloud Sphere's node, then Attributes Editor > Extra Attributes. This allows for fine-grained tweaking of the cloud appearance.

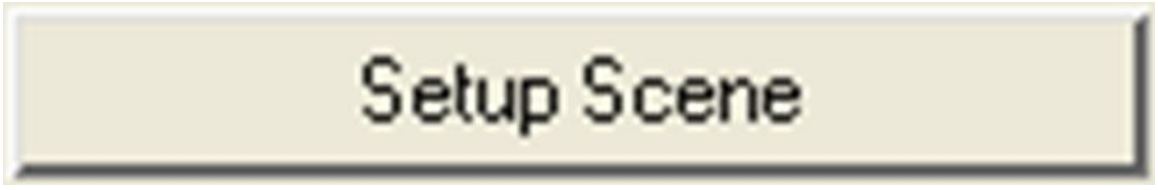


6. Refresh Clouds



Although automated as much as possible, sometimes the texture or shape of clouds rendered will need updating to reflect some parameter change. Clicking on this button will force them to update.

7. Setup Scene



This button will set up the initial empty Maya scene with groups and data necessary for a Home sky scene. Upon setting up the scene, you'll notice the Sky Objects list populate with Sun and particleCloudShadingParam nodes.

8. Make Atgi Locator



This button creates a transform node in the Outliner. The transform node allows you to import an .atgi file, which is a file created when you export a scene. The .atgi file will show a wireframe of the exported scene, so that you can model the sky around your scene.

9. Refresh List



Although automated as much as possible, sometimes the Sky Objects list will need updating to reflect changes. Clicking on this button will force them to update.

10. Refresh Cloud List



Although automated as much as possible, sometimes the Cloud Objects list will need updating to reflect changes. Clicking on this button will force them to update.

11. XML Export Path



All clouds in the scene export to single a user-specified cloud library .xml file.

The full path for the destination .atmos file needs to go into the XML file path text box.

12. Export Cloud Data

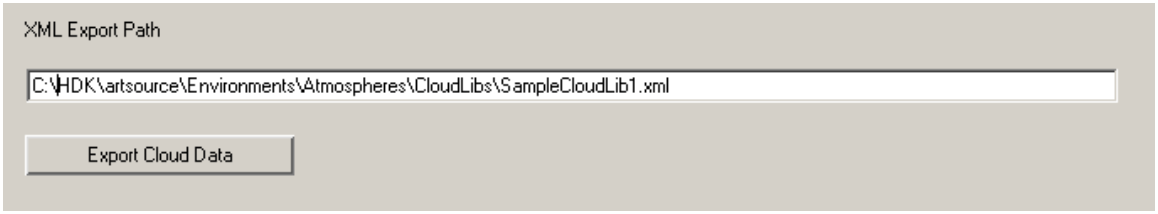


After you enter the file path and name into the XML Export Path, select this button to export the cloud data.

All clouds in the scene export to single a user-specified cloud library .xml file. Also exported to this .xml file is the particleCloudShadingParams node.

Any location can be specified for export, but it is encouraged that some directory within Artsource not related to a specific scene is to be specified. This will allow the clouds to be used in multiple scenes.

For example, C:\HomeHDK\artsource\Environments\Atmospheres\CloudLibs



Check the console output after pressing the Export Cloud Data button to see the status of the export.

Sky Tool UI

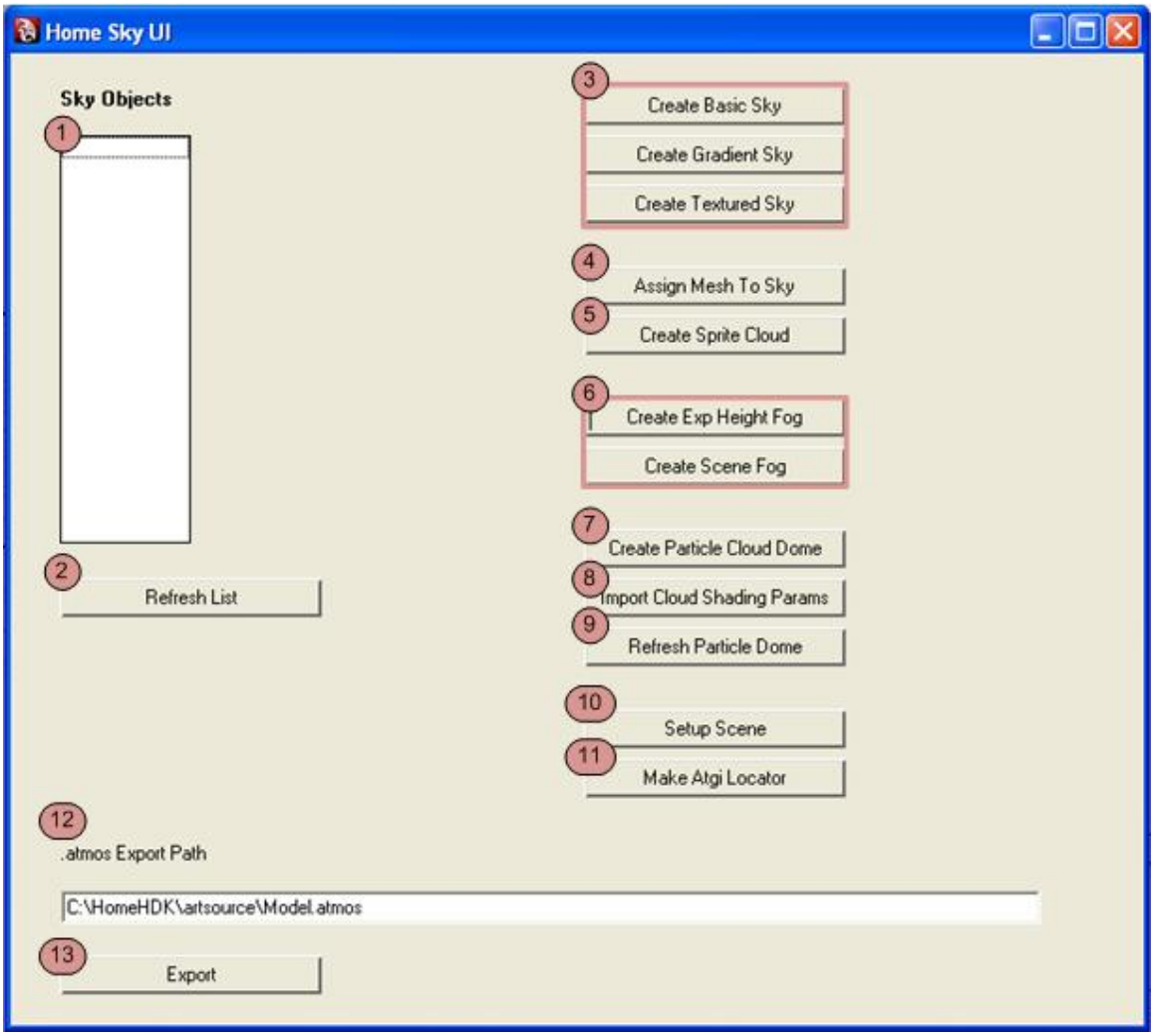
The Sky UI is the main sky building mode in the Sky Tool.

With this UI you can:

- Create and manage the Home Maya sky scene.
- Set the background color for the sky.
- Populate the sky with clouds and textures.
- Create fog.
- Export the sky scene to the PlayStation®3.

To launch the Home Sky UI, click  on the Home_Env tool shelf.

The Home Sky UI



1. Sky Objects List

This is a list of the sky objects that you have created.



In the Home Sky UI the Sky Objects list is empty until you create an element of the sky (for example, Basic, Gradient, Textured Sky). When you create an element of the sky, you also create a node in the Maya Outliner.

> Although the nodes are in the Maya Outliner, selecting from the Sky Objects list is easier to use as it also shows the appropriate node attributes in the Attribute Editor.

2. Refresh List



Although automated as much as possible, sometimes the Sky Objects list will need updating to reflect changes. Clicking on this button will force them to update.

3. Create Sky Buttons

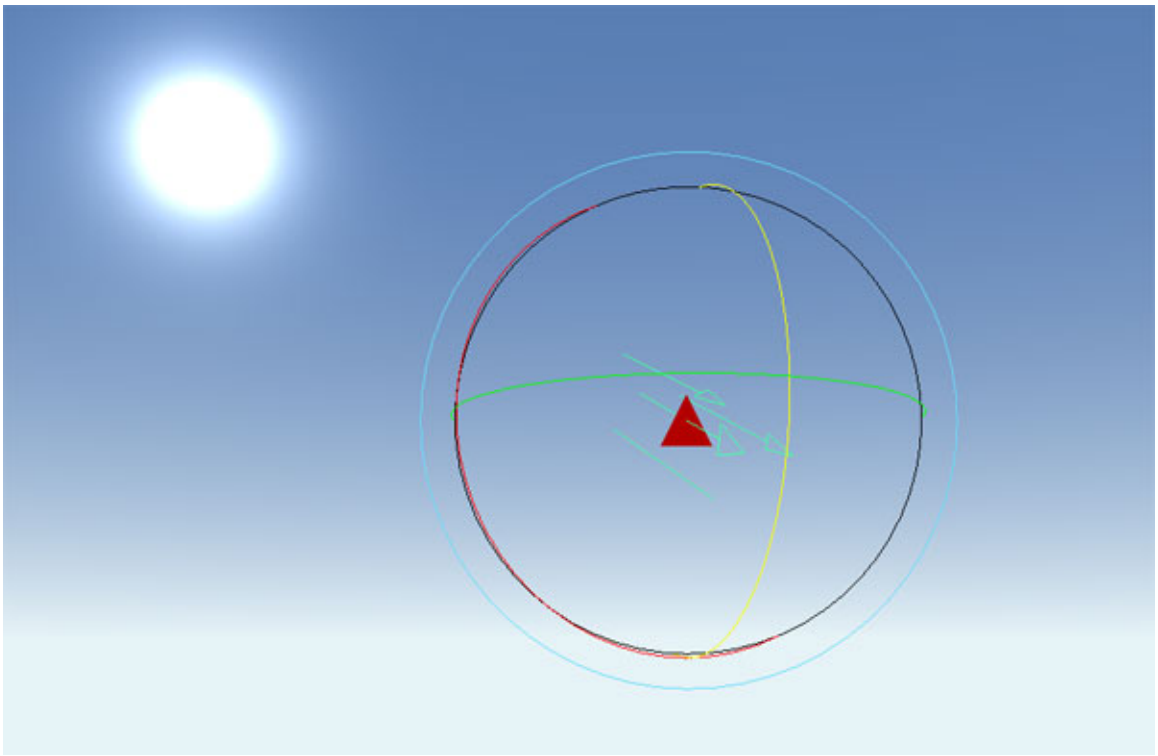


These provide the background 'coloring' of the sky and sun before any other sky elements are rendered.

> Only one of these background node types should live in a sky scene.

Create Basic Sky

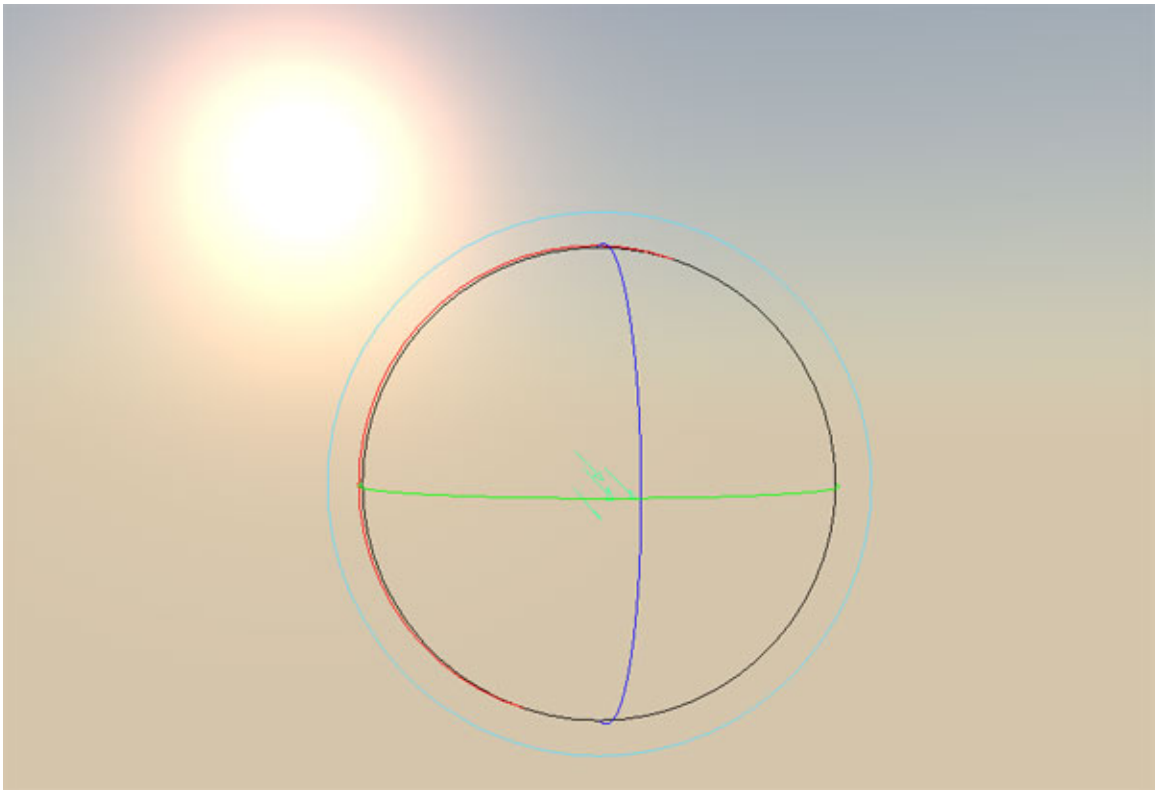
This sky background type is defined by two colors: a Zenith color and a Horizon color. It's the most basic form of sky and its use is encouraged for early stages in a sky modeling process. Pressing the button also renders a procedural sun, making use of the homeSkySun node.



Create Gradient Sky

This sky background type is defined by a 1-dimensional HDR color texture. The color gradient texture is sampled with UV position

0.0 at the bottom to 1.0 at the zenith. This sky type allows for much more artist control over the sky coloring. A procedural sun is also rendered, making use of the homeSkySun node.



After downloading the samples from https://home.scedev.net, example 4 x 512 gradient DDS files should be extracted and found here:

```
C:\HomeHDK \Build\Environments\Atmospheres\SkyGradients
```

Create Textured Sky

This sky type is defined by a standard HDR cubemap texture.



In this case the procedural sun with not be taken into account, although it will still be used to light the procedural cloud layer. The artist adjusts its direction, color, and intensity to match the HDR sky's sun position.

4. Assign Mesh To Sky



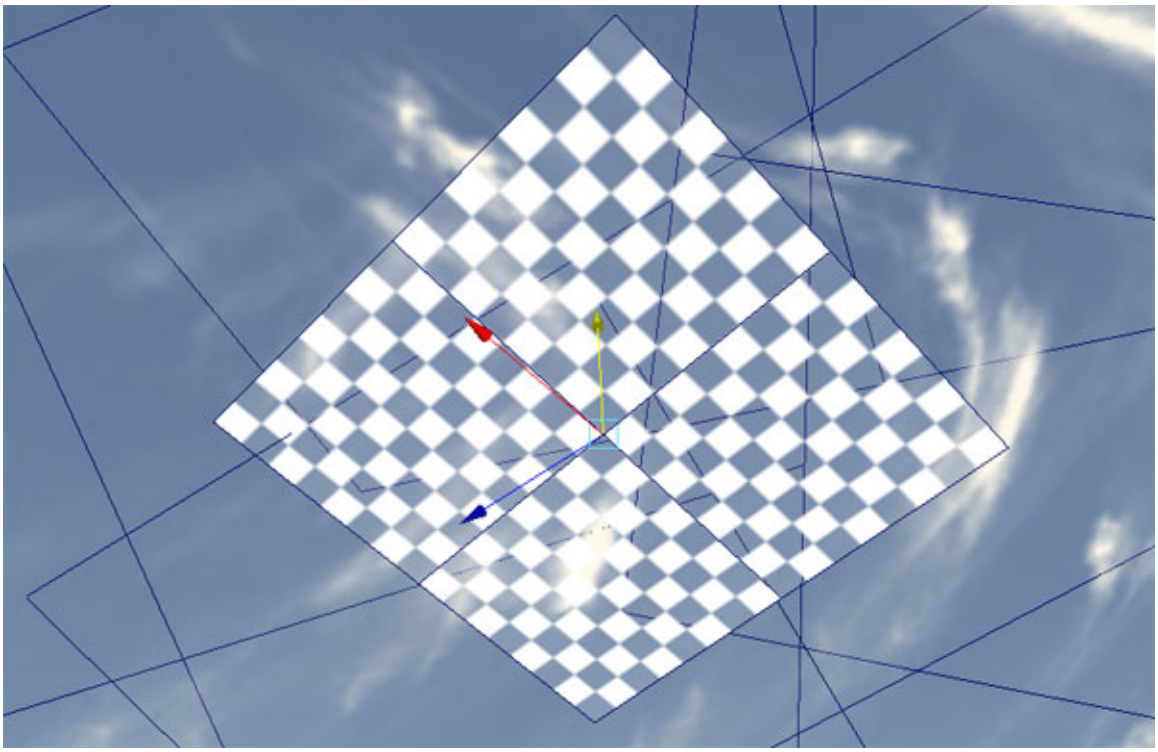
This button will assign/parent the currently selected Maya mesh to the sky scene, allowing it to be used as a sky sprite cloud.

5. Create Sprite Cloud

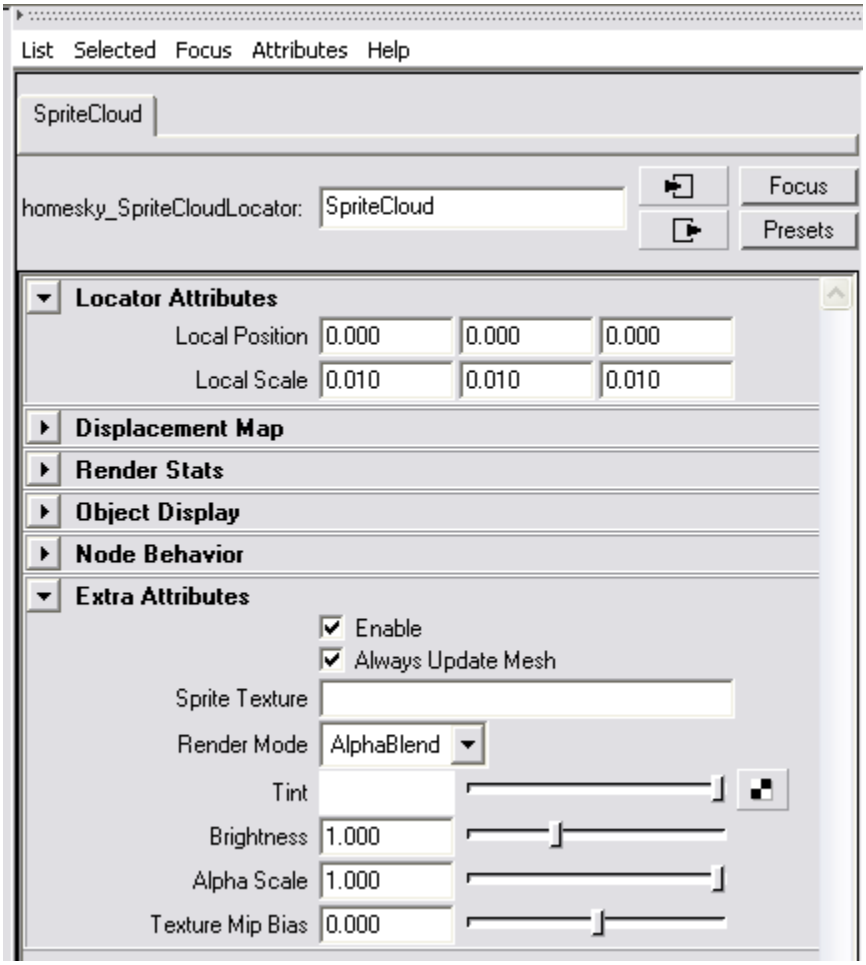


This button automatically creates a simple square Maya polygon mesh with 9 vertices and a default checkered texture. Move and position the new polygon as you would any usual Maya mesh.

Sprite meshes provide a way of rendering arbitrary textures in the sky. For example, wispy high clouds, the moon, airplane vapor trails, etc.



Assign the texture by selecting the mesh in the Sky Objects list, then place the path for the .dds texture in the Attributes Editor > Extra Attributes.



The DDS texture should be stored in the Build folder to allow for correct exporting.

An example texture is available from https://home.scedev.net. It should be extracted to:

`build\environments\Atmospheres\CloudTextures\wispycloud.dds`

6. Creating Fog

Create Exp Height Fog



This creates the nodes responsible for fogging the sky and sky elements. This fogging type is a single colored fog, with density that falls off exponentially with altitude. It gives a natural look for large distances where things fade as they recede towards the horizon.



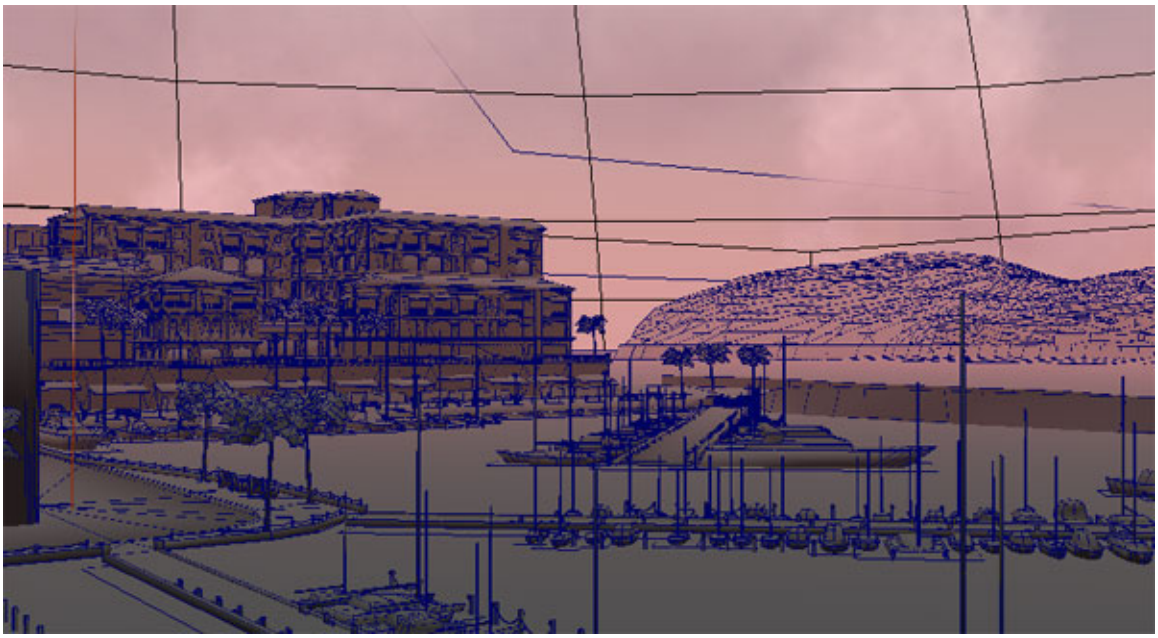
Each fog (Exp Height and Scene) can have different colors, but this will not appear natural when viewed at runtime. It is encouraged for each to have the same exact color setting (but not necessarily the same density settings). Parameters to allow synchronizing with each are available on each fog node.

Create Scene Fog



This creates a sceneFog nodes responsible for fogging the main Home scene geometry. This fogging type is a basic single colored uniform density fog model.

If an AtgiLocator node is present in the scene, then the sceneFog node will render the AtgiLocator geometry as solid and fogged. This allows for quick preview of how the scene geometry will look with fogging.



Each fog (Exp Height and Scene) can have different colors, but this will not appear natural when viewed at runtime. It is encouraged for each to have the same exact color setting (but not necessarily the same density settings). Parameters to allow synchronizing with each are available on each fog node.

7. Create Particle Cloud Dome

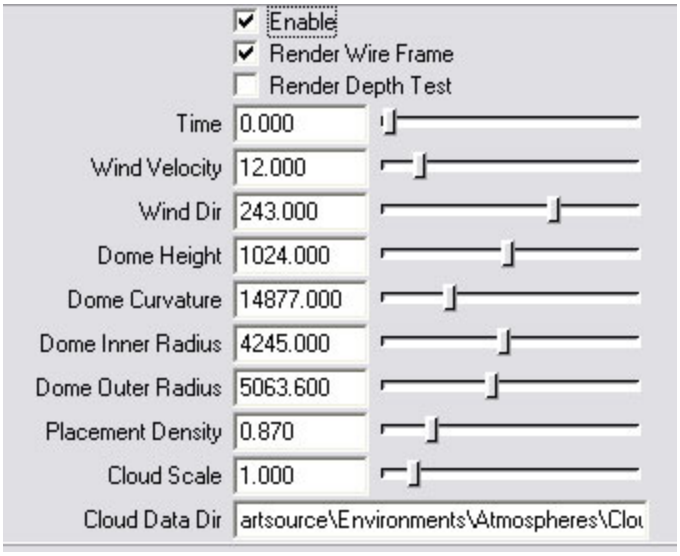


Clicking this button will open a file finder dialog. Use this to locate a cloud library .xml file as modeled with the Cloud Modeler UI. If no clouds have yet been modeled by the user, then the sample cloud libraries provided in the examples can be used, for example:

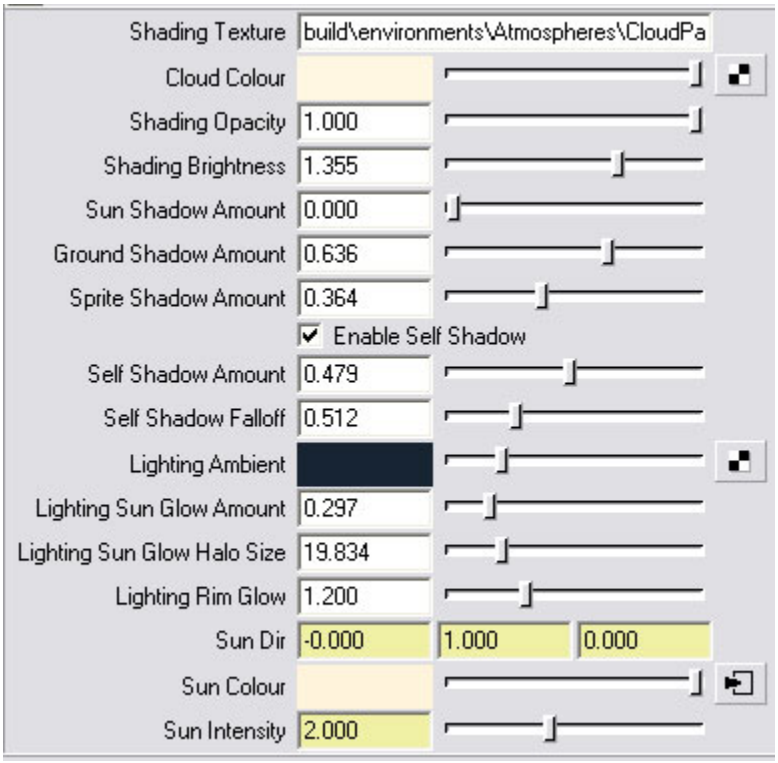
```
artsource\Environments\Atmospheres\CloudLibs\SampleCloudLib1.xml
```

After a file is selected, a particleCloudDome node and a particleCloudShadingParams node will be created in the scene.

Parameters to control the dome height, radius, curvature and cloud placement are located on the particleCloudDome node.



Parameters to control the shading of the clouds are found on the particleCloudShadingParams node.



The Shading Texture attribute takes path to the texture used for rendering the particle sprites. It uses a special format. For more information, see Sky Design UI Nodes.

An example texture is provided in the HDK:

```
build\environments\Atmospheres\CloudParticleTextures\cloud_particle.dds
```

8. Import Cloud Shading Params



This allows the user to import only the cloud shading parameters from a cloud library. Clicking the button will open a file dialog. As with creating the Particle Cloud Dome, direct this to a cloud library .xml file.

9. Refresh Particle Dome



Although designed to be automated as much as possible, sometimes the texture or shape of clouds rendered by the ParticleCloudDome will need updated to reflect a parameter change. Clicking this button will force an update.

10. Setup Scene



This button sets up the initial empty Maya scene with groups and data necessary for a Home sky scene, creating the homeSkySun and skySceneSettingsShape nodes.



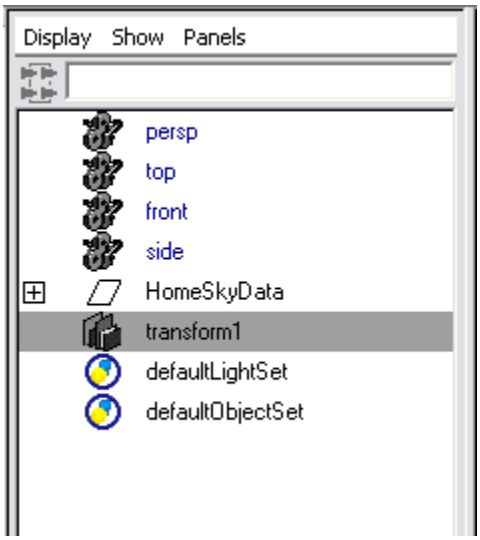
11. Make Atgi Locator

An Atgi Locator is a node which allows the display of a previously exported Home scene for reference within Maya as a wireframe. It does this by reading the .atgi intermediate export file.

1. Click the Make Atgi Locator button. This will create an Atgi Locator node in the Outliner called transform#, in this example, transform1.



2. Select the AtgiLocator node (in this example, transform1). You will see the node's attributes in the Attributes Editor window.



3. Direct the AtgiLocator node to read an .atgi file by clicking  in Attributes Editor > Atgi Locator Attributes.

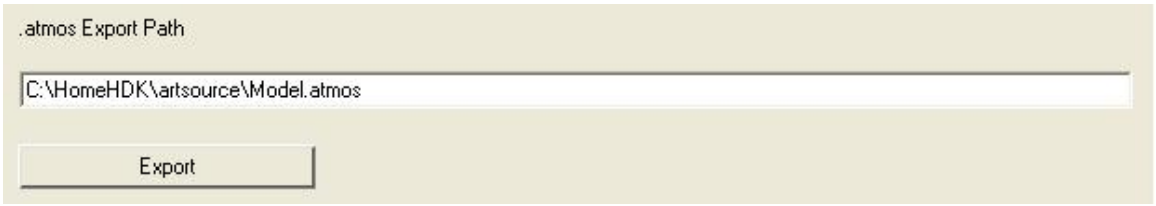


An example location is:

```
C:\HomeHDK\Intermediate\Environments\Example_Scene\Example_Scene.atgi
```

The purpose for this is to allow for easier modeling and positioning of cloud elements in the scene. The Atgi Locator is not essential and is ignored for final export.

12. .atmos Export Path



The Home sky scene data will export to single file with extension .atmos.

The full path for the destination .atmos file needs to go into the XML file path field.

13. Export Button



Clicking on the Export button will save out the data. Check the Maya Script Editor window for export feedback info.

> If a ParticleCloudDome is in use, a second file with extension .cdata will also get exported to the same location. This .cdata file contains the vertex geometry and shading data for the clouds.

Creating a Home Sky

To create a Home sky scene:

1. Initialize a Home Sky scene by clicking the Setup Scene button in the Sky UI.
2. Create a transform node (an .Atgi Locator node), and import an .atgi file.
3. Create a background sky.
4. Create clouds.
5. Add fog.
6. Export the .atmos file.

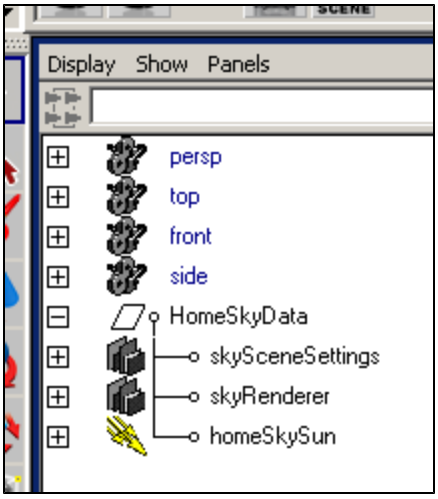
You can then add the .atmos asset to your scene through the Scene Editor. See Adding Sky to Your Scene

The creation of the sky, the clouds, cloud dome, and fog are all optional for a scene. You do not need to create all or any of these elements to successfully export your scene. But, there are advantages to using the sky tool to create your scene than by creating a sky dome as part of the static scene. You can create weather with the Sky Tool, from a beautiful day with a light breeze and a few roaming clouds in the sky, to a maelstrom of violent clouds and wisps swirling around the scene. Or you can use the sky tool for scenes that do not have a sky, like an underground scene with steam that undulates around the users' feet.

Initializing a Home Sky Scene

1. Open the Home Sky UI in a new Maya scene by selecting File > New Scene
2. Click Setup Scene to initialize the empty Maya scene to be a Home Sky scene.

   This creates a new Group in the Maya Outliner called HomeSkyData. It should contain 3 nodes:



- skySceneSettings: This node contains parameters which affect the rendering of the sky scene on the PlayStation®3
- skyRenderer: This node is necessary for rendering of the sky elements in Maya, and is ignored during export.
- homeSkySun: This is a Maya directional light. It is used to define the direction, color and intensity of sunlight used by the various sky nodes.

Creating a Transform Node and Importing a .atgi File

The second step to sky design is to create a transform node (an .Atgi Locator node). The node will allow you to import the .atgi file of the scene that you are designing the sky around. The .atgi file is a wireframe model of your scene that is created when you export your scene from Maya. The .atgi file is automatically generated in your <HDK_INSTALL_DIR>\Intermediate folder.

> This step is not necessary, but a scene's wireframe is very useful as a guide when placing sky elements.

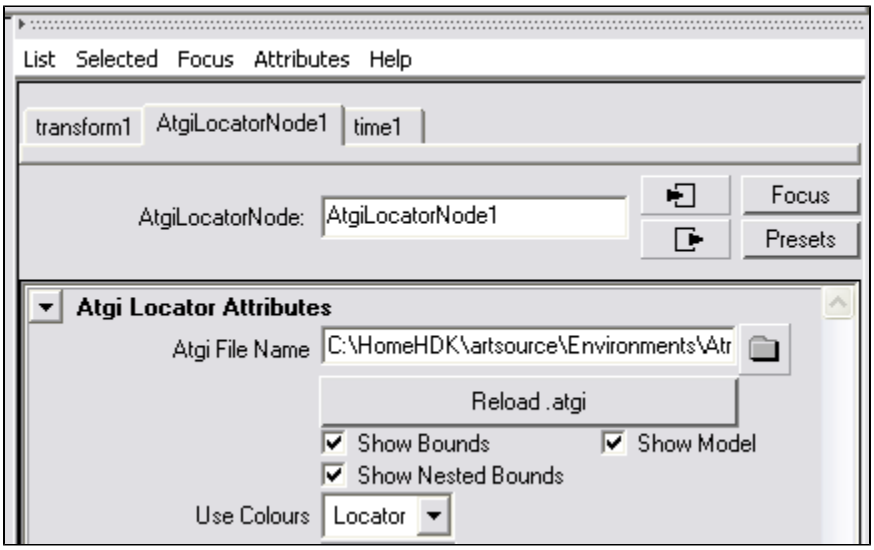
Click the Make AtgiLocator button.

In the Maya Outliner there should now be a node called transform1. This is the new AtgiLocator node. Select it and view its attributes in the Attribute Editor.

Importing an .atgi File

In Attribute Editor > AtgiFileName, the file name must be pointed to an .atgi file. The examples provided with the HDK Sky Tool make use of a sample .atgi file located here:

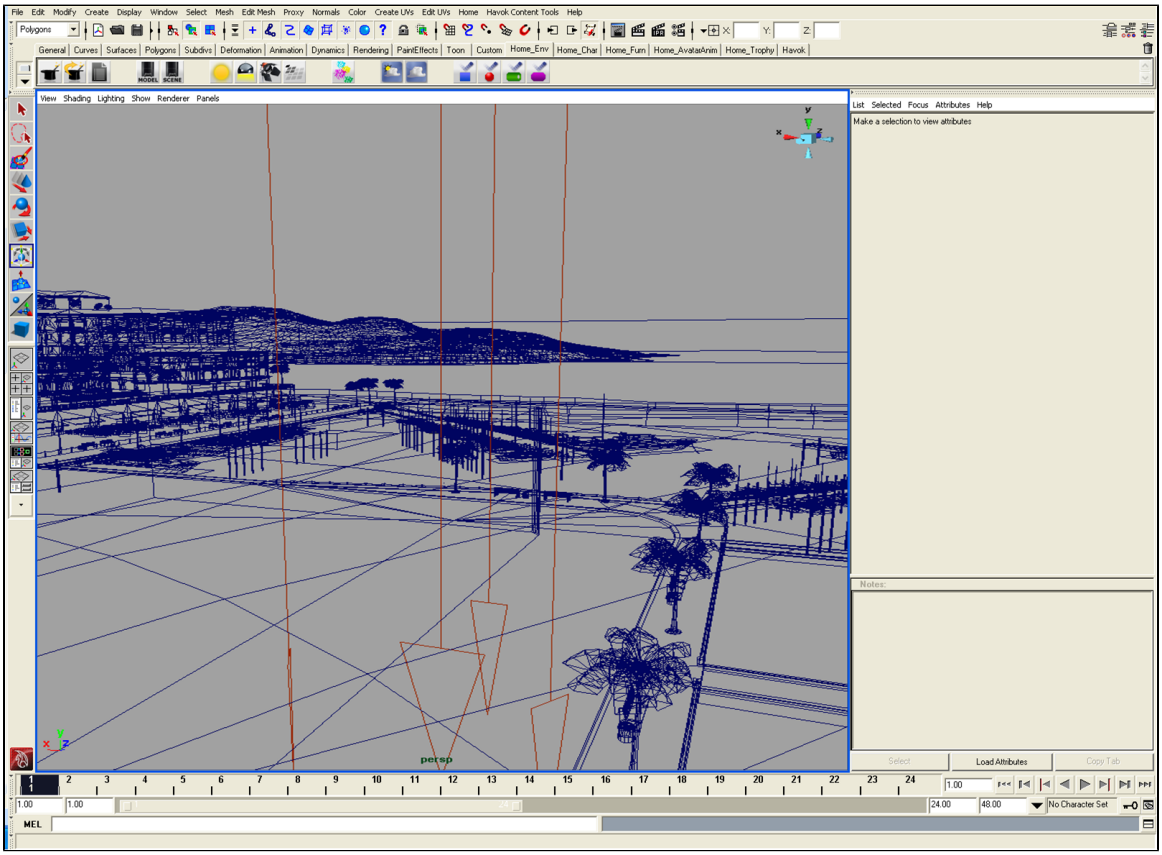
`artsource\Environments\Atmospheres\ExampleMayaScenes\Example_ATGI_Scene.atgi`



Use this file if you are currently not working on a Home environment. If you are working on a Home environment, you can find the .atgi file in the intermediate directory. For example:

c:\HDK\Intermediate\Environments\My_Scene\My_Scene.atgi

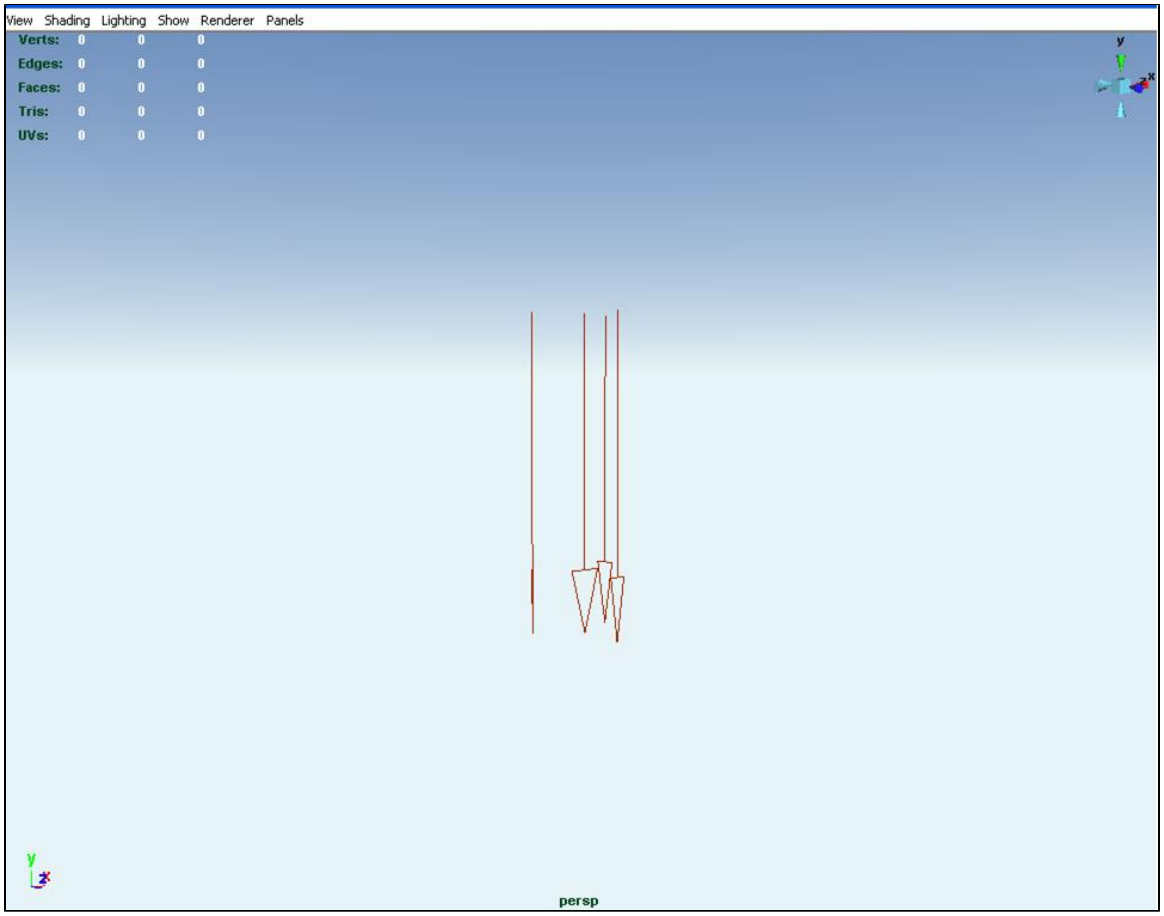
You should now see a wireframe rendering of the Home environment in Maya. We are now ready to start building the Sky.



Creating a Sky

The background color is the gradual sky color that stretches from horizon to zenith and will render behind all other sky elements in the scene. The easiest and fastest way is to create a background sky color is to use the BasicSky node. Follow the appropriate steps below for creating a basic sky, gradient sky, or textured sky.

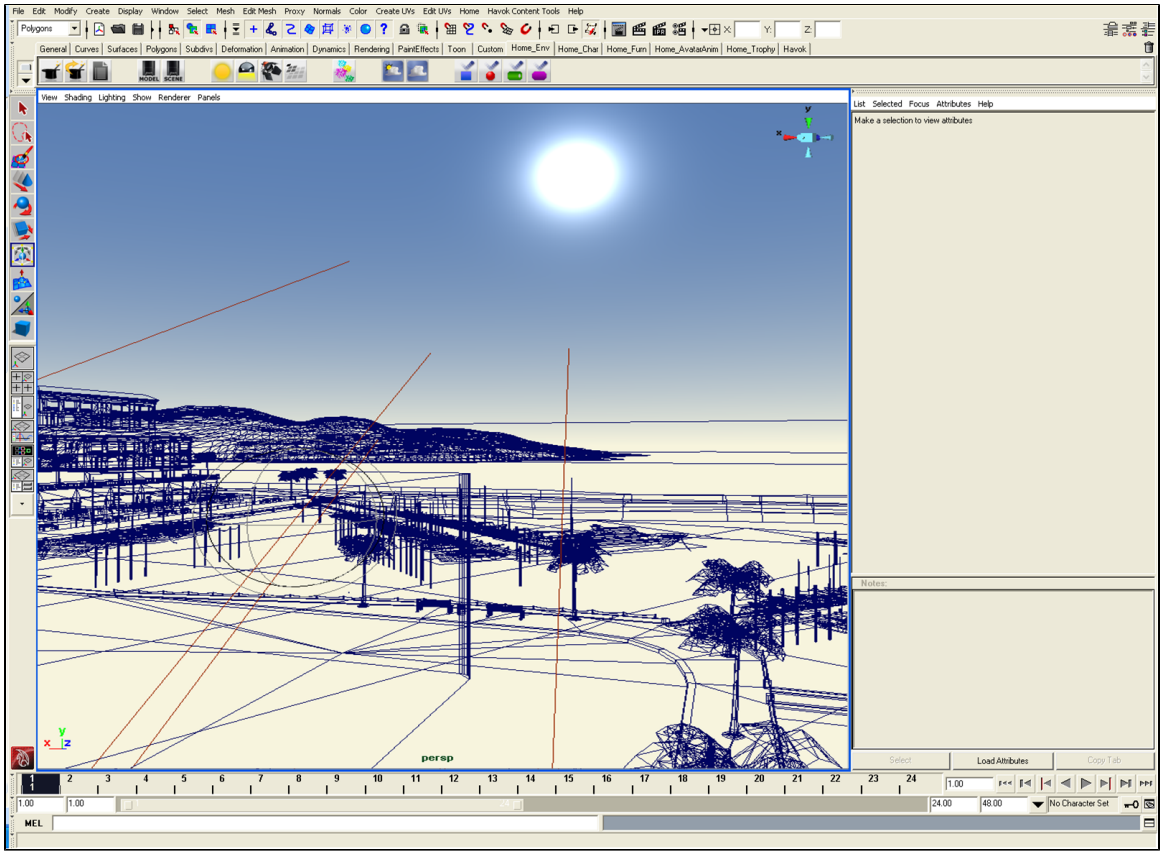
Creating a Basic Sky



Click the Create Basic Sky button in the Home Sky UI. The BasicSky node will be visible in the Outliner and in the Home Sky UI, and the sky scene will be colorized.

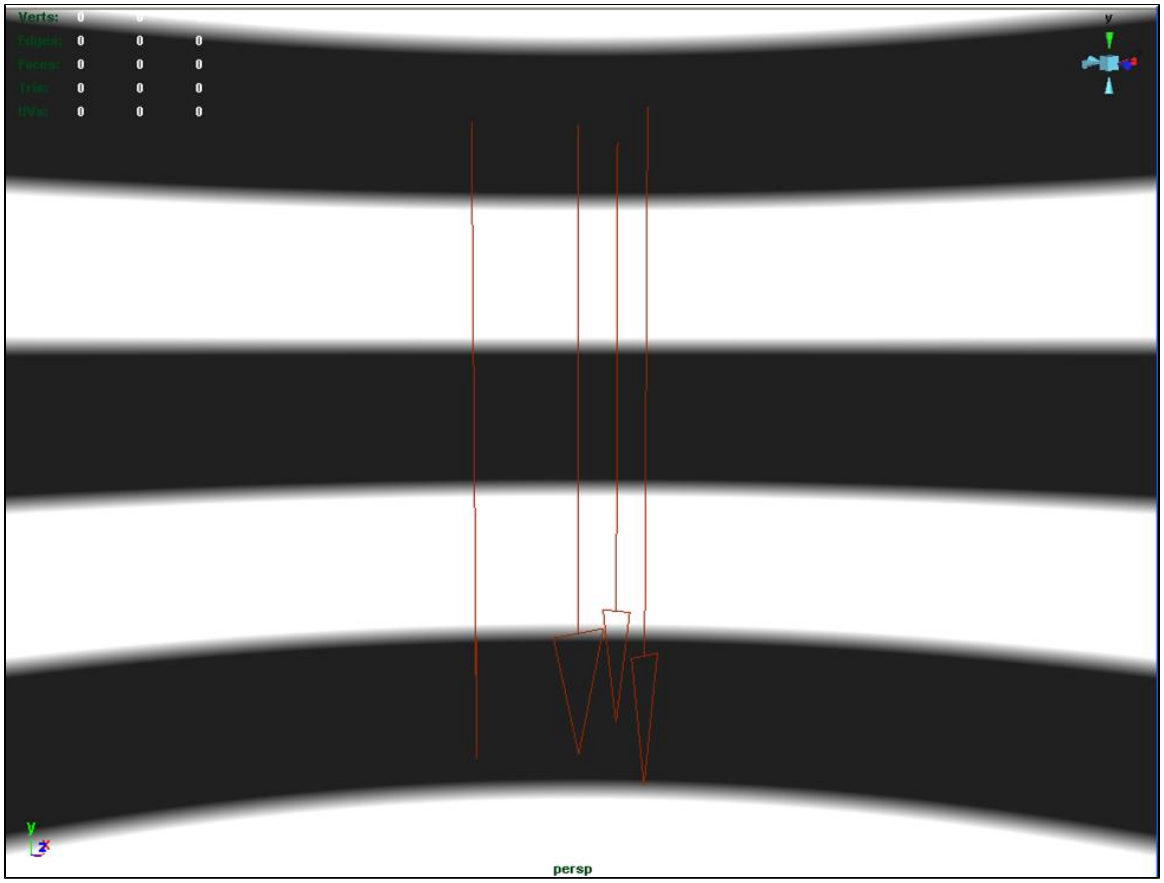
Changing the Background Sky Color

With the BasicSky node selected in the Home Sky UI, view its attributes in the Attribute Editor. It colors the sky by defining a zenith color and a horizon color, and interpolates between them using a falloff ramp value. The BasicSky node will also render a sun which can be tweaked in the Attributes Editor.



You can also adjust the sun by selecting homeSkySun node and going to the Attributes Editor.

Creating a Gradient Sky



Click the Create Gradient Sky button in the Home Sky UI. The Gradient Sky node will be visible in the Outliner and in the Home Sky UI, and the sky scene will be striped black and white by default.

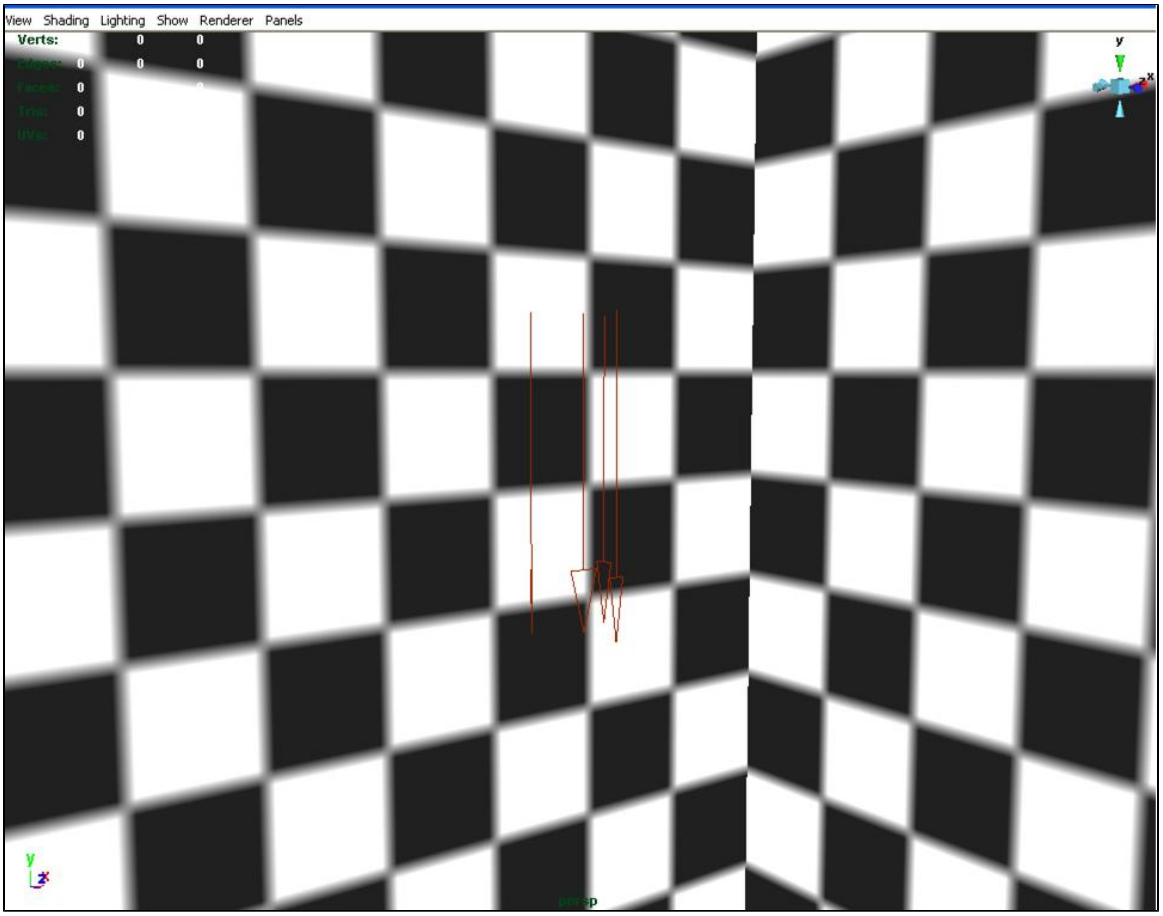
Loading a Gradient Texture

With the GradientSky node selected, go to Attribute Editor > Extra Attributes > Gradient Texture and enter the file path of the gradient texture that you want to use.

The file must be located in your <HDK_INSTALL_DIR> folder.

Once the texture has been loaded, adjust the Brightness and Xcoord values.

Creating a Textured Sky



Click the Create Textured Sky button in the Home Sky UI. The TexturedSky node will be visible in the Outliner and in the Home Sky UI, and the sky scene will be chequered black and white by default.

Loading a Cube Map Texture

With the GradientSky node selected, go to Attribute Editor > Extra Attributes > Cube Map Texture and enter the file path of the cube map texture that you want to use.

The file must be located in your <HDK_INSTALL_DIR> folder.

Once the texture has been loaded, adjust the Brightness and Rotation values.

Creating Clouds

You can add the following types of clouds to your scene:

- Particle clouds (puffy clouds): To add puffy clouds:
    1. Use the Cloud Modeler UI.
    2. Adjust the shape of the cloud and the density of particle spheres.
    3. Create a Particle Cloud Dome through the Home Sky UI that will randomly fill the sky with the particle clouds that you created.
- Sprite clouds (wispy clouds): You add the clouds through the Sky UI.

Creating Particle Clouds

Click the Create Cloud button in the Home Cloud Modeler UI. Once created:

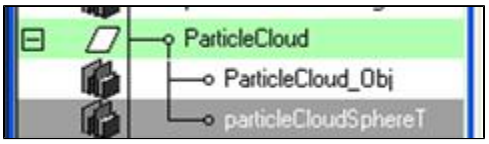
- Scale (for example, stretch, rotate) and position the cloud sphere at an appropriate place on the scene.
- Tweak the particle density and particle size parameters on the ParticleCloud_Obj and particleCloudSphere nodes to fill the cloud with particles.

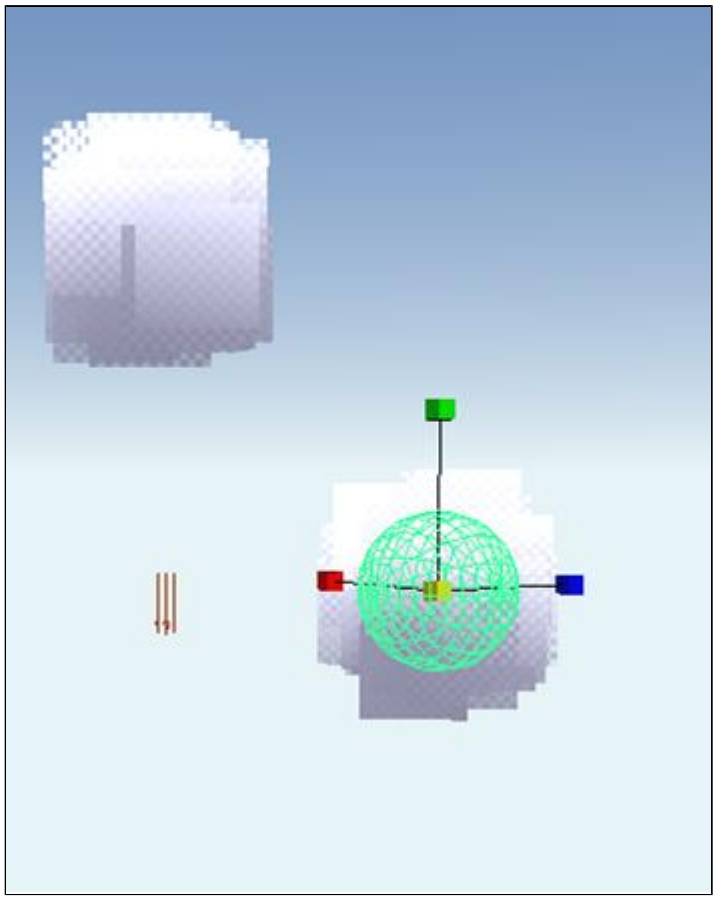
Adjusting Cloud Spheres

You can adjust the shape of each ParticleCloud through particleCloudSpheres. Each Particle Cloud is created with a

ParticleCloudSphere node, which you can select in both the Home Cloud Modeler UI > Cloud Objects, and in the Maya Outliner.

1. Select the ParticleCloudSphere node for the Particle Cloud you want to adjust.



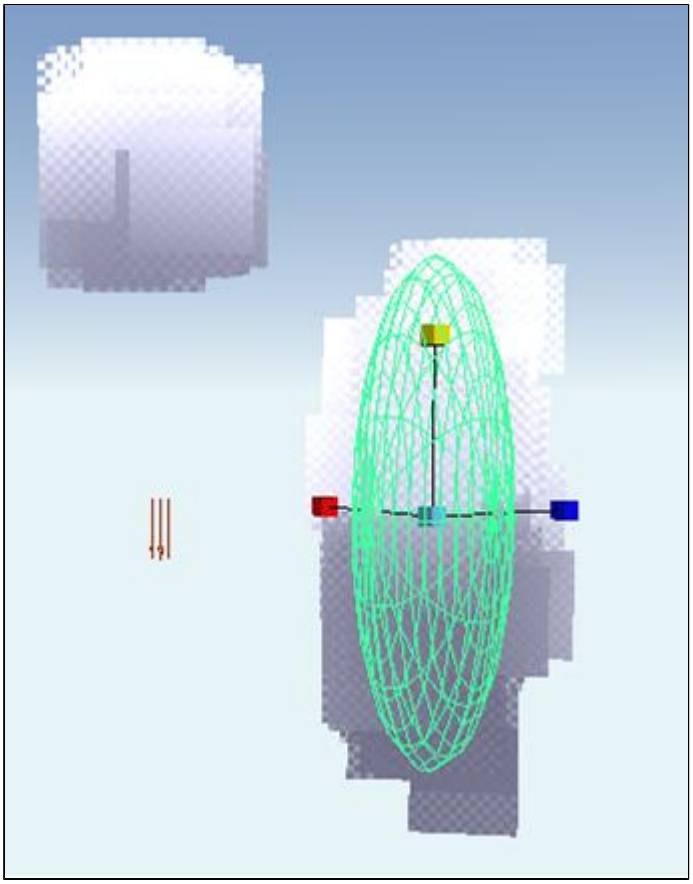


2. Adjust the Particle Cloud using the:

Maya Tool	Effect

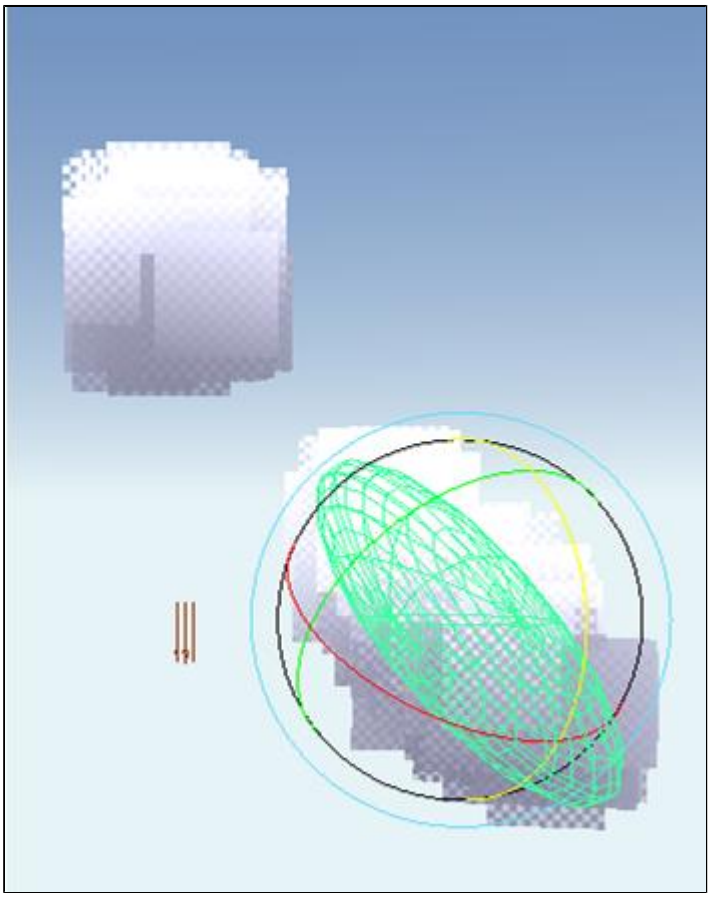


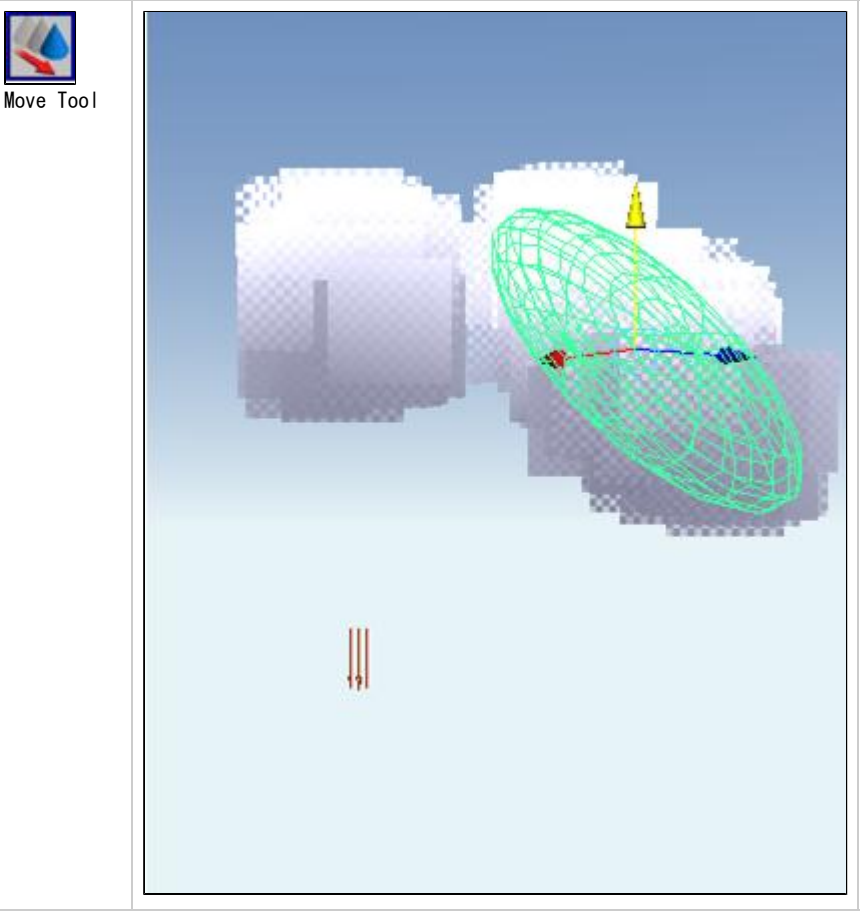
Scale Tool





Rotate Tool





3. If required, add another particleCloudSphere node by selecting particle cloud, then Home Cloud Modeler UI > Create Cloud Spheres, and repeat step 2.
4. Repeat steps 2 and 3 for each particle cloud until you are finished.

> Do not make more than five particle clouds for a sky scene.

Creating a Particle Cloud XML File

Once finished adding and adjusting particle clouds, in the Home Cloud Modeler UI:

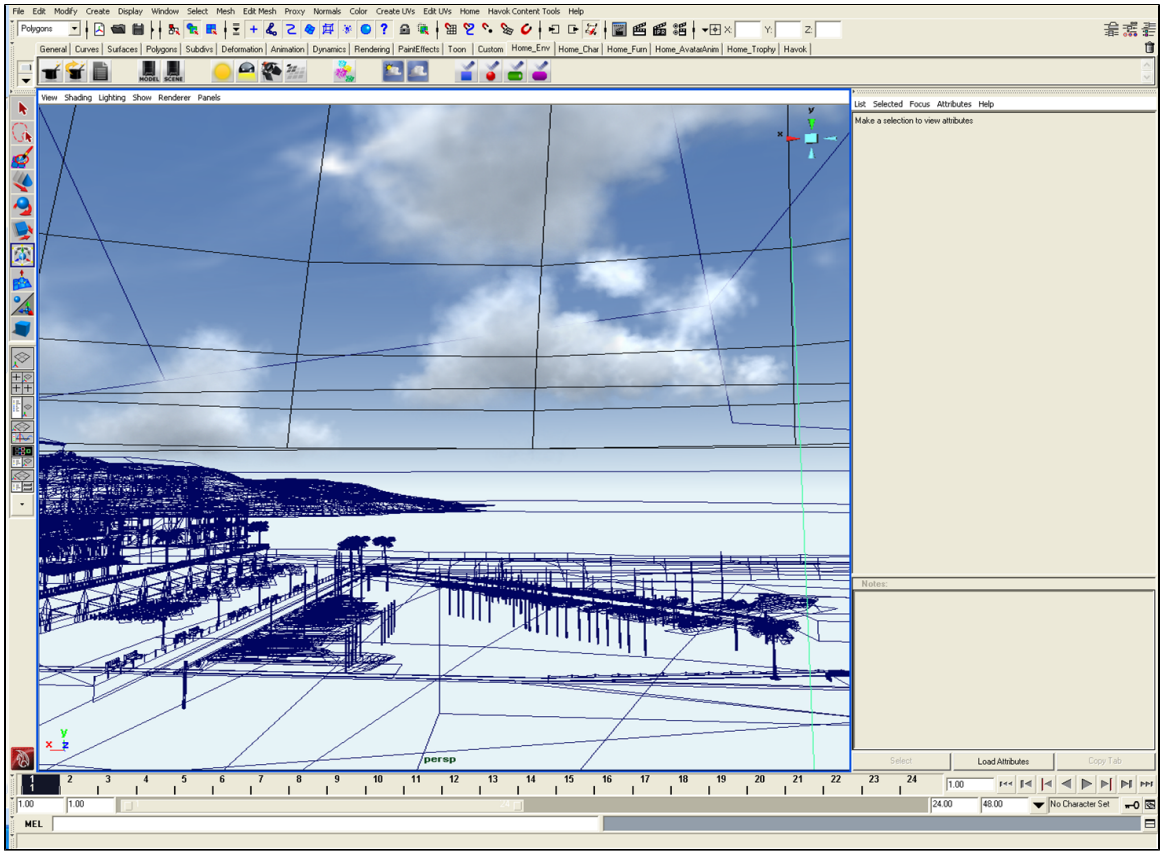
1. Enter a path and filename into the XML Export Path field.
2. Click Export Cloud Data.

> The XML Export file path needs to be in your <HDK_INSTALL_DIR>. We recommend using a filename that you will easily recognize for your scene (for example, mySceneClouds.xml).

Creating a Particle Cloud Dome

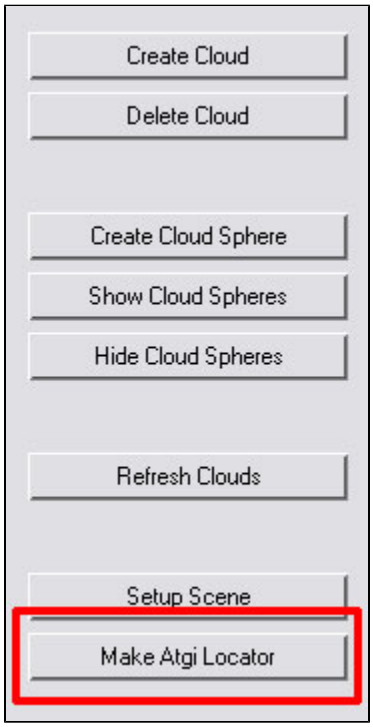
Click the CreateParticleCloudDome button in the Home Sky UI. A file dialog will appear. Direct the dialog box to the particle cloud .xml file that you just created.

You should now see the sky filled with your particle clouds.



Adjusting the particle cloud dome's properties in the Attribute Editor.

> Output of the number of clouds generated and memory consumption will be shown in the script editor.



The maximum number of clouds allowed is 250.

Adjusting the ParticleCloudShadingParams Node

Another node that is created when you create the ParticleCloudDome is Particle Cloud Shading Parameters. This node defines the shading, coloring and texturing parameters used for rendering the clouds. Select this node, then go to the Attribute Editor to adjust its parameters, for example, cloud color and shadowing amounts, etc.

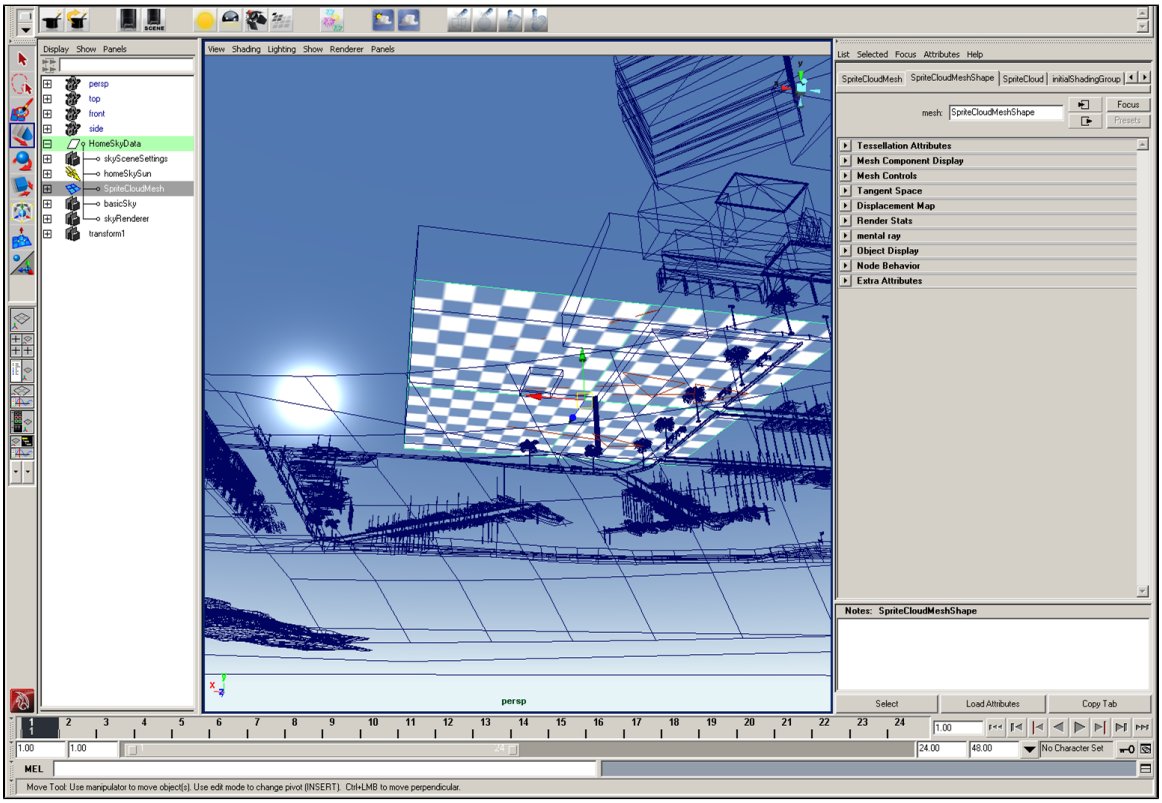
There is an example cloud particle texture located in <HDK_INSTALL_DIR>\build\environments\Atmospheres\CloudParticleTextures\cloud_particle.dds.

Creating Sprite Meshes

Sprite meshes provide a way of rendering arbitrary textures in the sky, for example, wispy high clouds, the moon, airplane vapor trails, etc.

Click Home Sky UI > CreateSpriteCloud.

The new SpriteCloudMesh object will appear in the Sky Object list and in the Maya Outliner. Initially it will be a quad shaped mesh created at the origin, and will render with a checkerboard texture.

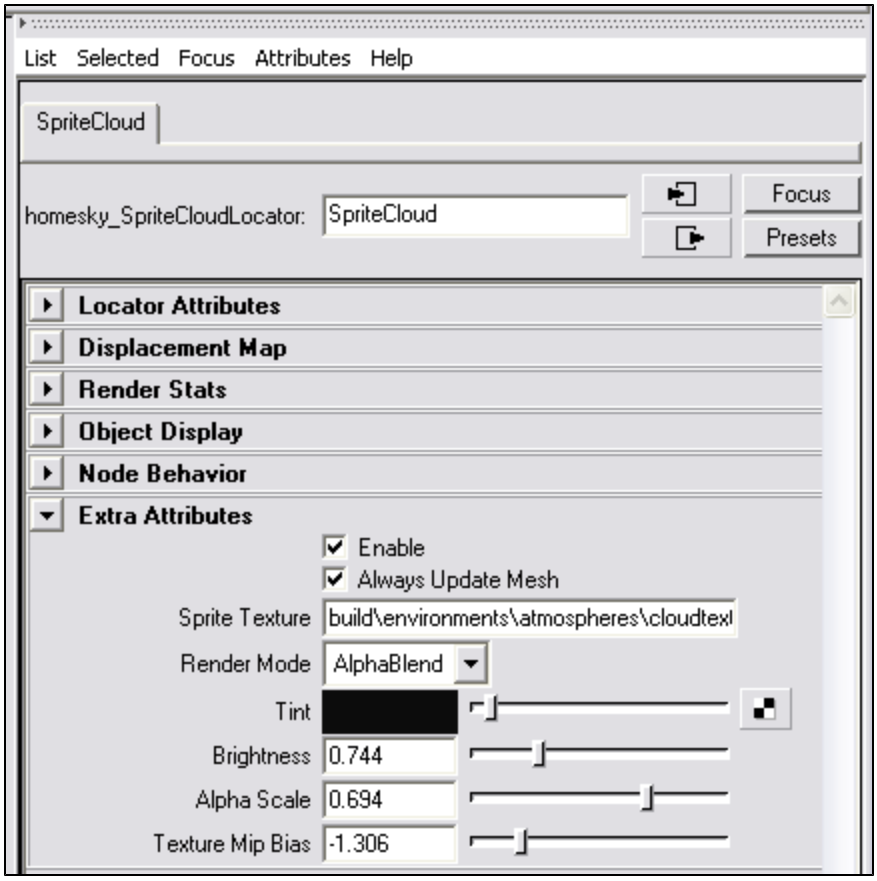


Setting a Sprite Cloud Mesh

Select SpriteCloudMesh in the Sky Objects list, then go to Attribute Editor > Extra Attributes > Sprite Texture.

The example below uses the cloud texture provided in the HDK. Copy and paste the path:

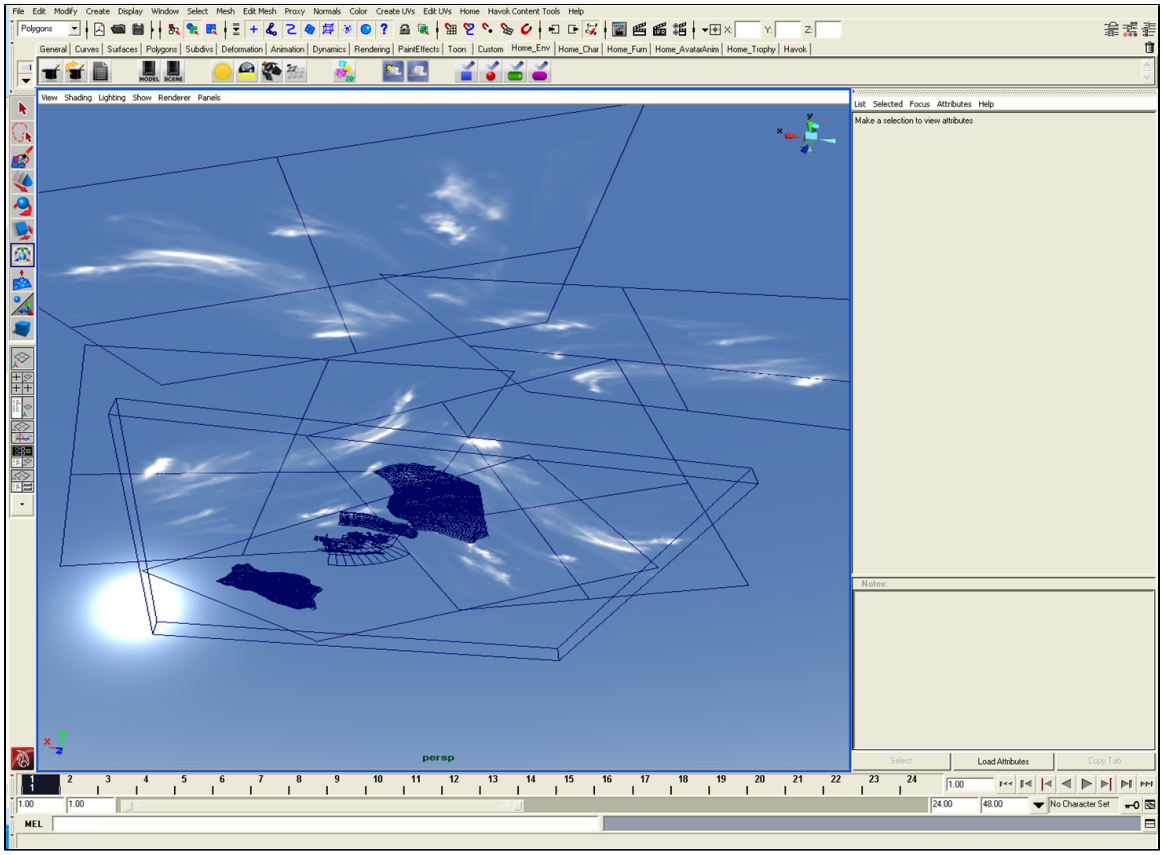
build\environments\Atmospheres\CloudTextures\wispycloud.dds



You can also adjust the brightness and color of the texture here.

> No Maya materials or Shaders are used by the Home Sky Tool. All texture paths and shading attributes are defined on the nodes themselves.

Move, scale and rotate the mesh into an appropriate location in the sky. Duplicate several times to provide a good coverage across the sky.



Creating Fog

There two fog types that you can add to your scene, each with their own effects: sky fog (fog effect on the sky) and scene fog (fog for the Home scene geometry).

For sky fog:

- Move Maya camera position to the height in the scene in which you want the fog to appear
- Create an ExpHeightFog node.
- Adjust ExpHeightFog settings.

For scene fog:

- Create Scene Fog node.
- Adjust Scene Fog settings.

If you use both sky fog and scene fog in a scene, the fog colors must be the same, otherwise a sharp discontinuity will appear when the environment geometry meets the sky.

Your scene may not require fogging, depending on artistic reasons or small walking / viewing distances a given scene may have. When this is the case it is better to skip this stage as the fog would increase the rendering times of a Home space on the PlayStation®3.

Creating Sky Fog

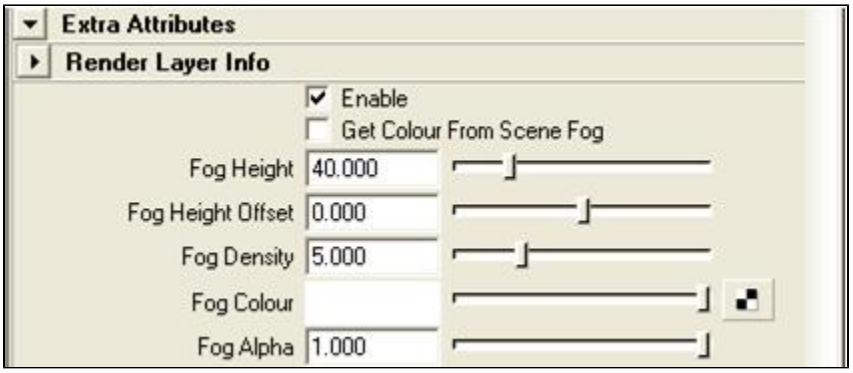
If the horizon in your destination Home environment cannot be seen, you may not need to use this node.

1. First, position the Maya camera to the location of your scene where you expect the player to occupy. The reason for this is that the effect given by sky fog varies greatly depending on the location of the camera. So it is best to model it using the intended view position in game. Once positioned, it may be useful to switch to Maya's flytool camera mode to view and model your scene in this section.
2. Click Home Sky UI > Create Exp Height Fog.

If your scene appears flat white after pushing this button, it means your camera is too low in the Y axis. To fix select the ExpHeightFog node in the Home Sky UI, and in the Attribute Editor > Extra Attributes adjust the FogHeightOffset attribute until you can see things again.

Adjusting ExpHeightFog Properties

For an explanation of each property, see Sky Design UI Nodes.



Fog Alpha: The higher the value, the more opaque the fog. To make modelling easier, choose a smaller value.

If you have already added scene fog, then be sure to check the Get Colour from Scene Fog box. The sky fog and scene fog need to be the same color.

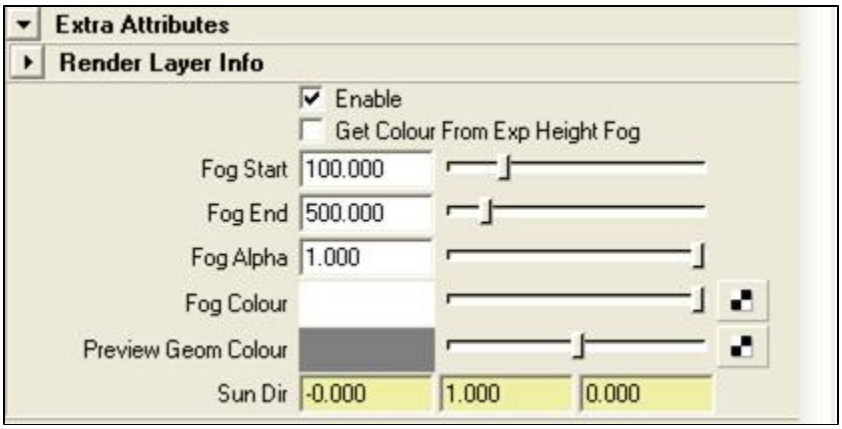
Creating Scene Fog

Click Home Sky UI > CreateSceneFog.

You will first notice the wireframe preview of your Home space, as rendered by the AtgiLocator, is now rendering solid. This is done by the SceneFog node, giving a preview of the fogging in the final space..

Adjusting Scene Fog Settings

Adjust the various attributes on this node to get a fogging value that looks right for your scene.



If you have already added sky fog, ensure you check the Get Colour from Exp Height Fog box. The sky fog and scene fog need to be the same color.

Exporting Your Home Sky

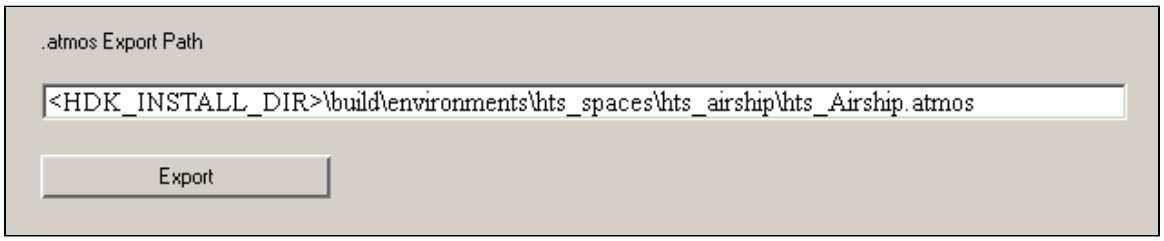
The Home Sky tool exports to an .atmos XML file. It will also produce a .cdata file of the same name if a particle cloud dome was used.

1. Go to Home Sky UI > .atmos Export Path.
2. Specify the full path of the desired .atmos file.
3. Click Export.

> The full path should be somewhere in <HDK_INSTALL_DIR>. It is encouraged to be exported to the same directory, and named the same, as the destination Home space.

> This convention is not essential. You can use a single .atmos file between multiple scenes.

For example, C:\HDK\build\environments\My_Scene\My_Scene.atmos



A dialog showing Finished is displayed indicating success. If a failure is shown, Output in the script editor will give information about possible reasons for the fault.

View Your Sky

You can export and load a sky asset developed in Maya to PS Home without reloading the entire scene through the PS3 Scene Viewer. You can also view your sky through the Dev Debug Console, and the HDK Browser.

You can view the sky after it is finished in a scene through the Scene Editor. See Adding Sky to Your Scene.

This tutorial uses the HTS Aeronautilus sample scene, which you can download from https://home.scedev.net/projects/samples.

Viewing Your Sky During Creation

To view your sky scene as you are creating it in Maya, load the .atmos file in the PS3 Model Viewer. You can load the PS3 Model Viewer from either Maya, or by launching the .self file from the Target Manager.

> We recommend loading the PS3 Model Viewer before working on the sky scene, especially if you want to view the sky as part of a scene.

Launching the PS3 Model Viewer from Maya

Maya can only launch the PS3 Model Viewer with successfully exported scenes, it looks for a .mdl (model) file to load, so it cannot load an .atmos file directly.

To load the PS3 Model Viewer from Maya:

1. Run Target Manager and connect to your PlayStation®3.
2. Load a successfully exported scene in Maya.
3. Click 

or select Home > Environment > PS3 Model Viewer.

Launching the PS3 Model Viewer from Target Manager

This method is faster as you do not need to load a scene in Maya first. You can also change the .mdl (model) file while in the PS3 Model Viewer.

To load the PS3 Model Viewer from Target Manager:

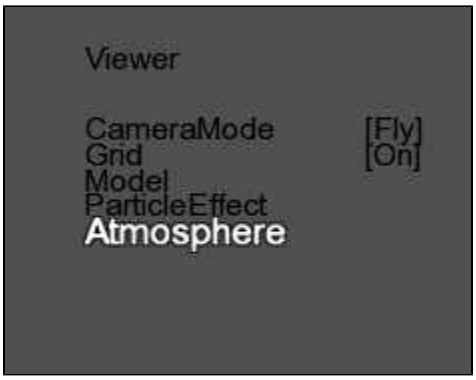
1. Click the Open Executable on Target button.
2. Enter --hdkbrowser in the Command line parameters.
3. Load the .self file.

For more information on launching the .self file, see Launching the Home Developer SELF.

Loading Your Sky

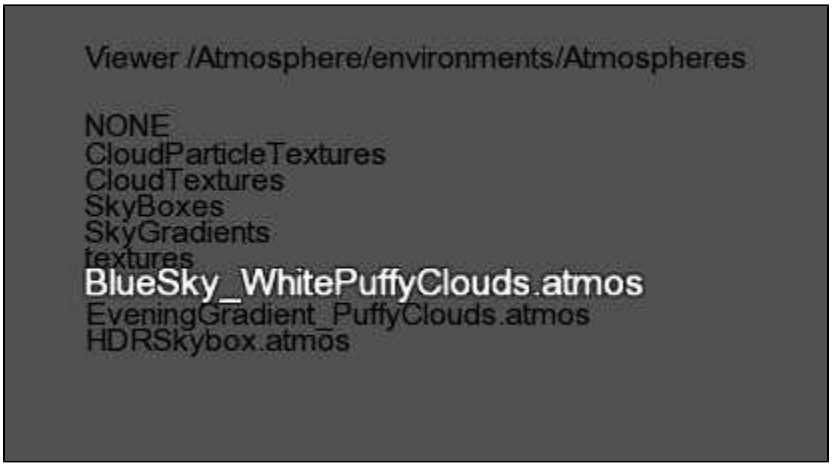
To load your sky:

1. Open the PS3 Model Viewer and select Atmosphere.

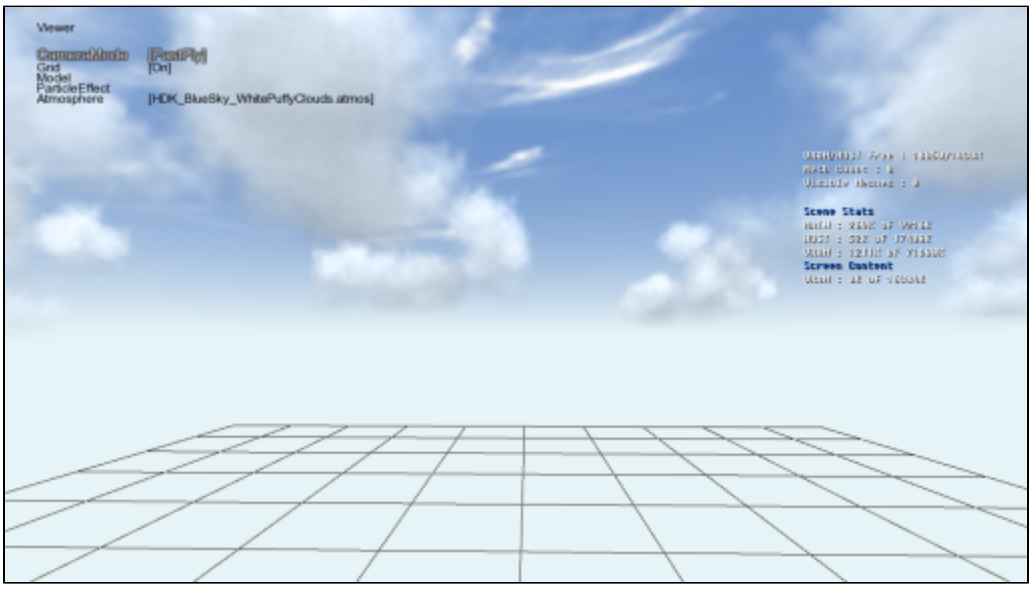


2. Navigate the <HDK_INSTALL_DIR> to find and select your .atmos file.

   This tutorial uses the sample sky scene, located at <HDK_INSTALL_DIR>\build\environments\Atmospheres\BlueSky_WhitePuffyClouds.atmos.

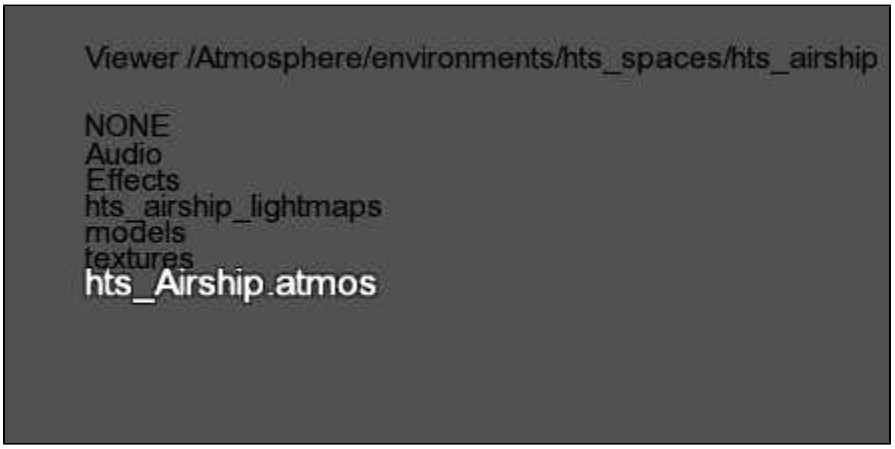


   The .atmos file loads and displays your sky.



   This tutorial uses the HTS Aeronautlius sky scene, located at

<HDK_INSTALL_DIR>\build\environments\hts_spaces\hts_airship\hts_Airship.atmos.



The .atmos file loads and displays your sky.



Quick Reload

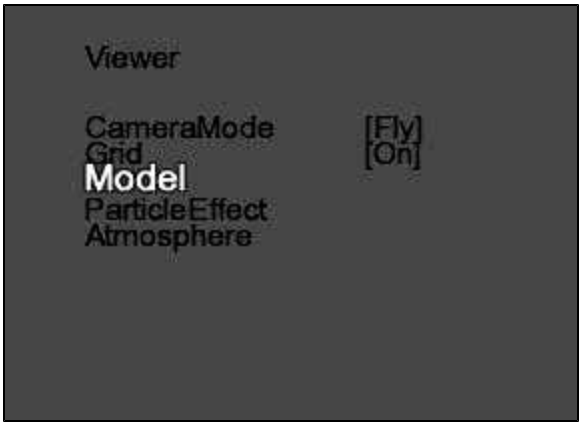
.atmos files can be viewed as soon as you export it from the Home Sky UI. You do not need to reload the PS3 Model Viewer. You do not need to reload Maya. This allows you to view changes to your sky scene almost instantaneously.
With Maya and PS3 Model Viewer running:

1. Edit your sky scene, and then export the .atmos file from the Home Sky UI.
2. Load the .atmos file in the PS3 Model Viewer (Follow steps 1 and 2 above).
3. Repeat as often as required.

Loading a Scene

To view a sky as part of a scene:

1. Open the PS3 Model Viewer and select Model.



2. Navigate your <HDK_INSTALL_DIR> to find and select the .mdl file for the scene that you want.

   This example uses the HTS Aeronautlius model file, located at <HDK_INSTALL_DIR>\build\environments\hts_spaces\hts_airship\hts_Airship.mdl.



   The .mdl file loads and displays the scene.



Quick Reload

Repeat steps 1 and 2 above to load a different .mdl file, or reload an updated .mdl file. This useful feature allows you to see

how your sky fits in with several Home scenes.

Viewing Your Sky Through the Dev Debug Console

Using the Dev Debug Console, you can execute console commands, profile the sky's effect on a scene, and quickly reload the .atmos file if you make an update in Maya.

Make sure that your scene has an Atmosphere File with an .atmos file allocated to it in the Scene Editor.

Opening the Dev Debug Console

There are several ways to open the Debug Console. See The Debug Console.

Reloading an Updated .atmos File

1. Update and export the .atmos file that is used in the scene that you are running in the Home Developer Self.
2. In the Dev Debug Console, type `atmos_reloadFromXML`.

Calling this without any arguments reloads the current .atmos scene directly from the disk. This is invaluable for tweaking final color and fogging values. The sky updates immediateIt. Any profiling GUIs that you have enabled are updated as well, showing the rendering effects the new .atmos file has on the scene.

For a full list of the console commands available, see Profiling in the Client.

> You cannot swap .atmos files using the Deb Debug Console. You can reload only the .atmos file that has been allocated to the scene through the Scene Editor.

To view Sky through the Scene Editor, see Adding Sky to Your Scene.

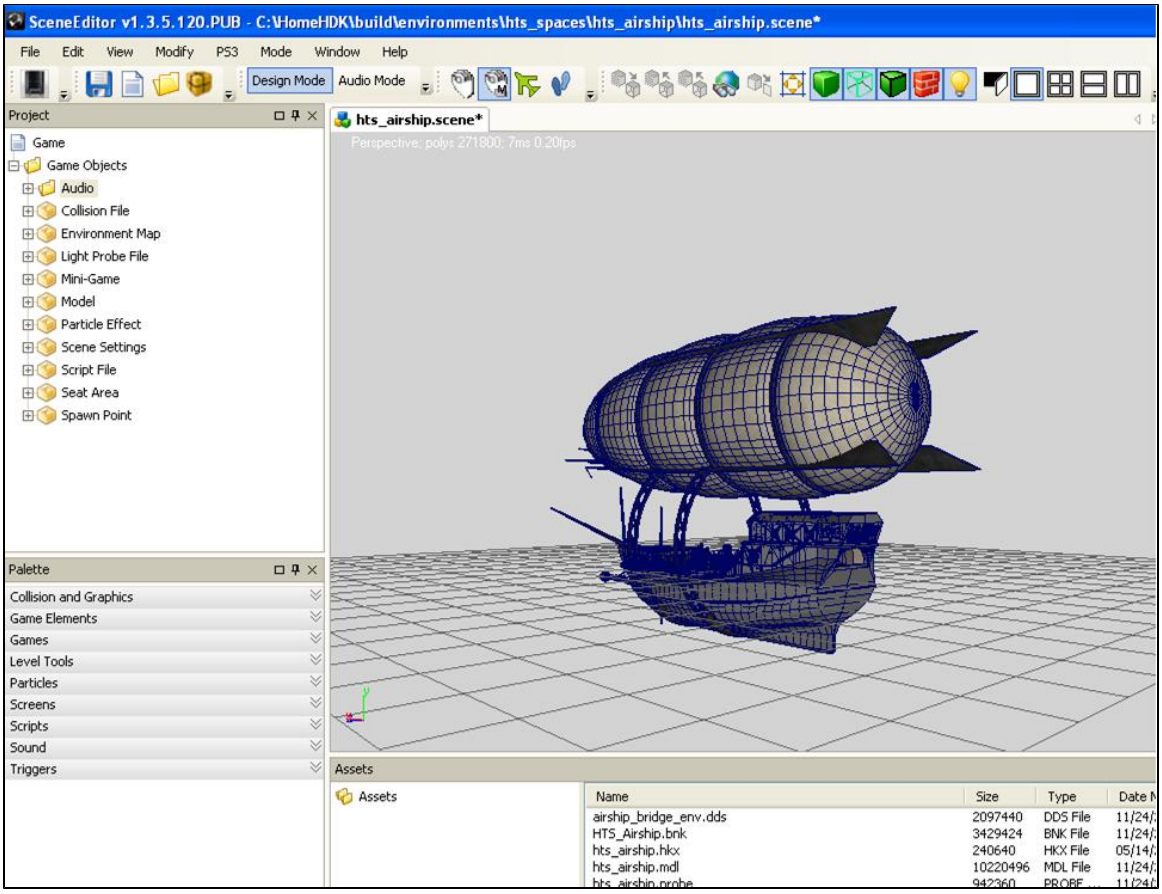
Adding Sky to Your Scene

You add your sky (.atmos file) to a scene using the Scene Editor.

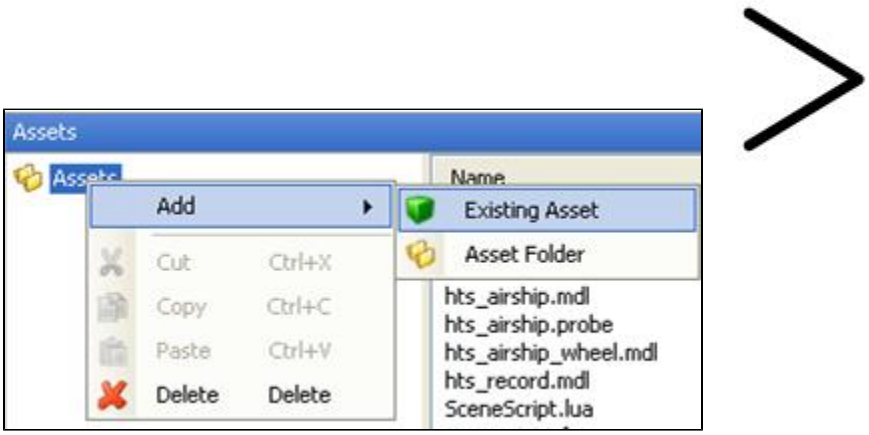
To add a sky to a scene:

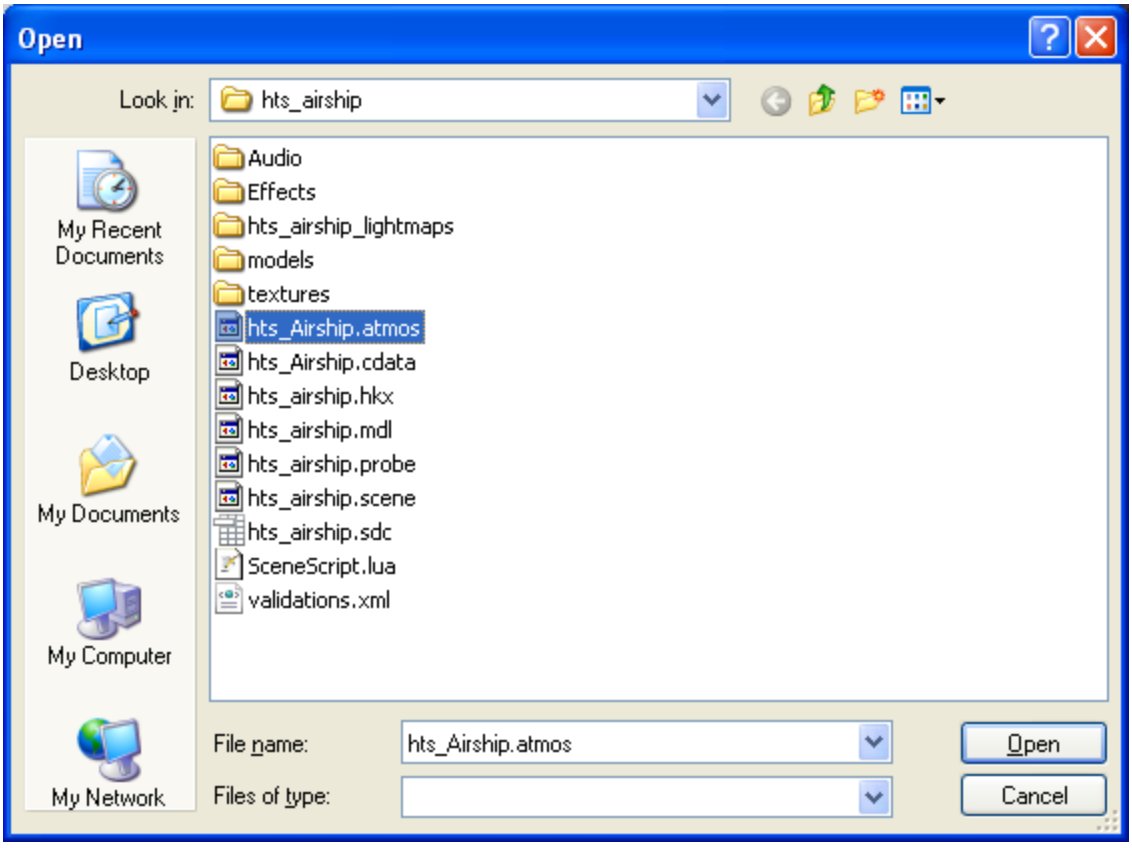
1. Open the Scene Editor and load the Home scene that you want to use.

   The scene example uses the HTS Aeronautilus sample which you can download from https://home.scedev.net/projects/samples.

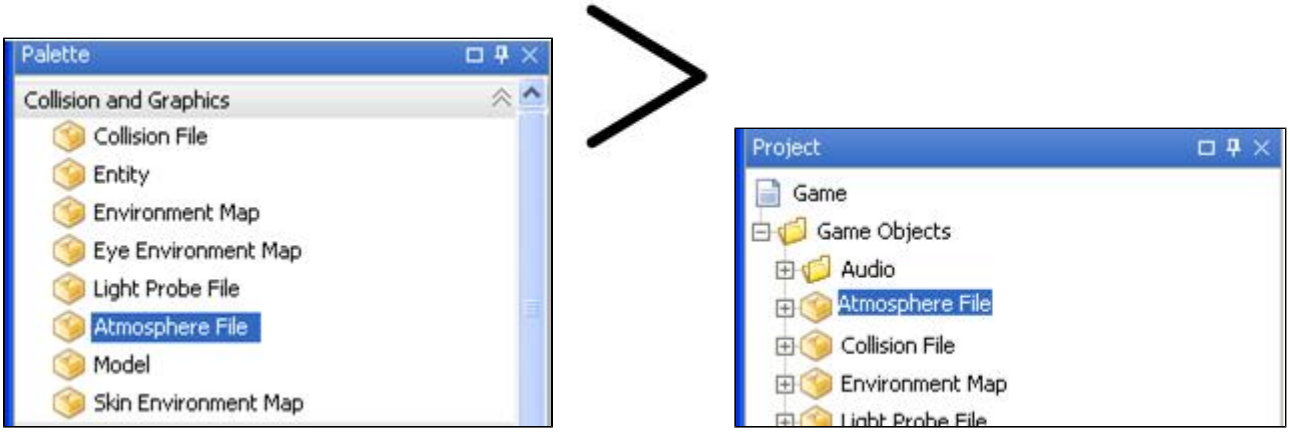


2. Add the .atmos file to your Assets by going to the Assets panel, right-clicking Assets > Add > Existing Assets, and then navigating to the .atmos file that you want to use.





3. Add an Atmosphere File to your Game Objects by going to the Palette panel, opening Collision and Graphics, and then dragging Atmosphere File object into the Project panel.

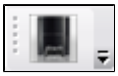


4. Expand the Atmosphere File object in the Project panel to see the [file] node. Drag the .atmos asset to this node. When successfully allocated, the node will turn green.



After you have added an .atmos file to an Atmosphere File object, the sky is visible when you launch your scene.

Previewing your Scene

To preview your scene, click  or select PS3 > Launch Home.

> Sky Profiling covers tools specific to sky profiling. For more information on profiling PS Home content, see Profiling in the Client.

Sky Profiling

You can use the HDK profiling tools to profile your sky scene, and make sure that it meets the Validation and Submission requirements.

For more information on profiling other types of content, see Profiling in the Client.

Sky Rendering Rates

The rendering of a typical sky scene should be between 1.5 to 2.0 milliseconds (the budget for the whole PS Home scene should be around 20 milliseconds (see Recommended Rates for Rendering and Processing - Scenes). So if you find the sky rendering over 2.5 milliseconds then it will need to be investigated further to establish what is causing the impact on resources.

Typical problems include:

- Too many sprites in the clouds (each should have between 100-200).
- Too many clouds in a scene.

A profiling view of the scene can be bought up by typing `profileMode overview` into the console in PS Home. Type `profileMode off` to turn it off.

The Dev Debug Console

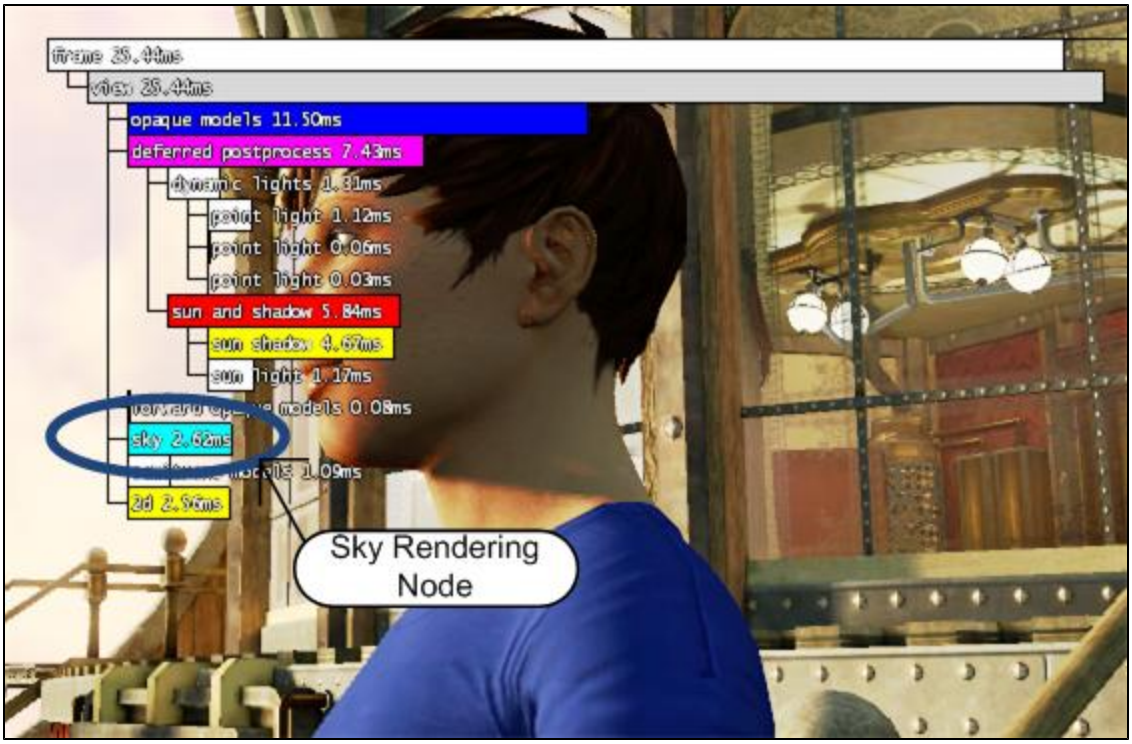
For a step-by-step guide to opening your scene, and the Dev Debug Console, see Profiling Scenes.

Opening Sky Profiling Tools

In the Dev Debug Console, type `profilemode overview`.

The profile frame rate overview will appear on the screen. Sky rendering has its own node, which usually appears towards the bottom.

> The sky rendering node appears only if your scene is rendering a sky.



Atmos Commands

The Dev Debug Console contains a number of commands related to the sky tool. They all start with prefix atmos_.

atmos_reloadFromXML

Calling this without any arguments reloads the current .atmos scene directly from the disk. This is useful for tweaking final color and fogging values. The sky will instantly update.

> Any Profile GUI panels that you have enabled are updated as well, showing the rendering effects the new .atmos file has on the scene.

Arguments	Effect
[None]	N/A

atmos_RenderEnable

This command takes a boolean that turns on/off sky rendering.

Arguments	Effect
1	Enables sky rendering.
0	Disables sky rendering.

atmos_SPUSortEnable

This command is used for internal purposes only. It has no profiling effects.

atmos_UpdateEnable

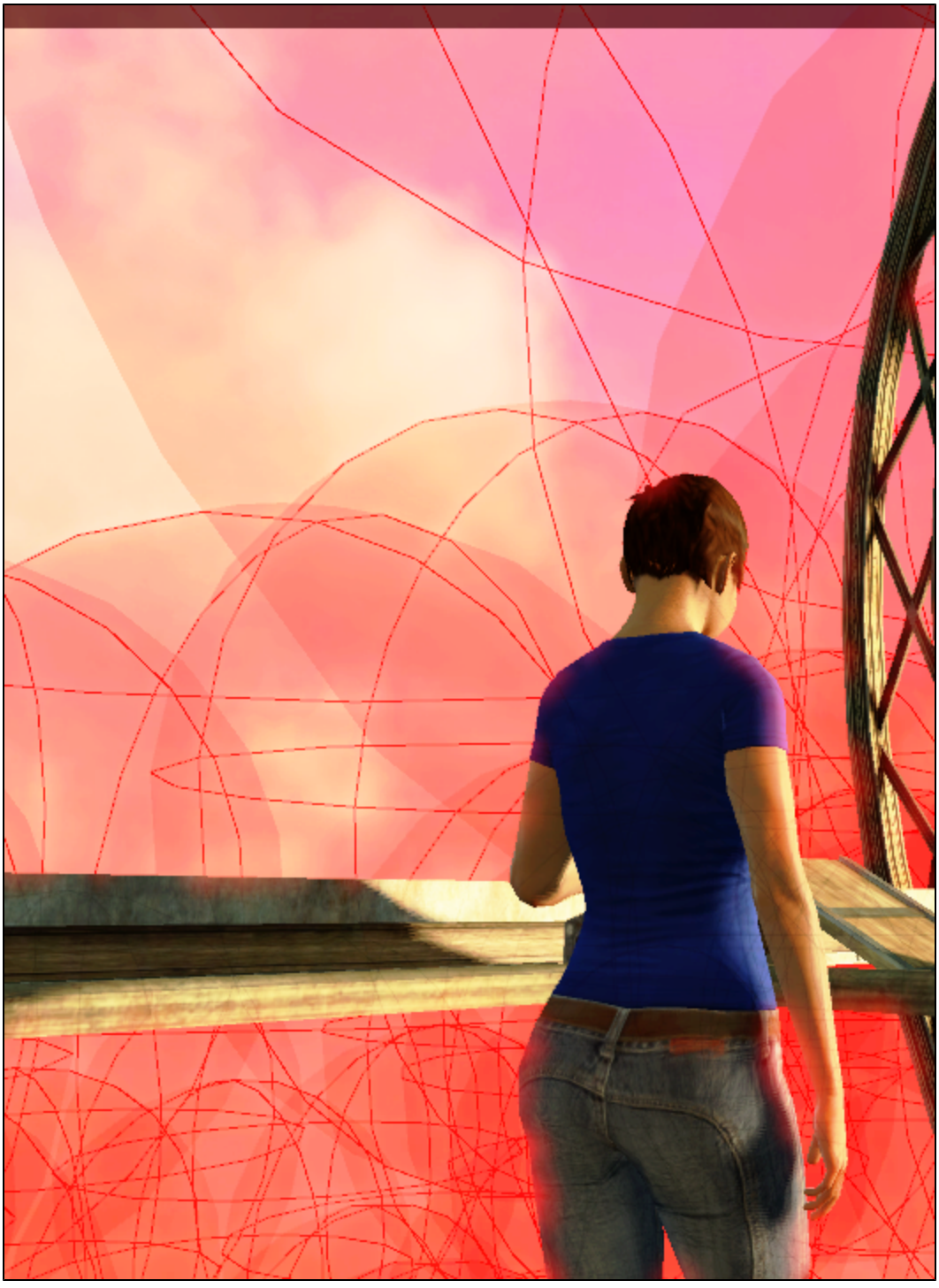
This command takes a boolean that enables/disables the updating of dynamic clouds.

Argument	Effect Explanation	Effect

1	Enables dynamic cloud updating.	
0	Disables dynamic cloud updating.	

atmos_RenderDebugData

This command takes a boolean that renders the dynamic cloud bounding boxes.



This image shows the particleCloud's bounding boxes.

Arguments	Effect
1	Enables dynamic cloud bounding boxes.
0	Disables dynamic cloud bounding boxes.

atmos_SpriteMipMapBias

This command is used for internal purposes only. It has no profiling effects.

atmos_stopTime

This command takes a boolean that stops time for the particleCloudDome.

Arguments	Effect
1	Starts time for particleCloudDome.
0	Stops time for particleCloudDome.

Sky Design UI Nodes

Home Sky UI Nodes

homeSkySun Node

This is the directional light object for your scene. The homeSkySun node controls where the sun is in the sky scene. When you change the direction, color and intensity of the sun, its effects on other nodes will update automatically.

Dynamic Shadows

You can specify which items cast dynamic shadows by setting the geometry on the items to override default behavior. You can also change the quality of the shadows cast by dynamic lights or the scene's sun.

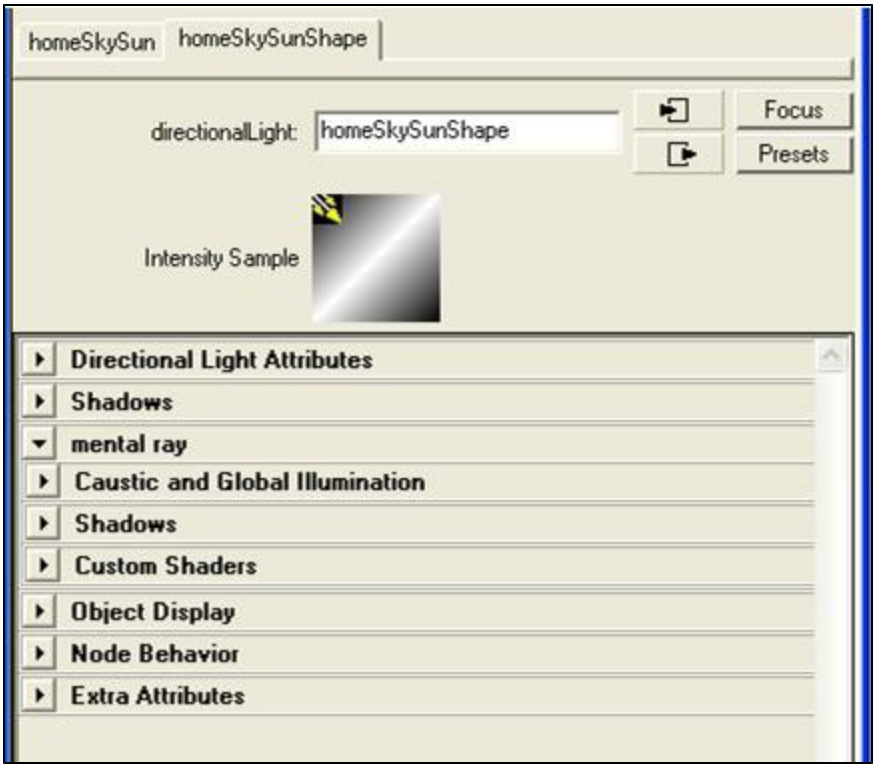
For information on dynamic shadows, see the Dynamic Shadows section in the Environment Lighting.

Creating a homeSkySun Node

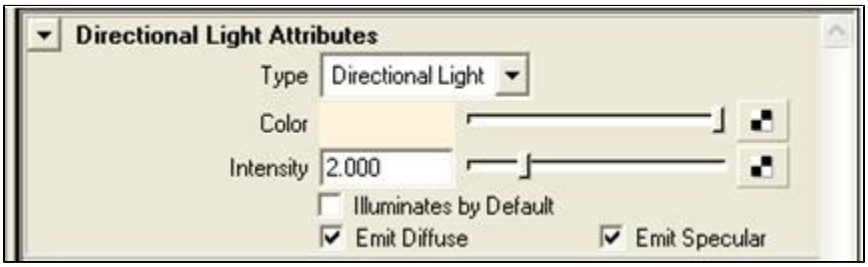
The first step to creating a sky is to press the Setup Scene button in the Home Sky UI, which will create the homeSkySun node. However, if you skip this first step, pressing any of the following buttons in the Home Sky UI or Home Cloud Modeler UI will create the homeSkySun node if it is not already present:

- Create Basic Sky
- Create Gradient Sky
- Create Sprite Cloud
- Create Exp Height Fog
- Create Scene Fog
- Create Particle Cloud Dome
- Create Cloud

Changing the homeSkySun Properties



Directional Light Attributes



The following properties are available:

- Type: Keep this setting as Directional Light.

- Intensity: Changes the intensity of the sun's color. The lower the value, the darker the color.
- Illuminates by Default: Ticking this box creates a default light tab.
- Emit Diffuse: This is a property for a standard Maya light and has no effect in PS Home.
- Emit Specular: This is a property for a standard Maya light and has no effect in PS Home.

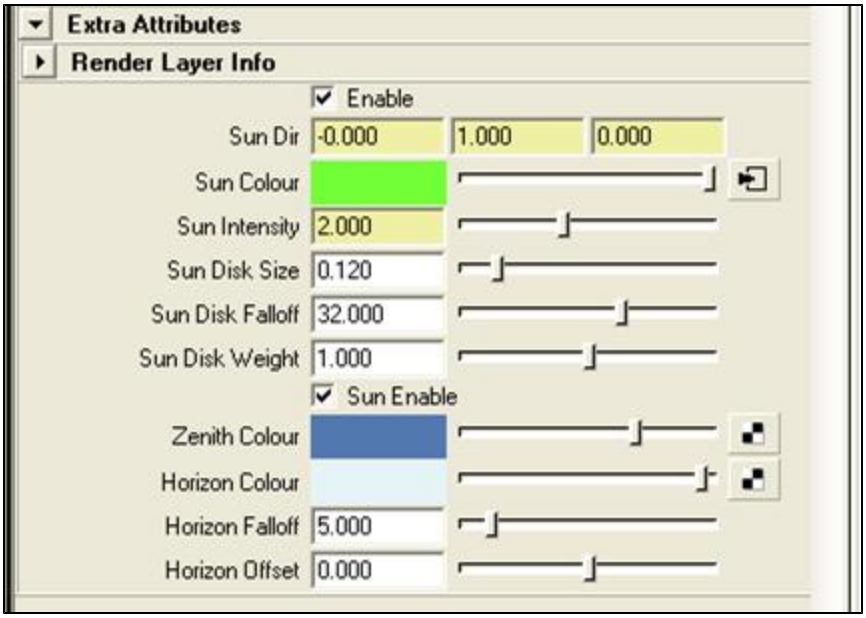
Basic Sky

The Basic Sky node creates the background sky, which is the gradual sky color that stretches from horizon to zenith and will render behind all other sky elements in the scene.

This sky background type is defined by two colors: a Zenith (top) color and a Horizon (bottom) color. It is the most basic form of sky and its use is encouraged for early stages in a sky modeling process.

Changing the Basic Sky Properties

After creating the Basic Sky node, select it in either the Home Sky UI or Maya outliner, and then go to the Attributes Editor > Render Layer Info.



Sun Properties

You cannot change most sun properties through the Basic Sky node. You can only edit the sun properties from the homeSkySun node, which is created when you select the Setup Scene, or create Sky or Fog nodes. These sun properties are present for reference only while designing your background sky colors.

Only the following property can be changed:

- Sun Enable: Enables/disables the homeSkySun node through.

Basic Sky Properties

The following properties are available:

- Enable: Enables/disables the basic sky.
- Zenith Colour: The zenith color is the color seen above the horizon. It is the sky color on top in a scene. Select the color box to change the color of the zenith, and move the slider to change the intensity of the color.
- Horizon Colour: The horizon color is the color below the zenith. It is the sky color on bottom in a scene. Select the color box to change the color of the horizon, and move the slider to change the intensity of the color.
- Horizon Falloff: The horizon falloff controls the blurring/melding between the horizon and zenith. The greater the value the sharper the contrast between zenith and horizon. At low values, the horizon blends into the zenith, and at very low values the horizon blends completely over the zenith.
- Horizon Offset: The horizon offset is the absolute height (y-axis) at which the horizon appears. Below is a table that explains the effects of the value.

Horizon Offset Value	Effect
0	Horizon is a straight line, seen at eye-level of character.
<0	Horizon is below eye-level. As the value decreases, the horizon becomes more concave and further below eye-level and the scene.
>0	Horizon is above eye-level. As the value increases, the horizon becomes more concave and further above eye-level and the scene.

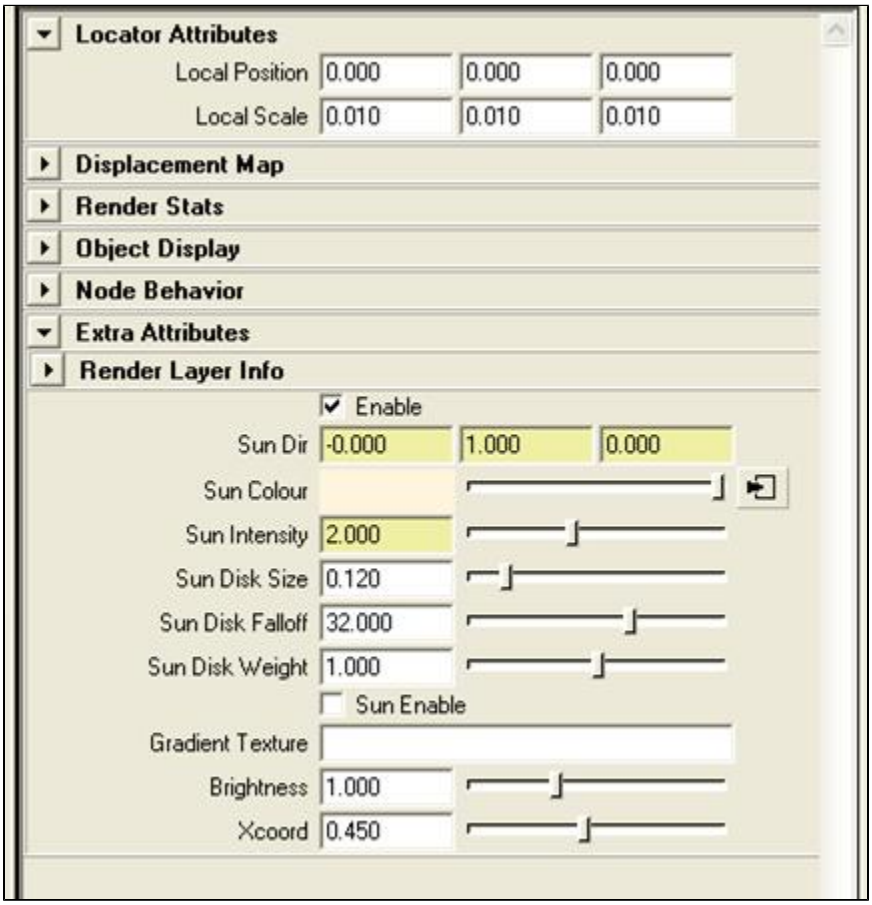
Gradient Sky

This is a background sky coloring node that procedurally shades the sky. It is defined by an HDR 1-dimensional gradient texture that provides the color gradient from the zenith to the bottom of the sky dome. A 2-dimensional color texture can be provided, where an x-coord parameter defines the position of the 1-dimensional slice that will be taken. A procedural sun is also rendered. The color and direction of the sun is driven by the homeSkySun node.



The texture should be in 4 channel HDR format. If HDR information is not needed, leave the alpha channel white.

Changing the Gradient Sky Properties



Sun Properties

You cannot change most sun properties through the Gradient Sky node. You can only edit the sun properties from the homeSkySun node, which is created when you select the Setup Scene, or create Sky or Fog nodes. These sun properties are present for reference only while designing your background sky colors.

Only the following property can be changed:

- Sun Enable: Enables/disables the homeSkySun node.

Gradient Sky Properties

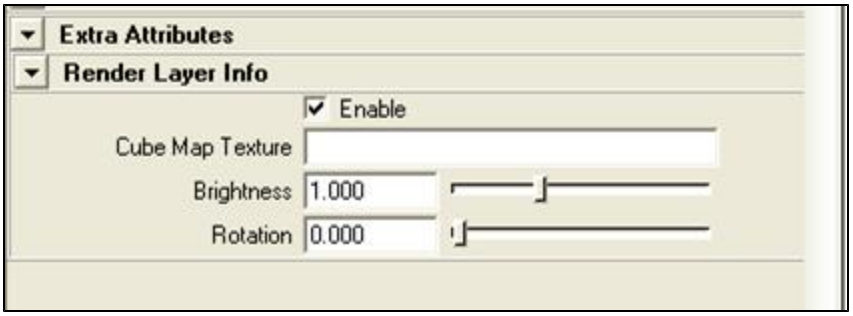
The following properties are available:

- Gradient Texture: Project-rooted texture path for the gradient texture. For example: `build\environments\atmospheres\skygradients\evening_gradient1.dds`.
- Brightness: Multiplier on the texture result.
- Coord: Defines the X (or U) position to be used when accessing the texture. The Y (V) position always ranges from 0.0 -> 1.0. (bottom to zenith).

Textured Sky

This is a node that uses an HDR cube map texture to define the background color of the sky.

Changing the Textured Sky Properties



The following properties are available:

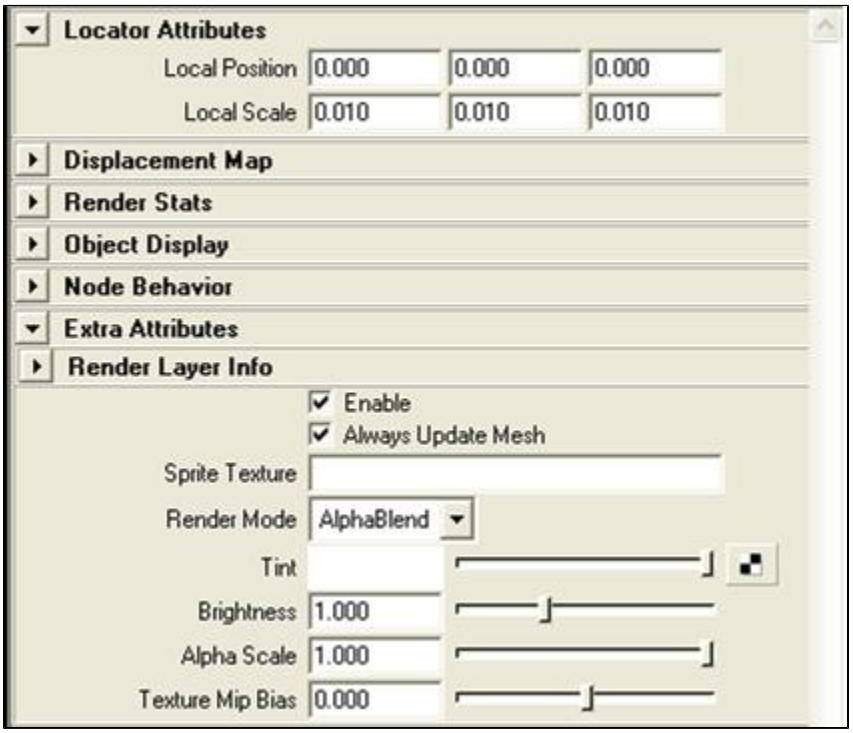
- Enable: Enables/disables the Textured Sky node.
- CubeMapTexture: Project-rooted texture path for the cube map texture. For example: `build\environments\Atmospheres\SkyBoxes\hdrsky.dds`.
- Brightness: Multiplier on the texture result.
- Rotation: Rotation around the Y axis.

Sprite Cloud

This node provides means to render an arbitrary texture in the sky. Use for things like high wispy clouds, moon, jet trails, etc. Sprite Cloud can render either an RGBA HDR opaque image (alpha channel represents HDR level) or an RGBA alpha blend image.

> Sprite mesh is rendered without depth test, to take care to avoid overlapping triangles in mesh.

Changing the Sprite Cloud Properties



The following properties are available:

- Enable: Enables/disables the Sprite Cloud node.
- Always Update Mesh: Enables/disables automatic update of cloud shape whenever mesh representation is changed.
- Sprite Texture: Path for the sprite texture. This needs to be located in your <HDK_INSTALL_DIR>\build folder.
- Render Mode: Opaque/AlphaBlend. Sets the render type of the texture.
- Tint: Applies a tint multiplier on the texture, which allows you to color the cloud without changing the texture.
- Brightness: Multiplier on texture.
- Alpha Scale: Multiplier on alpha component.
- Texture Mip Bias: Allows for sharpening or softening of the texture on the PlayStation®3.

ExpHeightFog

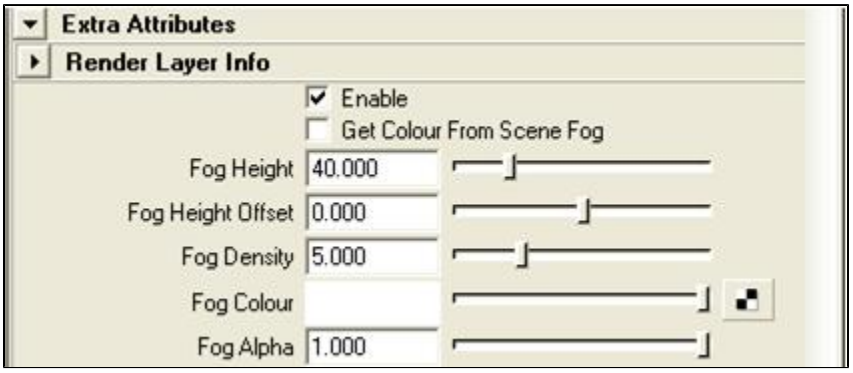
This node allows for rendering an exponential height fog effect on the sky. This means the fog will reduce with increasing altitude which gives the faint horizon-hugging haze that is seen in natural landscape images.



The ExpHeightFog node defines a fogging volume where density drops off exponentially with altitude. Fogging is the key to having a realistic horizon that appears to fade with distance, and also helps connect the sky to the main Home scene in a seamless manner.

The fog is rendered after sky background nodes and sprite clouds. The Particle Clouds will fog correctly with respect to distance and altitude. Some of these parameters may seem to give the same effect, but each is slightly different and you are encouraged to experiment and tweak each as you are trying to find the correct fog settings for the scene.

Changing the ExpHeightFog Properties

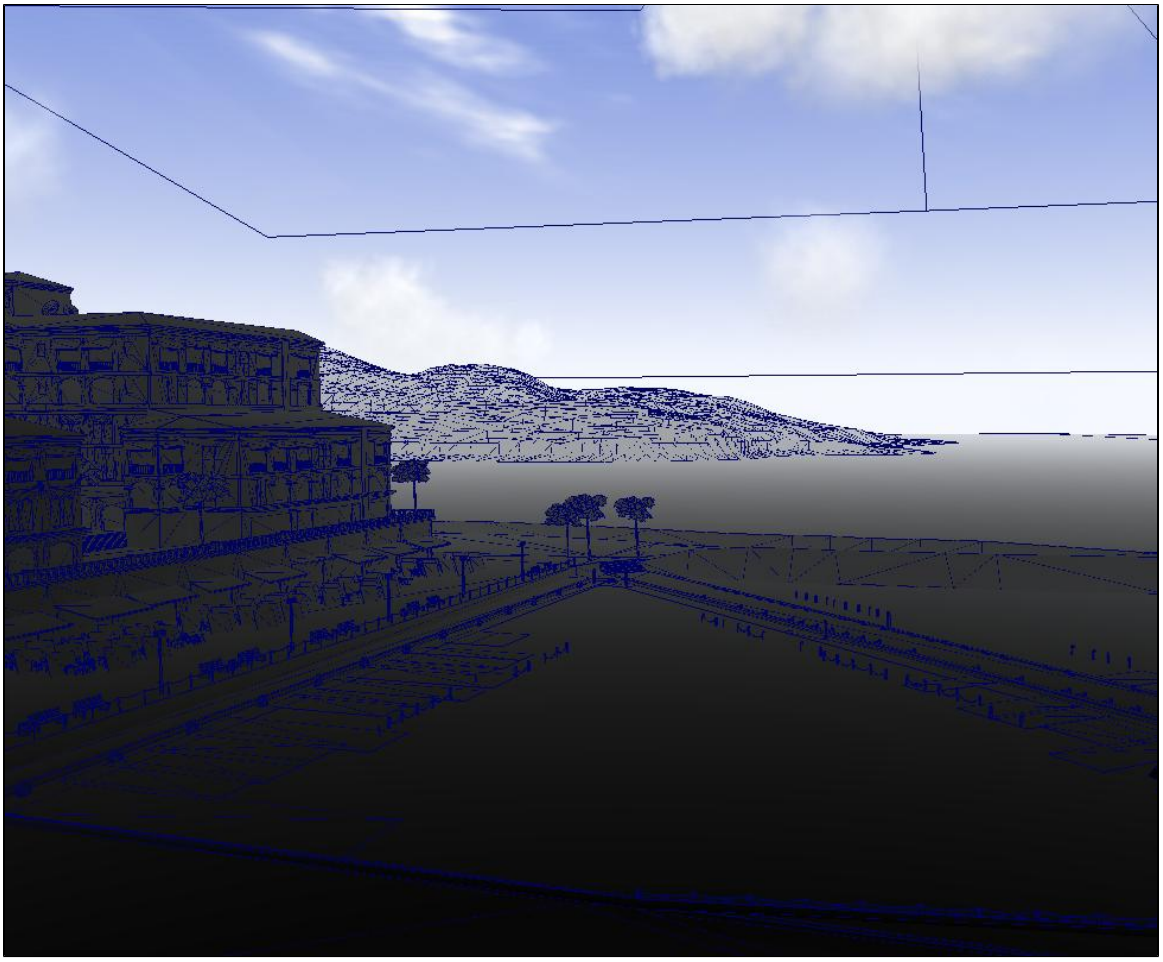


The following properties are available:

- Enable: Enables/disables the ExpHeightFog node.
- Get Colour From Scene Fog: Enables/disables the color value driven by the Scene Fog node.
- Fog Height: How quickly fog fades off with altitude.
- Fog Height Offset: Offset for altitude.
- Fog Density: Sets the thickness of the fog.
- Fog Colour: Color of the fog (can be driven by scene fog).
- Fog Alpha: Alpha value for fogging effect.

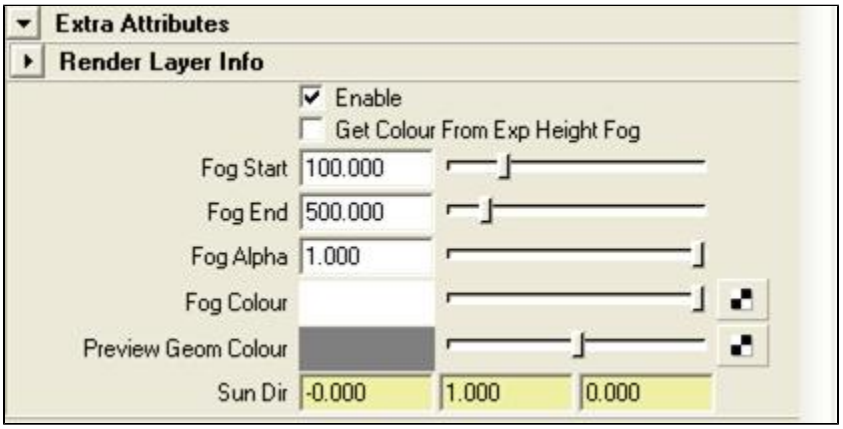
Scene Fog

The Scene Fog node defines fogging for the main Home geometry scene. This fog is a standard uniform density fog. If an AtgiLocator exists in the scene, then the node will render the geometry as solid and fogged for preview purposes.



SceneFog can increase render times for the Home scene. Make sure that you use this node only if the effect is truly needed.

Changing the Scene Fog Properties



The following properties are available:

- Enable: Enables/disables the Scene Fog node.

Sometimes you must toggle Enable/Disable to force an update of the AtgiLocator geometry.

- Get Colour From Exp Height Fog: Enables/disables the color value driven by the ExpHeightFog node.

- Fog Start: The distance from camera where the fog effect will start.
- Fog End: This distance from camera where geometry will be fully fogged.
- Fog Alpha: Alpha value for fogging effect.
- Fog Colour: Color of the fog (can be driven by the ExpHeightFog node).
- Preview Geom Colour: Color of preview AtgiLocator geometry.
- Sun Dir: Sun direction used for shading of AtgiLocator geometry (can only be altered in the homeSkySun node).

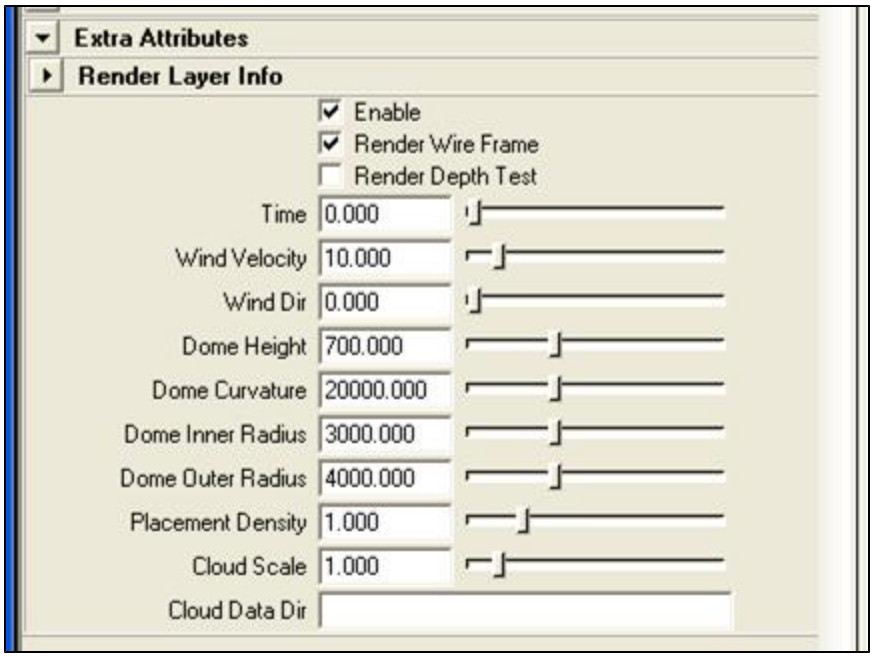
ParticleCloudDome

The ParticleCloudDome node provides a means to scatter a group of pre-modeled particle clouds throughout the sky and have them fade and move in a natural arc across the sky.

A particle cloud dome references a cloud library xml file as saved by the Cloud Modeling UI. It randomly scatters copies of the clouds throughout the sky in the shape of a dome centered at the origin.

> Only one particle cloud dome can exist in a sky scene.

Changing the ParticleCloudDome Properties



The following properties are available:

- Enable: Enables/disables the ParticleCloudDome node.
- Render Wire Frame: Enables/disables wireframe rendering. Wireframe rendering allows visualization of the structure of the dome.
- Render Depth Test: Enables/disables depth test rendering within Maya (PlayStation®3 always renders without depth test).
- Time: Time slider to preview the speed in which clouds will move when in PS Home (ignored at export).
- Wind Velocity: Speed in which clouds move when in PS Home.
- Wind Dir: Direction in which clouds move when in PS Home.
- Dome Height: Height in meters of the apex of the dome.
- Dome Curvature: Control how quickly the dome falls/curves as it approaches the horizon.
- Dome Inner Radius: The distance in which clouds will start fading.
- Dome Outer Radius: The distance in which clouds will be completely faded.
- Placement Density: How dense is the scattering of clouds.
- Cloud Scale: Size scalar for clouds.
- Cloud Data Dir: Path to cloud lib xml file (should point to artsource location, ignored for export).

Home Cloud Modeler UI Nodes

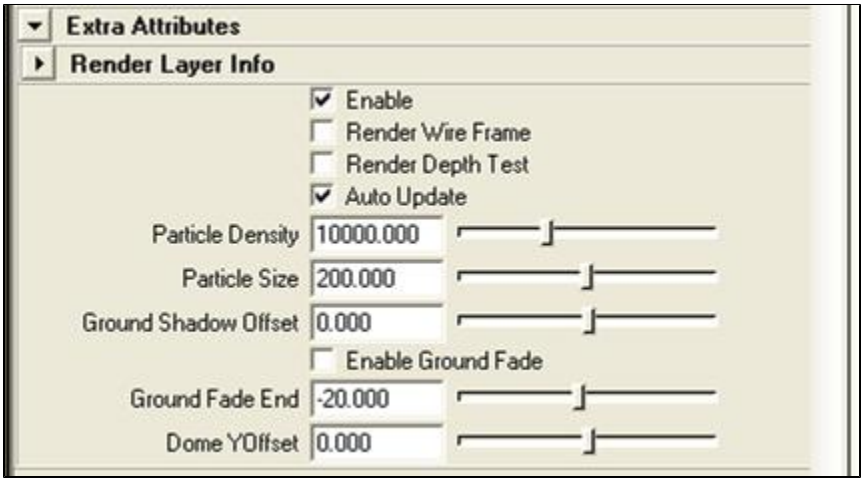
Particle Cloud

A particle cloud is a Cumulus-style fluffy cloud, which is made up of many particle sprites. Each ParticleCloud node has two nodes that appear in the Cloud Objects box, ParticleCloud_ObjShape and ParticleCloudSphere.

When a ParticleCloud node is created, a particleCloudShadingParams node is also created. This node defines the physical shape of the cloud.

- You can have multiple ParticleClouds nodes per sky.
- Each ParticleCloud can have multiple ParticleCloudSpheres.
- There can only be one ParticleCloudShadingParams node per sky.

Changing the ParticleCloud_ObjShape Properties



The following properties are available:

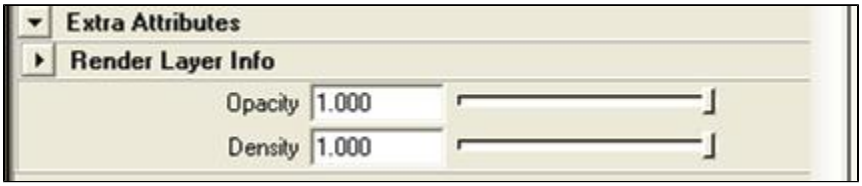
- Enable: Enables/disables the node.
- Render Wire Frame: Enables/disables wireframe rendering. Wireframe render allows visualization of the particles that make up a cloud.
- Render Depth Test: Enables/disables depth test rendering within Maya.

> PlayStation®3 always renders without depth test.

- Auto Update: Enables/disables automatic repopulation of the cloud whenever a parameter change is made.
- Particle Density: Controls how many particles are created within the cloud. The number is a distance function and is not related to how many particles are generated. For a readout of the number of particles generated, watch the console output.
- Particle Size: The size of each particle sprite in meters.
- Ground Shadow Offset: A ground shadow is where the bottom half of the cloud is darkened for shading. This allows tweaking of where the midpoint of the cloud is positioned for this effect.
- Enable Ground Fade: Enables/disables the ground fade effect. This effect flattens the bottom of the cloud.
- Ground Fade End: Defines the position of the fade.
- Dome YOffset: Defines where the midpoint of the cloud is for ParticleCloudDome positioning (this can be visualized if the Render Wire Frame box is checked).

Changing the ParticleCloudSphere Properties

A particle cloud sphere is a simple wireframe sphere which, when combined with others, defines the overall shape of a cloud. Each provides parameters which allows localized tweaking of cloud settings.



The following properties are available:

- Opacity: Scales the ParticleCloud opacity setting for contained particles.
- Density: Scales the ParticleCloud density setting for contained particles.

ParticleCloudShadingParams

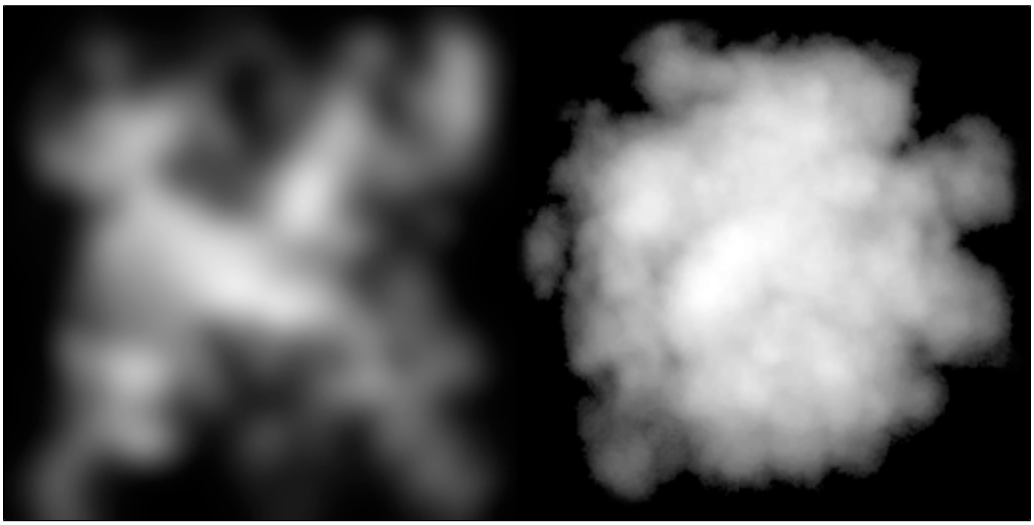
This is the node that controls the shading of all particle clouds and particle cloud domes in the scene. It references the particle texture and defines all lighting and parameters.

particleCloudShadingParams .dds Texture

The particle texture used for cloud rendering is of a special format. It is in RGBA format. It holds 2 grayscale particle sprite images. One in the Green channel and the other in the Alpha channel.

> ⚠ The particle cloud texture needs to be specified, otherwise a checkerboard texture is displayed.

Below is an example, with both sprites side by side. For each rendering of a particle, a random blend of each texture is used. This helps break up the repetition that would otherwise be seen.



If the particle clouds do not display the correct texture, the link to the texture to be used in the tool will need to be assigned. The Sprite texture location should be:

```
build\environments\Atmospheres\CloudParticleTextures\cloud_particle.dds
```

Example Textures

Example particle textures are located here:

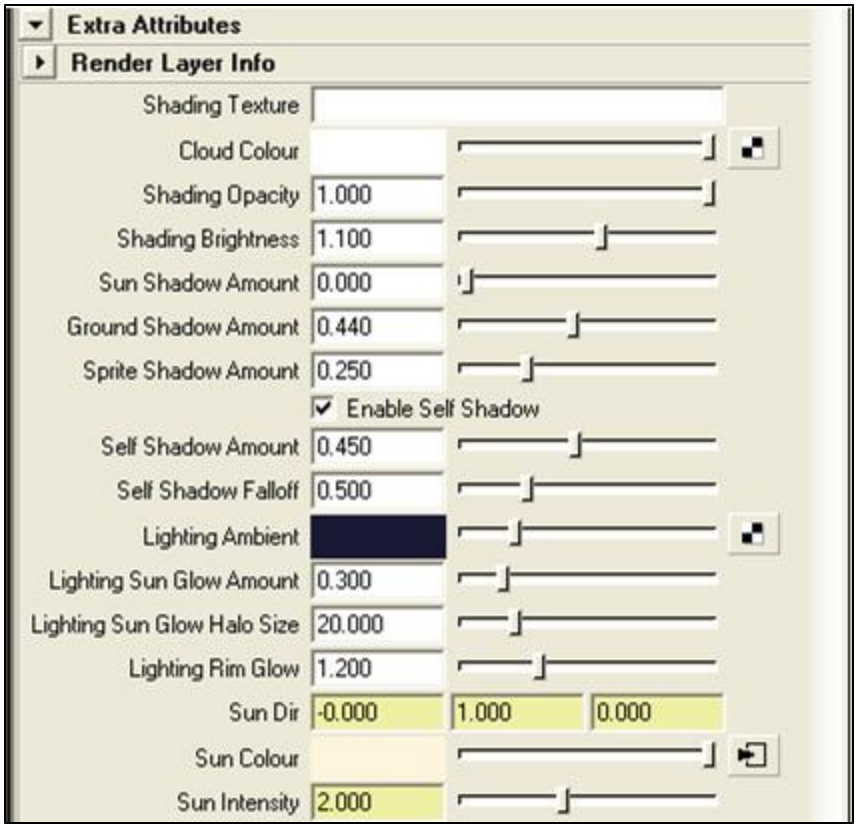
```
artsource\Environments\Atmospheres\SourceTextures\cloud_particle.tga
build\environments\Atmospheres\CloudParticleTextures\cloud_particle.dds
```

Self Shadowing

One quirk with the cloud shading is the idea of self shadowing. This allows the cloud to shadow itself in a realistic manner, but doubles the size of the final PlayStation®3 cloud data. Self shadowing is baked and cannot change at runtime.

If memory size is important, or if you intend to change sun direction in PlayStation®3(For example as part of a Lua Script), then disable self shadowing and use sun shadowing instead. At all other times use self shadowing with no sun shadowing.

Changing the Particle Cloud Shading Parameter Properties



The following properties are available:

- Shading Texture: The particle sprite texture path.
- Cloud Colour: The base color of the cloud.
- Shading Opacity: Alpha value for cloud. (0.0 = transparent, 1.0 = opaque).
- Shading Brightness: Multiplier on the cloud color.
- Sun Shadow Amount: Darkens the side of the cloud that is facing away from the sun. Assumes cloud is shape of a sphere (very crude, best left at 0.0, self shadow is better).
- Ground Shadow Amount: Darkens the bottom side of the cloud.
- Sprite Shadow Amount: Darkens the edges of the sprite shape to give the cloud a more detailed appearance. Best left at low values, for example, 0.25.
- Enable Self Shadow: Enables/disables self shadow (self shadow looks better than sun shadow, but increases memory usage, and cannot change at runtime).
- Self Shadow Amount: Darkens the cloud as if is shadowing itself.
- Self Shadow Falloff: How quick the shadow darkens as light passes through the cloud.
- Lighting Ambient: Ambient/constant color term.
- Lighting Sun Glow Amount: Controls how much the cloud lightens when in line with the sun.
- Lighting Sun Glow Halo Size: The size/angle in which the cloud lightens when in line with the sun.
- Lighting Rim Glow: Lightens the edges of the cloud when in line with the sun (note: only visible in PS Home, not visible in Maya).
- Sun Dir: Direction of sun glow.
- Sun Colour: Color of sun glow.
- Sun Intensity: Intensity (color multiplier) of sun glow.

Atmosphere Lua Reference

You can change some of the parameters in a Home atmosphere scene via Lua. You can edit existing parameters in Lua but not create nor destroy a sky scene. You must build a scene in Maya first after which you can then use Lua script to change the parameters in the scene. You use the scene Lua library Atmos to edit scenes in Lua. The following commands are available:

```
Atmos.GetParam()
Atmos.SetParam()
```

The syntax is:

```
boolean\number\string\Vector4 Atmos.GetParam( string paramName)
boolean\number\string\Vector4 Atmos.GetParam( string paramName, number index)
boolean Atmos.SetParam( string paramName, boolean\number\string\Vector4 paramVal)
boolean Atmos.SetParam( string paramName, number index, boolean\number\string\Vector4 paramVal)
```

Examples:

```
Atmos.SetParam( "BASICSKY_HORIZONCOLOUR", horizonColour);
Atmos.SetParam( "EXPHEIGHTFOG_FOGDENSITY", fogDensity );
Atmos.SetParam( "SPRITECLOUD_TINTCOLOUR", 4, s\[roteCloudTint);
horizonColour= Atmos.GetParam("BASICSKY_HORIZONCOLOUR");
numSpriteClouds = Atmos.GetParam("NUM_SPRITECLOUDS");
spriteCloudTint = Atmos.GetParam("SPRITECLOUD_TINTCOLOUR", 4 );
```

horizonColour is a Vector4

fogDensity is a number

spriteCloudTint is a Vector4

Reference

Basic Sky Variables

Basic Sky Variable	Type	Description
BASICSKY_SUN_INTENSITY	float	Intensity (color multiplier) of procedural sun
BASICSKY_SUN_DIR	Vec4	Normalized direction vector of procedural sun
BASICSKY_SUN_COLOUR	Vec4	Color of procedural sun
BASICSKY_SUN_DISKSIZE	float	Size of sun disk
BASICSKY_SUN_DISKFALLOFF	float	Falloff/size of the sun halo
BASICSKY_SUN_DISKWEIGHT	float	Alpha value for the sun effect
BASICSKY_SUN_ENABLE	boolean	Enable/disable sun effect
BASICSKY_ZENITHCOLOUR	Vec4	Color of the sky at the zenith
BASICSKY_HORIZONCOLOUR	Vec4	Color of the sky at the horizon
BASICSKY_HORIZONFALLOFF	float	Controls the sharpness of the falloff rate between the zenith to horizon color
BASICSKY_HORIZONOFFSET	float	Offset to allow tweaking position of horizon

Textured Sky

Textured Sky	Type	Description
TEXTUREDSKY_ROTATION	float	Rotation around the Y-axis
TEXTUREDSKY_BRIGHTNESS	float	Multiplier on the texture result

Gradient Sky

Gradient Sky	Type	Description
GRADIENTSKY_SUN_INTENSITY	float	Intensity (color multiplier) of procedural sun
GRADIENTSKY_SUN_DIR	Vec4	Normalized direction vector of procedural sun
GRADIENTSKY_SUN_COLOUR	Vec4	Color of procedural sun
GRADIENTSKY_SUN_DISKSIZE	float	Size of sun disk

GRADIENTSKY_SUN_DISKFALLOFF	float	Falloff/size of sun halo
GRADIENTSKY_SUN_DISKWEIGHT	floato	Alpha value for the sun effect
GRADIENTSKY_SUN_ENABLE	boolean	Enable/disable sun effect
GRADIENTSKY_BRIGHTNESS	float	Multiplier on the texture result
GRADIENTSKY_XCOORD	float	Defines the X (or U) position to be used when accessing the texture. The Y (or V) position always ranges from 0.0 - 1.0 (bottom to zenith)

Sky Scene Settings

Sky Scene Settings	Type	Description
SKYSCENESETTINGS_USEGAMESUNDIR	boolean	Allow the game (nova) sun direction to override the sun direction
SKYSCENESETTINGS_CLOUDBUFFERSIZE	float	The resolution of the cloud rendering (lower is faster). The range is 0.0 - 1.0

Exponential Height Fog

Exponential Height Fog	Type	Description
EXPHEIGHTFOG_FOGDENSITY	float	Sets thickness of fog
EXPHEIGHTFOG_FOGHEIGHT	float	Sets how quickly fog fades off with altitude
EXPHEIGHTFOG_FOGHEIGHTOFFSET	float	Offset for altitude
EXPHEIGHTFOG_FOGALPHA	float	Alpha value for fogging effect
EXPHEIGHTFOG_FOGCOLOUR	Vec4	Color of the fog (can be driven by scene fog).

Scene Fog

Scene Fog	Type	Description
SCENEFOG_ENABLE	boolean	Enable/disable scene fog effect
SCENEFOG_FOGSTART	float	The distance from camera where the fog effect will start
SCENEFOG_FOGEND	float	The distance from camera where geometry will be fully fogged
SCENEFOG_FOGALPHA	float	Alpha value for fogging effect
SCENEFOG_FOGCOLOUR	Vec4	Color of the fog

SCENEFOG_FOGCOLOR only accepts the first three Vec4 parameters. The fourth should not be used.

Particle Cloud Dome

Particle Cloud Dome	Type	Description
PARTICLECLOUDDONE_DOMEHEIGHT	float	Height in meters of the apex of the dome
PARTICLECLOUDDOME_DOMECURVATURE	float	Control how quickly the dome falls/curves as it approaches the horizon
PARTICLECLOUDDOME_DOMEINNERRADIUS	float	The distance in which clouds will start fading
PARTICLECLOUDDOME_WINDVELOCITY	float	Speed at which clouds move when in PS Home
PARTICLECLOUDDOME_WINDDIR	float	Direction in which clouds move when in PS Home

Particle Cloud Shading Parameters

Particle Cloud Shading Parameters	Type	Description

PARTICLECLOUDSHADING_CLOUDCOLOUR	Vec4	The base color of the clouds
PARTICLECLOUDSHADING_BRIGHTNESS	float	Multiplier on the cloud color
PARTICLECLOUDSHADING_SUNINTENSITY	float	Intensity (color multiplier) of procedural sun
PARTICLECLOUDSHADING_SUNDIR	Vec4	Normalized direction vector of procedural sun
PARTICLECLOUDSHADING_SUNCOLOUR	Vec4	Color of procedural sun
PARTICLECLOUDSHADING_SUNGLOWAMOUNT	float	Controls how much the cloud lightens with in line with the sun
PARTCILECLOUDSHADING_SUNGLOWHALOSIZE	float	The size/angle at which the cloud lightens when in line with the sun
PARTICLECLOUDSHADING_RIMGLOWAMOUNT	float	Lightens the edges of the cloud when in line with the sun
PARTICLECLOUDSHADING_AMBIENTCOLOUR	Vec4	Ambient/constant color term

Particle Cloud

Particle Cloud	Type	Description
PARTICLECLOUD_SHADINGOPACITY	float	Alpha value for cloud 0.0=transparent 1.0=opaque
PARTICLECLOUD_BIRGHTNESS	float	Multiplier on the cloud color
PARTICLECLOUD_SUNSHADOWAMOUNT	float	Darkens the side of the cloud that is facing away from the sun. It assumes the cloud is a the shape of a sphere
PARTICLECLOUD_SPRITESHADOWAMOUNT	float	Darkens the edges of the sprite shape to give the cloud a more detailed appearance
PARTICLECLOUD_SELFSHADOWAMOUNT	float	Darkens the cloud as if it is shadowing itself. (NOTE: has effect only if self shadow is enabled on clouds in skytool)
PARTICLECLOUD_GROUNDSHADOWAMOUNT	float	A ground shadow is where the bottom half of the cloud is darkened for shading
PARTICLECLOUD_GROUNDSHADOWOFFSET	float	This allows setting of the cloud midpoint used for the groundshadow effect

Obtain number of clouds in scene by querying NUM_PARTICLECLOUDS.

Sprite Cloud

Sprite Cloud	Variable Type	Description
SPRITECLOUD_TINTCOLOUR	Vec4	Tint multiplier on texture (allows for coloring of cloud without changing texture)
SPRITECLOUD_ALPHASCALE	float	Multiplier on alpha component
SPRITECLOUD_BRIGHTNESS	float	Multiplier on color component

Obtain number of clouds in scene by querying NUM_SPRITECLOUDS parameter.

Sun Parameters

Sun Parameters	Variable Type	Description
SUN_INTENSITY	float	Color of procedural sun (applies to all SUN_INTENSITY parameters in scene)
SUN_DIR	Vec4	Normalized direction vector of procedural sun (applies to all SUN_DIR parameters in scene)
SUN_COLOUR	Vec4	Color of procedural sun (applies to al SUN_COLOUR parameters in scene)

These apply the value to all instances of each parameter in the scene; for example, changing SUN_DIR here will apply it on all GradientSky, BasicSky and ParticleCloudShadingParams nodes.

Read-Only Parameters

Read-Only Parameters	Variable Type	Description
NUM_PARTICLECLOUDS	float	Number of particle clouds in the scene
MUM_SPRITECLOUDS	float	Number of sprite clouds in scene

Sky Design Quick Start Tutorial



This tutorial demonstrates how to make a typical sky scene for a Home environment. It makes use of example resources already provided in the HDK. The result will be final .atmos and accompanying .cdata files which can be either viewed directly using the HDK browser, or linked into a scene via the Scene Editor to provide background sky.

The examples below are included by default in the HDK, and do not need to be downloaded separately.

A completed tutorial is located in the HDK at:

```
artsource\Environments\Atmospheres\ExampleMayaScenes\BlueSky_WhitePuffyClouds.ma
build\environments\Atmospheres\BlueSky_WhitePuffyClouds.atmos
build\environments\Atmospheres\BlueSky_WhitePuffyClouds.cdata
```

This documentation also uses assets from the HTS Aeronautilus: a fully-featured developer space. Download HTS Aeronautilus from https://home.scedev.net/project/samples. The HTS Aeronautilus should be extracted to your <HDK_INSTALL_DIR> so that you can access the assets with the HDK.

You can export and load the results in PS Home reloading the scene. After step 4 of this tutorial the sky scene can be loaded by the PlayStation®3.

Cloud modeling is not essential for this section because an example set of pre-modeled clouds have already been provided for use in the HDK at this location: artsource\Environments\Atmospheres\CloudLibs

This tutorial covers the following tasks:

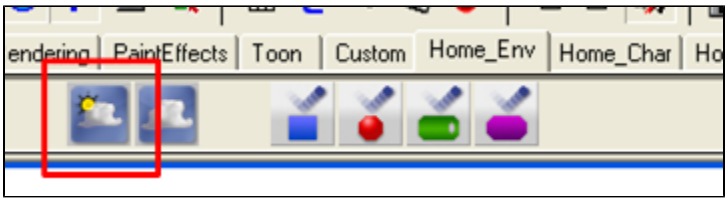
1. Initializing a Home Sky Scene
2. Creating Background Sky Color
3. Creating High Wispy Clouds
4. Creating Particle Cloud Dome

5. Creating Sky Fog
6. Creating Scene Fog
7. Exporting and Viewing the Sky on the PlayStation®3

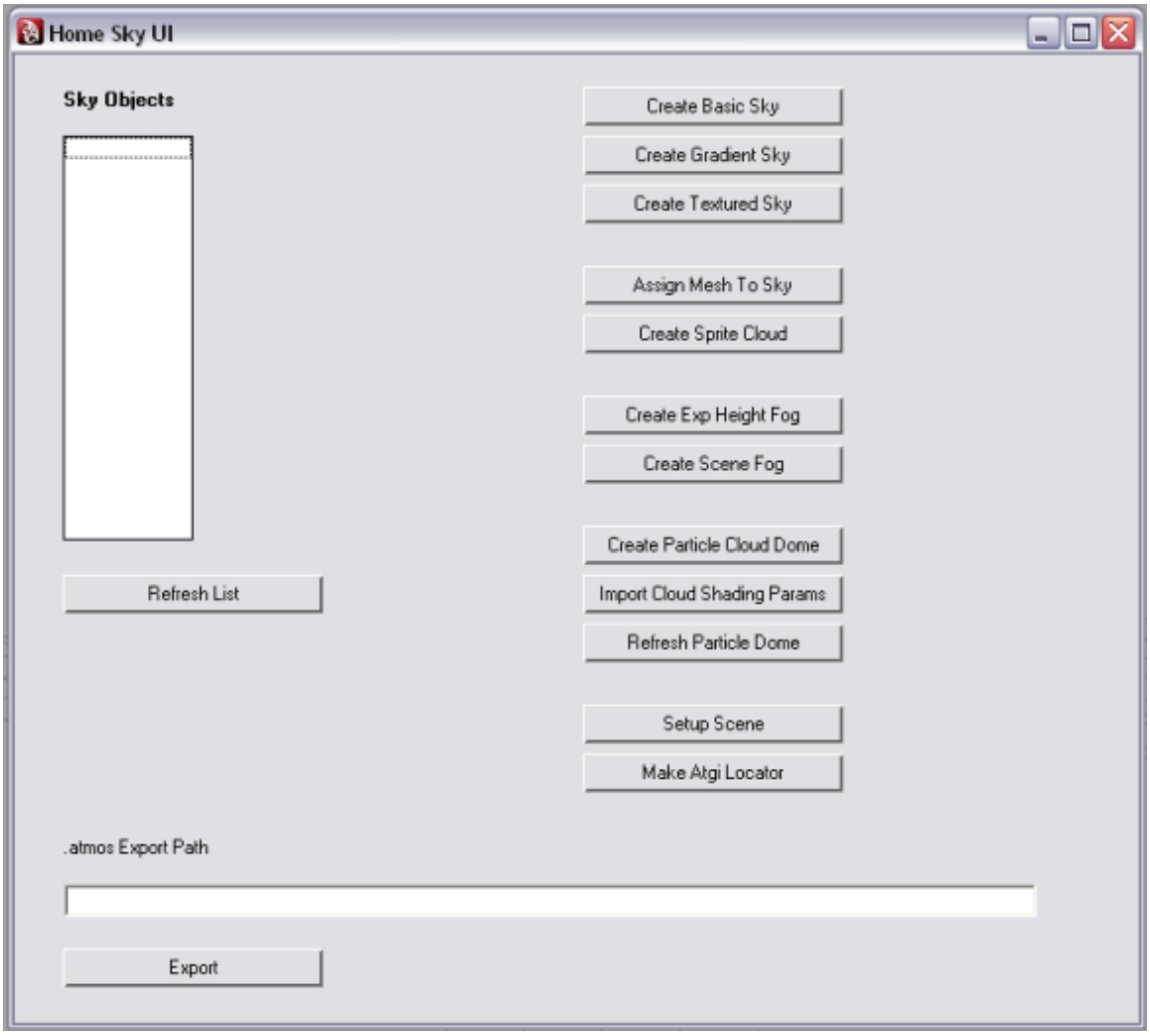
Initializing a Home Sky Scene

To initialize a Home Sky scene:

1. Open Maya.
2. Select File > New Scene, navigate to the Home_Env tool shelf and select the Launch Home Sky Tool button.

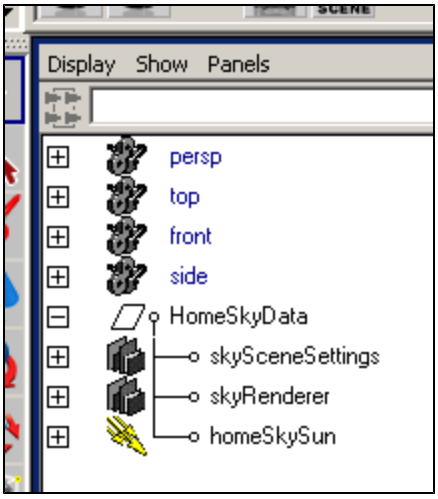


The Home Sky UI dialog is displayed:



3. Click the Setup Scene button to initialize the empty Maya scene to be a Home Sky scene.

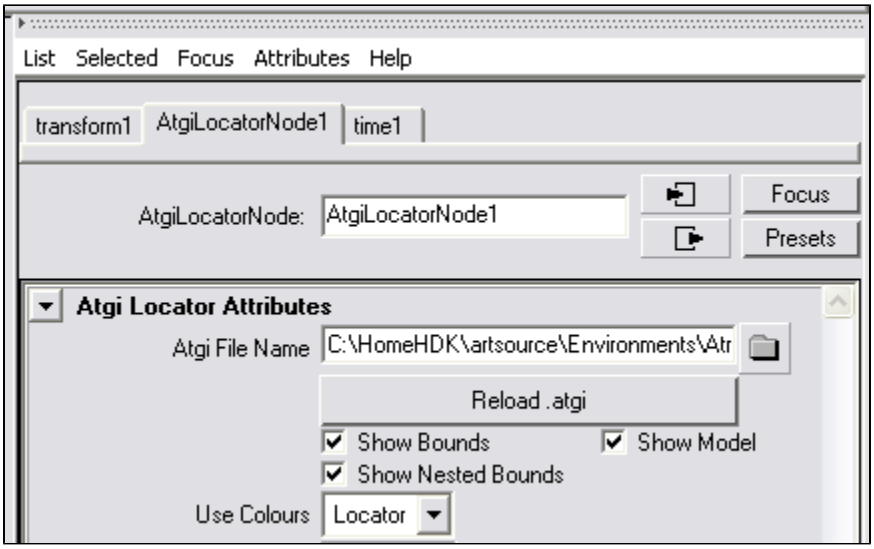
   This creates a new Group in the Maya Outliner called HomeSkyData. The group contains three nodes:



- skySceneSettings: This node contains parameters which effect the rendering of the sky scene.
- skyRenderer: This node is necessary for rendering of the sky elements in Maya, and is ignored during export.
- homeSkySun: This is a Maya directional light. It is used to define the direction, color and intensity of sunlight used by the various sky nodes.

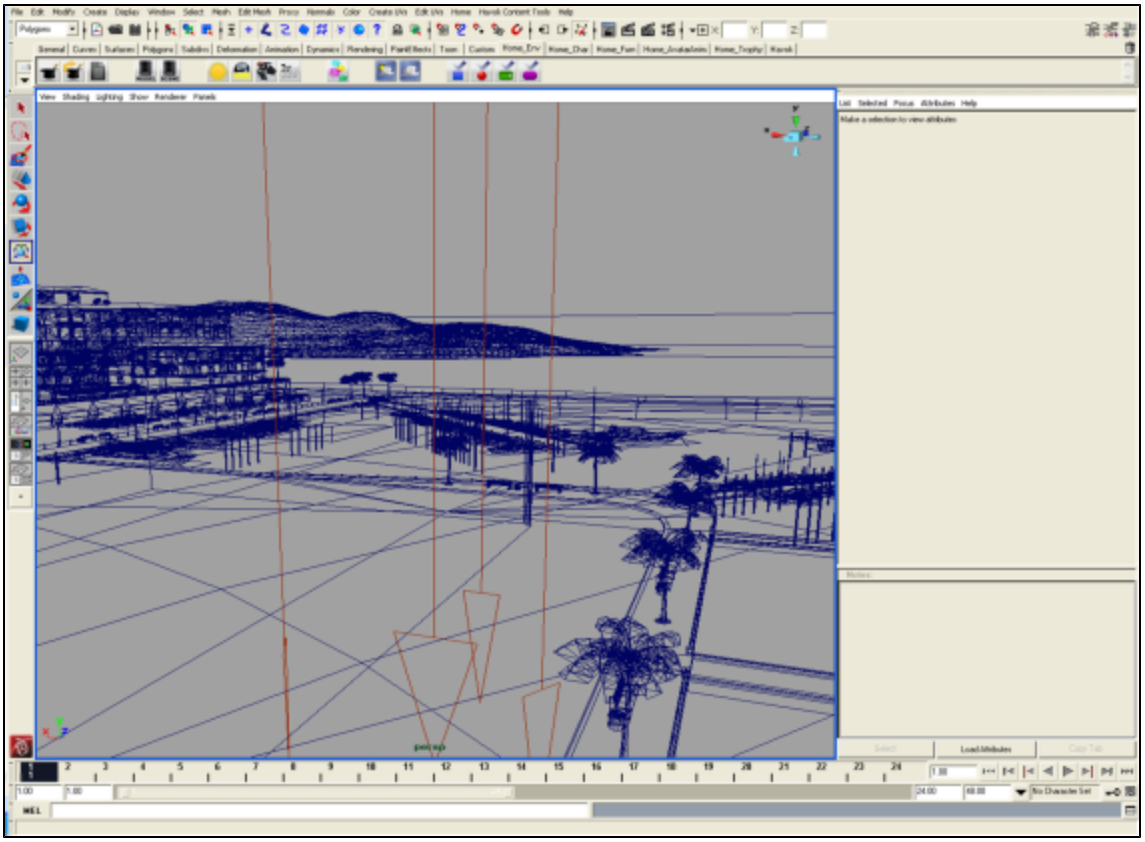
4. Click the Make AtgiLocator button to create an AtgiLocator node to render the destination Home environment geometry in Maya as wireframe. In the Maya Outliner there should now be a node called transform1. This is the new AtgiLocator node.
5. Select the transform1 node and view its attributes in the Attribute Editor.

In Attribute Editor > AtgiFileName, the file name must point to an .atgi file. The examples provided with the HDK Sky Tool make use of a sample .atgi file located in `artsource\Environments\Atmospheres\ExampleMayaScenes\Example_ATGI_Scene.atgi`



Use this file if you are currently not working on a Home environment. If you are working on a Home environment, you can find the .atgi file in the intermediate directory. For example:
`c:\HDK\Intermediate\Environments\My_Scene\My_Scene.atgi`

You should now see a wireframe rendering of the Home environment in Maya. We are now ready to start building the Sky.



Next: Creating Background Sky Color

Creating Background Sky Color

The background coloring of the sky is the gradual sky color that stretches from horizon to zenith and renders behind all other sky elements in the scene.

To create a BasicSky node:

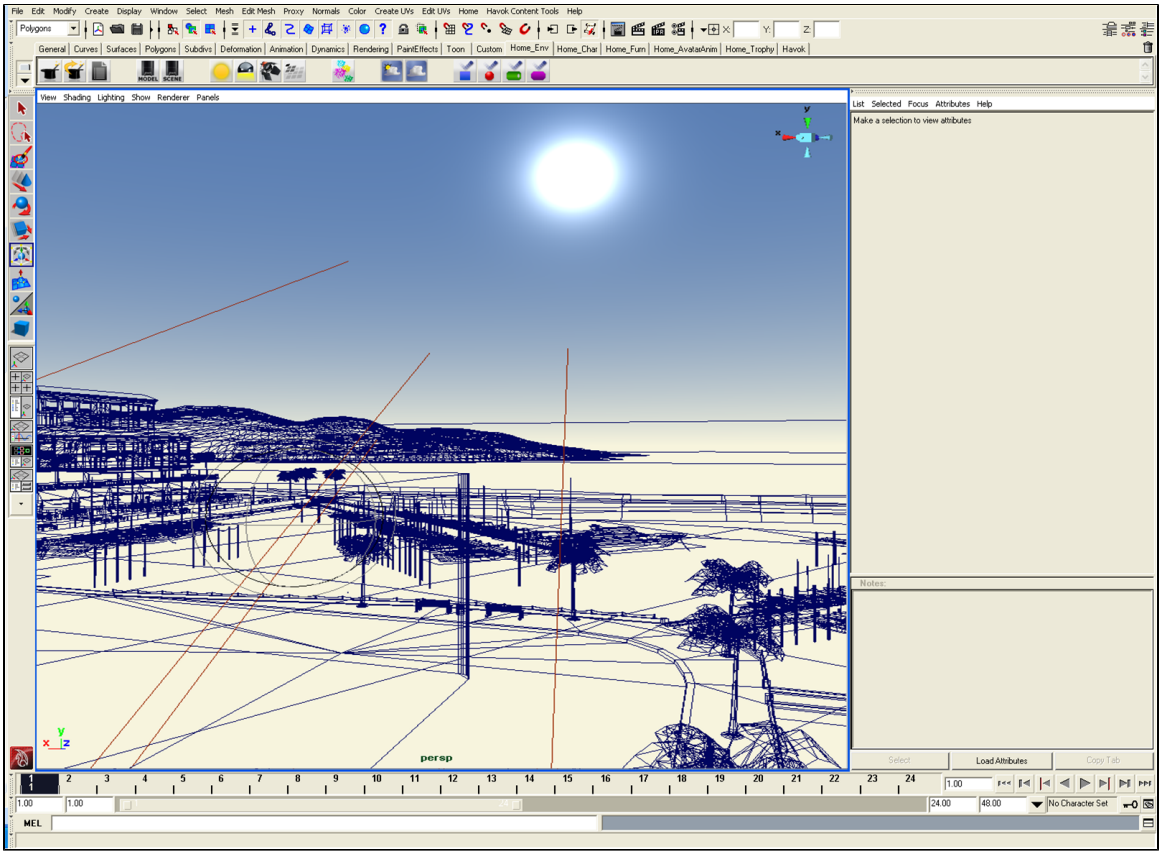
1. Clicking Create Basic Sky button in the Home Sky UI.

   A BasicSky node is visible in the Outliner and in the Home Sky UI, and the sky scene is colorized.

2. Select the BasicSky node in the HomeSkyUI to view its attributes in the Attribute Editor.

   It colors the sky by defining a zenith color and a horizon color, and interpolates between them using a falloff ramp value.

   The BasicSky node also renders a sun which can be tweaked in the Attributes Editor.



3. To adjust the sun, select the homeSkySun node and go to the Attributes Editor.

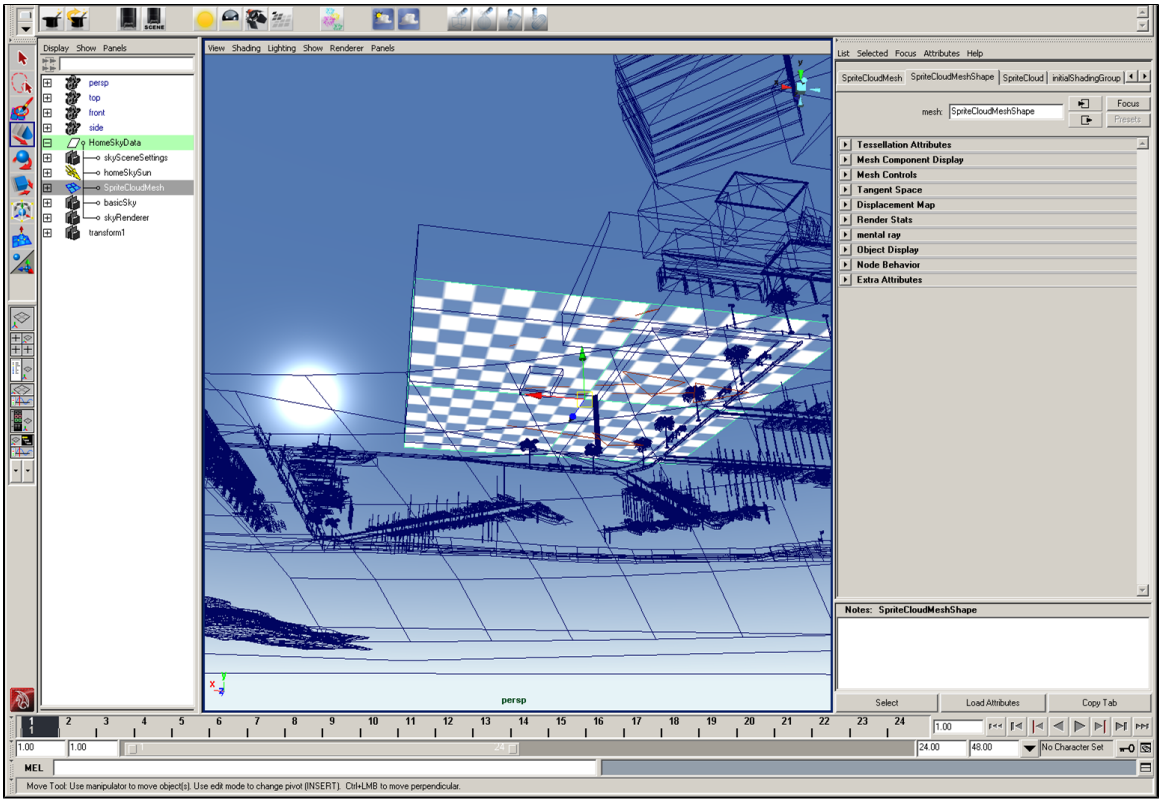
Next: Creating High Wispy Clouds

Creating High Wispy Clouds

To create high wispy clouds, use the Sprite Cloud node. A Sprite Cloud allows us to render a basic textured mesh in a sky scene.

1. Click the CreateSpriteCloud button in the Home Sky UI. This creates a new SpriteCloudMesh object which appears in the Sky Object list and in the Maya Outliner.

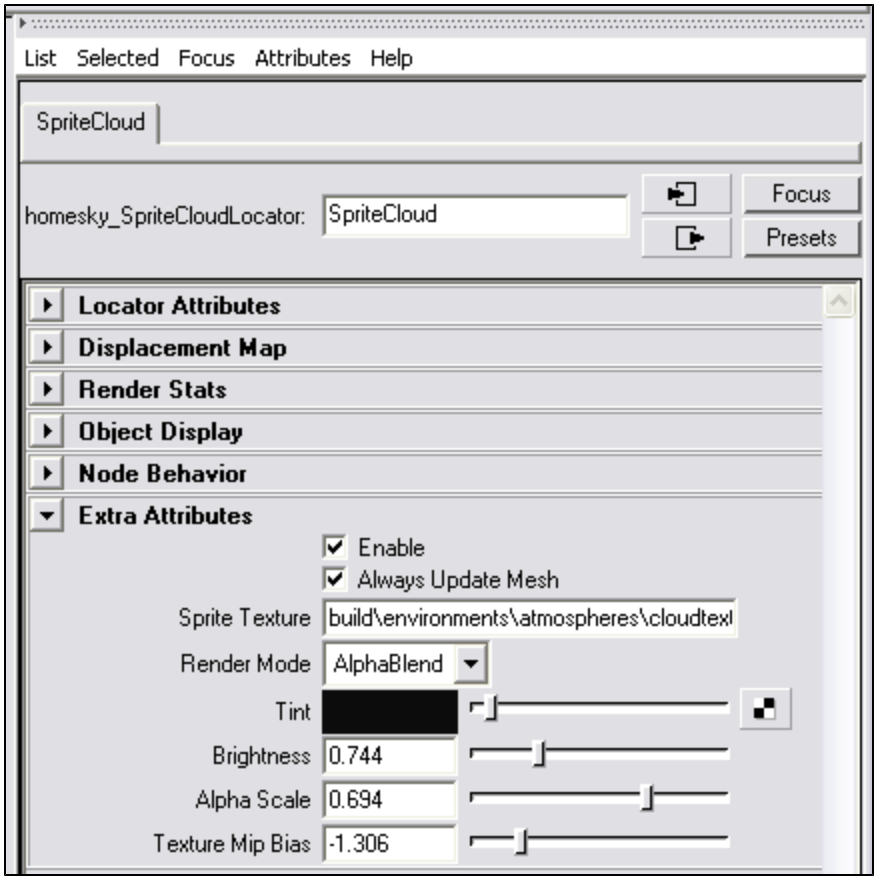
   Initially it is a quad shaped mesh created at the origin, and renders with a checkerboard texture.



2. Select SpriteCloudMesh in the Sky Objects list to display its attributes in the Attribute Editor.

   We need to provide a cloud texture for this mesh to render. For now we can use an example cloud texture provided in the HDK. Copy and Paste the following path into Attribute Editor > Extra Attributes:

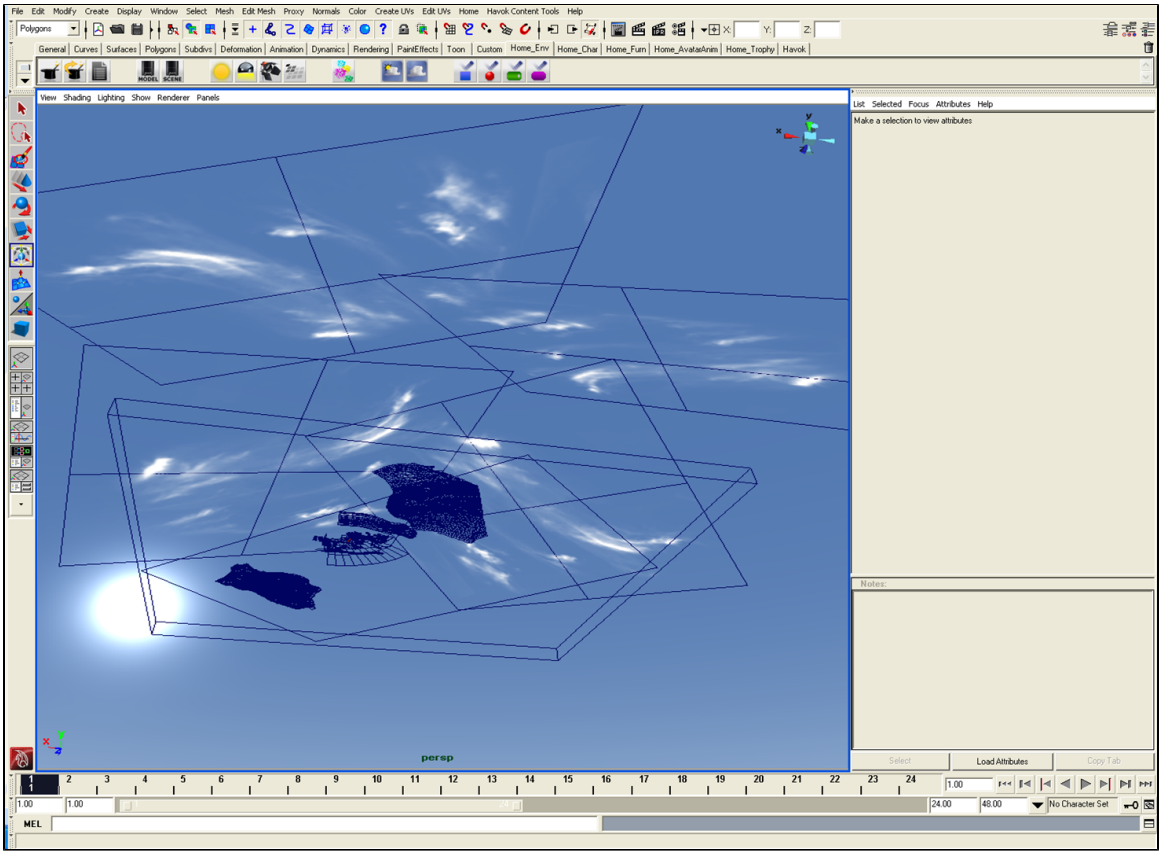
   ```
   build\environments\Atmospheres\CloudTextures\wispycloud.dds
   ```



You can also adjust the brightness and color of the texture here.

> No Maya materials or Shaders are used by the Home Sky Tool. All texture paths and shading attributes are defined on the nodes themselves.

Move, scale and rotate the mesh into an appropriate location in the sky. We can duplicate this node three or four times to provide a good coverage.



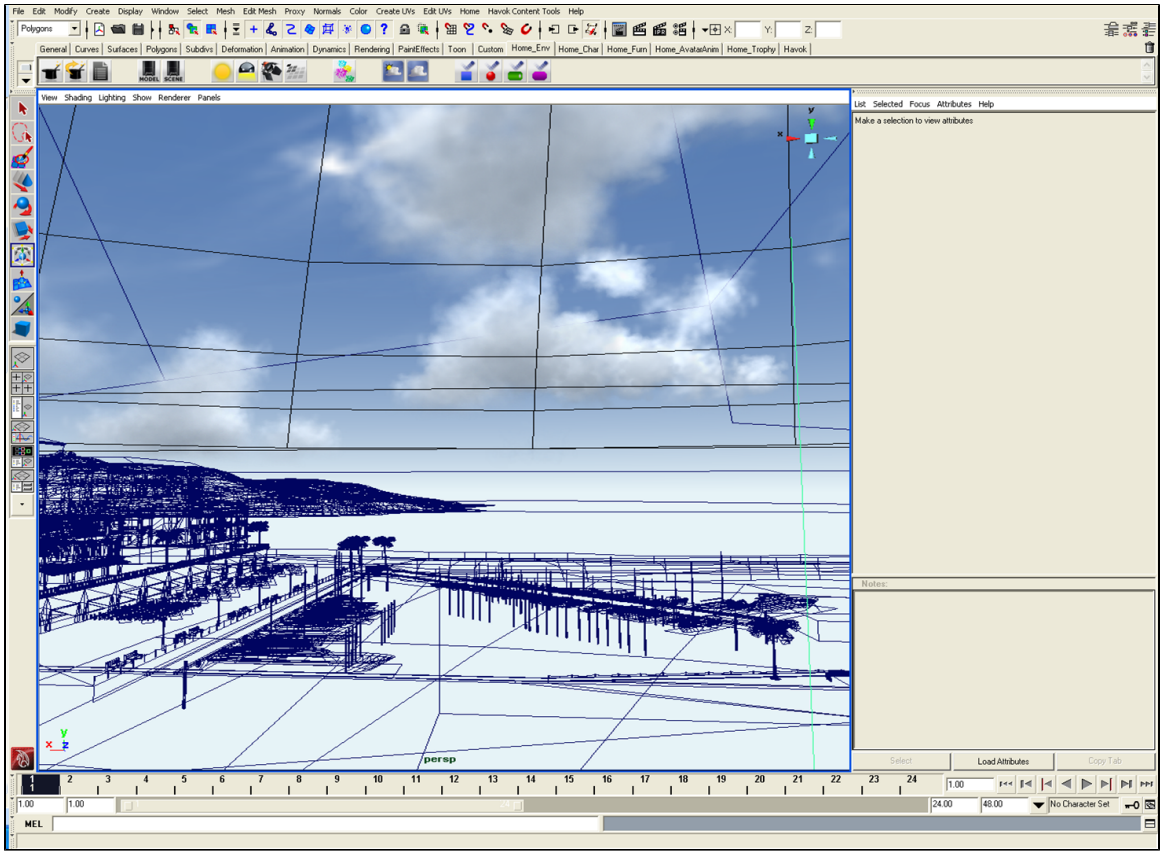
Next: Creating Particle Cloud Dome

Creating Particle Cloud Dome

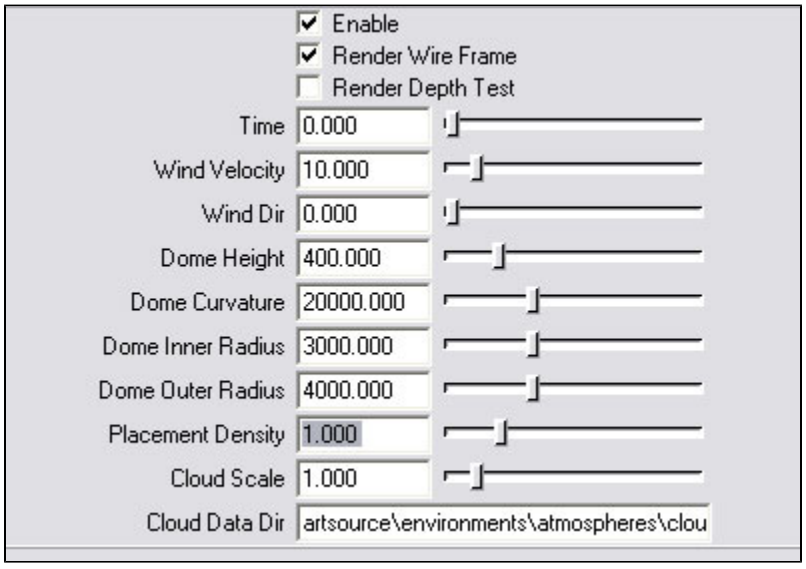
You use Particle Cloud Dome to create fluffy cumulous type clouds. This tutorial uses a pre-modeled cloud set provided in the HDK.

1. Click the CreateParticleCloudDome button in the Home Sky UI. A file dialog is displayed.
2. Browse to `artsource\Environments\Atmospheres\CloudLibs\SampleCloudLib1.xml` and click Open.

   The file loads and displays a sky with white puffy clouds.



A cloud library file specifies a set of clouds (usually 4 or 5) which have been modeled using the Cloud Modeling UI. A Particle Cloud Dome loads the clouds from one of these library files and scatters copies of the clouds randomly over sky in a natural dome shape.



It has attributes that control the shape and scale of the dome, as well as density and scale of the clouds. Wind Velocity and Wind Dir control where and how fast the clouds move in game.

Dragging the Time slider is a quick way to preview this in Maya.

When adjusting cloud density and scale controls, output of the number of clouds generated and memory consumption is displayed in the script editor.



The maximum number of clouds allowed is 250.

Another node that is loaded from the cloud library is Particle Cloud Shading Parameters. This node defines the shading, coloring and texturing parameters used for rendering the clouds. Select this node in the Home Sky UI to tweak a few of the parameters (for example, cloud color and shadowing amounts, etc.). It references an example cloud particle texture located in the HDK:

```
build\environments\Atmospheres\CloudParticleTextures\cloud_particle.dds
```

This texture is of a special format, which is described more in Sky Design UI Nodes.

Next: Creating Sky Fog

Creating Sky Fog

You can create fog or haze to give the horizon a more natural feel, using the ExpHeightFog node:

1. Position the Maya camera at the location in the scene that you expect the player to occupy.

   The sky fog effect varies depending on the location of the camera, so model it using the intended game view position. When the camera is in position, switch to Maya's flytool camera mode to view and model your scene in this section.

2. Create an ExpHeightFog node by clicking the Create Exp Height Fog button in the Home Sky UI.

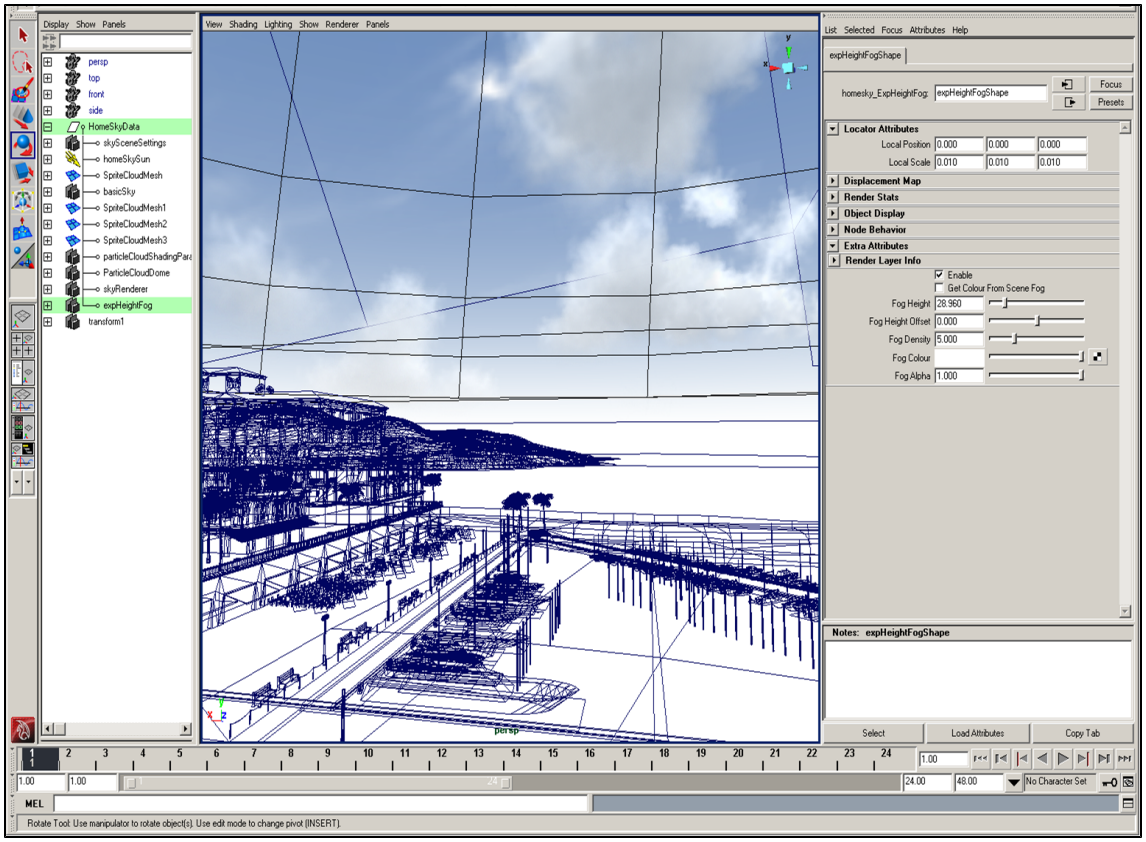
   The ExpHeightFog node defines a fogging volume where density drops off exponentially with altitude.

   If your scene appears flat white after clicking this button, your camera is too low in the Y axis. To correct the positioning:

   a. Select the ExpHeightFog node in the Home Sky UI.
   b. In Attribute Editor > Extra Attributes, adjust the FogHeightOffset attribute until you can see the scene properly again.

3. Adjust the various attributes on this node to see the different effects you can create. Initially, some attributes may seem to give the same effect, but each has its own subtle function. Experiment with each attribute until you get the desired result. Notice how the Particle Clouds sink into the fog volume on the horizon in a natural way.

If the horizon in your destination Home environment is not visible, you may not need to use this node.



Next: Creating Scene Fog

Creating Scene Fog

If your scene does not require a scene fog, skip this step.

A Home Sky scene also defines fogging for the main Home environment geometry via the SceneFog node.

1. Click the CreateSceneFog button in the Home Sky UI.

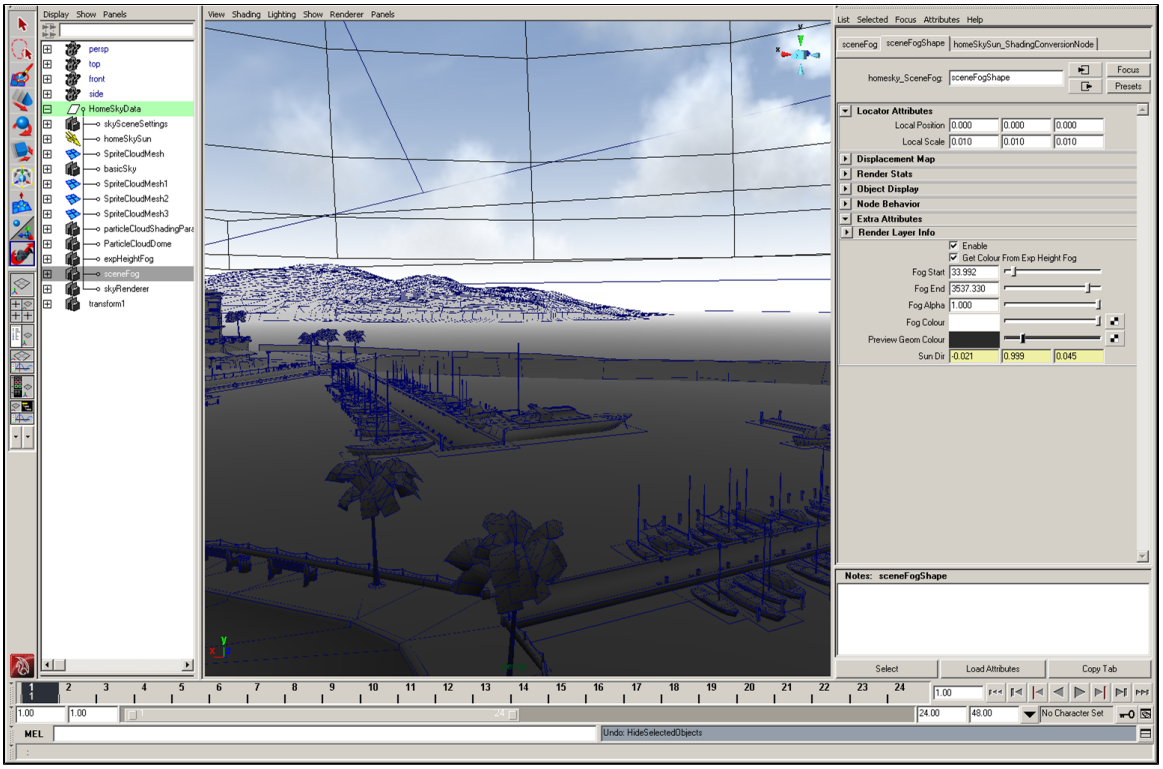
   The wireframe preview of your Home space is rendered and displayed. This is done by the SceneFog node.

   Adjust the various attributes on this node to get a fogging value that looks right for your scene.

   > ⚠ The color of this fog must match the color in the ExpHeightFog node. Otherwise a sharp discontinuity appears when environment geometry meets the sky.

2. With the SceneFog node selected, go to Attribute Editor > Extra Attributes and enable Get Colour From Exp Height Fog.

   This forces the scene fog color to always be the same as the ExpHeightFog node.



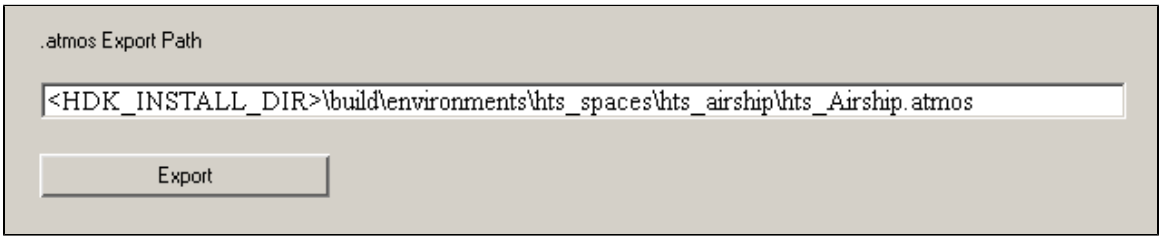
Next: Exporting and Viewing the Sky on the PlayStation®3

Exporting and Viewing the Sky on the PlayStation®3

The Home Sky tool exports to a .atmos XML file. It also produces a .cdata file of the same name if a particle cloud dome was used (as was done in this tutorial).

The export path is located at the bottom of the Home Sky UI.

1. Specify the full path of the .atmos file for export and click the Export button. The full path must be a location in the HDK build directory.
2. Export the file to the same directory, and name the file the same name as the destination Home space. For example `C:\HDK\build\environments\My_Scene\My_Scene.atmos`



This convention allows you to share a single .atmos file between multiple scenes.

When the export is complete, a dialog is displayed. If the export fails, the output in the script editor provides more information about the failure.

ATG Particle Effects Tool

You can edit particle effects for use with the Home Particle System runtime, using the ATG Particle Effects Tool. The runtime allows a more granular approach to constructing particle effects, without loss of efficiency. The tool enables you to create effects by placing a number of nodes into an effect graph, and tying together attributes on each node to represent the data flowing between the nodes.

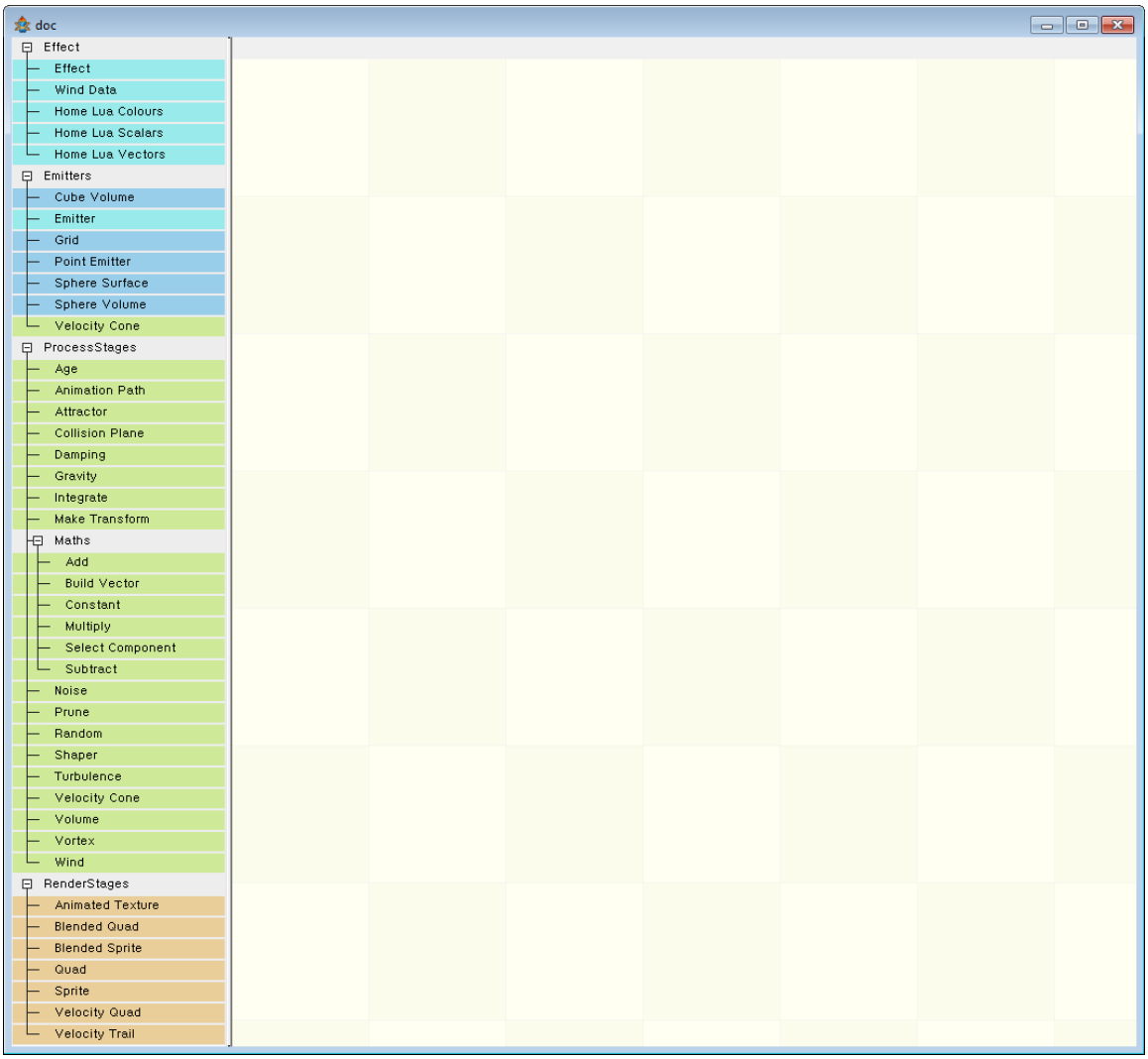
You can add particle effects only to scenes or scripted objects, not to furniture or character components.

This section describes how to create particle effects. It also provides guidance on how to make sure that particle effects are valid and successfully exportable so that they can be packaged within scenes and submitted for quality assurance.

ATG Particle Effects Tool Interface

The ATG Particle Effects Tool allows you to open multiple effect documents. An effect document consists of a Graph Display and an Effect Display.

Graph Display

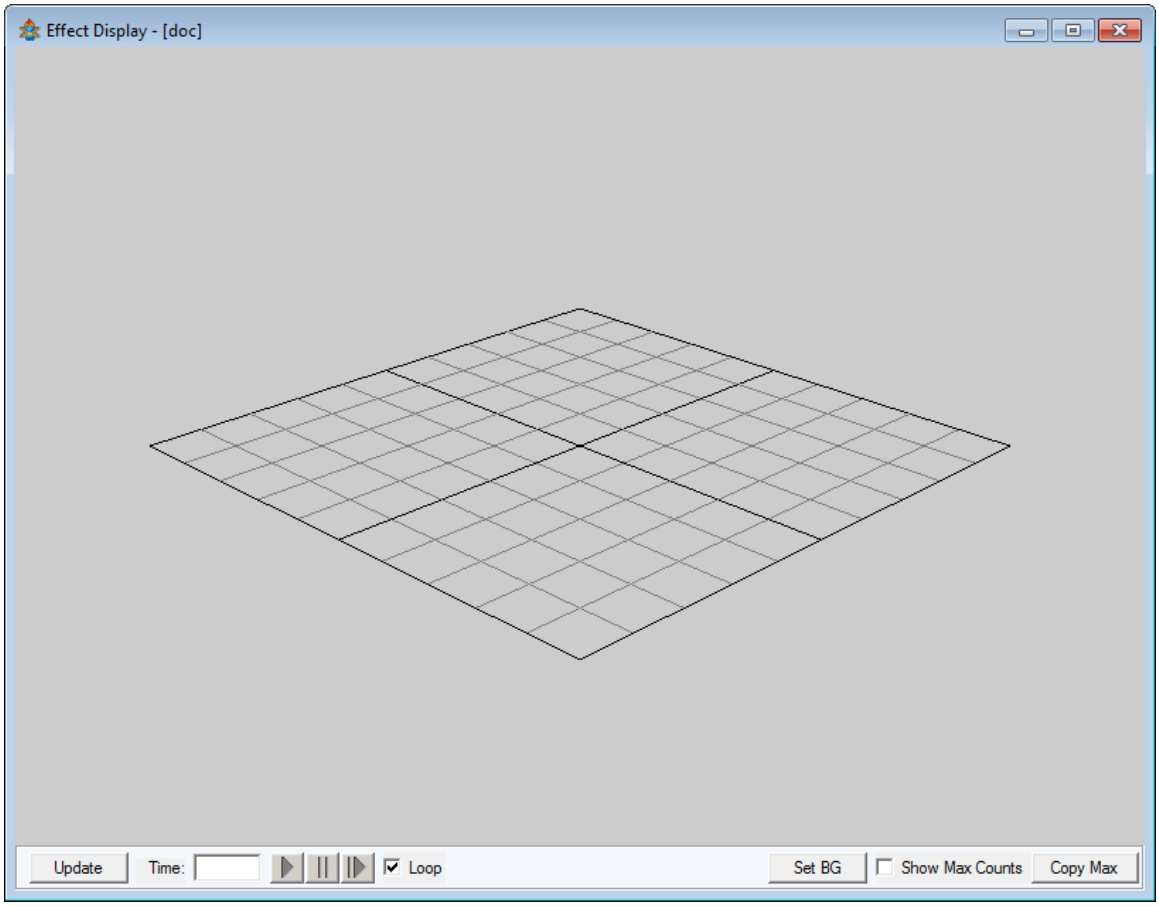


You can also open multiple Graph Displays for each document.

For more information on the individual nodes, see Node Types.

Effect Display

The Effect Display shows a standard viewport, with a unit grid. There are play, pause and singlestep buttons to control the playback of edited effects. The camera uses controls similar to those found in tools such as Maya - dragging the left mouse button orbits the focus point, dragging the middle button pans the focus point while dragging the right button controls the zoom level. Pressing R resets the camera position:



When using the Effect Display, press Ctrl + scroll with the mouse wheel to zoom.

The Graph Display is split into two sections:

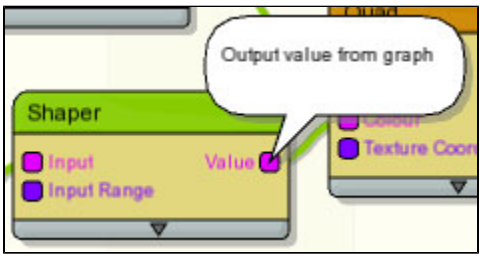
- On the left is a tree view of node types.
- On the right the graph is displayed.

New nodes in the Graph Display are placed by dragging them from the types list into the Graph Display.

Particle Effect Nodes

Dynamic Help

The tool provides dynamic help for node types, their inputs and outputs. Hover the mouse over the input or output port or the node title to see a description of the purpose:

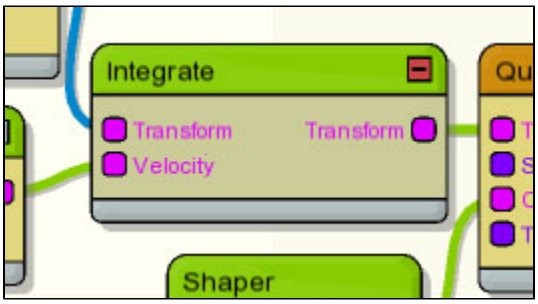


This feature is only available in English.

Node Bypass

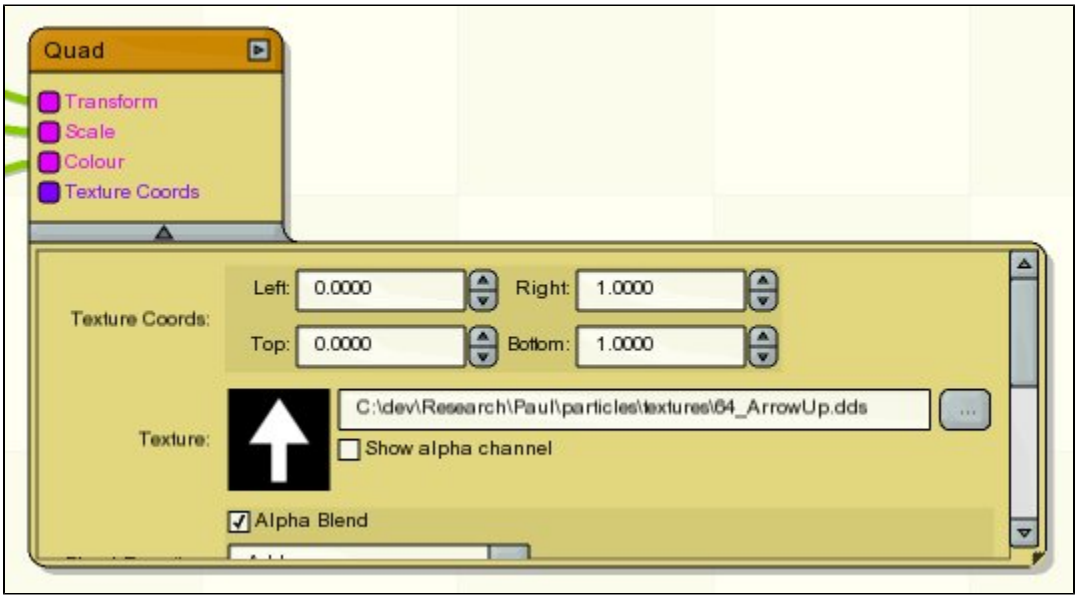
Most nodes in the tool can be optionally bypassed. When a node is flagged for bypass it simply copies any relevant inputs to their corresponding outputs without performing any modification on their values. This is convenient for disabling nodes without

removing them from the graph and works as expected across the various node types, for example, disabled emission triggers do not emit new particles, disabled process stages do not modify data and disabled render stages perform no rendering. The bypass flag, if present, is toggled by clicking the arrow button on the right of the node title bar. When bypassed the button turns red with a horizontal line through it as in the following image:



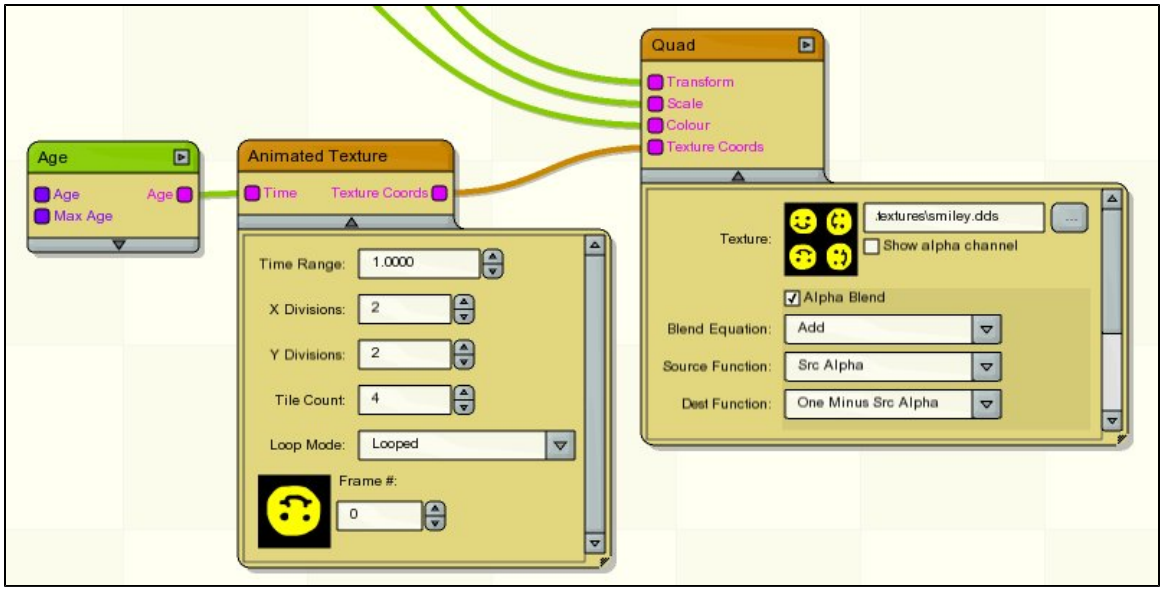
Texture Coordinates

All the current render stage nodes render using a single textured quadrilateral. The texture coordinates (UVs) used by each corner is by default {0,0} for the top left and {1,1} for the bottom right, mapping the entire texture onto the quad:



The texture coordinate parameter of each render stage node allows you to adjust these coordinates to use either a sub-section of the texture or to animate them under the control of another node.

The Animated Texture node is designed to work with a texture divided into regular animation cells, such as the example 4 frame rotating smiley texture:



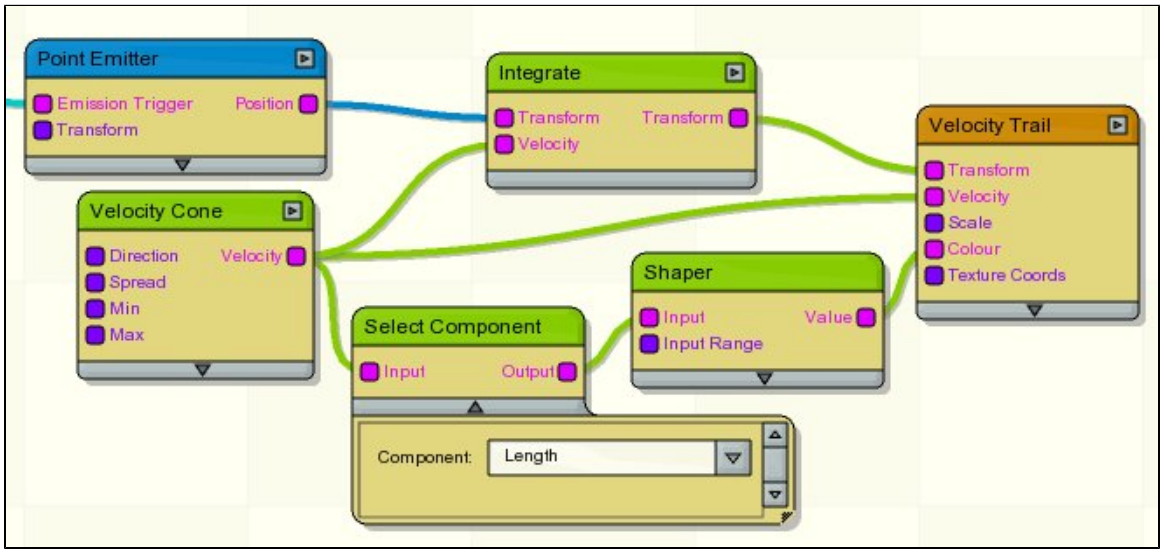
The parameters for the animated texture node consist of the Time Range (how long the complete animation should take) and three settings for how to divide the texture up; horizontal X Divisions, vertical Y Divisions and the Tile Count (number of tiles/cells in the texture).

The Loop Mode determines how the input Time value should be mapped onto the animation frames. When this is set to Once, the animation runs once then holds on the last frame. When set to Looped the animation repeats endlessly, while Ping Pong shuttles the animation forwards then backwards repeatedly.

Like the Shaper node, there is a single Time input port to specify which animation frame to use. Typically you would use the particle age so the texture coordinates animate at a regular speed as the particle ages. However you could also control the frame more precisely by using a Shaper node to finely constrain the input value based on another particle attribute, or use a Select Component node to extract the particle speed so the animation frame depends on the speed of the particle. Like all nodes in the system, you are free to experiment with inputting whichever edge you choose.

Select Component Node

When vector data is used (for example, particle position, velocity, etc.), it can sometimes be useful to extract a particular component to use as input to another node. One common example is to use the length of the velocity vector (i.e. the speed) to shape the particle scale or color so fast-moving particles appear differently to slow moving ones. The example effect selectnode.effect demonstrates how this can be achieved:

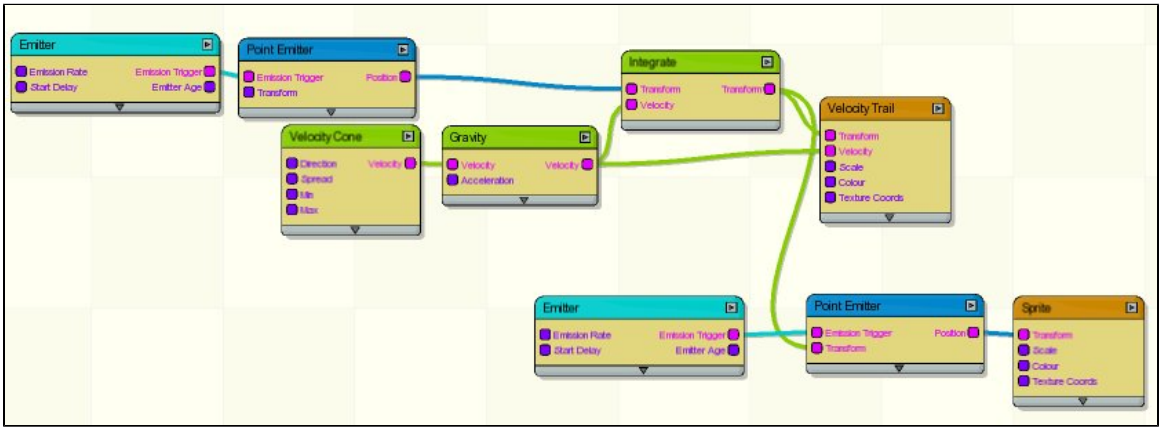


This effect uses the Select Component node to extract the length of the particle velocity to use as input to a Shaper node. The Shaper is configured so high value inputs (i.e. faster particles) produce less opaque particles in a basic attempt at faking motion blur. The Select Component node can also be used to extract any of the 3 vector components, X, Y or Z or the reciprocal of the length. You could, for example, extract the Y component of a particle's position to change the properties of the particle based on its height in the world.

Note how the output of the Select Component node does not need to be fed into a Shaper node. It could equally be fed into any number of other nodes, for example the Strength input of a Wind node, so particles with large Y positions are affected by a strong wind force. The system is designed for you to experiment with plugging edges or components of edges into any number of nodes to create your desired effect.

Hierarchical Effects

Hierarchical effects are effects containing several independent groups of particles where one group depends on data from another. The first, dependent, group is referred to as the child group, while the other group is referred to as the parent group. The simplest hierarchical effect is the rocket with smoke effect, where one set of particles is used to represent the rockets while a second set of particles is used to represent the smoke trailing from them. The example effect rockettrail.effect shows a very basic rocket/smoke particle effect and looks something like this:



There are two render nodes in this particle effect and two independent sets of nodes. The top 6 nodes from Emitter to Velocity Trail are used to produce a stream of rockets with initial velocity set by a Velocity Cone node, falling under a weak gravity field. The lower 3 nodes form the smoke trail. You will notice the 3 node construction of an Emitter node to trigger new particles, a Point Emitter to specify where the particles are emitted and a Sprite node to render them.

What makes this a hierarchical effect is the way the Transform input to the Point Emitter node is taken from the output of the Integrate node of the rocket nodes. Connecting two independent groups together in this way forms the parent/child relationship. The tool is doing the following when running the smoke effect group:

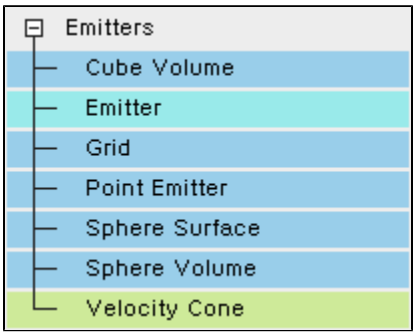
1. The Emitter node produces the Emission Trigger count each frame.
2. For every particle in the parent group, the Point Emitter node reads the Emission Trigger count from the Emitter node and the Transform from the particle in the parent group to produce a new particle in the list of smoke particles.
3. The Sprite render node draws sprites for each particle in the smoke particle group.

Note how the Point Emitter is run multiple times, once for each particle in the parent group. If the Emission Trigger outputs a count of 1 new particle then a new smoke particle will be produced for each active rocket particle.

There is no limitation on the complexity of the child group - this basic example could be augmented with Integrate, Gravity, Shaper and any other type of node, and it is quite legitimate for it to feed data to another child group, though you will quickly swamp the system with new particles if you introduce grandchild and great-grandchild particle groups.

Node Types

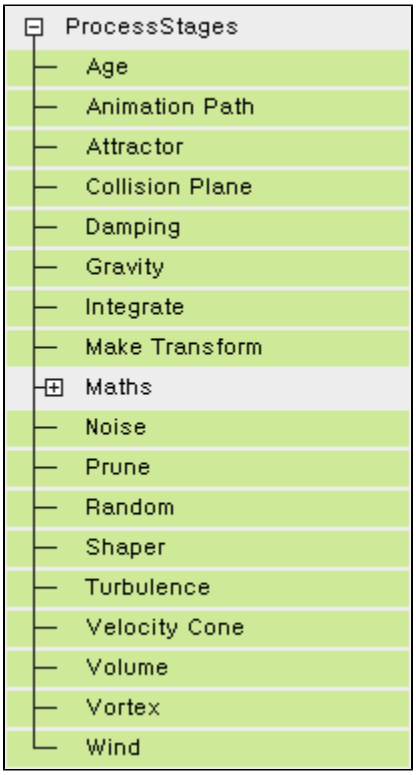
Emitters



Node	Description

Emitter	Emits new particles. This node is required by all effect graphs and is responsible for spawning new particles. The parameters for the trigger specify how particles are to be emitted, in random, fixed or instant modes. The node outputs an emission trigger that is hooked into an emission node such as the Sphere Surface node.
Grid	Produces transformations for a uniform grid. Emits from regularly spaced points on a plane with X divs in X direction, Y divs in Y direction. Emission mode can either be random to select a random regular point, or sequential to emit from each point in turn.
Point Emitter	Emits from a single point in space.
Sphere Surface	Produces transformations for a sphere surface. Produces Position and Normal data for a randomly sampled position on the surface of a sphere. The first is the emission position on the surface of the sphere. The second is the normal at that point. This could allow the artist hook into an initial velocity or velocity cone node.
Velocity Cone	Produces velocities in a uniform cone about the supplied direction vector. Produces vector3 velocities within a cone oriented along "direction", with spread range and minimum and maximum lengths.

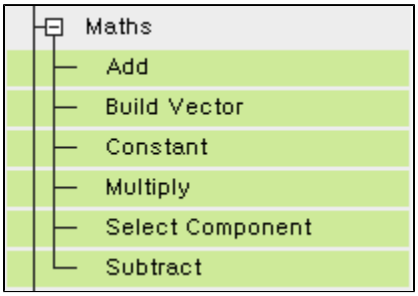
Process Stages



Node	Description
Age	Increments age value. Typically used as input to a shaper node to select a value given the particles age.
Attractor	Attracts particles to a point with a given strength.
Collision Plane	Collides particles against a 2D plane, specified by a transformation and scale. This node outputs a trigger when collisions occur that can be used to spawn new particles.
Damping	Multiplies all components of the input by the damping factor.
Gravity	Adds a linear acceleration.
Integrate	Integrates position by velocity.
Make Transform	Combines position and orientation into a transform attribute.
Noise	Sample procedural noise at a given position.
Prune	Deletes particle when trigger edge is set.
Random	Produces random values between min and max for each output component.

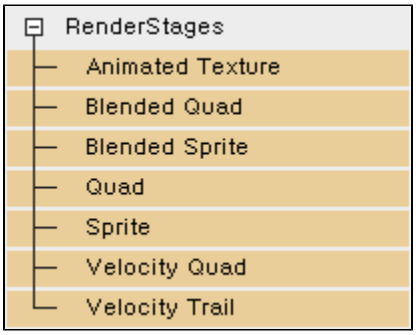
Shaper	Returns a value based on a curve.
Turbulence	Adds a random force scaled by a magnitude factor.
Volume	Tests whether a transform is within a certain volume. The volume is specified as a sphere, cube or cylinder and the return value can be produced either as a 0/1 value for outside/inside or a distance from volume. This is useful for modifying an attribute based on the particle position relative to a volume, for example, a turbulence field could be implemented by hooking a volume node into the turbulence strength input, possibly via a shaper to control the exact strength.
Wind	Adds a linear acceleration in a particular direction with a particular strength.

Maths



Node	Description
Add	Performs the numerical function of Adding.
Build Vector	Allows you to construct a vector from the 3 constituent components. Useful for specifying exact values for each component, for example, to rotate around one particular axis.
Constant	Can be used for specifying a constant in a single place. This is for convenience only, and will be optimized out by the exporter.
Select Component	Allows you to isolate any of the three vector components, or extract the length or reciprocal length of the vector. Useful for keying data from a single component or length of a vector.

Render Stages



Node	Description
Animated Texture	Produces animated texture coordinates by splitting a texture into regularly spaced tiles. X divisions, Y divisions and Tile Count specify how the cut the texture into animation frames.
Blended Quad	As Quad, but using a blend factor between two textures.
Blended Sprite	As Sprite, but using a blend factor between two textures.
Quad	Draws particles as an oriented quad with a texture.
Sprite	Draws particles as a camera facing quad rotated on the Z-axis.
Velocity Trail	Quad oriented in velocity direction.

Working with Particle Effects

Launching the ATG Particle Effects Tool

To launch the ATG Particle Effects Tool:

- Select Start > All Programs > Home Development Kit > ATG Particle Effects Tool or,
- Run <HDK_INSTALL_DIR>\dev\bin\ParticleToolLauncher.exe

File Locations

When authoring effects on your computer, store files in the artsource folder, for example, artsource\Environments\Your_Environment_Name_Here\ATGParticles in one of the following subfolders:

- Effects: where the ATG Particle EffectTool .effect files are stored.
- Textures: where the hi-res work in progress is stored temporarily (remember that the .effect file should point at the textures in the build directory when exporting).

Exporting for Publication in PS Home

To export valid particle effect files for PS Home, you must structure the files in the build directory, for example, build\…\…\… in the following sub folders:

- ATGParticles\efx: Where the exported .efx files are stored.
- ATGParticles\Textures: Where the final world-ready .dds and .gtf textures are stored.

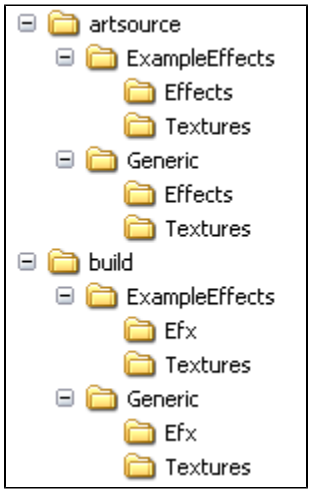
> A .gtf file is a texture file format that is organized more efficiently and optimized for the PlayStation®3. Both .dds and .gtf formats support the same range of texture types and both are acceptable in Home.

Working with Existing Particle Effects

There are existing particle effects provided with the ATG Particle Effects Tool in the HDK in the ExampleEffects and Generic folders.

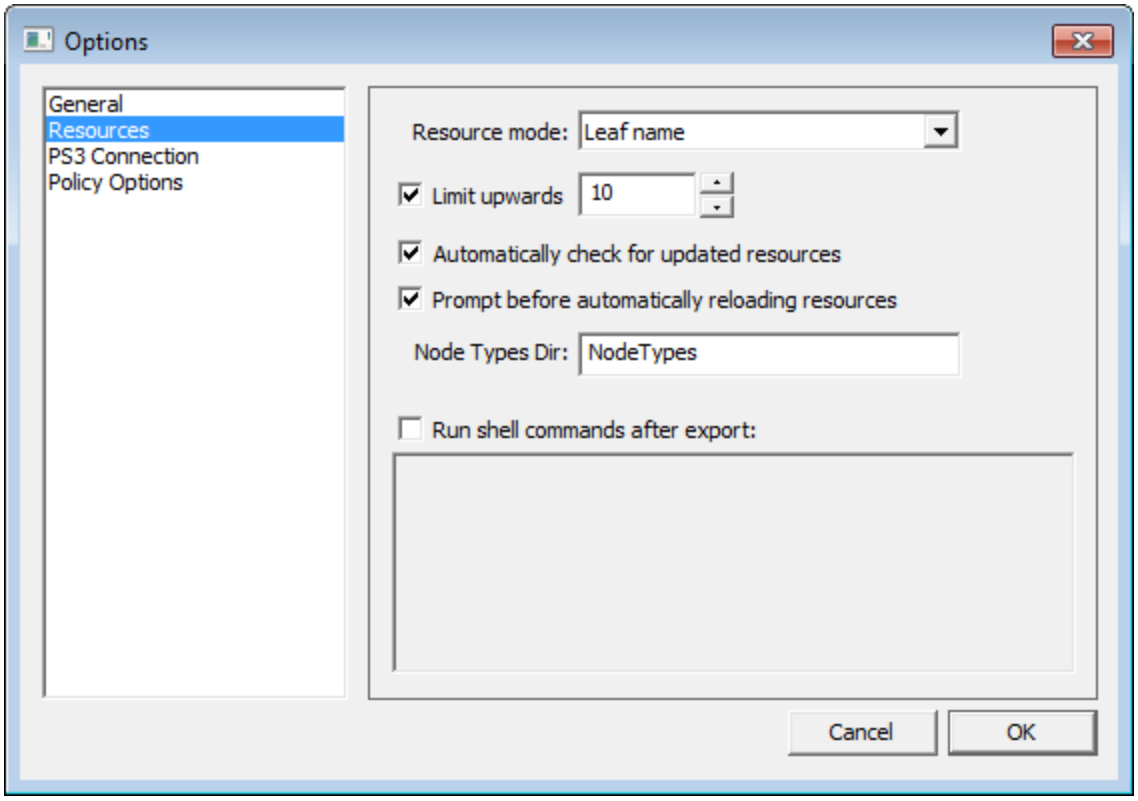
These contain the effects and textures, both in unexported format (i.e. in the artsource folder) as well as exported format (i.e. in the build folder).

The ExampleEffects folder contains examples of how various particle effects work. The Generic folder contains ready-made effects that can be easily modified or incorporated into your work.



Ensure that the effect tool is set to the correct Resource Mode as follows:

1. Select Tools > Options. The Options dialog is displayed.
2. Select Resources on the left side of the dialog:



3. Select Relative path from the Resources drop-down list.
4. Click the button next to Base path and specify the correct location.
5. Click OK.

Particle Effects and Backward Compatibility

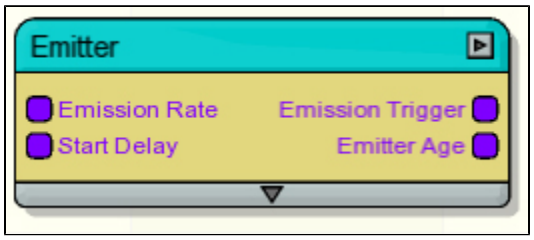
From HDK v1.1 onwards, the particle effects are backwards compatible. Re-export particle effects created before this version as follows:

1. Open your .effect file in the ATG Particle Effects Tool.
2. Ensure the texture references are updated.
3. Save the updated .effect file.
4. Export the particle effects to get an updated .efx file.
5. Test your scene that contains the particle effects to make sure the update was successful.

Creating a Particle Effect

To create a new particle effect:

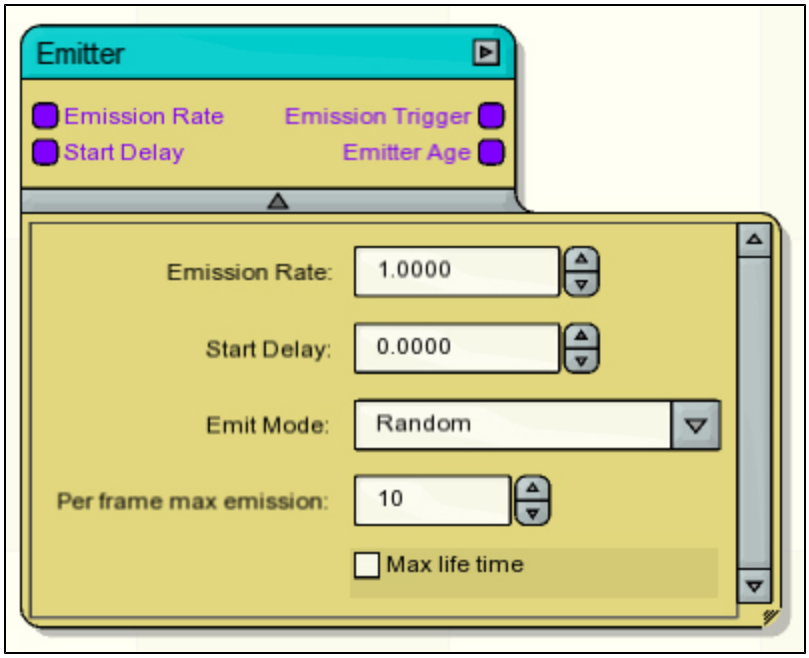
1. Select File > New.
2. Drag an Emitter node type from the node type list onto the Graph Display:



The central section lists attributes the node expects to receive or output, with inputs on the left and outputs on the right. The Emitter has two inputs and two outputs:

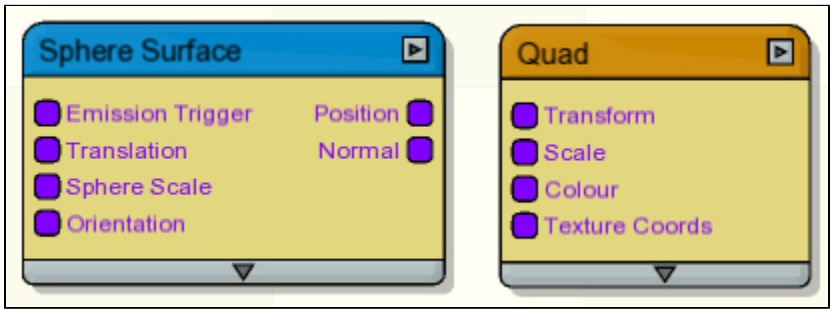
- Inputs: Emission Rate and Start Delay
- Outputs: EmissionTrigger and EmitterAge

3. To reposition a node in the graph, select the name of the node and move it as required.
4. To access the properties for a node, click the grey bar across the bottom of the window.



In the graph, emitters produce data on the left of the graph, process stages modify it in a left-to-right order and render stages produce graphics on the right of the graph. This is enforced by having input attributes listed on the left of a node object, with output attributes on the right.

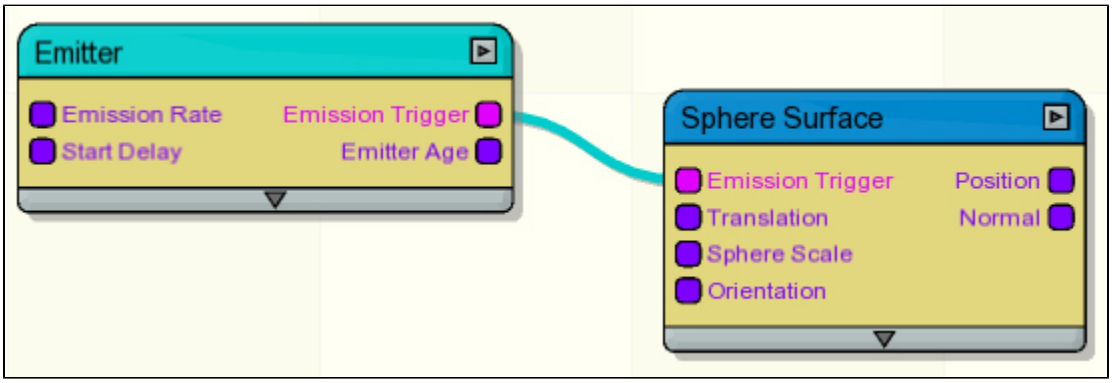
5. Drag a Sphere Surface node and a Quad node onto the Graph Display:



This node takes a number of inputs, specifying the properties of the Sphere Surface emitter and produces two Transform outputs: Position and Normal. The Position output is the emission position on the sphere, the Normal output is the normal at that point. This is useful for hooking into an initial velocity or velocity cone.

   a. To flow data from the emitter to the sphere surface node, drag a connection from the purple square next to the Emission Trigger output attribute on the emitter node, to the corresponding input attribute on the sphere surface node.

      Note that the edge turns red when the mouse is hovering over a valid target, and invalid connections are marked with an 'X'.



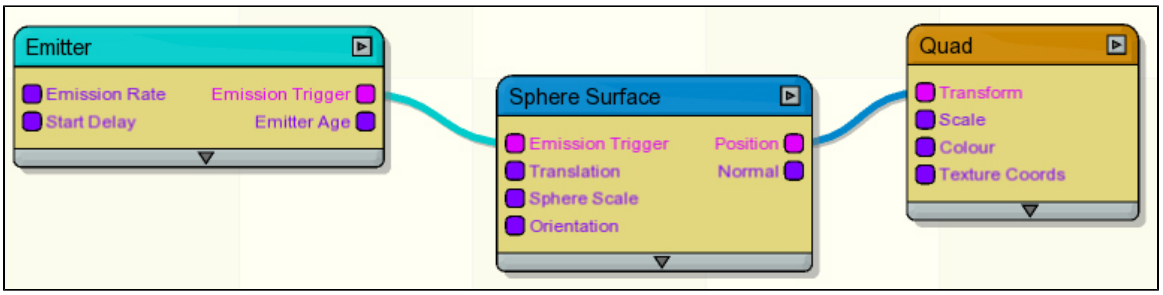
      Each edge represents the flow of data from one node to another, tying an output attribute of one node to an input attribute of another. The tool prevents you from making bad connections, such as connections that would result in

a loop or short circuit in the graph. Each output attribute can connect to many input attributes of subsequent nodes, but each input can have only one edge arriving.

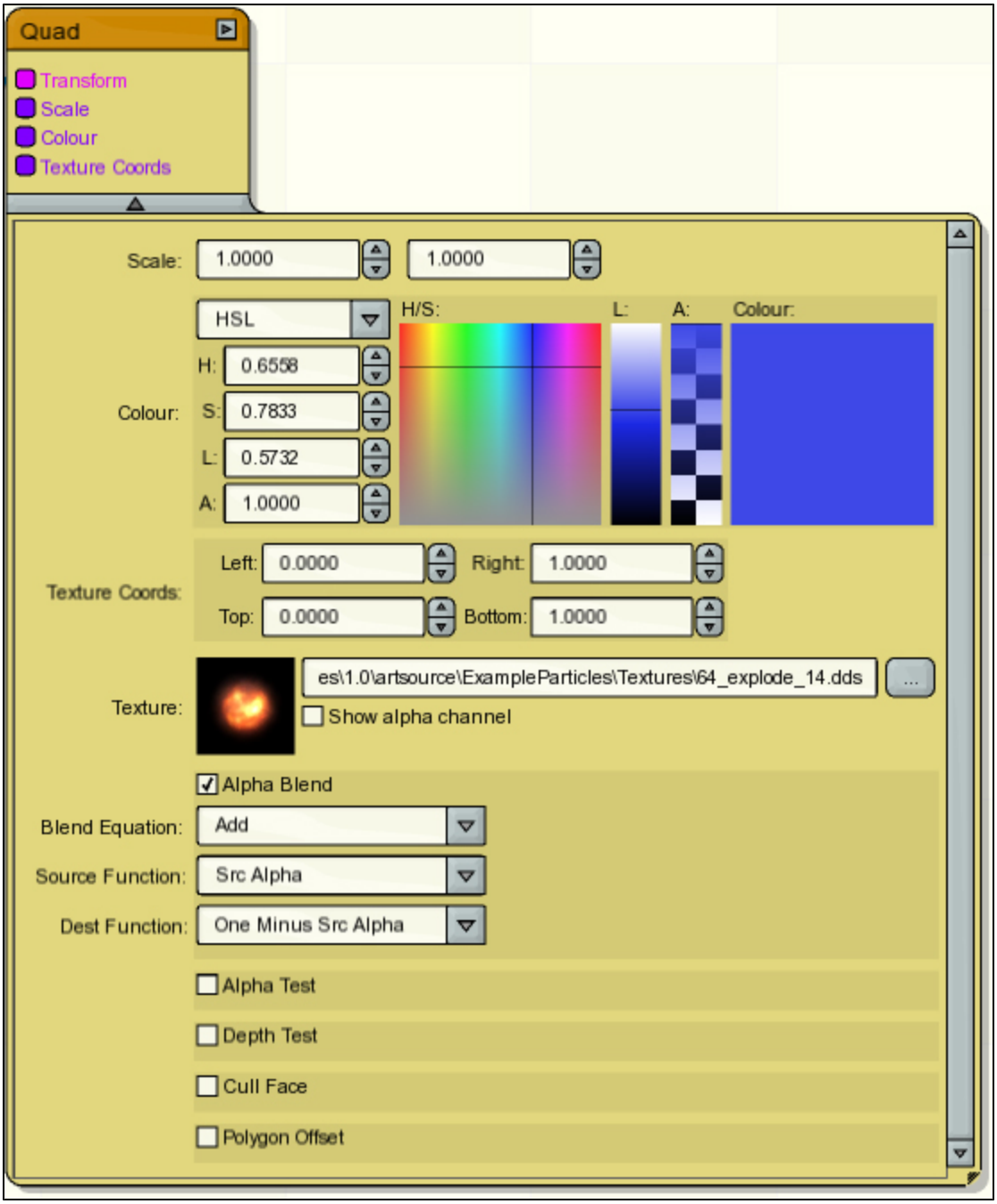
b. To clear an attached edge, click on the input attribute.
c. To create an edge, start from either end.
d. To select a node, click once in the title bar (this turns red to indicate it is selected).
e. To select multiple nodes, hold the Shift key and click the title or hold the Ctrl key and drag out a selection rectangle.
f. To reposition a node, drag the title bar of the required node.
g. To pan the display, drag non-active parts such as the background.
h. When the graph is quite complex, with many nodes with properties to edit, use the mouse wheel to zoom in or out of the display.
Certain nodes can display properties in the Effect Display window. The Sphere Surface node can optionally display the position, orientation and size of the sphere used for emission.
i. Open the property panel for the SphereSurface node and check the View inparticle display box. This enables you to position, scale and rotate the sphere into the required position.



6. Connect the nodes by dragging in a Quad node and connecting the Position output of the SphereSurface node to the Transform input on the Quad node:



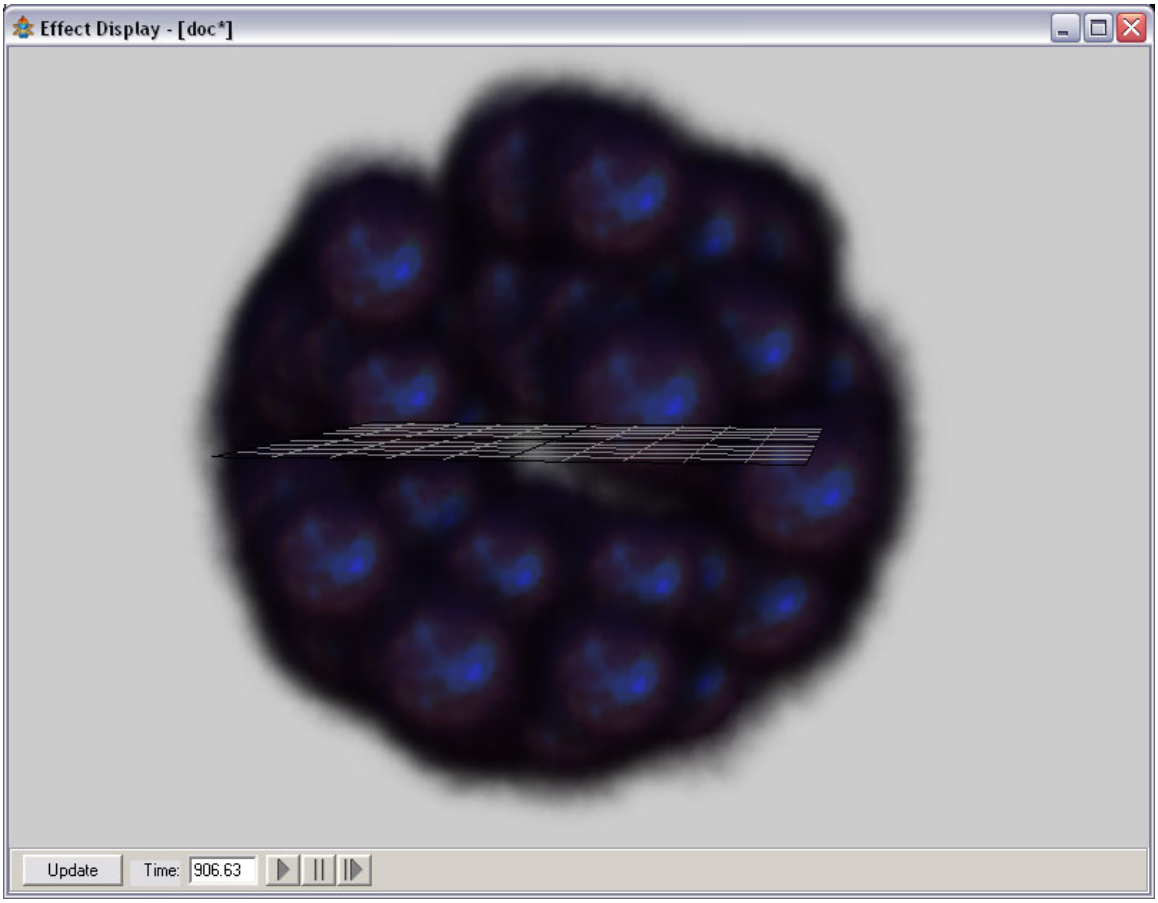
a. Opening the property page for the quad node shows a number of controls for setting how the node renders the particles (as shown below). Certain properties, in this case Scale, Color and Texture Coords, can be given constant values or can be sourced from other nodes. Others, such as the texture and alphablend states, are fixed for the node.



The color picker control operates in either HSL or RGB modes, selected from the drop-down menu. Exact values for HSL/A and RGB/A can be entered in the four number entry controls, while clicking/dragging within the square or two columns picks the color under the mouse cursor. The color is shown in the right-hand block, without alpha (at the top) and with alpha overlaid on a grid pattern (at the bottom).

b. To select the texture to use, click 

to the right of the file name control box and select an appropriate .DDS or .GTF texture file.
c. Check the ShowAlphaChannel box to display only the alpha channel of the loaded texture. The remaining parameters specify standard RSX graphics operations such as alpha blend, alpha test etc. Later we add nodes to adjust the scale and color dynamically. These three nodes are sufficient to produce a very basic particle effect.
d. Moving back to the Effect Display window, click the Update button to start the effect. This displays a stationary set of particles created on the surface of a sphere.



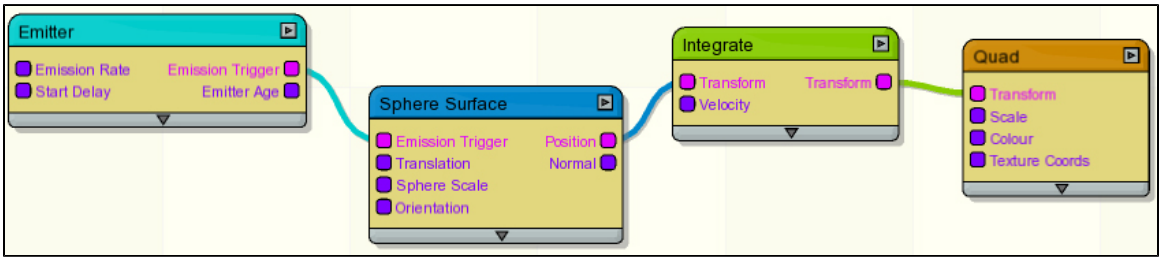
e. To pause, resume and single-step the update process, click the VCR controls at the bottom of the display window. The effect keeps running until a broken connection is encountered in the graph which prevents it from forming a valid effect and produces an error message.
f. After it is repaired, click Update again to restart the display.

7. Insert an Integrate node.

The Integrate process stage moves the particle transformation along a supplied velocity.

a. Move the Quad render stage to the right and add an Integrate node between the SphereSurface and Quad nodes.
b. Connect the Position attribute output from the Sphere Surface node to the Transform input on the Integrate node and again from the Transform output on the Integrate node to the Transform input on the Quad node.

The graph is as follows:



c. Open the property panel for the Integrate node to specify the velocity to add in.

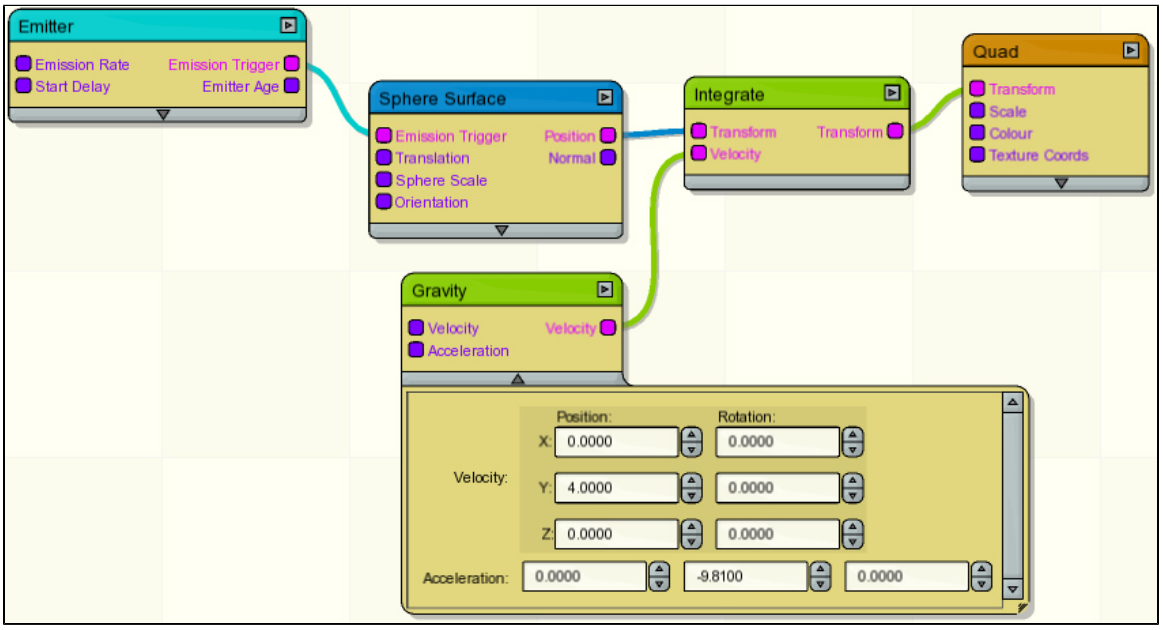
When the Velocity input to the Integrate node is disconnected, as it is now, you specify the velocity as a constant value. Later we connect the velocity input to the output from another node to use dynamic velocities and the constant velocity property in the property panel disappears.

Velocities and transforms in the particle system combine both positional and rotation components. These are specified by positions along or rotations about the X, Y and Z axes. Set the Y component of the velocity position to 1.0 to give the particles a vertical trajectory.

d. Click Update again in the particle display window to show the particles moving upwards at a constant speed.

8. Include a Gravity node to accelerate the particles.

    a. Move the Integrate and Quad nodes to the right and insert a Gravity node between the SphereSurface and Integrate nodes.
    b. Hook the Velocity output from the Gravity node into the Velocity input on the Integrate node and set the initial values as follows:

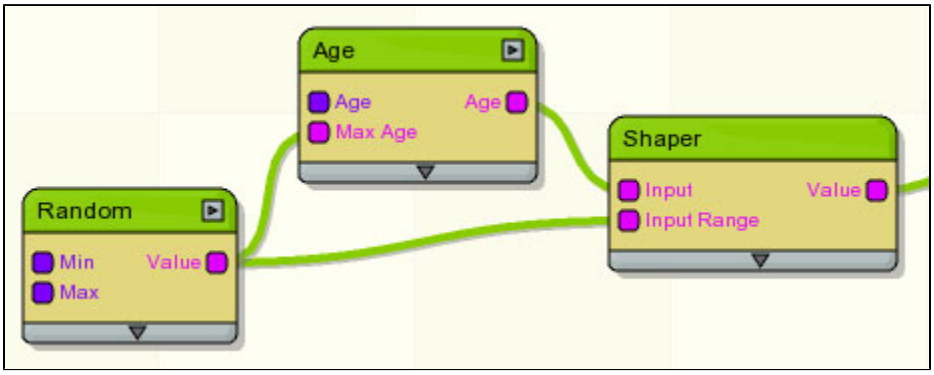


With the initial velocity set to 4m/s and the acceleration at -9.81m/s/s (meters per second per second), the particles are emitted with a vertical velocity, but soon drop under the influence of the gravity node.

Using Shapers in Particle Effects

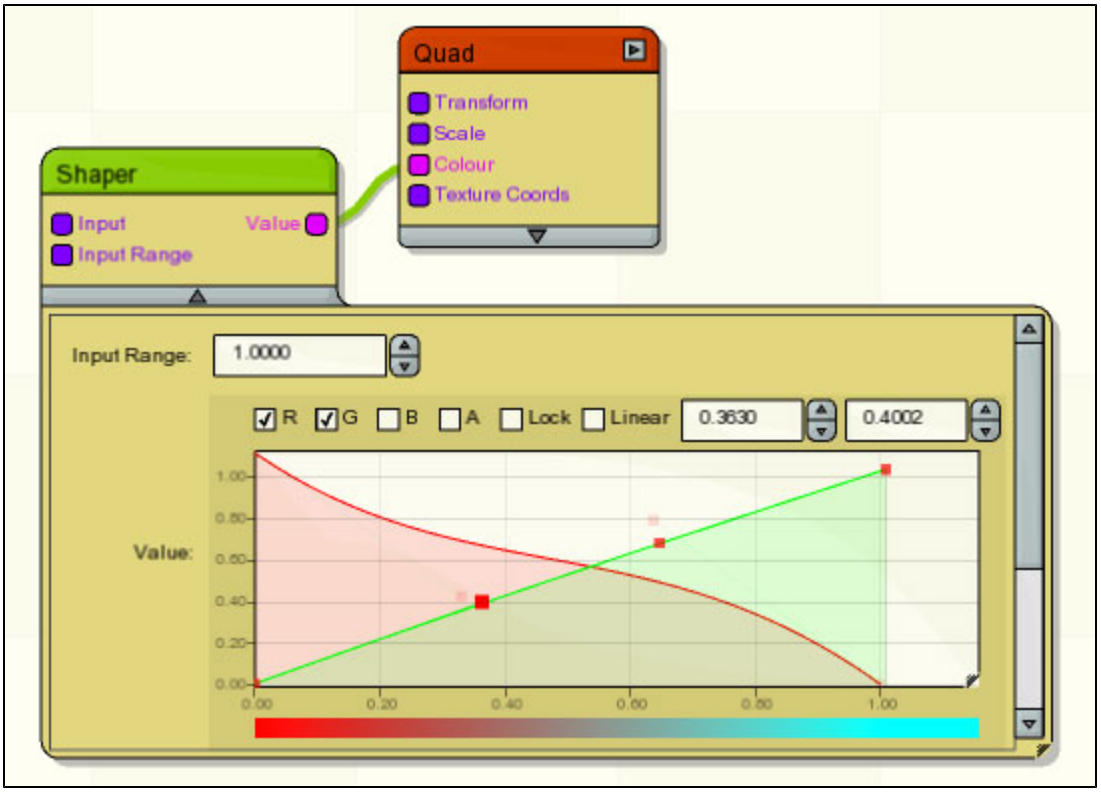
You can use shapers to control particle attributes via a curve or a line. Shapers have two inputs.

- Input 1 is the position to read in the graph, usually the time value.
- Input 2 is the 'Input Range' and specifies the scale of the horizontal axis of the graph. This is necessary if you want to shape values over the age of a random life time particle, as in the following subgraph:

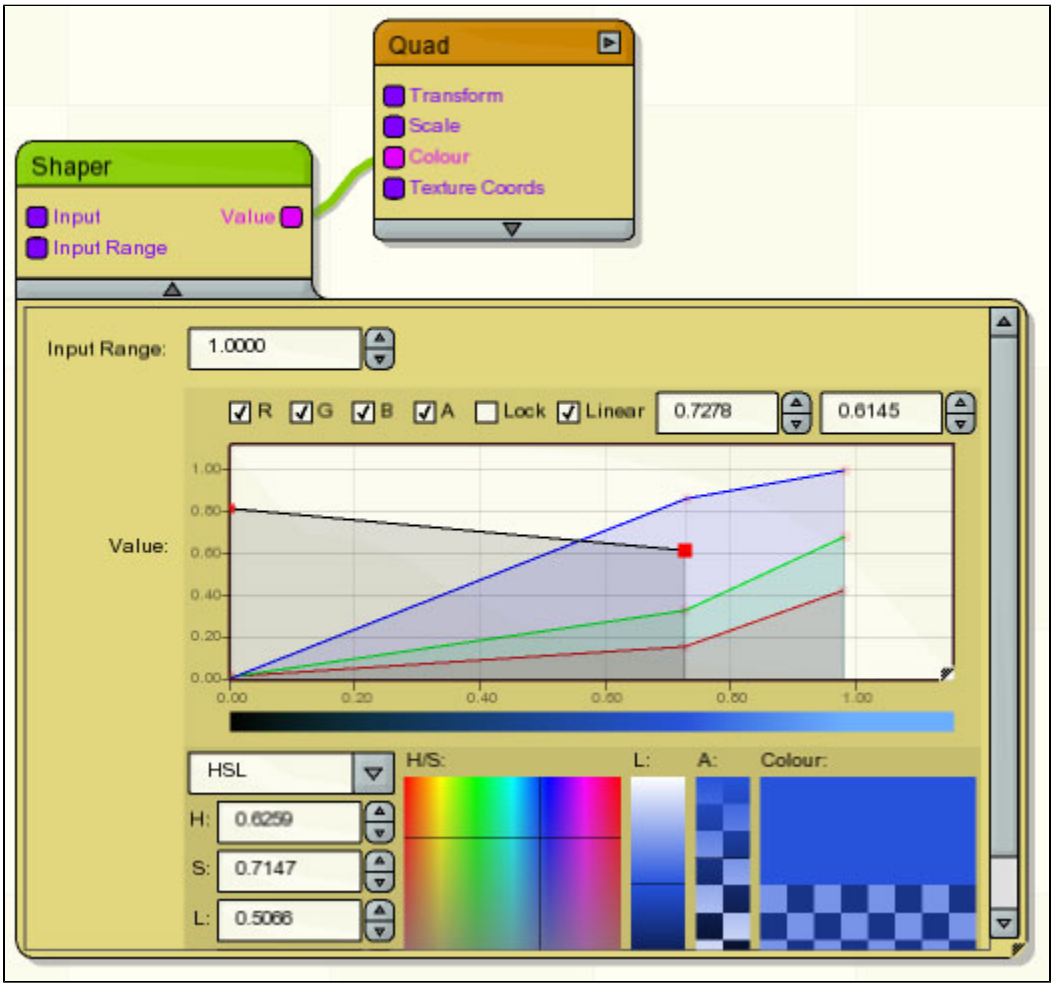


They then produce output in a form usable by a range of attribute types, including single scalar values, vector values, lines and colors.

Curves (with Linear unchecked)



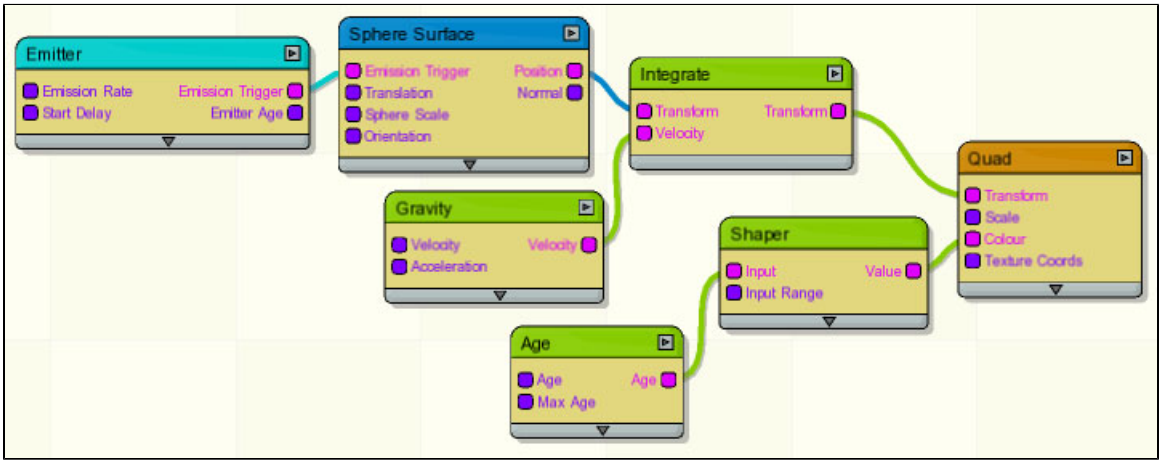
Lines (with Linear checked)



The Value field shows the curve, with Input along the X axis and returned Value along the Y axis, in this case color. There are four curves to edit, one each for the R, G, B and A components of the color. When used as a color source, the color represented by the curves is displayed under the graph, in this case a red to cyan graduated bar. Which color curves are shown in the graph is controlled by the four boxes labeled R, G, B and A. Components may be locked together by making them active and checking the Lock box. All components that are locked together output the same value, i.e. locking R, G and B together would simplify producing grayscale output.

Curves are represented either by a number of Bezier segments or, if the Linear box is checked, by line segments. Drag the handles to shape the curve, or select a handle and type exact values into the two text edit controls. When dragging a handle, the Shift key constrains one axes (for example, to keep the X value the same) while Ctrl snaps the value to the nearest 0.05 position. New segments are created by clicking on the curve at the point you wish to form a new join, while pressing Delete when a join handle is selected removes that segment, adjusting the curve to fit the existing shape as well as possible. Dragging the background to the curve editor repositions the origin, the mouse wheel controlling the zoom, and the Resize tab on the bottom right of the graph allows you to change the visible area displayed.

You can use shapers to control a range of things in an effect graph.



The basic gravity effect augmented with a shaper controlling the color of the quads

The input to the shaper is an Age node. This node simply increments an age value with the time step each frame and is used to key values from the particle age. The Shaper node could take its input from the emitter age or any other attribute in the system. The Select Component node selects components from vector attributes so color can be keyed from particle Y position or speed, for example.

Particle Effect Console Commands

This section summarizes the ATG Particle Effect Tool console commands in PS Home.

Console Flags

```
atgParticlesUpdateEnable  < BOOL >
```

Disable or enable per-frame evaluations of all particle systems (effectively pause/unpause time for particles)

```
atgParticlesRenderEnable  < BOOL >
```

Disable or enable rendering of all particle systems.

Console Commands

```
atgParticlesReloadEffects
```

This command reloads all active particle effects (.efx files) from disk. This enables a quick turnaround between authoring in the ATG Particle Effects Tool and visualization in-game.

For more information on console commands, and launching the Home Self, see the The Home Client.

Particle Effects Profiling

For Japanese Live Page
最新の日本語版ページはこちら 「パーティクルエフェクト･プロファイリング」

Particle effects use VRAM (for textures), MAIN (for effects), HOST (for data) and push buffer memory. The push buffer reflects memory consumed by commands sent to the GPU to render the particle effect.

Profile the memory used by particles using the Profile GUI. If a user exceeds the budget it will result in some of the particle effects not being rendered in PS Home.

VRAM, MAIN, HOST memory

Particle effects must fit within the budget of whatever content type they are a part of. For example:

- A personal space (including the particle effects within the space) would need to fit within the memory set for personal spaces.
- An active (including the particle effects within the active) would need to fit within the memory set for Active Items.

Push Buffer memory

The push buffer reflects memory consumed by commands sent to the GPU to render the particle effect. You must keep to the following pushbuf (push buffer) limits:

- Scenes pushbuf = 128 KB
- Actives pushbuf = 2 KB

Particle effects are not supported in other types of content (such as companion objects or furniture objects).

Exporting Particle Effects

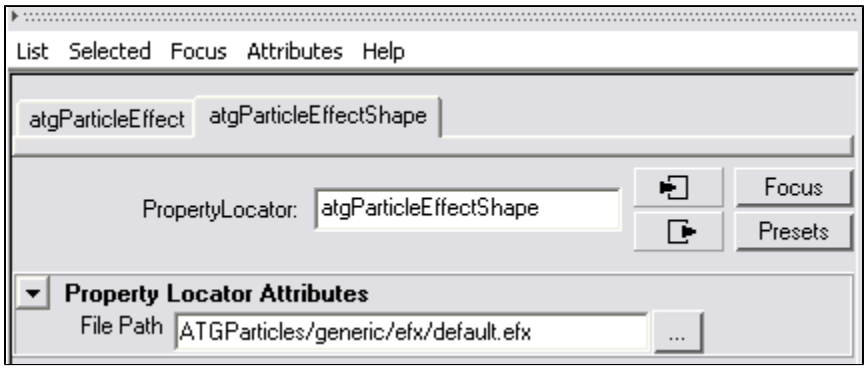
To export an .efx file format, select File > Export. If you have adhered to the required file structures, this creates a valid particle effect file that can be referenced from (and integrated into) an environment from Maya, or scripted into a scene.

For more information on the process of incorporating a particle effect into a scene by creating a particle locator, see Introduction to Maya.

Integrating Particle Effects in Maya

To reference the particle effect from the DCC Tool (Maya):

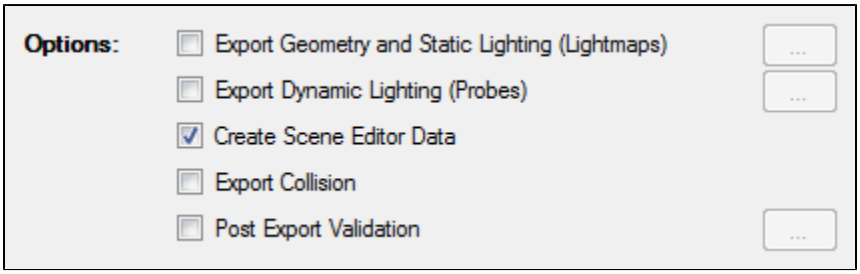
1. Create an ATGParticles locator in Maya, by clicking  on the Home_Env Shelf, or selecting Home > Environment > Create Particle Locator.
2. Position the locator in the environment.
3. In the Property Locator Attributes panel, click the  button next to File Path to browse to the effect.



The File Path is the .efx file to be used in the effect.

> The paths are rooted in the build directory, for example, to reference '…¥Build¥ATGParticles¥generic¥efx¥BUBBLE_EFFECT.efx', enter 'ATGParticles¥generic¥efx¥BUBBLE_EFFECT.efx'.

The exported particle data is handled by the Scene Editor Data. Therefore, you do not necessarily need a full scene export to make changes to particle data. You can just re-export the scene editor data by itself and reload the scene in PS Home.



Avatar Animation

The character animation system allows you to use multiple libraries of animations. These libraries are referred to as repertoires. For a given avatar, there is a base repertoire determined by the sex of the avatar.

You can add more repertoires to an avatar from other objects, such as clothing items, portable items, mini-games, and actives. This allows for flexibility and variety in the animations available for an avatar.

Animation Network

An animation network is a collection of animation states. Each state within the animation network defines a set of animation resources that are played when that network is active. Before a network is activated, it has its network conditions evaluated to see if it is a valid network. If the network has no conditions, it is taken to be a valid network which can be activated.

When a network is activated, all the states have their conditions evaluated to determine if they are currently valid. From the generated list of valid states. one is randomly selected by the client code to become active. After that state terminates, a new state is selected by evaluating the state conditions again. You can override the first state selected by specifying the `initial_state` value of the network.

Types of Animation Networks

There are two types of animation networks:

- Behavior: A behavior is an animation network that is active indefinitely (loops). See Repertoire Behaviours.
- Action: An action is an animation network that has a finite lifetime and self terminates. An action network with a lifetime of zero terminates when the first state terminates. An action network with a non-zero lifetime terminates when its network age is greater than or equal to the lifetime. The lifetime of an action can currently be set only by the client and is used for text chat and voice chat actions. See Repertoire Actions.

The animation system allows for only one behavior to be active at any given time, but more than one action may be active at the same time as a behavior.

Animation Resources

Animation resources are the `.ani` files created in Maya. The following animation resources are available:

- Skeletal Animations: to animate the joints in a skeleton.
- Animation Blend Trees: to combine multiple source animations using control variables.

Puppet Rig and Creating Animations using the Puppet Poser show how to create animation resources.

For information about how to add an animation resource, see Adding an Animation Resource.

Avatar Animation Repertoires

Repertoires Introduction

Repertoires are a connection of nodes that will control how and when your avatar animation will play. The repertoire will allow you to define what an animation resource is.
For example:

- Is it a loop?
- Does it only need to play once?
- Will you need to blend several animations together?
- Do you want to directly control the animation in some way?

All of these things are possible but the core of this is still the animation exported from Maya. You may find that you need to change or adjust the animation as well as the Repertoire.

Repertoires can be used in:

- FullBodySuits
- Minigames

While the repertoire will function in the same way for both a FullBodySuit and a minigame, the way in which you call the animations and play them is different. Most of these differences will become obvious once you choose which type of item you wish to attach your repertoire to.

Repertoires are a network of nodes similar to the particle tool. It is possible to create a very simple or very complex repertoire. The most important thing it to keep a clear final result in mind, because of the large number of options available when creating the repertoire, it can be daunting and multiple solutions can and often will be available to you.

There are however only two basic ways to play an animation: Actions & Behaviors.

For information on how the elements of a repertoire are related, see Repertoire - Relationships.

FullBodySuit

A repertoire for a FullBodySuit is simply a way to add animations to a FullBodySuit character component. It doesn't contain any scripted elements and the animations can only be played based on what the repertoire can do internally.

In PS Home all the animations will appear in the emote menus. These are the menus that all the default emotes such as waves and dances already appear.

Additional menu entries can be added and your animations can appear under them. It is also possible to place your custom animations under an existing emote menu, by naming it exactly as that existing menu item is named.

Minigame

Minigames use the repertoire to apply custom animations to the avatars in the game.

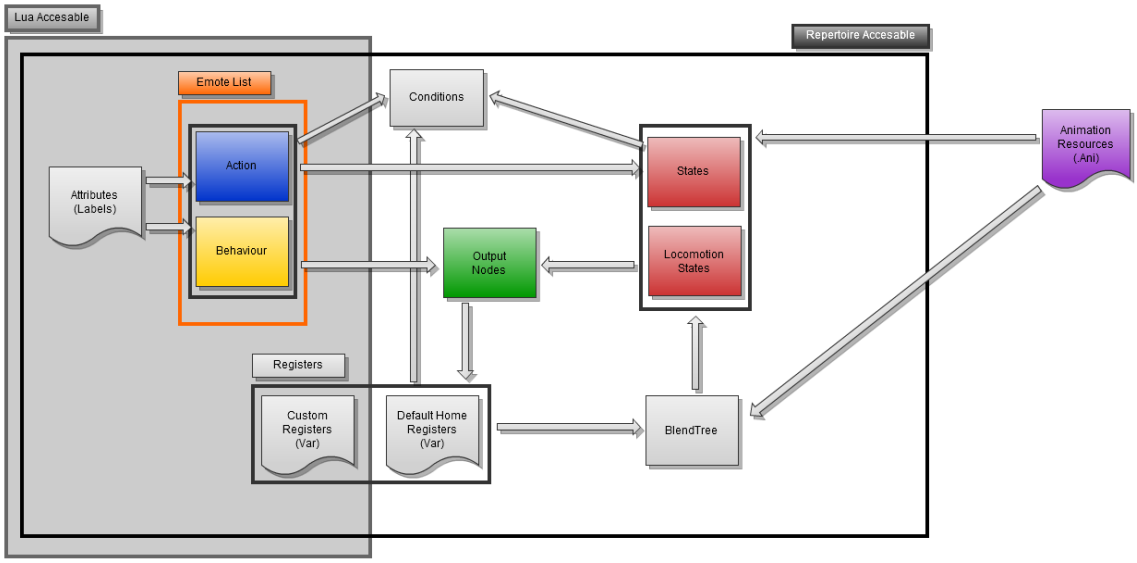
How and when the animations are called is controlled by both the repertoire and the Lua API that relates to it. This means that it is possible to create a more complex interaction and have more logic that will determine when and how the animations should play.

Animation Blend Trees will also allow you to directly control how an animation will play, for example directly controlling how much a character will lean based on the joypad.

Repertoire - Relationships

Repertoire Relationships

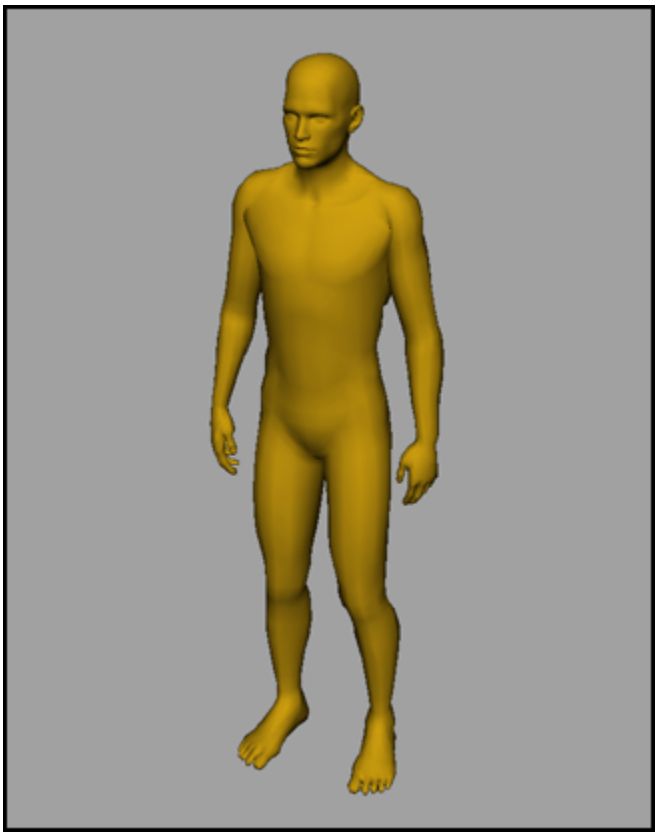
The diagram below shows the main elements of the repertoire and how they relate to each other. The large boxes show which elements are accessible by the repertoire or by Lua (see the Lua API Reference).



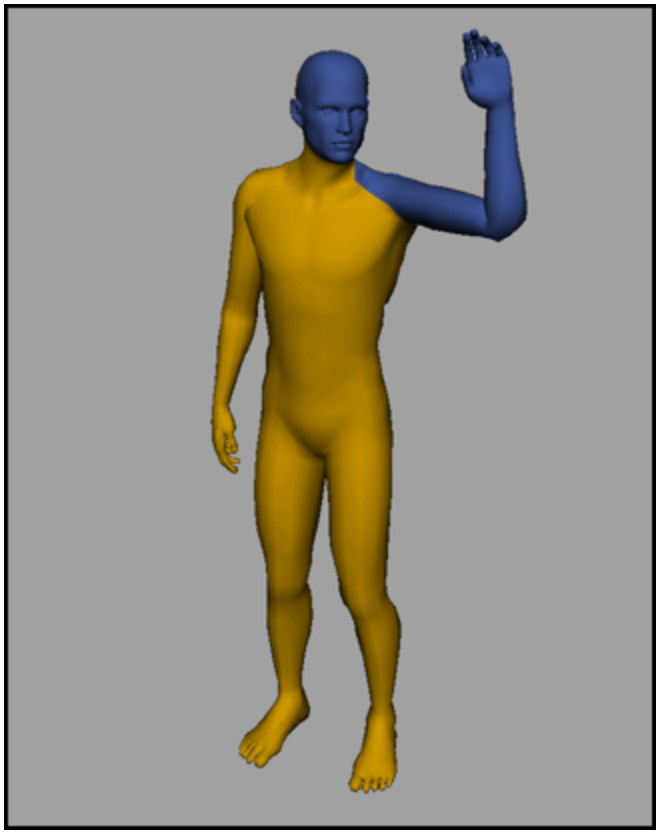
Repertoires – Actions & Behaviors

Actions and Behaviors

Actions and Behaviors are the two types of animation that can be called from a repertoire. At any time the avatar can have a number of Actions playing. The avatar will always have one Behavior playing (this is often an idle).



The avatar with just a behavior (yellow) playing on the whole body



The avatar with the previous Behavior (yellow) as well as an

additional Action (blue) playing over the top (Head and Arm)

Behaviors and Actions are made in the following way. The main node is the Behavior or Action (B,A) then states are added (S) and finally animations are added to the states.



Behaviors

The Behavior is the base animation. It will loop and play continuously. The idle animation, the dances and poses are all Behaviors. Once selected or played via the Lua API they will play continually until replaced by a different Behavior.

There are some special cases, for example Locomotion. Locomotions are a form of behavior that is controlled by the speed the avatar is moving - e.g. when the avatar is stationary it will swap back to the idle animation and then resume when the avatar moves again.

Common Output Node Functions:

- Output - Looping Behaviors
- Behavior Properties

Actions

Actions are single, play once animations - e.g. a wave or a jump or anything that will play a single time and then not repeat.

Actions play over the top of Behaviors. There are several ways to control how the Action will play in conjunction with the underlying Behavior.

Common Output Node Functions:

- Output - Set Next Behavior

- Action Properties

States

All Animations are added to a State and the State in turn is added to the Action or Behavior. When an Action or Behavior is set on the avatar, the State is selected and the attached animation will play.

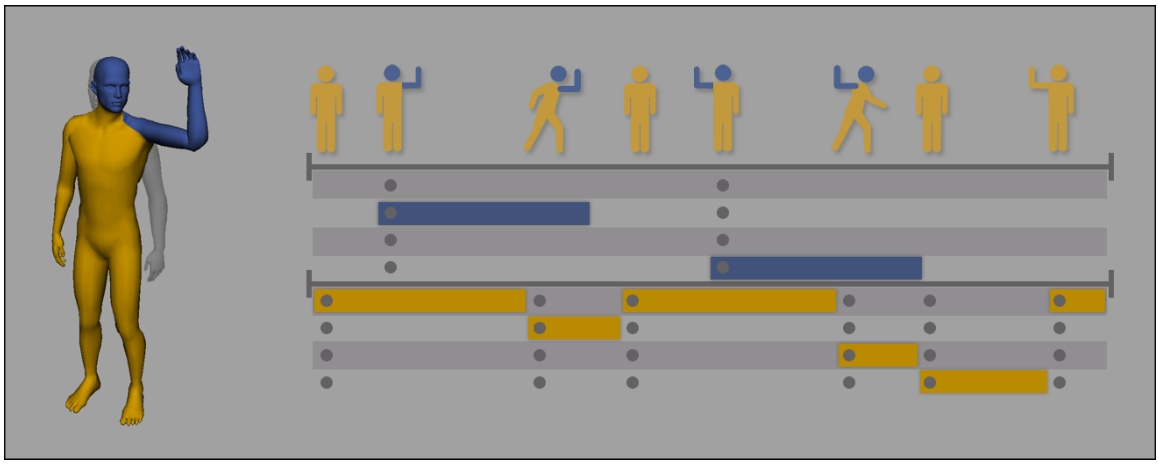
- State and Animation Properties
- Animation Resources
- Animation Blend Trees

The image below shows an Action (blue) and a Behavior (yellow) playing. The yellow Behavior loops, so it will continually select animations in its State to play, picking a new one each time an animation ends.

The blue Actions are played when the player pushes a button. You can see how they play over the top of the Behavior.

Each of the gray dots in the 2 tracks represents a state in that Action or Behavior. When the Action is called or animation ends in the behavior a new animation is selected from this list at random.

In this example there are 4 states for each but you can have as many or as few as you like. If there is only one state it will always be selected.



States and Conditions

Conditions can also be added to help control which State is selected and when.

This can be useful for something like an idle where you may want several subtlety different animations to be chosen from a list.

- Conditional Logic

Output

Actions Behaviors and States can all have an output. This can be used to set registers and change other elements of the repertoire or the Action, Behavior or State.

- For detailed information on the Output, see Repertoires – Output Nodes

Action Properties

Attributes

The Action node has a number of properties that can be set. They are outlined here with examples of their usage.

Blocking

Having blocking set on your Action will prevent the Action from being interrupted by player input. This can be used for a number of purposes, in most cases blocking is used to prevent avatar moments from interrupting the Actions animation.

Example Usage:
1. If the Avatar is transitioning from sitting to standing you do not want the the player input to interrupt the transition animation . By setting the animation to blocking the animation will play out in full. 2. If you have an animation such as a wave that will only affect the upper joints of the Avatars rig you can set the animation to Blocking so that while running the wave will continue to play even if the locomotion animation uses the same upper body joints.

Duration

Duration will override the existing animations length.

Example Usage:
You may have multiple animations of different lengths making up an Action, by overriding the duration you can chose how long the animation should play for.

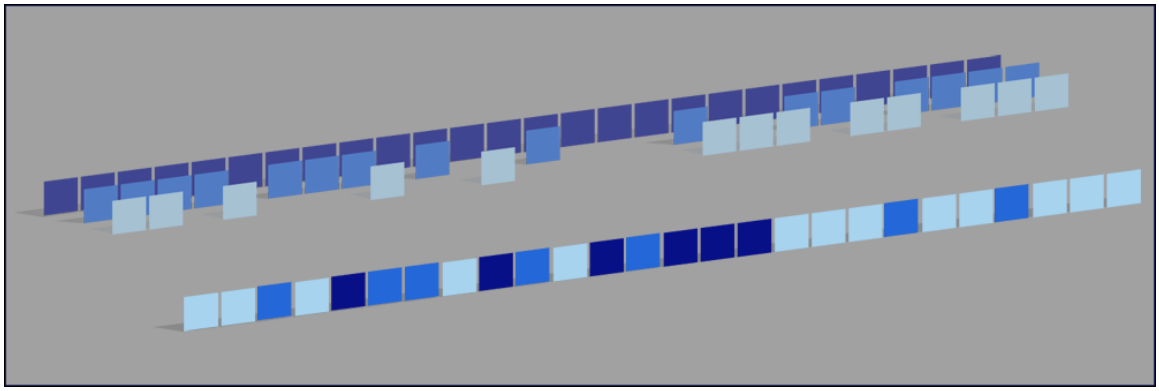
Initial State

If the Action contains more than one state, you can chose which one should play first.

Layer & Priority

In order to play multiple Actions at once you will need to control the Layer they play on. You may also need to control the priority of these layers.

The image below shows 3 layers.



Each layer has an animation playing on it. These animations use a number of different joints. By playing them on different layers the final result will be a blend of all the animations.

Priority can be set for these layers in order to change which joints will show through.

The example image shows the final result of the three layers with the light blue having the highest priority and the dark blue having the lowest.

Name

The name of the Action. You will call this name in Lua when playing the Action.

Behavior Properties

Initial State

If the Behavior has multiple states this state will be always be chosen to play first. This can be useful when using conditions to control the play order of states by giving a fixed starting point.

Attributes

The Attribute acts like a label. A string can be assigned to the Behavior that can be recalled via Lua to help identify the Behavior in game via Lua.

The attributes name must be set to: **sys_category**.

To retrieve the value you will need to use the Lua command `Person.GetBehaviorCategory`.

Ideally should be set to one of the defaults listed in the Lua API.

Default Attribute Values:

- "dancing"
- "posing"
- "sitting_on_floor"
- "sitting_on_seat"
- "standing"
- "moving"

Name

Name of the Behavior.

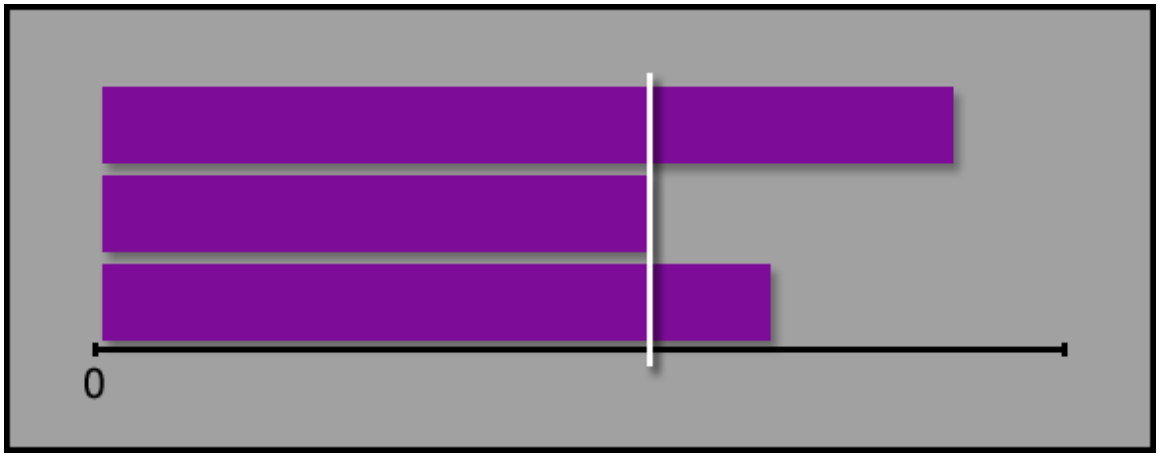
State & Animation Properties

State Properties

Duration

An override for the length of the Animation.

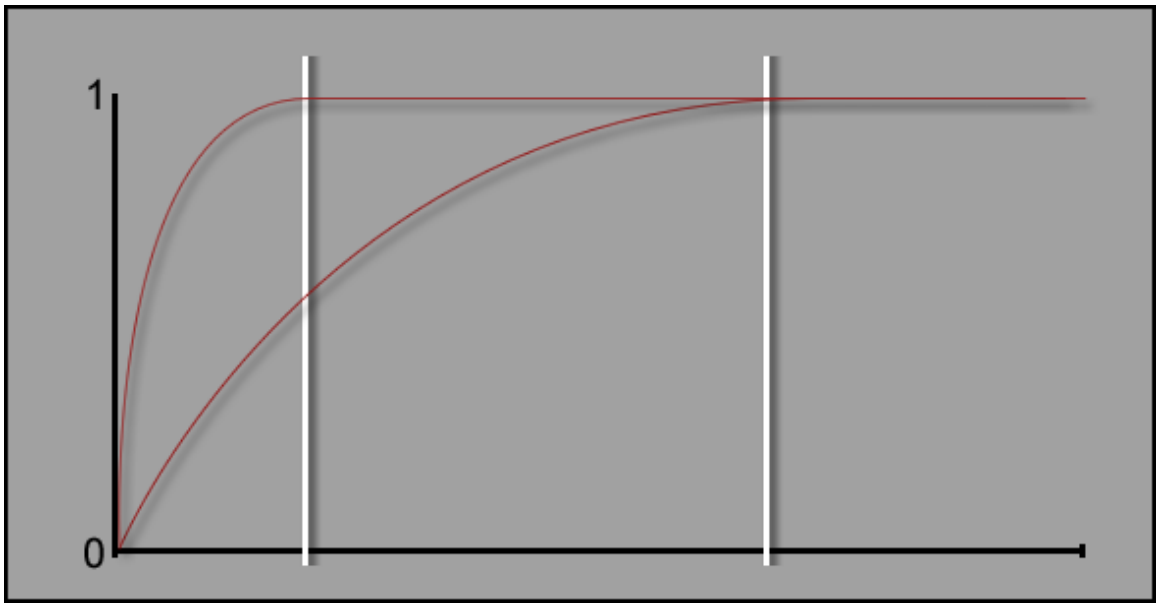
For example, if you chose to use an animation but only need the first section or wish to use multiple animations and want to crop them, setting the duration will override the existing length.



In this example the 3 animations are different lengths. The duration is set to the shortest length to make sure that they do not play till the end of the longest.

Ease In & Ease Out

Ease in and out will allow you to define a period over which the animation will blend in and out to help create a smoother transition. This can be useful when the new animation has extreme motions that will be very different from the base animations.



0 - 1 – is the amount of the animation visible. The white lines represents the time that is defined for the ease in or out.

Name

The name of the state.

This is not referenced by Lua so it for reference only.

Animation Properties

Name

The name of the animation node.

This is not referenced by Lua so it for reference only.

Override Duration

Will allow you to forcibly reduce or increase the length of the animation. By default, the length is determined by the number of frames in the animation file (.ani).

Resource

The Resource animation file (.ani), which is added to the object.

Repertoires – Animation Resources

Animation Resources (.ANI)

All animations are created using the Home Avatar Animation Rig. Animations will be exported are .ani files that will then be added to an object via the Repertoire Editor as resources. These resources will then show up within the repertoire.

- Animation Properties

Animation Types

Skeletal Animations

Skeletal Animations are the primary animation type used in repertoires. Blend tree animations also use skeletal animations by blending them together.

Skeletal animations connect to states (S in red). It is possible to connect multiple animation resources to a single state – which means it can play multiple skeletal animations at the same time.



This can be useful if you want to create a composite animation that uses multiple simpler animations. For example, an upper and lower body animation can be used to create a full body animation.

> ⚠ In order for this to work correctly the animations should be exported with keys on only the joints that are animated. If the same joint is animated in both animation files, the results could be unpredictable.

Blend Tree Animations

Blend Trees blend multiple skeletal animations together. The net result of the blend is then treated as an animation resource that can be assigned to a state. How the animations are blended is controlled by a custom register (value).

It is possible to set this register via Lua to give you direct interactive control of the blend. One register will be needed for each level of the blend tree.



The image above shows the blend tree animation being assigned to the state. Within the blend tree multiple animation resources will be blended together.

Each blend within the blend tree can have up to 8 animations. The result of the blend will be the combination of these animations based on the blend value at any given time.

It is possible to - for example - blend two blends with each other. In effect blending 8 animations with 8 animations to create a brand new animation. This is an extreme example. In most cases 2 or 3 animation would be blended.

Multiple Blends

If you need to blend multiple sub trees together you can create a dummy register and set its value to 0.5. This will result in half of each of the sub trees being used.

Repertoires – Blend Tree Animation

Blend Tree Animations

Blend Tree Animation is a specific type of blended animation.

Blend Tree Animations are just like other animations (e.g. Skeletal animations) - they get assigned to Actions and Behaviors via States. The difference is that the Blend Tree Animation is dynamically controllable.

You can control the blended animation with a custom register (value), your animation can be made up of multiple animation resources blended together, and you can control how they blend.

There are a few things to be be aware of before creating a Blend Tree Animation.

1. Animation assets should be of the same length (number of key frames)
2. Blending animations where all the joints are keyed will often lead to a much reduced motion. This is because you will only see a portion of each animation. Try to only key the joints that each animation will need.

Will It Blend?

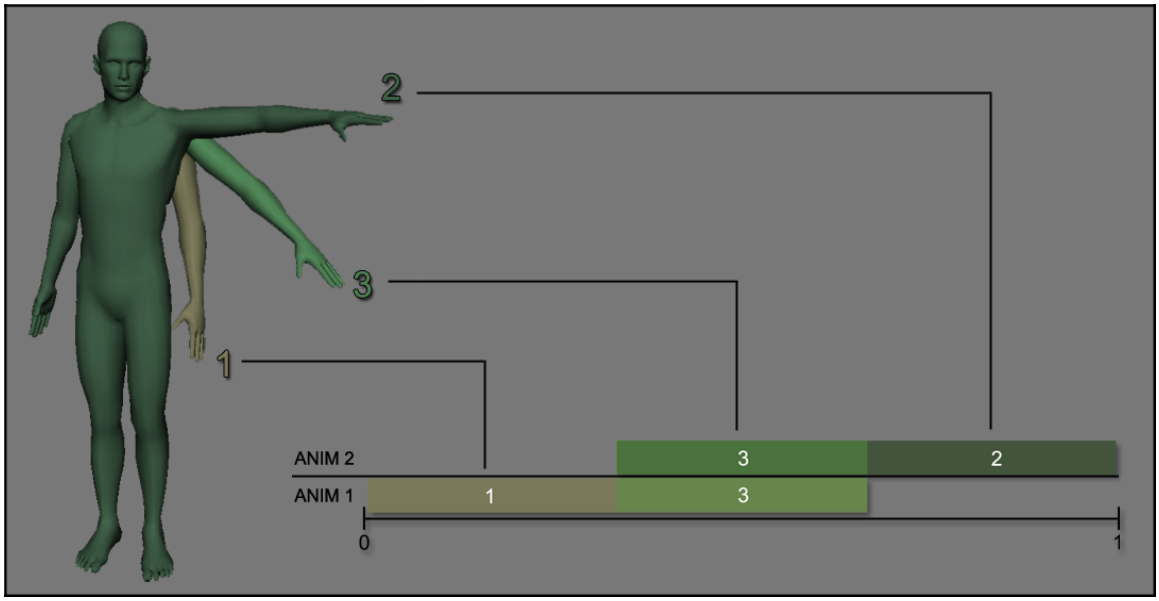
When creating animation resources to be blended you will want to think about which joints will be affected. The main issues to watch out for are many competing joints.

When blending animations it is possible to have more than one animation resource added to the blend, however you can only have two animations blending at any time from a single blend node.

The example below shows animation 1 and 2 blending within the repertoire. The range over which they blend is controlled by the custom register added to the repertoire, in this case 3 represents the blend between anims 1 and 2.

The blend is controlled by the custom register and will have a value between 0 - 1, at 0 only animation 1 will be visible, at 1 only animation 2 will be visible. At 0.5 a blend of 50/50 will be seen, like in number 3 in the image.

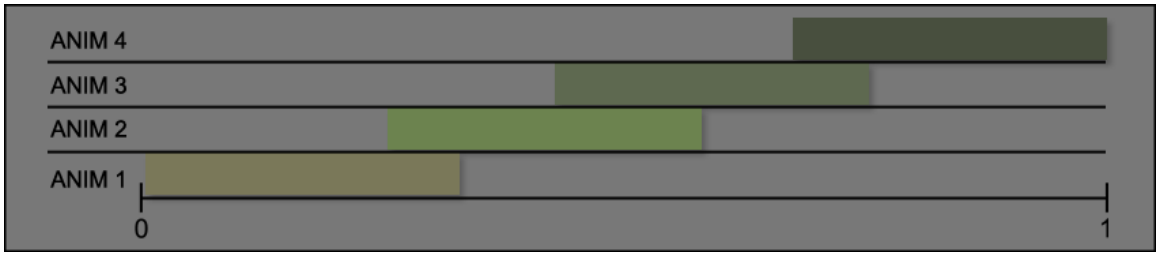
- Detailed Information on Custom Registers



Up to 8 animations can be blended together but only 2 animations can overlap at a time.

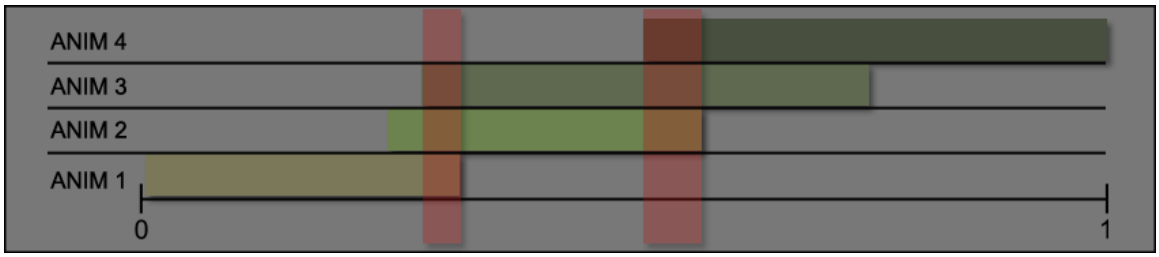
Correct Blending

The image below shows 4 animations blending over the 0-1 range. at any point in the range only 2 animation are blending.



Incorrect Blending

This image shows the same 4 animations but the sections in red show where multiple animations are blending at the same time. This would not be an acceptable blend.



Multiple Blend Animation Blending

It is possible to take a number of blend nodes and then blend them together. This works in exactly the same way as above where only 2 of the blend nodes can be blended together at any time.

It iss best to think about the blend node output as if it where any other animation resource.

When blending blend nodes together, keep in mind you will only get half of each of the results from the previous blends.

Blend Result

The nature of blending means that with each blend your original animations motions and movements will be reduced as you combine the net result of each blend. More often than not you will see 50% of each of the animation. This holds true when blending blend nodes together as well.

To keep this problem from arising, try to use animations that have just the necessary joints keyed in Maya. This will make your animations blend together more easily and prevent them from fighting to all blend the same joints.

Repertoires – Conditions

Conditions

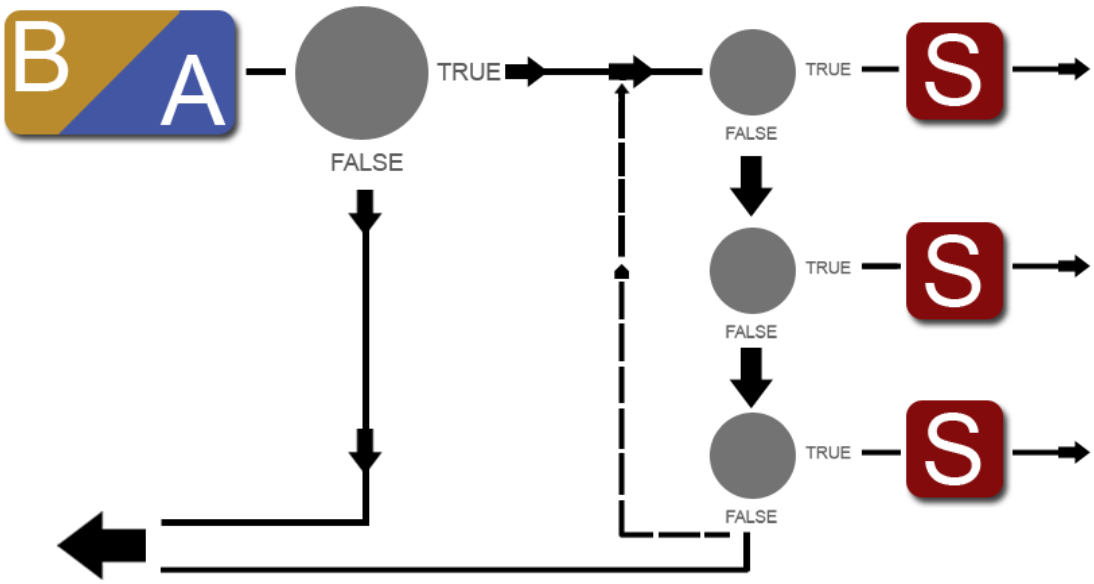
Conditions are used to decide if an Action, Behavior or State can play.

The result of any condition is always a boolean. If the result is True (1) then the animation or state can play, if False (0) then it will not play and the network will continue.

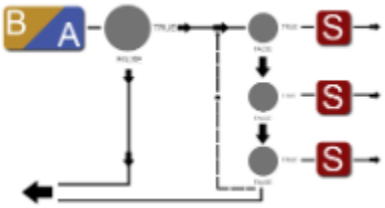
The result of False is often the return to the previous Behavior, in may cases the Idle animation. When False is returned within the State another State will be looked for.

The example below shows the logical flow for conditions. The first condition decides if the Action or Behavior can play at all. If the result is true and the Action or Behavior can play then the next decision is to select which state can play. Each state has its own condition.

Note that all conditions are optional you are not required to have them at any level. Where and when you add conditions is entirely up to you.



Click this thumbnail below for animated version:



Conditions can be placed at two levels within the repertoire.

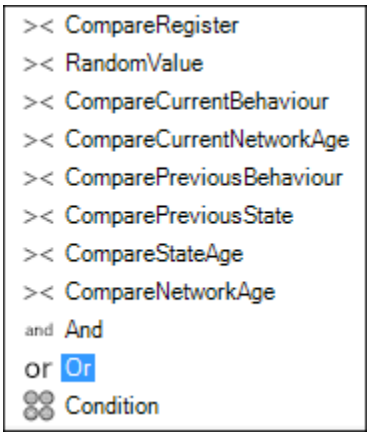
1. Action / Behavior - This will determine if the Action or Behavior as a whole can play or not.
2. State - This will allow you to chose between multiple states, selecting what order they should play in or which one of them is most appropriate to the situation.

Example Conditions

- See here for some example conditions

Condition Operators

These are the available conditional operators:



Concepts

Age - Age will always refer to the length of time the Action, Behavior or State has been playing. When Network Age is referenced it represents how long the network has been evaluated for. For example: with a looping Behavior, you can use Network

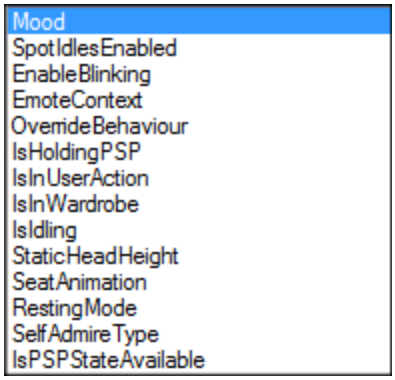
age to get the total time it has been looping.

Current - The Action, Behavior or State this is playing now

Previous - The previously played Action, Behavior or State. This can be used to check what should play next based on the past status.

Register - It is possible to create and then check against a register. There are also a number of registers used in PS Home that you can check against.

Home Registers List



Keep in mind that you may not need to use these registers and that its also possible to create your own *Custom Registers if you are scripting in Lua. They can then be accessed via Lua and changed based on parameters in your game.

Register	Description
Mood	Used to drive the facial animation. Setting the Mood or checking against the avatars mood to detrmin which animation or state should play
SpotIdelsEnabled	Flag checked in core idle behaviors (standIdle, sitIdle), to see if spot idles should be used (ie turning head, shifting feet etc)
EnableBlinking	Flag used by client to see if the blinking should be handled by the client (true), or by the facial animation(false)
EmoteContext	Used to determine which emotes menu is shown (values shown in dropdown)
OverrideBehaviour	The behavior that is used when in scripted or locked control modes- This will be the behavior that is viewed by the remote player.
IsHoldingPSP	Flag set by client when the menu screen is opened (currently used by the idle animation to change to idle pose to show the holding pose)
IsInUserAction	Set by partial body actions to let the idle behaviors know not to start a spot idle (should be set on entry to the action, and reset on exit)
IsInWardrobe	Flag set by client when the user goes into the wardrobe - used by client to disable the spot idles (similar to SpotIdlesEnabled)
IsIdeling	Flag set by core idle behaviors to indicate the user is idle
StaticHeadHeight	Value set by animation networks to give a camera head height (when walking the height goes up and down, so we don't want to track the head exactly)
SeatAnimaiton	Value set by client when sitting - determines which seating animation(s) to use in the core sitIdle behavior
ResetingMode	Value set by repertoire when in idle behaviors, used to determine which resting state you are in : standIlde, sitIdle, sitOnFloor, undefined . used by client to determine when emote menus are available (in conjunction with EmoteContext)
SelfAdmireType	Value set by client when selecting a clothing item, is used by the internal admire action to determine which admire animation to use
IsPSPStateAvalable	Lets the client know if the behavior has a specific psp holding state. ie a FullBodySuit, with their own idle animation, and had a specific animation for holding the menu screen, then this would be set to true on entry into the network (and reset on exit)

Custom Registers

Custom Registers can also be added to objects.

- Custom Registers

Action / Behavior Level

Conditions can be added at two levels within the repertoire. The Action or Behavior or for the stats within the Action or Behavior.

You may want to prevent the Action or Behavior from playing altogether if certain parameters have not been met within a game. For example, during a jump you may want to prevent attack animations from playing. This could be handled in your Lua code but it could also be added to the repertoire and handled with a condition that checks against a custom register.

State Level

Conditions at the State level will help you define which states should play. For example, you may have a several versions of an animation depending on situations like running, sitting or standing. The register can then be checked so that the correct animation state is played for the position of the avatar.

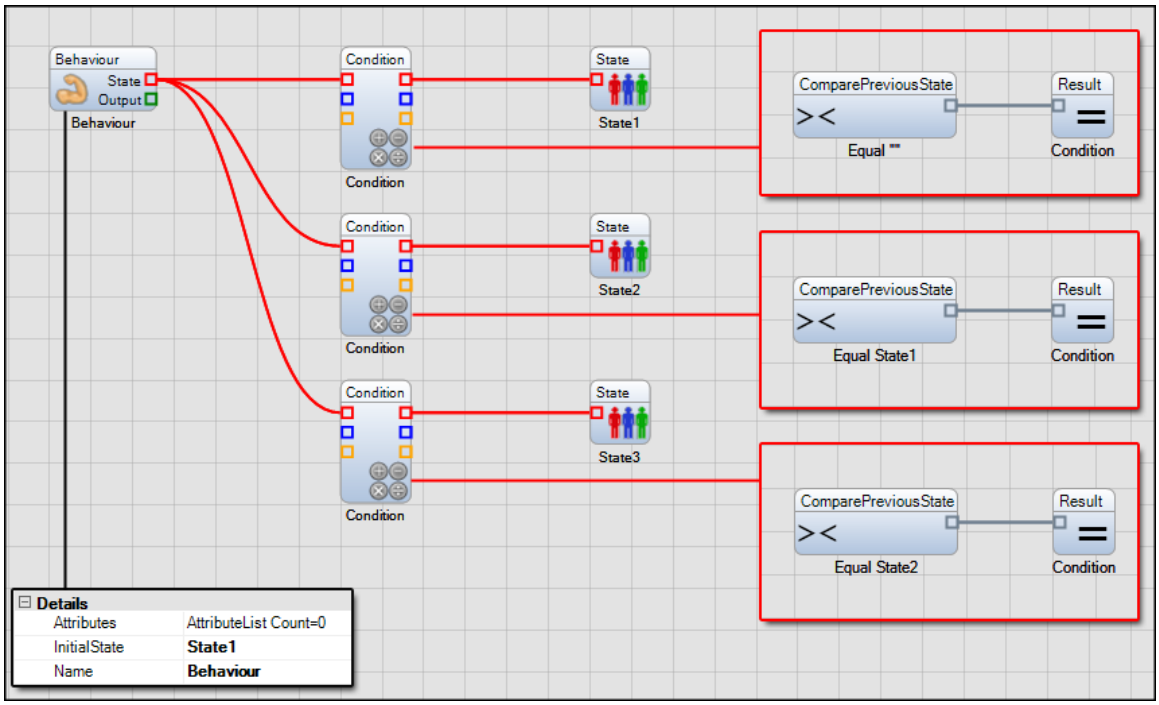
Repertoire – Example Conditions

Play In Order

The example below shows 3 states setup to play in order.

The first state Attribute is set in the behavior node.

A condition is added to each state, which compares the previous state. This forces them to play in order, as they will only play when the previous state is specified.



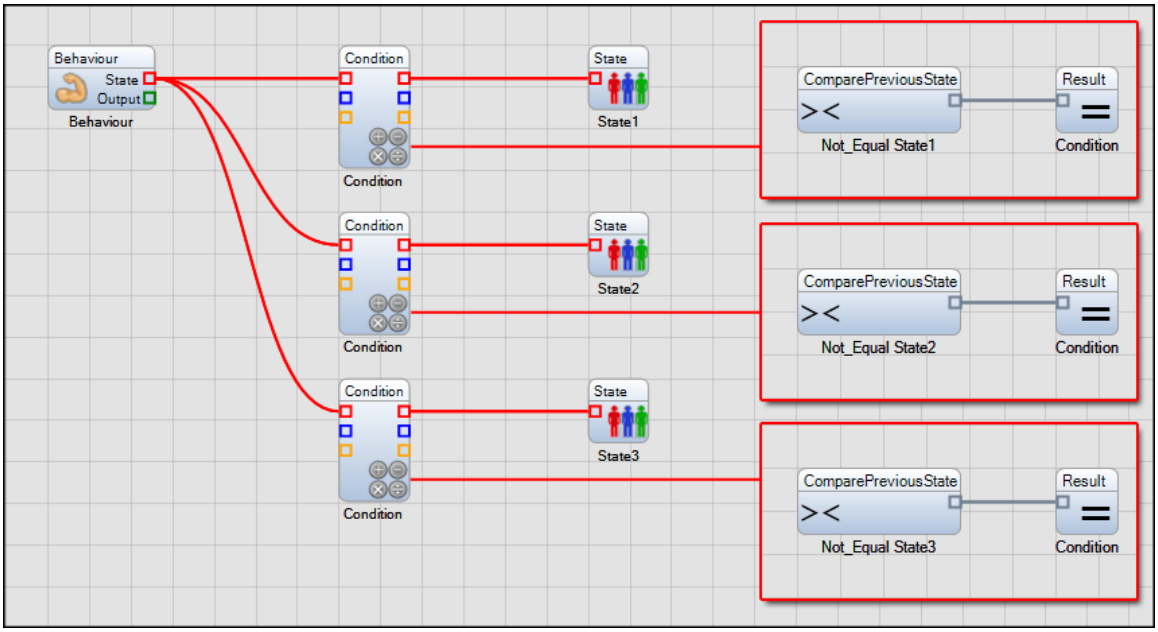
Random But Never Twice

This example shows how to play random states from within a behavior.

> Truly random would mean the possibility of an animation playing twice in a row. But this doesn't seem random, so there is an additional check in place to prevent such a repeat.

In addition to random order, there is a check in place to make sure no animation plays twice in a row. Therefore, each state has a condition that checks the previous state to see if it is the same. This will prevent the same state from playing twice in a row. This is a good example of using the default random behavior and applying some basic control to what is selected.

The repertoire will keep randomly selecting till it is able to play a state by meeting the conditions.



Repertoires – Custom Registers

Custom Registers (Variables)

PS Home contains a number of registers for use within Repertoires.

- Repertoire Register List

It is also possible to add additional registers to your object that can be accessed by the Repertoire and Lua.

Custom Registers are primarily used for:

- Blend Trees
- Conditions

When creating a Blend tree, the blend is controlled with a custom register of the type Number. Range can be set to anything you like but is best set as 0 to 1

Conditions can also use custom registers to help decide if an animation can play. This register can be set by Lua based on something that has happened within a game.

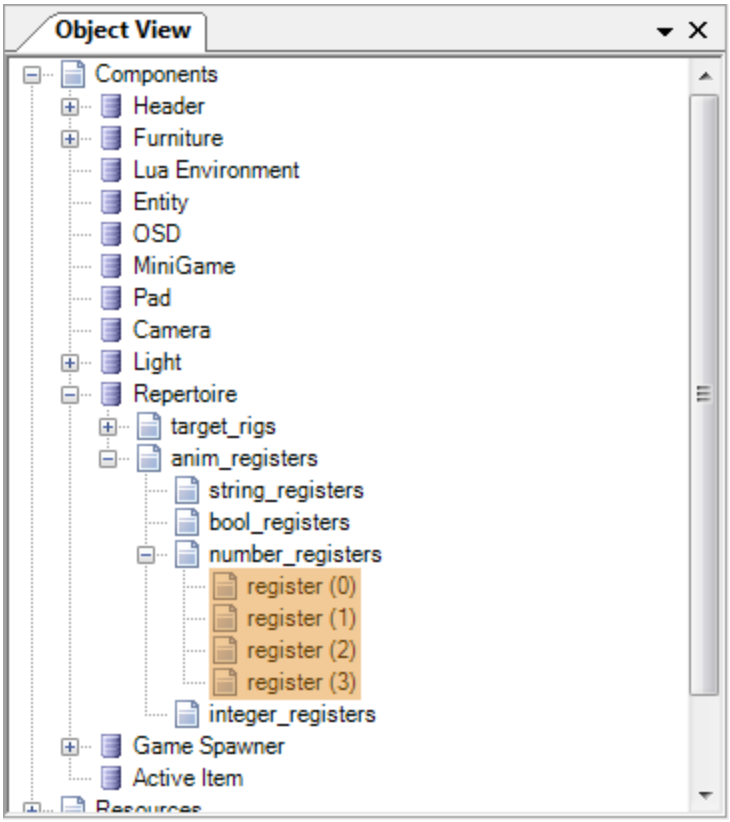
> ⚠ Custom Registers are the only way to pass information into a repertoire from Lua.

Custom Registers are available in the following types:

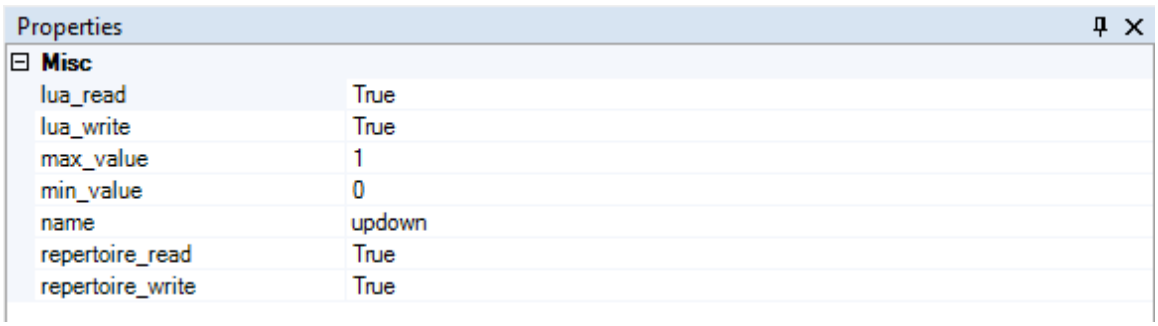
- String
- Bool
- Number
- Integer

To add one to the object go to Object > Add New Register

In this example, 4 custom registers have been added. They are all of type Number and will be used to control an Animation Blend Tree



The following properties are available on the Register.



Lua Read / Write & Repertoire Read / Write

These properties will control which elements of PS Home have access to the register. It is possible for both Lua and the repertoire to set the values for the register.

Min / Max

This is the minimum and maximum value. In the case above, 0-1 is used to control the blend from no blend (0) to full blend (1).

Name

The name of the register. This will be accessible via Lua and will appear within the repertoire listed alongside the default PS Home registers.

Repertoires – Output Nodes

Output Node – On Entry & On Exit

The Output node allows you to take an Action when an animation begins or ends.

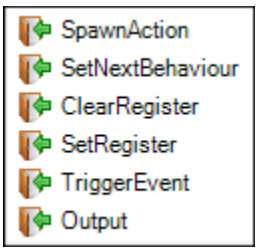
Nodes that connect to an output node will have to be connected to On Entry or On Exit. This determines when the node is evaluated and any commands within it are executed. Most nodes will need to be connected to On Entry, this is because On Exit is after the whole network has been exited and is often too late to execute the commands.

On Entry - On entry to the node network - the network starts with the Action or Behavior node

On Exit - After the whole network has completed, at which point the next network to play (Action or Behavior) has been selected already.

> On Exit will most often only be used to set or clear a register.

Basic Node Types



Above is the list of output nodes. These break down into three groups.

SpawnAction and SetNextBehaviour

These groups are used to control what should play after the current animation.

- SpawnAction - will play the chosen Action when entering the Behavior. Add this node to an Action will cause the Action to be overridden with the new one.
- SetNextBehavior - will set which Behavior should play after the current Behavior has finished. This will not override or force you out of the current Behavior.

ClearRegister and SetRegister

ClearRegister will reset the register to the default value.

SetRegister will allow you to set a value for a register. The register can be a default home register or a custom register added to the object. Custom registers are mostly used for Blend Tree Animations but can be used with conditions as well to control what should play and when.

- Repertoire Register List

TriggerEvent

There are two options for trigger event: SetControlMode *and *SetEmoteContext.

SetControlMode - This option is primarily used to make behaviors loop. The following control modes are available:

- move_normal - The behavior used (either standIdle - for standing - or move) depends on input from the controller, which is handled by the client
- transient - This mode will make the Behavior loop and will stay in that behavior until input is detected from the controller, at which point the control mode reverts to move_normal (used for dances, poses, etc)
- locked - This will lock the avatar playing this behavior and will stay that way until the behavior or control mode is changed (via lua or the client). This is primarily used by the client for the sitting down on seat behaviors (sitIIdle: sit_low, sit_low_recline, sit_mid, sit_mid_recline, etc)

SetEmoteContext - The emote context will control which of the emote menus are visible. This is to control which emotes are available when the avatar is for example sitting.

Action and Behavior Output

Actions and Behavior output nodes are in general used for setting the next Behavior or Action and making the Behavior loop continuously.

- See Output Node Examples

State Output Node

Using the Output node within the state can allow you to set a register or undertake another Action based on the exact State that is chosen.

For example, you may have several states within the Action or Behavior and - based on which state is chosen - you may want to set

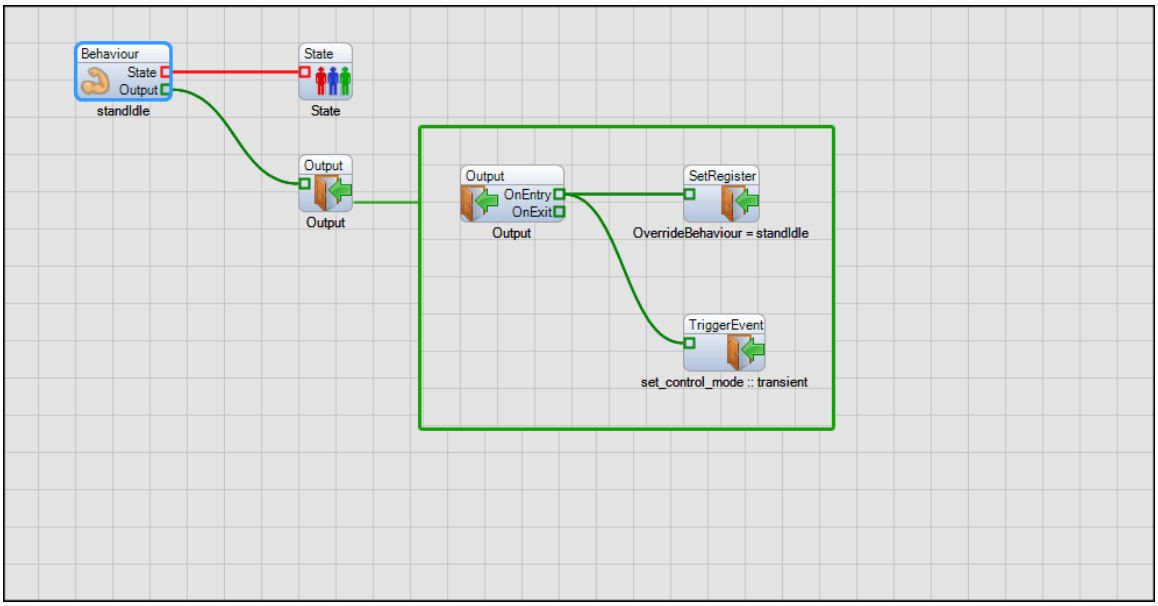
a different next Behavior.

Repertoires – Output Nodes Examples

Looping Behavior

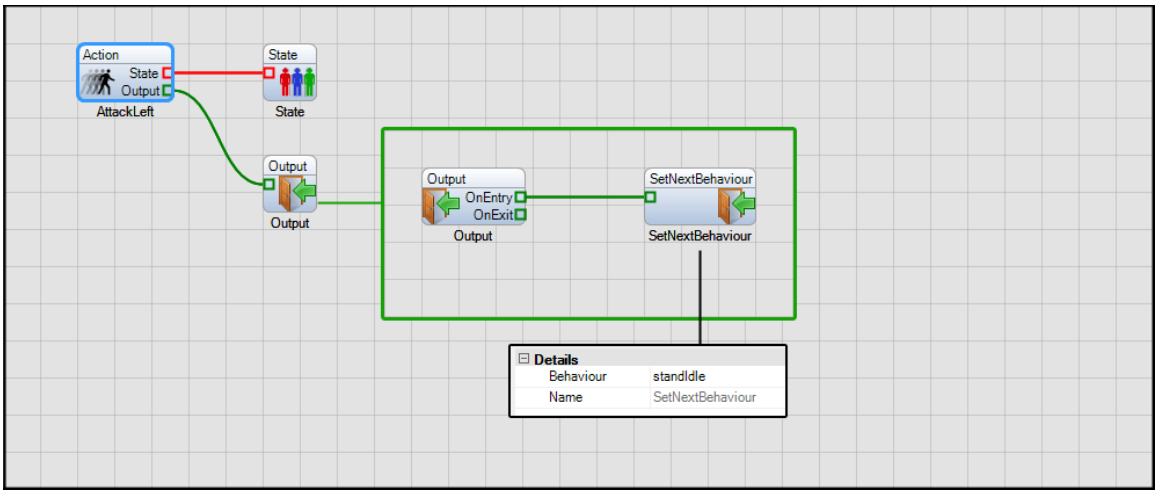
The Behavior has an Output node that contains:

- `OverrideBehaviour = Standidle` - This is the same name as the Behavior that you want to loop. The `OverrideBehaviour` node will make sure that the remote player sees this Behavior.
- `Set_Control_Mode = Transient` - This will make the Behavior loop until the avatar moves at which point it will be interrupted.



Set The Next Behavior To Play

The Output node is added to the Action. It contains a `SetNextBehavior` node with the next Behavior set in the node's properties. The Output is added `onEntry` to make sure the next Behavior is set before it plays.



Repertoire Editor

A Repertoire is a state-based animation system that plays custom animations for an avatar.

You can use the following types of animation in PS Home:

Animation Type	Description
Upper Body	These animations use only the upper half of the skeleton. They can play on top of other animations. For example, you can play an upper body animation while running along. The locomotion still handles the legs running and your upper body animation then plays over the top.
Lower Body	These animations use only the lower half of the skeleton. Unlike upper body animations, lower body animations cannot play on top of other standard animations.
Full Body	These animations use all the joints in the body. They override whichever animation is being played on the avatar when it is called.
Combinations	You can combine Upper and Lower Body animations to produce something like a Full Body animation, but made up of smaller animated components. This approach is more flexible than a single Full Body animation. For example, you can have an upper body animation that is swapped for different actions or behaviors, or one that is used with conditions that choose which upper body animation to play.

You can play a single animation or blend different types of animations to save memory. You can combine an upper and a lower body animation, or play an upper body animation while the avatar is walking or sitting.

See also Avatar Animation Packs.

Repertoire Resources

A repertoire resource is a container for animation networks, which are state machines used to keep track of the animations being played on an avatar. For more information, see Repertoire States.

Repertoire Animations

A repertoire animation allows you to create new avatar animations or replace existing ones.

The HDK supports repertoires for:

- FullBodySuits
- Mini-games
- Realtime games

Custom Animations

When creating custom animation for avatars, you can:

- Have multiple animations
- Replace existing emotes and animations with custom ones
- Control when the avatar is animated

For example, you can have a mini-game that has a different avatar animation for each level that the user completes.

When creating custom animations for avatars in a mini-game, you must have a male and female custom animation, to accommodate avatars of either gender that play the game. For information about naming conventions and overriding defaults, see:

- Repertoire Emotes
- Repertoire Condition Rules

The Repertoire Editor

The Repertoire Editor is a node-based system that you access through the Object Editor. You can use the Repertoire Editor to create state trees for use with mini-game, realtime game, and FullBodySuit custom animation. The Repertoire Editor creates an .xml file containing this data.

To access the Repertoire Editor, your object must have one of the following components, otherwise the option is grayed out:

- Must contain a MiniGame component
- Must contain a RealTime Game component
- Must be a FullBodySuit with a Repertoire component attached to it

Custom avatar animations, and the Repertoire Editor are available only for mini-games and FullBodySuits.

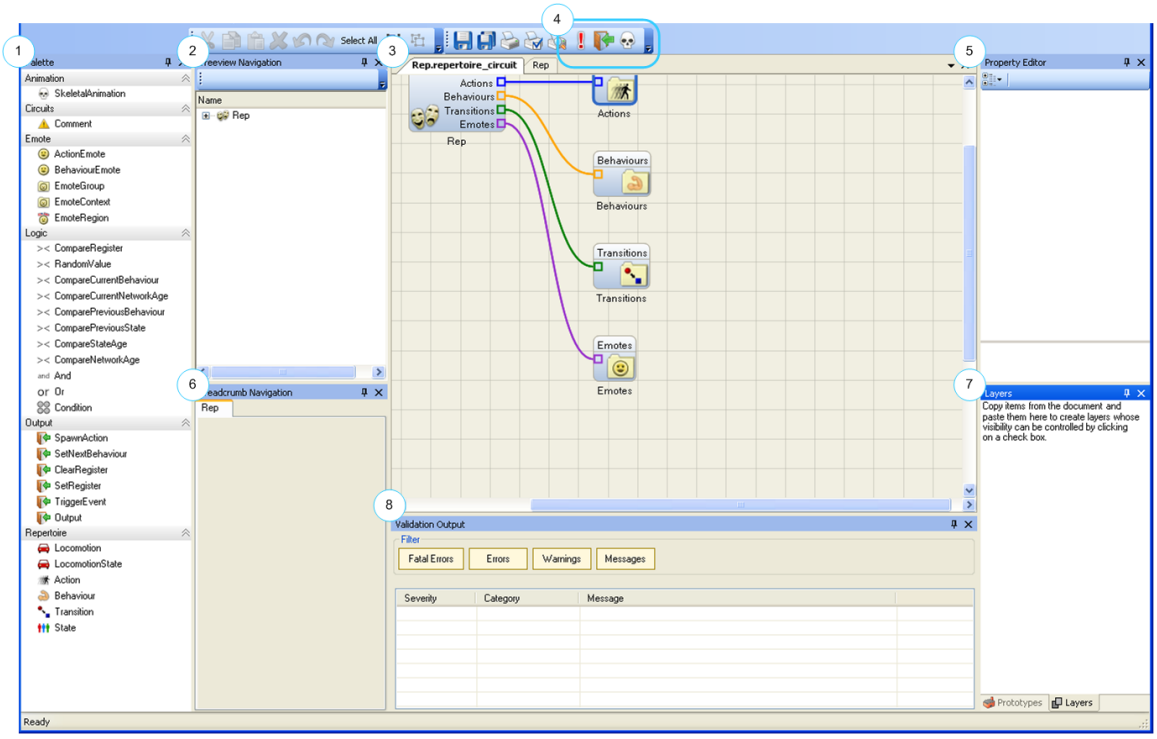
Each repertoire requires three files:

- Two .xml files repertoire and emote that contain the actual data the game will use.
- One circuit file. The circuit file is the working file for the editor.

You can load an .xml file into the editor and save a circuit file. However, use a circuit file created in the tool whenever possible. With a circuit file, you can:

- Add extra data to the repertoire, such as labels.
- Save prototypes for sections, or even whole repertoires.

Repertoire Editor Interface



Item Number	Name	Description
1.	Palette	This is the list of all of the nodes. You drag these into the work area to create/edit Repertoires.
2.	Treeview Navigator	This is a branched view of your Repertoire, showing the Actions, Behaviours, Transitions, and Emotes.
3.	Work Area	The work area is where you drop and end the Repertoire nodes from the Palette. This structure is saved as the Circuit file.
4.	Validate Circuit, Close Circuit, Open Animation Resource Dialog buttons	 Validate Circuit: Validates the current circuit for errors.  Close Circuit: Closes off the circuit.  Open Animation Resource Dialog: Opens the dialog where the animation resources can be specified.
5.	Property Editor	Repertoire node properties appear here when they are selected.
6.	Breadcrumb Navigator	This window shows the previous level of a Repertoire.
7.	Layers	This shows the layers of your Repertoire. Layers are a way to group nodes together.
8.	Validation Output	This is the validation output log. The log is populated upon exporting a repertoire.

Navigation

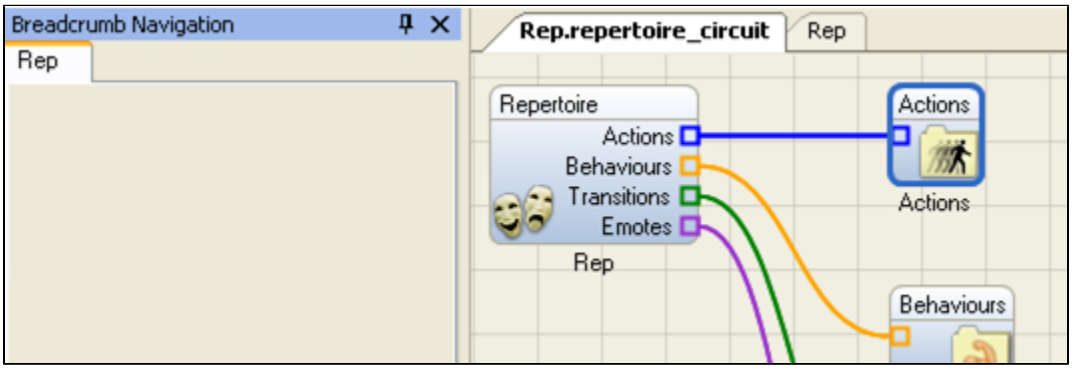
There are four main types of navigation in the Repertoire Editor:

- Work Area
- Tabs
- Breadcrumb
- Tree View

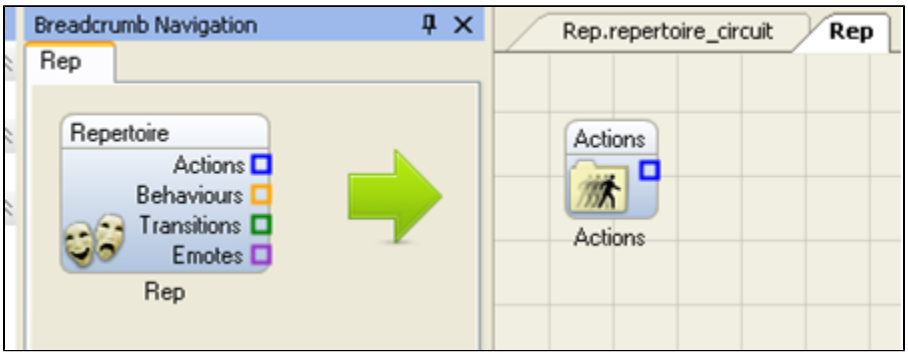
Work Area

The Work Area is not only a form of navigation but also where nodes are created and connections between them are made. You can navigate through the network of nodes by double clicking on them; each double-click takes you into the node and allows you to create new nodes at the next level. The previous levels are always visible in the Breadcrumb Navigation panel.

For example, in the main view, double-clicking 'Action' displays the Actions Work Area:

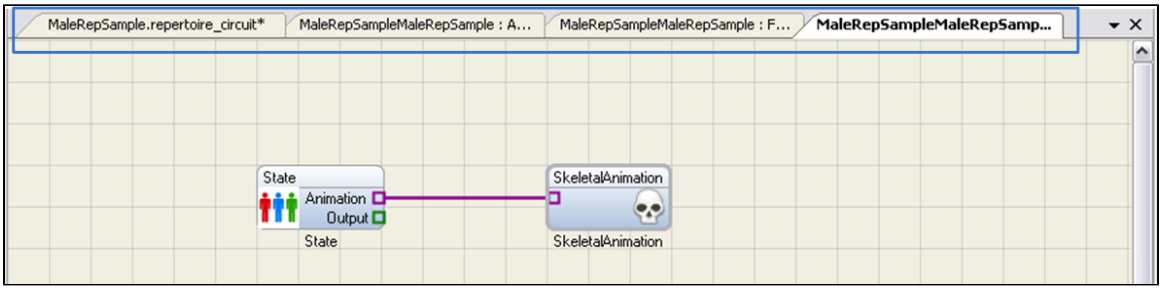


The main view is accessible via the Breadcrumb Navigation panel. In the Actions Work Area you can add new nodes from the Palette. The main Work Area has four groups: Actions, Behaviours, Transitions, and Emotes. These groups will house all of your animations of those types (i.e. the Actions group houses all of the Actions animations). Double clicking on a group will bring you to the next level:



The work area shows the state node you are within and the nodes that are its children.

Tabs



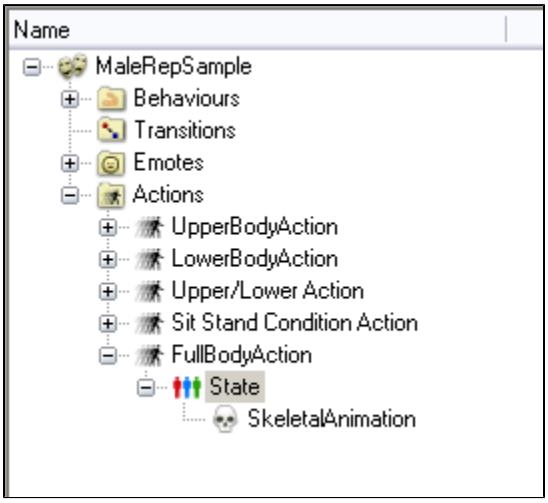
As you progress through the nodes (by double-clicking to proceed to the next level) you will also notice that tabs are created as you step through the Repertoire nodes. These can be seen in the image above.

Breadcrumb



When you access a node using the Work Area, the bar at the top of the screen shows you a new node as well. The breadcrumb navigation tracks your progress. You can use this to jump back up the chain quickly by selecting any of the nodes displayed. The breadcrumb view shows as far as the state node but not the nodes you are editing within it.

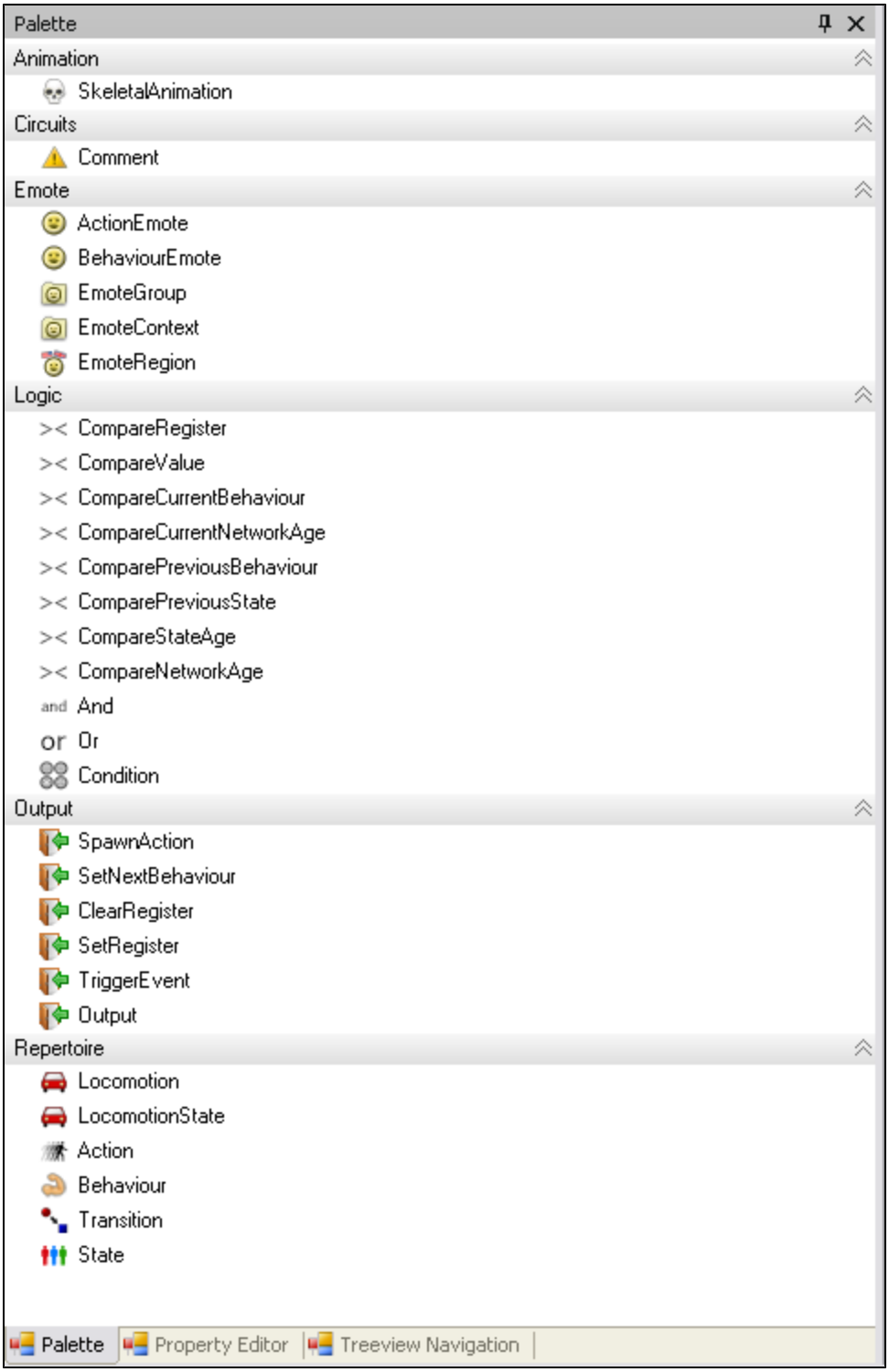
Tree view



A tree view displays all of the nodes in the circuit file. To navigate to a node, double-click the node in the tree view. The tree view allows you to view large sections of the repertoire at once and you can use it to jump to a specific node. The tree view shows the state nodes and its child animation node 'SkeletalAnimation'.

Tools

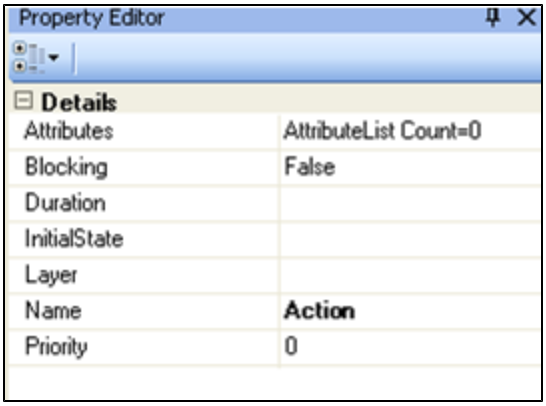
Palette



The Palette is a list of all the available node types. Nodes are created by dragging them from the Palette into the work area. Nodes in the palette are organized by type. Groups can be collapsed by selecting the small double chevron on the right of the title bar.

For more information on the types and uses of the nodes, see Repertoire Editor Node Reference.

Property Editor

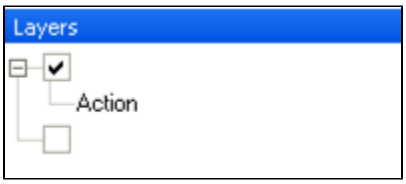


Some nodes have properties that you can edit. To access, select the node and then open the Properties tab.

> By default, this panel is on the left hand side of the Repertoire Editor, in the same area as the Palette.

Each node contains properties relevant to its type. Some nodes do not have any editable properties.

Layers

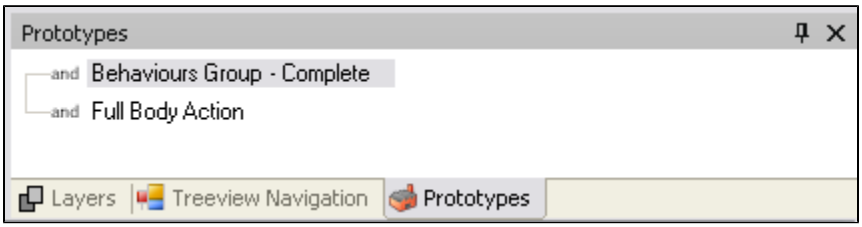


Layers can be used to group nodes together and allow you to toggle visibility within the Work Area. When a layer of nodes is unchecked, they become invisible on the Work Area.

To add Layers:

1. Right-click in the Layers panel and select Add Layer Folder.
2. Copy the node and paste it into the folder.

Prototypes



The Prototypes panel allows you to save a section of, or whole Repertoires, to use later. This works in much the same way as the Windows clipboard, allowing you to then access these useful Repertoire sections to reuse elsewhere.

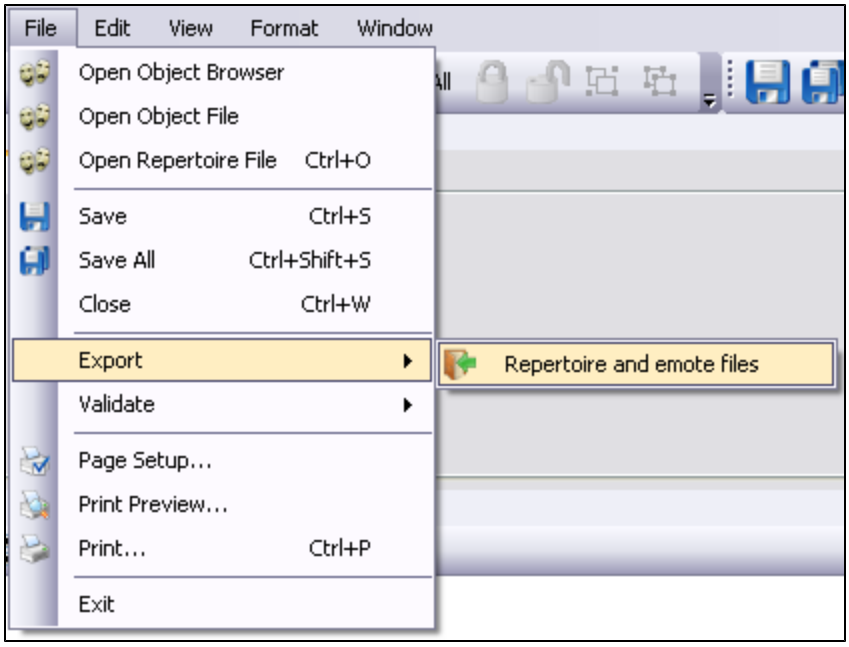
To add a selection to the Prototypes panel:

1. Select the nodes you want to be able to reuse and copy the selection.
2. In the Prototypes panel, paste the selection.

This creates an entry in the list. All of the nodes below the one you copy and pasted will also be dragged into the Prototype panel, with all of the connections intact. By default this selection is named 'Prototype'. To rename it select it in the list and press F2.

Saving and Exporting

When saving your Repertoire, you are only saving the circuit file not the actual game data XML files. In order to save these you will need to select File > Export:



This will then validate the Repertoire for issues and if none are found, export the Repertoire data and add it to your object.

Adding a Repertoire

To add a repertoire:

1. Select the object in the Object Editor or create a new object using the Object Wizard.
2. Select Object > Edit Repertoires.

   The object closes and the Repertoire Editor opens.

3. For new objects that require a Repertoire Component, the Repertoire Component Wizard is displayed (see Using the Repertoire Component Wizard).
4. For existing objects, select the repertoire target rig that you want to edit, for example Male, and then click OK.

   The Repertoire Editor loads the added objects.

Using the Repertoire Component Wizard

When you select Object > Edit Repertoires for an object that requires a Repertoire component, the Repertoire Component Wizard opens.



To create Repertoire and Emote files:

1. Click Next. The following is displayed:



> ⚠ Create your Repertoire and Emote .xml files in your object's folder.

2. For scripted objects, add the full path that the Repertoire and Emote files will be exported to, for the male and female target rigs. These are the .xml files that the Repertoire Editor saves Repertoire data to.
3. For mini-game objects and real-time game objects used in animation blending, add the full path for the Repertoire only (for the male and female rigs).

4. For FullBodySuits, create files only for the FullBodySuit's gender.

> ⚠ You create the .xml files once. Selecting Object > Edit Repertoires displays the Repertoire Editor.

5. When you click Next, the wizard adds the Repertoire component and target rigs to your object.

   The Repertoire Editor then loads the added objects.

Adding an Animation Resource

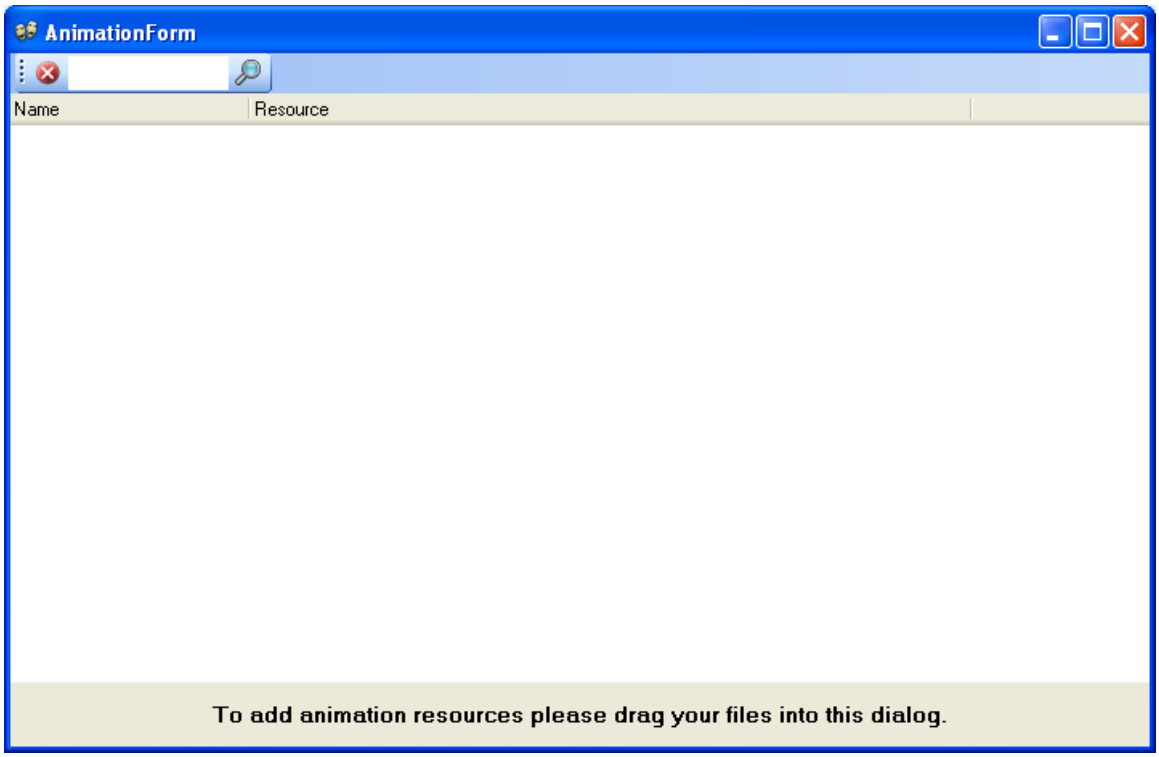
There is no hard limit to the amount of animation resources you can have on an object. Your only restrictions are the object's memory limits.

The Repertoire Editor plays the .ani files through Actions, Behaviours and Locomotions.

Custom avatar animation files (.ani) should not use deferred loading. For more information, see Avatar Animation.

To add a animation resource to the Repertoire:

1. Click  .
2. Browse to and select the animation. You can also drag and drop the animation into the window to add them:

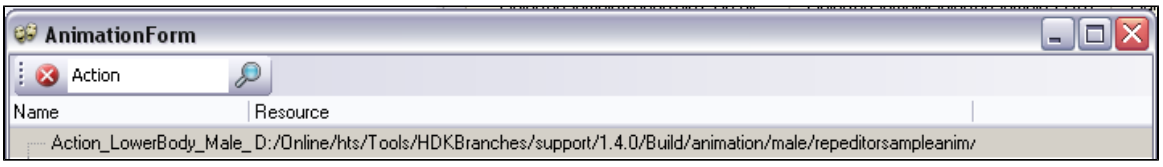


Adding an **.ani** Resource

Drag the animation .ani files from your <HDK_BUILD_DIR> folder and dropping them into the dialog.

Removing an **.ani** Resource

To remove an animation file, use the delete button at the top of the dialog:



You can use the search field to filter animations with a particular name or prefix. The example shows a search for Action.

Repertoire Actions

Actions are single animations, for example a wave or a throw. When an Action is called it plays its animation, and when the animation stops, it returns the avatar to either the idle animation or the animation it was playing before the Action was called.

Using Actions

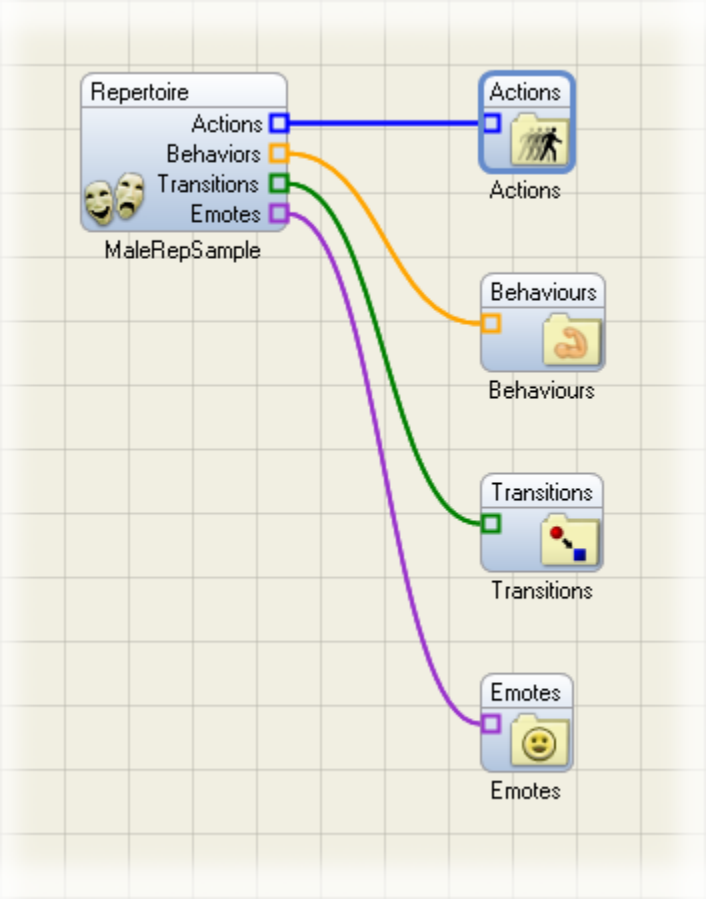
Use Actions where one State is needed, for example a wave. The wave animation plays only once, and it may play over the top of an existing animation such as walking.

> You can have variations of the wave animation to provide variety. This is where Conditions and Logic are used.
> See Repertoire Conditions and Logic.

Use the Lua command `LocalPlayer.DoAction` to call an Action. This command plays a State from the Action taking any conditions into account.

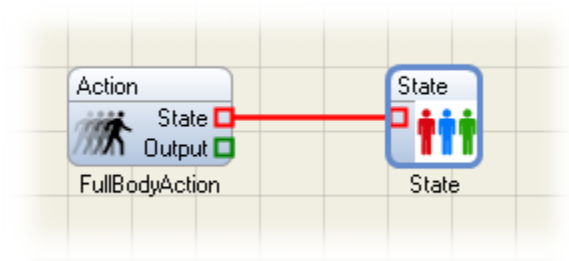
Creating Actions

1. Double-click on the Actions group in the main Work Area.
2. Within the Actions group, drag an Action node from the Palette into the Work Area.
3. Connect the Action group node to the Action node:



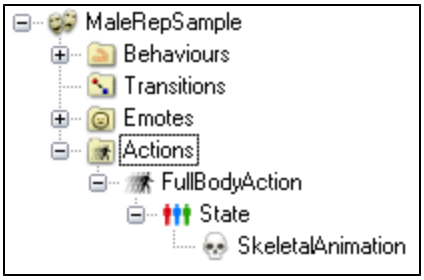
4. Double-click on the Action node to proceed into that Action's Work Area.
5. Add the Animation nodes (see Repertoire States).







Below is the Treeview Navigator view for a basic Action:



In the view above, there are three nodes (the minimum) to the animation: FullBodyAction Action node, the State node, and the animation node. This is the simplest form of animation that you can create. The ability to play an animation depends on where you assign the Repertoire. For example, if it is part of a FullBodySuit and part of the Emote group, you can play the animation from the Emotes menu. However, if you assign the Repertoire to a scripted object, you need to use the game's Lua script and the command `LocalPlayer.DoAction` to play it.

For more information on using Emotes in FullBodySuits, see Repertoire Emotes.

For more information on calling actions from script, see Mini-games and Realtime Games.

Attributes

Action nodes have an Attributes property. When you select the property, it opens the AttributeForm window, where you can add attributes. You can add a value to each action.

> ⛔ You cannot use `getUpFromSeat` as a name for your Action.

Repertoire Behaviours

Use Behaviours to plan animations that you want to loop until a user stops it, for example, a dance. Behaviours, like Actions, can have more than one state. Actions with multiple states play just one State, and then return the user to their idle animation (or whatever animation was playing before the Action was called). Behaviours, continually select from and play all of the different states within the Behaviour, looping through the States until the Behaviour is exited.

Using Behaviours

Use Behaviours for animations that continually loop (an idle animation for example). The Behaviour selects from the States within

it based on Conditions.

> If there are no Conditions then one of the states is randomly selected.

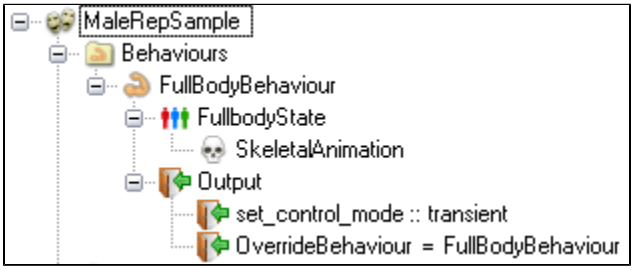
Creating Behaviours

To create a behaviour:

1. Double-click on the Behaviours group in the main Work Area.
2. Within the Behaviours group, drag a Behaviour node from the Palette into the Work Area.
3. Connect the Behaviour group node to the Behaviour node.
4. Double-click on the Behaviour node to proceed into that Behaviour's Work Area.
5. Add the Animation nodes (see Repertoire States).

With these two Outputs now attached to the OnEntry point, the Behaviour plays in a continuous loop until either the user does something (for example, moves), or another Behaviour is triggered.

Below is the Treeview Navigator view of the Behaviour:



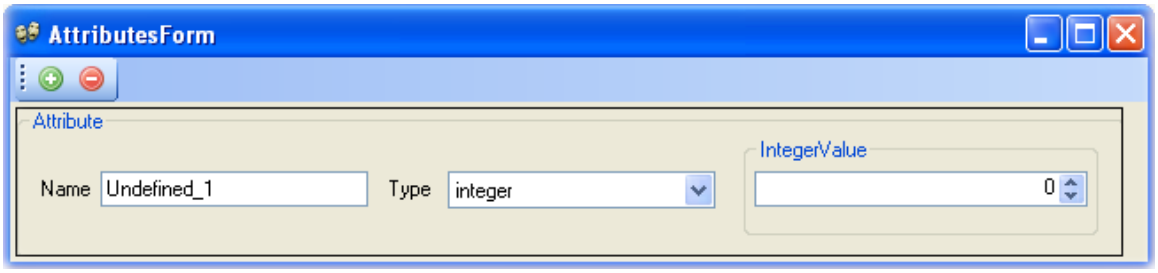
This is a simple form of a Behaviour. The ability to view this in PS Home depends on what you assign the Repertoire to. If it is part of a FullBodySuit, for example, then adding it to an Emote group lets you play it from the Emotes menu. If you assign the Repertoire to a scripted object, you must call the Lua command `LocalPlayer.SetBehaviour`.

Attributes

Behaviour nodes have an Attributes property. When you select the property, it opens the AttributeForm window, where you can add attributes. You can add a value to each behaviour.

For example, in a 'Simon Says' Mini-game that has dancing and jumping behaviorus, you can give each dancing behaviour a 'dancing' attribute value, and each jumping behaviour a 'jumping' attribute value. The Simon Says Mini-game can then boot users who jump when it tells them to dance, or vice versa, by seeing what behaviour they are playing using the function `Person.GetBehaviorCategory()`.





The following fields are available:

- Name: This value must be sys_category.

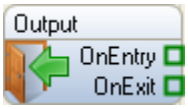
- Type: The type of value you want to `Person.GetBehaviorCategory` to return.
- Value: The value that `Person.GetBehaviorCategory()` returns.

You cannot use the following names for your Behaviours:

- sitIdle
- sit_low
- sit_mid
- sit_high
- sit_low_recline
- sit_mid_recline

Repertoire Editor's Output Node

Use the Output node to control Actions and Behaviours when an Action, Behaviour, or State is initially called or exited.



OnEntry: Nodes attached to this point will be activated when the user enters the Action/Behaviour/State.

OnExit: Nodes attached to this point will be activated when the user exits the Action/Behaviour/State.

Events that can cause a user to enter/exit an Action, Behaviour, or State could be the user triggers it (for example, dance, wave), the user moves, a Condition causes it to end, or the animation finishes playing.

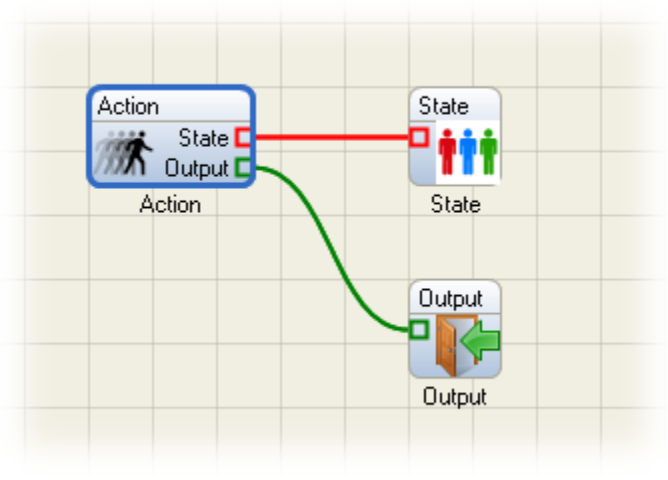
You can add the following nodes to an Output node:

- SpawnAction: Plays an animation .ani file.
- SetNextBehaviour: Sets the next Behaviour to play. Use only with OnExit.
- ClearRegister: Sets the Register to False.
- SetRegister: Sets the Register to True.
- TriggerEvent: Puts the user in a particular mode.

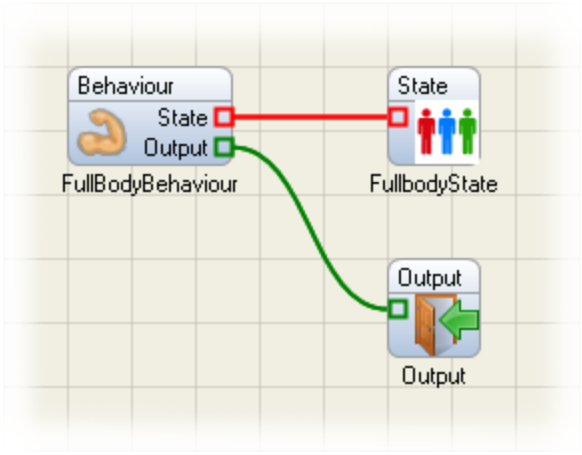
You can also connect multiple nodes into an Output node. A good example of the use of the Output node is within behaviours, both the TriggerEvent and SetRegister nodes are used to make a Behaviour loop continually.

Creating Outputs

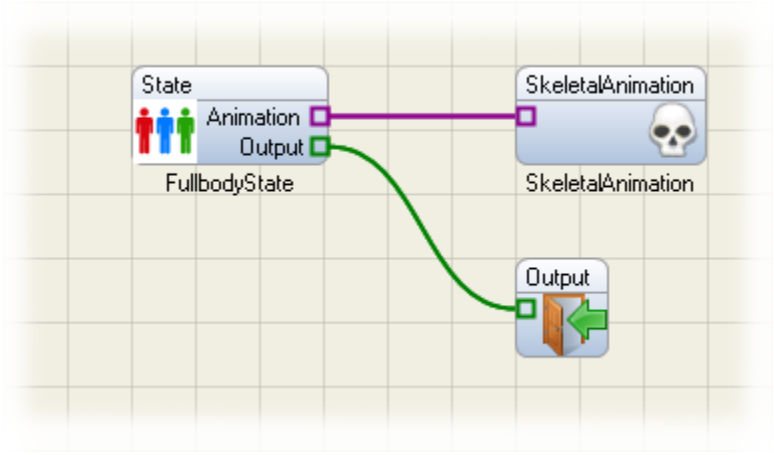
To attach an Output node to an Action, Behaviour, or State, drag an Output node from the Palette into the Work Area and connect the node. The following images show the Output node connected to an Action, a Behaviour, and to a State respectively.



All States within the Action will have the Output applied to them.



All States within the Behaviour will have the Output applied to them.



Only this state has the Output applied to it.

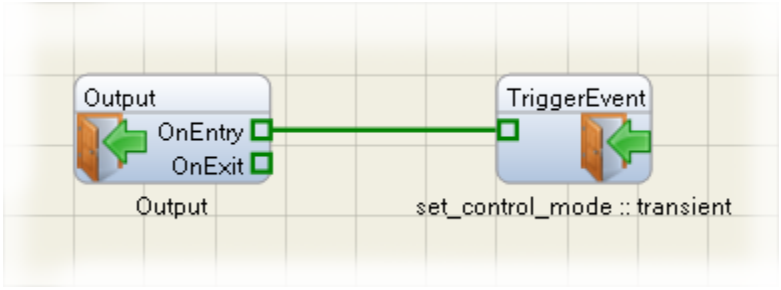
Output Example

This example shows you how to set up Output nodes to make a Behaviour loop.

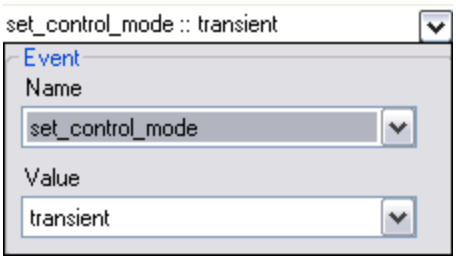
To create a Behaviour that loops and plays continually (for example, a dance), you must add an Output node to the Behaviour, and then edit the Output node itself.

1. From within the Behaviour Work Area, add an Output node from the Palette.
2. Double-click on the Output node to enter the Output Work Area.
3. Add a TriggerEvent node from the Palette, and connect it to the OnEntry point.

   OnEntry is connected to events you want to occur upon calling the Behaviour.

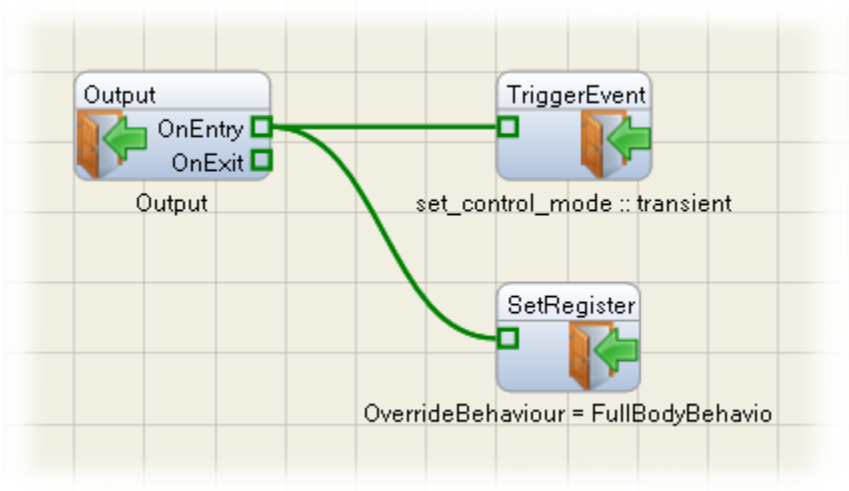


4. In the TriggerEvent's Properties > Event, set the set_control_mode to transient.



Setting the control mode to transient will makes the Behaviour play continuously until it is interrupted by another Behaviour or user movement or input.

5. Add a SetRegister node from the Palette. Connect this node to the OnEntry point as well.



6. In the SetRegister Properties > Register, put the Register as OverrideBehaviour and the StringValue to the name of the Behaviour, for example, FullBodyBehaviour.

Repertoire Conditions and Logic

When there are several States within an Action or Behaviour, you can use Conditions to tell PS Home which State to play. The parameters that you can use to make conditions can vary from the last state or Behaviour played to querying if the character is sitting or standing. Due to the large list of possible parameters this section will instead show you the set-up of a simple condition. The example should give you a basic idea of how conditions work.

If all Logic nodes return false, the Condition returns false. When a Condition returns false, the Action, Behaviour, or State that it is attached to does not play.

For a full description of each condition type, see Repertoire Editor Node Reference.

You can use conditions for the following tasks:

- Make an Action or Behaviour with multiple states play in a certain order (for example, play animation 1, then play animation 2).
- Choose which animation should play based on the user's Mood.
- Play animations randomly, but never play the same animation twice.
- Choose a State to play based on whether the user is sitting or standing.

Creating a Condition

The condition in this example is of a Behaviour with three different states. The States in the condition play in the following sequence:

1. No 1 BehaviourState
2. No 2 BehaviourState
3. No 3 BehaviourState
4. Repeat

Setting up the Work Area

1. Use the steps in Repertoire Behaviours to create a Behaviour. In this example, the Behaviour is called Condition Sequence Behaviour.

2. Add the following states to the Behaviour:

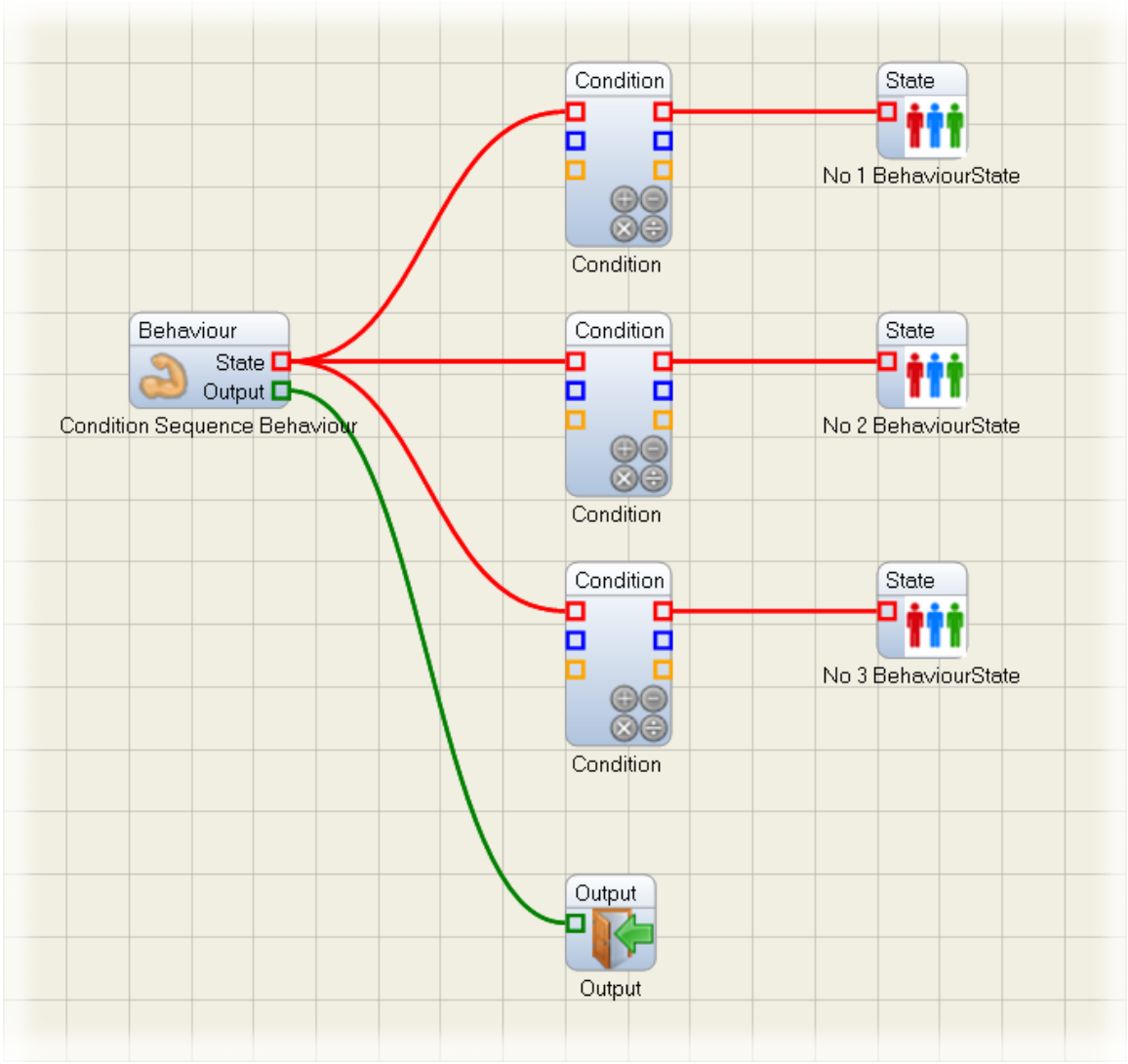
    - No 1 BehaviourState
    - No 2 BehaviourState
    - No 3 BehaviourState

3. Attach three different animations to the Behaviour:



4. Drag three Condition nodes using Palette > Logic and drag an Output node onto the Work Area.
5. Connect the State box on the Behaviour to each of the Conditions, and then connect the red box on each Condition to one State.
6. Attach the Output box to the Output node.

Your Work Area should look this:



Editing Conditions

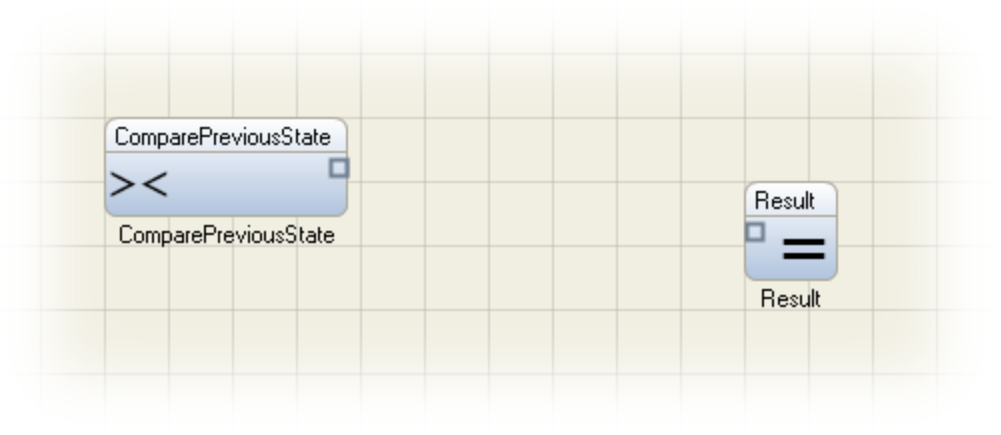
First State's Conditions

The first conditions, those applied to No 1 BehaviourState tell PS Home to play this State if one of the two conditions is true:

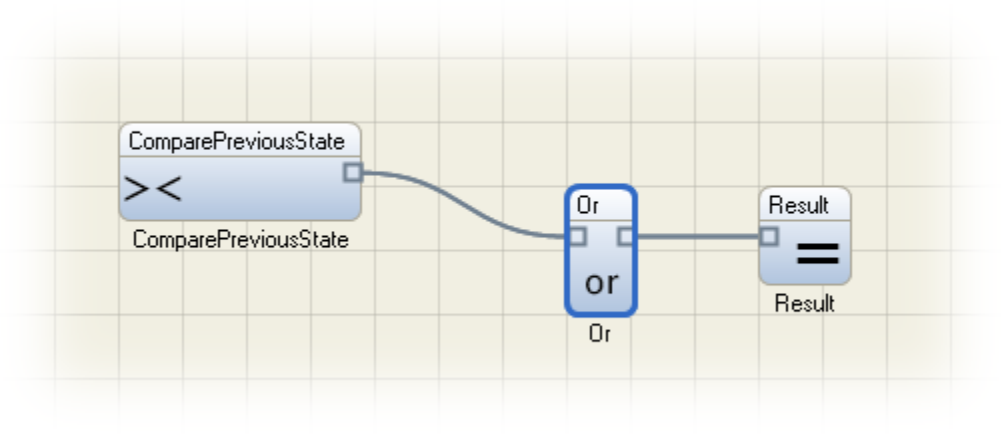
- The Behaviour is called.
- No 3 BehaviourState finished playing.

To set up this condition:

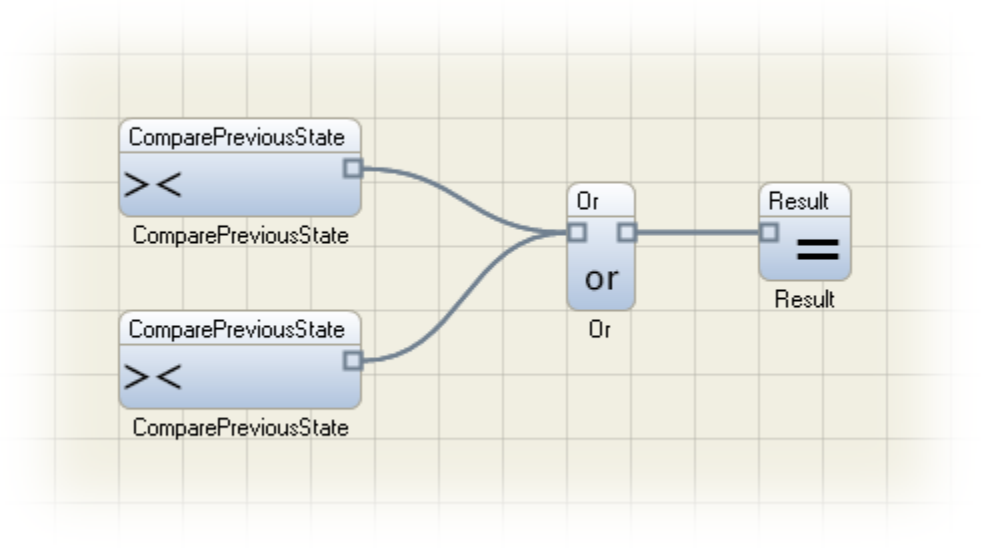
1. Enter the first condition's Work Area (the condition connected to No 1 BehaviourState) by double clicking on it.
2. Inside the Work Area, drag a ComparePreviousState node from the Palette > Logic:



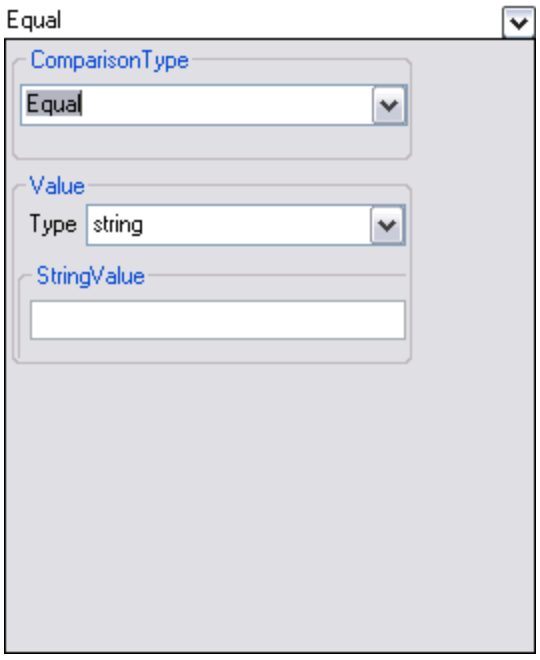
3. Drag an Or node from the Palette > Logic.
4. Connect the ComparePreviousState node to the Or node, and then the Or node to the Result node.



5. Add another ComparePreviousState node, and connect it to the Or node as well.

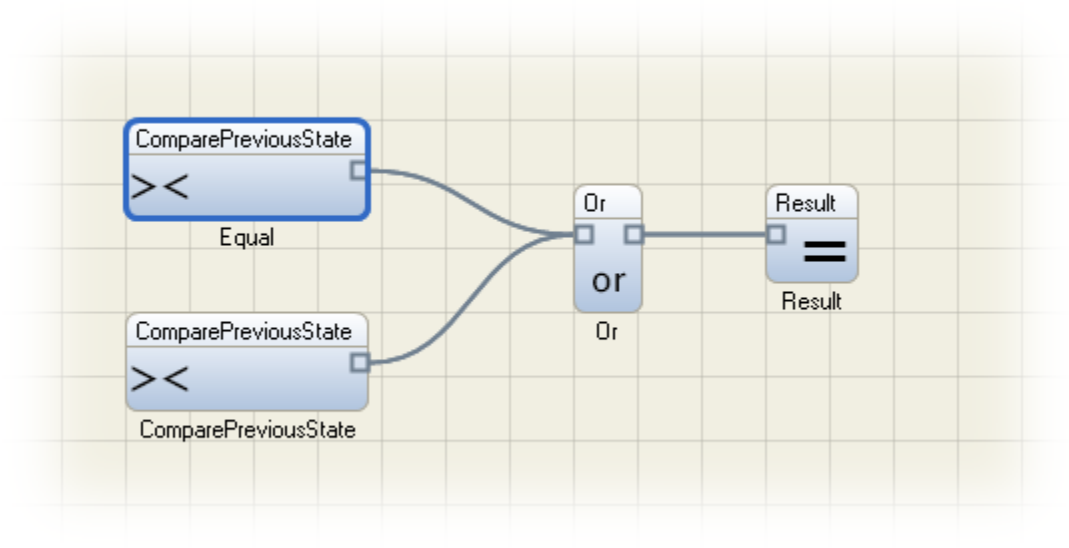


6. Select the top Logic node > Property Editor > Comparison. A dialog is displayed.
7. Set the ComparisonType to Equal, Value Type to string, and leave StringValue blank:



This Logic checks to see if the previous state returns no value. This only happens when entering a Behaviour, because no state within the Behaviour could have been played when a user enters it. Since there was no previous state, this ComparePreviousState Logic returns True.

The name of the top Logic (the one just edited) will change in the Work Area to reflect its ComparisonType:

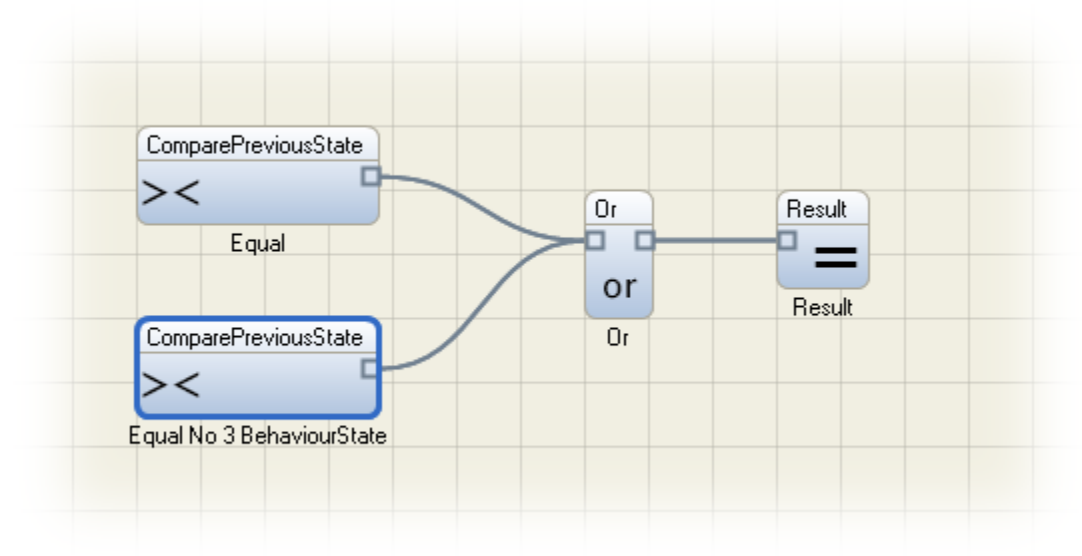


8. Select the bottom Logic > Property Editor > Comparison. A dialog is displayed.
9. Set the ComparisonType to Equal, Value Type to string, and StringValue to No 3 BehaviourState:



This Logic checks to see if the previous state was No 3 BehaviourState; if it is, the Logic returns True. This creates a loop. As soon as No 3 BehaviourState has finished playing, it plays No 1 BehaviourState.

The name of the bottom Logic (the one just edited) will change in the Work Area to reflect its ComparisonType and StringValue:



10. The Condition for the first state (No 1 BehaviourState) is now finished. Return to the previous Work Area to work on the second Condition for the second State.

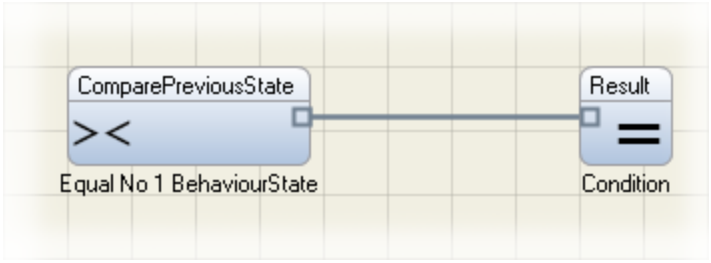
The Or Logic node needs an input value to return True. When checking Logic nodes, it starts with the top-most node in the Work Area, and works down checking to see if any Logic node returns True.

In this example, the Or node checks the top Logic (checking to see if there were no previous States played), and if the top Logic returns False, moves down to the bottom Logic. When the Or node receives True, it passes True to the Result node, which then plays the State .

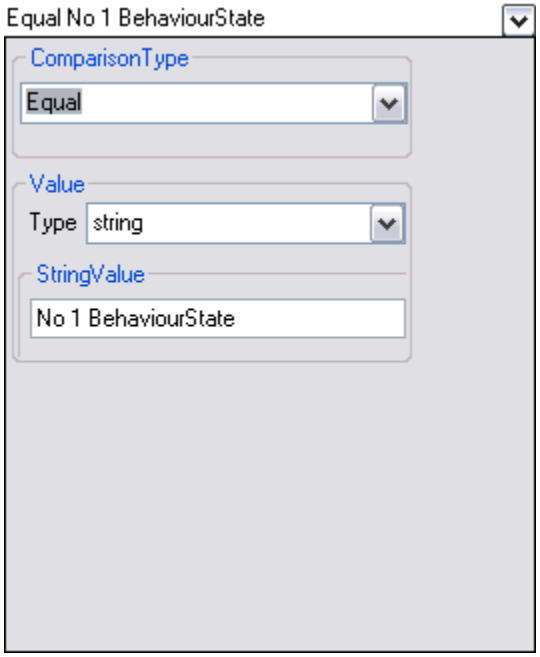
Second State's Condition

Return to the Behaviour Work Area, which has the three States each attached to a Condition.

1. Enter the second condition's Work Area (the condition connected to No 2 BehaviourState) by double clicking on it.
2. Add a ComparePreviousState node and connect it directly to the Result node.



3. Select the Logic node > Property Editor > Comparison. A dialog is displayed.
4. Set the ComparisonType to Equal, Value Type to string, and StringValue to No 1 BehaviourState.



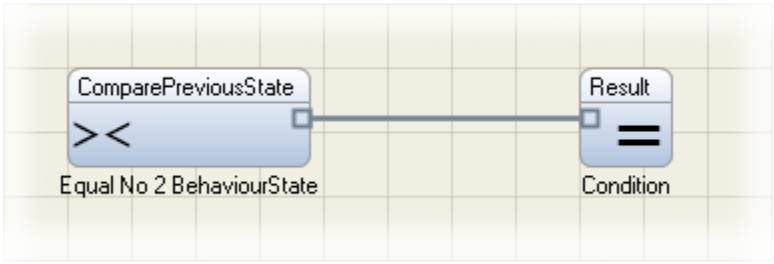
5. This Logic checks to see if the previous State was No 1 BehaviourState, and if it was returns True.
6. The Condition for the second state (No 2 BehaviourState) is now finished. Return to the previous Work Area to work on the third Condition for the third State.

There is only one condition. It checks to see if the previous State was No 1 BehaviourState, and if so plays No 2 BehaviourState.

Third State's Condition

Return to the Behaviour Work Area, which has the three States each attached to a Condition.

1. Enter the third condition's Work Area (the condition connected to No 3 BehaviourState) by double clicking on it.
2. Add a ComparePreviousState node and connect it directly to the Result node:



3. Select the Logic node > Property Editor > Comparison. A dialog is displayed.
4. Set the ComparisonType to Equal, Value Type to string, and StringValue to No 2 BehaviourState:



This Logic checks to see if the previous state was No 2 BehaviourState, and if so plays No 3 BehaviourState.

5. Conditioning for all states is now complete.

There is only one Logic node. It checks to see if the previous State was No 2 BehaviourState, and if so plays No 3 BehaviourState. For the entire Behaviour, when Condition Sequence Behaviour is called the States will play sequentially, and then loop continuously in the order created by the Conditions until the user does something to break the Behaviour (for example, move).

Repertoire Condition Rules

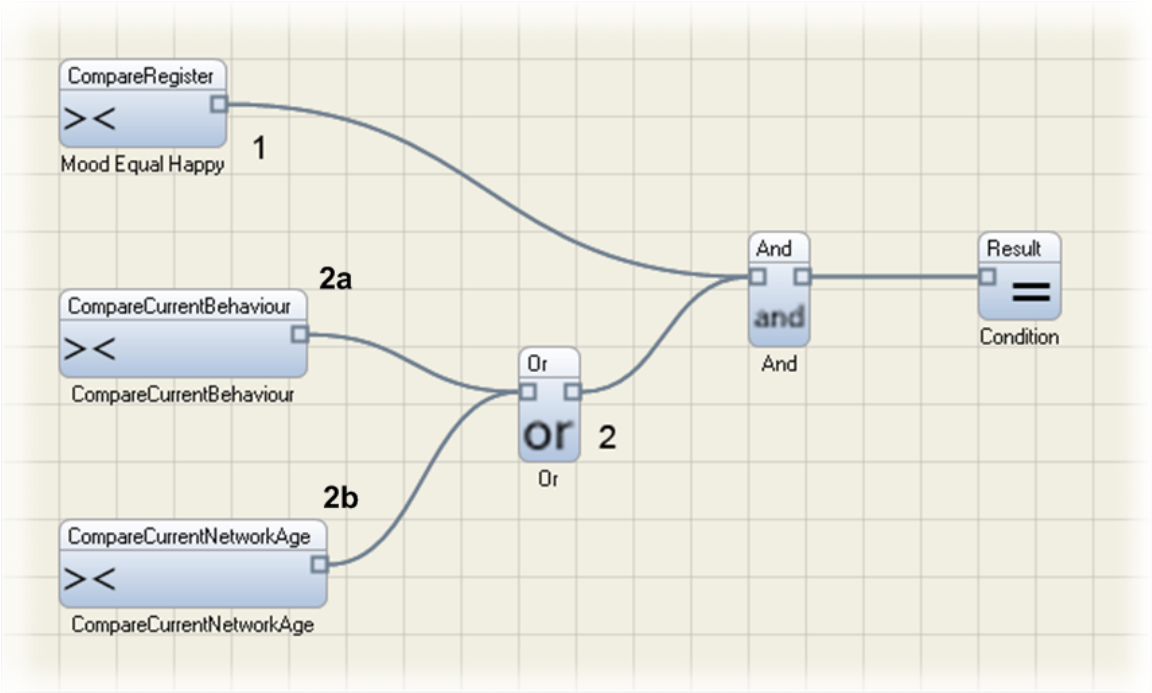
Conditions and States

Bear the following in mind when working with conditions and states:

- If all Logic nodes return false, the Condition will return false. When a Condition returns false, the Action, Behaviour, or State that it is attached to will not play.
- Conditions have a one-to-one relationship with States. Only one condition can feed into any one State.
- One condition cannot feed into two states. For example, a condition that checks if the user is sitting cannot feed into two separate states that need to know if a user is sitting. Instead, two conditions that each check if a user is sitting.
- Two conditions cannot feed into one State. Instead, within the condition, use the 'And' Logic node to connect two or more other Logic nodes.

Calculation Order

It is good practice to optimize larger, more complex Conditions. The layout of the Logic nodes in the Work Area affects how and when the nodes are calculated. The Repertoire Editor outputs the nodes into the .xml files in order from top to bottom.



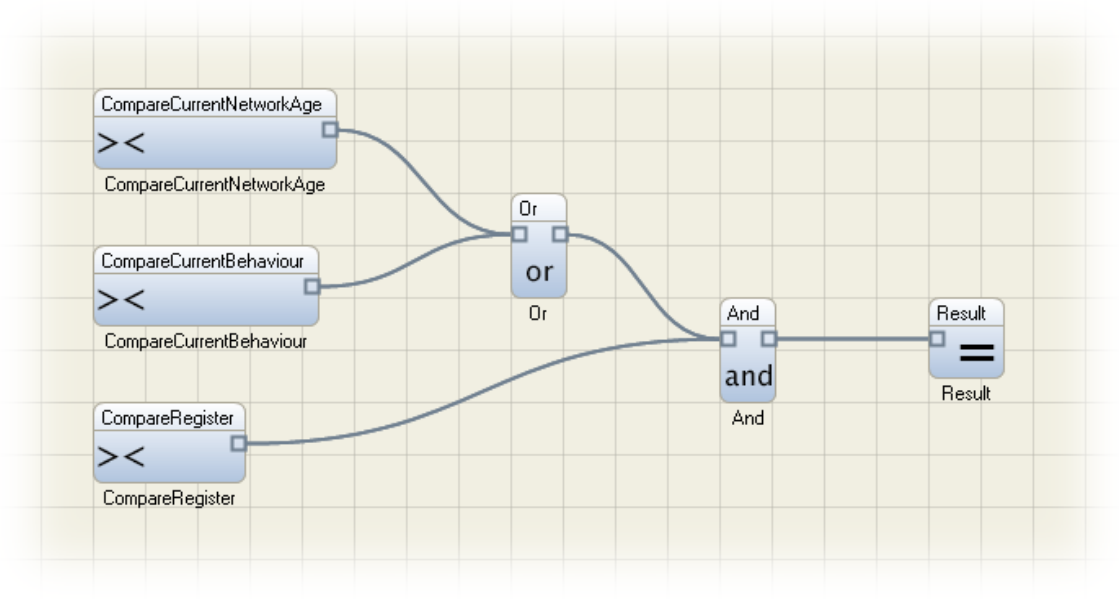
The Condition computes the Logic in the following order:

1. Mood Equal Happy
2. CompareCurrentBehaviour (Only if Mood Equal Happy returns True)
3. CompareCurrentNetworkAge (Only if CompareCurrentBehaviour returns False)

Why this condition is optimized

Regardless of the arrangement of the Logic nodes, Mood Equal Happy must return True to play the state. If Mood Equal Happy returns False, there is no point in computing the other Logic nodes.
If CompareRegister returns False, the maximum amount of calculations will be 1.

The image below shows a less efficient arrangement:



The Condition computes the Logic in the following order:

1. CompareCurrentNetworkAge
2. CompareCurrentBehaviour (Only if CompareCurrentNetworkAge returns False)
3. CompareRegister (Only if one of the two previous states returns True)

Why this Condition is not Optimized

Regardless of the arrangement of the Logic nodes, Mood Equal Happy must return True to play the state. In this case, up to two calculations are made before the CompareRegister Logic node is calculated.

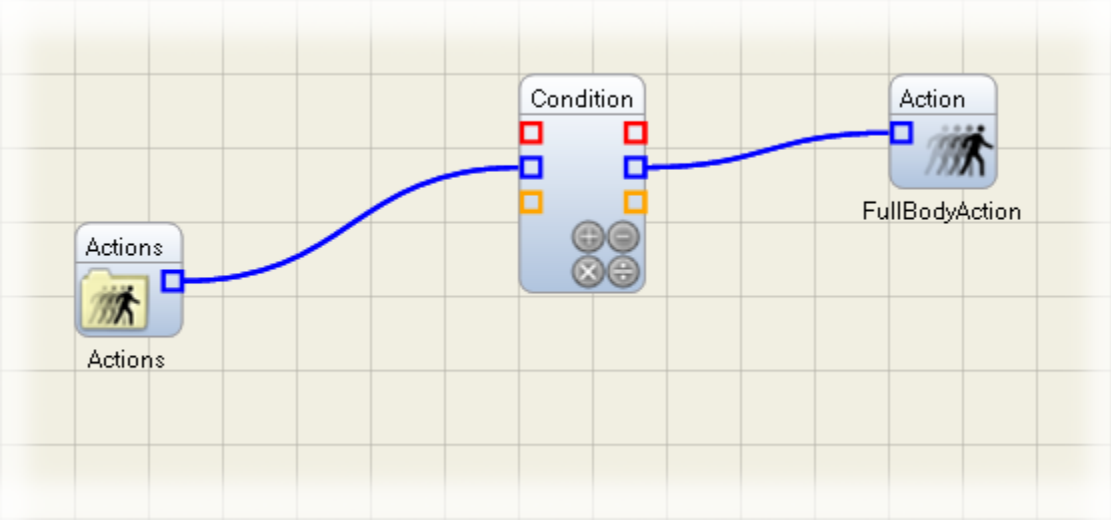
If the CompareRegister node returns False, the minimum amount of calculations will be 2, the maximum will be 3.

> By putting the single Logic at the top (as shown in the first image), the Condition is more efficient, and the minimum calculations are fewer.

Ensure you optimize your Conditions by limiting the number of Conditions used for each Action, Behaviour, or State. This helps to reduce the processing time and to make your animations run smoother.

Conditions Outside of Actions and Behaviours

You can have a condition outside of an Action or Behaviour as shown in the following diagram:

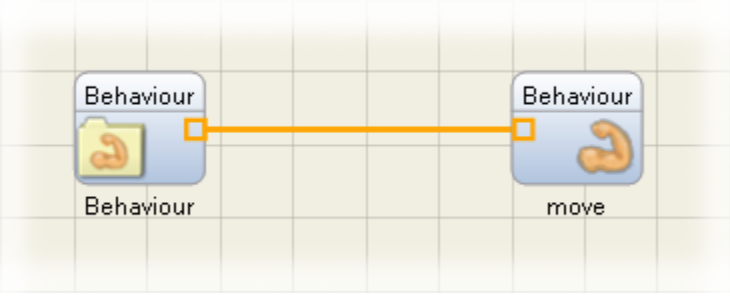


The Condition is attached to the main Actions group and to a specific Action node. The Condition works in the same way as a Condition within a State. However if the Condition returns false, the Action and all of the States within it do not play.

Repertoire Locomotions

Locomotions enable you to replace the default walk animations of the avatar. Locomotions are Behaviours, but need specific names to override the standard locomotion animations. They are also different from standard Behaviours, because they are triggered by the movement of the avatar.

The most common locomotion is named 'move' (case sensitive). The image below shows a Behaviour named 'move'.



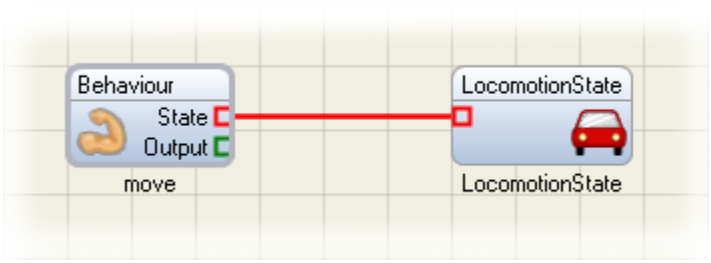
Locomotions consist of 3 different animations (for example, 'Walk', 'Jog' and 'Run'), that play depending on the position of the PlayStation DUALSHOCK®3 analogue stick. The locomotion controls how and at what positions on the analogue stick the animations should blend and swap.

Creating Locomotions

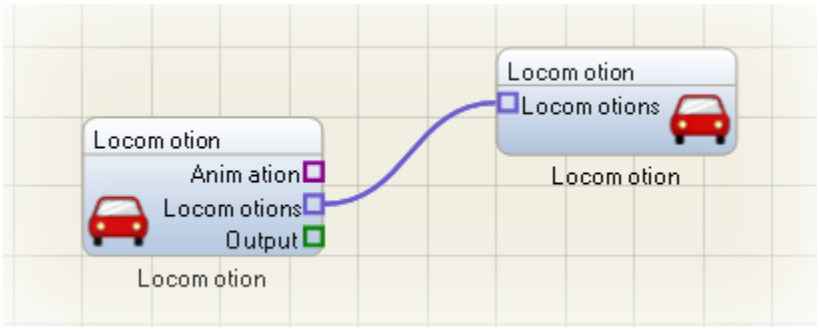
Locomotions are Behaviours with LomotionStates within them.

To create a Locomotion:

1. Follow the steps to creating a Behaviour, stopping at the stage where you would add a State.
2. Add a LocomotionState from the Palette (instead of a standard state):

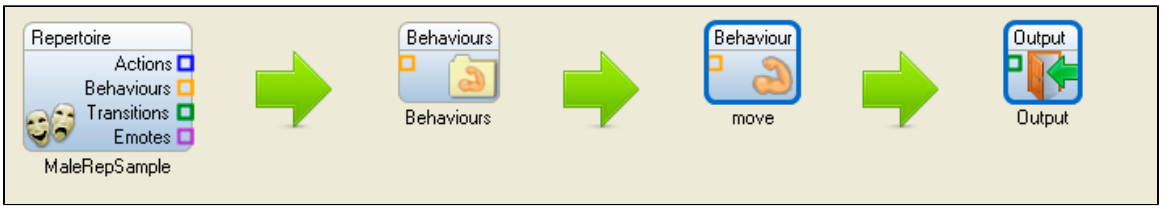


3. Double-click to enter the LocomotionState Work Area.
4. Add a Locomotion from the Palette:



> The Animation connection exists only as a legacy feature, allowing you to open old repertoire files. If there are Animation nodes in your Locomotion, delete them. Replace Animation nodes with Locomotion nodes.

5. Select the Locomotion.
6. In the Property Editor, change the name (if desired), set the animation resource, adjust the speed and threshold (explained below).
7. Add an Output node to the Behavior node:



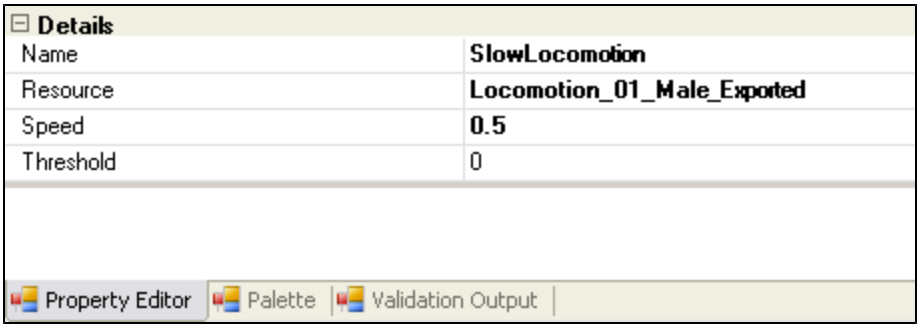
8. Add a TriggerEvent node and set the TriggerEvent to set_emote_context :: standIdle:



Your Locomotion is ready.

9. To add more Locomotions, for example a jogging and running animation, repeat steps 4-6.

Locomotion Properties

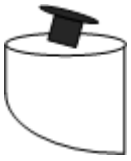


- Name: A label to identify the locomotion. It is best to use functional names like 'Walk,' 'Jog,' 'Run', a name that denotes the speed of the Locomotion.
- Resource: The animation .ani resource. (See Adding an Animation Resource).
- Native Speed: The speed at which your animation will play at its normal rate (explained below).
- Threshold: The minimum speed at which your avatar starts this animation (explained below).

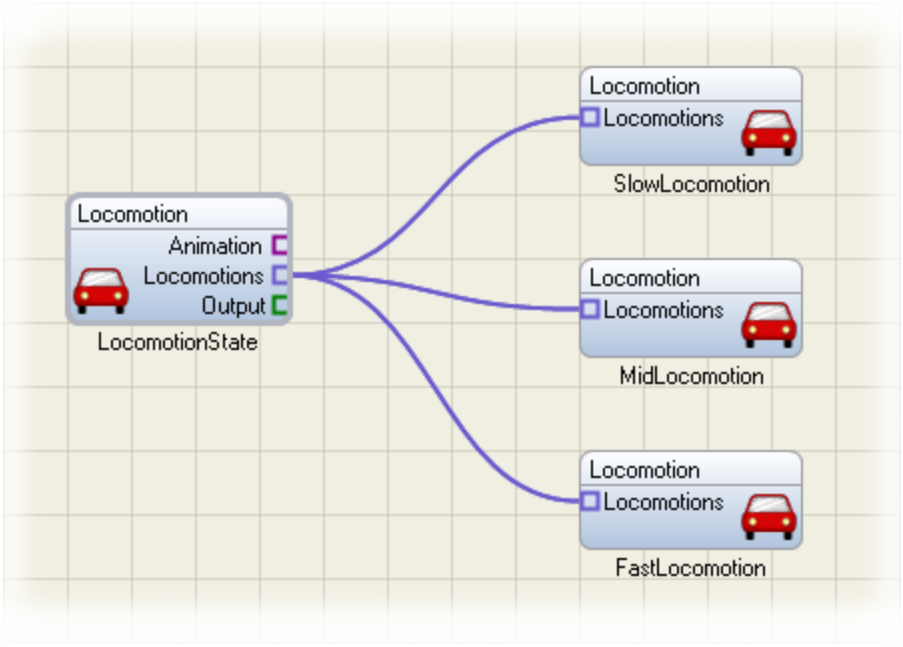
Speed is measured in meters per second (m/s).

The Speed and Threshold Speed

You can set the speed for a particular animation through the Speed and Threshold properties. The following example shows the animations and the corresponding speeds and position on the PlayStation DUALSHOCK®3 controller's analogue stick.

Threshold	Locomotion Name	Animation Resource	Effect in PS Home	Analog stick position
N/A	N/A	N/A	Idle/Standing	
0	SlowLocomotion	Walk.ani	Walking	
1	MidLocomotion	Jog.ani	Jogging	
2	FastLocomotion	Run.ani	Running	

The following image shows a complete set-up for a 'move' Locomotion.



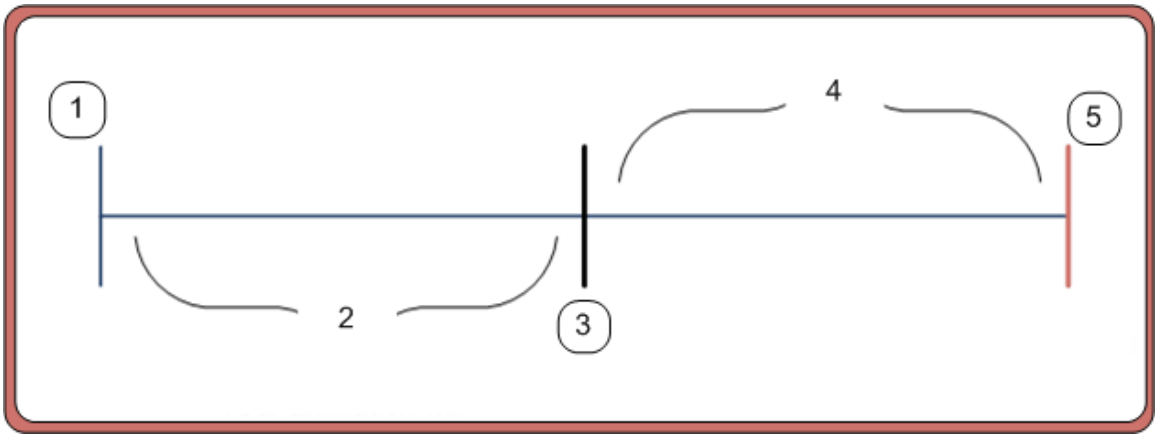
This 'move' Locomotion has three nodes:

Locomotion Name	Animation	Threshold Value	Native Speed Value
SlowLocomotion	Walking	0	0.5
MidLocomotion	Jogging	1	1.5
FastLocomotion	Running	2	2.5

The first Locomotion, 'SlowLocomotion', is the animation that plays when the user changes from standing still to moving. Starting Locomotions need to have the Threshold set to 0, since they start that animation as soon as the user moves from standing (0 m/s).

The second Locomotion, 'MidLocomotion', is the animation that plays when the user increases speed from a walking to jogging. This Locomotion's Threshold needs to be greater than the previous Locomotion's Speed.

The third Locomotion, 'FastLocomotion', is the animation that plays when the user increases speed from jogging to running. The Threshold needs to be greater than the previous Locomotion's Speed.



1.	The Threshold speed. This is the speed at which Locomotion starts.
2.	During this period, the Locomotion's animation will play slower than its normal speed. This is to simulate the transition between, for example, starting to sprint and getting into the full sprint.

3.	This is the Speed. This is when the animation will play at its normal rate. When testing locomotions, use the DevDebug command runSpeed and set the value to the same number as the Speed.
4.	The Locomotion's animation will play faster than normal when above the Speed value. This is to simulate the transition between, for example, walking fast and jogging.
5.	This is the speed at which the next Locomotion animation will start to play; it is the Threshold speed for the next Locomotion.

Testing Locomotions

When testing locomotions, you can use the Dev Debug Console command runSpeed <value> to change the maximum running speed.

Repertoire Emotes

Emotes are containers that enable users to select an Action or Behaviour for their avatar to perform. For example, in PS Home, if a user clicks [R1] > Greeting > Wave, their avatar performs the wave Action. If they press [R1] > Dance > Casual, their avatar performs the casual dancing Behaviour.

You can assign custom Emotes to FullBodySuits, Mini-games and Realtime games.

If using Emotes with scripted objects, ensure you do not lock the [R1]. If you do, you will need to create a custom OSD menu to give users access to your custom Emotes. Users do not have access to the full range of default Home Emotes using this method.

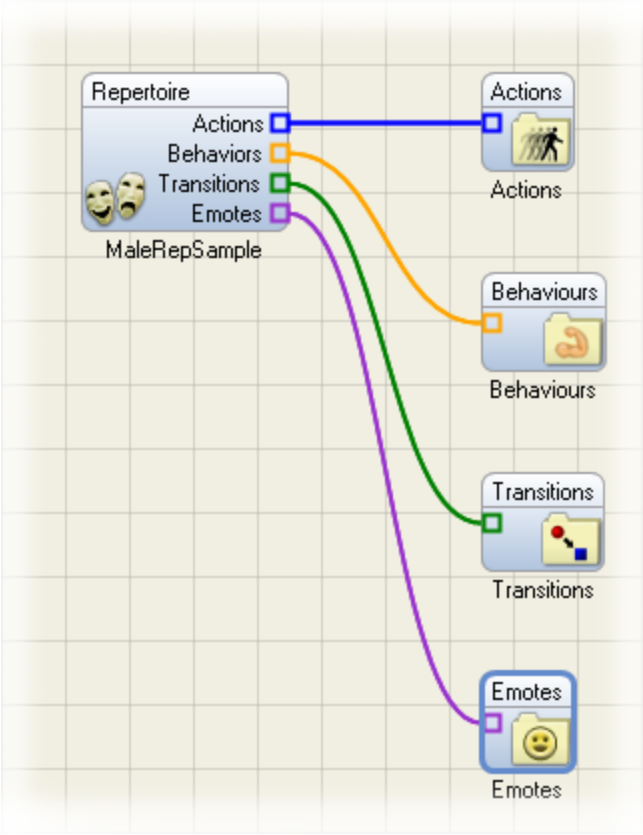
When creating custom Emotes, you can add new Emote groups to the Avatar's Emote menu (the menu is accessed by pressing [R1] on the controller while in PS Home), or you can add Emotes to the existing groups.

Creating Emotes

Prerequisites

Before creating Emotes, you must first create the Actions and/or Behaviours that you want users to be able to access with [R1]. When those are created, follow these steps to creating Emotes.

PS Home does not use the Repertoire XML and Emote XML files to create the emote menus. The menus and button names come from the object's Localization XML. Ensure that each EmoteGroup, ActionEmote, and BehaviourEmote node that you create has a unique name; this name is required by the Localization XML.

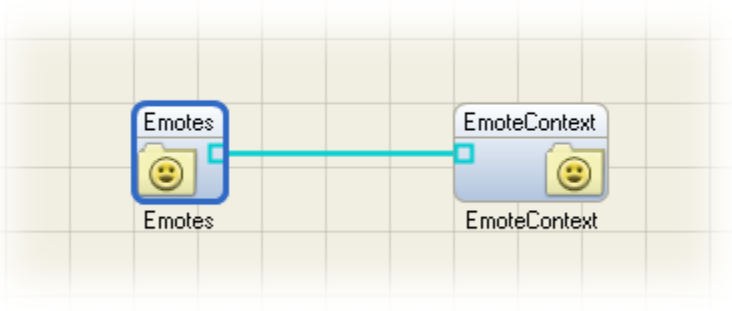


Creating an Emote

To create an Emote:

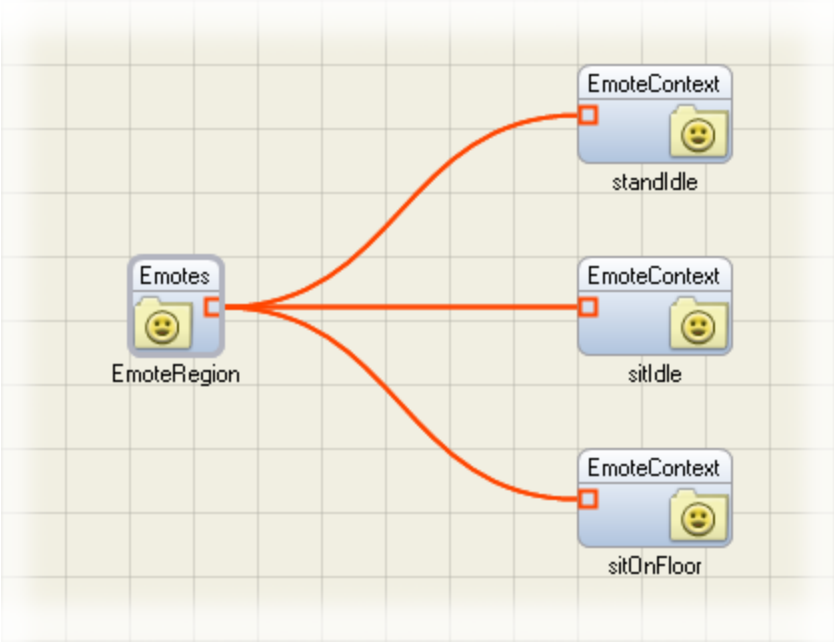
1. Double-click the Emotes group from the main Repertoire Work Area.
2. Drag either an EmoteContext or EmoteRegion node onto the Work Area (see EmoteContext vs EmoteRegion).

> ⚠ You cannot have both EmoteContext and EmoteRegion nodes in your Emote Work Area. You can only have one type, either EmoteContext or EmoteRegion.



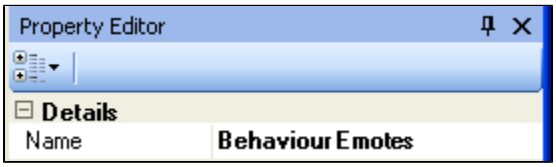
3. Only for EmoteRegion node: Select EmoteRegion > Property Editor > Region. Select a region.
4. Double-click the node you added to enter its Work Area. You will see the following nodes in each EmoteContext and EmoteRegion node:

    - standIdle: Use this branch to assign Actions and Behaviours for the user to select while standing.
    - sitIdle: Use this branch to assign Actions and Behaviours for the user to select while sitting on furniture.
    - sitOnFloor: Use this branch to assign the user to select while sitting on the ground.



The Emotes menu changes depending on whether the user is standing, seated on furniture, or sitting on the floor.

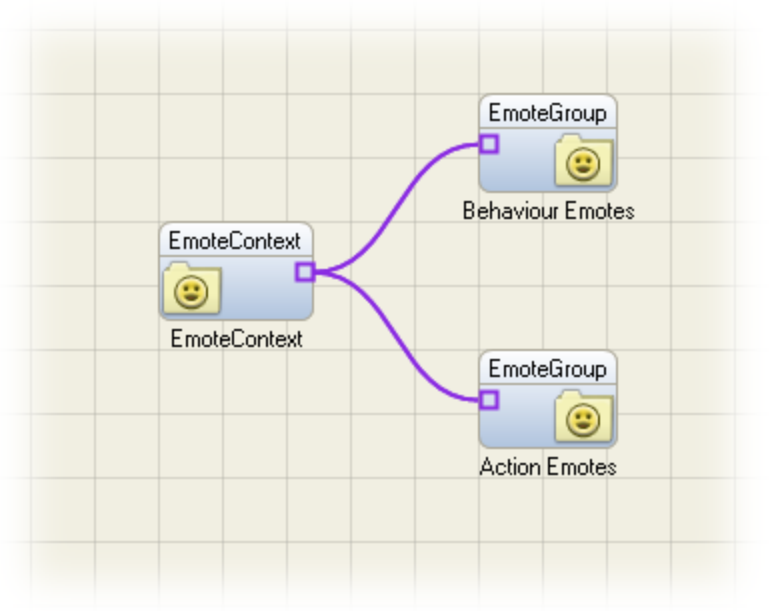
5. Double-click on the node that you want to add/edit. This example creates emotes for standIdle.
6. Drag an EmoteGroup node from the Palette into the Work Area.
7. Select the EmoteGroup, go to Property Editor > Name, and change the name.



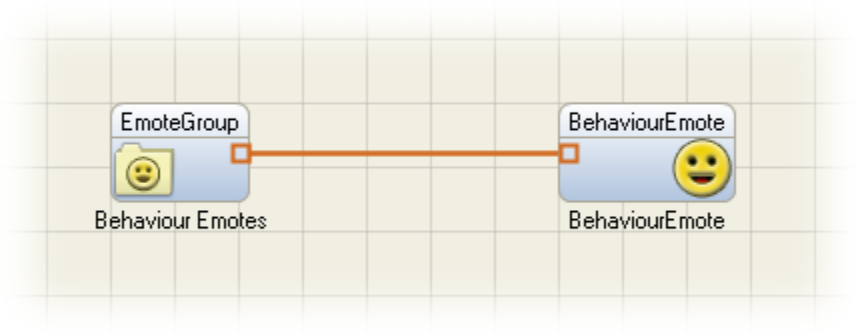
> ⚠ This name is used by the Localization XML. It is required, and must be unique.

Use descriptive names. For example, if the EmoteGroup contains only kung-fu Actions, call it 'Kung-Fu.'

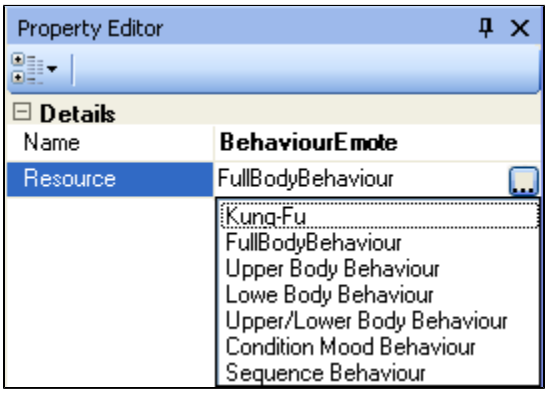
The following image shows two EmoteGroups added to the EmoteContext, Behaviour Emotes and Action Emotes.



8. Double-click on an EmoteGroup to enter its Work Area. This example uses the 'Behaviour Emotes' EmoteGroup.
9. Drag either an ActionEmote or a BehaviourEmote from the Palette.



10. Select the ActionEmote/BehaviourEmote > Property Editor > Name, and give the BehaviourEmote a unique name. This name is used by the Localization XML.
11. Select Property Editor > Resource. A drop-down list is displayed showing either all of the Actions/Behaviours in your Repertoire. Select the desired Action/Behaviour from the list.



12. Your Emote is now finished, but cannot yet be tested or properly exported. You first need to add localization. See Repertoire Emote Localization.

EmoteContext vs EmoteRegion

EmoteContext: Makes all Actions/Behaviours within this node accessible to users in all regions.

EmoteRegion: Allows only users in the defined country/region to have access to the Actions/Behaviours within the node.

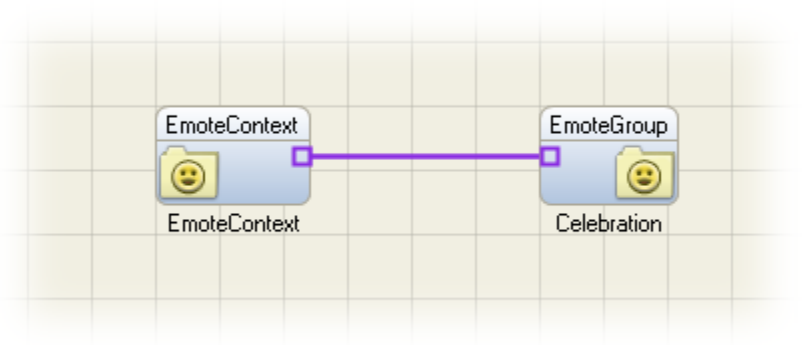
> ⚠ If you use EmoteRegion, you must create an EmoteRegion node for each country/language supported in the region in which the object is published.

Adding an Emote to an Existing Menu

To add an Emote to an existing menu in PS Home (such as Actions, Poses, Dance), follow Creating an Emote above, but for the EmoteGroup name, use the name of the menu in PS Home. You can add to the following existing emote menus:

- Greeting
- Conversation
- Celebration
- Disappointment
- Poses
- Dance

For example, the following image shows an example of an Emote being added to the Celebration menu in PS Home:



> ⚠ Note that:
>
> - The Localization XML must still have localization information for the menus.
> - The Localized text must be unique for that menu (i.e. you cannot add another 'Wave' Emote to the Greetings menu).

Overriding Emotes

You can replace certain default PS Home emote animations with your own custom animations. For example, instead of using Greeting > Wave using the default PS Home wave, you can override it with your own custom animation.

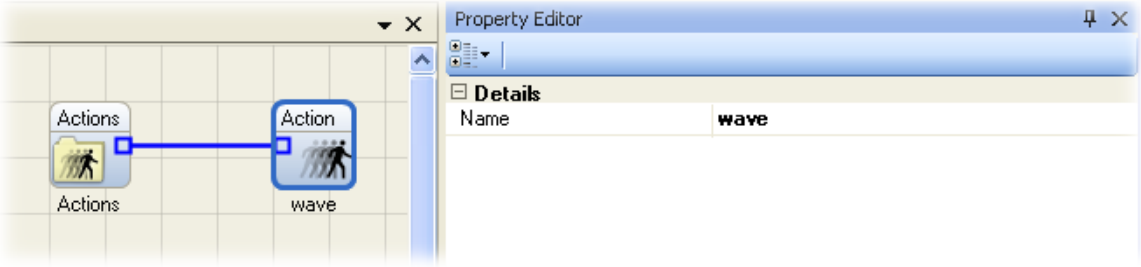
You can override only certain action Emotes (that is, only emotes that trigger actions). You cannot override default PS Home behaviour Emotes.

You can only override the following action Emotes:

- agree
- beckon
- bow
- cheer
- clap
- disagree
- frustration
- laugh
- point
- wave

To override an action Emote:

1. Create an action.
2. When you name it, use the name of the existing action Emote that you want to replace.



3. Save and export the Repertoire as normal.

> Do not follow the normal steps in creating an Emote. Those steps are not required when overriding the default PS Home Emotes.

The process for overriding the default PS Home Emotes is not the same as creating a new Emote. This means that you cannot override (replace) default Emotes using the Emotes nodes in the Repertoire Editor.

Repertoire Emote Localization

You must add localization for each EmoteGroup, ActionEmote, and BehaviourEmote that you add to a Repertoire, otherwise the emotes do not appear in PS Home. You localize the emotes in the object's `localisation.xml` file. When you edit the `localisation.xml` file manually, you do not immediately see the changes in the Object Editor or the Repertoire Editor, so you need to update the localization resource in the Object Editor.

> If you need to update Object localization information, you must edit the `object_master.xml` file, not the `localisation.xml` file. It is The `object_master.xml` file that produces a localisation.xml file for each object. See Localizing Objects.

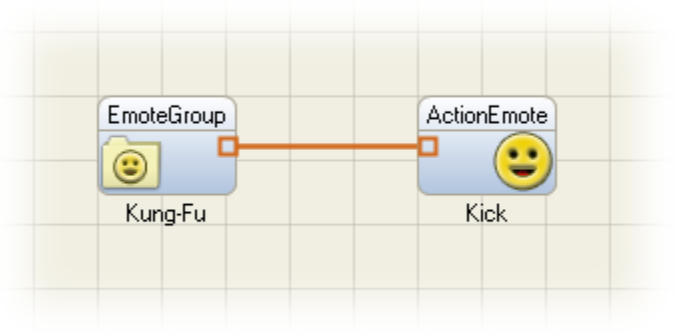
Localizing Repertoire Emotes

The `localisation.xml` file is located in the object's folder, in `<HDK_INSTALL_DIR>/build/objects`.

To edit the localisation file:

1. Select the object in the Object Editor (File > Explore).
2. Select `localisation.xml` from the list of object files, and open it in an editor.
3. Add a <PHRASE> tag for each EmoteGroup, ActionEmote and BehaviourEmote in the object's Repertoire.
   Each <PHRASE> tag contains the <LANG ID=" "> tags. These tags are the localized text for each emote.

For example, if you have an EmoteGroup named 'Kung-Fu', and an ActionEmote named 'Kick':



You might add the following XML tags:

```
<PHRASE>

	<LANG ID="REF">Kung-Fu</LANG>
	<LANG ID="en-GB">Kung-Fu</LANG>
	<LANG ID="fr-FR">Kung-Fu</LANG>
	<LANG ID="de-DE">Kung-Fu</LANG>
	<LANG ID="it-IT">Kung-Fu</LANG>
	<LANG ID="es-ES">Kung-Fu</LANG>

</PHRASE>

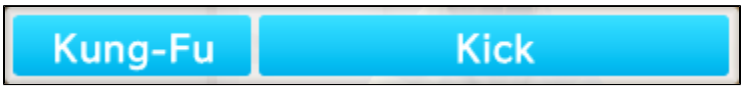
<PHRASE>

	<LANG ID="en-GB">Kick</LANG>
	<LANG ID="fr-FR">Lancer une Ruade</LANG>
	<LANG ID="de-DE">Kicken</LANG>
	<LANG ID="it-IT">Calcia</LANG>
	<LANG ID="es-ES">Patear</LANG>

</PHRASE>
```

In the en-GB region, the localized names appear in PS Home like this:





In the it-IT region, they look like this:





Updating the Localization Resource in the Object Editor

To reflect changes in the Object Editor and Repertoire Editor:

1. Save the updated `localisation.xml` file.

> If you are testing emote names, skip steps 2-4 and follow the steps in Viewing Repertoires in PS Home to immediately see the updated localized emote names.

2. If the Repertoire Editor is open, close it.
3. In the Object Editor, select another object, and then re-select the object with the Repertoire. The Object View > Localisation branch is updated.
   This forces the Object Editor to reload all of the object's resources.
4. If required, re-open the Repertoire Editor. This step is only necessary if you are validating the Repertoire.

Repertoire Transitions

Transitions were a way in previous HDKs to combine animations together to play in a particular order. In previous versions of the Repertoire Editor, you would set the beginning, middle and final animations (called From, Via, and To animations). The animations would play in this order: From > Via > To.

Transitions play from 'Animation A' via 'Animation B' to 'Animation C'.

Animation B must be very short, it is in effect the transitional animation between A and C.

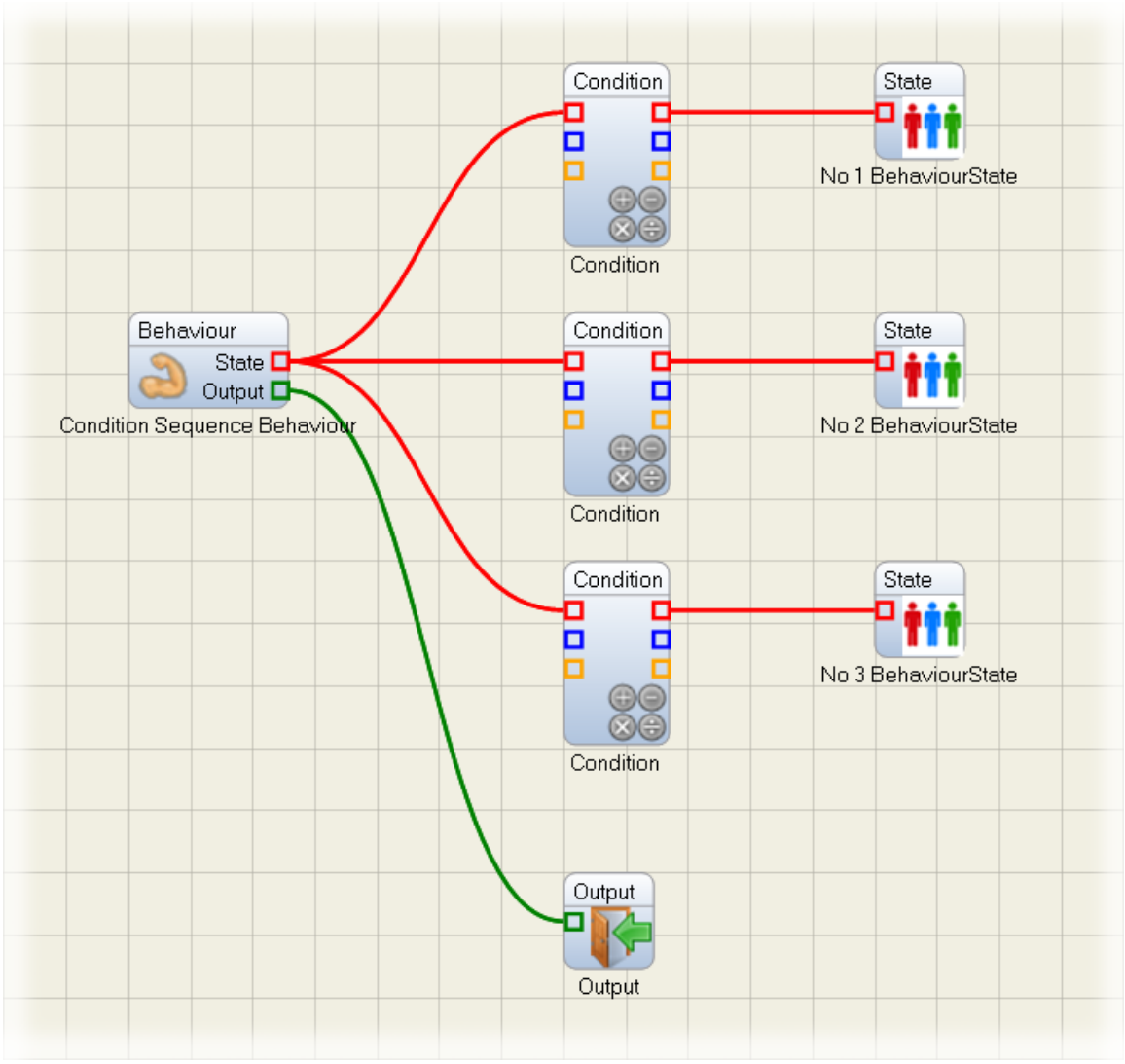
> You can still create Transitions, however it is not recommended.

Transitions work on a different principle from the state based system which the other elements of the Repertoire use. Transitions are triggered not by states or conditions but by the animations. When animations are combined into a Transition, they become inseparable in the client. Meaning that no matter where the animation is called from within the Repertoire, it will play the entire Transition and not just the single animation.

Alternative to Transitions

Instead of Transitions, use Conditions to control how and when animations play. Conditions work well within the Repertoire Editor's state-based system, and are easier to control and debug.

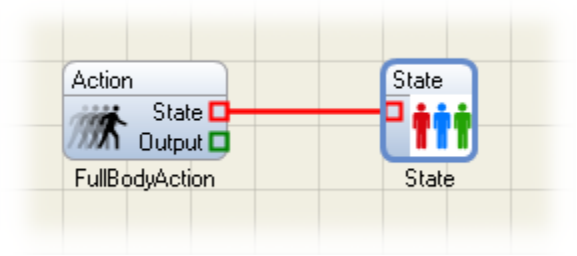
For an example that shows how to play animations in a particular sequence, see the example in Repertoire Conditions and Logic.



Repertoire States

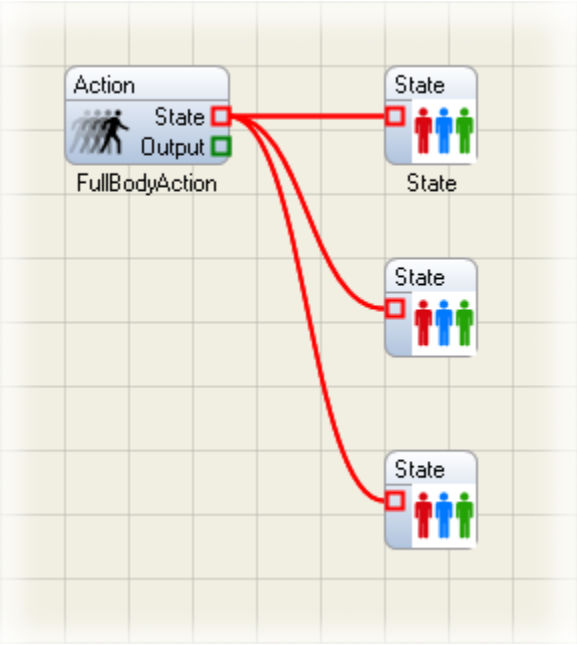
The basic structure of a Repertoire is the group, for example, Action group/Behaviour group > Type of Animation > State > Animation . The Repertoire Editor however is based on States.

States are the core of the Repertoire. Actions, Behaviours, and Locomotions all use states. For example, a simple Action contains a single state that has a single animation attached to it. The following example shows a simple action called 'FullBodyAction' with one state attached to it. When you call the Action to play within PS Home, it plays the state within the Action, not the Action itself. You can have more than one state in an Action or Behaviour.



States and Conditions

You can use Conditions and Logic to select states and the order in which the states play. The following image shows an Action with several states. A state is chosen at random when you call the Action or Behaviour. You can use this to add variations to your Action by having several states with different animations.



For more information, see Repertoire Conditions and Logic.

Creating a State

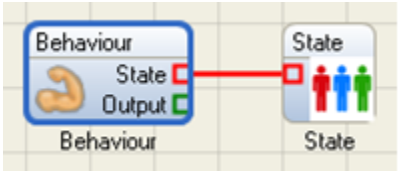
To create a state:

1. Double-click an Action or Behaviour.
2. Drag either an Action/Behaviour from the Palette into the Work Area, and connect the nodes:





3. Double-click the Action/Behaviour that you added to from the Palette to open up the next Work Area.
4. Drag the State from the Palette to the Work Area.
5. Connect the Action/Behaviour to the State:



6. Double-click on the State to enter it (i.e. open it in the Work Area).
   The state is displayed in the Treeview Navigation:



You can now add animations and outputs to the Action or Behaviour.

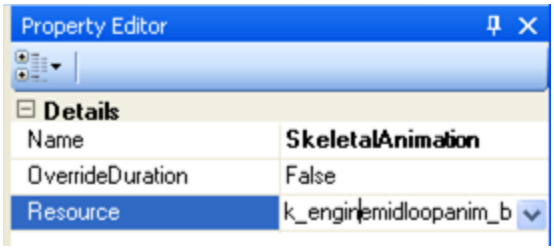
Adding an Animation to a State

To add an animation to a state:

1. Create a state node (see Creating a State above).
2. Enter the State by double-clicking on it.
3. Drag an animation node from the palette into the Work Area.
4. Connect the state to the animation node:

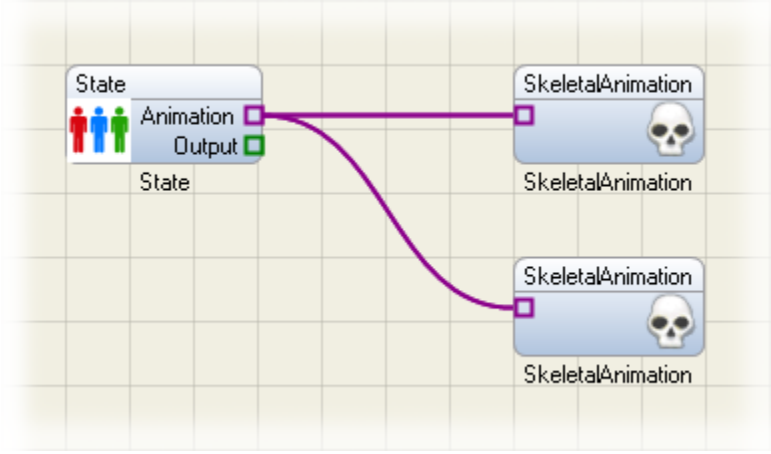


5. To add an animation resource, select Property Editor > Resource and select an animation from the list:



Multiple Animations

You can have more than one animation in a state, i.e. a skeletal animation for the body, or an upperbody animation and a lowerbody animation. To do this, connect multiple Animation nodes to the State node:



See also Adding an Animation Resource.

Repertoire Animation Blending Workflow

The workflow for repertoire animation blending is as follows:

	Task	Description
1.	Create an object or edit an existing object	Create a Mini-game or Realtime game object using the New Object Wizard
2.	Launch the Repertoire Editor	Launch the Repertoire Editor from the Object Editor to add the Repertoire component and the target rigs
3.	Re-open the object in the Object Editor	Close the Repertoire Editor and reopen your object in the Object Editor to view the updated object
4.	Define animation registers to use within your animation repertoires	Add animation registers to the Repertoire component of the object
5.	Continue working on your object in the Repertoire Editor	Re-launch the Repertoire Editor to continue working on your object

> You can add animation blending to mini-games and real-time games only. You cannot add animation blending to clothing objects, such as FullBodySuits.

Repertoire Animation Blending

> You can add animation blending to mini-games and real-time games only. You cannot add animation blending to clothing objects, such as FullBodySuits.

Animation Blend Trees

Animation Blend trees are animation resources used in an animation state. An animation blend tree is a graph of connected animation blend nodes. The tree itself is defined as either a skeletal animation tree, or a blend shape animation tree.

The Blend tree evaluates each blend node, and uses the output animation of the blend node as an input to another blend node, or as the final result of the blend tree.

Animation Blend Node

The blend node comprises the following:

- A domain control variable
- A list of resource inputs (maximum of 8)
- A sequence of domain ranges
- An output tag
- A curve type

These components determine which animations should be evaluated and blended to produce an output animation. A minimum of one animation, and a maximum of two animations are evaluated for any given value of the normalized control variable.

For any blend tree, you must use all blend nodes either as resource inputs or flag them as the output. You can flag only one blend node as an output node, and you cannot use that node as a resource input.

Domain control variable

The domain control variable determines the weighting of each animation that is blended in to produce the blend nodes output animation. The control variable specifies the name of an animation register, which should be of the native type number. The control variable also specifies the range that this variable should fall into, and uses those range values to clamp and normalize the run-time value.

Resource input

A resource input can either be an animation resource, or a reference to another blend node. If it is an animation resource, then the type of the animation resource must be the same as the type of the blend tree. If it is a reference to a blend node, then it cannot be a reference to itself. The order of the resources is important; the domain ranges use an index into the resources to indicate which resources should be considered for evaluation for a given normalized control variable value.

> The resource cannot be a blend tree animation.

Domain range

The range description describes which resource inputs are considered for a given range. If only one input is specified, then that input is considered to have a blend weighting of 1.0. If two inputs are specified, the first input specified has its weighting decrease from 1.0 to 0.0 over the range, and the second input has a (1-x) weighting, where x is the weighting of the first input.

Domain range constraints

- The first domain must start at 0.0
- The last domain must end at 1.0
- Each domain range must have at least one resource input specified (by index)
- Every domain range except the first must start at the same point or after the previous domain range start point
- Every domain range except the last must end before, or at the same point as the next domain range end point.
- The end point of a domain range must start at or before the start point of the next domain range.
- All the inputs must be specified at least once in the domain ranges.

Animation Blending Examples

Fat/Thin locomotion

You can use a fatness control variable to blend a fat animation/ walk animation with a thin/walk animation to obtain a walk animation.

You can also use the fatness control variable to blend a fat run animation with a thin run animation to obtain a run animation.

You can then use the resulting walk and run animations in a locomotion animation network to get the blended locomotion.

Mini-game

A mini-game can have multiple walk animations depending on the weapon the avatar is carrying, for example, Hammer Walk, Sword Walk, Gun Walk, Unarmed Walk.

You can use control variables, for example, *player_weapon *to specify which animation the repertoire uses.

See also:

- Adding a Repertoire
- Defining Animation Registers
- Repertoire Editor Node Reference

Validating Repertoires

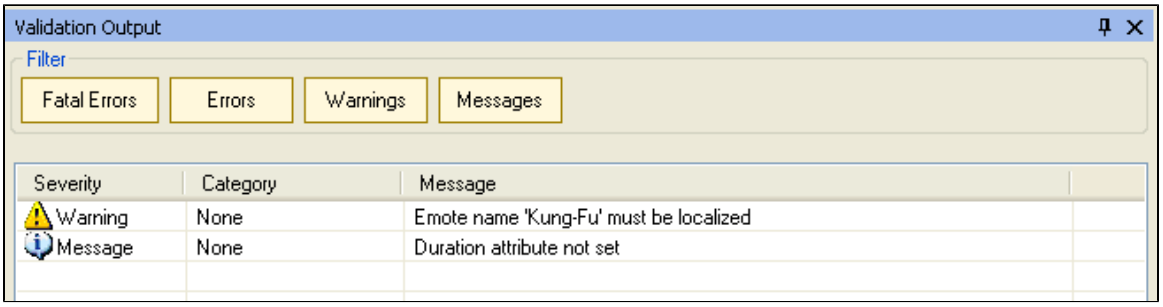
The Repertoire Editor has a validation function that checks all of the nodes in a Repertoire to check that they have all the necessary minimum settings to successfully package and work in PS Home. The checks include:

- A Message if a State node does not have a Duration time set.
- A Fatal Error if an animation node does not have an animation resource.
- A Warning if an EmoteGroup, ActionEmote and BehaviourEmote does not have localization.

Validating a Repertoire

To validate a Repertoire, click  or select File > Validate > Validate current repertoire circuit.

Errors appear in the Validation Output dialog, for example:

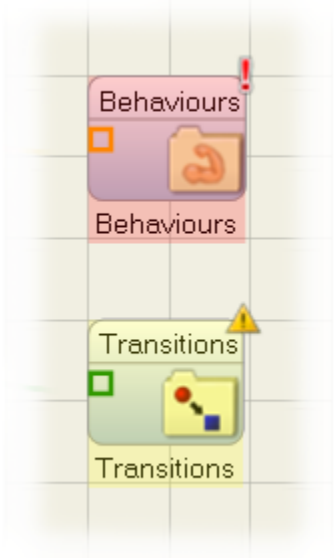


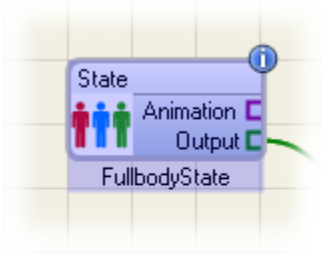
Fixing Validation Errors

To fix any validation errors found:

1. In the Validation Output dialog, double-click any Warnings or Messages that appear after validating.

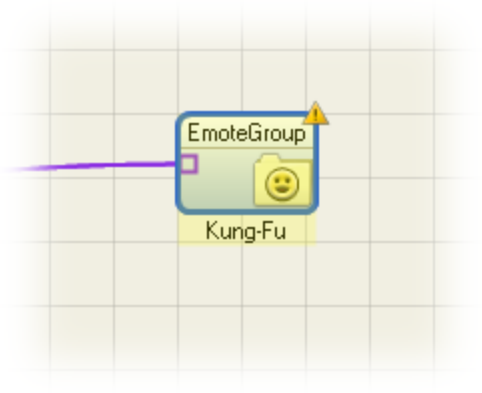
   The Work Area displays the node that caused the error:





2. After you have fixed a validation error, click  . The Validation Output dialog is updated.

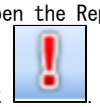
Resolving Localization Warnings



To resolve a localization warning:

1. Double-click the Warning in the Validation Output dialog.

   A localization warning is also displayed when you misspell the node name in the <LANG ID="REF"> tag.

2. Fix the error in the object localisation.xml file.
3. Save the file and close the Repertoire Editor.
4. Re-open the Repertoire Editor.
5. Click  . If there are no spelling errors, then the localization warning for that node no longer appears in the Validation Output dialog.

Viewing Repertoires in PS Home

To view a Repertoire during the Repertoire development process:

1. Make sure to set the default state in the Repertoire Editor (i.e. check the initial state box).
2. Make sure to have deferred loading set to FALSE if you are not going to manually load in the Repertoire.
3. Scripted objects only: Make sure to only load the Repertoire for the given avatar's sex (i.e. load only the male one for males, female one for the females).

Viewing Actions and Behaviours

You can view and test Actions and Behaviours in scripted objects using the Lua Person library. You must add them to Emotes before FullBodySuits can access them.

To view an Action or Behaviour:

1. Create the Action or Behaviour in the Repertoire. See Repertoire Actions and Repertoire Behaviours.

   If using Conditions, set them up. See Repertoire Conditions and Logic.

2. Save the Repertoire.
3. Select File > Export > Export repertoire and emote files.
4. Launch PS Home, and join the scripted object.
5. Use the scripted object to call the following functions:

```
LocalPlayer.AddRepertoire()
Person.Lock()
LocalPlayer.DoAction or LocalPlayer.SetBehaviour()
Person.Unlock()
```

Making Quick Updates

You do not need to reload PS Home to view changes you make in the Repertoire Editor. To see changes, follow steps 1-4 above, unload and reload the Repertoire file, and call the Action/Behaviour again. You must reload the scripted object (or the scene) if you add more Actions and Behaviours, or change their names. Adding Actions and Behaviours or changing names requires you to also change the Lua script file, which requires a reload of the object.

Emotes

You can view and test Emotes using scripted objects and FullBodySuits.

To View Emotes:

1. Create the Emotes.
2. Update the object's localisation.xml file.
3. Save the Repertoire.
4. Select File > Export > Export repertoire and emote files.
5. Launch PS Home, and play the scripted object or wear the FullBodySuit.
6. Press [R1] and select the Emote.

Making Quick Updates

You do not need to reload PS Home to view changes you make to Emotes in the Repertoire Editor; this works if you add new Emotes or change the names of Emotes. To view your changes:

1. Follow steps 1-4 above.
2. Exit from the scripted object, or remove the FullBodySuit (i.e. make the avatar wear another piece of clothing) in PS Home.
3. Re-enter the scripted object, or re-equip the FullBodySuit. This force reloads the object data and Repertoire files.

Accessing Repertoires in Script

> For Japanese Live Page
> 最新の日本語版ページはこちら 「スクリプトによるレパートリーへのアクセス」

The Repertoire file for scripted objects and FullBodySuits are essentially the same. However, you cannot reserve the [R1] in a Mini-game/Realtime game if you want users to access the default emotes.

Scripting Repertoires

The following table lists the Lua commands necessary to use repertoires:

Command	Description
`LocalPlayer.AddRepertoire()`	Adds a Repertoire to a local player.
`LocalPlayer.RemoveRepertoire()`	Removes a Repertoire from a local player.
`LocalPlayer.DoAction()` `Person.DoAction()`	Plays an Action on a user. You must lock the player before calling the Action. The Action must exist in the Repertoire.
The result of this function is broadcast to other users in the space.	
`LocalPlayer.SetBehaviour()` `Person.SetBehaviour()`	Plays a Behaviour on a user. You must lock the player before calling the Behaviour. The Behaviour must exist in the Repertoire.
The result of this function is broadcast to other users in the space.	
`Person.GetCurrentAction()` `Person.GetCurrentBehaviour()`	Returns the name of the Action/Behaviour being played on a user, if one is being played.
`Person.GetRigField()`	Returns the gender of the local user.
`Person.HasLock()`	Checks the lock status of a user.
`Person.Lock()`	Locks a user.
`Person.Unlock()`	Unlocks a user.

Calling Actions and Behaviours

Before you can call an Action/Behaviour, you must first lock the user, play the Action/Behaviour, and then unlock the user. The script you use to call an Action/Behaviour should look similar to this:

```
Person.Lock()
Person.SetBehaviour()
Person.Unlock()
```

You need to lock the user only while calling the Action/Behaviour. The avatar still performs the Action/Behaviour after being unlocked.

Using repertoires in a Mini-game or Realtime game

When using repertoires in a Mini-game or Realtime games, be aware of the following considerations:

- You must add and removed repertoires from the local player at the right times: you add them on or after `OnLocalPlayerJoin()` and remove them on or before `OnLocalPlayerLeave()`
- Custom avatar animation files should not use deferred loading.
- Repertoires should not use the `is_exclusive` tag

For more information on creating custom animations for avatar, see Repertoire Editor.

When to Add and Remove Repertoires

Adding resources too soon and failing to clean up resources that are no longer used is not good practice for scripts in general. It is very important to remove resources when they are no longer needed, and this applies to repertoires as well. Explicitly removing repertoires in Lua frees the repertoire slot for use by another Mini-game. It also prevents a known bug whereby the repertoire remains attached to the local player even when the player relocates to a different scene.

For Mini-games and Realtime Games

- Add repertoires on or later than `OnLocalPlayerJoin()`
- Remove repertoires on or before `OnLocalPlayerLeave()`

Do not add repertoires when a Mini-game is initialized, because it adds the repertoire to the local player regardless of whether the player has joined the game.

For Actives

The same rules that apply to Mini-games/Realtime Games also apply if they are a part of an active item.

For FullBodySuits

This is handled automatically by the client.

When to use Deferred Loading for Repertoires and Animations

Set `deferred_loading` to false for Mini-games and FullBodySuits.

When to use the Is Exclusive Flag

Set the `is_exclusive` flag to false for Mini-games, Realtime games, and FullBodySuits.

Repertoires and Gender

All repertoires must have animations for the rigs supported.

For Mini-games and Realtime Games

Mini-game and Realtime game repertoires must have animations for both rigs (male or female). Also, scripts must elegantly handle avatars changing gender partway at any time while in the Mini-game or Realtime game.

If you have a repertoire in a Mini-game that contains two sets of animations, one for male and one for female, and the animations have identically named actions and behaviors, you do not need to call the male and female animations separately. Instead, you can use the value `nil` in the `LocalPlayer.AddRepertoire` function. The client will automatically use the appropriate gender animation.

Example: `LocalPlayer.AddRepertoire( nil, 1 )`.

For Actives

The same rules that apply to Mini-games and Realtime Games also apply if they are a part of an active.

For FullBodySuits

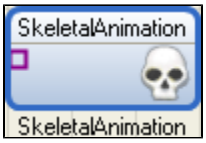
For FullBodySuit repertoires, the animation must match the rig of the FullBodySuit. For female FullBodySuits, a female animation must be available, and for male FullBodySuit, a male animation must be available. Swapping FullBodySuit repertoires happens only when swapping the FullBodySuit itself. This swap is handled automatically by the client.

Repertoire Editor Node Reference

This section provides a summary of the Repertoire Editor nodes and their properties.

Animation

Skeletal Animation

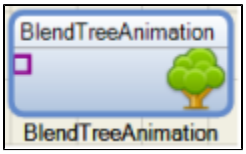


Attaches an animation resource (.ani) to an Action or Behaviour.

Properties

- Name: Custom name for the SkeletalAnimation node. This name has no effect. It is only for personal reference.
- OverrideDuration: If set to True, when this animation is played with another animation that is longer, it forces that other animation to stop playing when this animation has finished.
- Resource: The animation resource (.ani) file.

Blend Tree Animation



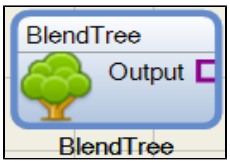
Attaches a blend tree animation resource (.ani) to an Action or Behaviour.

Properties

- Name: Name of the blend tree animation.
- BlendTreeResource: The name of the blend tree resource

BlendTrees

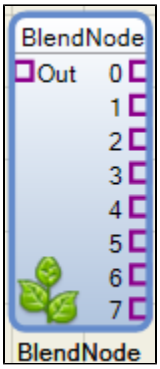
Blend Tree



Connects to a single root blend node which may combine inputs of both skeletal animations and and other blend nodes.

See Repertoire Animation Blending for more information about how Blend Trees are used.

Blend Node



Connects up to 8 skeletal animations or blend nodes to create a blend sequence.

Properties

- Register: The animation register used to blend.
- Curve: Determines whether the sequence blend is linear or not
- Sequence: The blend sequence.

  See Repertoire Animation Blending for more information about how Blend Nodes are used.

Circuits

Comment



Allows you to add a comment to the Work Area. This will not be exported to the Repertoire or Emote XML.

Properties

- Text: The comment text.

Emote

ActionEmote

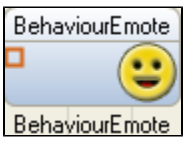


Attaches an Action to an EmoteGroup, which allows the user to play the Action by pressing [R1], and selecting it.

Properties

- Name: Name of the Emote. This name is required, and is used by the localisation.xml in the <LANG ID="REF"> tag.
- Resource: The Action to play. A drop-down list is displayed, allowing you to select any of the Actions in the Repertoire.

BehaviourEmote



Attaches a Behviour to an EmoteGroup, which allows the user to play the Behaviour by pressing [R1], and selecting it.

Properties

- Name: Name of the Emote. This name is required, and is used by the localisation.xml in the <LANG ID="REF"> tag.
- Resource: The Behaviour to play. A drop-down list is displayed, allowing you to select any of the Behaviours in the Repertoire.

EmoteGroup

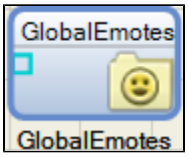


The EmoteGroup contains ActionEmotes and BehaviourEmotes. In PS Home, it is the first menu that appears when a user presses [R1] (for example, Greeting, Dance, Pose).

Properties

- Name: Name of the EmoteGroup. This name is required, and is used by the localisation.xml in the <LANG ID="REF"> tag.

GlobalEmotes

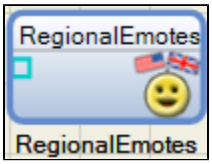


GlobalEmotes make all Emotes within it accessible to users in all regions.

Properties

- Name: Displayed only after double-clicking. Name of the EmoteContext. This name has no effect. It is only for personal reference.

RegionalEmotes



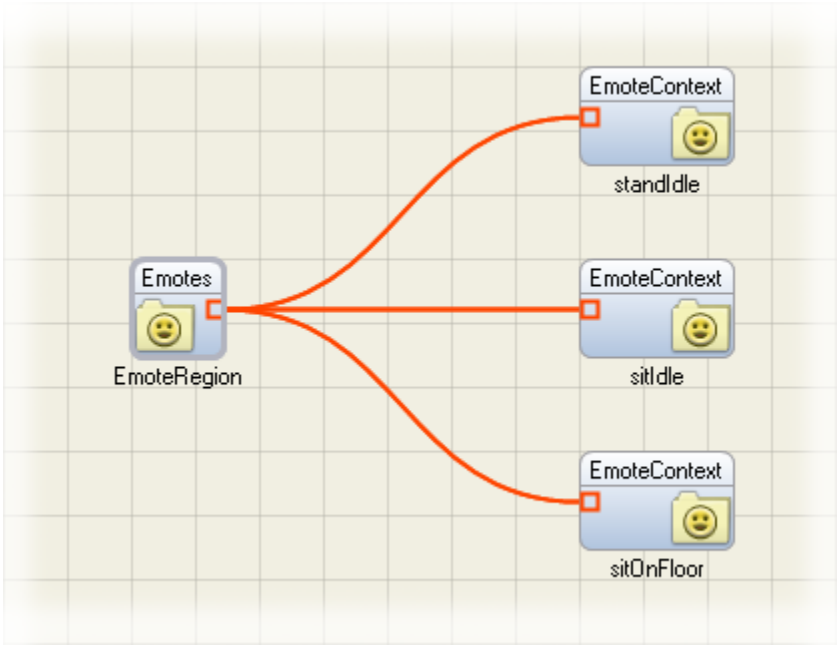
RegionalEmotes permit only users in specified region to access the Emotes within it.

> ⚠ If using RegionalEmotes, you must have a RegionalEmote node for each country in the region that the object will be published.

Properties

- Region: A drop-down list is displayed. Select the region in which the Emotes within will be available.
- Name: Displays only after double-clicking. Name of the EmoteContext. This name has no effect. It is only for personal reference.

GlobalEmotes and RegionalEmotes Subgroups

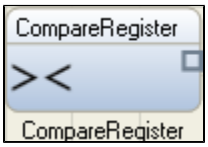


Double-click on either GlobalEmotes or RegionalEmotes to bring up this Work Area.

- standIdle: Put Emotes in here are available to users when they are standing.
- sitIdle: Put Emotes in here are available to users when they are sitting on furniture.
- sitOnFloor: Put Emotes in here are available to users when they are sitting on the floor.

Logic

Compare Register

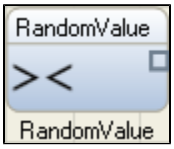


Compare against a register using the values set in the Properties.

Properties

- Comparison: A dialog is displayed with three properties to set: Register, Comparison Type, Value.
- Name: Name of the Condition, created automatically based on the Condition's properties. This name has no effect. It is only for personal reference.

RandomValue

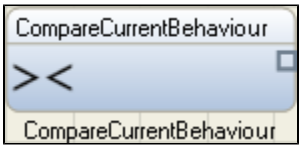


Enables you to generate a number randomly, either based on a list of numbers or within a range of numbers. The node returns True based on the Comparison Type property.

Properties

- Name: Name of the Condition, created automatically based on the Condition's properties. This name has no effect. It is only for personal reference.
- Value: A dialog is displayed with three properties:
    - Integer Value: This is the value that the random number will check against.
    - ComparisonType: Can be set to:
        - Equal: Returns True if IntegerValue equals random value.
        - Not_Equal: Returns True if IntegerValue is not equal to the random value.
        - Less_Than: Returns True if IntegerValue is less than the random value.
        - Less_Than_Or_Equal_To: Returns True if IntegerValue is less than or equal to the random value.
        - Greater_Than: Returns True if IntegerValue is greater than the random value.
        - Greater_Than_Or_Equal_To: Returns True if IntegerValue is greater than or equal to random value.
    - Value: Allows you to choose how the random number is generated:
        - integer:range: Set a range of numbers.
        - integer:selection: Create a list of numbers. Click the  button to add a value to the list.

Compare Current Behaviour



Compares the Behaviour that the users is currently playing against the name of the Behaviour (or against a list of names of Behaviours) in the Condition.

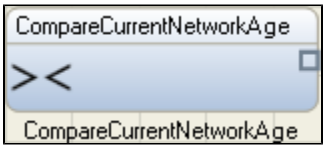
Properties

- Comparison: A dialog is displayed that lets you choose the comparison type and the Behaviour (or string of Behaviours) to compare the user's current Behaviour against:
    - Comparison Type: Can be set to:
        - Equal: Returns True if IntegerValue equals value.
        - Not_Equal: Returns True if IntegerValue is not equal to the value.

> The following values have no effect: Less_Than, Less_Than_Or_Equal_To, _Greater_Than, Greater_Than_Or_Equal_To.

- Value: Can be set to:
  - String: A field is displayed. Enter the name of the Behaviour to compare against.
  - String:selection: Create a list of Behaviours to compare against. Click the  button to add a Behaviour to the list.
- Name: Name of the Condition, created automatically based on the Condition's properties. This name has no effect. It is only for personal reference.

Compare Current Network Age

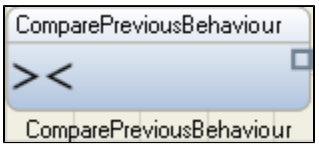


Compares against the current age of the network.

Properties

- Comparison: A dialog is displayed that lets you choose the comparison type and number.
  - ComparisonType: Can be set to:
    - Equal: Returns True if IntegerValue equals value.
    - Not_Equal: Returns True if IntegerValue is not equal to the value.
    - Less_Than: Returns True if IntegerValue is less than the value.
    - Less_Than_Or_Equal_To: Returns True if IntegerValue is less than or equal to the value.
    - Greater_Than: Returns True if IntegerValue is greater than the value.
    - Greater_Than_Or_Equal_To: Returns True if IntegerValue is greater than or equal to value.
  - Value: Lets you choose either an integer, or a number taken to the third decimal place (0.000).
- Name: Name of the Condition, created automatically based on the Condition's properties. This name has no effect. It is only for personal reference.

Compare Previous Behaviour



Compares the Behaviour that the users previously played against the name of the Behaviour (or against a list of names of Behaviours) in the Condition. Returns no value when entering a Behaviour.

Properties

The following properties are available:

- Comparison: A dialog is displayed that lets you choose the comparison type and the Behaviour (or string of Behaviours) to compare the user's previous Behaviour against.
  - Comparison Type: Can be set to:
    - Equal: Returns True if IntegerValue equals the value.
    - Not_Equal: Returns True if IntegerValue is not equal to the value.

> The following values have no effect: Less_Than, Less_Than_Or_Equal_To, _Greater_Than, Greater_Than_Or_Equal_To.

  - Value: Can be set to:

- String: A field is displayed. Enter the name of the Behaviour to compare against.
- String:selection: Create a list of Behaviours to compare against. Click the  button to add a Behaviour to the list.

- Name: Name of the Condition, created automatically based on the Condition's properties. This name has no effect. It is only for personal reference.

Compare Previous State



Compares against the previous state that was played on the user against the name of the State (or against a list of States) in the Condition.

Properties

The following properties are available:

- Comparison: A dialog is displayed that lets you choose the comparison type and the State (or string of States) to compare the user's previous State against.
    - Comparison Type: Can be set to:
        - Equal: Returns True if IntegerValue equals the value.
        - Not_Equal: Returns True if IntegerValue is not equal to the value.

        > The following values have no effect: Less_Than, Less_Than_Or_Equal_To, Greater_Than, Greater_Than_Or_Equal_To.

    - Value: Can be set to:
        - String: A field is displayed. Enter the name of the State to compare against.
        - String:selection: Create a list of States to compare against. Click the  button to add a State to the list.
- Name: Name of the Condition, created automatically based on the Condition's properties. This name has no effect. It is only for personal reference.

Compare State Age



Compares the age of the current of the State.

Properties

The following properties are available:

- Comparison: A dialog is displayed that lets you choose the comparison type and number.
    - ComparisonType: Can be set to:
        - Equal: Returns True if IntegerValue equals value.
        - Not_Equal: Returns True if IntegerValue is not equal to the value.
        - Less_Than: Returns True if IntegerValue is less than the value.
        - Less_Than_Or_Equal_To: Returns True if IntegerValue is less than or equal to the value.
        - Greater_Than: Returns True if IntegerValue is greater than the value.
        - Greater_Than_Or_Equal_To: Returns True if IntegerValue is greater than or equal to value.
    - Value: Lets you choose a number taken to the third decimal place (0.000).

Compare Network Age



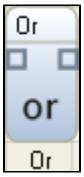


A logical operator node to allow for multiple cases in a condition. Returns True if all of the nodes attached to it (on the left side) return True.

Properties

None

Or



A logical operator node to allow a choice between multiple conditions. Returns True if any one of the nodes attached to it (on the left side) return True.

Properties

None

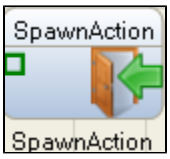
Condition



All logic and comparison nodes will be contained within a Condition node. This node returns True or False. If True, it plays the attached Action, Behaviour, or State.

Output

Spawn Action

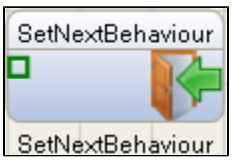


Play and action, either on entry or on exit.

Properties

- Name: This cannot be edited.
- Value: Opens a list with all Actions in the Repertoire.

Set Next Behaviour



Play a Behaviour, either on entry or on exit.

Properties

- Behaviour: Opens a list with all Behaviours in the Repertoire.
- Name: This cannot be edited.

Clear Register



Sets the specified register to False.

Properties

- Name: Displays the name of the Register chosen in the Register Property.
- Register: Opens a list of available registers.

Set Register



Sets the specified Register to either True or False, depending on Boolean Value chosen in Register property.

Properties

- Name: Displays the name of the Register and Boolean Value chosen in the Register property.
- Register: Opens a list of available registers, and allows you to choose to set their value to either True or False.

Trigger Event

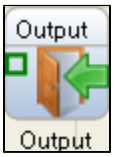


Trigger event puts the user in a particular mode.

Properties

- Event: A dialog is displayed that lets you choose the following:
    - Name: The type of mode to control, either set_control_mode allowing you to lock/unlock the user, or set_emote_context, allowing you to put the user in:
        - sitIdle: When the avatar is in the idle pose and sitting on furniture.
        - sitOnFloor: When the avatar is in the idle pose and sitting on the floor.
        - standIdle: When the avatar is in the idle pose and standing.
- Name: Displays the name of the Trigger Event based on the values chosen in the Event property.

Output



The Output node contains all other nodes in the output section of the palette. It will allow nodes to connect to an OnEntry and OnExit pin.

Properties

None

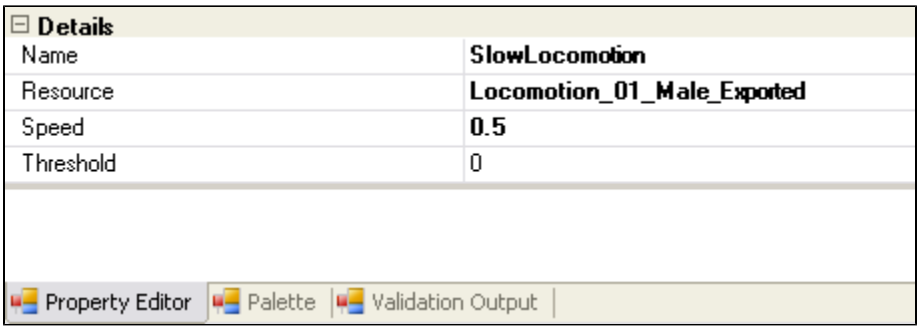
Repertoire

Locomotion



Added to a locomotion state to control what locomotion animation should play at which speeds.

Properties



- Name: A label to identify the locomotion. It is best to use functional names like 'Walk,' 'Jog,' 'Run', a name that denotes the speed of the Locomotion.
- Resource: The animation .ani resource (see Adding an Animation Resource).
- Native Speed: The speed at which your animation will play at its normal rate (explained below).
- Threshold: The minimum speed at which your avatar starts this animation (explained below).

> Speeds are measured in meters per second (m/s).

Locomotion State



Added to a Behaviour to create a Locomotion

Properties

- Duration: Set a duration for the animation.
- EaseIn: Blend-in value.
- EaseOut: Blend-out value.

Action



Action node added to the Actions group. It houses States for the Action. See Repertoire Actions.

Properties

- Name: This is the name of the Action. This is required and must be unique. Emotes, and Lua use the Action name if calling an Action.

The following properties are only visible after double-clicking on the Action node.

- Attributes: This has no effect. Do not use.
- Blocking: Toggles ability for Behaviours to interrupt the action. Set to False by default.
- Duration: The duration the Action should play for.
- InitialState: If more than one state exists this State will play first.
- Layer: The layer the Action should be played on. This can be a new custom layer.
- Priority: If there is more than one Action on a layer, this sets the Action's the priority compared to other Actions (for example, if Action1 has priority 1, and Action2 has priority 2, and both are called at the same time, then Action1 will play since it has higher priority).

Behaviour



Behaviour node added to the Behaviours group. See Repertoire Behaviours.

Properties

- Name: This is the name of the Behaviour. This is required and must be unique. Emotes, and Lua use the Behaviour name if calling a Behaviour.

The following properties are only visible after double-clicking on the Action node:

- Attributes: Enables you to add values to the sys_category attribute. This value can be queried using the Lua function `Person.GetBehaviorCategory()`.
- InitialState: If more than one state exists this state will play first.

Transition



Used to play a small transition animation between two main animations.

> This is NOT Recommended. This exists for legacy reasons (i.e. importing custom animations from previous HDKs). See Repertoire Transitions.

Properties

- FromAnimation: The starting animation.
- Name: The name of the transition.
- ToAnimation: The animation to transition to.

State



State nodes attach a SkeletonAnimation to both Actions and Behaviours. See Repertoire States.

Properties

- Duration: How long the state should play for.
- EaseIn: Blend-in value.
- EaseOut: Blend-out value.
- Name: The name of the State. This is required, and must be unique.

Avatar Animation Packs

> For Japanese Live Page
> 最新の日本語版ページはこちら 「アバターアニメーションパック」

An Animation Pack is a type of portable item. When equipped (in the same way players would equip or activate a portable), a custom animation for the avatar starts, and will last until such time as they deactivate it or replace it with a different portable item.

Users can activate the pack by selecting Activate from their Inventory Catalog Browser on the Menu Screen, and deactivate it using the Inventory Items Browser. Animation Packs automatically deactivate whenever a new portable item is activated. Like companions, Animation Packs remain active when moving between scenes.

> You cannot have a companion equipped while an Animation Pack is active.

You can create Avatar Animation Packs through the Object Editor using the File > New Object Wizard menu item and then selecting the Avatar Animation Pack option. Completing the wizard will create a new Avatar Animation Pack object, to which you can add animations, emotes and actions. Players can then use this object to extend their repertoire of animations. The player accesses these new animations from the Emotes menu in game.

You must provide the repertoire and associated animation resources. See Adding a Repertoire and Adding an Animation Resource.

You must provide animations for both male and female rigs, then bind the animations to the rigs using the Repertoire Editor.

> The default Avatar Animation Pack created through the wizard contains sample animation, repertoire and emote resources to give an example of how the finished object should work.

In the Repertoire Editor, you must also:

- Assign the names to be displayed on the Emotes menu
- Provide translations for these names
- Include the link from the menu option to the action played

The Repertoire Editor checks if the repertoire's object is an Avatar Animation Pack (using the template property in the object's Header component). If the object is an Animation Pack, the Repertoire Editor checks that the proposed emote category does not already exist.

You can also add sound effects to play with certain actions (but not behaviors). See Adding Sound Effects below.

> Animations are replicated, so everyone in the space sees the animations a user plays.
>
> Audio is not replicated. Only the animation pack user hears any sounds.

> You are not allowed to extend the default Emote categories.

Adding Sound Effects

Step 1 – Add a Sound Bank

You can add a sound effect to be fired when an animation begins. To associate audio with certain actions, you must add a soundbank, then update the object's configuration file to specify which sound effect to play, and when to play it.

Step 2 – Choose the Rig

You can define the sound effect to be played for either male or female rigs, or both:

Rig	Description
male	The sound effect plays only for male rigs
female	The sound effect plays only for female rigs
all	The sound effect plays for both male and female rigs

Step 3 – Choose a Trigger

For each rig, you specify the event that triggers the sound effect. Each event must have the following information:

- name – the name of the sound to play (from the bank)
- volume – (optional) the volume to play the sound at (0 to 1). The default is 1.0.
- delay – (optional) introduce a delay between the animation start and the sound playing. The default is 0.0 (no delay).

Example Sound Effect Defintions

The following example shows the code for playing different sound effects for male and female rigs:

```
<xml>
 <male>
  <load>bnk.male<load>
  <event on="Attack" volume="1.0" delay="0.5">manly_roar</event>
 </male>
 <female>
  <load>bnk.female<load>
  <event on="Attack" volume="1.0" delay="0.5">hair_raising_scream</event>
 </female>
</xml>
```

To apply the sound effect to both rigs, you use the **<all>** element:

```
<xml>
 <all>
  <load>bnk.audio<load>
  <event on="Attack" volume="1.0" delay="0.5">success</event>
 </all>
</xml>
```

Memory Usage

If you add audio, memory usage differs between remote and local Animation Packs.

This is because audio is not replicated (only the local animation pack user hears the audio) whereas animations are replicated (all users, local and remote, see animations).

Below are some memory guidelines, please confirm full memory requirements on the Memory Limits for Scripted Objects page.

Local Animation Pack

- Main and Host memory usage = 256 KB
- Loads the repertoire for a single gender (animations can be split between Main and Host memory)
- Loads an optional soundbank (to Main or Host)
- Script runs logic to allow sound effect to play (based on animation being played)

Remote Animation Pack

- Main and Host memory usage = 256 KB

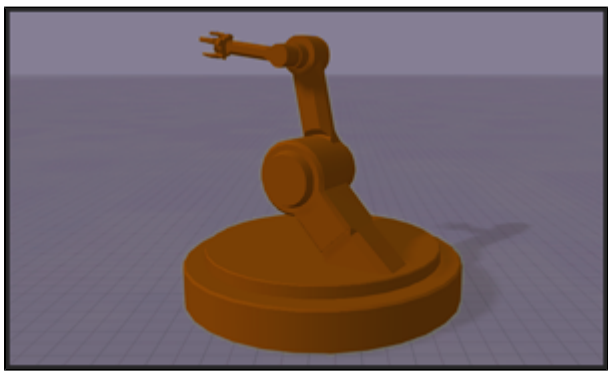
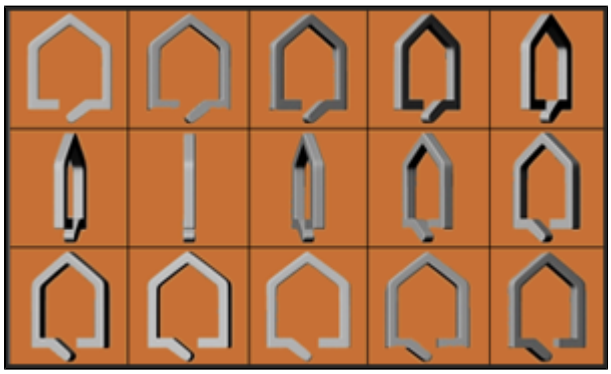
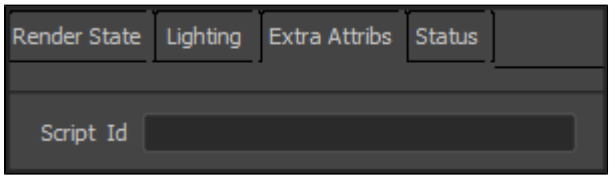
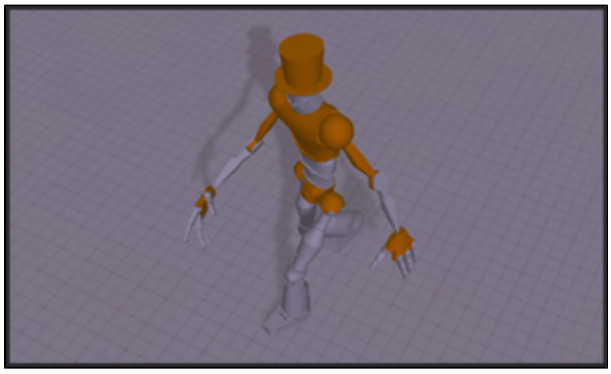
- Loads the repertoire into portables memory – all work is done by the client

Other Animation and Animation Effects

Other Animation and Animation Effects (not including Avatar Animation)

> For a technical outline of what you need to do in Maya to animate anything that is not the avatar – see the Joint Animation and Rigid Body Animation section in the Art Tools.

You can animate almost everything in PS Home. If you can design it in an art tool or attach an entity to it in Lua, you can animate it in some way.

	Creating a Simple Animated Object Create animated models in Maya, export them, and you have animated models ready to go into your scene. Just add an animated model to your scene through the Scene Editor and watch it move. This could be Joint Animation and Rigid Body Animation – see the Joint Animation and Rigid Body Animation section in the Art Tools.
	Scripted Entity Animation Add animation to materials, such as clothing or furniture.
	Scripted Material Animations An ID that can be queried by Lua.
	Scripted Entity Animation Attach model files (.mdl) and animation (.ani) files to an entity, add some Lua animation blending functions, and you have created a very powerful tool for animating in PS Home.

Validating Joint Animations with Joints and Bones

Joint animations (such as non-player characters and animals) are similar to avatar animations. They have must joints and bones and use the combination of `.mdl`, `.skn`, and `.ani` files to produce animations. However, instead of through the Repertoire Editor, as for avatar animation, you call the joint animation files using Lua script, or you add them to a scene using the `Entity Collision` and `Graphics` element in the Scene Editor.

Joint animations are not created using any tools provided in the HDK. You create them in Maya, through the standard animation process. The HDK tools affect only the export process.

See also Exporting Animation Files.

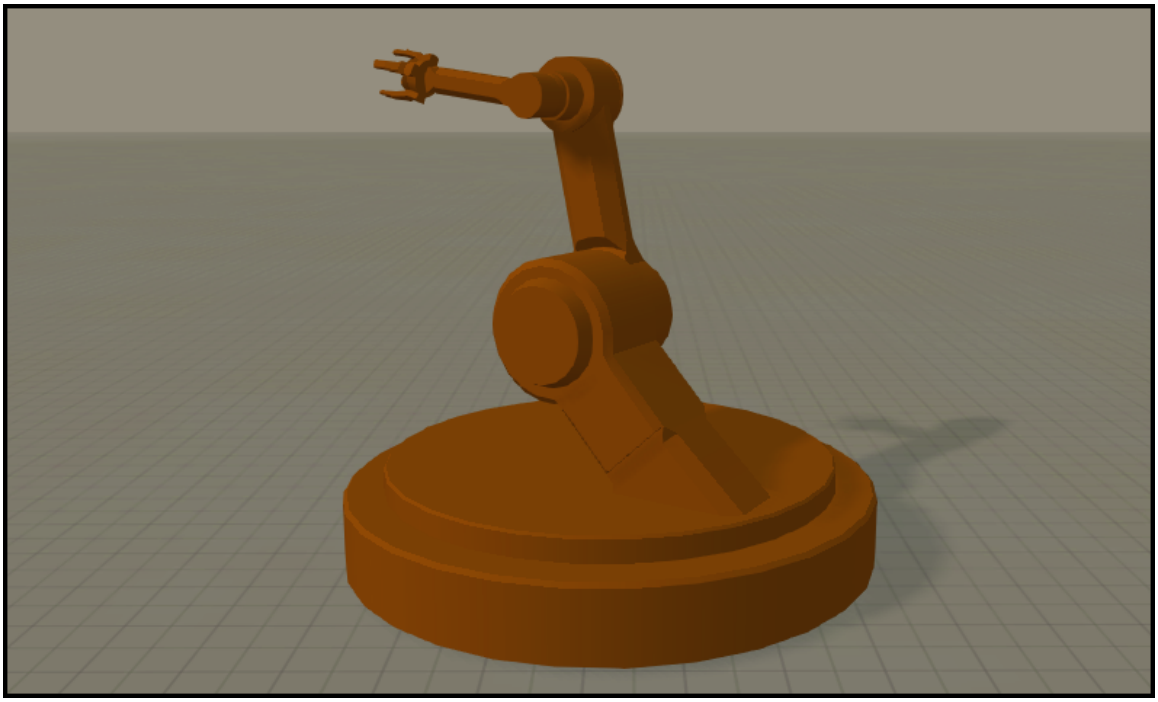
Creating a Simple Animated Object

> This page covers the typical process for creating Joint Animation, but much of the process is generally applicable to Rigid Body Animation as well.

You can use Maya and the Scene Editor to add animation to your scene geometry. This section covers the basic workflow for getting the artwork from Maya into PS Home and displayed in game.

You can export both skinned and unskinned hierarchies of objects. The translation, rotation and scale values (both uniform and non-uniform) of a transform node can be animated and exported. The position of an object's pivot point is also taken into account when the object is animated in PS Home.

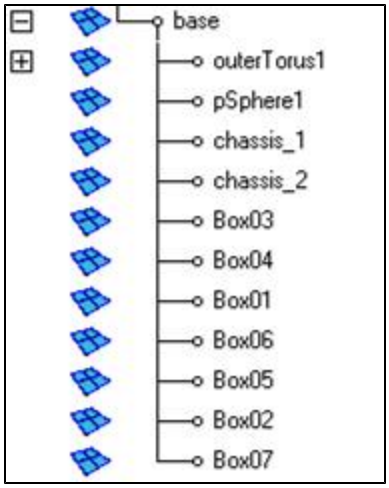
Joint Animation Models In Maya



The first step is to create the model file (i.e. geometry) that you want to animate. This documentation does not show you how to create geometry in Maya.

For more information on creating model files in Maya, see Introduction to Maya.

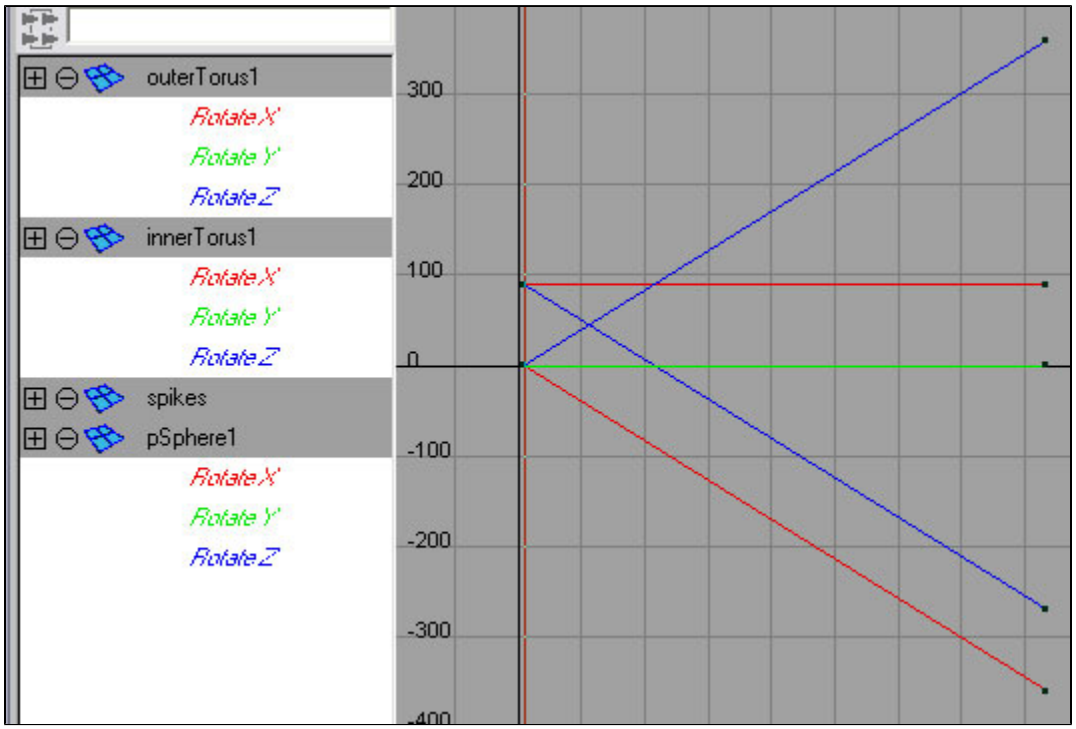
The Simple Animation sample displays a hierarchy of objects:



The shapes have had various rotations applied to them over a number of frames.

Setting up a hierarchy in this way is not a precise requirement of PS Home animated objects. Any sensibly structured hierarchy should export correctly. Equally, the geometry could have been rigidly bound to joints (bones) and those animated. Simple Animation aims to support whatever way the artist prefers to work.

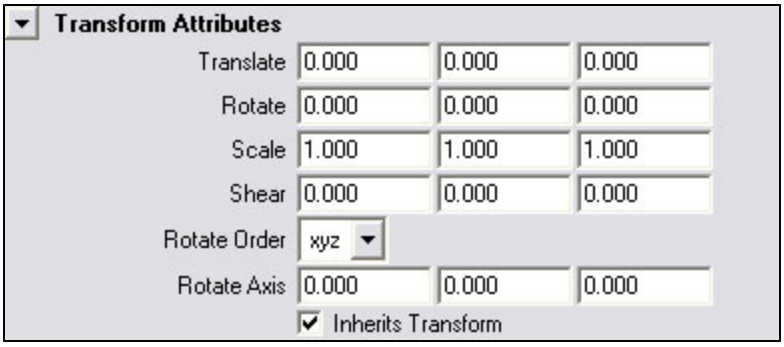
In this example, the animation curves can be seen in Maya's Graph Editor:



The curves here are linear so the objects rotate at a constant speed but once again there are no restrictions on what types of animation curves or animation methods can be used, including the use of additional animation data that could be, for example, created by the use of expressions.

Inherits Transform

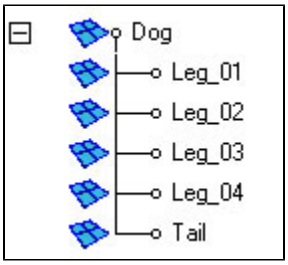
In Maya's Attribute Editor, all transform nodes in Maya have Inherits Transform checked by default:



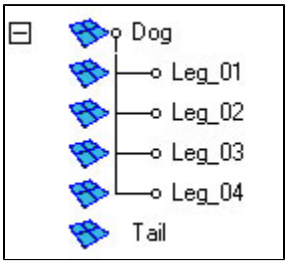
This is what gives the scene hierarchy its power, i.e. if you translate a transform, all transforms underneath it will be translated too. However, you can break this relationship by unchecking the box for every transform group you wish to not inherit.

While this is not recommended practice, it can be useful occasionally. The HDK does not recommend this use, but will support nodes that have the Inherits Transform unchecked. The animation export recognizes this state and supports it by moving the node that should not inherit outside of the original scene hierarchy, thus breaking up the hierarchy.

For example, we may start with a scene such as this where the legs and tail are all parented to the Dog node:



Now if we uncheck Inherits Transform for the Tail transform node, the exporter will modify the hierarchy to reflect the change. It will produce this:



Every transform node with Inherits Transform unchecked will become a root node in the scene (like the Tail node) regardless of how deep it was in the hierarchy before. The exporter removes the 'Tail' from inheriting any transforms by making sure it has no parent node to inherit from.

Users will normally be completely unaware of the hierarchy modification unless there is anything that relies on the hierarchy, which will fail. An example of this would be a Lua script that references a bone hierarchy.

Exporting a Simple Animated Object

This page covers the typical process for exporting Joint Animation, but much of the process is generally applicable to Rigid Body Animation as well.

The Joint Animation pipeline is a way to export skinned animated hierarchies of joints. You can create a custom hierarchy of up to 128 joints, and you can use the export tools to create custom compression profiles for each joint. Files types output are model (.mdl), skeleton (.skn) and animation (.ani).

Joint Animation must have joints.

Rigid body animation cannot have joints.

Exporting Joint Animations

When you have created your animation using the standard animation process for Maya, you can export the animation files using the Home Export Wizard. You can access the wizard from the Home Tool Shelf, or from the Home drop-down menu.

> The Quick Export option repeats the last export sequence you performed without taking you through the Export Wizard dialog.

To export your joint animation files:

1. Click  or select Home > Export Wizard to display the Export Wizard dialog.
2. Select Jonit Animation from the Profile drop-down list.



3. Click  to display the Joint Animation Compression Options dialog:



Joint Animation Compression

The Joint Animation Compression Options dialog allows highly customizable compression options. With these options, you can choose combinations of compression, from a unique profile for each joint, a blanket profile for all joints, or a combination of custom profiles on some joints, with a blanket compression on the rest.

For detailed information the compression tools available for animations see Animation Compression Tools.

Adding a Compression Profile

> Save the compression profiles only after you have tried to export. If you save your animation before attempting to export, the compression profiles are lost.

Saving and Exporting the Animation Files

The Build folder for the object now contains three new files, which will be referenced to create the animated entity:

- .mdl (model) file
- .ani (animation) file
- .skn (skeleton) file

When the export has successfully completed, you can edit the scene or object/component that you created in Maya, and package it in the Scene Editor or Object Editor, as appropriate. The scene or object/component must successfully export and be packaged by the associated Editor before you submit it for Quality Assurance.

Creating an Entity with a Simple Animated Object

Creating an Entity In the Scene Editor

After you have validated and exported the Animated Model (with associated skeleton and animation assets), use the Scene Editor Tool to create an entity with animation.

For creating and exporting see Creating a Simple Animated Object and Exporting a Simple Animated Object.

> ⚠ You must have a Home environment (.scproj file) already loaded.

To create an entity with animation in the Scene Editor:

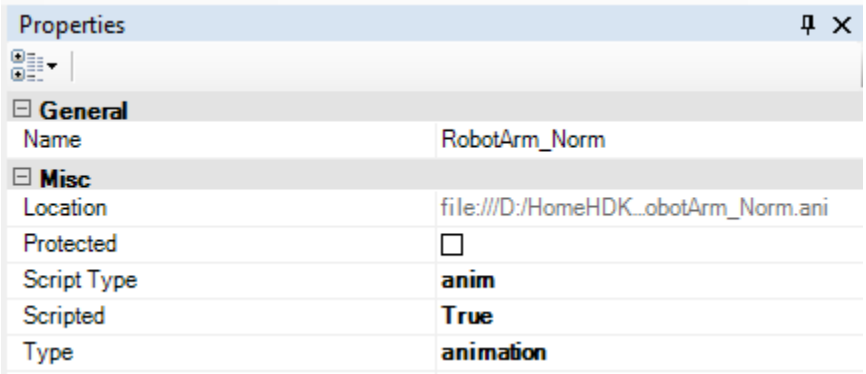
1. Load the assets:

    a. From the Assets panel, right-click the Add Assets icon and select Add > Existing Asset.
    b. Navigate to the Build folder and load the three asset files (.ani, .mdl, and .skn files) one by one.

    Currently all three assets are given the same name in the Assets panel. Although the model file (**RobotArm_Norm_** – see the image below) can easily be identified by its thumbnail image, the skeleton and animation assets will have there names appended with a number to prevent naming conflicts.



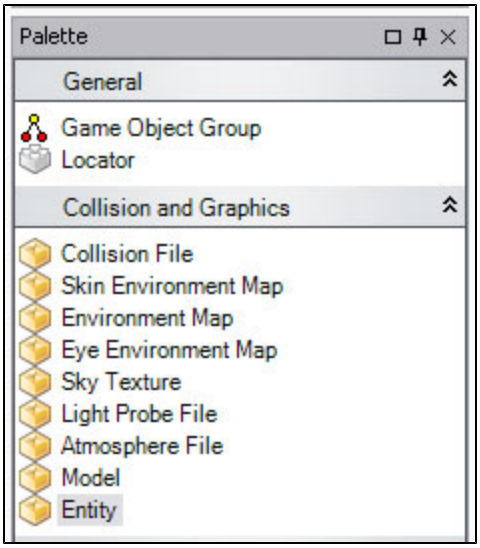
    The Properties panel is a lot more informative. When each of the assets is selected, it will show which is which in the Script Type field. You will also notice that the names of the assets as they will be used are given as xxxxx_1 (animation), xxxxx_2 (model) and xxxxx_3 (skeleton).



    > ⚠ For all three files, **model**, **animation** and **skeleton**, set Scripted to True so that the entity performs correctly.

2. Create a new Game Object of the type Entity to contain the animated object components:

    a. From the Palette, select Collision and Graphics > Entity and drag it into the Work Area:

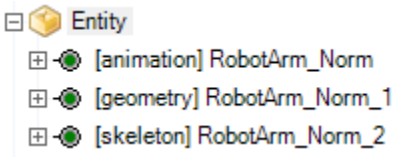


A newly created 'blank' Game Object appears as below in the Work Area and is added to the list of Game Objects for that scene:

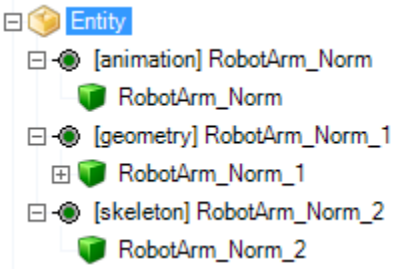


3. Assign Assets to the Entity:

   a. Click ⊞ to expand the Entity. The entity has three children, one for each component.

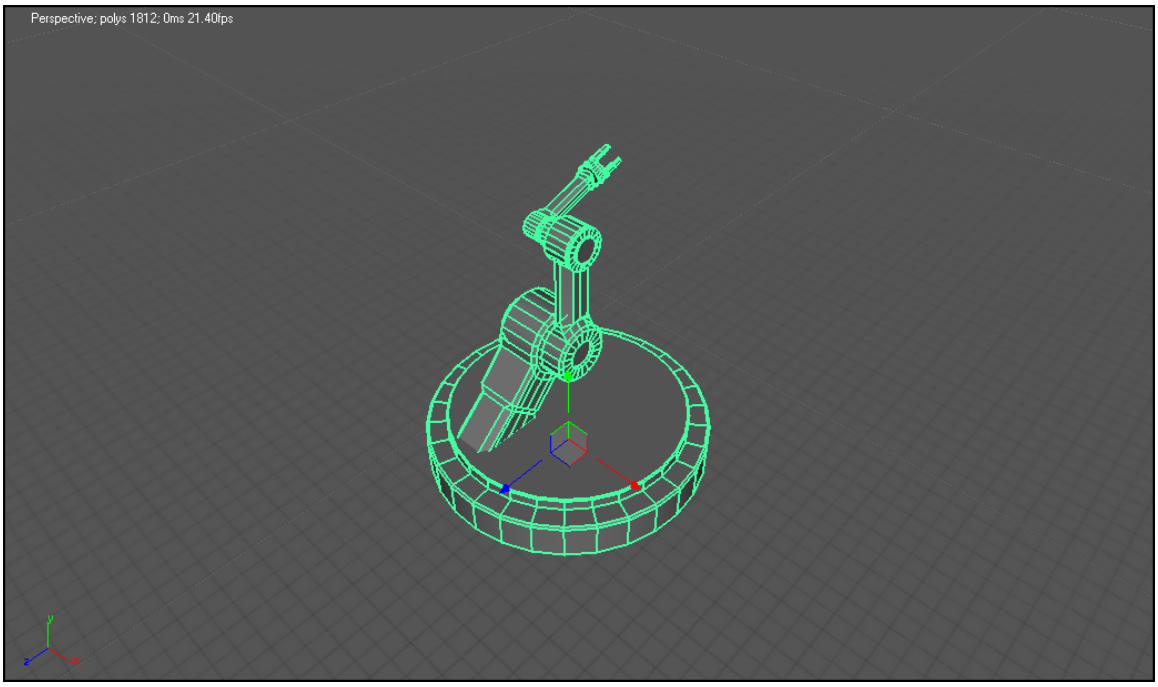


   b. Assign assets to the Entity by dragging each asset to its corresponding node in the Entity folder. The entity is attached as a child of the node (it turns green if this is successful).
   c. Rename the Entity to reflect its contents. An example is as follows:



   The Entity icon also updates in the Work Area to show the animated object's geometry.

4. Use the translation tools to move the Game Object into the scene:



5. Save the updated scene file.
6. Select PS3 > Launch Home to launch the scene in PS Home.

   When the scene is compiled and is running on the PlayStation®3 the animated object appears where it has been placed with the animation looping continually.

Scripted Entity Animation

The Lua API contains functions to load, play, change, and blend animation files (.ani) using the Entity and Animation libraries. There are two ways for scripts to play entity animations:

- Entity.PlayAnim(): Plays a single animation; it stops any other animation on the entity. This method takes a few parameters and is limited.
- Entity.BlendAnimIn(): Blends animations together, playing them simultaneously. This method can play up to 12 animations together. This method is much more fluid, allows you to make complex animations, supports handles and custom callbacks.

This feature is available with the following:

- Active Items
- Mini-games
- Realtime Games
- Model files* - called in through script.

*Model files are not a part of the static scene mesh; they are .mdl files that you add to a scene through the Scene Editor.

This section explains:

- PlayAnim and BlendAnimIn
- Handles
- Setting priority
- Combining PlayAnim and BlendAnimIn
- Other Entity animation functions
- Entity animation guidelines

Entity.PlayAnim()

This method calls the function to play one animation. Any other animations already playing on the entity stop. The following Lua API Reference example creates an entity, loads the model and skeleton files, and then plays an animation.

```
entity = Entity.Create()

Entity.SetModel( entity, "mymodel" )

Entity.SetSkeleton( entity, "myskel" )

--Play the one animation.

Entity.PlayAnim( entity, "myanim" )
```

Entity.BlendAnimIn()

This method enables you to play up to 12 animations on a single entity at the same time. The animations are blended with each other using blend weights, where the higher the weight the more a particular animation will show. If you loop the animation, it will play continuously until you stop the animation or blend the animation out.

> The client automatically blends the animations in. You can smoothly blend out a looping animation by calling `Entity.BlendAnimOut()`.

Looping an animation continuously has a performance impact.

Non-looping animations automatically blend out when the animation finishes.

Using Blend Animations

The following steps show the best practice for using blended animations:

1. Create the entity.
2. Create the animation handle(s).
3. Play animations, setting them to handles.
4. (Optional) Blend animations out.
5. (Optional) Reassign handles to other animations.

For example:

```
--Create the entity.

entity = Entity.Create()

entity:SetModel( "mymodel" )

entity:SetSkeleton( "myskel" )


--Create a handle

handle = Animation.Create()


--Play the animation. It is an idle animation that loops, and is assigned to the handle.

entity:BlendAnimIn( "idle", true, 0, AnimBlendType.Linear, 0, 0, handle )


--Blends the animation out.

if (handle:GetTime() > 30 ) then

      entity:BlendAnimOut( handle )

end


--Reassign the handle to another animation.

entity:BlendAnimIn( "spin", true, 0, AnimBlendType.Linear, 0, 0, handle )
```

Handles

You can give animations a handle. A handle is an animation object created by calling `Animation.Create()`. The handle is attached to an animation file and then used to set the blend weight, retrieve information on its animation (blendstate, duration, etc), and for clean management of animation files.

Task	Function
Create a handle	`handle = Animation.Create()`
Assign a handle to an animation	`entity:BlendAnimIn( "animationFile", true, 0.0, AnimBlendType.Linear, 0.0, 0, handle )`
Reassign a handle to another animation	`entity:BlendAnimIn( "anotherAnimFile", true, 0.0, AnimBlendType.Linear, 0.0, 0, handle )`
Assign one handle to the value of another	`handle1 = Animation.Create()` `handle2= handle1`

Handles are the appropriate method for managing an entity's animations. They have the following purposes:

- Retrieve Information: The `Animation` library contains functions for retrieving information on an animation that are impossible to retrieve otherwise. For example:

  `entity:BlendAnimIn( "idle" )`: Calling this function will play the animation file called idle. No handle is set to it, so it just plays, and none of its settings can be retrieved.
  `entity:BlendAnimIn( "idle, handle )`: Plays the animation file called idle, and assigns it to the handle.

  Now the script can call functions like `Animation.GetBlendState()` to retrieve that the blend state is `AnimBlendType.Linear`, or `Animation.GetName()` to see that the name of the animation file being played is `idle`.

  Also see `Entity.GetActiveAnim()`.

- Swap Animations: The handle method is also ideal for changing animations. For example:

```
--Changes the idle animation if it has been playing for longer than 30 seconds.
entity:BlendAnimIn( idle1, idleHandle )
function OnUpdate( )
      if handle.GetDuration > 30 then
           entity:BlendAnim( idle2, idleHandle )
      end
end
```

In the example above, the idle animation plays and is assigned to the handle idleHandle. When the idle animation has played for more than 30 seconds, the script changes the idle animation that is played. This also makes the idle animations occupy only 1 of the 12 available slots on an entity. Without using a handle, 2 slots would have been taken.

- Clear Entity Animations Slots: For an animation to clear from one of the 12 slots, one of the following must occur:
    - (non-looping animations) The animation finishes playing.
    - Entity.StopAnim() is called on the animation.
    - Entity.BlendAnimOut() is called and the animation reaches the end of the blend duration.

Setting Priority

Both Entity.PlayAnim() and Entity.BlendAnimIn() have a priority argument allowing you to specify when animations are processed in relation to each other. The lower the number, the higher the animation's priority. If two or more animations are given the same priority, the animations will be processed/blended in the order that they were called.

For example, take three animations:

1. Points left arm straight up. (Animation name 'up.anim')
2. Points left arm straight down. (Animation name 'down.anim')
3. Points left arm to the side. (Animation name 'outToSide.anim')

If you set the priority as c=0, b=1, a=2, the system:

1. Evaluates outToSide.anim.
2. Evaluates down.
3. Blends down with outToSide. This is blend 1 result.
4. Evaluates up.
5. Blends up with blend 1 result.

If you set the priority as a=1, b=2, c=3:, the system:

1. Evaluates up.
2. Evaluates down.
3. Blends down with up. This is blend 1 result.
4. Evaluates outToSide.
5. Blends outToSide with blend 1 result.

If animations are played without a priority, they will be processed in the order that they are called.

Combining Entity.PlayAnim() and Entity.BlendAnimIn()

Entity.PlayAnim() stops any other animation playing on the entity, and plays the one called in the function. Entity.BlendAnimIn() plays animations on top of each other, even on top of animations called using Entity.PlayAnim(). For example:

```
entity = Entity.Create()
Entity.SetModel( entity, "mymodel" )
Entity.SetSkeleton( entity, "myskel" )
--Plays the idle animation on the entity.
Entity.PlayAnim( entity, "idle" )
--Plays the look around animation on top of the idle animation.
lookAroundHandle = Animation.Create()
entity:BlendAnimIn( "lookAround", true, 0.0, AnimBlendType.Linear, 0.5, 0, lookAroundHandle)
```

In the example above, the idle and lookAround animations are always playing on the entity. The lookAround animation will blend into the idle animation.

Other Entity Animation Functions

- `Entity.PauseAnim()`: Pauses all currently playing animations on an entity.
- `Entity.ResumeAnim()`: Resumes all animations on an entity if they are paused.
- `Enity.StopAnim()`: Stops the specified animation on an entity. If no animation is specified it stops all animations playing on the entity.
- `Animation.SetPlaybackRate()`: Applies a speed multiplier to an animation.
- `Animation.GetPlaybackRate()`: Returns the current speed multiplier on an animation.

Entity Animation Guidelines

- Only 12 animations can play on a single entity at any one time. Any addition animations are ignored and if they have a handle will return 'nil' to the handle.
- After you call `Entity.BlendAnimIn()`, the animation plays. For looping animations, this means that they will be processed by the client, and thus have an impact on the client, until the animation is stopped.
- Setting a blend weight to 0 does not stop an animation. It makes it invisible, but the client still processes it.
- Set a priority on animations when the order that the animations are processed or blended is important.

Scripted Material Animations

You can add a passable value to ATG Materials that can be queried in Lua. The value, a ScriptID, is not crucial to any material animations in PS Home, but can be a useful way to identify a particular ATG Material, or to pass values to a script.

You can create animation on models by changing the textures using Lua APIs.

You can use scripted animation with only:

- Active Items
- Mini-games
- Realtime games
- Models added as separate entities

> Models added as separate entities are those that are added to scenes using the Scene Editor (i.e. models that are not a part of the static scene mesh).

With a ScriptID attached to an ATG Material you can, for example:

- Swap textures on a model at runtime.
- Set constants on the shader (for example, modulate shader output color, or fade items in/out).
- Perform UV animations more complex than scrolling.

To script object materials, you need to add a ScriptID attribute in Maya , and script the Lua interface. Adding the ScriptID attribute allows you to name the material that you want to animate using the Lua interface.

The APIs you need to add script to your objects are:

- `Entity`
- `Material`
- `Vector4`

The Material API has the following functions:

- `SetAlphaRef( material, value )`: Sets the alpha reference value for when the blend mode is 'cutout'. Pixels with an alpha less than this value are treated as invisible, pixels with a larger alpha are opaque.
- `SetBlendMode( material, blendMode )`: Sets the blending mode (for example, opaque/blend/cutout, etc.) so alpha blending can be turned on/off dynamically.
- `SetDepthPrepass( material, enable )`: Enables or disables a depth prepass for a material. This allows you to make an object transparent without seeing all the internal bits of the mesh.
- `SetVector( string name, Vector4 value )`: Sets the shader uniform with the given string name to the given Vector4 value (for example, `material:SetVector("tintClr", tintCol)`).
- `SetMatrix( string name, Matrix44 value )`: This is the same as the `SetVector` function, except it allows scripted control of the texture matrix.
- `SetTexture(string name, Texture tex )`: This function replaced a texture on the material.
- `GetParam( string name )`: This function gets the value of a float extra attribute that was added to the material, which allows parameters for the material animation to be passed in by artists. An example use of `GetParam` would be if you have three flashing objects in a scene, and you want each to flash at a different speed. Give each object the same ScriptID (for example, 'flash'), and add the speed as an extra attribute. When you call `material:GetParam("speed")` Lua retrieves a table of all of the flashing materials, and adjusts the flash speed for each object.

If you wish to animate the tint or emissive color, add the `tintClr` or `emissiveClr` attribute, respectively, to the Extra Attribs tab as well.

Available tintClr custom attributes:

- tintClrR
- tintClrB
- tintClrG
- tintClrA

Available emissiveClr custom attributes:

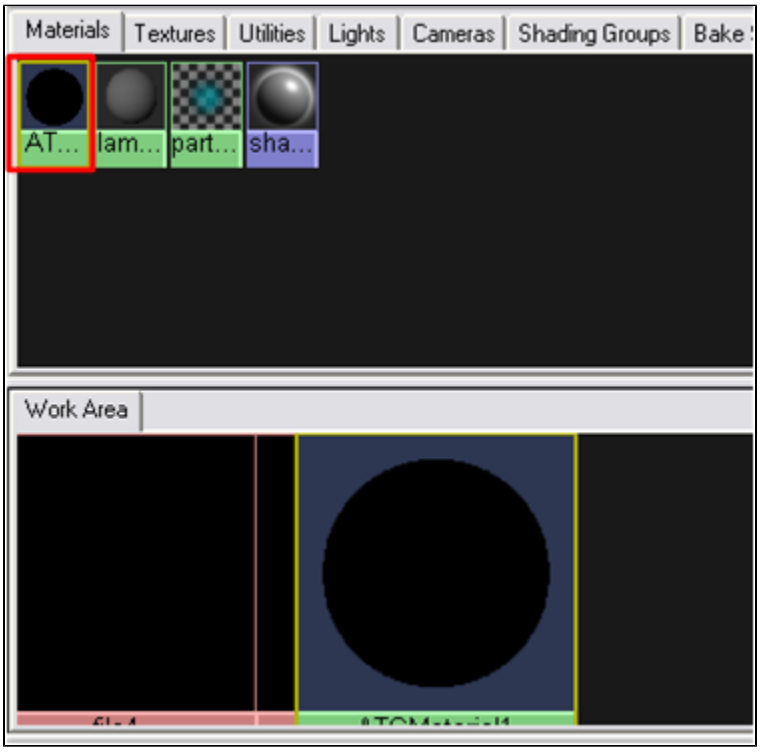
- emissiveClrR
- emissiveClrB
- emissiveClrG
- emissiveClrA

Adding Scripted Animation to your Objects

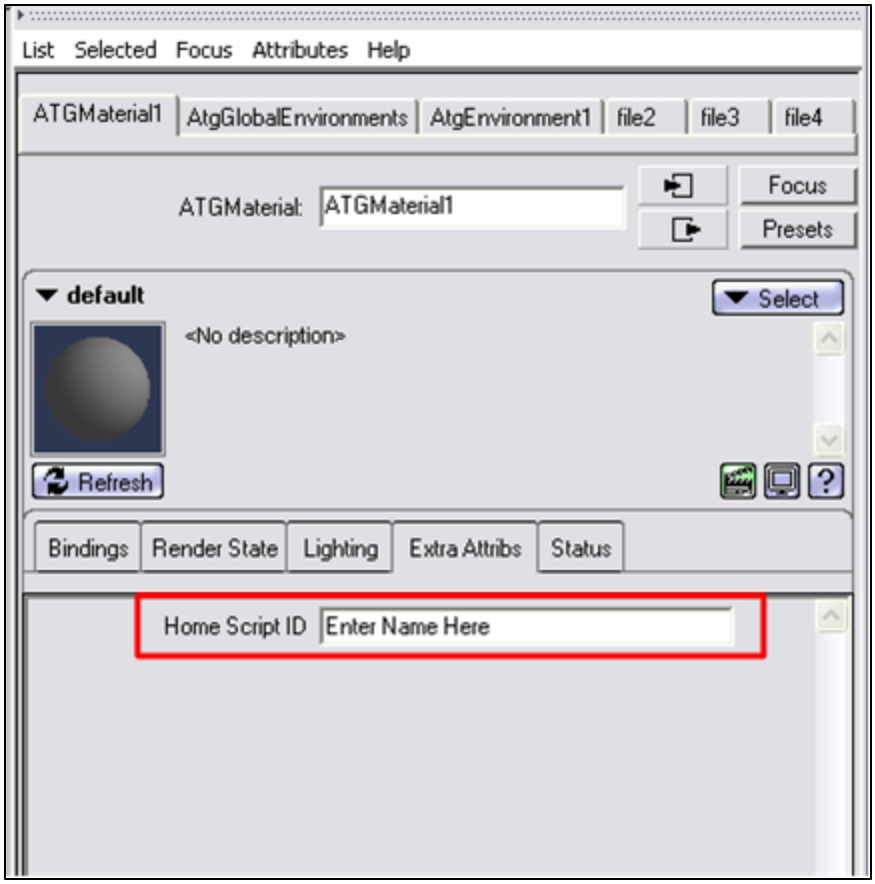
Maya

To add a ScriptID attribute to each ATG material you want to reference in Lua:

1. Select the ATG Material that you want to script:



2. Select Home > Material Effects > Add ScriptID Attribute.
3. Enter a name for the Script ID in the Home Script ID field:



> The same Script ID can be used on several materials. If you want several objects in your scene to have a flash animation, you can name the Script ID on each "flash".

4. If you want to animate the tint or emissive, select Modify > Add Attribute. A dialog is displayed.
5. Complete the required information:
    - Enter the name of the property you want to animate (for example, tint or emissive) in the Long name field.
    - Select Float under Data Type.
6. Click Add and then click OK. The ScriptID and other attributes appear in the Extra Attribs tab.

Lua

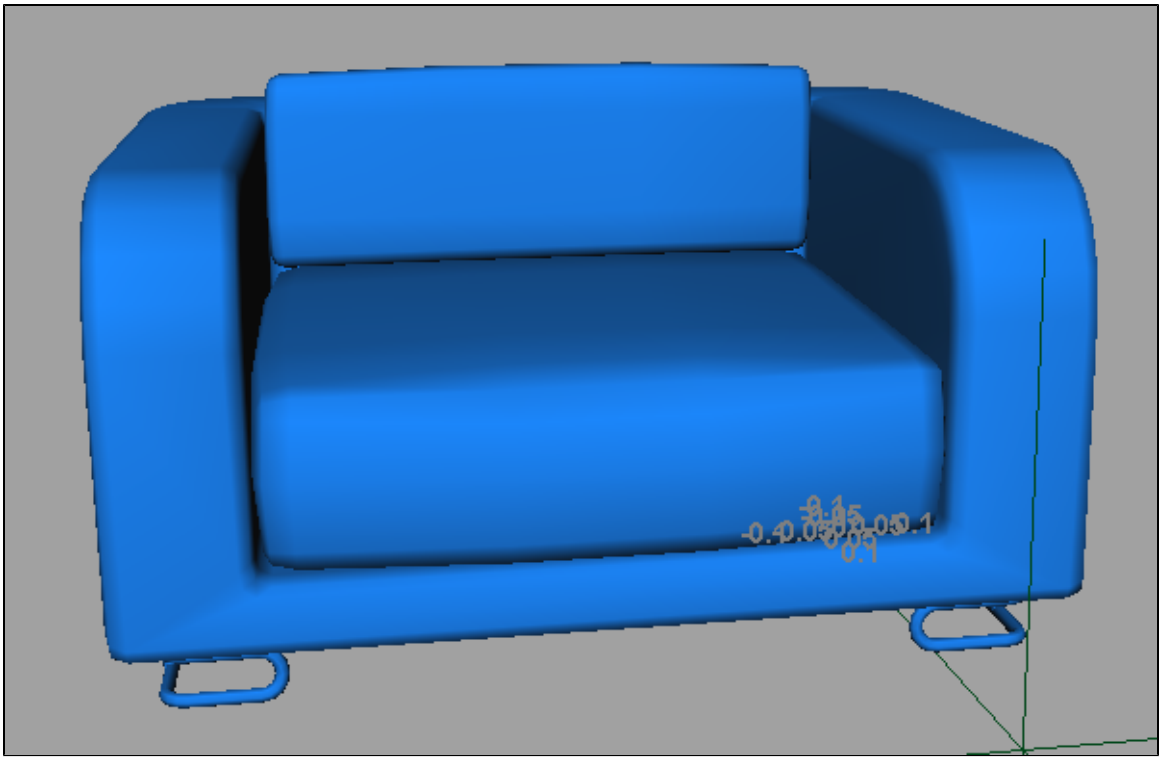
1. Create an entity.
2. Set your model on it using the function `Entity.SetModel()`.
3. Get an array of materials from the entity by calling `Entity.FindMaterials()`.
4. Iterate over the materials, setting the attributes you want to animate using the appropriate functions, for example, `Material.SetMatrix()`.

Scripted Material Animation Example

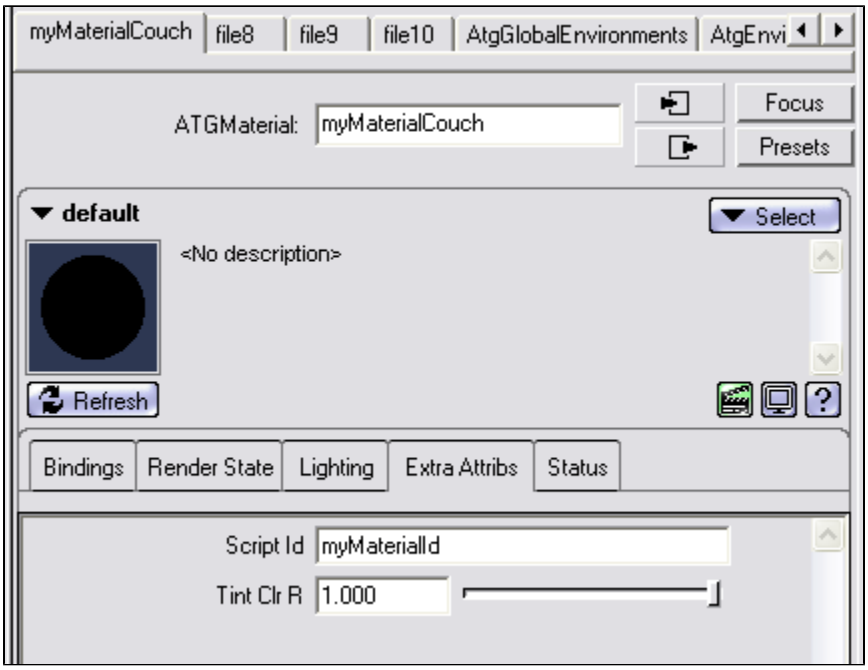
Material Animation

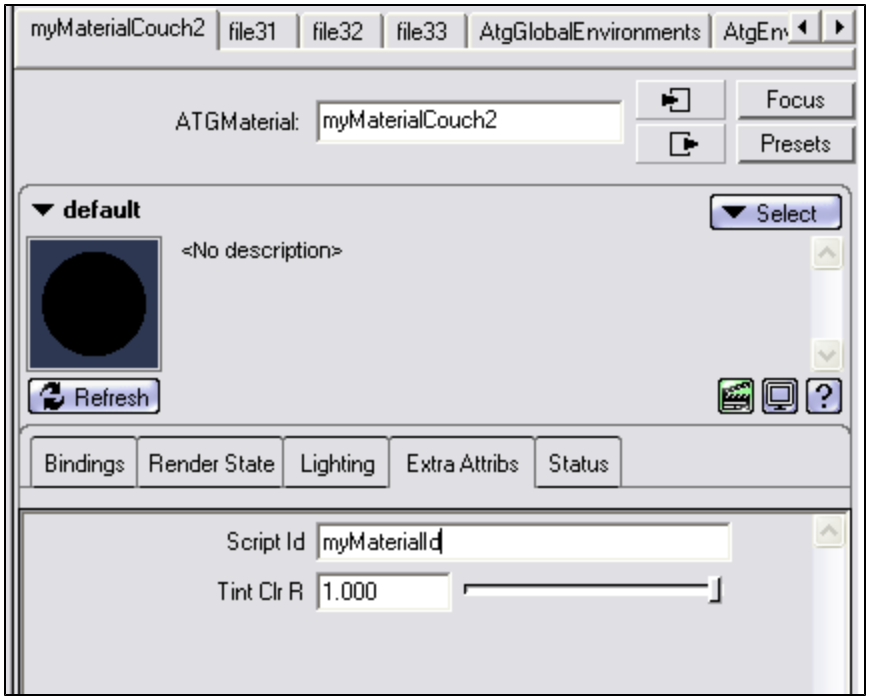
Create a model with at least one ATG material that you want to reference in Lua.

This example uses an active item, which has two ATG materials with a Script Id.



The Script Id for both of the ATG Materials is myMaterialId.





Each ATG material also has the custom attribute tint Clr R so that the tint color can be changed using Lua.

Lua

```
LoadLibrary( "Matrix44" )

LoadLibrary( "Vector4" )

LoadLibrary( "Entity" )

LoadLibrary( "Texture" )

LoadLibrary( "Material" )


myEntity = nil

myMaterials = nil

pulseSpeed = 2


function Init()

-- create entity

myEntity = Entity.Create()

-- set the model

myEntity:SetModel("my_model.mdl")

-- fetch all materials with ScriptID "myMaterialId"

myMaterials = myEntity:FindMaterials("myMaterialId")

end


function AnimateMaterial(time)

-- fade the object in/out over time - note for this to work the material type must support alpha
blending.

local alpha = 0.5*math.sin(time * pulseSpeed) + 0.5

local tintCol = Vector4.Create(1,1,1,alpha)


-- iterate over the materials, setting the tint colour

for i,mat in ipairs(myMaterials) do

mat:SetVector("tintClr", tintCol)

end

end
```

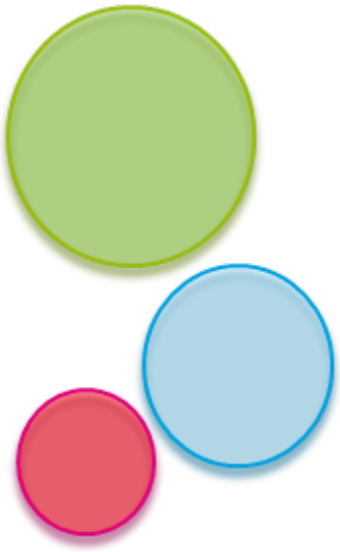
Media

In PS Home you can have screens that:

- Display images
- Play video
- Play audio
- Stream media

In fact, you can play audio anywhere you have a sound sphere. For example, you could have a scripted object that supports sound - like a radio.

You can also upload videos of Home to the web.



Media Memory Usage

> For Japanese Live Page
> 最新の日本語版ページはこちら 「メディアのメモリ使用量」

Screen Memory Usage

A scene's memory budget for screens is separate to the normal scene memory budget and is determined by the type of space (scene type). The table below shows the memory budgets for each scene type.

Scene Type	MAIN (MB)	VRAM (MB)
Public Space	28	16
Public Space (No Video)	0	0
Game Space (16 Players)	0	0
Game Space (32 Players)	0	0
Game Space (60 Players)	0	0
Video Space	43	16
Personal Space	28	16
Clubhouse	28	16

Screen content is always allocated from these budgets, except for images displayed on screens that were created using the Lua function `Screen.Create()`. In this case, the memory is allocated from the scene budget.

The Dev Debug Console command `enableVideoStats 1` displays the following profiling information when the console is open:

Profiling Information	Description

MAIN	The available and total amount of screen MAIN memory.
Largest free	The largest amount of contiguous MAIN memory. Additional videos must be allocated from contiguous memory, so this value indicates the largest amount of memory an additional video can use.
VRAM	The available and total amount of screen VRAM.
Num Videos	The total number of videos in the scene.
C	The number of videos playing from file cache.
P	The number of videos playing through progressive download (the count moves to 'C' when downloads complete).
H	The number of non-cached HTTP video streams.
L	The number of videos playing through HTTP Live Streaming.

If screen content exceeds the available screen memory pool, cache or bandwidth, content may fail to display or load. For more information on cached screen content restrictions and recommendations, and limitations and restrictions that apply to streamed content, see Streaming Content on Screens.

See also:

- Testing, Validating and Submitting Content for more information on Technical Requirements
- Profiling in the Client for more information on profiling

Image Memory

Memory for images displayed on screens is allocated from the screen's 6 MB VRAM budget. The amount of VRAM used depends on the texture compression method used.

The following table shows how many bits per pixel (bpp) are required for each file type:

Source File Format	Texture Compression	Bits Per Pixel
DDS	DXT1	4
DDS	DXT3	8
DDS	DXT5	8
PNG (no alpha channel)	DXT1 (after DDS conversion)	4
PNG (with alpha channel)	DXT5 (after DDS conversion)	8
JPEG	DXT1 (after DDS conversion)	4

When using MIP maps, extra VRAM is required for both the extra pixels in the MIP maps and as padding to accommodate MIP maps with dimensions which are not a power of 2.

The following table shows examples of the difference between file size and actual memory allocated for images with four MIP maps.

Width	Height	File Size (KB)	VRAM Usage (KB)
128	80	6.78	9
272	160	28.3	39
416	240	64.9	91
560	320	116	164
848	480	264	372
896	512	297	420
1024	576	382	540
1280	780	597	845

When MIP maps are not used, the amount of VRAM required for a DDS file is equal to its file size.

Audio

MP3

Memory usage for MP3 files is negligible; 98 KB is allocated from the audio buffer, but no memory from the screen budget is allocated.

Sound Streams

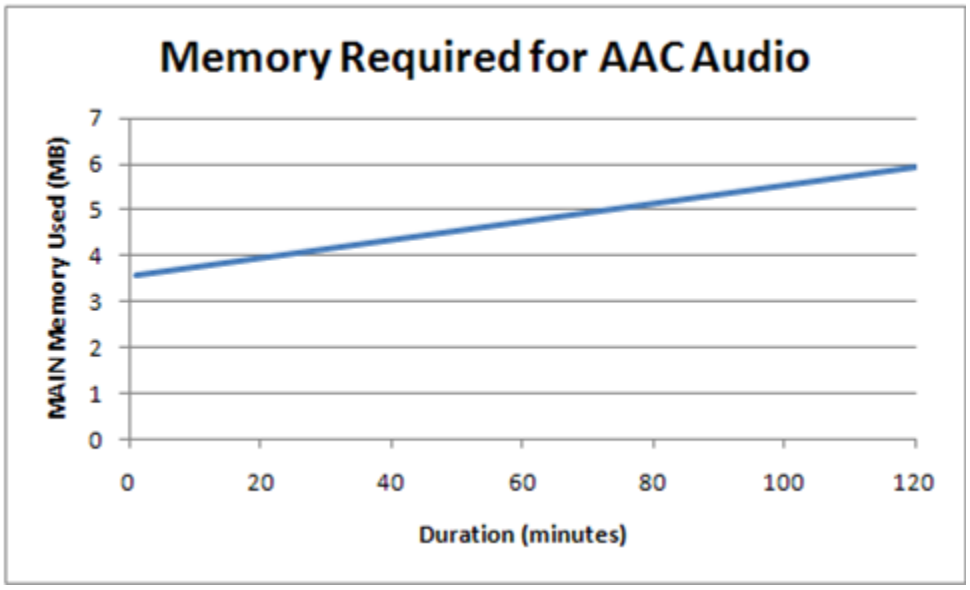
MP3 audio playback or any audio from screens is limited to 4 stereo streams at one time.

AAC

When playing AAC audio, MAIN memory is allocated from the Screen video MAIN memory pool. The minimum memory needed for each screen is approximately 3.5 MB. The actual amount required depends on the duration of the audio file.

There is no specific maximum for the number of screens playing AAC files at once - you are limited only by what will fit within the memory and resource budget for your scene.

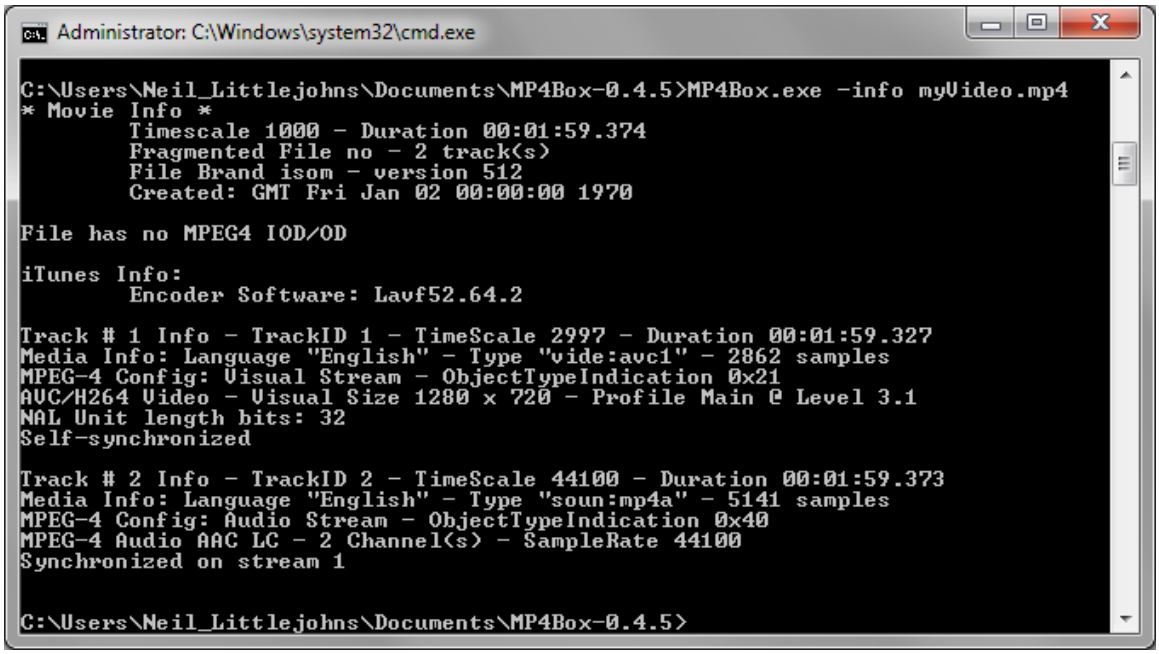
You can use the following graph to estimate memory usage:



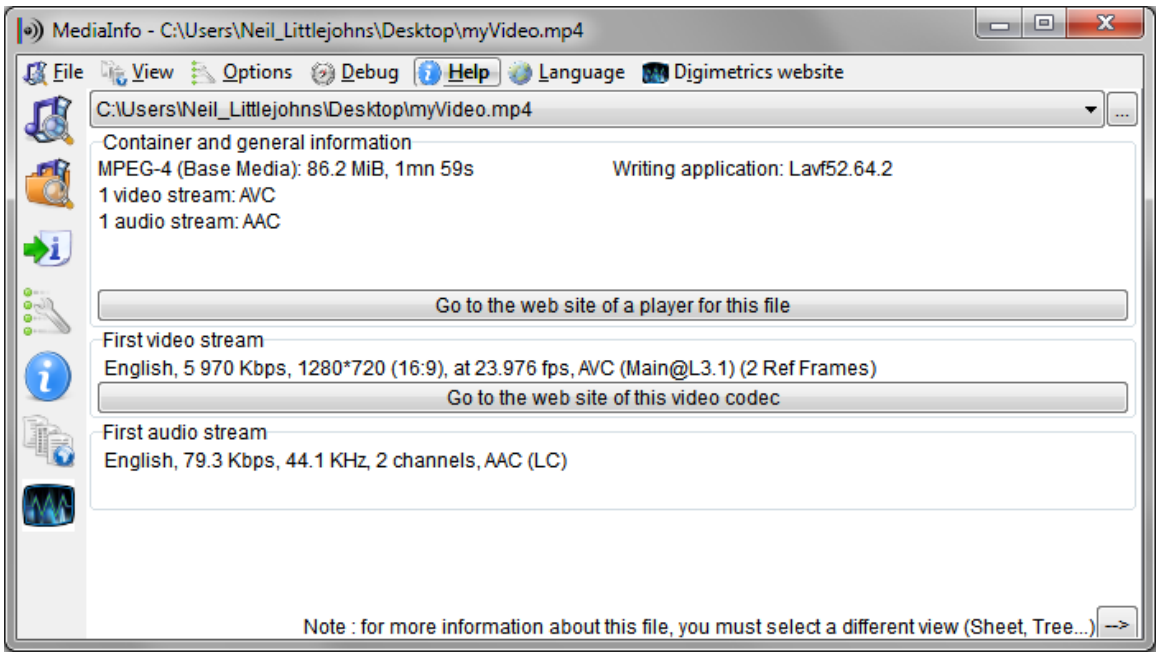
Video

Memory allocated for video playback on screens is determined by the Profile Level of the content. A Profile Level defines a maximum resolution, frame rate and bit rate that a video can use. MPEG-4 video includes Profile Level information in its metadata. This metadata is read by the client and memory is allocated accordingly. If this metadata is incorrect, the PS Home client may allocate more memory than is required in order to play the video, causing memory to be wasted (if the Profile Level is higher than necessary), or the PS Home client may allocate less memory than is required (if the Profile Level is lower than necessary), in which case the video will not play.

The PS Home Developer Client can display Profile Level information using the screendebug video console command (see Troubleshooting Screens). Alternatively, several tools are available to check the Profile Level of a video file. The command line too MP4box can report the Profile Level using the -info flag:



The GUI tool MediaInfo displays Profile Level information upon opening a file:



MPEG-4 Part 2 Simple Profile Memory Usage

This format is suitable for low resolutions and is the least demanding in terms of CPU and memory requirements. It is best used in situations where video quality is not a primary concern, such as in the distant background of a large screen. The following table shows the memory usage for each supported profile level:

Level	Width	Height	Maximum Bit Rate (kbps)	Frame Rate	Total MAIN (MB)	Total VRAM (KB)
1	176	144	64	15	4.72	297
2	352	288	128	15	7.15	1188
3	352	288	384	30	7.19	1188

MPEG-4 Part 2 Advanced Simple Profile Memory Usage

This format supports a wider range of resolutions and bit rates and is suitable for more general use. The CPU and memory

requirements are slightly higher than for Simple Profile, but the resulting image quality is better. The following table shows memory usage for each supported profile level:

Level	Width	Height	Maximum Bit Rate (kbps)	Frame Rate	Total MAIN (MB)	Total VRAM (KB)
1	176	144	12/8	30	5.24	297
2	352	288	384	15	7.15	1188
3	352	288	768	30	7.22	1188
4	352	576	3000	30	7.43	2376
5	720	576	8000	30	7.88	4860

MPEG-4 Part 10 (AVC/H.264) Memory Usage

This format produces the highest quality video playback at much lower bit rates than other formats, but has higher CPU and memory requirements. It is best used in scenes where video playback is the main focus of the scene and/or when playing longer videos as lower bit rates can be used while maintaining video quality. The following table shows memory usage for each supported profile level:

Level	Width	Height	Maximum Bit Rate (kbps)	Frame Rate	Total MAIN (MB)	Total VRAM (KB)
1	128	96	64	29.97	10.68	144
1.1	176	144	192	29.97	12.1	297
1.2	320	240	384	20	13.54	900
1.3	320	240	768	29.97	13.77	900
2	320	240	2048	19.97	13.69	900
2.1	352	480	4096	29.97	18.59	1982
2.2	720	480	4096	15	24.91	4050
3	720	480	10000	29.97	25.96	4050
3.1	1280	720	14000	29.97	32.20	10803

The bit rates given above are the maximum bit rates allowed, not the recommended bit rates. For recommended resolutions and bit rates, see Video Formats.

Maximum Number of Video Screens per Scene

There is no specific maximum for the number of videos playing in a scene - you are limited only by what will fit within the memory and resource budget for your scene. This number depends on how much memory each video uses. When using low resolution video, you can play up to nine different videos at once when using the scene type 'Video Space'.

Cached and streamed video:

- 3 unique videos per screen - you can have a maximum of 3 different (unique) cached videos playing on a screen simultaneously. There is no limit to how many videos a scene can contain, provided that no more than 3 of these videos are played at once.
- keep within memory - as long you do not exceed the memory budget, you can have more than 3 screens playing video at once, provided they are not playing more than three unique videos (see point above).
- max 1 stream - you can have a maximum of 1 screen per scene playing streamed video.

Sharing Screen IDs

If multiple screens share the same Screen ID, the memory required to play the video on all screens is the same as the memory required to play it on one. It is therefore possible to have a large number of screens in a scene as long as they share the same Screen ID.

Bit Rate and Detail

When encoding video, be mindful that the bit rate selected will have an impact on both the quality of the video playback and the amount of time and bandwidth required to download it. Video bit rate represents the amount of data storage used per second of

video, typically measured in kilobits per second (Kbps) or megabits per second (Mbps). The higher the bit rate, the more data the video encoder can utilize to produce the video, resulting in more detail and higher quality video.

You should determine an optimal bit rate for each video released, to keep download times as low as possible. While it is important that the video delivered in PS Home is of a high standard, it is also crucial to consider that many end-users use comparatively slow connections (3 Mb/s or less).

Caching Video

For pre-cached videos, each cached file must not exceed 300 MB. The recommended limit for cached content is less than 100 MB each. For information on cached screen content restrictions and recommendations, see the Streaming Content on Screens section.

Media Formats for PS Home

> For Japanese Live Page
> 最新の日本語版ページはこちら 「PS Homeのメディアフォーマット」

The PS Home client supports a wide variety of media formats, including multiple image and video formats. This section provides information on the media authoring tools that work well with PS Home, and suggests good practice when creating media for use in PS Home.

For more information on the media formats available within PS Home, see:

- Image Formats
- Video Formats
- Audio Formats
- Media Format Reference

Image Formats

> For Japanese Live Page
> 最新の日本語版ページはこちら 「画像フォーマット」

Images displayed on screens should be in DDS format, using the best possible image quality. Several tools are available to create DDS textures from a variety of source formats. PS Home supports the following image file formats:

- JPEG
- PNG (progressive only, not interlaced)
- DDS (using DXT1, DXT3, or DXT5 compression)

The client automatically converts JPEG and PNG files to DDS format to display them. Because this conversion is automatic, you cannot optimize images for the best quality. The resulting DDS textures are scaled to a maximum dimension of 512 pixels. As a result, we recommend the source image is provided in DDS format whenever possible, to ensure the best possible image quality.

DDS Compression Methods

PS Home supports the following DDS compression methods:

- DXT1 RGB: For use with images that do not contain an alpha channel.
- DXT1 ARGB: For use with images that contain a 1-bit (monochrome) alpha channel.
- DXT3 ARGB: For use with images that contain sharp transitions between multiple alpha levels (e.g. one pixel is 100%, the next one is 50%, and another neighbor is 12%).
- DXT5 ARGB: For use with images that contain smooth gradations of alpha levels.

Mipmaps

Generating mipmaps (also known as MIP Maps) for an image creates a number of smaller versions of the image which are seamlessly transitioned between in PS Home. This technique helps reduce aliasing. If an image appears to be heavily aliased in PS Home when viewed from a far distance, consider adding extra mipmaps.

NVIDIA Texture Tools

NVIDIA Texture Tools is a suite of command line tools for image processing and texture manipulation. The suite includes nvcompress, a tool which can be used to batch convert images to DDS textures. The following example command converts input.png to the file output.dds with DXT1 compression:

```
nvcompress -bc1 -rgb input.png output.dds
```

For other compression methods, substitute -bc1 with either -bc2 (DXT3) or -bc3 (DXT5).

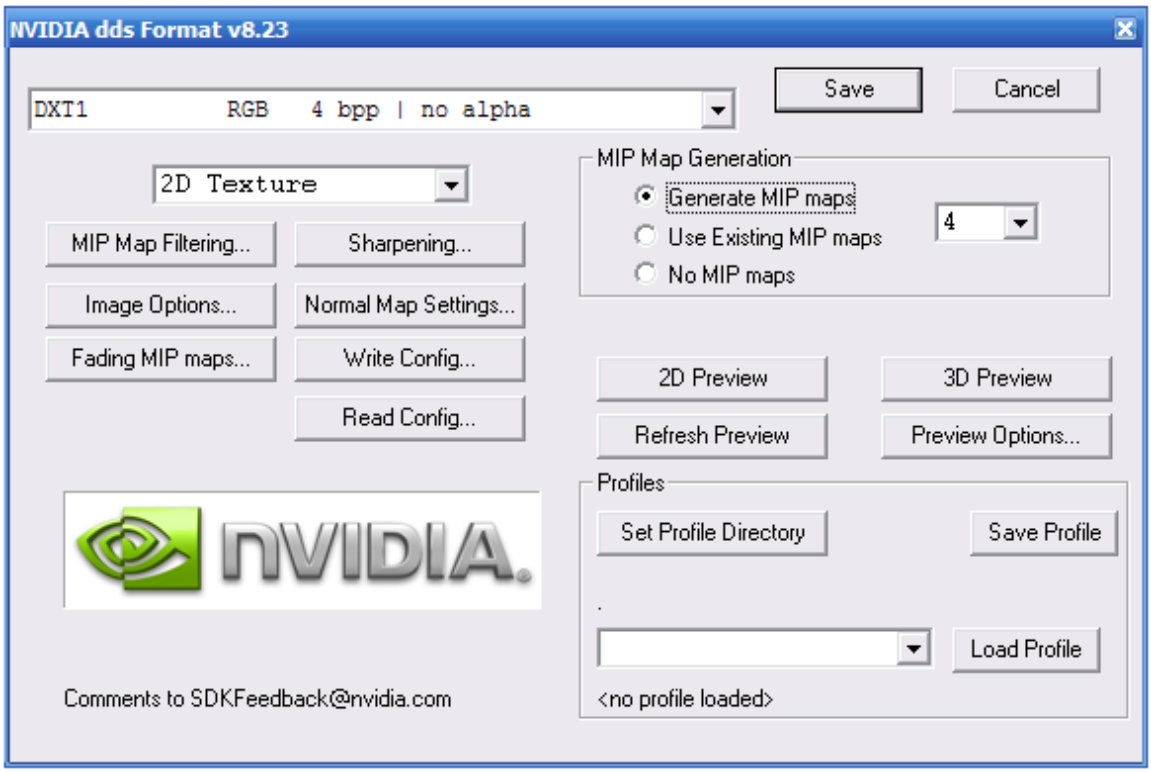
Full documentation for these tools is available from NVIDIA.

NVIDIA DDS Plug-in for Adobe Photoshop

You can download the NVIDIA DDS Plug-in for Adobe Photoshop from http://developer.nvidia.com/nvidia-texture-tools-adobe-photoshop .

To save a file in DDS format using Adobe Photoshop:

1. Select Save > Format > D3D/DDS to display the NVIDIA dds Format dialog:



2. Select the required compression method from the drop-down list and enable MIP map generation, if required.
3. Click Save to save the file.

Paint.NET

To save a file in DDS format using Paint.NET:

1. Select Save > Save as type > DirectDraw Surface to display the Save Configuration dialog:



2. Select the required compression method from the File Format drop-down list and check the Generate Mip Maps box, if required.
3. Click OK to save the file.

Video Formats

For Japanese Live Page
最新の日本語版ページはこちら 「ビデオフォーマット」

PS Home supports the following video codecs:

- MPEG-4 Part 2 (MPEG-4 Video) Simple Profile - up to Level 3
- MPEG-4 Part 2 (MPEG-4 Video) Advanced Simple Profile - up to Level 5
- MPEG-4 Part 10 (AVC/H.264) - up to Level 3.1

AVC/H.264 Level 3.1 (720p, 30fps) is supported only in video spaces. Other types of spaces or scene types (except game spaces) support up to Level 3.0.

PS Home supports the following video container formats:

- MPEG-4 Part 14 (MP4)
- MPEG-2 Part 1 Transport Stream (TS)

If you require audio, you can add an AAC audio track to the container. For more information on MPEG-4 Levels and supported resolutions, see Screen Memory Usage.

The recommended tool for encoding video for PS Home is FFmpeg (or an application based on FFmpeg), though any application which creates files which adhere to supported MPEG-4 standards should be compatible. For best results, make sure that both the video width and the video height are multiples of 16.

Video is automatically stretched to fit the screen, so you don't need to encode video at the correct aspect ratio for it to display correctly (assuming that the screen is the correct aspect ratio). This makes it easier to select video dimensions that are multiples of 16.

Selecting an Appropriate Bit Rate

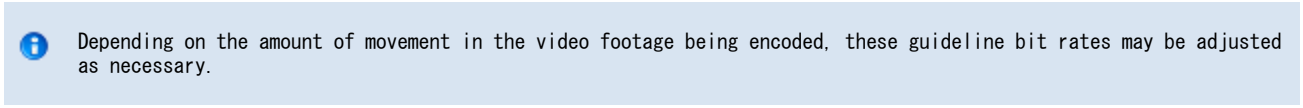
Although PS Home is able to play video with bit rates up to the maximum allowed for each supported MPEG-4 profile level (up to 8 Mbps for MPEG-4 Video and 14 Mbps for AVC/H.264), bit rates this high are rarely required to maintain a high level of video quality. The bit rate required will depend on the following factors:

- Resolution: High-resolution videos require a higher bit rate.
- Frame rate: Higher frame rates require higher bit rates, note that there is no visible benefit to using a frame rate higher than 29.97 frames per second (fps) as this is the frame rate of the PS Home client.
- Video format: MPEG-4 Video requires higher bit rates than AVC/H.264 at the same resolution in order to maintain similar quality levels.
- Video content: Video files with large amounts of fast-moving footage will require a higher bit rate, whereas video containing mostly static backgrounds can use a lower bit rate without sacrificing quality.

For these reasons, it is not possible to specify what bit rates should be used to ensure high-quality video in all situations, but the following examples may be used as guidelines when encoding video:

Format	Width	Height	Frame Rate (fps)	Bit Rate (Kbps)
MPEG-4 Part 2, Simple Profile	176	144	15	64
MPEG-4 Part 2, Simple Profile	352	288	29.97 fps*	384
MPEG-4 Part 2, Advanced Simple Profile	352	288	29.97 fps*	768
MPEG-4 Part 2, Advanced Simple Profile	640	480	29.97 fps*	1536
MPEG-4 Part 2, Advanced Simple Profile	720	576	29.97 fps*	2048
MPEG-4 Part 10 (AVC/H.264)	320	240	29.97 fps*	256
MPEG-4 Part 10 (AVC/H.264)	640	480	29.97 fps*	768
MPEG-4 Part 10 (AVC/H.264)	720	480	29.97 fps*	1024
MPEG-4 Part 10 (AVC/H.264)	1280	720	29.97 fps*	3072

*29.97 fps is a shorthand for (30000/1001) frames per second.



Depending on the amount of movement in the video footage being encoded, these guideline bit rates may be adjusted as necessary.

Using the FFmpeg Command Line Interface

FFmpeg features a wide number of options for video encoding, the full list of encoding options is available at http://www.ffmpeg.org/ffmpeg-doc.html. The most useful options when creating content for PS Home are described below.

Video Options

- **-b [bit rate]**: The target average bit rate of the video in bits per second.
- **-r [frame rate]**: The video frame rate.
- **-vcodec [codec]**: The video codec to encode with. For MPEG-4 Part 2 video, use `mpeg4`, for AVC/H.264, use `libx264`.
- **-aspect [aspect ratio]**: Sets the playback aspect ratio when creating anamorphic content (e.g. `16:9`, `4:3`). This setting is ignored by the PS Home client, but is useful when previewing content on a computer.
- **-vprofile [profile]**: (MPEG-4 Part 2 video only) Sets the video profile, can be either `1` (Simple Profile) or `15` (Advanced Simple Profile).
- **-level [level]**: (MPEG-4 Part 2 video only) Sets the video level, from `1` to `5`.

Audio Options

- **-ab [bit rate]**: The audio bit rate in bits per second.
- **-acodec [codec]**: The audio codec, must be an AAC encoder, e.g. `libfaad` or `aac`.
- **-an**: Do not include an audio track (useful in the first pass of a two-pass encode job).

General Options

- **-i**: The file path of the file to be encoded.
- **-pass [pass number]**: Used to enable two-pass encoding.
- **-fpre [ffpreset file path]**: (AVC/H.264 only) Use an encoding preset.
- **-f mp4**: Sets the video container type to MP4.
- **-threads 0**: Allows the application to determine the best number of threads to use, useful on multi-core processors.

Examples

The following example commands will create an AVC/H.264 video at using 2-pass encoding:

```
ffmpeg -i myvideo.mp4 -b 768k -r 30000/1001 --s 640x480 -vcodec libx264 -aspect 16:9 -an -pass 1 -fpre
C:\ffpresets\libx264-slow_firstpass.ffpreset -f mp4 -threads 0 NUL

ffmpeg -i myvideo.mp4 -b 768k -r 30000/1001 --s 640x480 -vcodec libx264 -aspect 16:9 -ab 160k --acodec
libfaad -pass 2 -fpre C:\ffpresets\libx264-slow.ffpreset -f mp4 -threads 0 output.mp4
```

These commands will create an MPEG-4 Part 2 ASP video using 2-pass encoding:

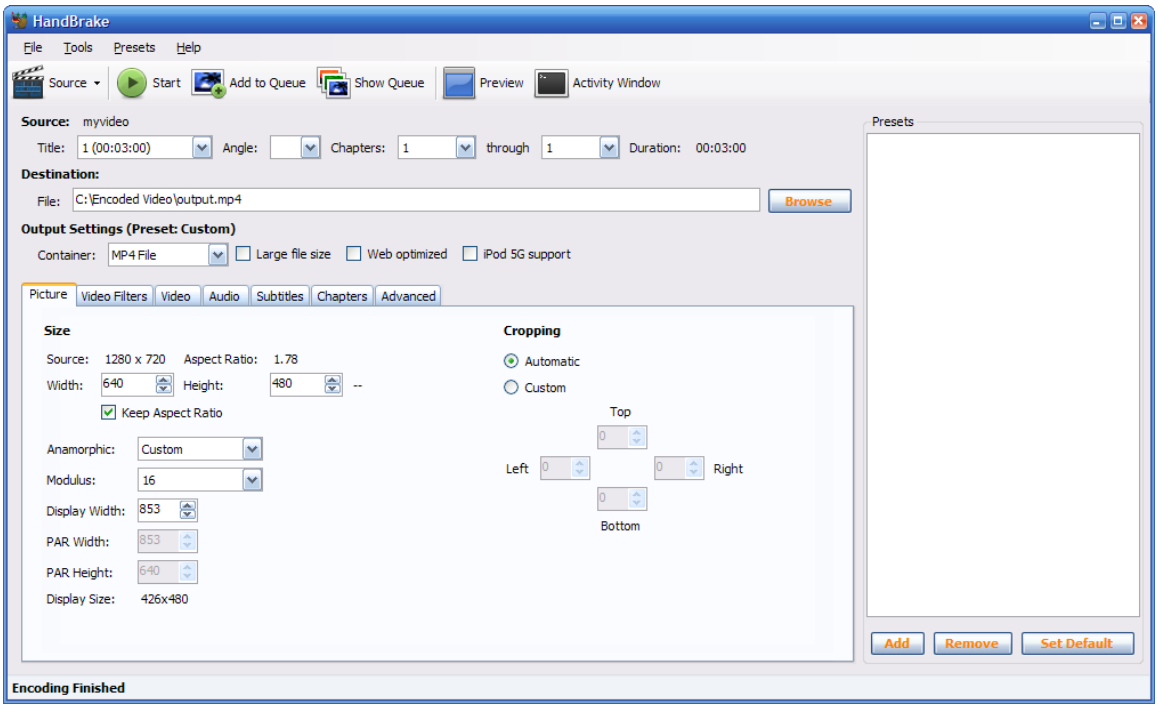
```
ffmpeg -i myvideo.mp4 -b 768k -r 30000/1001 --s 352x288 -vcodec mpeg4 -aspect 16:9 --vprofile 15
--level 3 -an -pass 1 -f mp4 -threads 0 NUL

ffmpeg -i myvideo.mp4 -b 768k -r 30000/1001 --s 352x288 -vcodec mpeg4 -aspect 16:9 --vprofile 15
--level 3 -ab 160k --acodec libfaad -pass 2 -f mp4 -threads 0 output.mp4
```

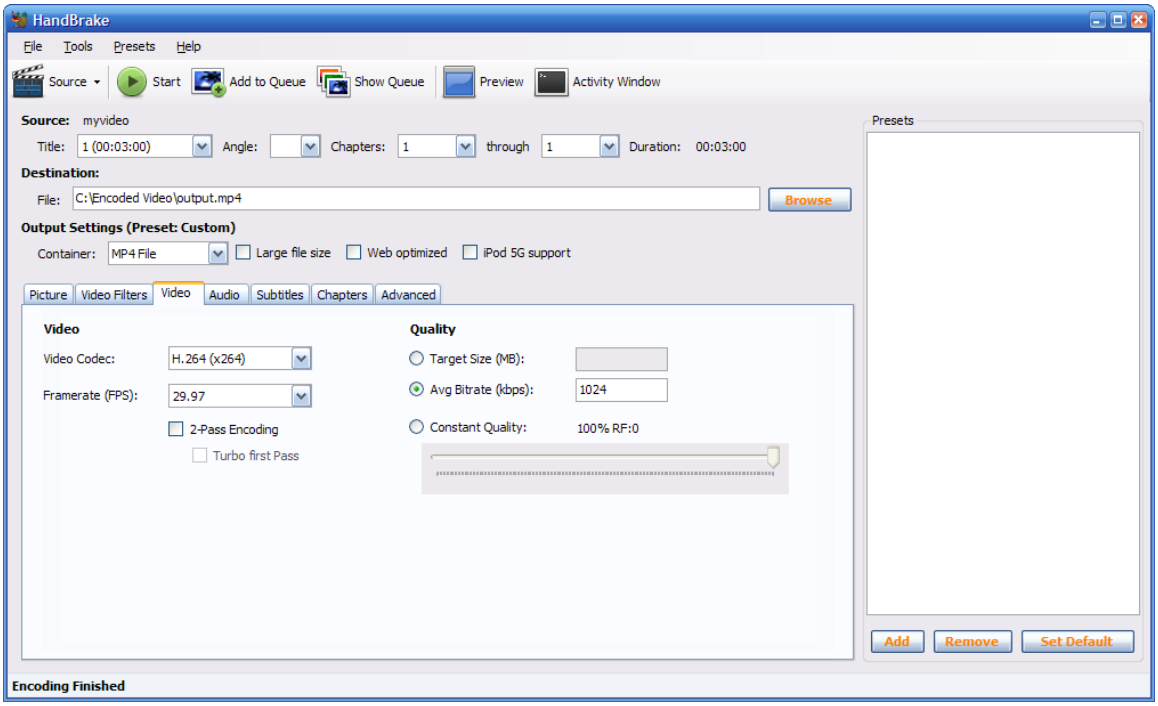
Using an FFmpeg GUI

There are many applications that provide a graphical user interface (GUI) for FFmpeg. As an example, video can be encoded for PS Home using HandBrake as follows:

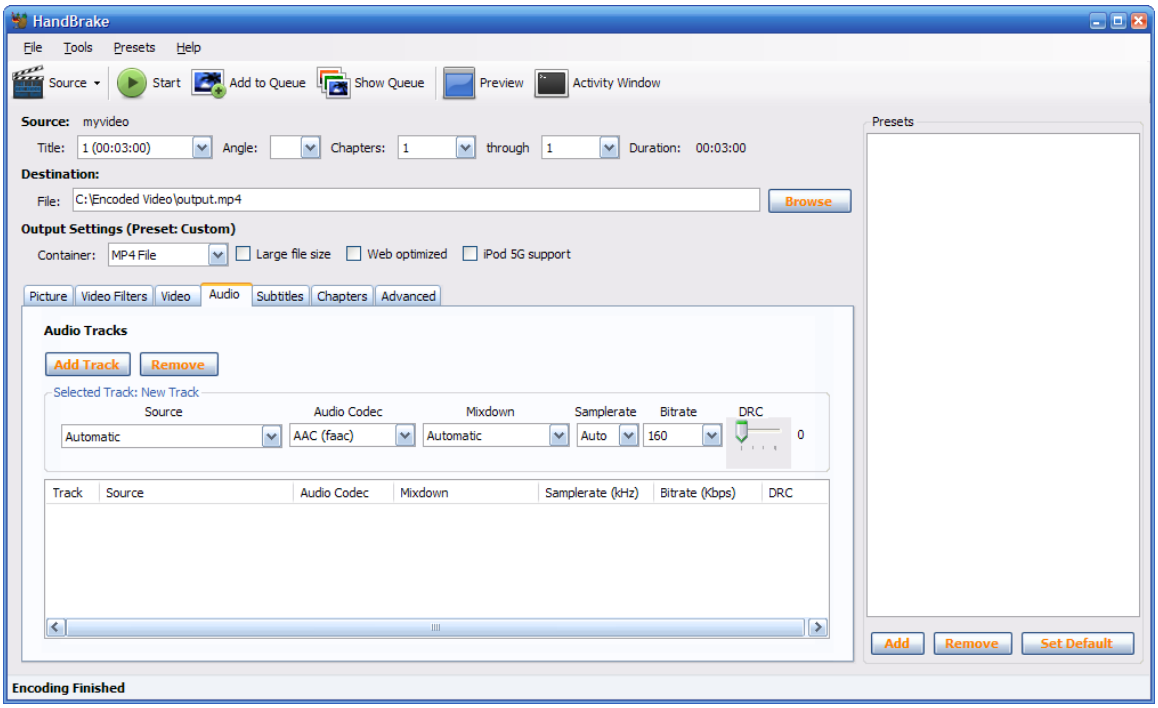
1. From the Source menu, select your video source (equivalent to the FFmpeg command -i).
2. Under Output Settings, select MP4 File from the Container drop-down list(equivalent to the FFmpeg command -f mp4):



3. In the Picture tab, select Custom from the Anamorphic drop-down list, then set the desired video width/height (equivalent to the FFmpeg command -s). For best performance, these values should be multiples of 16. The Display Width field will automatically adjust to the correct value to maintain the aspect ratio of the original file (equivalent to the FFmpeg command -aspect).
4. In the Video tab, select the desired video codec, frame rate and average bit rate (equivalent to the FFmpeg commands -vcodec, -r and -b, respectively):



5. In the Audio tab, ensure that AAC (faac) is selected in the Audio Codec drop-down list and the bit rate is as desired, and then click Add Track:

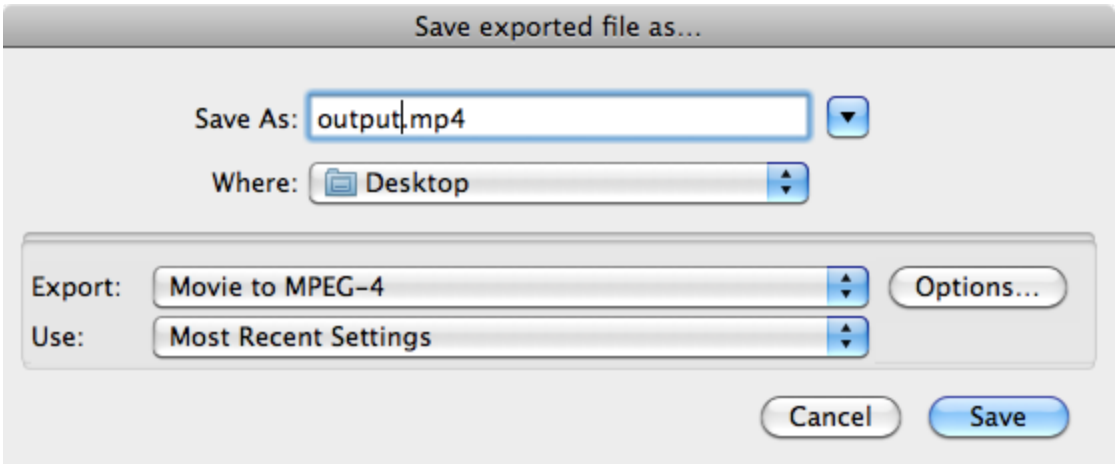


6. Click Start to begin encoding the video.

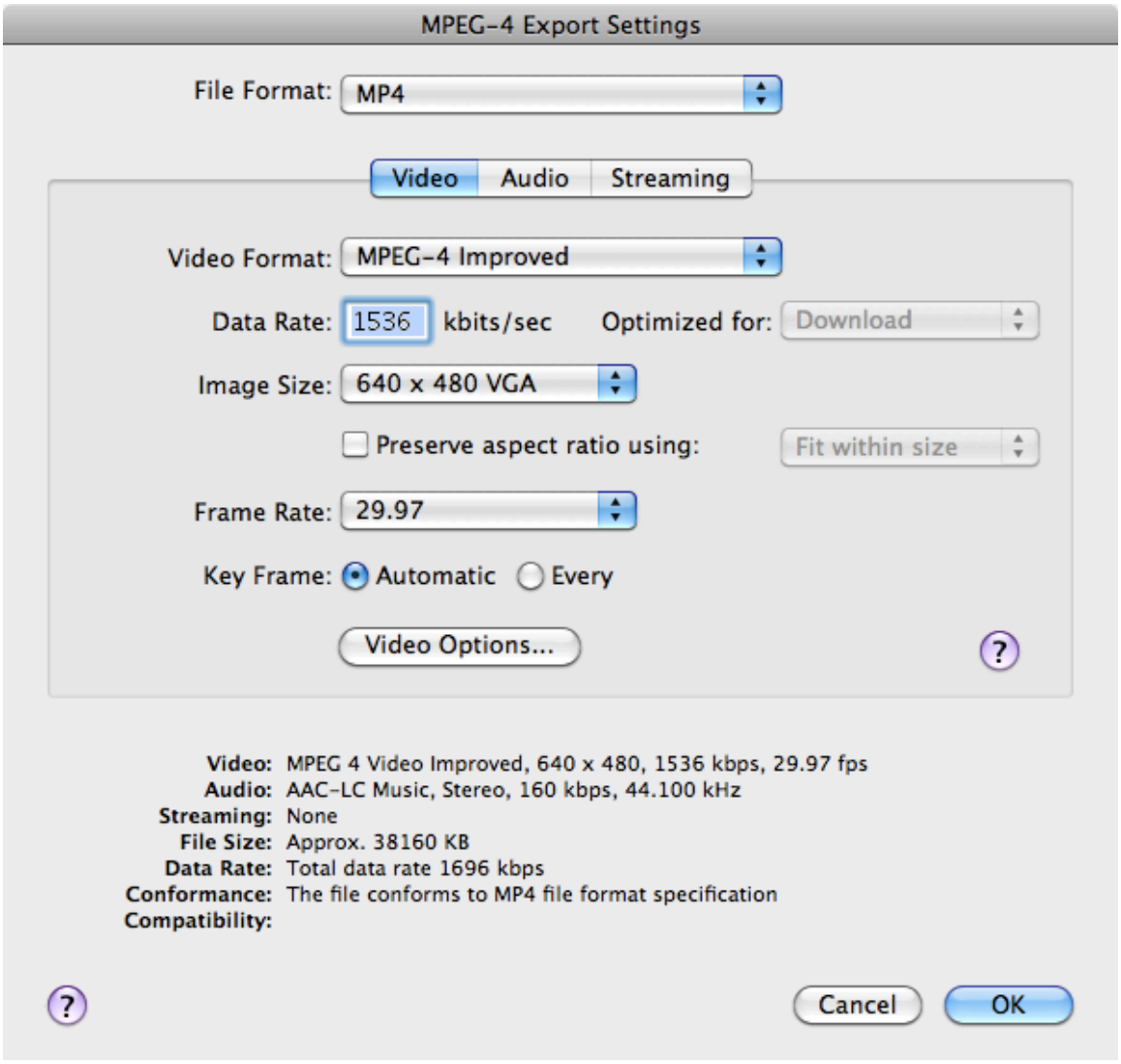
Using Quicktime Pro

To encode a video for playback in PS Home using QuickTime Pro:

1. Open the file and select File > Export.
2. Under Export: select Movie to MPEG-4 and then click the Options... button:



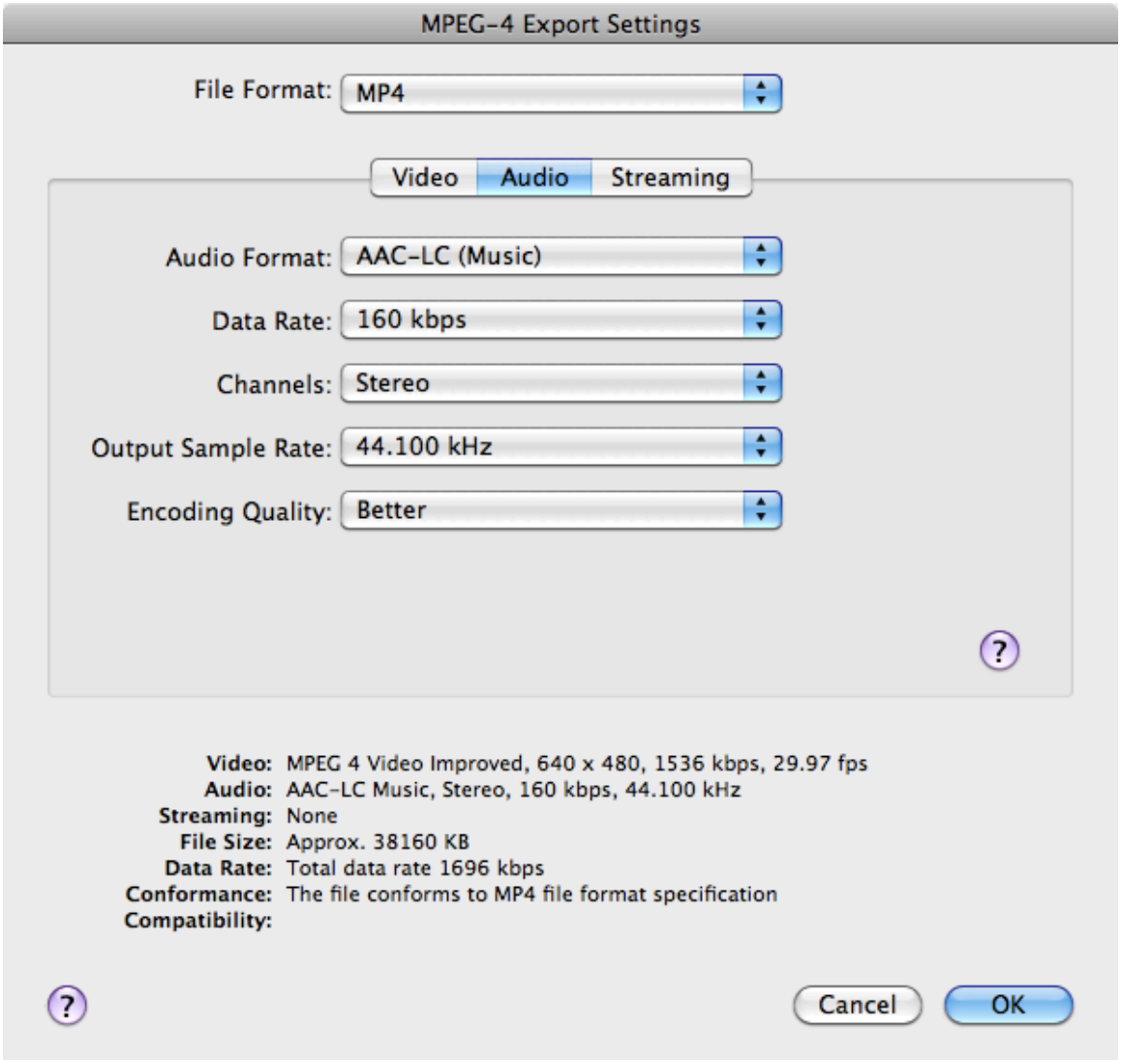
3. If the desired encoding options are unavailable, change the File Format from MP4 (ISMA) to MP4:



4. Under Video Format, select one of the following:

   - MPEG-4 Basic: Encode the video as MPEG-4 Part 2 Simple Profile.
   - MPEG-4 Improved: Encode the video as MPEG-4 Part 2 Advanced Simple Profile.
   - H.264: Encode the video as AVC/H.264.

5. Set the data rate (bit rate), image size and frame rate as desired, then click Audio:



6. Set the Audio Format to AAC-LC and specify the data rate and number of channels as desired.
7. Click OK, then Save to begin encoding.

HTTP Live Streaming

PS Home supports HTTP Live Streaming for audio and/or video delivery. Because HTTP Live Streaming is an emerging technology, the tools available to create streams that are fully compliant with the HTTP Live Streaming specification are not widely available. For this reason, we recommend that a professional video streaming service provider is used to deliver HTTP Live Streaming for PS Home. PS Home has been tested using the Wowza Media Server for both on-demand and live broadcasts.

For more information about streaming video on screen in PS Home, including streaming guidelines, see Streaming Content on Screens .

Video-On-Demand

To support video-on-demand, make the following changes to a Wowza Media Server configuration file (Application.xml):

```
<HTTPStreamers>cupertinostreaming</HTTPStreamers>

...

<HTTPStreamer>


      <!-- Properties defined here will override any properties defined in conf/HTTPStreamers.xml for
any HTTPStreamer loaded by this applications -->

      <Properties>


            <Properties>


                  <Property>


                        <Name>chunkDurationTarget</Name>

                        <Value>10000</Value>

                        <Type>Integer</Type>


                  </Property>


            </Properties>


      </Properties>


</HTTPStreamer>
```



Video must be encoded so that a keyframe appears at the start of every video segment. In the above example, the property chunkDurationTarget is set to 10 seconds, so the video must contain a keyframe every 10th second.

Live Video Broadcasts

To support Live Video Broadcasts, make the following changes to a Wowza Media Server configuration file (Application.xml).

```
<Streams>


      <StreamType>live</StreamType>

      <StorageDir>>$\{com.wowza.wms.context.VHostConfigHome\}/content</StorageDir>

      <KeyDir>$\{com.wowza.wms.context.VHostConfigHome\}/keys</KeyDir>

      <!-- LiveStreamPacketizers (separate with commas): cupertinostreamingpacketizer,
smoothstreamingpacketizer, cupertinostreamingrepeater, smoothstreamingrepeater -->

      <LiveStreamPacketizers>cupertinostreamingpacketizer</LiveStreamPacketizers>

      <!-- Properties defined here will override any properties defined in conf/Streams.xml for any
streams types loaded by this application -->

      <Properties>

      </Properties>


</Streams>

<!-- HTTPStreamers (separate with commas): cupertinostreaming, smoothstreaming -->

<HTTPStreamers>cupertinostreaming</HTTPStreamers>
```

AES Encryption

PS Home supports encrypted playback through HTTP Live Streaming with AES. Streams must meet the following criteria to be played by the PS Home client:

- The encrypted content must be streamed in an HLS format compliant with the draft-pantos-http-live-streaming-06 standard (http://tools.ietf.org/html/draft-pantos-http-live-streaming-06).
- The media files must be encrypted using the Advanced Encryption Standard AES_128 with a 128-bit key and PKCS7 padding.

128-bit AES requires the same 16-octet Initialization Vector (IV) to be supplied when encrypting and decrypting content. Varying this IV increases the strength of the cipher.

> The M3U8 file for an encrypted stream includes all the information needed to decrypt the stream and should therefore be delivered securely.

The following example shows an M3U8 file with encryption information included:

```
#EXTM3U

#EXT-X-VERSION:2

#EXT-X-TARGETDURATION:7

#EXT-X-PROGRAM-DATE-TIME:2011-06-29T14:50:05-04:00

#EXT-X-MEDIA-SEQUENCE:1

#EXT-X-KEY:METHOD=AES-

128,URI="http://www.developer.com/game/key",IV=0x08159d78190b204cc5a83012ff6d1377

#EXTINF:6,

1.ts

#EXTINF:6,

2.ts

#EXTINF:6,

3.ts

#EXTINF:6,

4.ts

#EXTINF:6,

5.ts

#EXT-X-KEY:METHOD=AES-

128,URI="http://www.developer.com/game/key",IV=0x24f8bee42be3780398cb8e3906cfe1ce

6.ts

#EXTINF:6,

7.ts

#EXTINF:6,

8.ts

#EXTINF:6,

9.ts

#EXTINF:6,

10.ts
```

Audio Formats

For Japanese Live Page
最新の日本語版ページはこちら 「オーディオフォーマット」

PS Home supports the following audio codecs:

- MP3: Up to 320kbps, 48kHz, Stereo
- AAC: Up to 320kbps, 48kHz, Stereo

- HE-AAC v1 (AAC+): Up to 320kbps, 48kHz, Stereo
- HE-AAC v2 (eAAC+): Up to 320kbps, 48kHz, Stereo

AAC

AAC audio is treated as video by the client. This is because AAC audio uses the same container format as MPEG-4 video. As a result, AAC audio files should be applied to a screen as video content and should carry the .MP4 extension. If you require audio, you can add an AAC audio track to the container. For more information on MPEG-4 Levels and supported resolutions, see Media Memory Usage.

MP3

MP3s must be encoded at a maximum of 48 kHz. All audio is resampled up to 48 kHz. Any audio from 12-48 kHz is allowed.

MP3 audio playback or any audio from screens is limited to 4 stereo streams at one time.

PS Home does not support variable bit rate (VBR) MP3 files and MP3 Surround (5.1 channel audio).

Media Format Reference

> For Japanese Live Page
> 最新の日本語版ページはこちら 「メディアフォーマットのリファレンス」

General Information

- Media RSS: http://www.rssboard.org/media-rss
- Levels for MPEG-4 Video Profiles: http://www.m4if.org/resources/profiles/index.html
- HTTP Live Streaming : http://developer.apple.com/iphone/library/documentation/networkinginternet/conceptual/streamingmediaguide/Introduction/Int

Content Authoring Tools

- NVIDIA Texture Tools: http://developer.nvidia.com/object/texture_tools.html
- NVIDIA DDS Plug-in for Adobe Photoshop: http://developer.nvidia.com/object/photoshop_dds_plugins.html
- Paint.NET: http://www.getpaint.net/
- FFmpeg: http://www.ffmpeg.org/
- MP4Creator: http://mp4creator.sourceforge.net/
- QuickTime Pro: http://www.apple.com/quicktime/
- HandBrake: http://handbrake.fr/

Introduction to Audio

This section provides information on creating audio for PS Home using the SCREAM Tool Runtime, adapted for the PS Home client. It provides guidelines for producing audio that works well in PS Home.

HDK Tools supports the production of audio by providing the SCREAM Tool Runtime modified for Home (supporting all elements except duckers and mixers). For more information about the SCREAM Tool, see Creating an Audio Asset.

Once you've created the audio, you can add it to your scene by using the Scene Editor.

For more information on using audio in the Scene Editor, see:

- Adding Audio Assets to a Scene
- Adding Sounds to a Scene
- Working with Sound Objects

Creating an Audio Asset

The SCREAM Tool

Scene Editor works with sound bank files created using any version of the SCREAM Tool, offered to all PlayStation®3 developers. The SCREAM Tool offers sound designers the ability to create complex, interactive sounds with a minimum of programmer support. The SCREAM Tool creates sound banks that work in conjunction with the SCREAM runtime engine. You can use the tool in conjunction

with a Sound Tool Server (running on a local PC), so you can audition and tune the audio before handing it to the development team.

> Only the PlayStation®3 version of the SCREAM Tool produces the files required.

SCREAM 1.0.1 is available through the PS Home Developer Network and offers two free applications:

- SCREAM Tool Design sound banks using an interface
- SCREAM Tool Server Audition and tune the audio while running on a PC at the same time as the SCREAM Tool

You can download the SCREAM Tool from https://home.scedev.net/projects/scream_tools.

> Only those with a valid username and password granting access to the PS3 Developer Network website can access this file.

> For Windows 7 users, the environment officially supported by the SCREAM Tool is the Windows 32-bit version only. If you need support for the 64-bit version, contact Technical Support. See Windows 7 Support with SDK 360.

If you are a Home-only developer, contact your regional SCE.

For installation and advice on using the SCREAM Tools, see the SCREAM documentation provided when you download the tool.

Exporting the Audio Files

The Scene Editor requires both the .bank and .bnk versions of the SCREAM sound bank. The .bank file is created when the bank is saved. The .bnk file is created when the bank for PlayStation®3 is exported.

> Both files must be in the same directory. This directory must be within the build folder of the directory containing your custom-created scenes and assets.

In addition:

- Keep the .bnk files to a 2 MB limit.
- Audio memory for a scene is included in the scene main memory limit (see Memory Limits).
- You can have a maximum of 18 sounds (sound streams or sounds in a sound bank) playing in a scene at any one time.

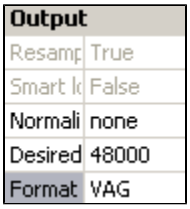
> PS Home does not currently support sounds that have defined looped sections, e.g. a 5 secs sound that plays once, then loops only the last 2 secs. Instead, cut the sound into two, one being the first 3 seconds, the other the last 2 secs. Then play the first sound once, followed by the second sound looped.

Exporting Compressed Audio

To keep memory usage of the audio to a minimum, ensure that every waveform is set to VAG.

To achieve this in the SCREAM Tool:

1. Click on the waveform in the Waveforms panel to select it.
2. In the Waveform Properties panel, select VAG from the Format drop-down list:



Adding Sounds to a Scene

You can place sounds into a scene using Sound Objects.

Types of Sound Objects

Home scenes support the following sound objects:

Sound Object	Description
Sound Zone	Sound Zones represent regions of space in your scene that: • Contain an ambient sound that is only heard when the listener is in that zone. • Specify a reverb setting that is applied to all ambient sounds positioned in that zone.
Ambient Sound	The ambient sound type is heard throughout the scene at a constant volume and panning. You can add more than one ambient sound to a scene.
Point Sound	This sound type is placed in a specific location in the scene, and can be placed within a sound zone. For point sources, several emitter shapes can be chosen. For instance, if two spheres are chosen, the Fall off start and Fall off end settings specify inner and outer spheres of listening. The sound is heard at the full volume specified when PS Home's camera is inside the inner sphere. The sound's volume fades to nothing as the camera moves outwards from the inner sphere.
Sound Portal	Place Sound Portals in a specific location in a scene to provide a smooth transition between two Sound Zones. The background Sound of one Sound Zone (set in the Sound Zone Properties) is blended with a second Sound Zone within the Portal. As PS Home's camera progresses through the Portal, the first sound fades out as the second sound fades in.



Bear in mind the following:

- You must place Sound Portals within Sound Zones. If you place sound emitters outside a sound zone in the scene, the client creates a sound zone that includes the whole scene. If the scene already contains sound zones and some emitters fall outside these zones, these sounds play as normal.
- You use a Sound Zone to allow Voice Chat and other client-created sounds. If you do not want your scene to include any background sound, but you want to allow users to broadcast their voice, you need to create a Sound Zone covering the entire scene. One will not be created automatically.
- Sound Zones must be overlapping in order for the Portal to work.

Listener Location

The listener can come from either the user's position or the camera's position, depending on what the user is doing:

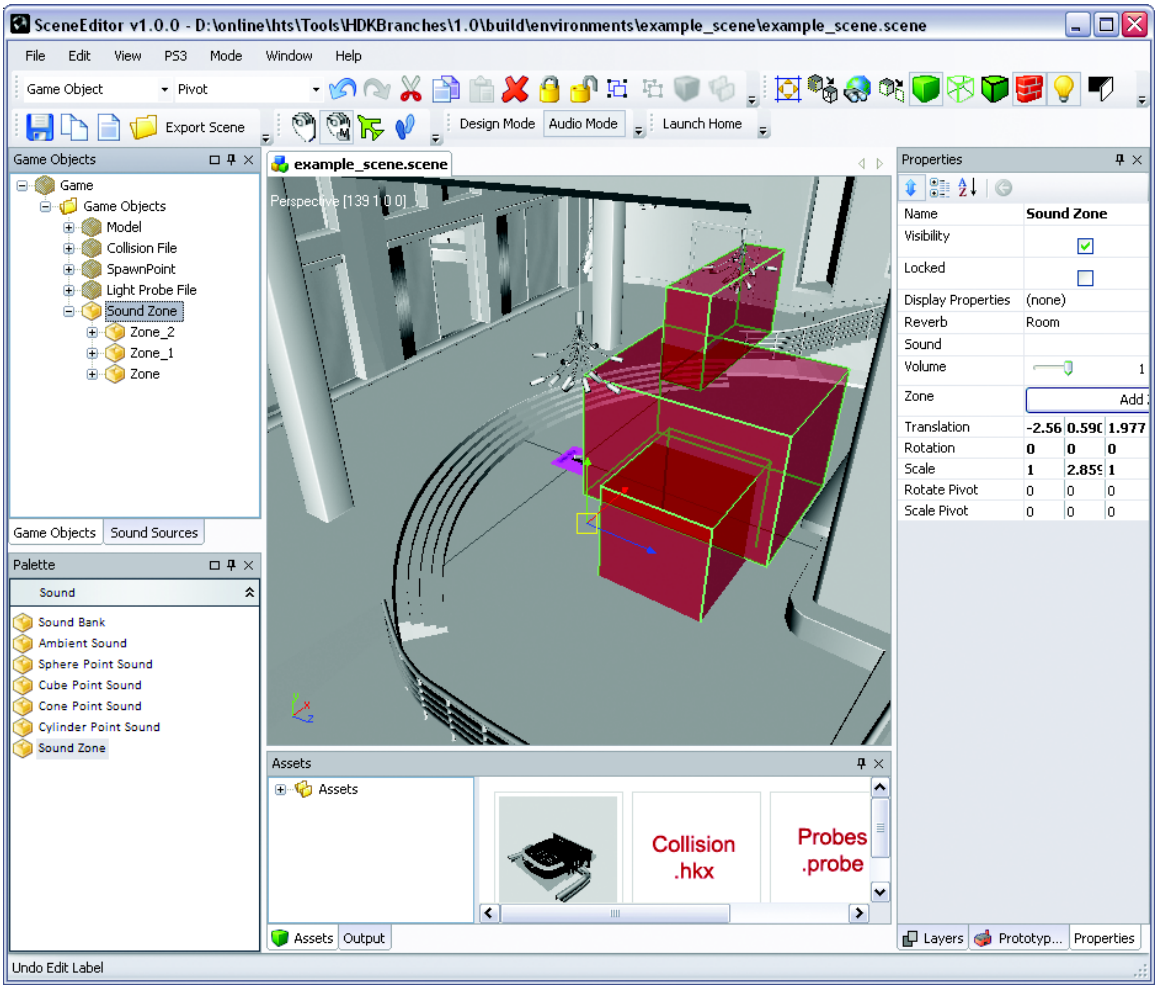
- User's position: The listener is based on the user's position if the user is active (i.e. controlling their avatar) or is they are idle.
- Camera's position: The listener is based on the camera's position if the camera is not focused on the avatar. This happens, for example, when the user is zoomed in on a screen, or when a Lua script takes control of the active camera and moves it away from the user.

Adding a Sound Object to a Scene

To add sound objects to a scene in the Scene Editor:

1. Select Mode > Audio Mode or click Audio Mode on the toolbar. Audio mode displays the audio objects for selection.
2. Select the Point Sound, Ambient Sound, Sound Zone or Sound Portal object you want to add to your scene in the Palette panel.
3. Drag the object from the Palette panel to the Game Objects folder in the Project panel, or directly onto the Design View. The object is added under the Game Objects folder.

You can also preview Point Sounds and Sound Portals in the Design View:

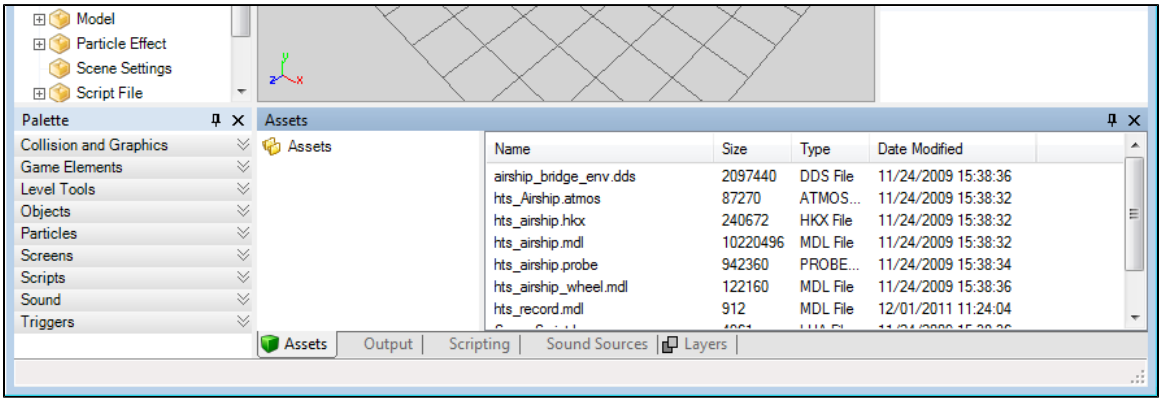


Adding Audio Assets to a Scene

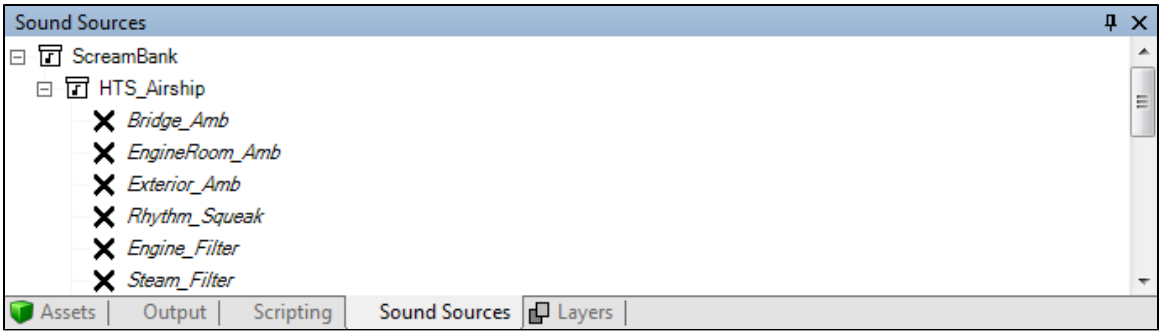
Loading the Sound Bank as an Asset

After creating the sound bank audio files, you can add them as an asset to a scene:

1. Open the scene that you want to add sound to in the Scene Editor and ensure that the Assets panel is displayed:



2. Right-click the Assets folder on the left side of the Assets panel and select Add > Existing Asset...
3. Navigate to the .bnk file and select it, then click Open. The file becomes available for selection in the right side of the Assets panel and the Sound Sources panel contains a list of all sound names within the Sound Bank:



The sound names are listed in italics, meaning that the sound bank has not yet been added to the scene. Exporting the scene will produce an error prompting for the assets to be placed inside the Game Objects folder in the Project panel.

There is a limit of one sound bank per scene.

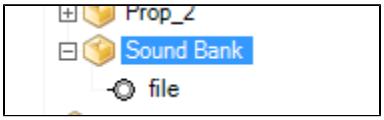
Adding the Sound Bank to the Scene

The asset is now loaded into the Scene Editor. To ensure it is used within the scene, create an object that references this asset:

1. Right click on the Game Objects folder in the Project panel.
2. Select Add > Game Object Folder to create a Sounds folder. This is an optional step that can be skipped, but is recommended.

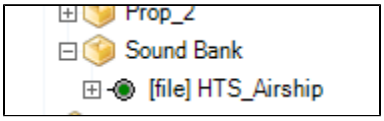
   You can change the folder name after the folder is created.

3. Select a Sound Bank object from the Palette panel and drag it into the newly created Sounds folder in the Project panel:



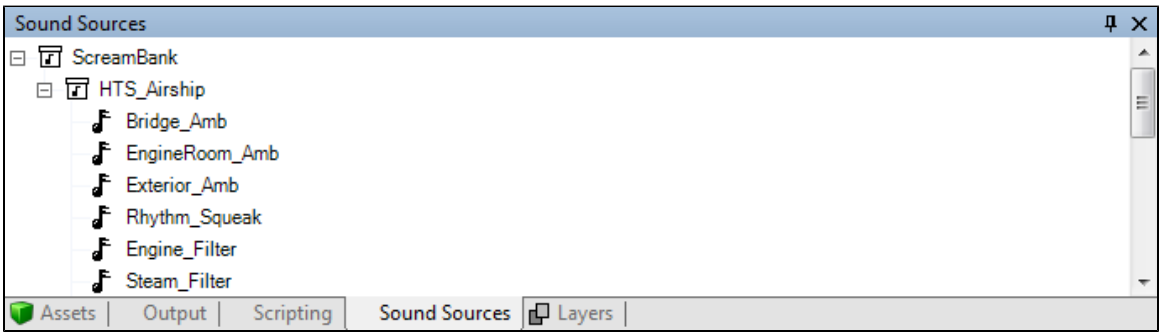
   The Sound Bank contains a grayed-out file, indicating that the object is empty.

4. Drag the sound bank file from the Assets panel to the new Sound Bank object:



   The green icon confirms that the sound bank has been added to the scene.

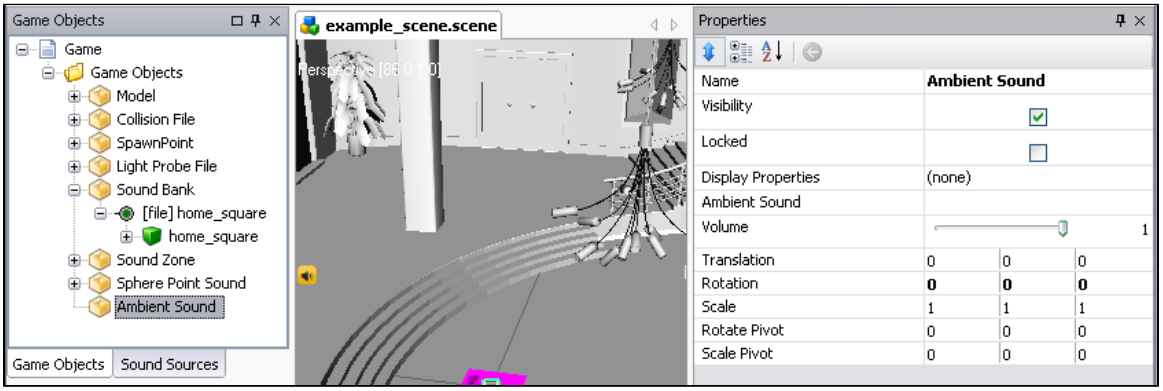
   To confirm if a sound bank has been added to a Sound Bank object, select the Sound Sources panel as shown below. The italicized list is replaced by a standard list:



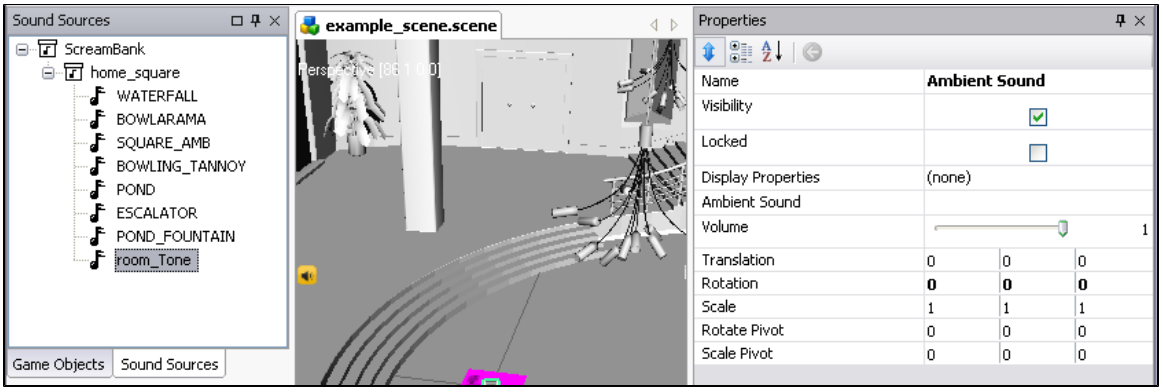
Adding Sound Assets to Sound Objects

After you have added sound objects (Ambient, Points and Zones) to your scene, you can assign sound assets to these sounds from the Sound Bank:

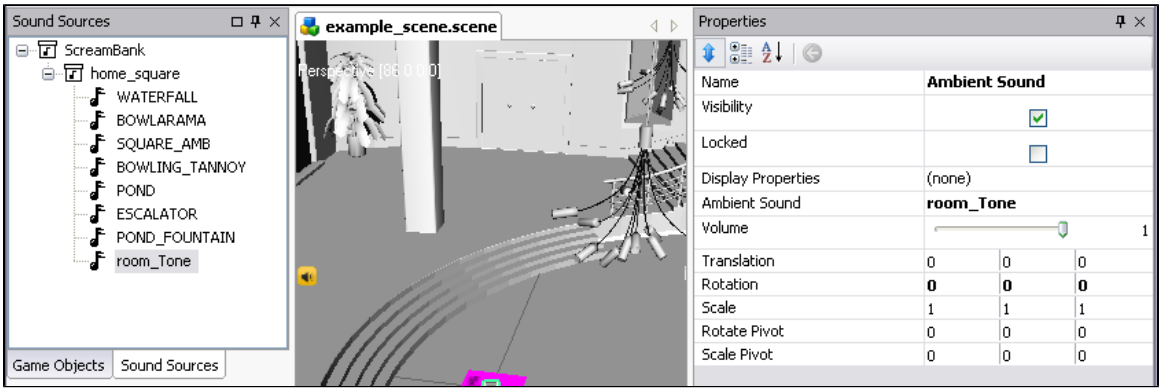
1. Select the sound object (e.g. Ambient Sound) to display its properties:



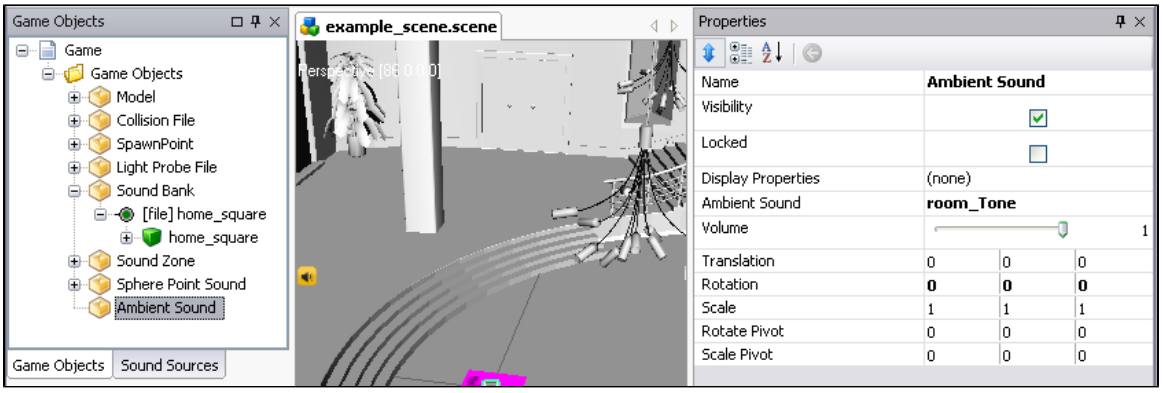
2. Select the sound asset in the Sound Sources panel:



3. Drag the asset to the sound object's Properties panel (e.g. to Ambient Sound in the Properties panel):



4. Now your sound object has an asset associated:



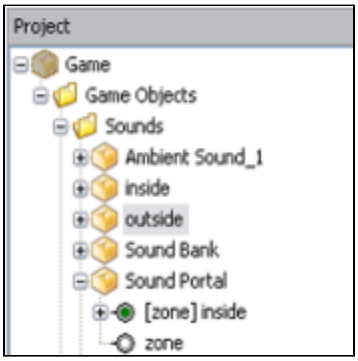
Adding Sound Zones to Sound Portals

After you have added a Sound Portal to your scene, you can assign sound zones from the Project panel:

1. Drag the first Sound Zone object onto the empty zone node:



2. A new empty zone is created under the Sound Portal object. Drag the second Sound Zone object onto the empty zone node:



The Sound Portal now has two Sound Zones linked and blends the sounds from both zones within its sphere.

> ⚠ Sound Zones must be overlapping for the Sound Portal to work.

Adding Sound Streams

You can add streamed audio to an object in the form of an .mp3 file (by adding a soundstream resource). However for any audio to play in a scene, you must add a sound bank as an asset.

> ℹ Streaming .mp3s using Media RSS files is only supported in screens. For more information on streaming Media RSS, see Media RSS File Specification.

Working with Sound Objects

Selecting a Sound Object

Point Sound objects are represented in the Display View of the Scene Editor by a point surrounded by an inner and outer shape. These shapes represent the position of the sound, followed by the fall off areas. For each of the point shapes, the emitter fall off start and end distances can be adjusted. Point Sounds are positioned in Surround Sound 5.1.

> ℹ Cube emitters are currently very expensive to process, and having more than 2 in a scene may result in framerate and audio quality issues.

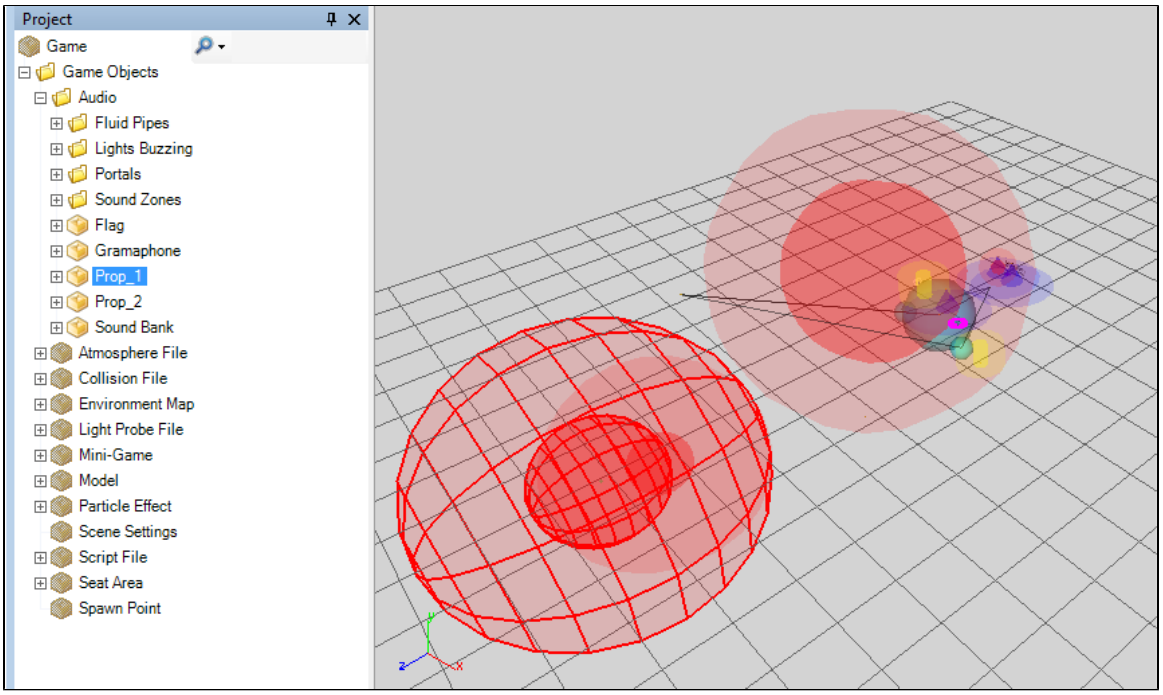
Ambient Sound objects are represented by a single point, with the three manipulation directional compass points visible if you click the Move button  on the toolbar.

Sound Zone objects are represented in the work area by a rectangular area. This shape can then contain a localized ambient sound. The ambient sound can contain reverb effects and exist inside the overall ambient sound.

Sound Portal objects are represented by a point surrounded by a sphere. The sphere defines the limit of the area in which sound is blended, the larger the sphere the longer the transition from one sound to another.

> ⚠ Ensure that the correct Sound Bank sound is associated with the sound object. See Adding Audio Assets to a Scene.

While clicking on the scene in the Display View can tend to highlight more than is desired, simply clicking on the object in the Project panel selects the exact object for modification, for example:



Selecting an Ambient Sound object reveals the location, but without an inner core or outer fall off, as Ambient sound is constant throughout the scene regardless of location within that scene.

Moving the Sound Object

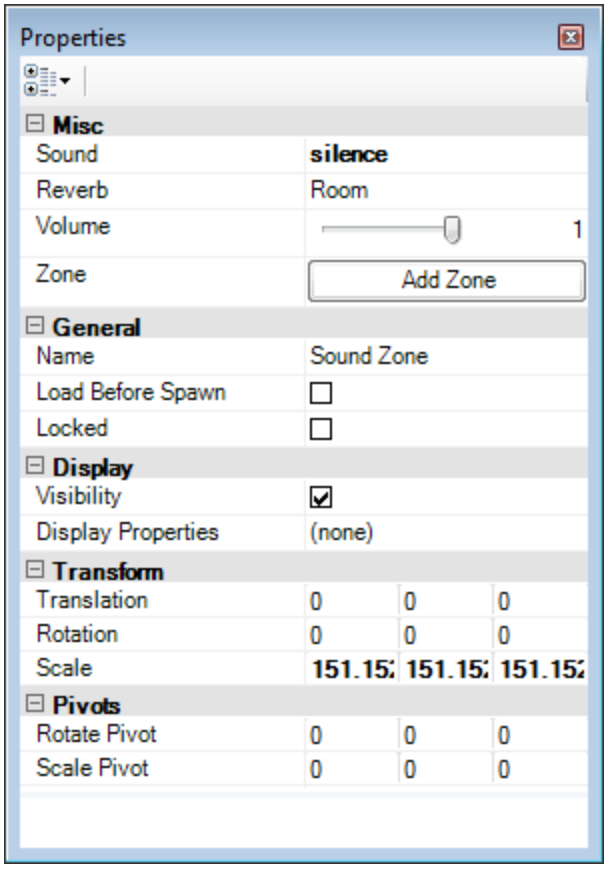
While a sound object is selected, you can use the three manipulation directional compass points, available when you click the Move button  on the toolbar, to drag the object along the axes.

> ℹ The Point Sounds currently offer no occlusion for sound geometry. Therefore, sound is unaffected by location – walls and other objects that may naturally be expected to block or reduce the sound have no effect on the sound.

> ⚠ Sound Portals do not have axis scale manipulators handles. When in Scale Manipulator mode (click the Scale button  on the toolbar), click on the transform handle (yellow square) and drag to resize the object.

Editing a Sound Object's Properties

To edit a sound object in your scene in the Scene Editor, select the object in the Design View or in the Project panel. The properties for the selected sound object are automatically displayed in the Properties panel, for example:



Edit the object's properties in the Properties panel as required. All of the objects available share common properties (those properties grouped under General, Display, Transform and Pivot). These are also sound object-specific properties that depend on the object selected:

- Ambient Sound Properties
- Point Sound Properties
    - Sphere Point Sound Emitter Properties
    - Cube Point Sound Emitter Properties
    - Cone Point Sound Emitter Properties
    - Cylinder Point Sound Emitter Properties
- Sound Zone Properties

Removing a Sound Object from a Scene

To remove a sound object in the Scene Editor, select it in the Project panel and do one of the following:

- Click the Delete button  on the toolbar.
- Select Edit > Delete.
- Right click on the object in the Project panel and select Delete from the menu displayed.

The selected sound object is removed.

> Any asset added to the deleted object remains in the scene unless you remove the asset as well.

Introduction to Screens

Use screens in PS Home to display multimedia content from the internet or from the local file cache to users. Screens are a simple and effective way to deliver engaging content to a scene, adding movement and providing a central point of interest for users. Screens are typically planar rectangular surfaces, although you can also display video content on more complex geometry. This section provides information on the types of media that you can display on a screen and how to add a screen to a scene.

Screen Types

Each screen has a type and one or more 'sources' that provide the link to each piece of media content. The screen type defines what content the screen can display.

> ⚠ All screen content must be approved by your regional SCE.

For information on supported screen and video formats, see Media Formats for PS Home.

Images

The IMAGES screen type can display only images. This screen type can display a single image or a slideshow of images, displaying each image for five seconds with a cross dissolve transition between images. When the slideshow completes, it restarts looping continuously.

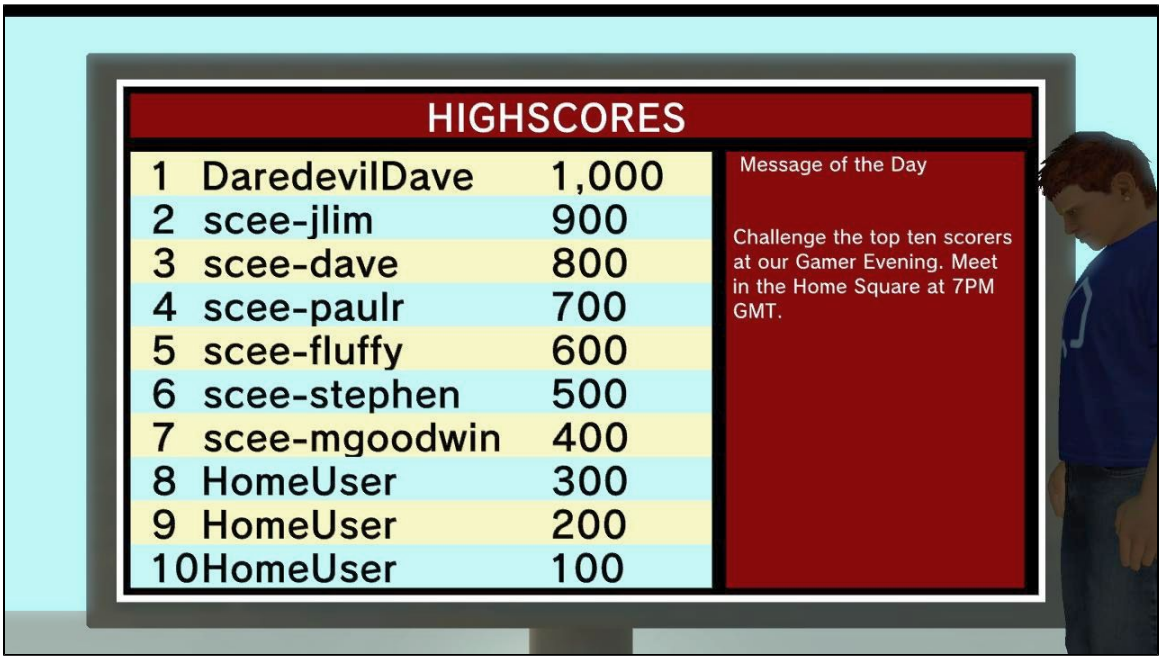
Video

The VIDEO screen type can display only video. As with an IMAGES screen type, it can display a single video, or a sequence of videos. When the sequence of videos completes, it will begin again from the start, looping continuously.

Home Screen Markup Language (HSML)

HSML is an XML format that you can use to arrange images, video, audio, text and colored boxes on a single screen. The HSML screen type can only support one HSML file at a time, but you can be change it at any time using Lua script.

HSML is particularly useful for displaying content such as leaderboards or other web content, because the markup can be dynamically generated by a server and frequently refreshed.



The HSML specification is defined in HSML File Format.

Media RSS

The PS Home client supports a subset of Yahoo's Media RSS specification. You can use Media RSS to create a playlist of images, audio and video. As with the HSML screen type, only one Media RSS file is supported for each screen at any one time, but you can change this at any time using Lua script. Feeds can also be dynamically generated by a server and retrieved by the client.

The Media RSS file specification is detailed in Media RSS File Specification.

For more information about the additional steps you must take to set up content, warnings, and precautions, see Streaming Content on Screens.

Lua

The LUA screen type can display the contents of a Renderer using the Lua function `Renderer.SetTarget()`. So, you can bring Renderer content, such as sprites, text and shapes, into the environment.

For more information on the `Renderer` library, see the Lua API Reference.

Creating Screens

Screens in PS Home can either be flat and rectangular or formed from scene geometry. These screen shapes are known as Planar Screens and Geometry Screens respectively. You can create Planar screens through the Scene Editor or via a Lua function call, whereas you can create Geometry Screens through the Scene Editor only. You can access all screens created in the Scene Editor via the Lua API using the function `Screen.Find()`. You can also attach Planar screens to an Entity to perform dynamic transformations.

> Geometry Screens do not support the HSML and LUA screen types.

Creating Planar Screens

For more information on screen creation using Lua, see the `Screen` library in the Lua API Reference, as well as the Screens sample available from https://home.scedev.net/projects/samples.

To create a Planar screen in the Scene Editor:

1. From the Palette panel, drag a Screen object into your scene.
2. Adjust the screen size using the standard Scene Editor manipulators.

   In addition to the standard Scene Editor manipulators, the following buttons are provided:



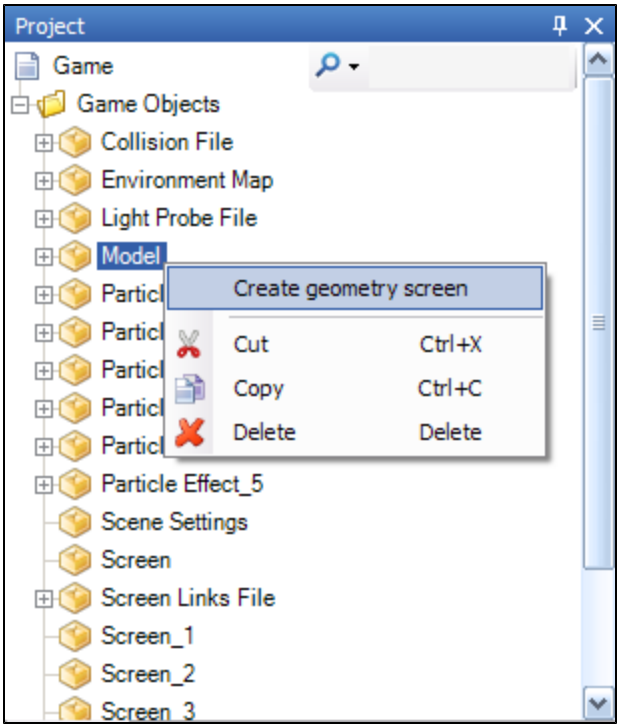
   - Move Single Point: Use this button to make move manipulators appear on each corner of the screen. This allows you to move just one corner of the screen. If one point is moved too much the screen is no longer planar and appears red. Use the Fix Points button to resolve this and make the screen planar again.
   - Move All Points: This is very similar to the  button, except you use it to move the entire screen from one of its four corners. This is useful for snapping your screen against another piece of geometry in the scene.
   - Fix Points: While adjusting the screen points, it may become distorted and no longer be planar. When this occurs the screen turns red and does not function. Use this button to flatten the screen, restoring any corners that are not planar.

Preparing a Geometry Screen in Maya

You can use the Scene Editor to turn any geometry created in Maya with an emissive ATG Material applied to it, into a screen. If the same ATG material is applied to multiple pieces of geometry, they are treated as one large screen. To create separate screens, apply individual ATG Materials to each piece of geometry. When using the same ATG Material on multiple pieces of geometry, combining the geometry into one polyshape may make the geometry's UVs easier to manage. To do this in Maya, select two or more pieces of geometry and select Mesh > Combine. Note that after geometry has been combined, the origin of the geometry is reset to the origin of the scene. This results in any screen audio being emitted from the scene origin. To remedy this, select the combined geometry in Maya and then select Home > Fix Video Screen Audio Meshes from the menu. After the scene is exported, you can create the Geometry Screens required in the Scene Editor.

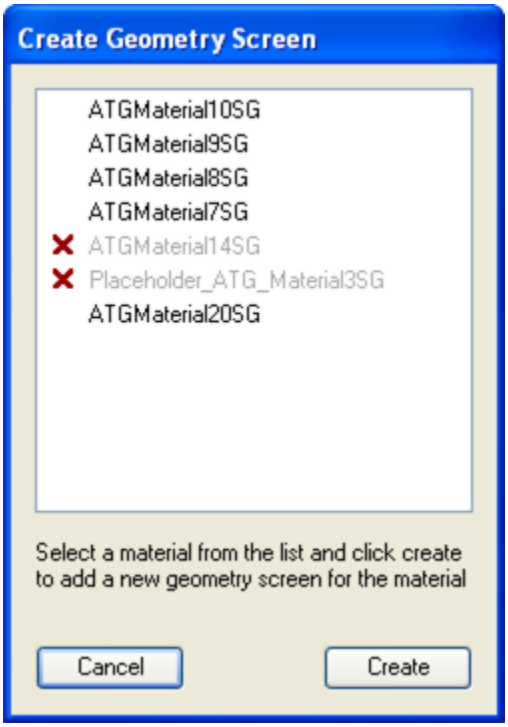
Creating a Geometry Screen in the Scene Editor

1. Open the scene in the Scene Editor.
2. Expand the Project > Game Objects folder.
3. Right-click Model and select Create geometry screen:



The Create Geometry Screen dialog is displayed.

4. Select the emissive shader that you chose for a screen and click Create:

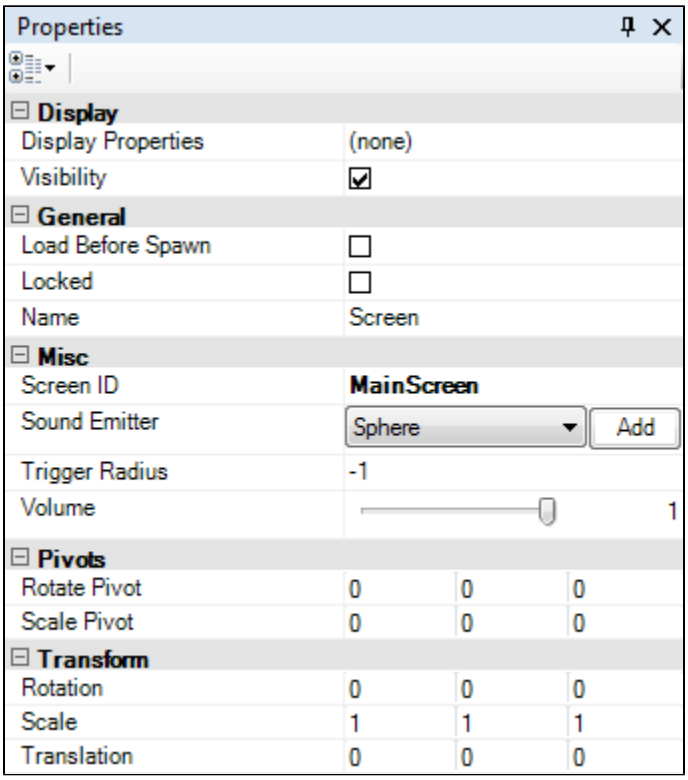


Any ATG Materials that are either invalid or already used for another screen are grayed out.

A new Screen object appears in the Project panel in the Game Objects list.

Configuring a Screen

Screen Properties



When you create a screen in the Scene Editor, a Screen node appears in the Project panel. The screen has the following properties:

Property	Description
Name	A unique name used to represent the screen in the Project panel and in the client. This name is used when accessing screens via Lua script using the function `Scene.FindScreen()`.
Screen ID	An identifier used when assigning screen content.
Sound Emitter	A screen has a spherical sound emitter by default. This can be removed and a number of other emitters (sphere, box, cone, cylinder) can be added. Each emitter is configured independently by expanding the Screen node in the Game Objects panel.
Trigger Radius	Planar Screens only: Controls the size of the trigger area for the screen. The Trigger Radius defaults to 15 in the client if its value is set to less than 1 in the Scene Editor. The Trigger can be disabled using the function `Screen.SetTargetable`. For more information, see Targeting System.
Volume	Controls the audio volume level of the screen. Sound emitters are configured independently by selecting them as sub-nodes of the Screen node in the Game Objects panel.
Fall Off End/Start	These properties control when the sound starts to fade out, and when it becomes inaudible. The values represent the distance in meters from the sound emitter's origin.

Assign Content to a Screen

You assign screen content through a Screen Links file. This is an XML file which defines the screen type and the content to show for each Screen ID used in the scene. The PS Home client reads Screen Links files from multiple sources; they can be embedded in a packaged scene or read from an online (and dynamically updated) source that is managed by Regional SCE. If more than one screen shares a Screen ID, both screens show the same content without using more memory than one screen.

A Screen Links file can link a Screen ID to:

- One or more video files (for screens of type VIDEO)
- One or more images (for screen of type IMAGES)

- A single Media RSS XML file, containing links to one or more pieces of content (for screens of type MEDIARSS)
- A single HSML XML file, containing links to one or more pieces of content (for screens of type HSML)

You can also assign and change screen content using the Lua functions including:

- Renderer.SetTarget()
- Screen.Create()
- Screen.SetContent()
- Screen.SetCurrentContentIndex()
- Screen.SetData()
- Screen.SetVolume()

For more information about these functions and more, see the Lua API Reference.



Posters and screens can also be used as links to scenes, teleporting the user to that scene. This feature cannot be accessed via the HDK only. Contact your Regional SCE to use this feature.

Screen Links XML Schema

The Screen Links file supports the following tags:

- **<SCREEN ID=" ">** – Defines a screen. This is the name used in the Screen ID property for a screen in the Scene Editor.
- **<TYPE>** – Defines the screen type. The possible values are: IMAGES, VIDEO, HSML and MEDIARSS (case sensitive). Screens of type LUA can be created only through Lua script, and not in the Screen Links file.
- **<SOURCE>** – Defines a media source. This can be a URL or a file path within the <HDK_INSTALL_DIR>. A URL may be optionally included which allows the user to launch a web page or enter a space related to the screen content when the screen is targeted.

  For example:

  Web page: `<SOURCE link="http://www.playstation.com">http://www.myserver.com/video.mp4</SOURCE>`

  Space: `<SOURCE link="scene://MySceneName">http://www.myserver.com/video.mp4</SOURCE>`



For the Screen Links file, the <SOURCE> can be a file path within the <HDK_INSTALL_DIR>\build usually only for testing purposes. Local sources must be added as a scene resource in the Scene Editor so that they are included in the scene package.

Speak with your regional SCE if you want to package and use local media files with your scene.

Below is a Screen Links XML that defines five Screen IDs and their content. There are four screens:

- 1 video screen that plays two videos
- 1 image screen that shows three images
- 1 HSML screen
- 1 Media RSS screen
- 1 video screen that plays a local video (primarily for testing purposes)

```
<XML>

	<SCREEN ID="MySceneVideoScreen1">

		<TYPE>VIDEO</TYPE>

		<SOURCE>http://www.myserver.com/video1.mp4</SOURCE>

		<SOURCE>http://www.myserver.com/video2.mp4</SOURCE>

	</SCREEN>

	<SCREEN ID="MySceneImageScreen1">

		<TYPE>IMAGES</TYPE>

		<SOURCE>http://www.myserver.com/image1.dds</SOURCE>

		<SOURCE>http://www.myserver.com/image2.dds</SOURCE>

		<SOURCE>http://www.myserver.com/image3.dds</SOURCE>

	</SCREEN>

	<SCREEN ID="MySceneHsmlScreen1">

		<TYPE>HSML</TYPE>

		<SOURCE>http://www.myserver.com/hsml.xml</SOURCE>

	</SCREEN>

	<SCREEN ID="MySceneMediaRssScreen1">

		<TYPE>HSML</TYPE>

		<SOURCE>http://www.myserver.com/rss.xml</SOURCE>

	</SCREEN>

	<SCREEN ID="LocalVideo">

		<TYPE>VIDEO</TYPE>

		<SOURCE>environments/screensexample/screencontent/video/PSHomeTrailer-P3.mp4</SOURCE>

	</SCREEN>

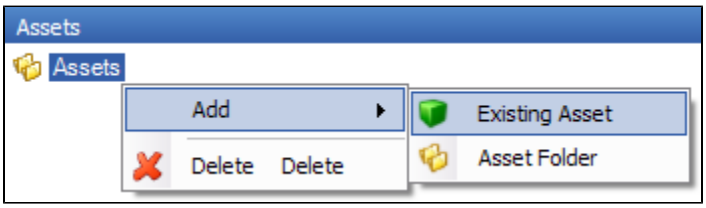
</XML>
```

Save your Screen Links file within the <HDK_INSTALL_DIR>\build folder. It is good practice to save it in the same folder as your scene.

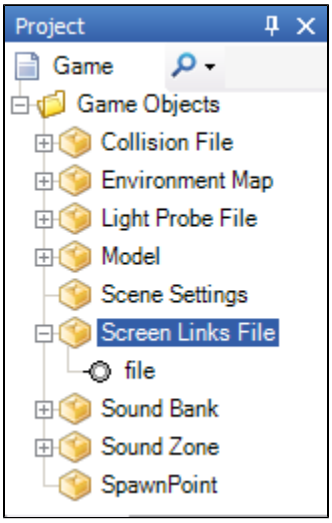
Adding a Screen Links File to a Scene

After you have created your Screen Links file:

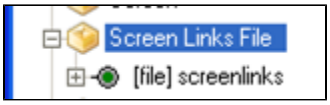
1. Open your scene in the Scene Editor.
2. In the Assets panel, right-click Add > Existing Asset and then select your Screen Links file:



3. From the Palette panel, drag the Screen Links File object to your Project > Game Objects folder.
4. Expand the Screen Links File node to expose a file node:



5. Drag the Screen Links File from the Assets panel to the exposed file node. The node turns green:



Streaming Content on Screens

Screen content is any content that can be used on a screen (images, audio, and video).

> ⚠ All screen content must be approved by your regional SCE.

Screens of type MEDIARSS support the following content playback methods:

- HTTP Progressive Download (cached)
- Non-cached HTTP Streaming
- HTTP Live Streaming

The streaming method required determines the file type which should be linked to from the Media RSS feed.

For more information on the Media RSS file specification, see Media RSS File Specification.

Caching Screen Content

All screen types support content delivery via HTTP Progressive Download, where content is downloaded and stored in the file cache, but begins to play as soon as enough of the file has been retrieved to create a playback buffer.

By default, video delivered via a Media RSS configuration will cache if the file is smaller than 100MB. If required, you can force video that's cached by default to stream. See Forcing Cached Video Content to Stream.

We recommend caching content instead of streaming.

Caching Guidelines

Use the following guidelines when caching screen content:

- Pre-cache only over HTTP, not over HTTPS.
- A cached file cannot exceed 300 MB.
- Cached content should be less than 100 MB each.
- Smaller files do not consume the PS Home HDD cache space as quickly, do not require the user to re-download content, and download faster.
- Enter `filecache list` in the dev debug console to see the total cache size, and usage.
- If pre-caching large files or numerous files: Unless the screen is the focus of the scene (such as a theater or auditorium), consider downloading content only when triggered by the user, or when the user enters a trigger radius.

Streaming Screen Content

You can stream content via Non-cached HTTP Streaming or HTTP Live Streaming. The advantage of using a non-cached delivery method via Non-cached HTTP is that larger files can be used without having to worry about managing or exceeding the cache limit.

Ensure that screens that stream are not active for longer than necessary.

Be aware that content delivered via Media RSS Non-cached HTTP Streaming or HTTP Live Streaming can cause problems in your scene, including:

- Preventing all HTTPS requests
- Constantly consuming user's bandwidth

Each scene can have 2 simultaneous HTTP requests, or 1 HTTPS request (HTTPS is exclusive). Streamed content constantly loops, and can permanently occupy the HTTP/HTTPS channels as they re-download the same video for as long as the user is present in the scene.

HTTP Progressive Download is the preferred method for presenting content, because of the reduced bandwidth usage, and because pre-cache content takes the lowest priority in the HTTP/HTTPS queue.

Streaming Guidelines

Use the following guidelines when streaming content on screens. This applies to all streamed content, that is, content delivered via Media RSS, Non-cached HTTP Streaming, and HTTP Live Streaming.

- Contact your Home Account Manager before using streamed content.
- Only one screen per scene may play streamed content. If you attempt to stream more than one video in a scene, the message "Sorry, we are unable to display video at this time" is displayed on the screen.
- Deliver all screen content over HTTP, not over HTTPS.
- Include a Lua script to interrupt streaming when the client/scene makes an HTTPS request.
- Unless you have a specific need to stream video, use cached video.
- Do not stream content in scenes that also make HTTPS requests.
- Unless the screen is the focus of the scene (e.g. a theater or auditorium), consider streaming content only when triggered by the user, or when the user enters a trigger radius.

Screen Content Playback Methods

HTTP Progressive Download

File Type Required: MP4

Example Media RSS XML:

```
<media:content url=" http://www.myserver.com/video.mp4" medium="video">
      <media:embed url="http://www.playstation.com/">
            <media:param name="deliverymethod">precache</media:param>
      </media:embed>
</media:content>
```

Non-Cached HTTP Streaming

File Type Required: MP4

Example Media RSS XML:

```
<media:content url=" http://www.myserver.com/video.mp4" medium="video">
</media:content>
```

HTTP Live Streaming

File Type Required: M3U8

Example Media RSS XML:

```
<media:content url=" http://www.myserver.com/video.m3u8" medium="video">
</media:content>
```

Creating User-Triggered Media RSS Screens

If you plan to deliver a large amount of content, either through a stream or by chaining cached videos, implement user-triggered screens so that users aren't affected by unseen bandwidth usage. For example, if a user is playing a Mini-game or arcade game in a scene, they're not watching a screen. However, unless scripted otherwise, the screen constantly streams/downloads content, thereby consuming bandwidth for media the user has neither requested nor seen.

HTTP/HTTPS pipelines are free to process other requests that may be more important to what the user is doing.

You can create user-triggered screens on both screens created from Lua and from screens added through the Scene Editor.

1. Only for screens added through the Scene Editor: Do not set your content in the screenlinks.xml file. For example:

```
<SCREEN ID="MediaRSS">
      <TYPE>MEDIARSS</TYPE>
</SCREEN>
```

2. In your script, use `Screen.SetContent` to give the screen a valid .xml file to use when the user is within a trigger radius, or has pressed a button to initiate the screen.

Forcing Cached Video Content to Stream

By default, video delivered via a Media RSS configuration will cache if the file is smaller than 100MB.

To force a video that's cached by default to stream, specify `stream` as the delivery method:

```
<media:embed>

      <media:param name="deliverymethod">stream</media:param>

</media:embed>
```

Troubleshooting Screens

For a summary of the current behaviour of the video screens in a scene, use **enableVideoStats** command in the Dev Debug Console. When the console is active this command activates a display indicated the current number and memory use of videos in the scene, and indicates whether they are playing from cache (C - Video file has been fully downloaded to the HDD cache), progressive (P - Video file is being downloaded to the cache and is playing the in-progress download), http streaming (H - video is streaming over http and not being cached), and http Live streaming (L - video is using a Http Live Streaming system (HLS) through an m3u8 file. It is not being cached).

For more detail on the status of a screen, including extensive details on the playback status of video files (which developers familiar with video codecs will find useful), use the **screenDebug** command in the Dev Debug Console to resolve screen-related issues. The command has two modes:

- **screenDebug all** - Displays general screen information on all screens in the scene, including the screen name, Screen ID, screen type, and file location.
- **screenDebug video** - Displays video-specific information on screens that are playing video, including the video format, frame rate, resolution, and audio information.



To view the total cache size and usage, use the `filecache list` command.

HSML File Format

Home Screen Markup Language (HSML) is an XML format that allows you to arrange text, video, images, audio, and colored boxes on a screen in your scene.

Below is an example of a simple HSML file:

```
<XML>
      <PAGE>
            <RECT X="0" Y="0" W="1280" H="720" col="#C0C0C0" />
            <RECT X="20" Y="20" W="1240" H="60" col="#70000000" />
            <TEXT X="430" Y="20" col="#FFFFFF" size="4">HSML SCREEN</TEXT>
            <RECT X="20" Y="100" W="1240" H="290" col="#70000000" />
            <TEXT X="120" Y="100" col="#FFFFFF" size="4">Image Test:</TEXT>
            <IMG X="140" Y="160" W="385" H="220">http://www.example.url/my_image.dds</IMG>
            <TEXT X="640" Y="100" col="#FFFFFF" size="4">Video Test:</TEXT>
            <VIDEO X="670" Y="160" W="385" H="220">http://www.example.url/my_video.mp4</VIDEO>
            <RECT X="20" Y="410" W="1240" H="290" col="#70000000" />
            <TEXT X="120" Y="440" col="#FFFFFF" size="4">Font Test:</TEXT>
            <TEXT X="140" Y="510" col="#FF0000" size="4">Text: Size 4</TEXT>
            <TEXT X="140" Y="570" col="#00FF00" size="3">Text: Size 3</TEXT>
            <TEXT X="140" Y="610" col="#0000FF" size="2">Text: Size 2</TEXT>
            <TEXT X="140" Y="640" col="#FFFF00" size="1">Text: Size 1</TEXT>
            <TEXT X="640" Y="440" col="#FFFFFF" size="4">Word Wrap Test:</TEXT>
            <TEXT X="670" Y="510" W="500" col="#C0C0C0" size="3">This is a long piece of size 3 text
with a width parameter added to test the wrapping of long text lines within a region.</TEXT>
            <AUDIO>http://www.myserver.com/audio.mp3</AUDIO>
      </PAGE>
</XML>
```

The five page element tags currently supported are:

- <IMG>
- <AUDIO>
- <VIDEO>
- <TEXT>*
- <RECT>*

*This consumes a negligible amount of memory, which is why they are not covered in the Memory Limits section.

The following attributes are supported:

Attribute	Description	Supported By
X	Integer value from 0 to 1280 representing the horizontal position of the item on the screen	IMG, VIDEO, TEXT, RECT
Y	Integer value from 0 to 720 representing the vertical position of the item on the screen	IMG, VIDEO, TEXT, RECT
W	Integer value defining the width of the item on the screen	IMG, VIDEO, TEXT, RECT
H	Integer value defining the height of the item on the screen	IMG, VIDEO, TEXT, RECT
col	Color of the item, specified in an HTML-style format (#RGB), but can optionally have an alpha component (#ARGB), e.g. #FF0000 is opaque red and #8000D000 is semi-transparent green	TEXT, RECT
size	Integer value from 1 to 5 defining the size of the text on the screen (default 3).	TEXT

The origin for HSML coordinates is the top left corner of the screen.

The rendering order of the graphics nodes is the order in which they are listed in the HSML.

⚠ Note that the parsing of HSML adheres to strict XML formatting, specifically:

- Tags and attribute names are case-sensitive.
- All attribute values must be contained with quotation marks.
- All tags must have a corresponding end tag or marker.

Media RSS File Specification

For Japanese Live Page
最新の日本語版ページはこちら 「Media RSSファイルの仕様」

You can use the following Media RSS tags for screen content in PS Home:

- **<rss>** - Signifies an RSS feed.
- **<channel>** - Declares the start of a Media RSS playlist.
- **<item>** - An individual item of media. Each <item> tag can have only one <media:content> element in it. The <media:content> tag supports the following:
    - **url** - The URL of the content.
    - **medium** - The type of content; can be image, video, or audio.
    - **duration** - Used with images to specify for how long the image should appear.

The Home mediaRss spec supports the following media parameters :-

- **priority** - true will use a higher priority download for this content, use with caution.
- **fullscreen** - true will display this screen fullscreen when viewed. HD videos will fullscreen by default.
- **mutescenesounds** - true will disabel ambient audio when viewing this screen.
- **deliverymethod** - specify precache or stream.

Add the following three lines as a sub-element of <media:content> to instruct PS Home to pre-cache video content, instead of streaming it:

```
<media:embed url ="http://www.playstation.com">

      <media:param name = "deliverymethod">precache</media:param>

</media:embed>
```

The URL that is entered within the <media:embed> tag has no effect; it is only there to comply with Media RSS specification, and is ignored by the client. The URL can be any web address, though we recommend using your region's PS Home URL.

MP3 files and files smaller than 100MB are pre-cached by default.

We recommend caching content instead of streaming. However, if you need to force video content to stream, add the following three lines as a sub-element of <media:content>:

```
<media:embed>

      <media:param name="deliverymethod">stream</media:param>

</media:embed>
```

Examples

Pre-Cache Video

The following code creates a Media RSS to play a single pre-cached video:

```
<rss version="2.0" xmlns:media="http://search.yahoo.com/mrss/">
      <channel>
            <item>
                  <media:content url=" http://www.myserver.com/video.mp4" medium="video">
                        <media:embed url="http://www.playstation.com/">
                              <media:param name="deliverymethod">precache</media:param>
                        </media:embed>
                  </media:content>
            </item>
      </channel>
</rss>
```

Image Slideshow

The following code displays an image slideshow:

```
<rss version="2.0" xmlns:media="http://search.yahoo.com/mrss/">
      <channel>
            <item>
                  <media:content url=" http://www.myserver.com/image1.dds" medium="image"
duration="10"/>
            </item>
            <item>
                  <media:content url=" http://www.myserver.com/image2.dds" medium="image"
duration="10"/>
            </item>
            <item>
                  <media:content url=" http://www.myserver.com/image3.dds" medium="image"
duration="10"/>
            </item>
            <item>
                  <media:content url=" http://www.myserver.com/image4.dds" medium="image"
duration="10"/>
            </item>
            <item>
                  <media:content url=" http://www.myserver.com/image5.dds" medium="image"
duration="10"/>
            </item>
      </channel>
</rss>
```

MP3 Audio File

The following code plays an MP3 audio file:

```
<rss version="2.0" xmlns:media="http://search.yahoo.com/mrss/">
      <channel>
            <item>
                  <media:content url="http://www.myserver.com/audio.mp3" medium="audio"/>
            </item>
      </channel>
</rss>
```

Screen Format Quick Reference

Use the following table as a quick reference for screen types and permissible media formats:

Media Type	IMAGES	VIDEO	HSML	MEDIARSS	LUA
DDS	Y	N	Y	Y	Y
JPEG	Y	N	Y	Y	Y
PNG	Y	N	Y	Y	Y
MP3	N	N	Y	Y	Y
AAC (treated as video, MP4 extension)	N	Y	Y	Y	N
MPEG-4 Part 2 Simple Profile	N	Y	Y	Y	N
MPEG-4 Part 2 Advanced Simple Profile	N	Y	Y	Y	N
MPEG-4 Part 10 (AVC/H.264)	N	Y	Y	Y	N
Text and colored boxes	N	N	Y	N	Y*
Renderer	N	N	N	N	Y

*This media is supported via the Renderer using the appropriate Lua library, for example. `Sprite` and `SoundStream`.

Video Recorder

In PlayStation®Home, you can use video recorder functionality in your script (e.g. minigame) that records video and can be shared, played back or uploaded to video sharing websites.

> For information on recording video of your content for your own purposes (e.g. for marketing or debug purposes), see VCR.

The typical uses for the Video Recorder include:

- Recording the moment the player wins a mini-game.
- Tours around a scene.
- Recording video for later editing on a PC.

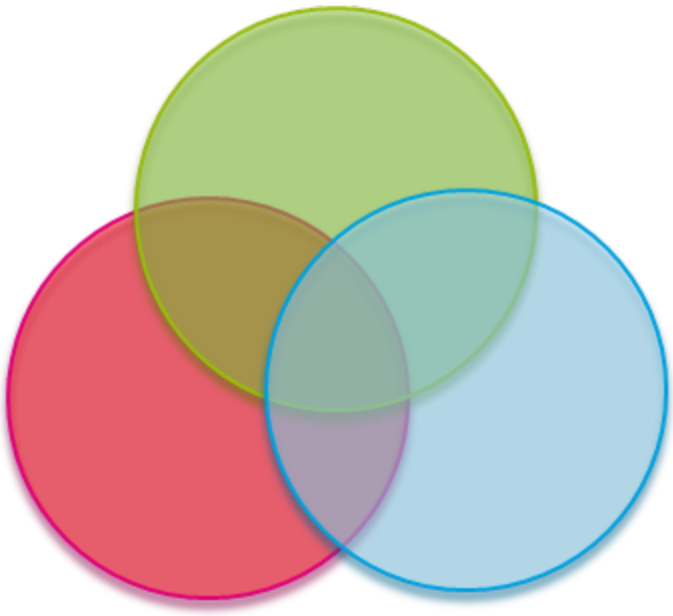
You can use Video Recorder only through Lua script. The Video Recorder can be used only by one script at a time. Make sure that your scripts can handle instances where they are unable to access the Video Recorder.

For the VideoRecorder object, You can specify a video and audio format and quality.

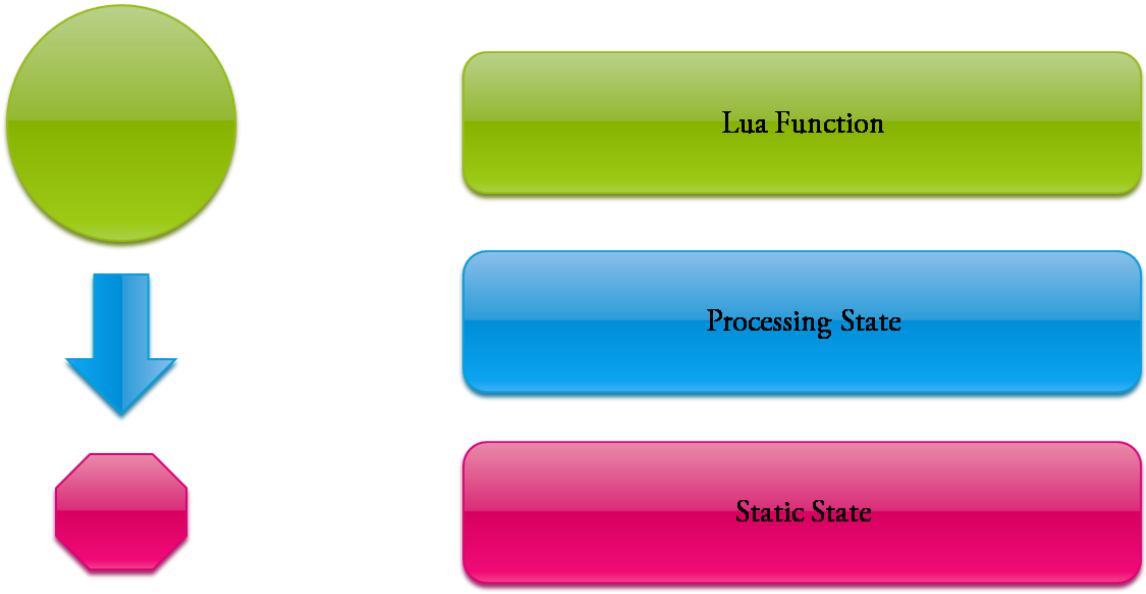
In order to use a library it must be loaded at the top of the script:

```
LoadLibrary( "VideoRecorder" )
```

You will need to do this for any library you use. Some libraries also require you to add a component in the Object Editor, so if you add the library to the script and it gives you the same message then you'll need to add the component.



Workflow



The Video Recorder is built on a state machine where the Video Recorder is always in a particular state, and only certain functions can be used in certain states.

There are two states:

- Processing state, when the Video Recorder is processing a function from the `VideoRecorder` Lua library
- Static state, when the Video Recorder is ready for the next function

You can query the current status of the Recorder at any time, using `VideoRecorder.GetStatus()`.

The Video Recorder workflow is as follows:

1. Check the status of the Video Recorder `VideoRecorder.GetStatus()`



2. If the expected status is returned, call the desired function.



3. The Video Recorder processes the function (processing state).



4. When processed, the Video Recorder goes to a static state.
5. Repeat.

The three main actions for the Video Recorder are: Start, Pause, and Stop recording. Each action requires calling in several functions when the Video Recorder is in a particular state.

To start recording

1. Initialize (prerequisite static state: **Initialized**)

```
VideoRecorder.Initialize()

Initializing

Initialized
```

2. Open (prerequisite static state: **Initialized**, or **Complete**)

```
VideoRecorder.Open()

Opening

Open
```

3. Record (prerequisite static state: **Open**)

```
VideoRecorder.Record()

Starting

Recording
```

To Pause (Optional)

1. Pause (prerequisite static state: **`Recording`**)

```
VideoRecorder.Pause()

Pausing

Paused
```

2. Resume (prerequisite static state: **`Paused`**)

```
VideoRecorder.Resume()

Resuming

Recording
```

To Stop Recording

1. Stop (prerequisite static state: **`Recording`**)

```
VideoRecorder.Stop()

Stopping

Stopped
```

2. Close (prerequisite static state: **`Stopped`**)

```
VideoRecorder.Close()

Closing

Finalizing

Complete
```

3. ShutDown (prerequisite static state: **`Complete`**)

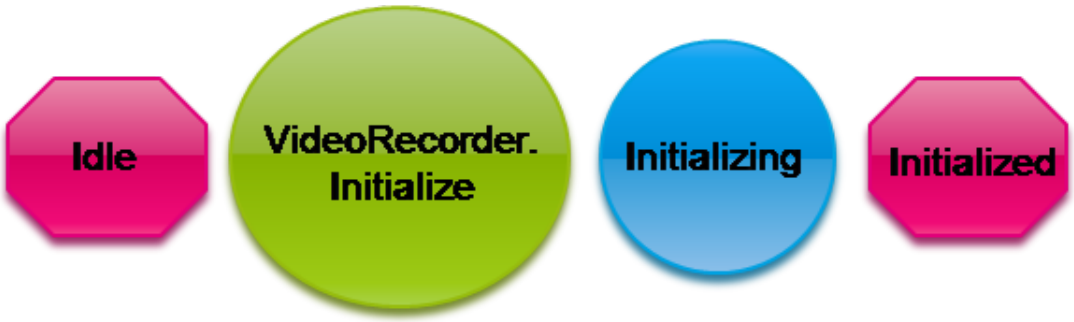
```
VideoRecorder.ShutDown()

ShuttingDown

Idle
```

Example

To initialize the video recorder:




```
-- (Idle) --> VideoRecorder.Initialise --> (Initializing) --> (Initialized)

if VideoRecorder.GetStatus() == VidRecStatus.Idle then


    local success = VideoRecorder.Initialize()



    if not success then

        print(VideoRecorder.GetErrorString())


    end

    while VideoRecorder.GetStatus() == VidRecStatus.Initializing do


        Sleep(0)


    end

    if not VideoRecorder.GetStatus() == VidRecStatus.Initialized then


        print(VideoRecorder.GetErrorString())


    end


end
```

Exporting video to the XMB™

A video is stored on the PlayStation®3 HDD while recording. This storage is temporary and the video is deleted when you record a new video, or the player leaves the scene. To keep a copy of the video, you must export it to the XMB™.

To export a video to the XMB™:

1. Retrieve a handle to the last recorded video using `VideoRecorder.GetOutputFile()`. If the last recording resulted in an error, the returned handle will be nil.
2. Export the video using `MediaLibrary.ExportVideo()`. Use the video's handle and specify the caption that will show on the XMB™.
3. If `MediaLibrary.ExportVideo()` returns False, the export was not successful. Otherwise, the video should have successfully exported after `MediaLibrary.IsExporting()` no longer returns true.

You can view exported videos on the Videos tab:

- Either press the PS button on the controller when the Video Recorder is in the Idle static state. This method does not work if PS Home was launched from the SELF.
- Or launch into the XMB™ using Target Manager.

Upload Videos to the Web

In addition to saving onto the XMB™, recorded videos, you can upload recorded video to video websites using the function System.UploadVideo(). The Video does not need to be saved to the HDD in order to upload to a website. PS Home cannot upload videos from the user's HDD to websites.

PS Home currently only supports uploading to YouTube.

To upload a video to YouTube, you must register as an API Developer.

Registering an API Developer Account

To register an API Developer account with Google and receive a developer key:

1. If a primary Google account does not exist for the product or company, create one using the following URL: https://www.google.com/accounts/NewAccount
2. Log on to Google using the account above, and access the YouTube Developer Dashboard using http://code.google.com/apis/youtube/dashboard.
3. If the account has not previously been registered with Developer access, fill in the necessary information and click Register.
4. Click on New Product, fill in the details for the product and save.
   The Developer Key is generated.

Notifying Google

When you receive the developer key, notify Google to ensure better tracking and support. Send an email using the following format:

```
========================================
To: ytapi-support@google.com
Cc: youtube-api-support-ps3@google.com
Subject: YouTube Upload API by PS3 SDK
========================================
Game publisher's Google Account:
Registered client id:
Registered developer key:
========================================
```

> ⚠ The client ID cannot have spaces, i.e. you can use 'ThisID' but not 'This ID'.

Uploading to YouTube

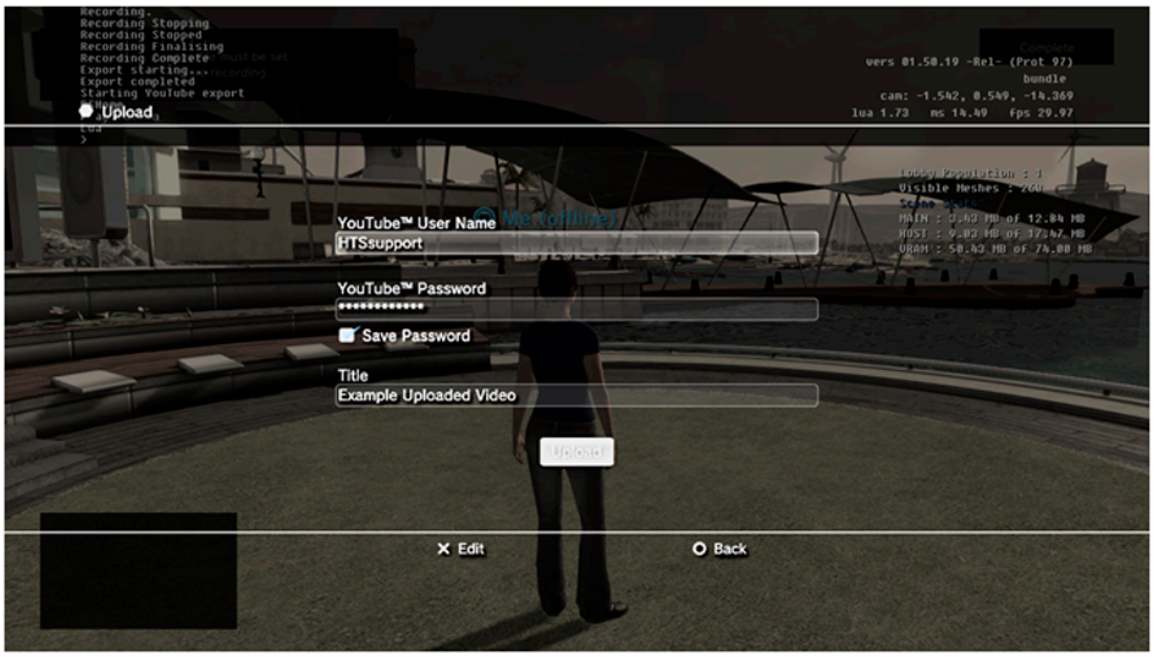
The Home client uses the YouTube API for uploading to the website. Developers are recommended to consult the YouTube Data API documentation at http://code.google.com/apis/youtube/2.0/developers_guide_protocol.html+ for information specific to YouTube.

When uploading to YouTube, System.UploadVideo() requires the following syntax (Required arguments are **videoHandle, VideoSite.YouTube, clientId, developerKey**):

```
System.UploadVideo(videoHandle, VideoSite.YouTube, clientId, developerKey, title, description,
keyword1, isPrivate, ageRating, keyword2, keyword3)
```

See the Lua API Reference for an explanation of each argument.

If a call to System.UploadVideo was successful, the XMB will open a prompt where the user can specify a YouTube username and password to upload the video to. The upload process will use the XMB interface from this point onwards, until the video upload is successful or has been canceled by the user.

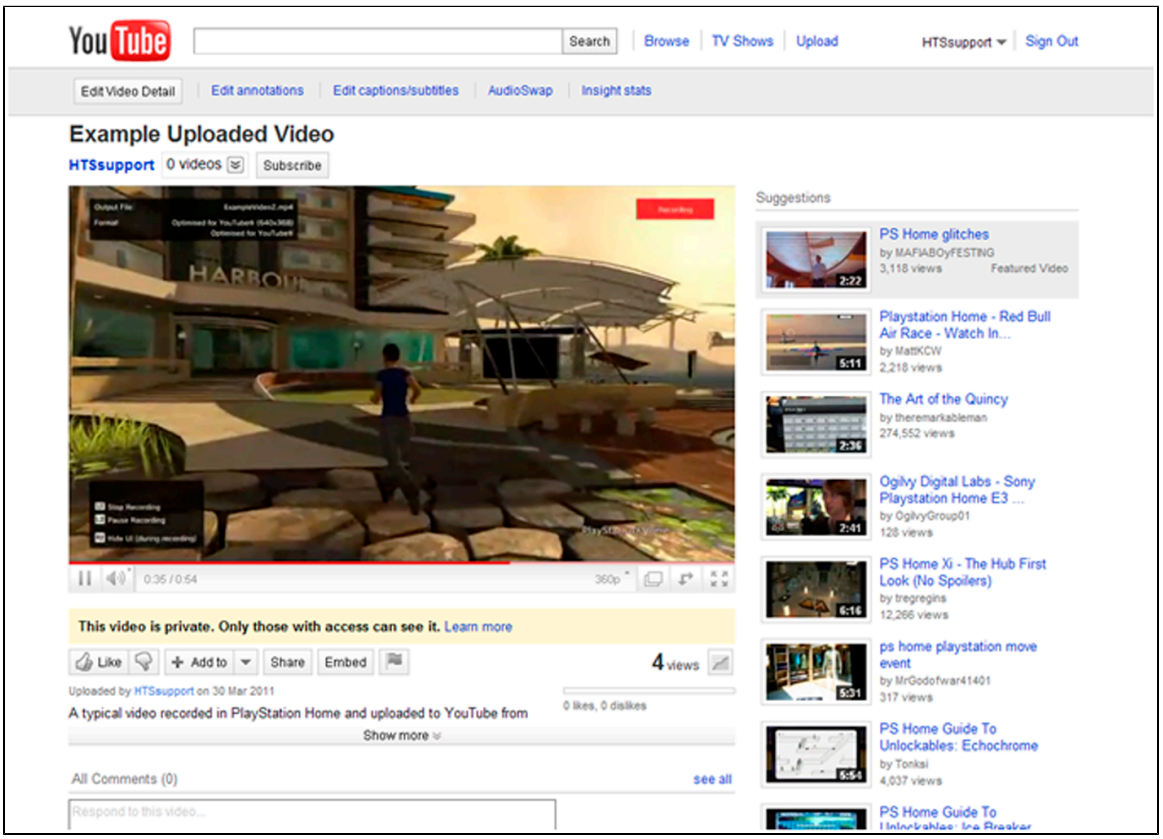


Users must specify their own login information when they are uploading a video. Home content cannot use its own account to upload videos.

Previewing The Video

After you have uploaded the video to YouTube, to view the video, you can either:

- Go back to the XMB™, then use the YouTube XL application to view the video (public videos only).
- Log on to the YouTube account for the video via a PC (all videos).



Guidelines

- Always check the status of the Video Recorder.
- Home content cannot be automated to use a fixed login to video websites. Users must enter their login details each time they want to upload a video.
- Never assume that scripts can always upload videos. Call `System.GetVideoUploadStatus()` to check that the client is idle and can upload a video.
- Due to issues with memory and compatibility with Adobe Flash Player, we do NOT recommend you use the in-game web browser to view uploaded videos from PS Home.
- Call `System.GetVideoUploadResult` to determine if a Video uploaded successfully.
- Call `System.GetVideoUploadUrl` to retrieve the uploaded video's URL.
- PS Home and Home content cannot upload videos from the user's HDD to websites.
- Videos can only be uploaded to YouTube.
- The client ID cannot have spaces. For example, 'ThisID' is allowed. 'This ID' is not allowed.

YouTube-Specific Guidelines

- A video cannot be set to Unlisted from PS Home. To set a video as such, upload the video as Private, and change it to Unlisted on the YouTube website.
- You cannot charge users fees for uploads to YouTube by, for example, selling tickets. This is stipulated in the YouTube Terms of Use.
- The YouTube XL player on the XMB™ cannot play private videos.
- We recommend you log onto the YouTube website via a PC to view videos. In addition to viewing private and unlisted videos, the videos can be edited with additional information (e.g. longer descriptions, more keywords).

Video Recorder Guidelines and Tips

Guidelines

- Only one Lua environment can use Video Recorder at any one time (per local client). Always make your script check the status of the Video Recorder before attempting to use it.
- The XMB, video playback, and OSK are unavailable during any Video Recorder state except `Idle`.
- Each Video Recorder function can only be used when the Video Recorder is in a particular state. If a function is called during the wrong state, it will return an error.

Tips

- Each VideoRecorder function has a prerequisite Static state. For example, before you can Initialize the Video Recorder (`VideoRecorder.Initialize()`), the static state must be `Idle`.
- Use `VideoRecorder.GetStatus()` to retrieve the state of the Video Recorder.
- If `VideoRecorder.GetStatus()` returns VidRecStatus.Error, call `VideoRecorder. GetErrorString()` to retrieve the error string.
- Use `ShutDown` to return the Video Recorder to its state to `Idle` if not actively using the Video Recorder.
- When you Shutdown the Video Recorder, you must export the video to the XMB™, or upload it to YouTube, otherwise the video is lost.
- (During testing) Resetting an object that uses Video Recorder (such as Pressing F2) may stop the Video Recorder from working. This is because the first instance of the object is still using the Video Recorder. To fix this issue, restart the client.

To change a screen's protected property, call `Screen.SetProtected()()` in your script. You can call this function for both Lua screens, and screens added to a scene through the Scene Editor.

Video Recorder Reference

Static States

State	Description	Allows
Idle	The Video Recorder is not initialised Restricted functionality can be used. Default state of Video Recorder. Also caused by `VideoRecorder.ShutDown()`.	`VideoRecorder.Initialize()`
Initialized	The Video Recorder has been initialised but a video has not yet been opened. Caused by `VideoRecorder.Initialize()`	`VideoRecorder.Open()`
Open	A video file has been created but recording has not started. Caused by `VideoRecorder.Open()`	`VideoRecorder.Start()`

Recording	The player's screen is being recorded to the prepared video file. Caused by VideoRecorder.Start(). Also caused by VideoRecorder.Resume() if paused.	VideoRecorder.Pause(), VideoRecorder.Stop()
Paused	Recording has temporarily stopped, but can be resumed. Caused by VideoRecorder.Pause()	VideoRecorder.Resume()
Stopped	Recording for the active video has finished, and must be finalised. No more recording can occur for the active video. Caused by VideoRecorder.Stop()	VideoRecorder.Close()
Complete	The active video has been completed, and a new video can be opened. Caused by VideoRecorder.Close() If you call VideoRecorder.Open() during this state, using a different filename you can create another video. If you use the same name you will replace the existing video.	VideoRecorder.ShutDown() VideoRecorder.Open()

Processing States

The following states are used when the Video Recorder is currently performing an action. While the Video Recorder reports one of the following states, Lua script should wait before attempting to call another Video Recorder function.

State	Description	Leads to
Opening	The Video Recorder is currently opening a video file. Caused by VideoRecorder.Open()	Open
Starting	The Video Recorder is starting to record into the active video. Caused by VideoRecorder.Start()	Recording
Pausing	The Video Recorder is attempting to pause recording, and will continue recording until it reaches the Paused static state. Caused by VideoRecorder.Pause()	Paused
Resuming	The Video Recorder is attempting to resume recording, and will not do so until complete. Caused by VideoRecorder.Resume()	Recording
Stopping	The Video Recorder is attempting to stop recording. Recording will still continue until it reaches the Stoped static state. Caused by VideoRecorder.Stop()	Stopped
Closing and Finalizing	The Video Recorder is attemption to close the Video Recorder session and finalize the video. Caused by VideoRecorder.Close()	Complete
ShuttingDown	The Video Recorder is ending the session and returning to an idle state. Once Idle, any other Lua environment can use the Video Recorder. Caused by VideoRecorder.ShutDown()	Idle

VideoRecorder Functions

Function	Description	Prerequisite Static State	Starts Processing State	Leads to Static State
Initialize	Initializes the Video Recorder	Idle	Initializing	Initilized
Open	Creates a new Video Recording, but does not start recording.	Initialized, Complete	Opening	Opened
Start	Starts recording a video.	Opened	Starting	Recording
Pause	Pauses recording	Recording	Pausing	Paused
Resume	Resumes recording.	Paused	Resuming	Recording
Stop	Stops recording.	Recording	Stopping	Stopped
Close	Closes the Video Recorder, and prepares the Video file	Stopped	Closing	Complete

ShutDown	Shuts the Video Recorder down, returns it to an idle state and allows other Lua environments in the scene to access it.	Complete	ShuttingDown	Idle
GetStatus	Returns the status of the Video Recorder.	–	–	–
GetErrorString	If any of the functions above return false or VidRecStatus.Error this retrieves the error string.	–	–	–
GetErrorCode	If any of the functions above (except GetErrorString) return false or VidRecStatus.Error this retrieves the error code.	–	–	–
Terminate	Terminates the Video Recording immediately. Deletes the video from the system cache. Used when the script is forced to exit while video recording in progress.	–	–	–

Supported Video Formats

The Video Recorder supports a number of video and audio formats that for encoding saved videos. The time required to finalize a video and the size of the resulting video increase as the resolution and/or bit rate increases.

Included are a set of formats optimized for YouTube playback. Videos intended for viewing on the YouTube service are recommended to use these formats.

More information concerning media formats is available in Media Formats for PS Home.

VidRecVideoFormat Value	Format	Resolution (Normal)	Resolution (Widescreen)	Bit Rate (kbps)
Mp4Small512k	MPEG-4 Part 2, Simple Profile	320px x 240px	368px x 208px	512
Mp4Small768k	MPEG-4 Part 2, Simple Profile	320px x 240px	368px x 208px	768
Mp4Med512k	MPEG-4 Part 2, Simple Profile	368px x 272px	480px x 272px	512
Mp4Med768k	MPEG-4 Part 2, Simple Profile	368px x 272px	480px x 272px	768
Mp4Large512k	MPEG-4 Part 2, Simple Profile	480px x 368px	640px x 368px	512
Mp4Large768k	MPEG-4 Part 2, Simple Profile	480px x 368px	640px x 368px	768
Mp4Large1024k	MPEG-4 Part 2, Simple Profile	480px x 368px	640px x 368px	1024
Mp4Large1536k	MPEG-4 Part 2, Simple Profile	480px x 368px	640px x 368px	1536
Mp4Large2048k	MPEG-4 Part 2, Simple Profile	480px x 368px	640px x 368px	2048
AvcmpSmall512k	MPEG-4 Part 10 (AVC/H.264), Main Profile	320px x 240px	368px x 208px	512
AvcmpSmall768k	MPEG-4 Part 10 (AVC/H.264), Main Profile	320px x 240px	368px x 208px	768
AvcmpMed512k	MPEG-4 Part 10 (AVC/H.264), Main Profile	368px x 272px	480px x 272px	512
AvcbpMed768k	MPEG-4 Part 10 (AVC/H.264), Baseline Profile	368px x 272px	480px x 272px	768
AvcbpMed1024k	MPEG-4 Part 10 (AVC/H.264), Baseline Profile	368px x 272px	480px x 272px	1024
AvcbpMed1536k	MPEG-4 Part 10 (AVC/H.264), Baseline Profile	368px x 272px	480px x 272px	1536
YouTube	MPEG-4 Part 2, Simple Profile (Optimized for YouTube®, 240p)	320px x 240px	320px x 240px	768
YouTubeLarge	MPEG-4 Part 2, Simple Profile (Optimized for YouTube®, 360p)	480px x 368px	640px x 368px	2048

Supported Audio Formats

VidRecAudioFormat Value	Format	Bit Rate (kbps)
Aac64k	MPEG-4 AAC	64
Aac96k	MPEG-4 AAC	96
Aac128k	MPEG-4 AAC	128
YouTube	MPEG-4 AAC (Optimized for YouTube®)	64

VCR

For Japanese Live Page
最新の日本語版ページはこちら 「VCR - JP」

VCR commands can be used to record video of your content for your own purposes (e.g. for marketing or debugging).

VCR commands let you record video and save it to the PS3 HDD.

You can specify a filename and path for your VCR recording.

It is not possible to specify the format for the VCR and the console commands can't be used to specify quality.

The VCR format is MPEG4, at 30fps, and the video resolution is 640x368 (16:9).

If you wanted more control over the quality of the recording, you will have to use Video Recorder in a script in your content.

For information on video recorder functionality in your script (e.g. minigame), see Video Recorder.

Video Recorder captures video that can be shared, played back or uploaded to video sharing websites. The typical uses for the Video Recorder include:

- Recording the moment the player wins a mini-game.
- Tours around a scene.
- Recording video for later editing on a PC.

Using VCR Console Commands

There are several options

1. Use the console commands by pressing select and typing them into the HUD.
2. If you have a devkit with switches on the front, push up switch 0. This will make select toggle the console directly and not invoke the Safe Screen.
3. Add `setDipSwitch 0 1` to your `hubstartup.txt` file. This will have the same effect as above.
4. In Target Manager, go to your kit, then Console Output. Right-click on the `USER 9` tab, Properties, and check `Echo input to screen`. You can now type console commands into channel 9, which will send them to the client and run them in the console. So you can enter the VCR commands into Target Manager without having the console open in the client. Be sure to swap to channel 6 after each command to make sure the VCR is reporting the correct state.

Make HUD invisible by pressing SELECT.
The video will keep recording (you can easily verify this because the Home watermark should remain visible, even when the HUD is hidden, while video records.

Note that the Safe Screen cancels recording.

List of Commands

To see a list of available commands, type 'vcr' into the debug console and hit TAB:

```
VCR.Close
VCR.Deinit
VCR.GetOutputFilePath
VCR.GetStatus
VCR.Init
VCR.Open
VCR.Pause
VCR.Start
VCR.Stop
```

To see which arguments the command accepts, type the command and then hit ENTER.

Expected use:

1. `VCR.Init` - Initializes the command

2. VCR.Open <Filename> - This specifies the filename to use for the resulting video
3. VCR.Start - Video recording starts
4. VCR.Stop - Video recording stops
5. VCR.Close - This finalizes the file
6. VCR.Deinit - Deinitializes the command

Other commands are also available:

```
VCR.GetStatus
VCR.Pause
VCR.GetOutputFilePath
```

You can find the output file by navigating to dev_hdd1 in Target Manager, in the File Sync menu under your target. It may be called dev_hdd0 rather than 1.

Art Tools

> These pages are specifically about creating content with the Art Tools. For general information on creating content such as environments, character components or furniture components, see the Objects and Scenes sections

Collapse all
Expand all Collapse all

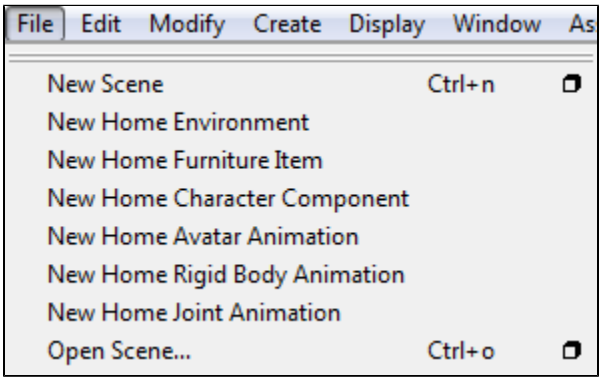
Maya and the HDK

The HDK uses Maya for modeling and animations, and there are a number of tools that integrate with Maya.

To use the HDK in Maya, you must launch the program through the Home Development Kit menu. You cannot use the HDK by launching the Maya.exe directly.

To use Maya:

1. Install the HDK.
2. Launch Maya by selecting Start > All Programs > Home Development Kit <version number> > Launch Home Maya <version number>:
3. When Maya is loaded, select a template by selecting File > New Home Environment:



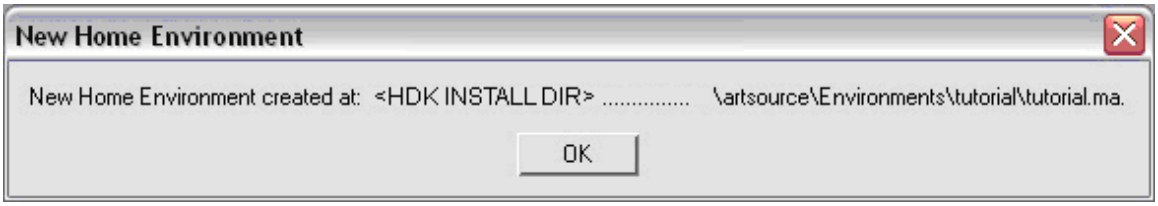
> To open saved scenes created with the HDK, select File > Open Scene.

4. Enter the name of the new environment and click OK:



The scene is saved in a subdirectory of the Artsource, for example, `Artsource\Environments` (for creating environments), `Artsource\Furniture` (for creating Furniture) and `Artsource\Clothing` (for creating Clothing), where the sub-directory name is the same as the filename, for example, `Artsource\Environments\mymayascene\mymayascene.ma`.

5. Click OK:



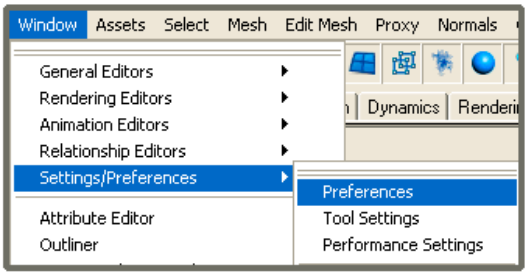
If a Maya scene with the same name exists, you are prompted to enter another name.

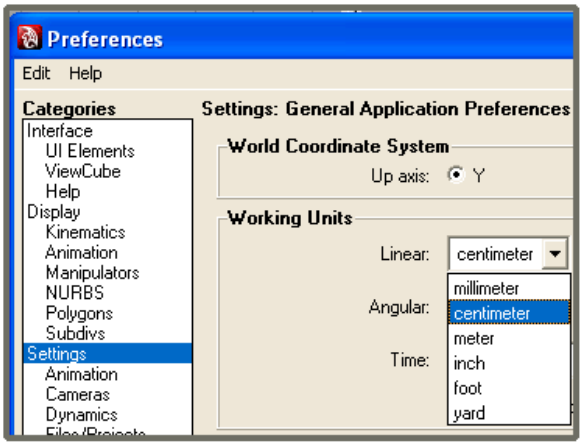
Changing Default Units in Maya

The HDK works in meters, but the HDK Maya toolset defaults to units in centimeters (cm) not meters (m). You can change this manually to suit your particular tasks, for instance when modeling buildings.

To change the default units, select:

Windows > Settings/Preferences > Preferences > **Settings** > Working Units > Linear



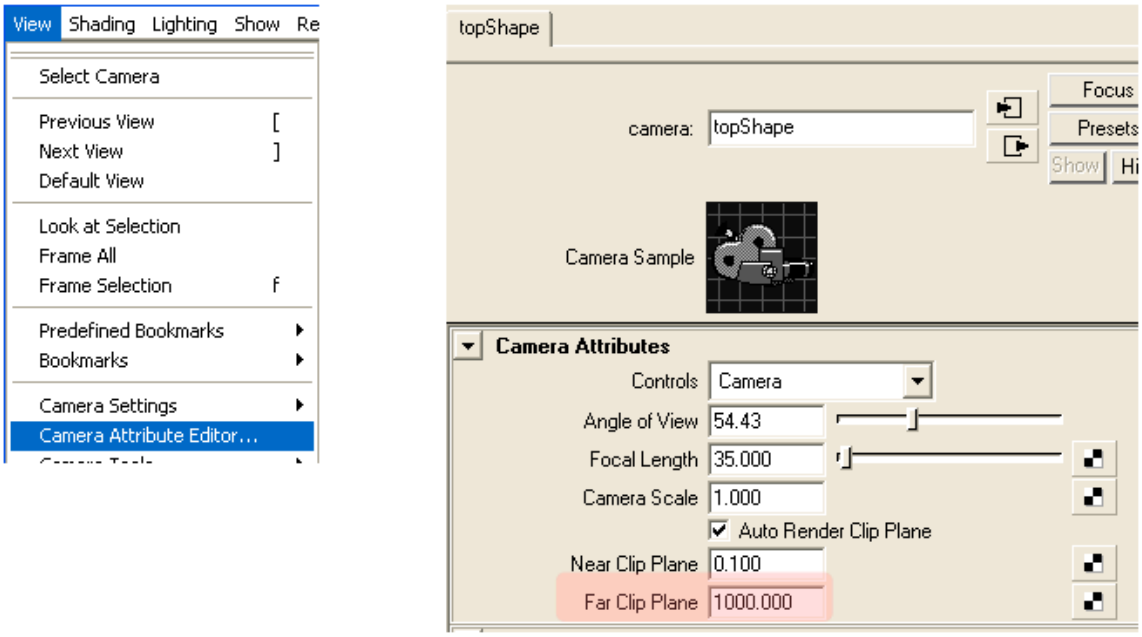


Adjusting Your Camera Clip Plane

When you change your units, your camera clip plane may need adjusting as well.

To set this in the viewport, select:

View > Camera Attribute Editor > Far Clip Plane



Maya Workflows

This section outlines workflows and best practice for using the HDK and Maya.

- General Maya Operations

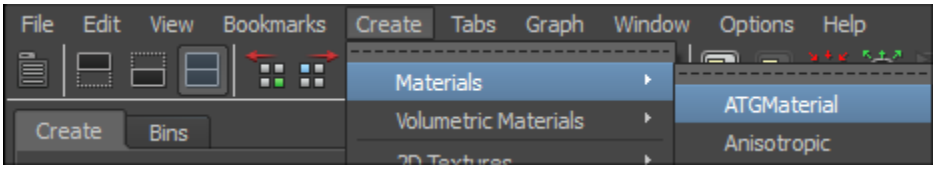
These sections outline ways to optimize your models and environments.

- Smooth Edge Optimization
- Environment Optimization

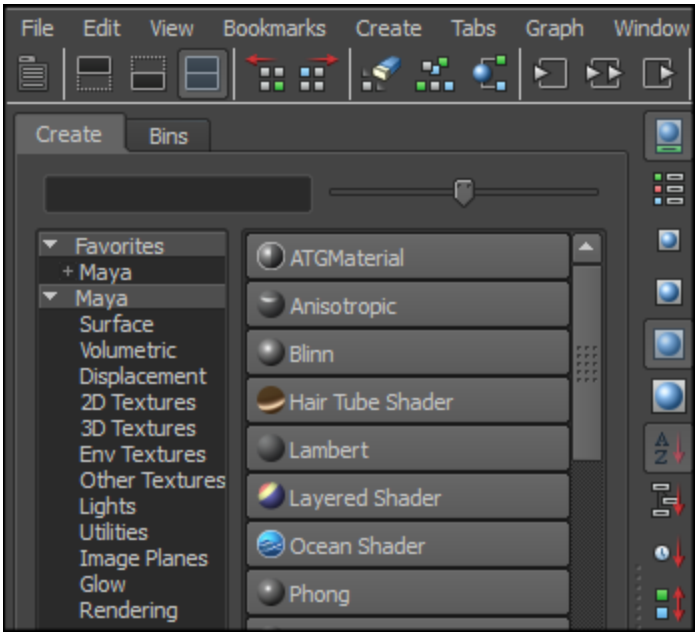
General Maya Operations

Creating an ATG material

ATG materials are creating via Maya's Hypershade.



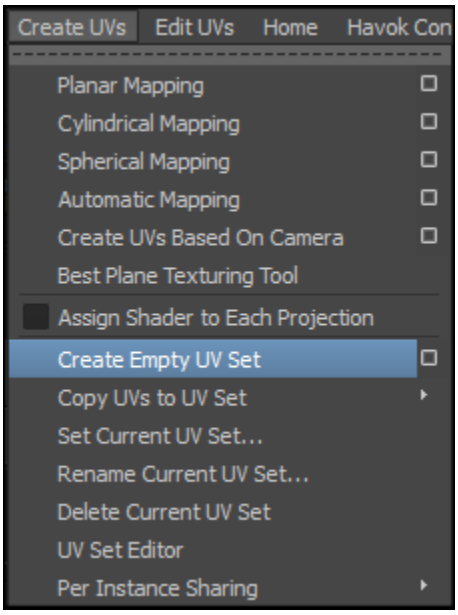
The material can be created from the Hypershade menu above or the Create panel below.



More information on the various shader types can be found in the Shader Reference.

Create UV set 2

Uv sets are created in the Maya menu

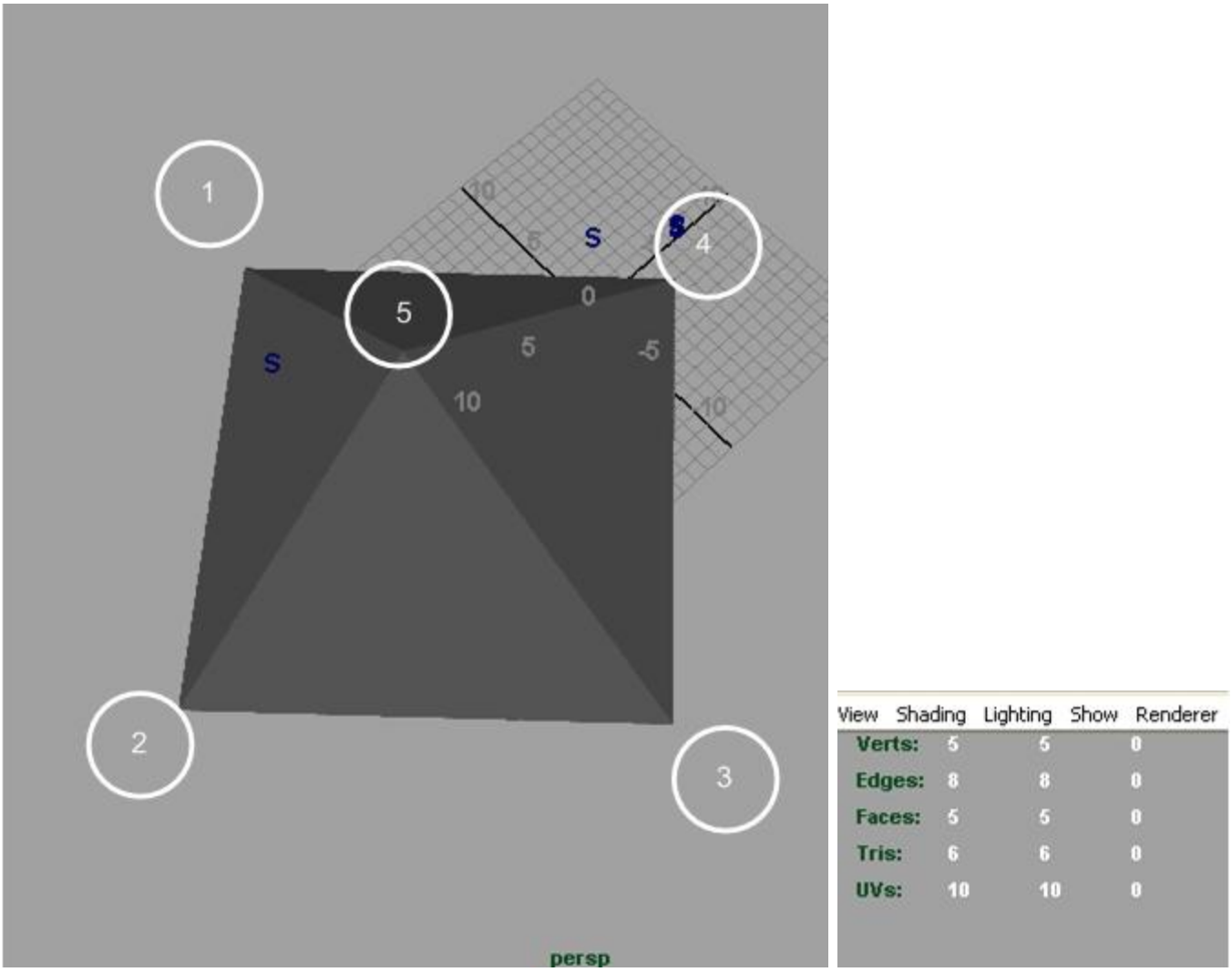


The set must be called map2

Only certain meshes will need a 2nd UV set. For more information on UV set 2 see Lightmap UV set 2

Smooth Edge Optimization

If your model has many hard edges that cause it to exceed the memory limits during export, try softening or smoothing the vertices, and applying a normal map. For example:



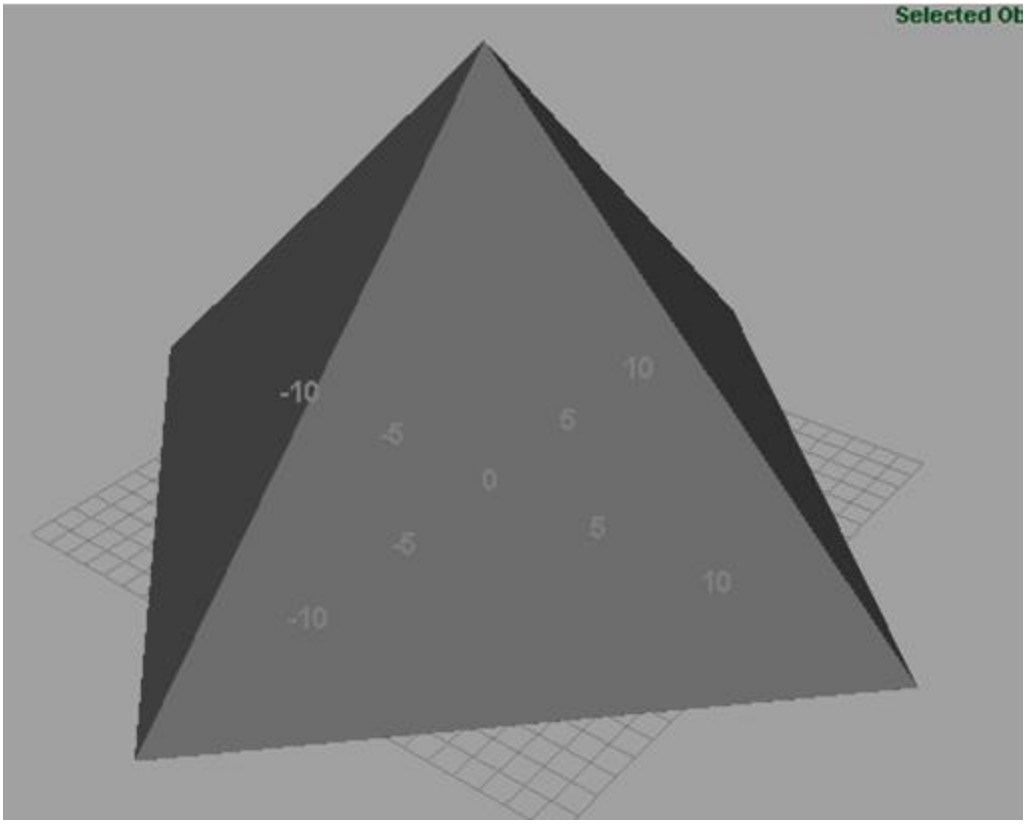
The image above shows a pyramid, with each of its vertices numbered. While the image and the art tools tell you how many vertices there are, the amount that the client renders may be more if you use hard edges.

With hard edges, vertices are not shared by sides of a shape. For example, vertex 1 is processed three times: once for each triangular side, and once for the square base. This means that vertex 1 acts as 3 vertices.

Reducing Memory Use

To reduce memory use:

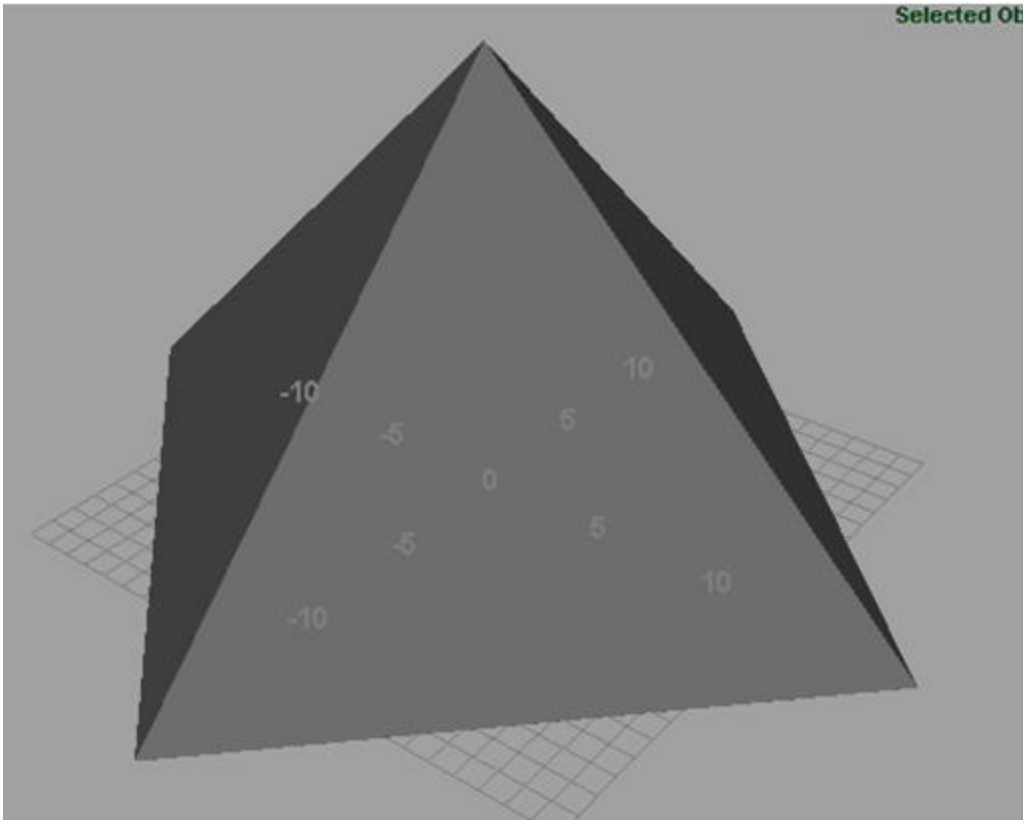
1. Create a normal map of your model (or of the shapes whose vertices you are going to soften/smooth), and apply it to your ATG Material. This will emulate the hard edges in PS Home, without consuming the memory.
2. Select the object(s) that you want to soften:

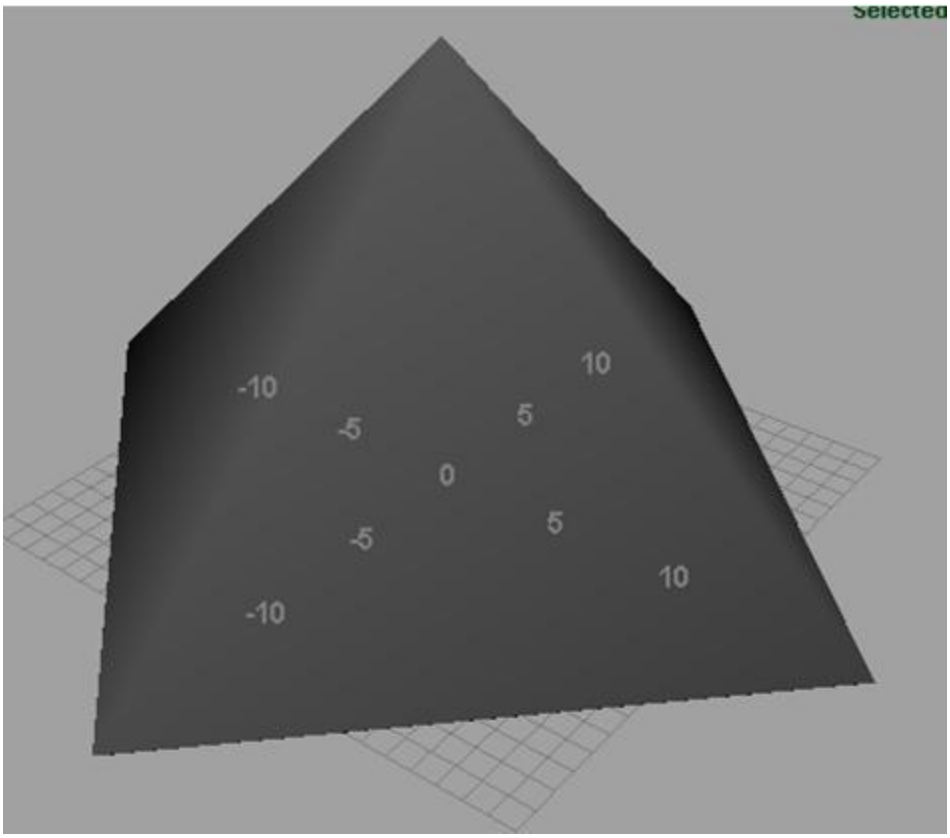


Then select Normals > Soften Edge:



The softened object is displayed:





Environment Optimization

General information, advice and factors to consider for efficient optimization of environments for PS Home.

Environment Optimization Techniques

When optimizing content pay attention to frame rates - especially rendered meshes and other graphic-related elements (as they affect the frame rate greatly).

Frame rate should be the major area of focus for optimization, budgets are fixed and well defined, to optimize in this area you will have to just remove data or reduce meshes to get back into memory.

Rendered meshes

- Rendered meshes will cull based on view. This is measurable by using the camera and the basic debug. The mesh will cull based on its bound so you need to think about what you combine to make it cull efficiently.
- Rendered meshes are created one per material as well as one per object (Maya mesh). Combining two objects with different materials will not reduce rendered meshes. For information on material-mesh combinations, see Models and Meshes.
- LODs are the most effective way to reduce visible meshes. If you LOD an object, you have the option to change the shaders on the LODs. For example: LOD 1 may have 18 shaders on the model, it making 18 meshes in runtime. LOD 2 can have 1 shader making it reduce the runtime meshes to 1. You can also use fewer texture maps or remove elements such as normal and specular maps on lower LODs to reduce the cost of the shader. This, combined with poly reduction, is a massive saving and will result in you being able to have more geometry on the high LODs than before. LODs work on both environments and models.
- Using the console command Profile Mode Overview will tell you everything that's rendering and what's is costing you the most.

Shaders

- Although there are some shaders that are more expensive than others, it is ultimately how you use them and the options that you apply that will make them expensive.
- Consider texture map sizes. High res normal and low res color (or vice versa) will make them much cheaper.
- Use tinting and alpha tinting to get basic alpha effects for free, Tint Alpha will allow you to use alpha on the whole mesh without having an alpha map.
- Only the default texture maps will result in a saving. When used, the engine will simply ignore the whole pass, i.e. no normal or spec will even be calculated. If you add a simple map it will do the calculation so it's not worth reducing the spec and normal quality too much, unless you are very low on VRAM.
- Consider using a different type of shader on a LOD. Maybe you don't need the water to be water at a great distance.
- Where special effects such as reflections are used consider using the reflection view editor to limit what is reflected. You can even do this on a LOD so that it reflects less as it goes further away.
- Use the lighting sets to adjust the shadow casting parameters of shaders to make dynamic content cheaper.
- Look into the engine settings Lua API. It has options to help adjust realtime shadow casting, such as the distance at which it will cull the shadows.
- The blend shader can save you VRAM. It's good for all kinds of effects and saves a lot on areas where multiple shaders and textures would be needed to get the same effect. It's not that much more expensive than a normal shader, but once again, it depends how it is used.

Lightmaps, Vertex and Realtime Lights

1. Combine lightmaps to make them more efficient and cut down on total wasted space. Use texel map attributes to make more reasonably scaled lightmaps.
2. Depending on the situation, Vertex lighting can be good for saving memory. Consider the look you are going for and see whether it may not be cheaper to add extra geometry rather than use a lightmap. It may even be worth considering combining the lightmap onto another map even at a small size.
3. Realtime lit objects can be expensive, but they can also save you memory. Consider objects that will LOD out completely or that can have the shadows cull away for realtime lighting. Realtime lit objects can, however, be a slow and expensive way of working.
4. Consider using a combination of all these things. No one single thing is the best technique.

Instancing – Saving Host Memory with Duplicated Models

1. While duplicating models is not technically true instancing, you will save memory on the initial loading as the .mdl file will only need to load once. This is much cheaper than having them all in the main scene as duplicate unique geometry.
2. You can now have a prelit model, but consider making this object dynamic. It can save a lot of time in working out placement and a few well-placed dynamic objects should not cost too much. You can LOD these as well, so that at a distance they are baked and up close they are dynamic.
3. If you have baked lighting they will all share one lightmap, which is much cheaper, but placement can be more difficult. Consider a more neutral lighting setup.

Designing the Level with Visual Breaks

Designing the level with visual breaks to keep visible meshes down means you can have more complex spaces without the cost of rendering it all at once.

Intelligent design of the level is the best optimization tool. If you intend to have a very large space try to break it down so that lines of sight are broken up to help reduce rendering times.

LODs and bottle-neck areas can help you break up the level.

If you have an extremely large space consider dynamically loading sections of it. You can break down the space into multiple

prelit models to help with streaming if you chose the deferred loading rout.

Alpha and Fill Rate

This is a problem in PS Home as it is in many engines. If you need a large number of alpha meshes you will have issues, try to keep alpha objects smaller if possible. It is the total screen space that matters here, not the complexity of the alpha itself. Try to LOD out to non-alpha materials. Consider using more geometry and less alpha to improve fill rate issues.

High Quality Normal Maps

High Quality Normal Maps are one of the best visual tools you have.

Environment based normal maps can often be flat and uninteresting. Using tools (like transfer maps in Maya and Z-brush) to create normal maps with a high degree of depth will help to improve visual quality immensely. This has the potential to save a massive number of polygons.

Consider a higher resolution normal and a lower resolution color map. This swap can often result in a memory saving but only a small or no drop in quality.

Save your normal as DXT5. It may cost you slightly more, but it should improve the quality of the normal map.

Efficient Art Working

Consider the use of more LODs - make objects and place them in the Scene Editor. These can cull away like any other mesh, but your memory overhead for the data will be smaller.

Think in terms of quality first and optimization second. You can always pull out polys later, but good LODing and use of the Scene Editor will make this easier. It can be as simple as adding an extra LOD so they swap out sooner. But you can't add quality after the fact.

The Maya Interface

The basic layout of the scene is configured and created automatically. To create and validate PS Home content, you can use the Maya tool shelf:



The tools shelves you will use most often are:

- Home_Env Shelf - to develop environments
- Home_Char Shelf - to develop character components
- Home_Furn Shelf - to develop furniture components
- Home_AvatarAnim Shelf - to develop avatar animations

The main menu bar in Maya also gives you access to the Home drop-down menu. See Home Drop-down Menu. Some of the options accessible from this drop-down menu are also available on the shelf.

Home_Env Shelf

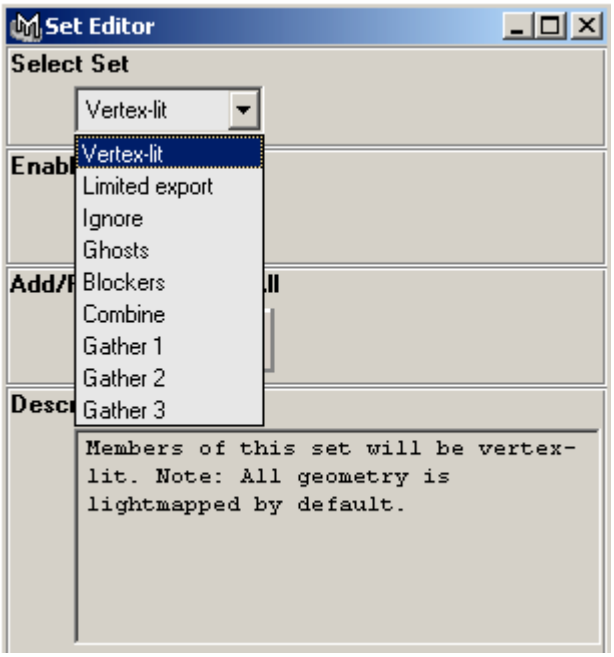
For information on creating environments, see Environments.

Home_Env



The following table describes each of the Home_Env shelf icons:

Icon	Description
	Export Wizard: Launches the Export Wizard.
	Quick Export: Exports the scene using the last settings from the Export Wizard. In Maya, each new file is called a 'scene'. The Quick Export allows you to export any type of Home content you create using Maya, for example, furniture, picture frames, models, scenes.

	Export Log: Open the Export Log. This only opens if you have already exported a scene/object from Maya in this session. See Validating and Exporting in Maya.
	PS3 Model Viewer: Launches the model on PlayStation®3 for preview.
	PS3 Scene Viewer: Launches the scene on PlayStation®3 for preview.
	Create Sun: Creates directional sunlight. This creates a new Lights group, and inside that group creates a directional light called 'sun' on the origin. The default color and intensity of 3.0 is suitable. Move the sunlight so that it is shining in the desired direction.
	Create DomeLight: Creates diffused daylight. This creates a shape node under the same Lights group as the sun, called 'dome'. This domelight simulates the indirect illumination that comes from all parts of the sky during daylight hours. By default, it is a slightly blue color to simulate the sky on a bright sunny day, however, this can be changed as desired. The default intensity of 0.200 should be suitable, but may need to be changed for some scenes, such as simulating the sun positioned low in the sky.
	Lighting Set Editor: Edits or assigns meshes to sets. Clicking the Lighting Set Editor button opens the following dialog:  The Lighting Set Editor creates different meshes via the use of Maya Sets. Meshes can be added or removed from these sets. Meshes can also be added and removed from Lighting Sets in a more conventional way using the Outliner in Maya. If an error is reported within the Description panel, follow the Fix/Migration message. Remember that you must create or open a Home environment before attempting to assign meshes to Sets. For more information on using Lighting Sets, see Creating Lighting Sets.
	LOD Lightmap Generation: Adds attributes to control the generation of low LOD Lightmaps.
	Create Particle Locator: Create particle locator for imported particle effect objects. See ATG Particle Effects Tool. Create particle locator for particle effect objects created using the Particle Systems Tool. See Creating Particle Locators.
	The left button launches the Home Sky UI. The right button launches the Cloud Modeler UI. See Sky Design.

Unable to render embedded object: File (collisionattribute.png) not found.	Creates a collision box, sphere cylinder or capsule shape, respectively. See Collision in PS Home

Validation of the scene is also provided to ensure the scene is ready before exporting to the required format.

Home_Char Shelf

For information on creating character components, see Character Components.

Home_Char



The Home_Char tab has the following icons:

Button	Name	Description
	Export Wizard	Launches the Export Wizard.
	Quick Export	Exports the scene using the last settings.
	Export Log	Opens the Export Log (log only opens if you have already exported a scene/object from Maya in this session).
	PS3 Character Viewer	Opens a tool to load and view your character component.
	Set units to centimeters	Sets Working Units within Maya to centimeters (normally set within Preferences > Settings). This is more useful for animations and character models than environments because it makes navigating within the work area more sensitive.
	Update texture files	Automatically updates all texture files (.DDS, .JPEG and .PSD) used within the scene. This is useful when texture files have been changed outside Maya because the files don't need to be manually refreshed.
	Edit Vertex Normals	Transfers one set of vertex normal values to another, or calculates an average between two vertex normals and applies the average to a third vertex normal. For details see the Vertex Normal Tool.
	Weighting Tool	Transfers a set of weights values from one vertex to another, or calculates an average between two weights and applis the average to a third vertex weight. For details see the Weighting Tool.
	Copy Weights	Propagates the edited weighting down to the other two LODs. For details see the Weighting Tool.
	Paint Skin Weights	Uses a paintbrush tool to apply skin weighting. For details see the Weighting Tool.
	Create blend shapes (LOD 1)	Sets up LOD 1 fat and thin variations for your mesh. For details see Creating Fat and Thin Targets.
	Copy blend shapes to LOD 2 and 3	LOD 1 fat and thin variations of your mesh are copied to LOD 2 and LOD 3. For details see Creating Fat and Thin Targets.
	Create initial skin weighting	Applies PS Home's custom weight setup to the three LODs. This is useful as a starting point from which to work. For details see the Weighting Tool.

	Copy Skin Weights from LOD1 > LOD2	Copies the skin weighting on LOD1 to LOD2.
	Copy Skin Weights from LOD2 > LOD3	Copies the skin weighting on LOD2 to LOD3.

Home_Furn Shelf

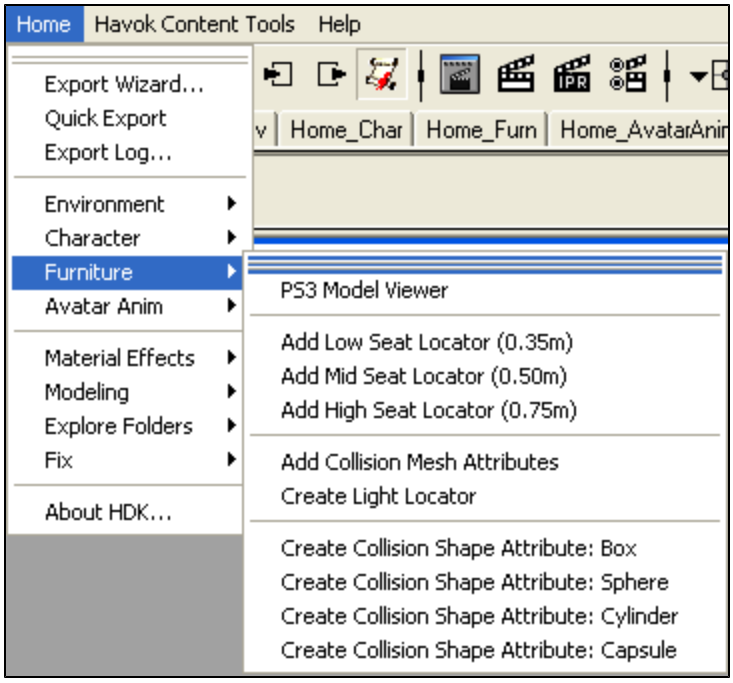
For information on creating furniture components, see Furniture and Decorations.

The Furniture Shelf, Home_Furn contains tools specific to creating furniture, as well as some common tools.

The Home_Furn tab has the following buttons:

Button	Name	Description
	Export Wizard	Launches the Export Wizard.
	Quick Export	Exports the scene using the last settings.
	Model Viewer	Launch on PlayStation®3.
	Add Seat Locator	Place a Character Seat Locator item at the origin of the current Maya Scene with the name 'HomeSeat'. These items show the seating positions on your furniture and display the space used by a seated character.
	Collision Mesh Attributes	Add a 'Convex Mesh' parameter to the currently selected shape.
	Create Light Locator	Add a light locator item that can be moved into position to determine the location of the point light source for a Lamp (see Creating a Lamp).
Unable to render embedded object: File (collisionattribute.png) not found.	Collision	Create a collision box, sphere cylinder or capsule shape, respectively. See Collision in PS Home.

These options are also available on the Home > Furniture menu:



Home_AvatarAnim Shelf

For information on avatar animations, see Avatar Animation.

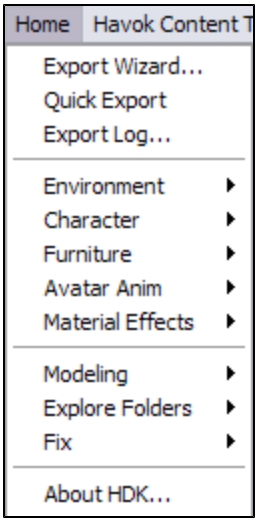
Home_AvatarAnim



	Export Wizard: Launches the Export Wizard.
	Quick Export: Exports the scene using the last settings.
	Launch Puppet Poser: Launches the Avatar Puppet Poser.
	Setup for Export: Prepares you animation model for export. ⚠ Save your avatar animation before clicking this button. Once preparation is complete, it cannot be undone.

Home Drop-down Menu

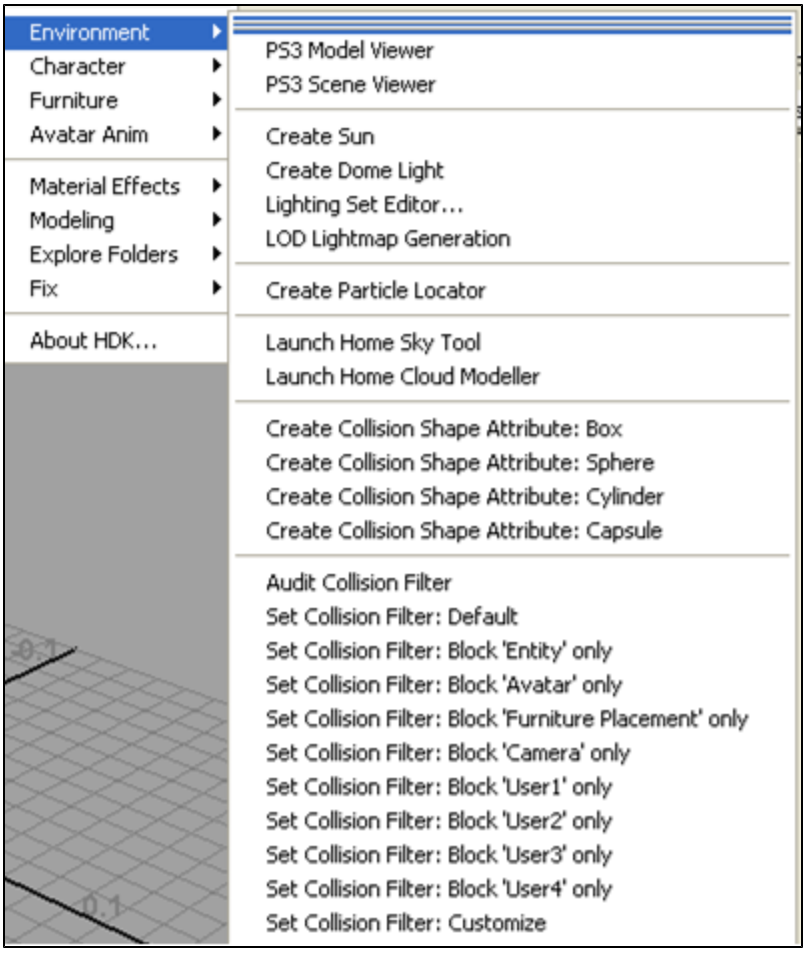
In addition to using the tool shelves for PS Home content creation and validation, you can also access the Home drop-down menu from the main menu bar in Maya:



Some of the options accessible from this drop-down menu are also available on the shelf.

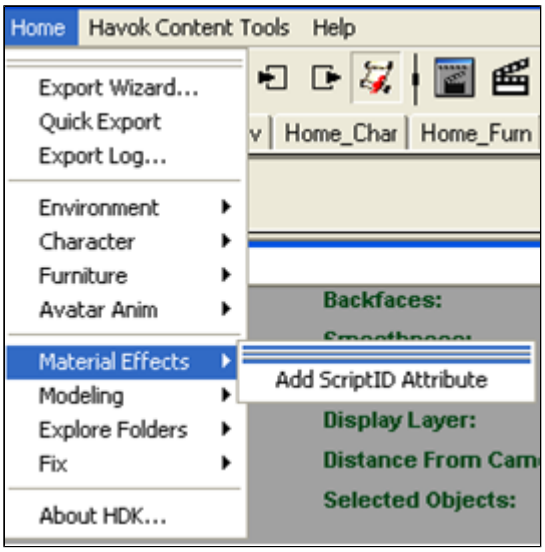
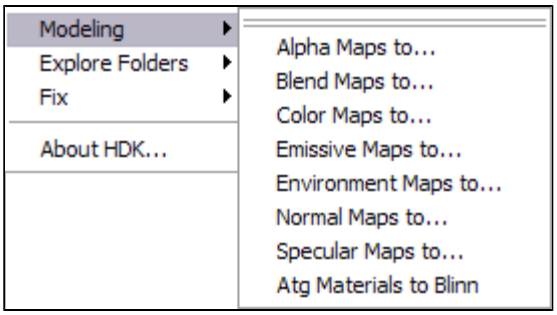
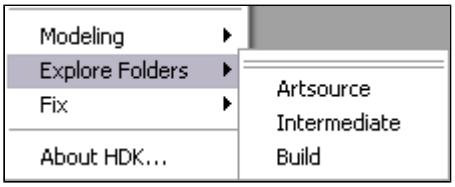
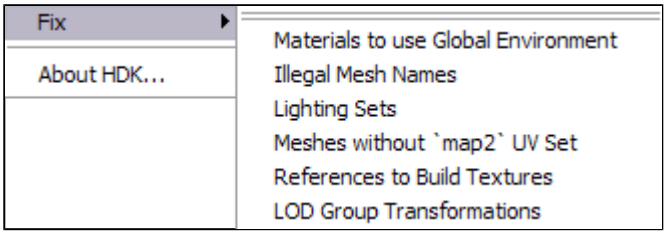


Option	Description
Export Wizard	Launches the Home Export Wizard. See Validating and Exporting in Maya.
Quick Export	Re-exports the scene with the last used export options as specified in the Export.
Export Log	Shows the Export Log generated by the previous export.



Option	Description
Environment	The first items are identical to the buttons that appear on the Home_Env Shelf (see above). • Audit Collision Filter: Opens a dialog showing the collision filters on a selected collision mesh. • Set Collision Filter: Each item in the list (Block 'Entity' only, Block 'Avatar' only, etc.) puts a collision filter on a selected collision mesh, causing only that blocked item to be affected by collision (for example, Block 'Avatar' only causes only avatars to be blocked by the geometry). The Block 'User' items are for setting collision filters for Mini-games and Realtime games where you can have multiple players.
Character	This sub menu contains the same items that appear as buttons on the Home_Char Shelf (see above).
Furniture	This sub menu contains the same items that appear as buttons on the Home_Furn Shelf (see above).
Avatar Anim	This sub menu contains the same items that appear as buttons on the Home_AvatarAnim Shelf (see above).

Options	Description

Material Effects	The only option here is Add ScriptID Attribute which tags a material with a ScriptID that you can reference in Lua script to animate the material. See Scripted Material Animations. 
Modeling	Selects all of certain types of maps and globally sets them. 
Explore Folders	Opens the default file explorer in the specified directory chosen. Please note that the browse options only work if a scene is open within the installation's build folder. The folder opened relates to the files utilized by the current scene. 
Fix	This set of options fixes many problems with existing scenes and migrates previous versions of the scene to the current format of the HDK. Below are descriptions of each fix, with a brief explanation of why you would use a fix and how it would alter your work (with an example wherever possible): 

- Materials to use Global Environment: Ensures the Global Environment node is referenced and the ATGMaterials are rendered correctly in the viewport.

  Why: The HDK uses a Global Environment node to render ATGMaterials in the viewport. This is automatically set up when new ... or open scene is selected from the File menu. However, it is possible that the Global Environment node can be deleted or become corrupted, with the result that the scene fails to display in the viewport. When this occurs, Fix > Materials to use Global Environment resets the node and ensure the viewport updates successfully.

  How: Any existing Global Environment node is removed and replaced with a newly created Global Environment node.

- Illegal Mesh Names: Removes invalid symbols and characters from mesh names. This is required as a mesh name is potentially a filename for its Lightmap file.

  Why: The following characters are invalid as mesh names: ¥/:?"<>| #(){}%&.

  How: Mesh with names containing these characters are renamed with underscores instead. for example, 'import:polysurfaceShape' is renamed as 'import_polysurfaceShape'.

- Lighting Sets: Modifies lighting sets and meshes.

  Why: Meshes are now Lightmapped by default. (In HDK 0.1 meshes were vertex-lit by default).

  How: Remove the redundant file_mesh set. Add default shader meshes into the lighting_vertexLight set. Add emissive shader meshes into lighting_ignore.

- Meshes without 'map2' UV Set: Changes previous formats so that they contain a 'map2 UV Set'.

  Why: All meshes now require a map2 UV Set.

  How: Add a map2 UV set for any mesh which does not have one already.

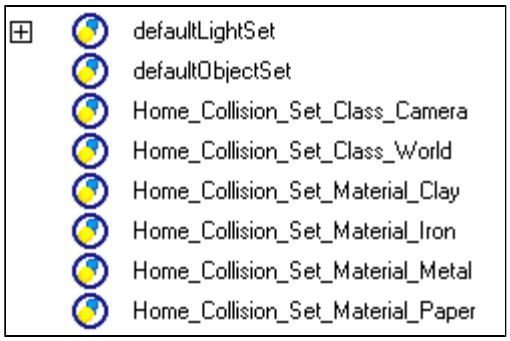
- References to Build Textures: Ensures that textures are within the correct folder.

  Why: Materials should only refer to textures in the artsource folder.

  How: Repoint materials to reference textures from the equivalent artsource folder. Copy the actual texture files into the equivalent artsource folder (this process does not copy over any existing files).

  Example: Here is an example of a texture in the build folder <HDK_INSTALL_DIR>\build\environments\myscene\textures\check.dds and its equivalent artsource folder <HDK_INSTALL_DIR>\artsource\environments\myscene\textures\check.dds

- Collision Sets: Creates a number of sets in the outliner. This feature is not supported by the HDK.



  Why: The Collision Sets are not supported in the HDK and all the information specified here is ignored by the exporter:

- Translated Pivots: Resets the translation pivots after alterations have been made. This feature is not supported by the HDK.

  Why: The Translated Pivots are not supported in the HDK and all the information specified here is ignored by the exporter.

- LOD Group Transformations: Ensures that groups of LODs (especially those with adjusted pivots or large pieces of geometry) work in PS Home.

  Why: This menu item adjusts the LOD group so as to make the groups' 'distance' attribute value more intuitive, allowing easier setup of LOD threshold settings. It also helps ensure that the LOD item switching results seen in the Maya viewport match those seen in PS Home.

  How: Moves the LOD group transform to the average center position of the meshes in the LOD group. Resets pivot to origin.

- Video Screen Audio Meshes Ensures that the audio meshes are positioned on the video screens.

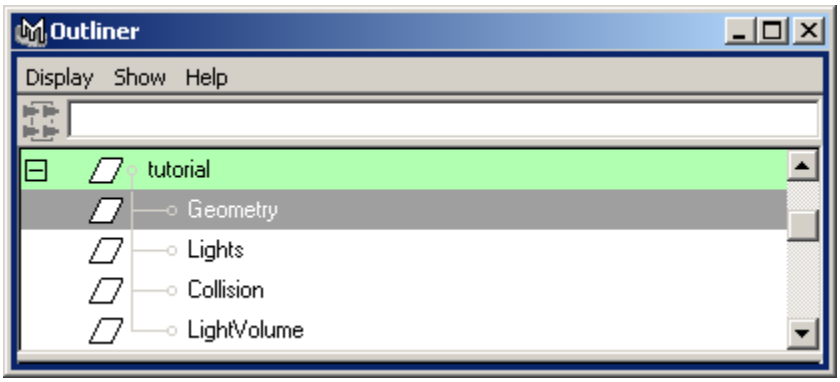
  Why: Sometimes when creating a video screen, the audio mesh in the scene editor appears at the origin rather than correctly oriented.

  How: This fix centers the selections pivot, snaps it back to origin, and then freezes the transformations.

Geometry Guidelines

Drag and drop your geometry within the Geometry subgroup. For convenient organization, we recommend that this group contain all the geometry meshes and transform nodes.

This example shows a typical group structure for environment geometry:



Guidelines

Your geometry must conform to the following rules:

- Keep within the requirements in HDK Tools Validations.
- Make sure the Maya Working Units are always set to meters and that this scale is always adhered to.
- For meshes, only polygonal geometry is valid. NURBS and SubDiv Surfaces are not directly supported and so if used, must be converted to polygonal geometry before they can be exported.
- There is currently no streaming of the environment, or culling of the geometry.
- DXT1, DXT3 and DXT5 comparable texture compression is used in DDS image format.
- All meshes must contain 'map1' and 'map2' UV sets.
- 'map1' UVs must be used for Color, Normal and Specular maps. 'map2' UVs are used for Lightmaps.
- 'map2' UVs must be within the 0 to 1 range and not overlapping. All UV sets must contain UVs.
- To check the UV sets attached to a selected mesh, select Window > UV Texture Editor… > Image-UV Sets.
- All mesh names must be valid as filenames (for example, no colons, angle brackets, etc.).

> Importing meshes by using File > Import often adds colons into mesh names. Remember to check that these are removed.

- All mesh names must be unique (this does not apply to the mesh's transform node).
- The recommended polygon triangle count for environments/scenes is 150,000 triangles. As there is currently no streaming of the environment, or culling of the geometry, this should be adhered to as far as possible.

Maya Textures

Artists should create their textures in artsource in Photoshop .PSD format. We recommend that PSD file should have a layer for each needed texture. Each layer would then be named and labeled explicitly.

When the texture is completed, save out the separate layers as .DDS files in the environment's textures directory.

Name each file and add a suffix to denote what type of texture it is, for example:

- Brick_c.dds = Colour map
- Brick_n.dds = Normal map
- Brick_s.dds = Specular map

Your Maya scene then references the .DDS files from the textures directory only. For example, `C:\HomeHDK\artsource\Environments\testroom_maya\textures`

If the texture is to be modified again at a later date, work on the PSD, save the PSD again (with all its labeled layers), and then save out the relevant layers as DDSs in the textures folder.

> ⚠ Label the PSD layers correctly so that other people can easily pick up your work when necessary.

Although saving out several layers to separate DDS files may be time-consuming, there will be a tool to automate this process in a future release.

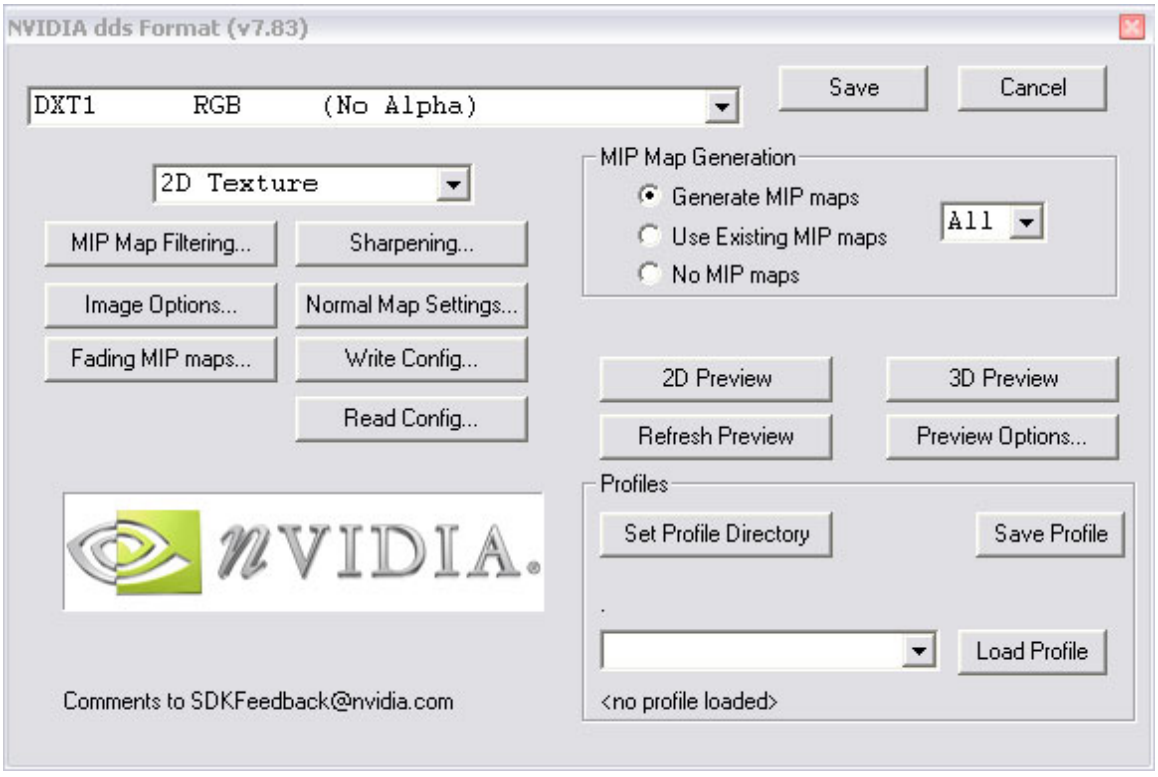
DDS File Types

We recommend using the NVIDIA Texture Tools DDS exporter that is available for Photoshop. This can be freely downloaded from their website.

You will notice there many DDS save formats. However, for now we will use four DDS formats as follows:

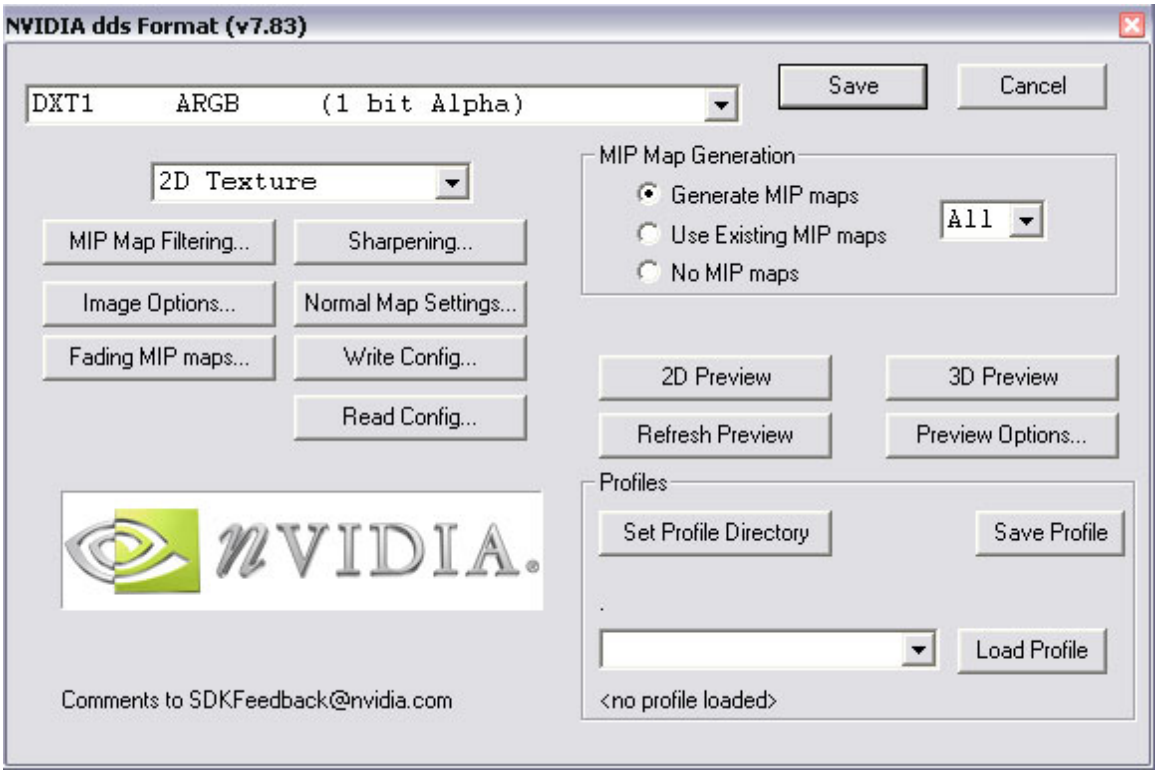
DXT1 RGB (No Alpha)

This is the standard texture with no alpha:



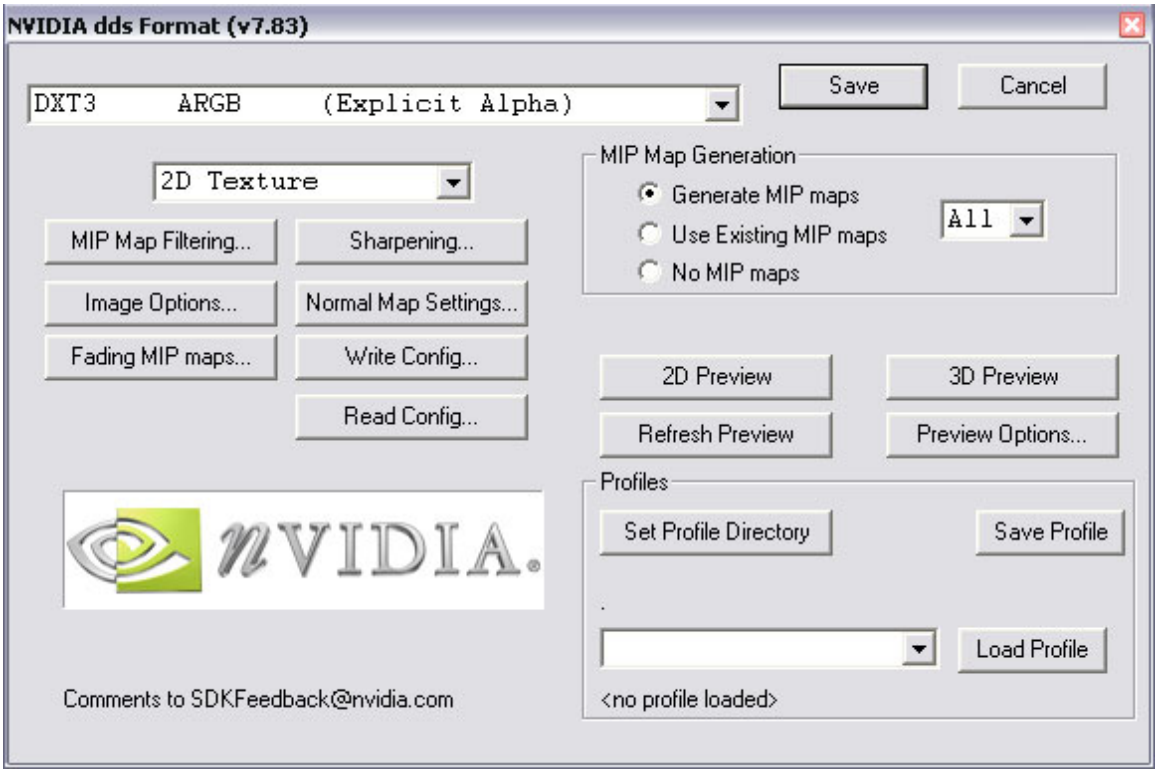
DXT1 ARGB (1 bit Alpha)

This is the texture with 1bit alpha; Alpha on or off, Stenciled Alpha, Black and White Alpha channel:



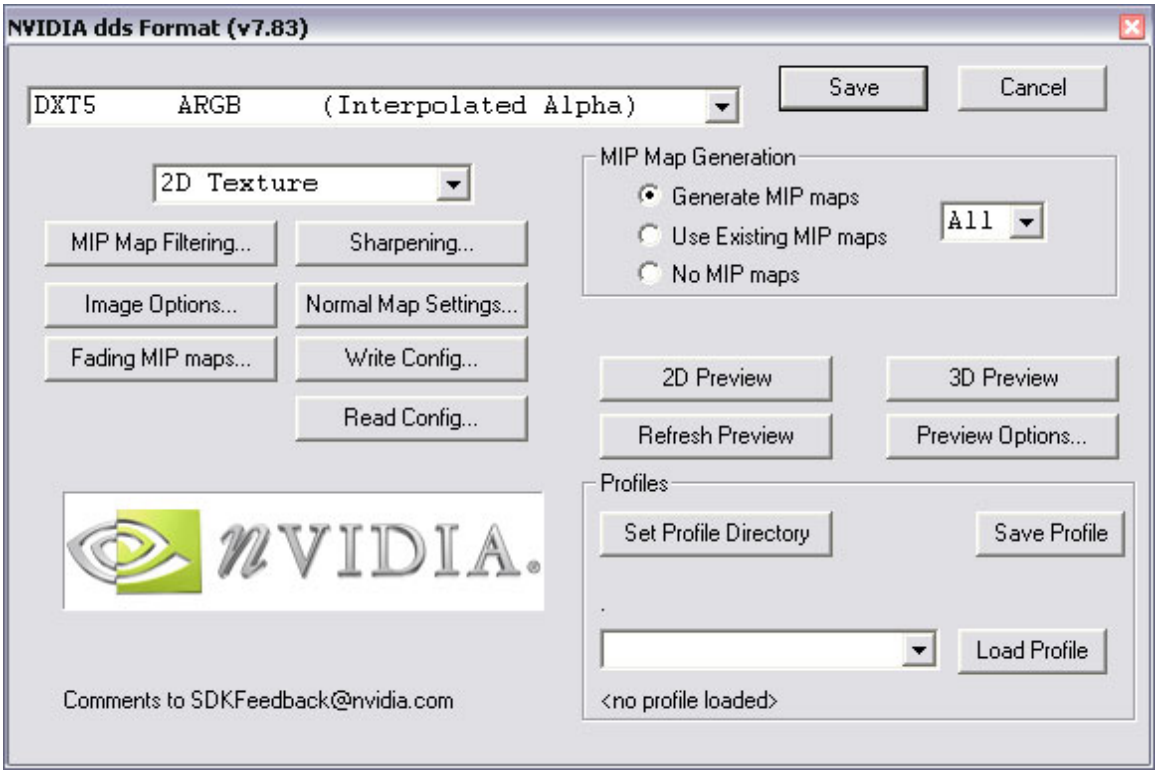
DXT3 ARGB

This is texture explicit non interpolated alpha, full grayscale alpha channel:



DXT5 ARGB

This is texture with fully interpolated Alpha, full grayscale alpha channel:

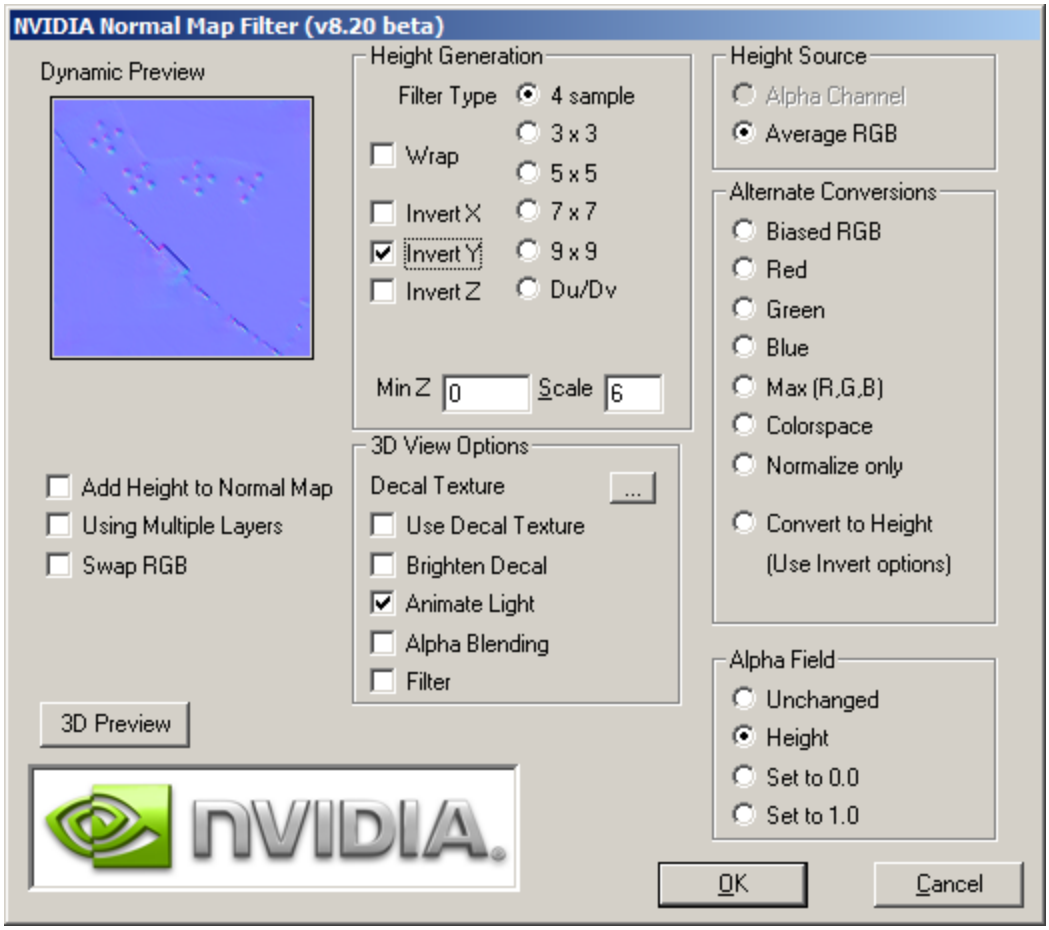


The general rules for choosing which of these three formats to use are:

- If your image has no alpha, use DXT1 compression. Using DXT3/5 will double your image size over DXT1 and not gain anything.
- If your image has 1-bit (on or off) alpha information, use DXT1 with one-bit alpha. If the DXT1 image quality is too low and you don't mind doubling image size, use DXT3 or DXT5 (which one doesn't matter, they'll give the same results).
- If your image has smooth gradations of alpha (fading in/out slowly), DXT5 is almost certainly your best bet, as it will give you the most accurate transparency representation.
- If your image has sharp transitions between multiple alpha levels (one pixel is 100%, the next one is 50%, and another neighbor is 12%), DXT3 is probably your best bet. You may want to compare the alpha results in DXT1, DXT3 and DXT5 compression, however, to make sure.

DDS Normal Map Settings

For many surfaces (relatively flat ones with small surface details, like wood grain or brick) normal maps can be created using the NVIDIA Photoshop filter. By creating the equivalent black and white bump map and running the filter will achieve the correct results. The following image shows the correct setup for the NVIDIA filter.

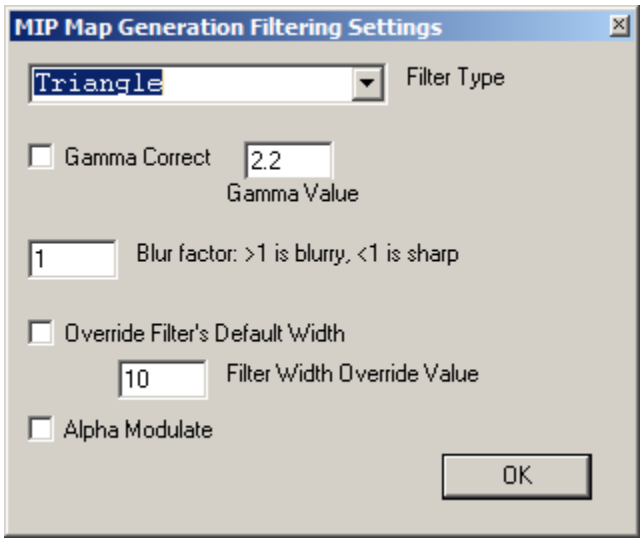


The only options you should change are the Filter Type and Scale. Smaller detailed textures such as fabric would suit a sample type of 4 or 3 and a scale of 5, whereas large details would suit a higher sample type and larger scale.

To produce normal maps that describe a more pronounced 3D geometry, we recommend the use of tools such as Zbrush or Mudbox for normal map creation.

Other Options

In export options, when you click the MIP Map Filtering button, the following dialog is displayed:



The Filter Type drop-down list allows you to specify which filtering method you want for your MIP maps (the default is Triangle). It may take longer for Photoshop to write out the file on certain settings, for example, a Gaussian Filter type may produce a more savage blurring of a given texture as it recedes (MIP maps) into the distance. This may be desirable if you have a particularly busy texture that is still noisy after standard MIP mapping.

Texture Sizes

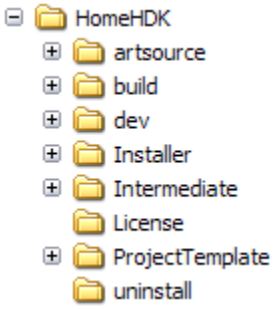
Home client requires textures to the power of 2, up to the limit of 2048 x 2048. However, this hard limit is for exceptional situations only, as this may affect the VRAM to an unacceptable level. A limit of 1024 x 1024 is recommended instead.

Generally the equivalent of a 512 texture set made up of three 512x512 pixel maps, a color map with alpha channel, normal map and specular map with alpha channel would be recommended. This 512x512 set can be split into any size as long as both dimensions are to the power of 2 i.e. 16, 32, 64, 128, 256. Possible sizes could include 256x256 and 128x512, etc.

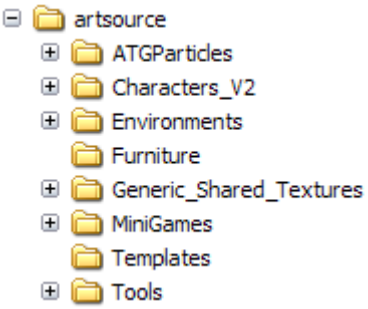
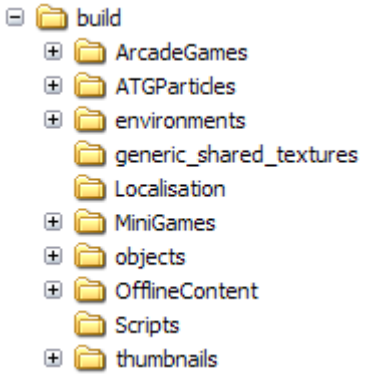
When flat color is used for materials (such as chrome), it is only necessary to create a color swatch 16x16 to achieve the required result.

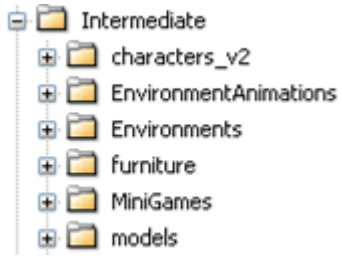
Maya File Structure

Home Art Tools is installed within the following structure:



Only the artsource, build and intermediate folders are covered.

Area	Description
artsource	The source files is located in the sub-folder, for example, an environment scene is stored under `artsource/environments`. This sub-folder is automatically generated: 
build	When you export, files ready for packaging and publishing to the PS Home World is output to the build folder: 

intermediate	Home tools require an intermediate folder for exporting the scene.  Intermediate files are created when you export from Maya. These files are then used by the Object and Scene Editors when you package your content. See Validating and Exporting in Maya.

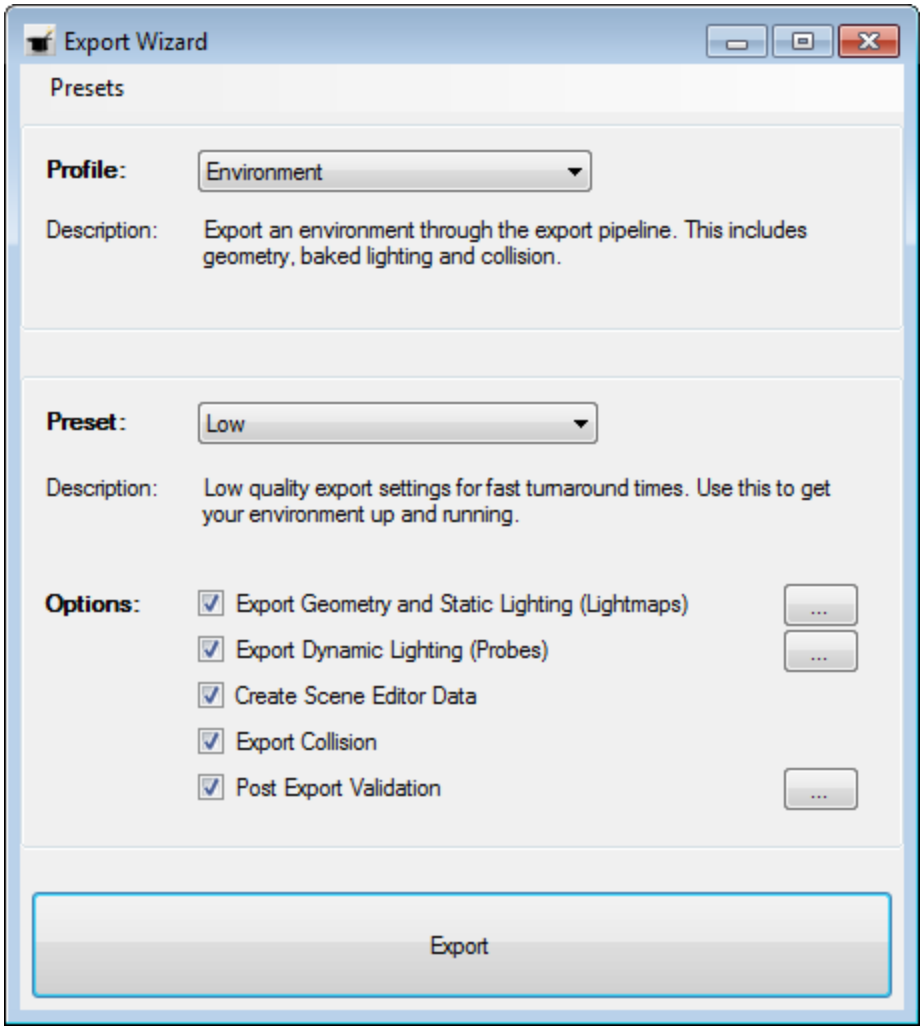
Validating and Exporting in Maya

To export your work:

1. Launch the Export Wizard from the Home drop-down menu or by clicking  .

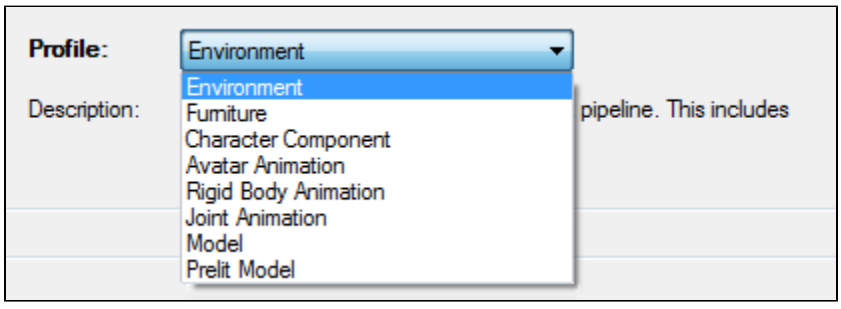
   The Quick Export option (also available from the Home Tool shelf or the Home menu) exports the scene using the last settings used.

   The Export Wizard dialog is displayed:



The default selections in this dialog can be mostly be left unchanged. However in certain circumstances (for example, if you were working on just your collision geometry) you can uncheck all boxes except Export Collision. This saves the exporter having to re-export the environment geometry or re-calculate the Lightmaps and probes.

2. Select one of the Export Profiles from the Profile drop-down list:



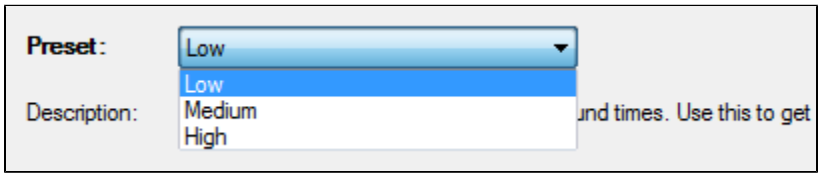
When you export, the project's data is exported to an intermediate folder, which is automatically created with the same name as the Maya file when the first export takes place. This folder contains a number of files required by PS Home to build the final environment and an export log file that contains useful data about the export.

See details of what the Export Profiles consists of here:

Maya Rigid Body Animation
Maya Avatar Animation Scene
Maya Character Component Scene
Maya Environment Scene

Maya Furniture Scene
Maya Model Scene
Maya Joint Animation Scene
Maya Prelit Model Scene

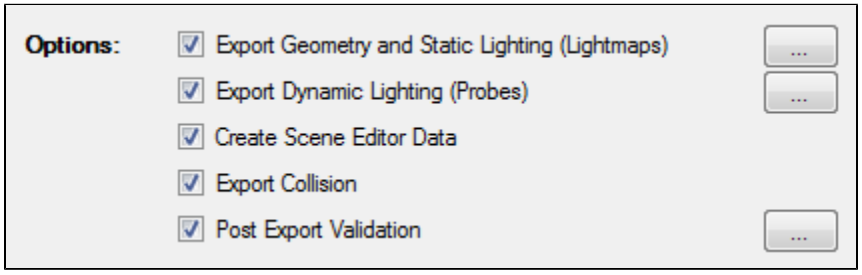
3. Select the export quality from the Preset drop-down list:



This gives the following export quality options:

- Low: The default option. Select this option when developing and debugging your environment artwork. It configures the exporter options to give the fastest export time.
- Medium: Select this option when developing your environment lighting. It improves the definition of the lightmapping and probe lighting. Be aware that it will affect export times. The exact impact depends on the complexity of your scene.
- High: Select this option when you are close to the final output for the lighting solution. The impact on export times is likely substantial. Depending on the complexity of the geometry, the lighting and the specs of your hardware, you might find that it takes several hours to compute the results. The render quality settings can be set higher still, but you would only want to use them to get the very highest quality results.

4. For custom presets, check or uncheck the export option boxes, as required:



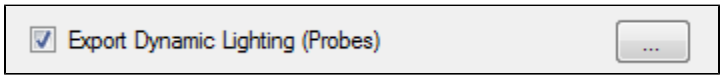
All these options are checked by default. However, you can choose to export only certain parts of an environment, like collision geometry by unchecking the boxes next to the options you don't require. This saves the exporter having to re-export the environment geometry or re-calculate the Lightmaps and probes.

You can also change the export parameters by clicking the 

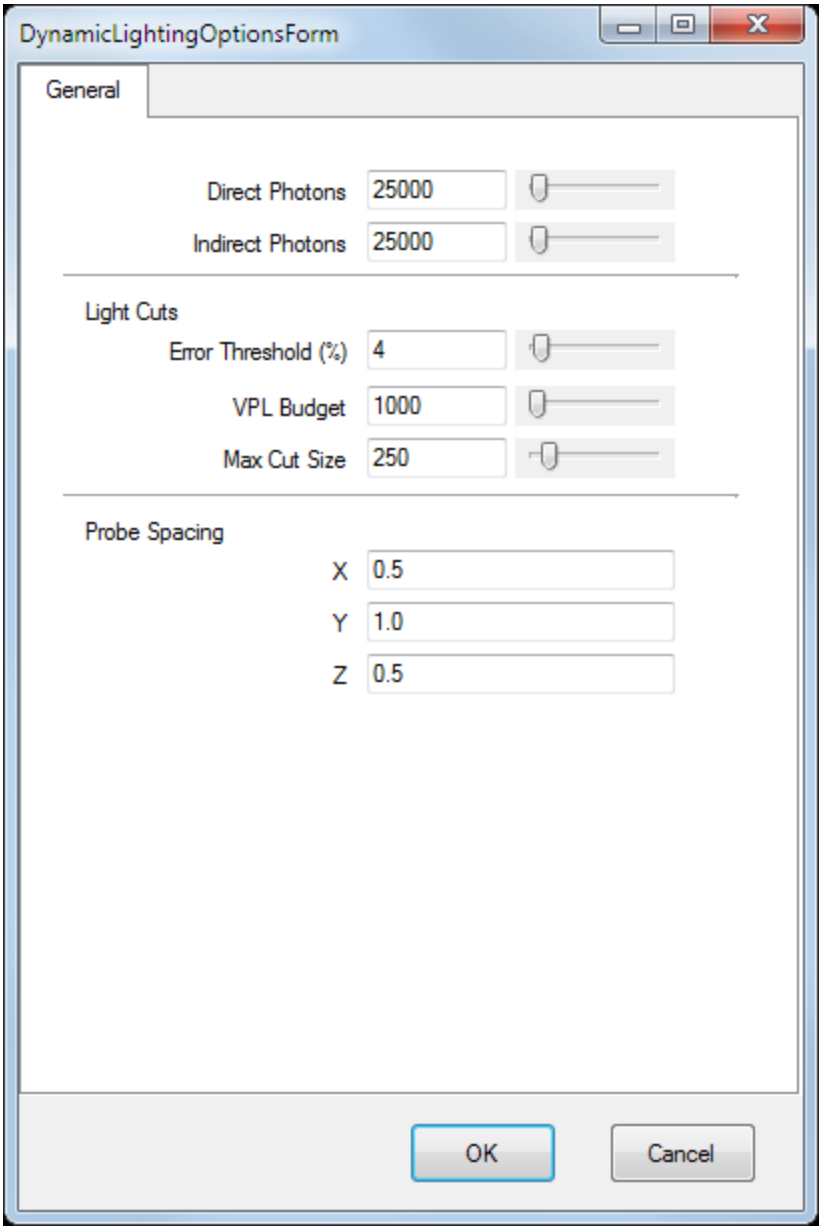
buttons for further options.

For example, if you want to change the probe density, click 

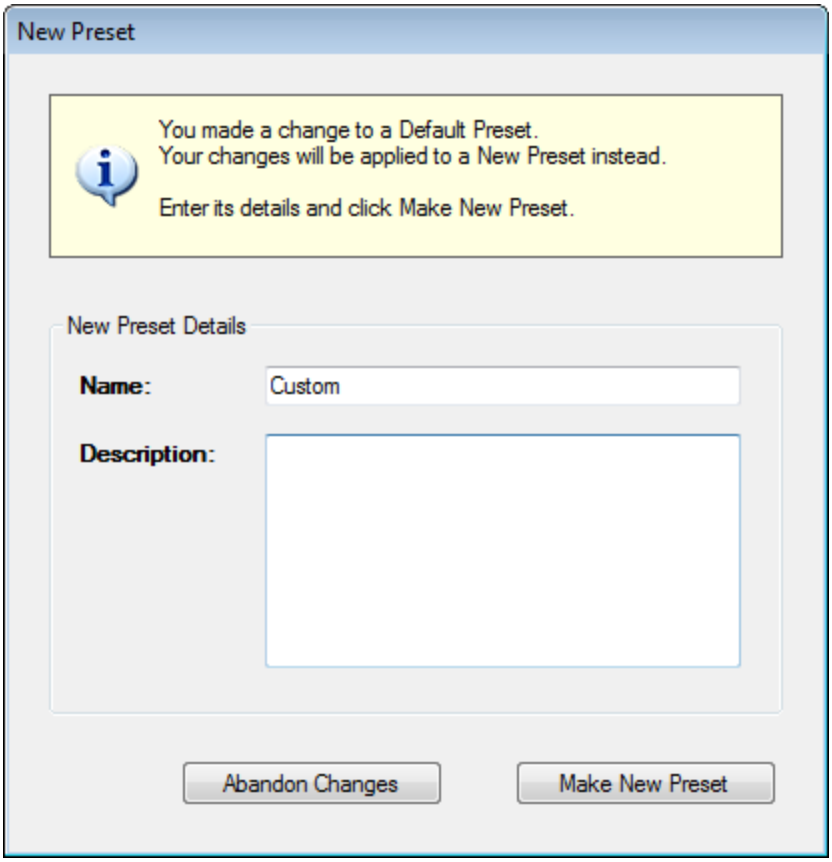
next to the Export Dynamic Lighting (Probes) box:



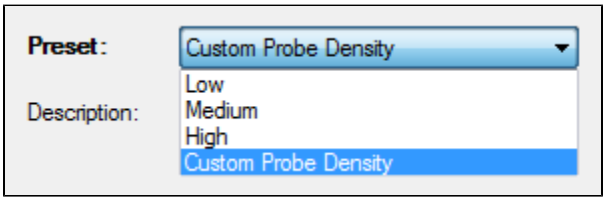
The DynamicLightingOptionsForm dialog is displayed and you can then modify the dynamic lighting parameters:



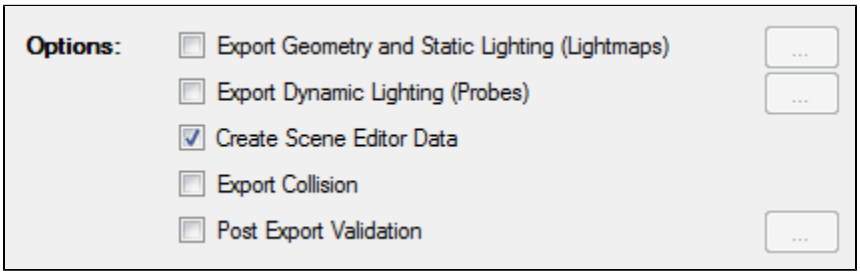
If you have, for example, changed the Probe Spacing X field from 0.5 to 0.25, the New Preset dialog is displayed so you can describe your new preset:



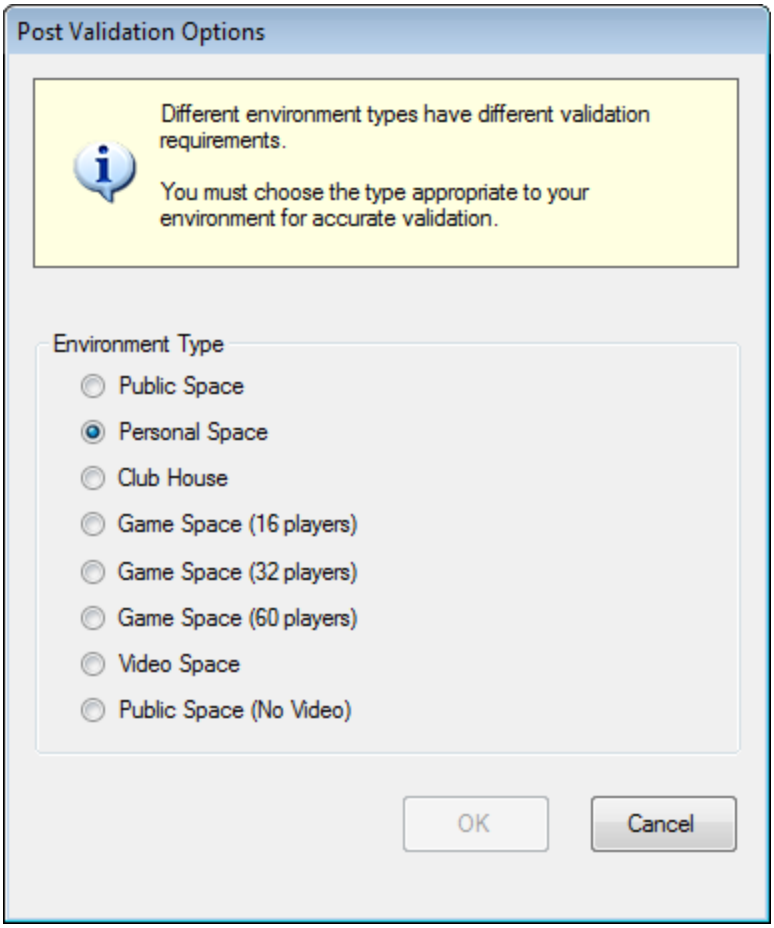
Any further parameters you change are then saved to this custom preset and the custom preset is then added to the Export Wizard's Preset drop-down list and is saved along with the Maya file:



5. Particle Effects only: To update particle effects, ensure that only the Create Scene Editor Data box is checked:



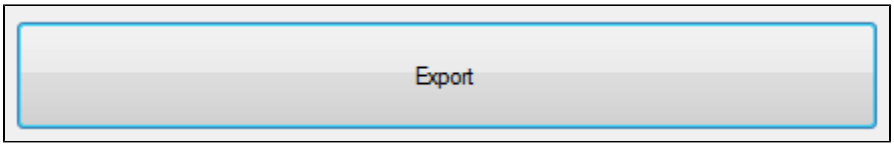
6. To select a scene type, click  beside Post Export Validation. The Post Validation Options dialog is displayed:



You can change the scene type later when packaging the scene in the Scene Editor for submission to the Content Delivery System (CDS). See PS Home Spaces for an explanation of each scene type.

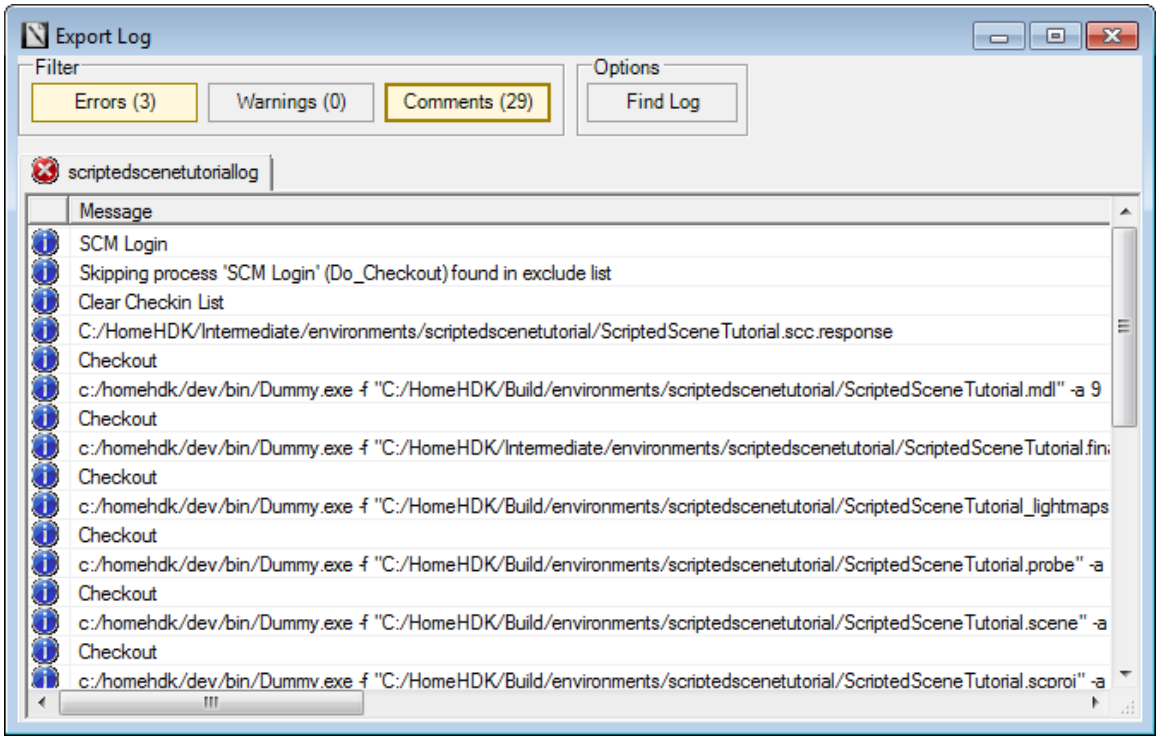
If the Post Export Validation box is checked, the Export Wizard will validate your content to make sure it complies with PS Home requirements. For more information on what is validated, see HDK Tools Validations. Only uncheck this box if you are sure you won't be packaging the content for submission to PS Home (for example, if you are just testing the export to Scene Editor or Object Editor).

7. Click Export:



The Home Exporter exports the Maya content.

At the end of the export process, the Export Log dialog displays the results of the export and lists any errors, warnings and comments, for example:



If there are no errors, select Next to move onto the next step. If the log reports errors, you must correct them before continuing.

When creating environments, the export process may take some time, depending on the complexity of your content. If your export is successful, "Export Complete" is displayed.

When validation and export have successfully completed, you can edit the scene or object/component created in Maya (if necessary) and package it in the appropriate editor. Use the Scene Editor for scenes and the Object Editor for objects or components. The scene, object or component must successfully export and be packaged by the associated editor before you can submit it for quality assurance.

Export Profiles

This section has an explanation of exactly what is in (required or produced by) each of the Export Profiles

- Maya Avatar Animation Scene
- Maya Character Component Scene
- Maya Environment Scene
- Maya Furniture Scene
- Maya Joint Animation Scene
- Maya Model Scene
- Maya Prelit Model Scene
- Maya Rigid Body Animation

Maya Avatar Animation Scene

Avatar Animation

This scene uses the default Home avatars animation rig. It will generate animations for use with Repertoires within PS Home.

Which shaders can be used:

- N/A - No meshes are exported from this scene type

Lighting Model:

- N/A - Only animation data is exported

Requirements:

Only files with the Home Avatar Rig are exported from this scene.

Output Files:

- .ani (Animation file - Contains only animation data to be used the Home Avatars Rig)

Avatar Animation Export Options

☑ Export Animation - Export the .ani file

Maya Character Component Scene

Character Component

The character component scene will contain either a body or head based character component weighted to the default Home avatar skeleton.

Which shaders can be used:

All shaders except the prelits

- Default
- Blend
- Glass
- Hair
- Skin
- Water
- Emissive

Lighting Model:

Models are real time lit. They will use the Light Probes and Dynamic Sun for there lighting. They will cast a dynamic shadow.

- Light Probe Data
- Sunlight

Requirements:

There must be a mesh with 3 Levels Of Detail. Some character components also require Fat and Thin blend shapes.

Mesh names must also follow the naming convention.

Output Files:

- .mdl (Model file) (LOD1)
- .mdl (Model file) (LOD2)
- .mdl (Model file) (LOD3)

Character Component Export Options

☑ Export Geometry (LOD1) - Export the **.mdl** file
☑ Export Geometry (LOD2) - Export the **.mdl** file
☑ Export Geometry (LOD3) - Export the **.mdl** file

☑ Export Update Object Catalogue - Updates the object catalog with the new component to allow selection in game.

☑ Post Export Validation - Will check against basic character component validations for you (Recommended this is left on)

Maya Environment Scene

Environment

The Environment is the main Home space. It will include prelit artwork and collision.

Which shaders can be used:

All shaders except the prelits

- Prelit Default
- Prelit Blend
- Prelit Glass
- Hair
- Skin
- Water
- Emissive

It is also possible to use the dynamically lit versions of the shaders, However you must not mix these on the same mesh. Meshes cannot have both prelit and default materials assigned.

- Default
- Blend
- Glass

Lighting Model:

Prelit Models have baked in lighitng, They can use the light sets and all other baked lighting tools. The sunlight placed in the scene will be for this model only. It will not affect the final scene you add the model to.

- Sunlight & Dome Light
- Lightmaps / Vertex Lighting
- Light Sets

Requirements:

A prelit model scene must have a Sun and Dome light and at least one mesh with a prelit material assigned in the scene to export.

Output Files:

- .mdl (Model file)
- .col (Collision file)
- Lightmaps
- Light Probe Data

Environment Export Options

☑ Export Geometry and Static Lighting - Export the **.mdl** file, this will include lightmaps based lighting

Additional options are available for Baked Lighting

☑ Export Dynamic Lighting (Probes) - Export the **.probe** file. This is generated based on the lights in the scene and the probe volume.

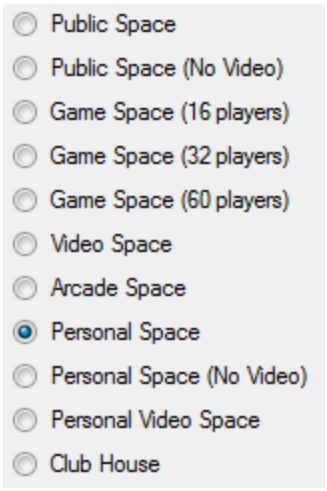
Additional options for Probe Lighting

☑ Create Scene Editor Data - Create a scene file for the environment and set the .mdl, .col and .probe files as resources.

☑ Export Collision - Export the **.col** file

☑ Post Export Validation - Will check against basic environment validations for you (Recommended this is left on)

Additional options allow you to set the scene type to validate against.



Maya Furniture Scene

Furniture

The furniture model can be used in a number of ways. It can be embedded in a scene or added to a furniture object (to allow it to

be placed by the player from the furniture menu). Its also possible to have active furniture items and arcade cabinets for arcade games. All of these object types use the furniture export profile.

Which shaders can be used:

All shaders except the prelits

- Default
- Blend
- Glass
- Hair
- Skin
- Water
- Emissive

Lighting Model:

Models are realtime lit. They will use the Light Probes and Dynamic Sun for there lighting. They will cast dynamic shadows.

- Light Probe Data
- Sunlight

Requirements:

Furniture scenes must have Mesh and Collision volumes to export.

Output Files:

- .mdl (Model file)
- .col (Collision file)

Furniture Export Options

☑ Export Geometry - Export the **.mdl** file

☑ Export Collision - Export the **.col** file

☑ Update Object Catalogue - Updates the object catalog with the new furniture item to allow selection in game.

☑ Post Export Validation - Will check against basic furniture validations for you (Recommended this is left on)

Maya Joint Animation Scene

Joint Animation

Joint Animations are similar to Rigid Body Animations (compare with the Maya Rigid Body Animation page) except they use Joints.

> For more information on animation, see also Joint Animation and Rigid Body Animation.

Joint Animations consist of an animated model that uses Joints, this is does not use the default Home avatars skeleton but instead has a custom skeleton. They can be used in the scene as an animated prop or as part of a game for an npc. Joint Animations are also used in companion objects.

All shaders (except the prelit shaders) can be used, e.g.:

- Default
- Blend
- Glass
- Hair
- Skin
- Water
- Emissive

Lighting Model:

Models are realtime lit. They will use the Light Probes and Dynamic Sun for there lighting. They will cast a dynamic shadow.

- Light Probe Data
- Sunlight

Requirements:

The Joint Animation must have joints that are skinned to the mesh. There can only be 4 influences when skinning.

Output Files:

- .mdl (Model file)
- .skn (Skeleton file - Contains the joints used)
- .ani (Animation file - Contains only animation data to be used with the .skn file)

Joint Animation Export Options

☑ Export Geometry - Export the **.mdl** file

☑ Export Skeleton - Exports the skeleton (Joint Data)

(info}When animations are the only thing that is needed the first 2 options can be disabled. This can be useful if you have multiple animations for the same object.(info}

☑ Export Joint Animation - Exports animation data.

Maya Model Scene

Model

A model can be placed in the Scene Editor or used via Lua. By exporting and placing a model rather than including it in the main Maya file as unique geometry, it is possible to save memory by reusing the same model file data over and over.

Which shaders can be used:

All shaders except the prelits

- Default
- Blend
- Glass
- Hair
- Skin
- Water
- Emissive

Lighting Model:

Models are realtime lit. They will use the Light Probes and Dynamic Sun for there lighting. They will cast a dynamic shadows.

- Light Probe Data
- Sunlight

Requirements:

There are no fixed requirements to export a model.

Output Files:

- .mdl (Model file)
- .col (Collision file)

Model Export Options

☑ Export Geometry and Static Lighting (Lightmaps) - Export the **.mdl** file

Additional options are available for Baked Lighting

☑ Export Dynamic Lighting (Probes) - Exports the **.probe** file

Additional options for Probe Lighting

☑ Export Collision - Export the **.col** file

Maya Prelit Model Scene

Prelit Model

A prelit model can be placed in the scene editor or used via Lua. By exporting and placing a model rather than placing it in the

main Maya file as unique geometry its possible to save memory by reusing the same model file data over and over. The prelit model differs from the model in its lighting format.

Which shaders can be used:

All shaders except the prelits

- Prelit Default
- Prelit Blend
- Prelit Glass
- Hair
- Skin
- Water
- Emissive

Lighting Model:

Prelit Models have baked in lighitng. They can use the light sets and all other baked lighting tools. The sunlight placed in the scene will be for this model only. It will not affect the final scene you add the model to.

- Sunlight & Dome Light
- Lightmaps / Vertex Lighting
- Light Sets

Requirements:

A prelit model scene must have a Sun and Dome light in the scene to export.

Output Files:

- .mdl (Model file)
- .col (Collision file)
- Lightmaps if used

Prelit Model Export Options

☑ Export Geometry - Export the **.mdl** file

☑ Export Collision - Export the **.col** file

Maya Rigid Body Animation

Rigid Body Animations

Rigid Body Animations are similar to Joint Animations (compare with the Maya Joint Animation Scene page) except they do not use Joints. To create an Rigid Body Animation you will just need to add keyframes to a mesh and animate it directly.

> For more information on animation, see also Joint Animation and Rigid Body Animation.

Which shaders can be used:

All shaders except the prelits

- Default
- Blend
- Glass
- Hair
- Skin
- Water
- Emissive

Lighting Model:

Models are real time lit. They will use the Light Probes and Dynamic Sun for there lighting. They will cast a dynamic shadow.

- Light Probe Data
- Sunlight

Requirements:

Rigid Body Animations must not have joints and must have keyed transforms for meshes.

Output Files:

- .mdl (Model file)
- .skn (Skeleton file - Contains the joints used)
- .ani (Animation file - Contains only animation data to be used with the .skn file)

Rigid Body Animation Export Options

☑ Export Rigid Body Animation (All) - all of the above files will be generated on export. It is not possible to export them separately.

Animation Tools

Home Animation Tools

This sections outlines the animation tools available within Home for both Avatar animations and Joint animations.

The key sections are the discussion of the Repertoire system and the Repertoire Editor used to create and format animations to be used with PS Home avatars.

The formats of joint animations are covered as well as the compression tools available for both.

- Animation Compression Tools
- Avatar Animation Tools
    - Avatar Animation - Creating and Exporting
    - Avatar Animation Puppet Poser
    - Avatar Animation Puppet Rig
    - Optimizing Animation to Reduce Memory Usage
- Joint Animation and Rigid Body Animation

See also:

Avatar Animation Repertoires

Animation Compression Tools

Animation Compression

It is possible to compress both Avatar and Joint animations in PS Home. The same rules apply to both, but due to the custom nature of Joint Animation, the tools for these are slightly different.

Template
The Avatar Animation tools allow you to load and save a template file with compression settings. A good general use compression template for Avatar Animations can be found here: Compression Template.

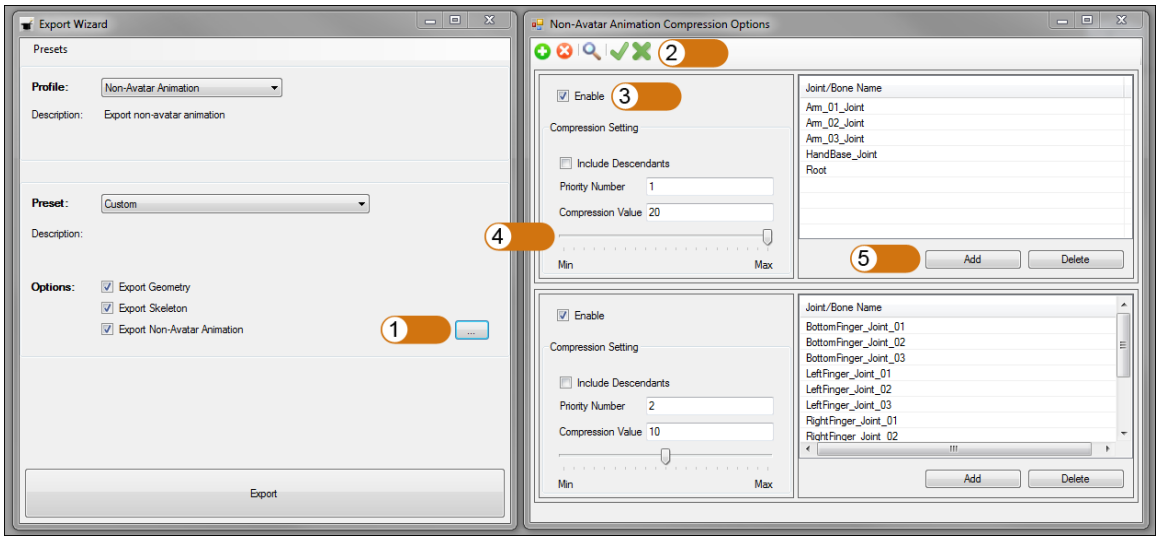
Joint Animation Compression will allow you to reduce the file size of your animations. The amount of compression and the memory saving will depend heavily on the number of joints and more importantly on the curves and key frames of your animation. In general terms: increasing the compression will reduce both the file size and the quality of your animation.

> The same rules also apply to Avatar and Joint Animation compression: More compression = Smaller file size + Less animation quality

All animation compression tools are found under the Additional Options button in the PS Home export dialog.

Joint Animation Compression Tools

During export, you will have an option for compressing Joint Animation. The image and explanations below shows the main areas for setting compression:



1. This button  will give you access to the compression options
2. Options to add groups of joints and to save the groups when done.

    - Add Group
    - Remove Selected Group
    - Examine Groups
    - Save
    - Cancel / Exit

    > Note: You must save the groups before you exit.

3. These are the basic options for the group. The check box will enable / disable the group

    - Enable: Enables or disables a compression profile.
    - Include Descendants: If the joint has descendants, for example, Root, it includes them in the compression profile.
    - Priority Number: If the same joint is included in several groups , only the group with the highest priority is used to compress the joint. Useful when using a root joint that has some descendant joints that you want to compress with different profiles, and thus with a higher priority.

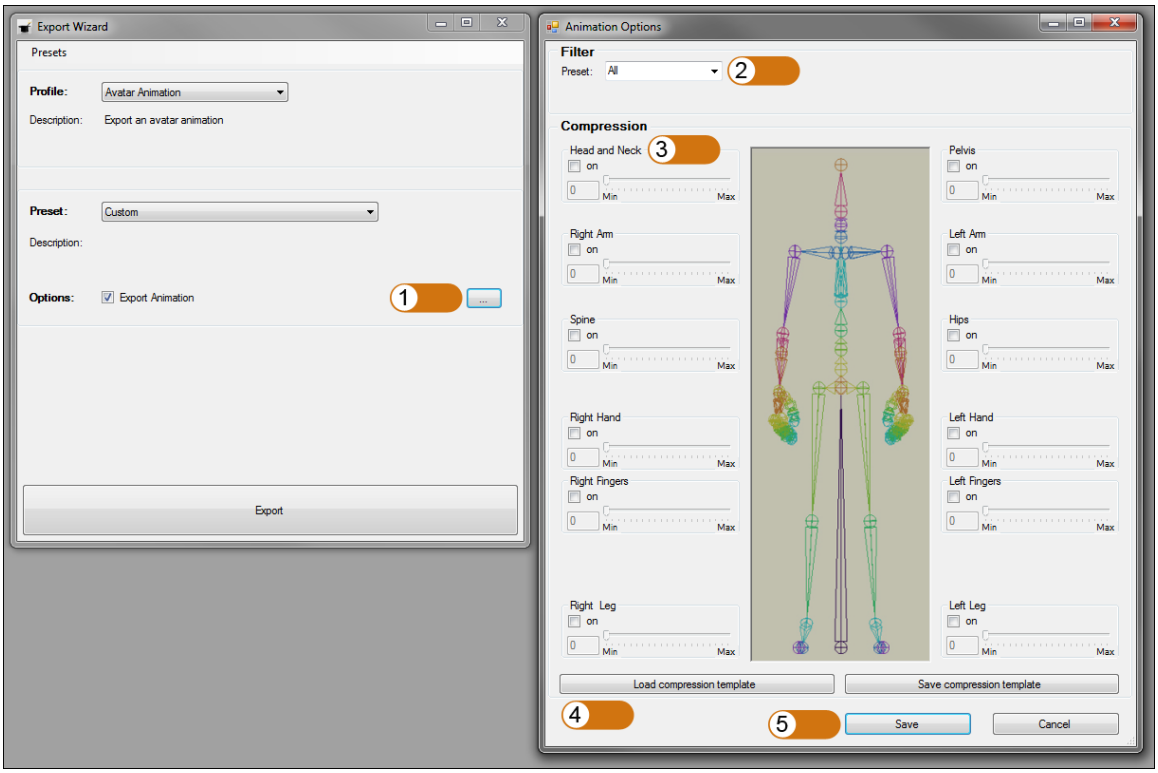
    > Set priorities because group compressions do not cascade
    > Because groups do not cascade, priority numbers are used.
    > For example, if the first group specifies a compression setting for the root - with "include descendants" on - it will apply to the whole skeleton.
    > If you then have a second group that applies to hands only, the root group compression settings will override the hands compression settings unless you specify a higher priority for hands.
    > To have your hands compression settings apply, set priority higher than root compression settings.

4. The compression value, 0 - 20 the higher the value the more compression.

5. Add and Delete Joints from the group. This will open an additional dialog to select the joints from.

Avatar Compression Tools

During export, you will have an option for compressing Avatar Animation. The image and explanations below shows the main areas for setting compression:



1. This button  will give you access to the compression options

2. Presets for which joint are compressed are available. This will limit which joints are included in the compression.

    - All
    - Head
    - Upper Body
    - Lower Body
    - Right Arm and Head

3. Joint Selections, the body is broken down into the main joint selections. They are turned on with the check box, the compression value can then be set per joint selection with the slider or by entering the value.

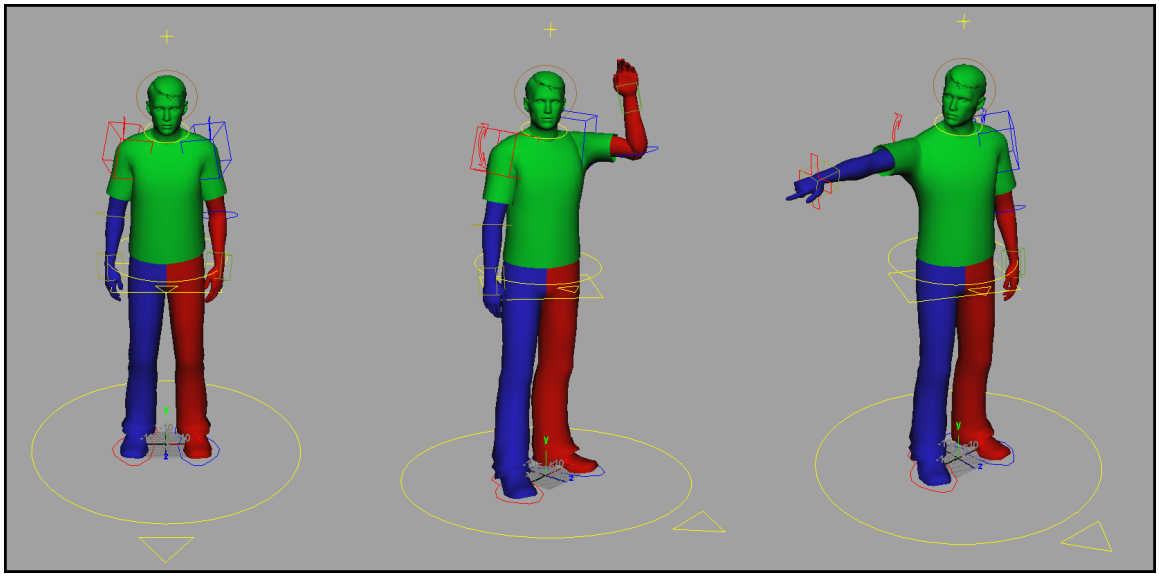
4. Load compression template *and *Save compression template. The current compression setup can be saved and loaded for use with other animations. This can be useful if you have multiple animations that will all need exporting with the same compression.

> Template
>
> A good general use compression template is available here: Compression Template

5. Save or Cancel compression values.

Avatar Animation Tools

Avatar Animations



Avatar Animations are used in Minigames, Realtime Games or FullBodySuits.

You use the default PS Home avatar rig (same skeleton used in all of PS Home) when creating an avatar animation.

See How to create a new Avatar Animation for step-by-step guidance.

The Puppet Poser is a tool that contains may of PS Home's default poses. It also has some tools to help with the animating process. Use it to set the initial pose to start animating from.

See the Puppet Rig for helpers and controllers that allow you to create PS Home avatar animation.

Male and Female Animations

The Male and Female rigs are very similar. The Male rig is slightly taller, but it is possible to copy animations from one rig type to the other.

For more information on differences between the rigs, see the Puppet Rig.

Playing Avatar Animations In PS Home

How and when avatar animations are played in PS Home is controlled via Repertoires. Repertoires are added to Minigames, Realtime Games or FullBodySuits via the Object Editor.

Once added to the Object Editor, you can edit the repertoires in the Repertoire Editor, which you launch from within the Object Editor.

In the Repertoire Editor you can construct a network of nodes that control how and when animations should play.

Avatar Animation – Creating and Exporting

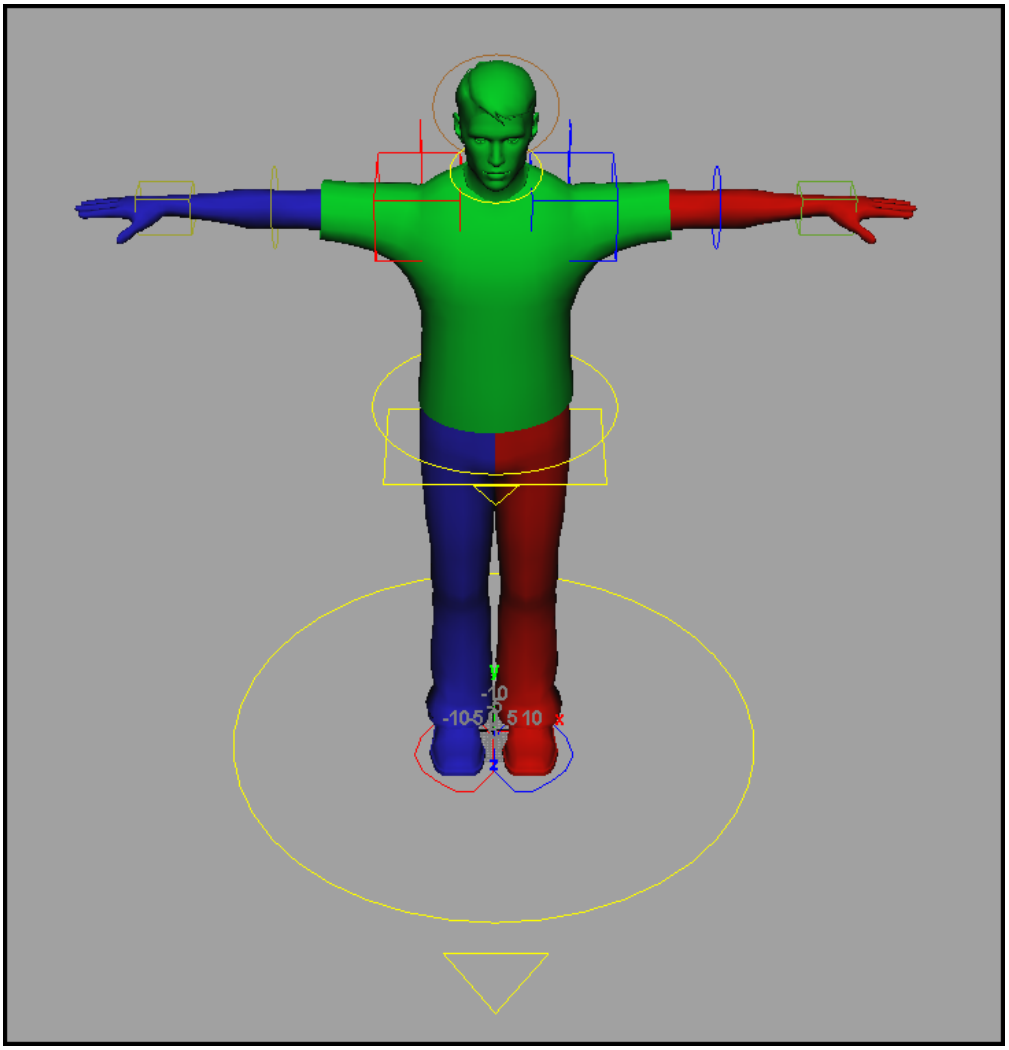
Creating an Avatar Animation

To create a custom animation:

- Select File > New Home Avatar Animation.

This will create a blank scene with the Home Avatar Rig in it.

The Puppet Poser contains several useful tools for editing the animation. Use it to set the initial pose to start animating from.

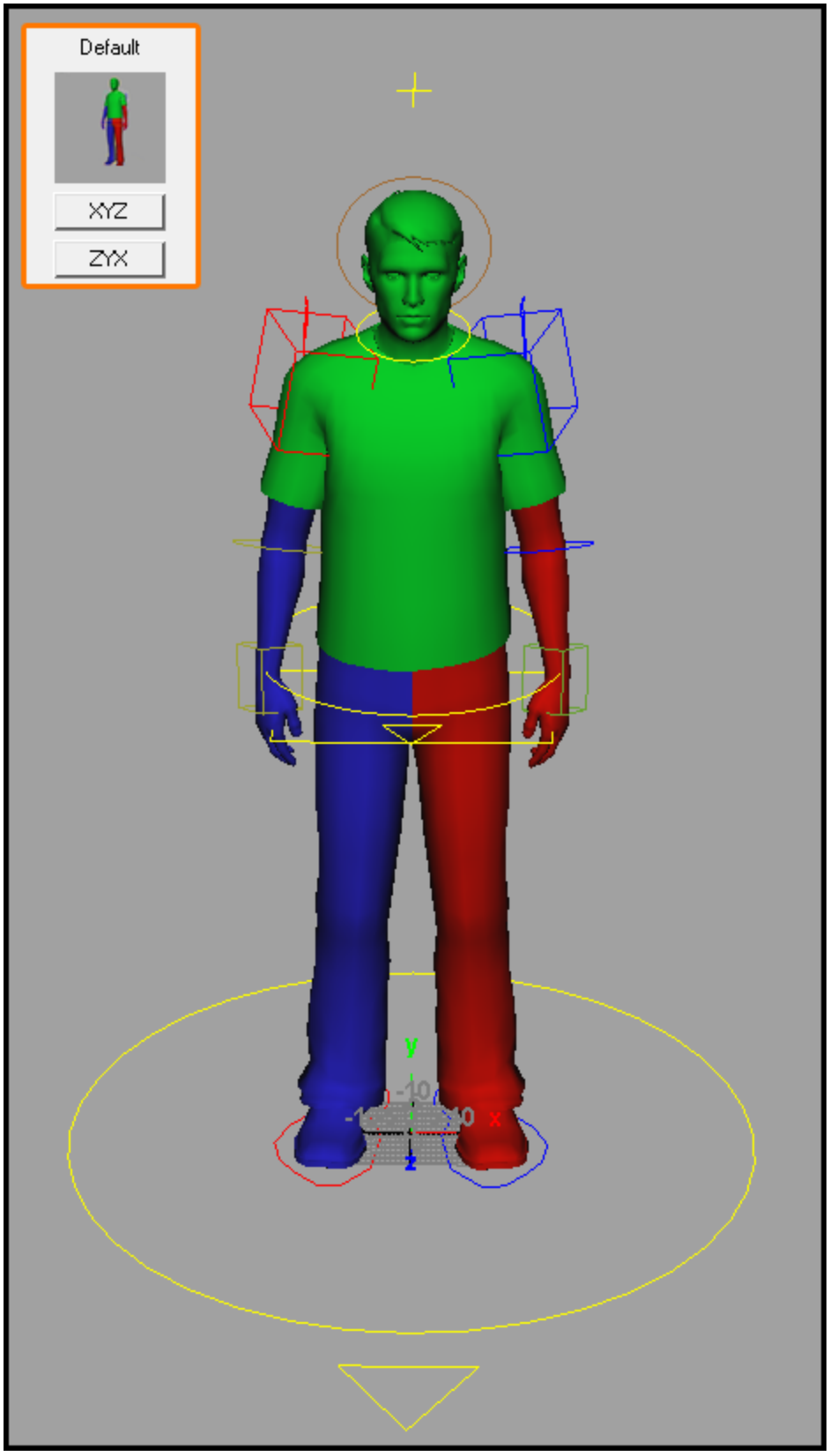


Selecting a Default Pose

Select FEMALE01 POSES or MALE01 POSES, as appropriate. There are a number of default poses for Male or Female rigs.

To begin the animation:

- select an appropriate pose
- to set the avatar from the default pose to the correct start pose, click the XYZ or ZYX buttons



XYZ is the preferred rotation axis.

To avoid gimble lock in the shoulders, the ZYX rotation axis is provided for the key poses.

XYZ is the recommended rotation order and you should normally use it, but you might need to experiment to see which one works

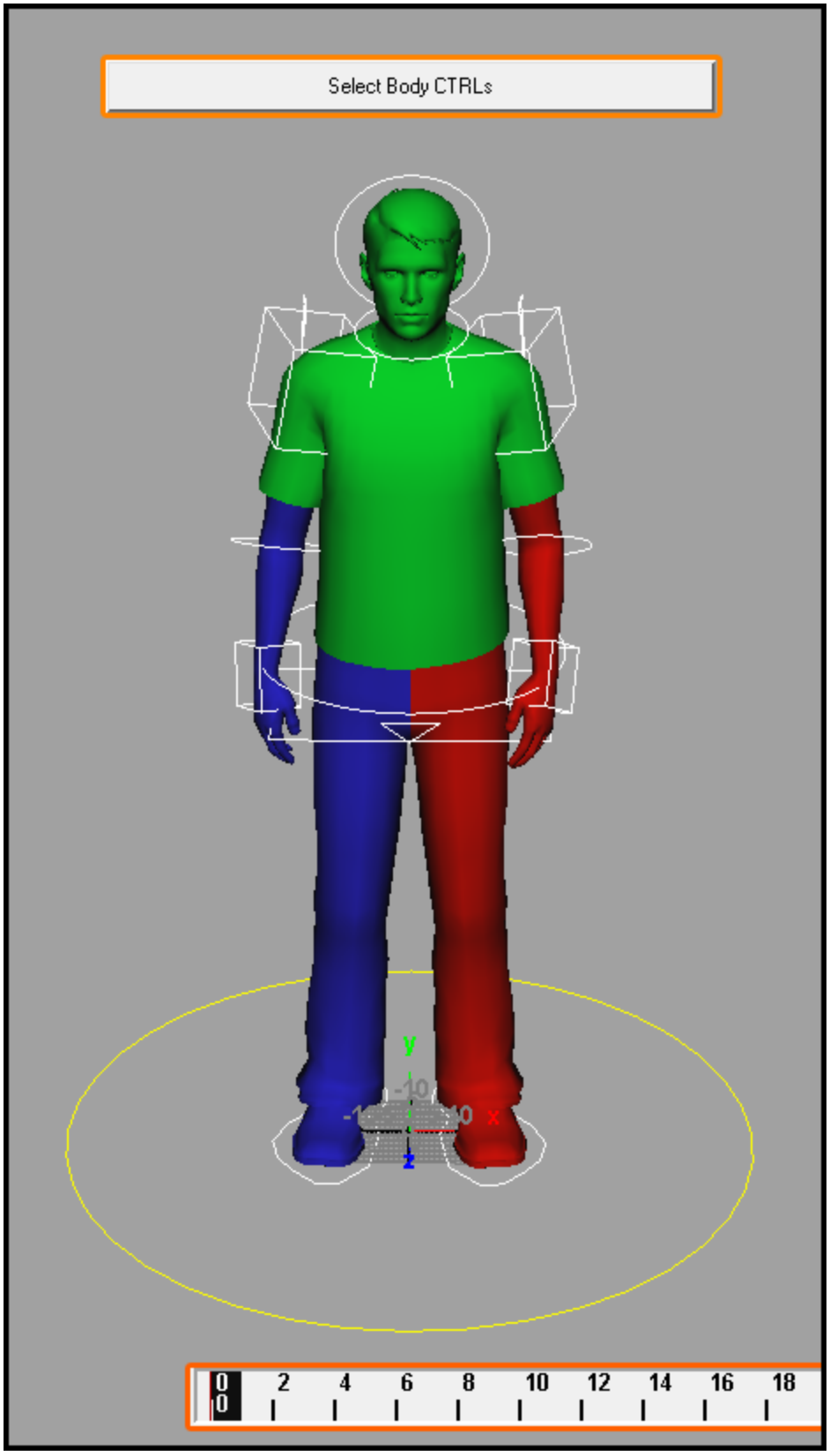
best. Whichever one you choose, you must keep to that rotation axis for the duration of the animation.



If you create an animation with a lot of shoulder movement, block the textures to reveal the skeleton during testing. With the skeleton revealed, play the animation to check that the arms move as expected.

Keying A Pose

Click Select Body CTRLs to select all the body controllers and set keyframes at the start and end frames of the animation:



This setting makes sure that whatever animation occurs between the two, it starts and ends in the correct default pose.

1. When switching from IK to FK or FK IK, click FK IK Snapping for animation blending.
2. To confirm that the process has worked correctly, check the motion by examining the animation curves in the graph editor.
   If there are Euler issues, such as unnecessary spikes, click Auto Euler Filter Baked Curves to fix it.
   If you need to reset the puppet rig and all locators to their default position, click Reset Body CTRLs.
3. Click Setup to transfer the animation from the puppet rig onto the export rig.

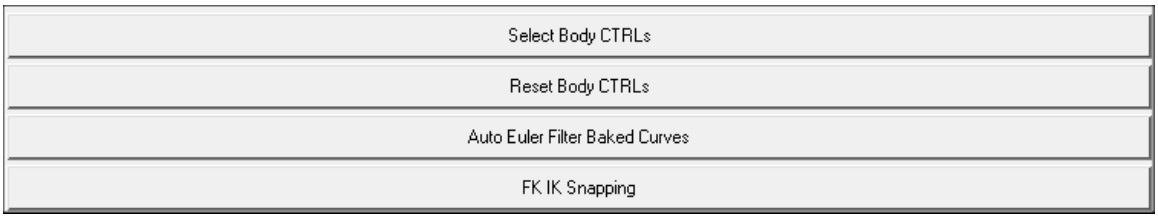
The puppet rig is deleted, leaving you with an export mesh and skeleton with baked animation keys.

You now have an export-ready animation. For more information on exporting, see Exporting Avatar Animation Files.

Avatar Animation Puppet Poser

Avatar Rig Tools

The following tools are available to help you manipulate the avatar rig.



Select Body CTRLs – Select all the controls on the rig except for the root control

Reset Body CTRLs – Set all the controls back to default. This will include translations, rotations and scales as well as IK, FK blending

Auto Euler Filter Baked Curves – Fix problem curves after the animation has been setup for export.

FK IK Snapping – Works on a selected animation control, if the control has both IK/FK (The legs and arms for example) this will toggle between IK and FK at 100% influence.

See also the Avatar Animation Puppet Rig.

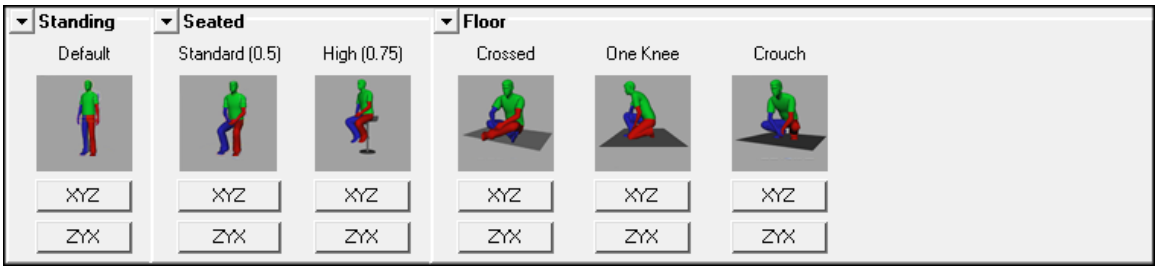
Avatar Poses

The core client poses in PS Home are the Male and Female default poses.

This means all animation must branch from and return to these default poses.

Each icon represents a default pose used in the core client.

The poses come in 2 categories FEMALE01 POSES or MALE01 POSES, the same rules apply to both.



The following table describes each default avatar pose:

Default Pose	Description
Standing	Standing animation can be full body, upper body, right arm/head, and head layers. All standing animation branches from a default pose. Each standing animation should start and end with this pose. Tip: The animation should have time to ease in and out of the default pose, regardless of coded blending values.
Seated	Seated animation can be upper body, right arm/head, and head layers. Full body versions exist but they apply only to core client idles and basic mechanics. All seated animation branches from the default seated pose. Each seated animation should start and end with this pose. Tip: The animation should have time to ease in and out of this default seated pose, taking into account the order in hand position and avoiding any intersection with the lap. There are two variations on the standard seated pose – the high and the low seated pose. Tip: To create an animation for the high or low seated pose, start with the default seated pose (standard height), and then adapt and re-use your upper body animation file for the other poses.

Floor	Floor poses (sitting on the floor) can use the upper body, right arm/head, and head layers. Full body versions exist but they apply only to core client idles and basic mechanics. Any animation that is created for the floor poses must be bespoke. Floor poses are all unique and can only rarely use animation from seated files. There are currently three default poses, each of which has the same set of rules and animation. For floor poses, like other poses, the Male and Female avatars have their own sets. But, unlike other poses, the Male and Female sets of floor poses differ more than usual. As a general rule, each version has its own set of bespoke animations due to its unique posture. Depending on the pose, you can sometimes re-use layered animation from the default seated set, or you can create bespoke files for each pose.
Hand:	The default hand pose is a fist for each hand.

Each pose has the an additional option to set the rotation order, XYZ or ZYX

> XYZ is the preferred rotation axis.



Avatar Rig Export Setup

Click to setup for export.



The setup script will remove all the current rig controls and components, and it will bake all the keys down onto the main export rig. This leaves you with the base avatar skeleton with keys directly on the joints.

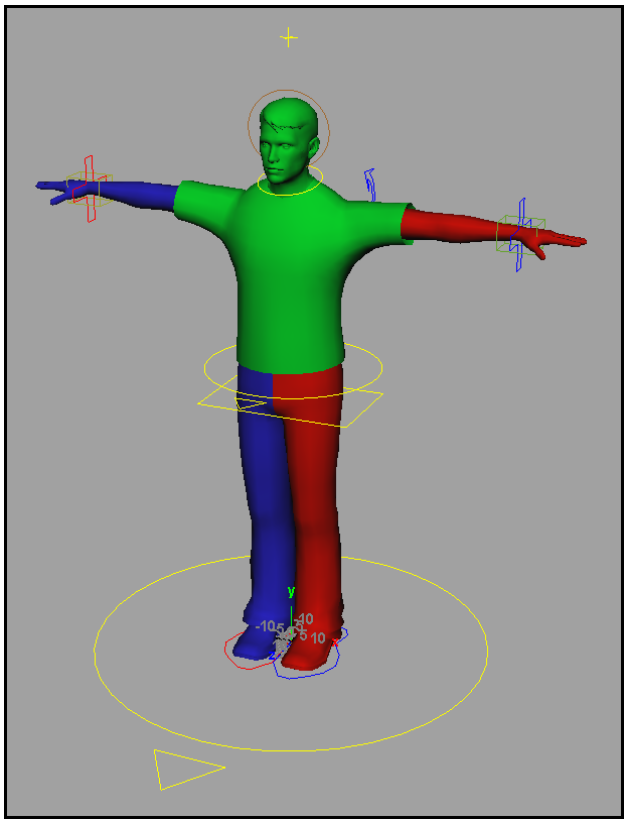
Recommendation: save this file as a new version with a suffix (e.g. _baked) to denote that you have baked the animation. The process cannot be undone and the removal of the rig controls can make it hard to edit the animations.

Avatar Animation Puppet Rig

Avatar Animation Rig Controls

The puppet rig is the rig the animators uses to create avatar animations. It has been developed with a comprehensive set of helpers and controllers that allow the artist to quickly and painlessly create PS Home avatar animation.

To open the puppet rig, select File > New Home Avatar Animation.



LeftHandCTRL	
Rotate X	0
Rotate Y	0
Rotate Z	0
Curl Index	0
Curl Middle	0
Curl Ring	0
Curl Small	0
Knuck Index	0
Knuck Middx	0
Knuck Ring	0
Knuck Small	0
Spread Index	0
Spread Middle	0
Spread Ring	0
Spread Small	0
Thumb Curl	0
Thumb Knuck	0
Thumb 1X	0
Thumb 1Y	0
Thumb 1Z	0
Parent World X	0
Parent World Y	0
Parent World Z	0
Parent	0

Control List:

homeCTRL

- Mesh Colour

rootCTRL

- Right Arm IK to FK
- Right Leg IK to FK
- Left Arm IK to FK
- Left Leg IK to FK

LeftHandCTRL and RightHandCTRL

- Multiple Hand Controls - Finger Curls and Thumb adjustments.

LeftFootCTRL and RightFootCTRL

- Toe Wiggle

LeftHand_CTRLik and RightHand_CTRLik

- Arm Twist

LeftFootCTRLik and RightFootCTRLik

- Multiple Leg and Foot controls

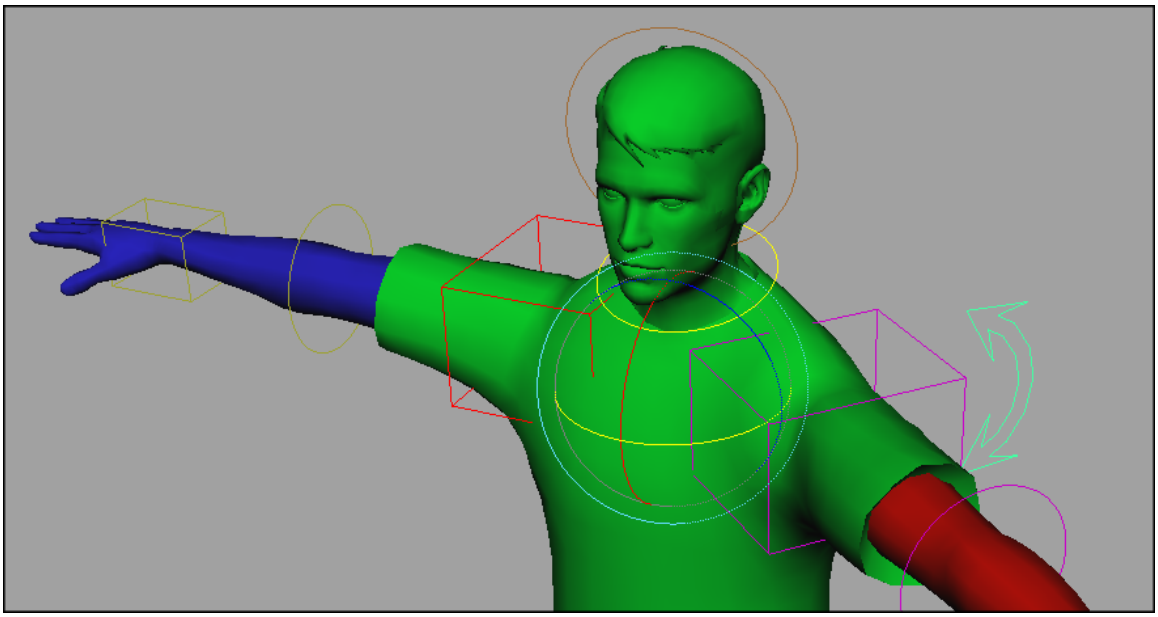
Edit the attributes (shown above) via the channel box. The image above shows the hands attributes.

Channel Box/Layer Editor

With the Channel Box/Layer Editor you can move, rotate and scale each body part by specific numeric amounts.
To open the Channel Box/Layer Editor, select Display > UI Elements > Channel Box/Layer Editor.

When you select a body part Control Curve on the character rig (or in the Maya Outliner), the Channel Box/Layer Editor will update - showing you the attributes that you can move, rotate, and adjust. It will also show you additional attributes (e.g. rolling the ball of the foot) for that particular body part.

Below is an example of changing the Z-axis rotation on the left shoulder:



Avatar Animation Male and Female Rigs

Both Male and Female versions of the puppet rig are included in the HDK. The Male rig is slightly taller than the female rig, but animation from both avatars is very similar and most custom animations should be transferable with some adjustments.

If you transfer animations from one gender to another, when adjusting keep the following in mind:

- the shoulders and clavicles have differing proportions (shoulders will be wider apart for males)
- translate and rotate (x, y, z) on the root control (root will be higher for males)
- translate and rotate (x, y, z) on the feet controls (feet will be wider apart for males)

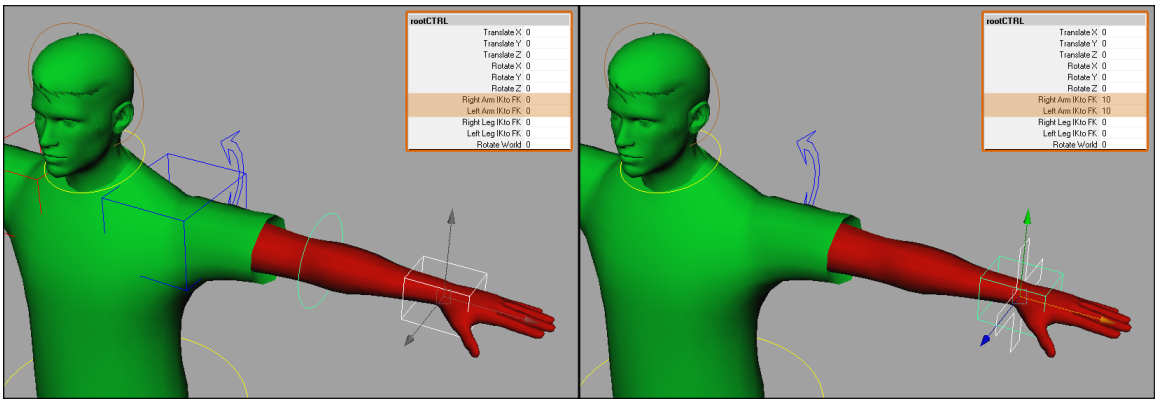
Avatar Animation Leg and Arm FK / IK switching

There are FK/IK (forwards kinematics / inverse kinematics) switching attributes on the rootCTRL. A value of 0 to 10, where 0 is FK and 10 is IK. This is a key-able attribute. Keep in mind that its possible to set the value between 0 - 10 to create a blend

between the two modes.

We have an FK/IK snapping tool available within the Puppet Poser GUI described in the Puppet Poser section.

The images below show the controller change when you adjust the blend.



Avatar Animation Mesh Controls

Fat blend shape control

In order to check that your animation works on all body shapes, you will find a BODY_FAT locator just above the rig's head, which has an attribute from -1 to 1.

Color-coded Limbs

The puppet rig has color-coded limbs. They are a handy visual guide, making it easy for the animator to read the body parts. The homeCTRL holds the attribute Mesh Colour.

Optimizing Animation to Reduce Memory Usage

You can use the Animation Compression feature to create an animation compression preset file. This type of file can control the ways in which animations are compressed in order to save memory at run-time. Compression values are applied on all animation channels (translation and rotation).

By using the Animation Options dialog, artists can define different compression values on different parts of skeleton. For example, when compressing a dancing animation, the artist may only want to compress head and arm animations. The quality and performance of the leg animation would not be changed.

Bear in mind that compression results are always at the cost of the animation quality.

Animation Layers

To add flexibility to the animation system, you can use the following animation layers:

Animation Layer	Description
Full body	Offers the least flexibility and is normally used for transitions into looped animation, as well as locomotion, basic mechanics, default poses and some standing/seated idles. It also serves as base animation, allowing layered animation to be played on top. You can split full body animation into upper body and lower body layers. For example, in the dance animation, the lead-in animation file is split, and the looped dance file is a full body animation. However, this division can cause blending issues because the upper body is blended but the lower body is not, to avoid unwanted foot slide.
Upper body	Probably the most flexible and commonly used layer. In some cases you can re-use the same upper body animation file across various standing, seated and sitting on floor poses. A large number of Emotes use upper body layers.
Right Arm and Head	Very limited in terms of what you can achieve, as it depends on the body position you are layering the animation over. The effect can also look a little wooden. The main advantage is the small size of the exported file.
Head	This layer also has limited uses. Its use is based on head direction and simple agree/disagree animation.

Anything other than Full Body is layered over a breathe loop.

Layer Controls

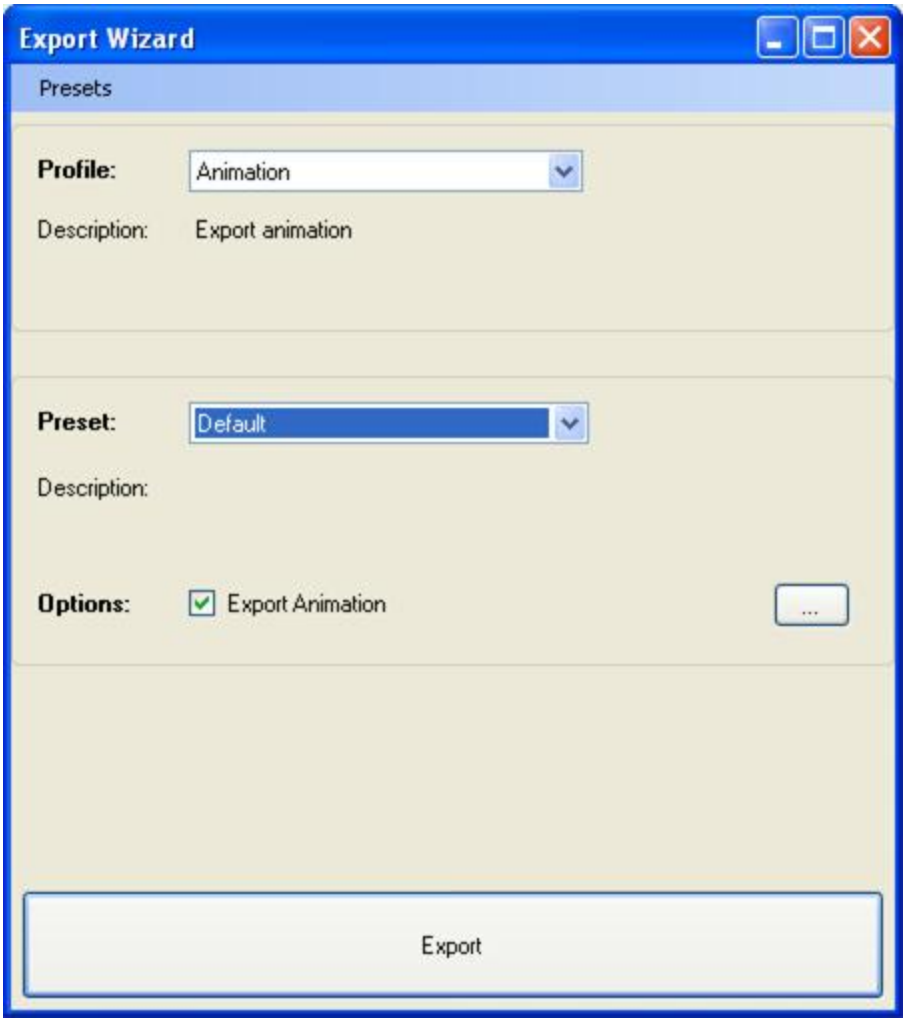
The following table lists the puppet controls that form the layers:

Layer	Puppet Controls
Full Body layer	All puppet controls
Upper Body layer	• SpineCTRL • Spine3CTRL • LeftArmCTRL • LeftShoulderCTRL • LeftForeArmCTRL • LeftHandCTRL • LeftHand_CTRLik • RightArmCTRL • RightShoulderCTRL • RightForeArmCTRL • RightHandCTRL • RightHand_CTRLik • NeckCTRL • HeadCTRL
Right Arm and Head layer	• RightArmCTRL • RightShoulderCTRL • RightForeArmCTRL • RightHandCTRL • RightHand_CTRLik • NeckCTRL • HeadCTRL
Head layer	• NeckCTRL • HeadCTRL

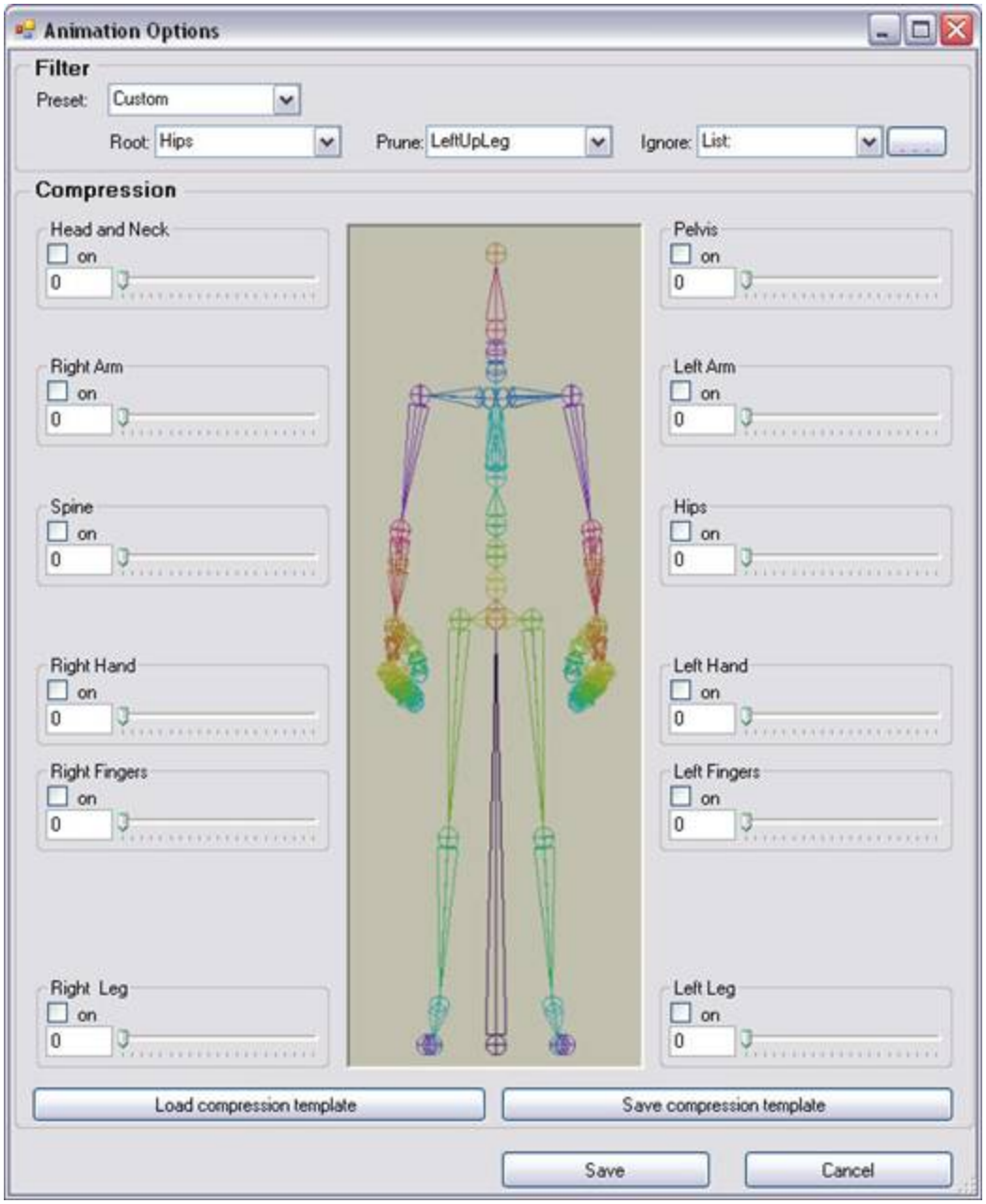
Filtering and Compressing the Animation

To filter the animation data and compress the animation:

1. In the Export Wizard dialog, select Animation from the Profile drop-down list.
2. Select the Export Animation box:

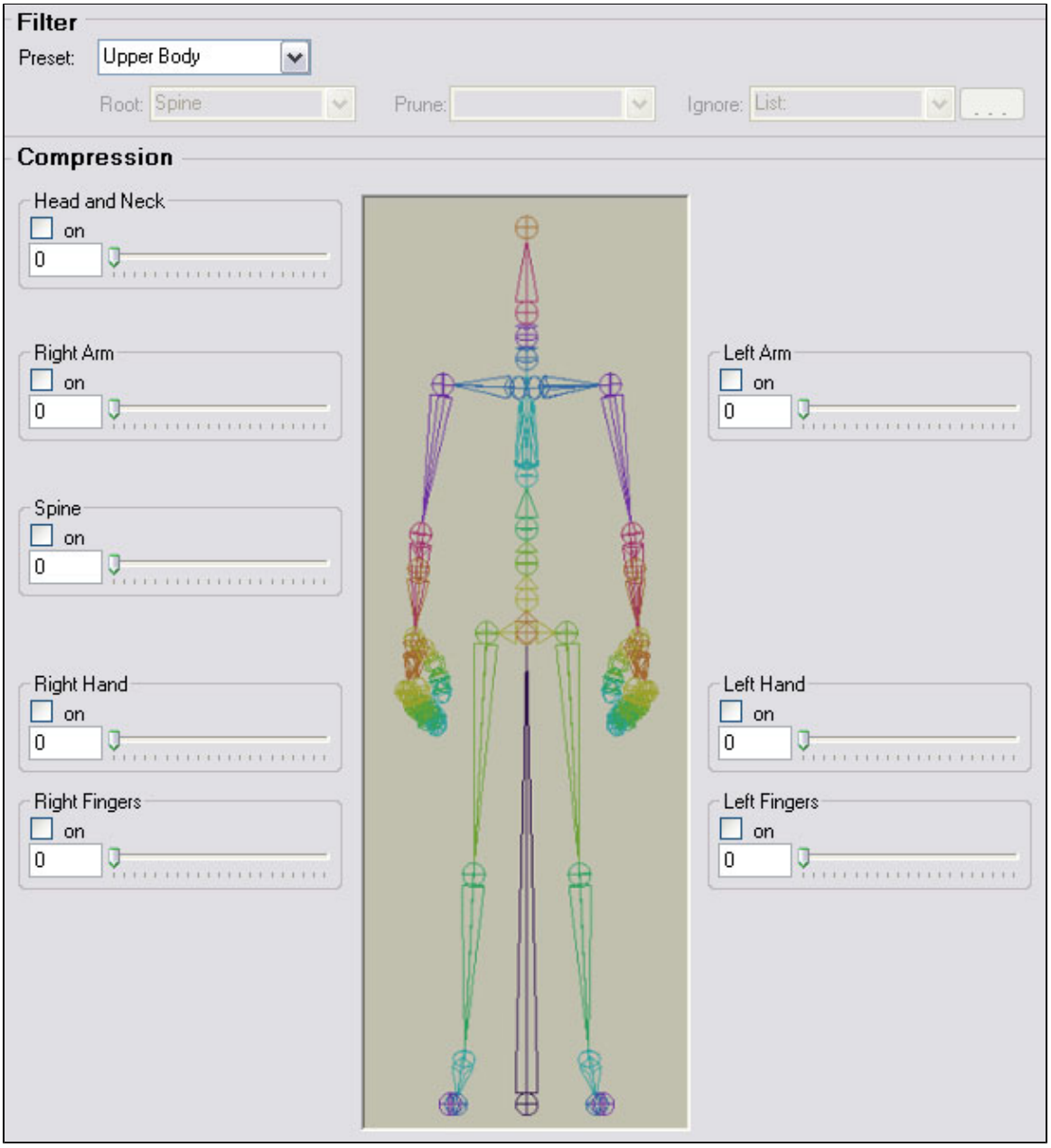


3. Click the More button to display the Animation Options dialog:



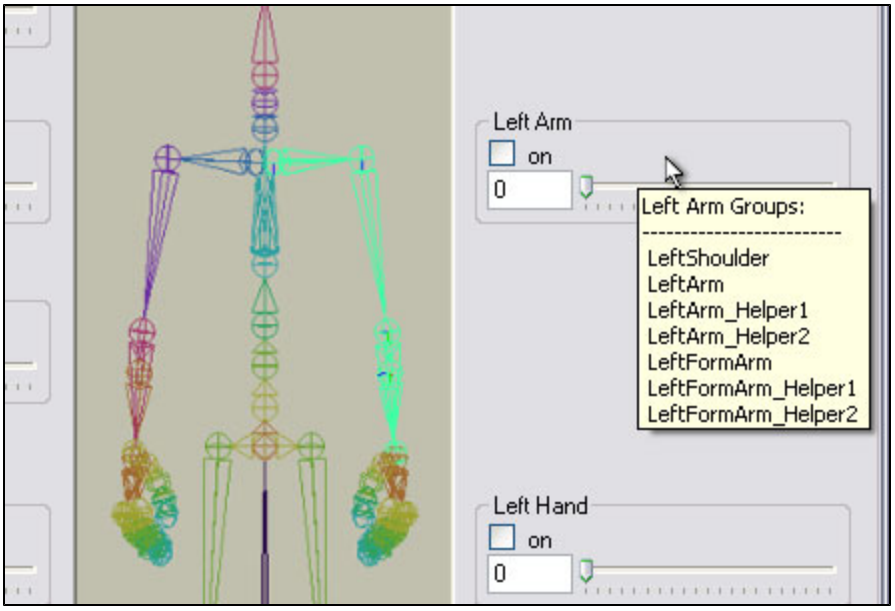
The Animation Options dialog enables you to filter the animation data and compress it on export.

4. To filter which part of the skeleton animation to export, select the part from the Preset drop-down list in the Filter panel.
The following image shows the Upper Body animation filtered for export:

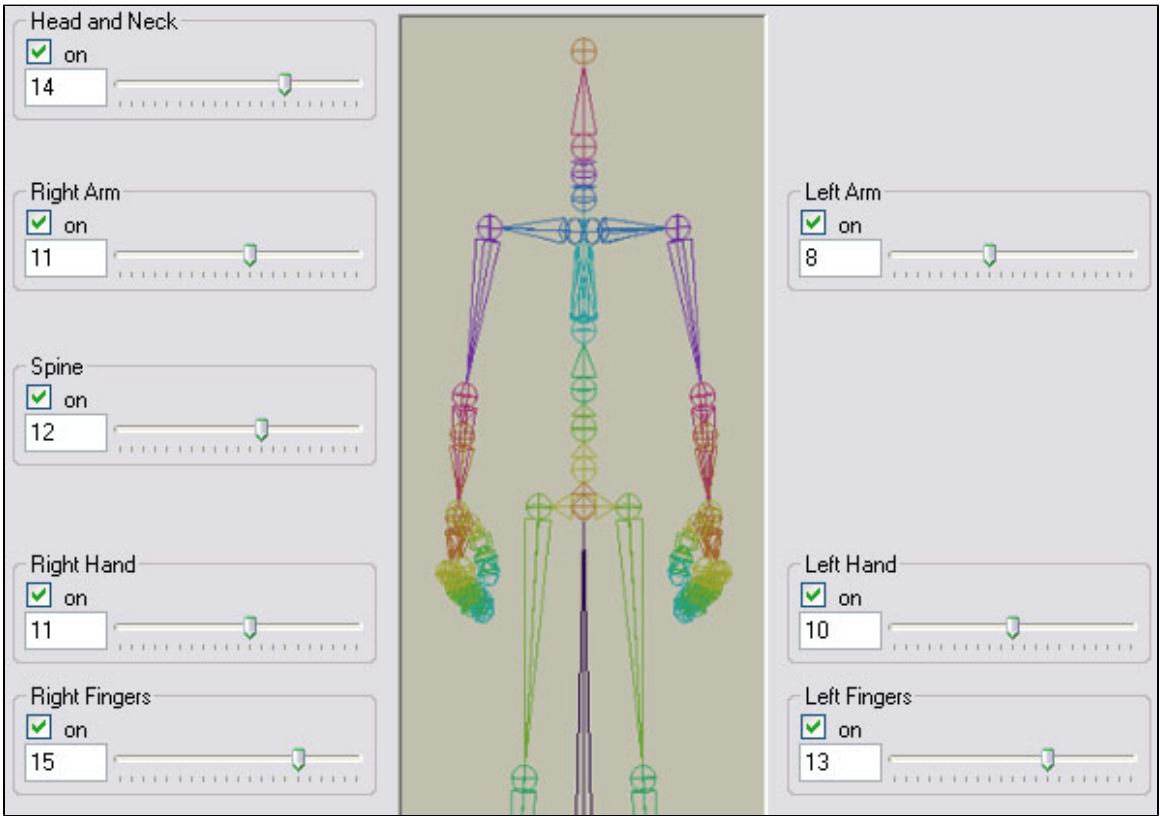


Notice how the filtering is also applied in the Compression panel, to show which parts of the rig or 'groups' will be exported with the current preset.

Hover the cursor over a group to highlight the bones contained in that group on the skeleton image, and to display a pop-up list of their names.



The Compression panel specifies the compression values on each group of the skeleton animation that will be exported. Export of the data takes into account all compression values applied to the groups:



5. To apply compression to a group, select the on box, then either select the required compression value on the slider (0 = no compression, 20 = a lot of compression), or enter your own value (which can be greater than 20, if required). Each group can have its own compression value. This is very useful for achieving the optimum compromise between quality and file size on any given animation.

> The compression values are not linear and each animation is unique, so you must test the effect of compression values on your animation by exporting and launching the animation in PS Home.

6. Do one of the following:

   If you want to save your filter and compression settings to use on other Maya animation files, click Save and Load Compression Template.
   If you want to save the settings as a file, click Save compression template. The setting are saved to an .acp file, in the \dev\config\AnimCompressPresetTemplates folder.



7. Click Save to save the completed setup with the current Maya file.

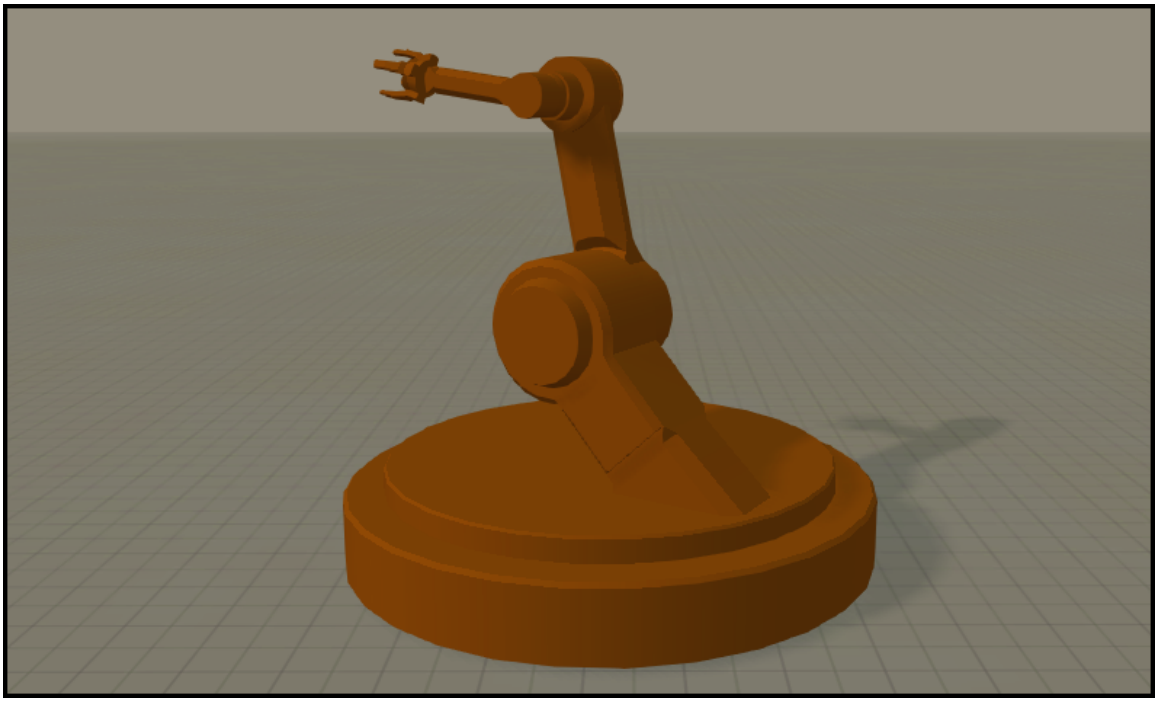
You can now export the animation data. The export creates an .ani file in the build\animation folder.

Joint Animation and Rigid Body Animation

> For general information about animation and effects not pertaining to avatars, see Other Animation and Animation Effects.

Joint Animations

Joint animations are any animation that contains joints but is not based on the standard avatar skeleton and will not be played on an avatar.



Joint Animation supports:

- Joints and smooth binding with up to 4 influences
- Joints with their Translations and Rotations keyed
- Scales can be used (but non-uniform scales on joints will result in the whole mesh skewing)

  > Note: with uniform scales the child joint will not inherit the scale

- A limit of 128 joints per Joint Animation

⚠ Blend Shapes are not supported.

Joint Animation Compression

As with Avatar Animation, compression can be used to reduce file sizes for joint animation.

For more information see Animation Compression Tools.

✔ Compressing your animation may result in a reduction in visual quality.

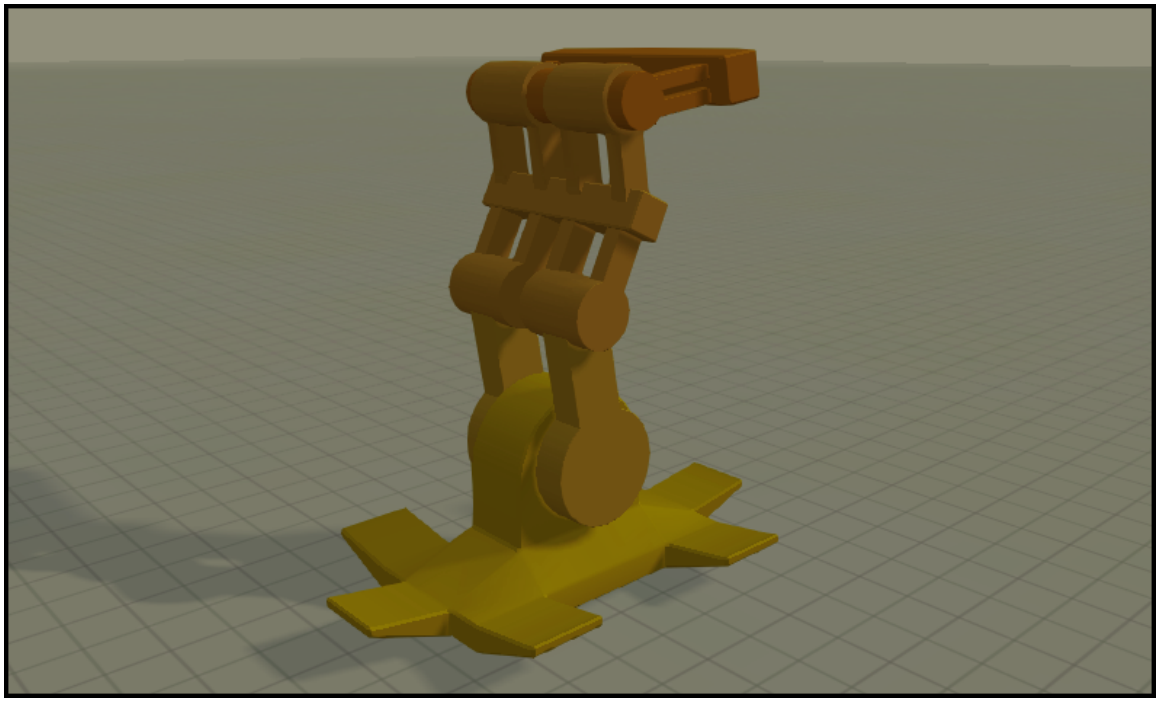
Rigid Body Animation

Rigid Body animation are very similar to Joint animation - except that they do not use Joints.

Meshes with their Translations and Rotations and Scales keyed directly (No Joints) should be exported as Animated Models, not as Joint Animations.

The image below shows the robot leg broken up into 4 meshes each a different color.

They are then parented into a hierarchy in Maya and keys are applied directly to the meshes.



ℹ Note: No compression is available for this type of animation.

Character Tools

The character tools are available in the HDK via the Maya Home menu or Home_Char Shelf.

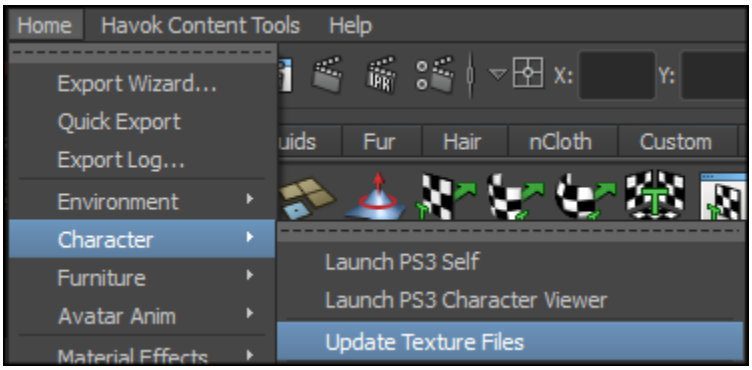
General Tools

Vertex Normal Tool

This tool is used to align the normals of the border edges of character components. This is most often used where skin meets skin. If the vertices are not aligned correctly a visible seam will appear in game.

For details see the Vertex Normal Tool.

Update Textures Files



Updated the textures associated with the shaders assigned to the character component. This is a simple refresh used when you are changing or editing textures and Maya has not reflected the changes.

Weighting

Weighting Tool

The weighting tool consists of a number of useful tools to help you in weighting your meshes.

For details see the Weighting Tool.

Initial Skin Weighting

Initial Skin Weighting will create basic weighting for your mesh. This will often need to be edited and will not be perfect. The more unusual the character component design the less successful the initial weighting will be.

You will have to modify the weighting to finalize it, this tool just gives you an easier starting point.

Copy Weights and Paint Skin Weights – Default Maya Tool

These tools are Maya standard tools, they have been added to the character tools menu for convenience. More information can be found within the Maya documentation .

Blend Shapes

Create Blend Shapes (LOD1) (LOD2 + LOD3)

This will create initial Fat and Thin blend shapes for the character component. These meshes can be adjusted after they are created.

> ⚠ Note: They topology cannot be changed or the blend shape will no longer function. Adjustment of the existing mesh is all that is possible.

For details see Creating Fat and Thin Targets.

Replace Blend Shapes: Note this is quite a complex process and will requite knowledge of Maya blend shapes.

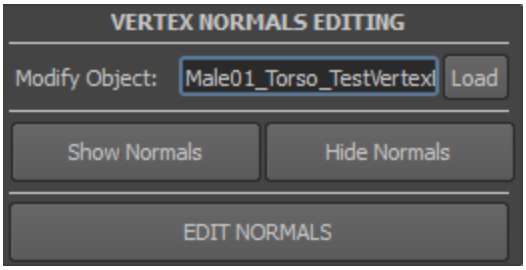
For more information on the steps to follow, see Replace Fat and Thin Blend Shapes.

Vertex Normal Tool

Vertex Normal Tool

The tool can be found under the Home Character menu.

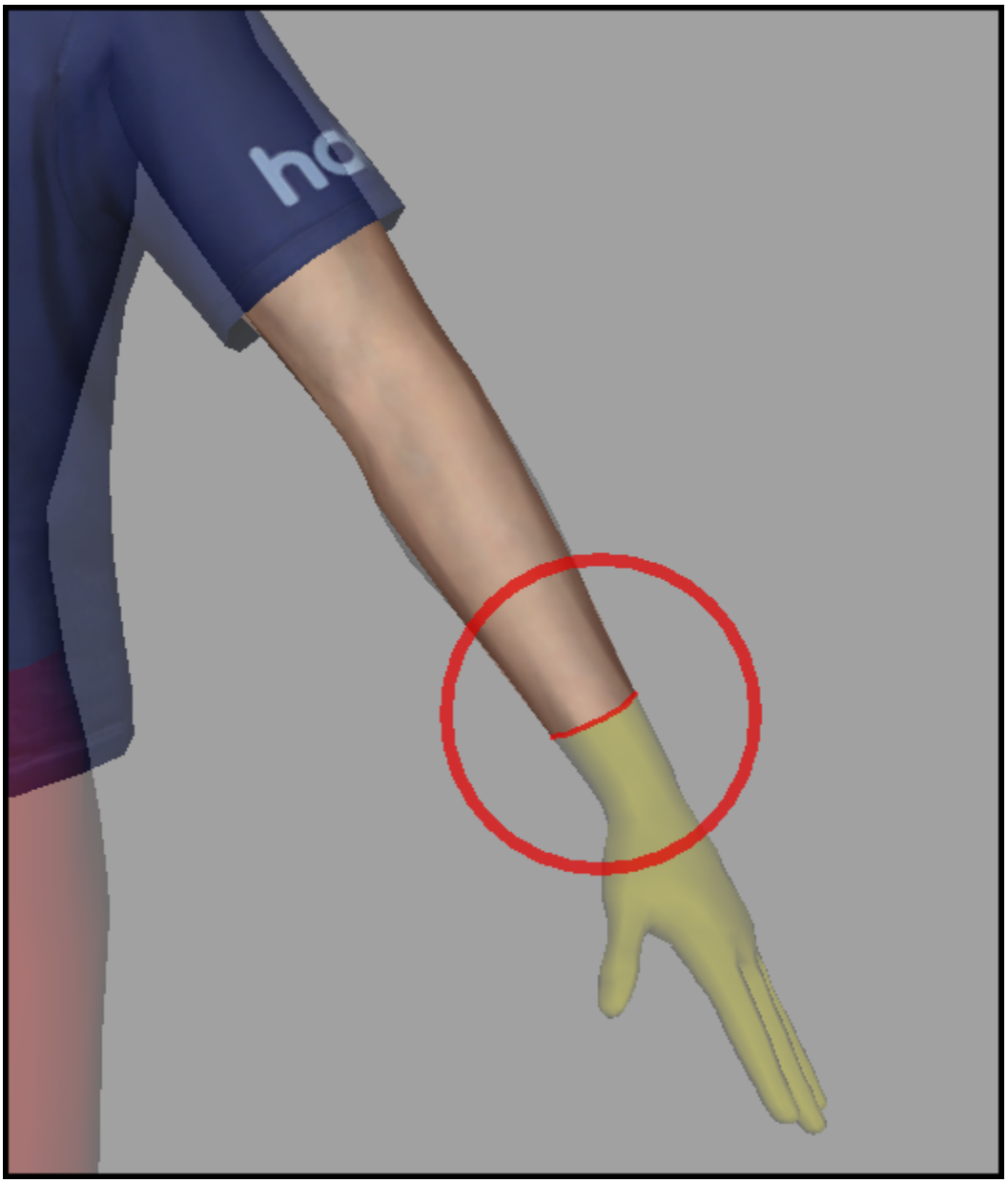
Home > Characters > Edit Vertex Normals



The image below shows a common use for the tool - the border between the skin on the Torso character component and the default hand of the avatar.

Both of these sections will have the skin shader assigned and any discrepancies in the normals can result in a hard visible line.

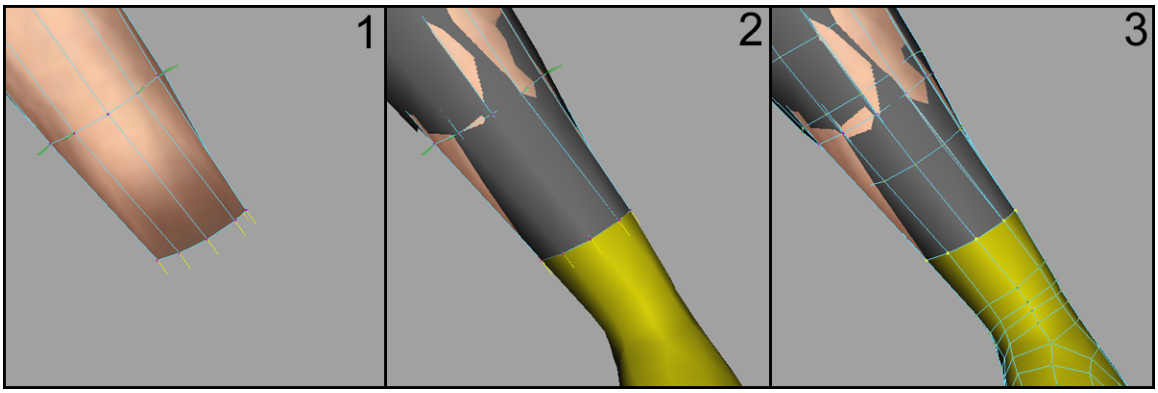
The skin reference mesh in Maya can be used to set the vertices. This is found under the reference group skin (LOD1, LOD2, LOD3). They are templated by default so you will need to untemplate them before use.



Using The Tool

1. Select the character component mesh and then click Load to set the modify object (after which you can also click on Show Normals to display them in the Maya view port)

2. Select the reference mesh and then select a vert that you want to copy from.
3. Now select the source mesh and select the vert you want to copy.
4. Go to Edit Normals and the normal information will be copied from the reference to the character component.

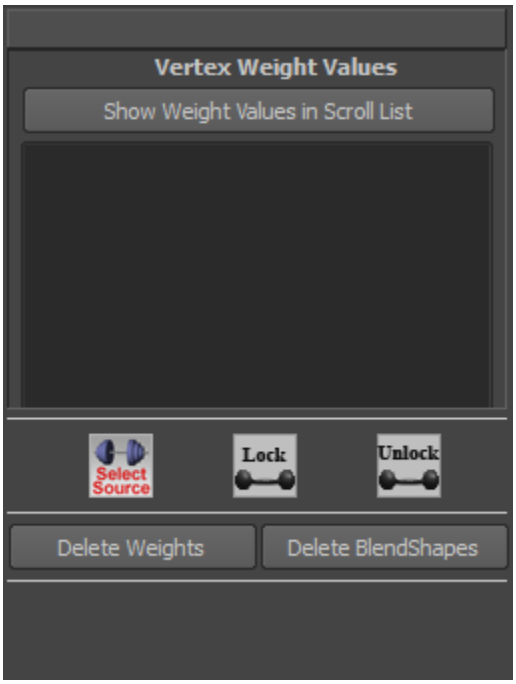


You can repeat this for each vert.

Weighting Tool

Weighting Tool

The weighting tool has a number of useful functions when generating weighing for a mesh.



Lock / Unlock

Lock / unlock the weights of the whole mesh.

Delete Weights

Delete all weighting information for the mesh.

Delete Blend Shapes

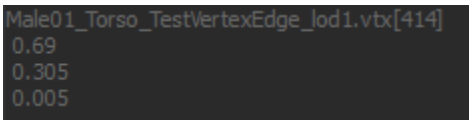
Delete all blend shapes from the mesh.

Note: If you have Fat and Thin blend shapes generated, these will no longer work and will either have to be reconnected or regenerated.

Show Weight Values in Scroll List

Show the weighting values for the currently selected vert.

See example below:



Select Source

Copy weights from one vert to another:

- Select a vert on the weighted mesh - Select Source
- The button will change to - Select Target
- Select a new vert and it will transfer the weighting info over.

Replace Fat and Thin Blend Shapes

This process describes how to override or replace the automatically generated fat/thin shapes.

This is a process that should not be attempted unless you are sure you have knowledge of Maya blend shapes.

Initial Mesh Setup

Step 1

Create the basic meshes LOD1, LOD2, LOD3 as normal.

Step 2

Create the initial skin weighting (if you have already applied the weighting, then you go to step 3).

Create LOD 1 Blend Shapes

Step 3

Duplicate LOD1 and move it to the fat and thin locations. The translations for them are:

Fat target

- x = 1
- y = 0
- z = -0.5

Thin target

- x = -1
- y = 0
- z = -0.5

Step 4

Rename the Fat and Thin targets to the following:

- Fat = FatShape
- Thin = ThinShape

Create the Blend Shape Deformer

Step 5

You will now need to create the blend shape deformer on for the LOD1 in the Animation Menu Set > Create Deformer > Create Belnd Shape. Go to the Options box.

The menu will let enter the name of the blend shape. The blend shape node will need to be called lod1BlendShape

Select the Fat and Thin target meshes then select the LOD1, click create and the new blend shape node will appear on the LOD1.

> ⚠ Note: You can check the connection is working by moving some of the verts on the fat or thin target mesh and then adjust the blend shape node and see the result.

Step 6

You now need to rename the Fat and Thin target meshes to the same as the LOD1 mesh but with _thin and _fat for the coresponding target.

See example below where blendshapetest is what you called your file and also Torso will be different.

- male01_torso_blendshapetest_lod1_thin
- male01_torso_blendshapetest_lod1_fat

Create LOD Blend Shapes

Step 7

You can now use the tools as usual to copy blend shapes to lod 2 and 3.

Step 8

The belend shape target meshes cannot be exactly the same as the main LOD you will need to move one vert slightly on the Fat and Thin target for all the LODs, the Fat and Thin targets should alos not be the same.

> ⚠ Note: The naming is important here so take extra care, if the export crashes at about 30% then its most likely because 2 of your targets have exatly the same geometry.

Character Component Reference

Head

All of the following Head components use the head joints. They do not have Fat and Thin blend shapes so when creating them they may need to be larger or have more volume to account for this.

This can be a particular issue for Facial Hair components as the lower part of the head and face changes with the Fat and Thin blend shapes, but the shape stays a fixed size.

- Facial Hair
- Hairstyle
- Headgear
- Headdress
- Full Headdress

> ℹ See also Creating Facial Components

The following components follow the same rules as other Head components, they are in essence the same type of component positioned differently on the Avatar. The choice of which type to make will determine where in the Home menus they appear.

- Jewellery (Left Ear) (Right Ear)
- Spectacles
- HeadPhones

Body

These are the core body component types. They all have Fat and Thin blend shapes.

- Hands
- Torso
- Legs
- Feet

The composite character components consist of a selection of adjacent core types (above) combined. The budgets are the sum of the core types.

The seams between the listed components are also no longer an issue and can be modeled as you wish. For example (Leg+Feet) will cover the ankle of the Avatar and so obscure what would normal be a seam.

- Torso + Legs
- Legs + Feet
- Torso + Legs + Feet

FullBodySuit – Replaces Whole Avatar

The FullBodySuit replaces the whole avatar mesh. It can be anything in any style but must still adhere to the avatar's skeleton and so also the avatar's proportions.

By replacing all elements of the avatar the budget for all components is available to you.

- FullBodySuit

Character Component Facial Hair



Facial Hair components cover the lower portion of the Male avatar's head.

Female avatars have no such component.

Memory Limits (KB)	Vram	Host	LOD1	LOD2	LOD3	Main
Male	128		48	18	5	2.5
Female	N/A		N/A	N/A	N/A	N/A

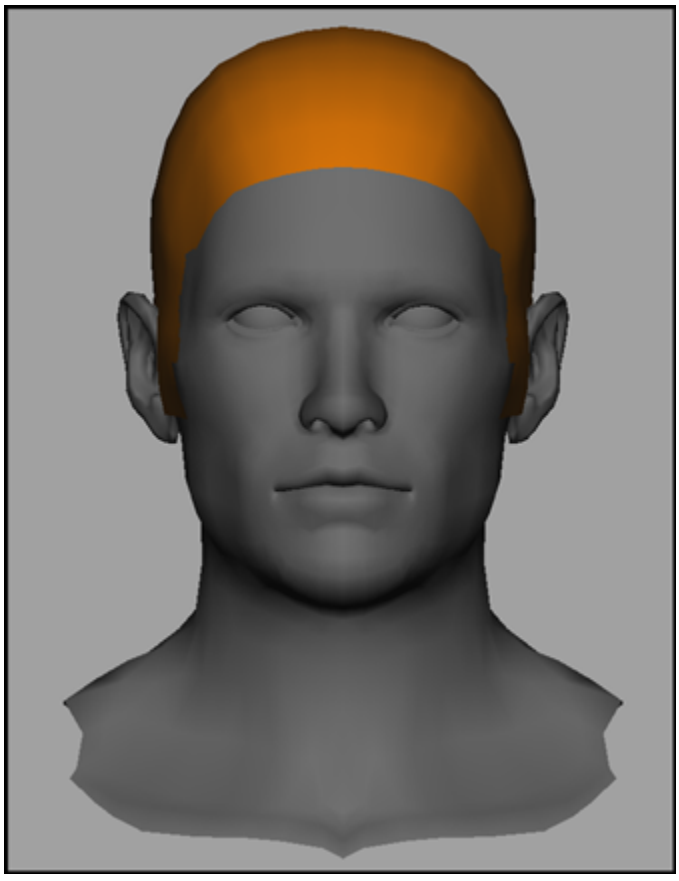
	Male	LOD1	LOD2	LOD3	Female	LOD1	LOD2	LOD3
Polygon Count (recommended)		700	350	150		N/A	N/A	N/A
Mesh Limits		5	3	3		N/A	N/A	N/A
Fat / Thin Blend Shapes		N	N	N		N/A	N/A	N/A

Shaders	Default, Emissive, Blend, Glass, Hair, Skin
Shader FX	Texture Scrolling, Animated Textures
Weighting	Smooth Skin 4 influences.
Joints	Head Joints

Facial Hair does not have blend shapes. As such the Fat and Thin head may clip through close fitting components. Facial Hair should be made with sufficient volume to prevent clipping by the Fat and Thin shapes.

> ⚠ Creating facial hair that sits close to the face is not advisable, as this leads to clipping.

Character Component Hair



Hair (Hairstyle) components cover top and sides of the head. Hair may also extend to other areas around the head or down the back (note that long hair will often clip into the avatar's back).

> ⚠ Headgear will replace the hair component when selected by the player.

Memory Limits (KB)	Vram	Host	LOD1	LOD2	LOD3	Main
Male	1708		64	35	22	2.5
Female	855		64	35	22	2.5

	Male	LOD1	LOD2	LOD3	Female	LOD1	LOD2	LOD3
Polygon Count (recommended)		1400	810	426		1400	810	426
Mesh Limits		5	3	3		5	3	3
Fat / Thin Blend Shapes		Y	Y	Y		Y	Y	Y

Shaders	Default, Emissive, Blend, Glass, Hair, Skin
Shader FX	Texture Scrolling, Animated Textures
Weighting	Smooth Skin 4 influences.
Joints	Head Joints

Long hair can have clipping issues when the head turns. While it is possible to create hair that lays over the shoulders it is not advised due to clipping issues and the lack of joints to weight to.

Character Component Headgear



Headgear components cover the upper portion of the head. They generally take the from of Hats

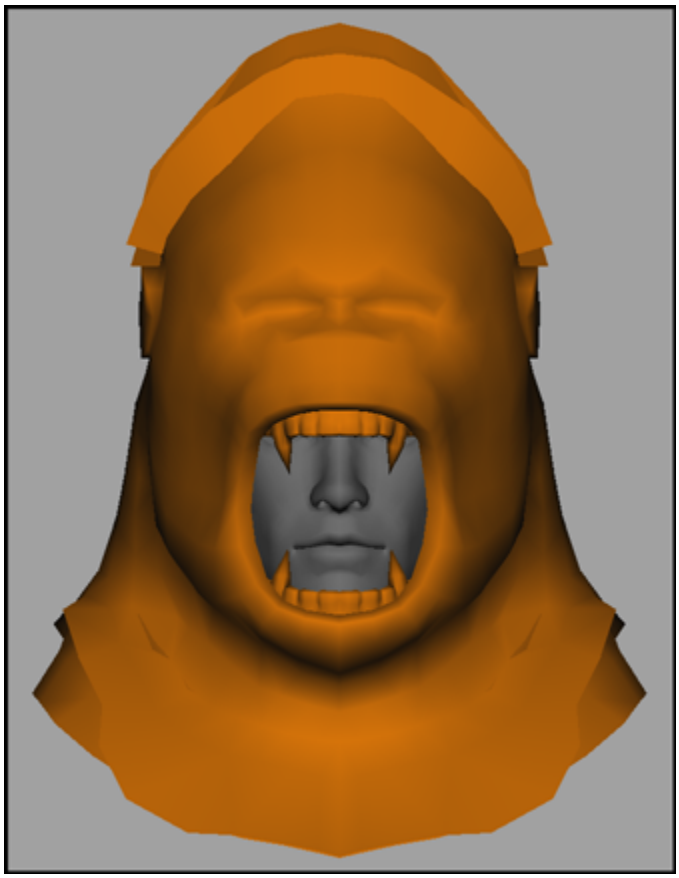
Memory Limits (KB)	Vram	Host	LOD1	LOD2	LOD3	Main
Male	1708		64	35	22	2.5
Female	855		64	35	22	2.5

	Male	LOD1	LOD2	LOD3	Female	LOD1	LOD2	LOD3
Polygon Count (recommended)		1400	810	426		1400	810	426
Mesh Limits		5	3	3		5	3	3
Fat / Thin Blend Shapes		N	N	N		N	N	N

Shaders	Default, Emissive, Blend, Glass, Hair, Skin
Shader FX	Texture Scrolling, Animated Textures
Weighting	Smooth Skin 4 influences.
Joints	Head Joints

Headgear components do not have Fat and Thin blend shapes.

Character Component Headdress



Headdress components cover areas of the head, but not all of it. The avatar's Head mesh is still shown so that areas are visible beneath or through the headdress.

Memory Limits (KB)	Vram	Host	LOD1	LOD2	LOD3	Main
Male	1908		102	55	33	13
Female	926		102	55	33	11

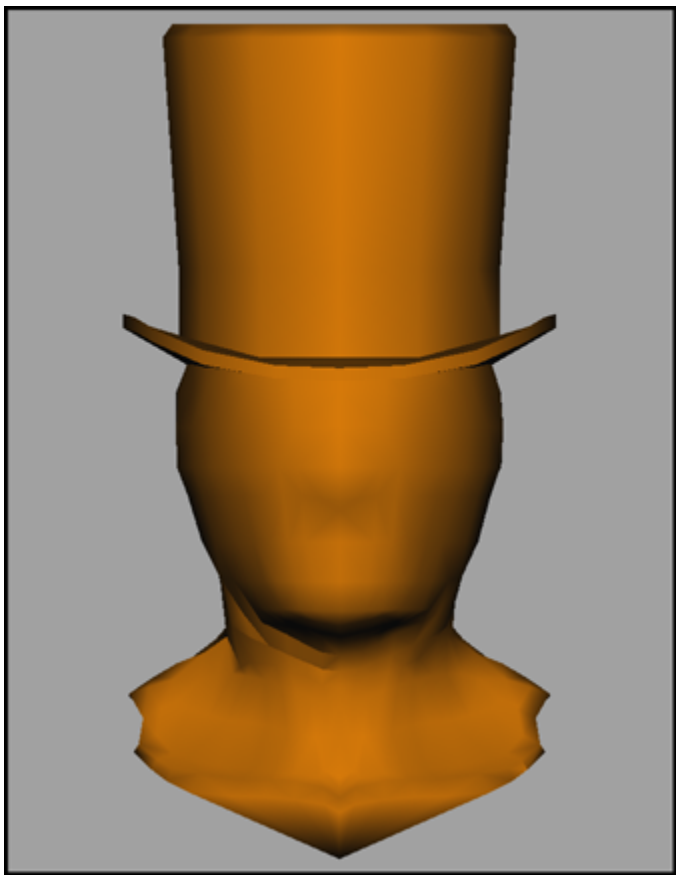
	Male	LOD1	LOD2	LOD3	Female	LOD1	LOD2	LOD3
Polygon Count (recommended)		3200	1810	746		3200	1810	746
Mesh Limits		25	15	15		25	15	15
Fat / Thin Blend Shapes		N	N	N		N	N	N

Shaders	Default, Emissive, Blend, Glass, Hair, Skin
Shader FX	Texture Scrolling, Animated Textures
Weighting	Smooth Skin 4 influences.
Joints	Head Joints

Boundariess between the Headdress and the other components should be considered. It is often best to create a generous neck area to guarantee it overlaps.

Budgets are a combination of Hair, FacialHair, HeadPhones, JewelLeftEar, JewelRightEar.

Character Component Full Headdress



Full headdress components cover the entire head of the Avatar. The Avatars head mesh is completely removed so nothing shows through.

Memory Limits (KB)	Vram	Host	LOD1	LOD2	LOD3	Main
Male	2075		506	146	57	21
Female	1102		506	146	57	19

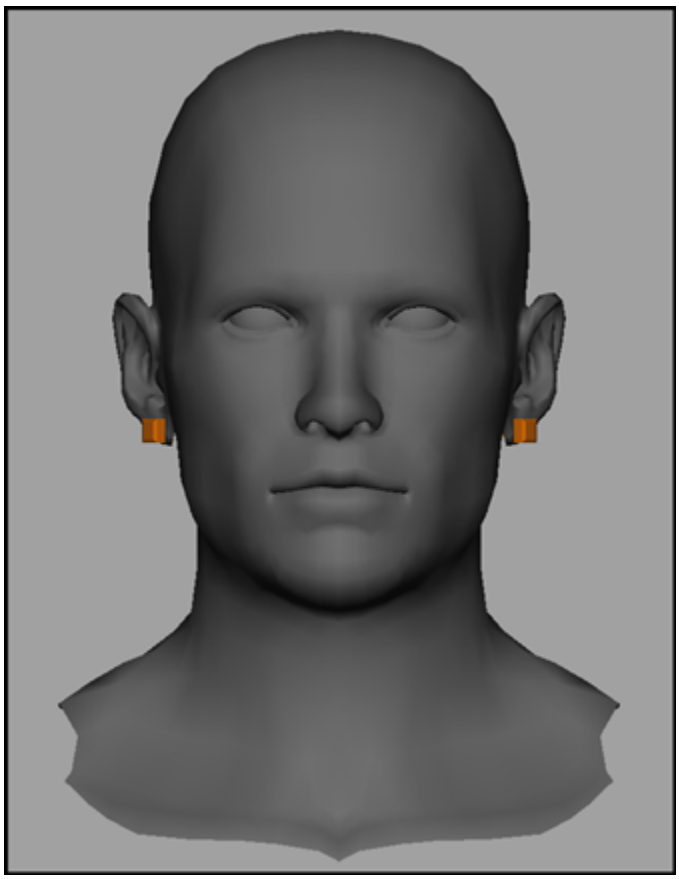
	Male	LOD1	LOD2	LOD3	Female	LOD1	LOD2	LOD3
Polygon Count (recommended)		3800	2110	896		3800	2110	896
Mesh Limits		38	21	21		38	21	21
Fat / Thin Blend Shapes		Y	Y	Y		Y	Y	Y

Shaders	Default, Emissive, Blend, Glass, Hair, Skin
Shader FX	Texture Scrolling, Animated Textures
Weighting	Smooth Skin 4 influences.
Joints	Head Joints

Boundary's between the Full headdress and the other components should be considered. It is often best to create a generous neck area to guarantee it overlaps.

Budgets are a combination of Head, Hair, FacialHair, HeadPhones, JewelLeftEar, JewelRightEar.

Character Component Jewellery (Left Ear) (Right Ear)



Jewellery (Left Ear) (Right Ear) components cover the ears of the Avatar. The information here is per jewellery component - i.e.

for either the Left or Right ear, not for both.

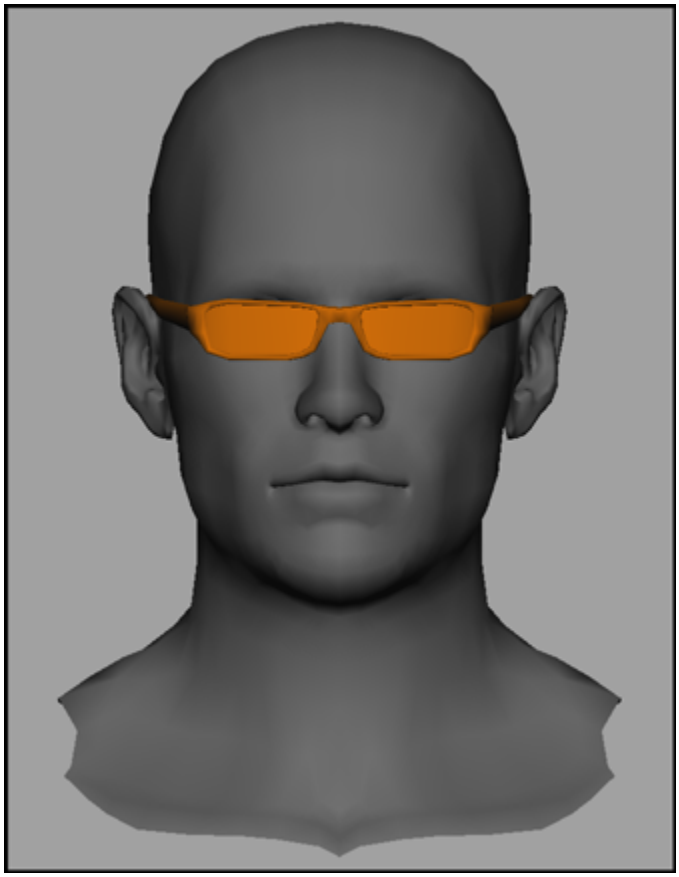
Memory Limits (KB)	Vram	Host	LOD1	LOD2	LOD3	Main
Male	14		8	4	2	2.5
Female	14		8	4	2	2.5

	Male	LOD1	LOD2	LOD3	Female	LOD1	LOD2	LOD3
Polygon Count (recommended)		250	175	50		250	175	50
Mesh Limits		5	3	3		5	3	3
Fat / Thin Blend Shapes		N	N	N		N	N	N

Shaders	Default, Emissive, Blend, Glass, Hair, Skin
Shader FX	Texture Scrolling, Animated Textures
Weighting	Smooth Skin 4 influences.
Joints	Head Joints

Jewellery components do not have Fat and Thin blend shapes.

Character Component Spectacles



Spectacle (glasses) components cover eye area of the Avatar.

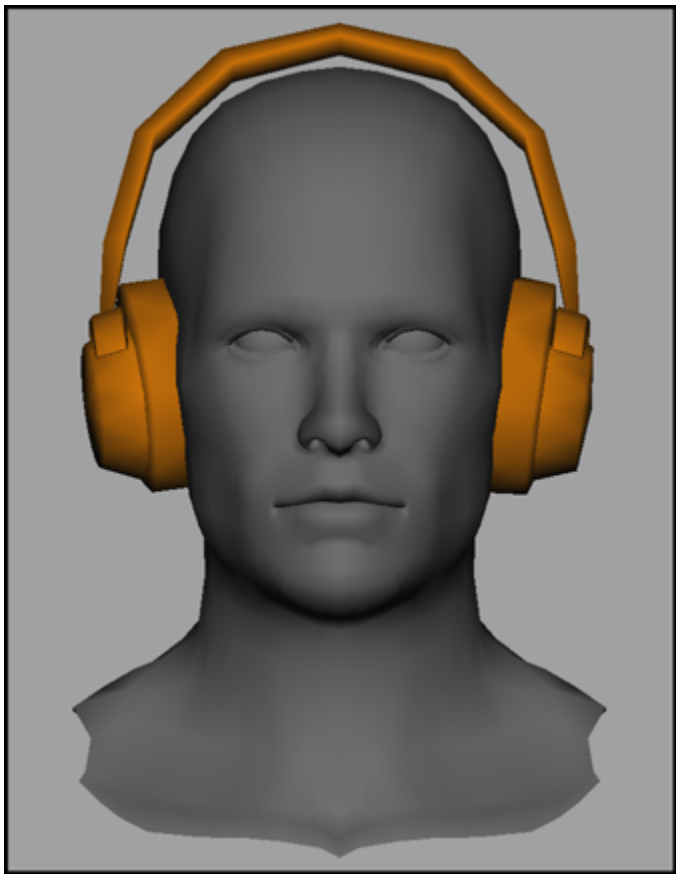
Memory Limits (KB)	Vram	Host	LOD1	LOD2	LOD3	Main
Male	33		24	13	7	2.5
Female	33		24	13	7	2.5

	Male	LOD1	LOD2	LOD3	Female	LOD1	LOD2	LOD3
Polygon Count (recommended)		600	300	150		600	300	150
Mesh Limits		5	3	3		5	3	3
Fat / Thin Blend Shapes		N	N	N		N	N	N

Shaders	Default, Emissive, Blend, Glass, Hair, Skin
Shader FX	Texture Scrolling, Animated Textures
Weighting	Smooth Skin 4 influences.
Joints	Head Joints

Glasses do not use Fat and Thin blend shapes.

Character Component HeadPhones



HeadPhones (Head Phones) components cover the upper section of the head.

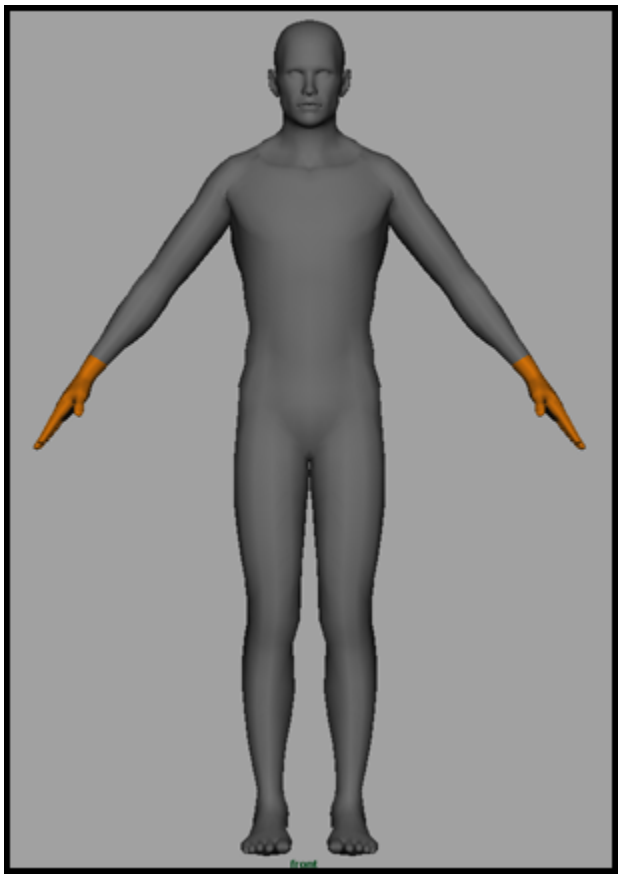
Memory Limits (KB)	Vram	Host	LOD1	LOD2	LOD3	Main
Male	44		22	12	7	2.5
Female	43		22	12	7	2.5

	Male	LOD1	LOD2	LOD3	Female	LOD1	LOD2	LOD3
Polygon Count (recommended)		600	300	70		600	300	70
Mesh Limits		5	3	3		5	3	3
Fat / Thin Blend Shapes		N	N	N		N	N	N

Shaders	Default, Emissive, Blend, Glass, Hair, Skin
Shader FX	Texture Scrolling, Animated Textures
Weighting	Smooth Skin 4 influences.
Joints	Head Joints

Headphones do not use Fat and Thin Blend shapes.

Character Component Hands



Hand components cover the wrist and hands of the Avatar

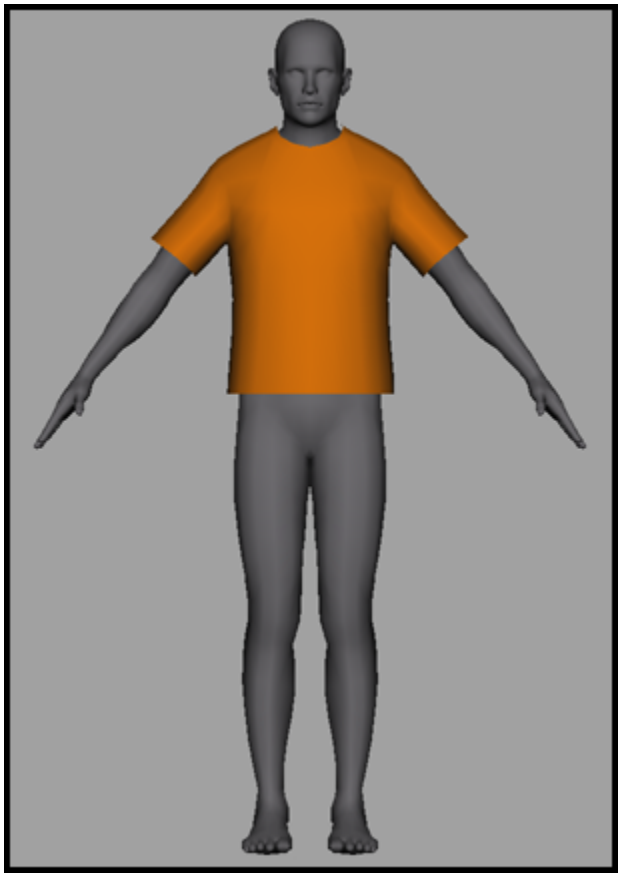
Memory Limits (KB)	Vram	Host	LOD1	LOD2	LOD3	Main
Male	1045		102	56	22	2.5
Female	1196		102	56	22	2.5

	Male	LOD1	LOD2	LOD3	Female	LOD1	LOD2	LOD3
Polygon Count (recommended)		2500	1220	390		2500	1220	390
Mesh Limits		5	3	3		5	3	3
Fat / Thin Blend Shapes		Y	Y	Y		Y	Y	Y

Shaders	Default, Emissive, Blend, Glass, Hair, Skin
Shader FX	Texture Scrolling, Animated Textures
Weighting	Smooth Skin 4 influences.
Joints	Whole Skeleton

Boundary edges between the skin sections will need to have their vertex normals aligned correctly. See the Vertex Normal Tool.

Character Component Torso



Torso components cover the upper body and arms of the Avatar.

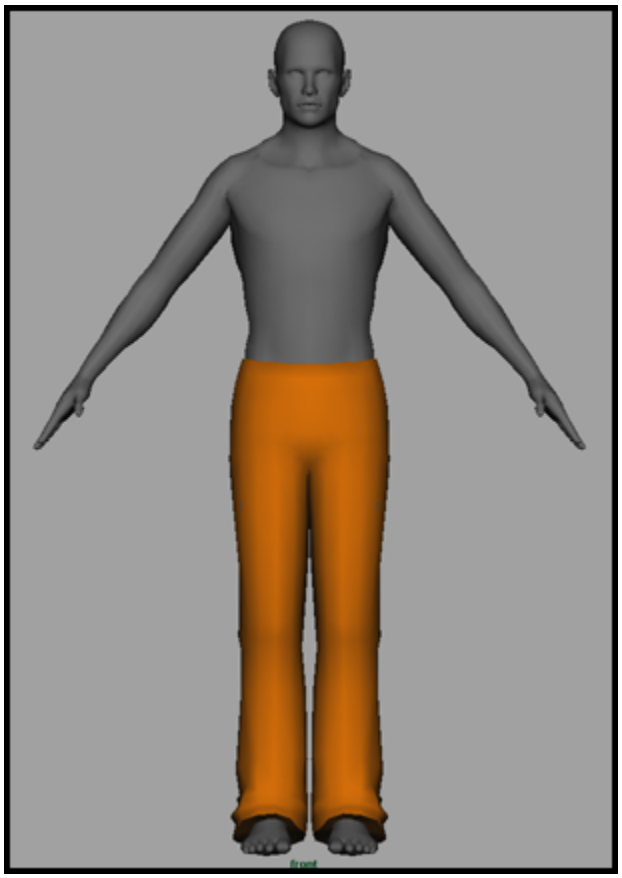
Memory Limits (KB)	Vram	Host	LOD1	LOD2	LOD3	Main
Male	1045		109	79	43	2.5
Female	1196		109	79	43	2.5

	Male	LOD1	LOD2	LOD3	Female	LOD1	LOD2	LOD3
Polygon Count (recommended)		2000	1100	470		2000	1100	470
Mesh Limits		5	3	3		5	3	3
Fat / Thin Blend Shapes		Y	Y	Y		Y	Y	Y

Shaders	Default, Emissive, Blend, Glass, Hair, Skin
Shader FX	Texture Scrolling, Animated Textures
Weighting	Smooth Skin 4 influences.
Joints	Whole Skeleton

Boundary edges between the skin sections will need to have their vertex normals aligned correctly. See the Vertex Normal Tool.

Character Component Legs



Leg components cover the Lower body and legs of the Avatar.

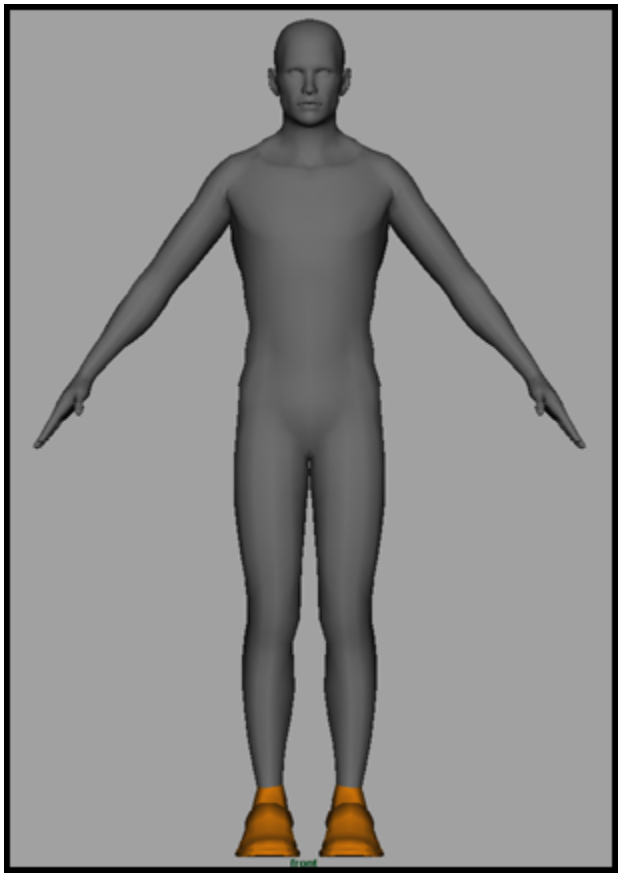
Memory Limits (KB)	Vram	Host	LOD1	LOD2	LOD3	Main
Male	683		68	42	18	2.5
Female	1217		68	42	18	2.5

	Male	LOD1	LOD2	LOD3	Female	LOD1	LOD2	LOD3
Polygon Count (recommended)		1515	830	338		2000	1100	470
Mesh Limits		5	3	3		5	3	3
Fat / Thin Blend Shapes		Y	Y	Y		Y	Y	Y

Shaders	Default, Emissive, Blend, Glass, Hair, Skin
Shader FX	Texture Scrolling, Animated Textures
Weighting	Smooth Skin 4 influences.
Joints	Whole Skeleton

Boundary edges between the skin sections will need to have their vertex normals aligned correctly. See the Vertex Normal Tool.

Character Component Feet



Torso components cover the upper body and arms of the avatar.

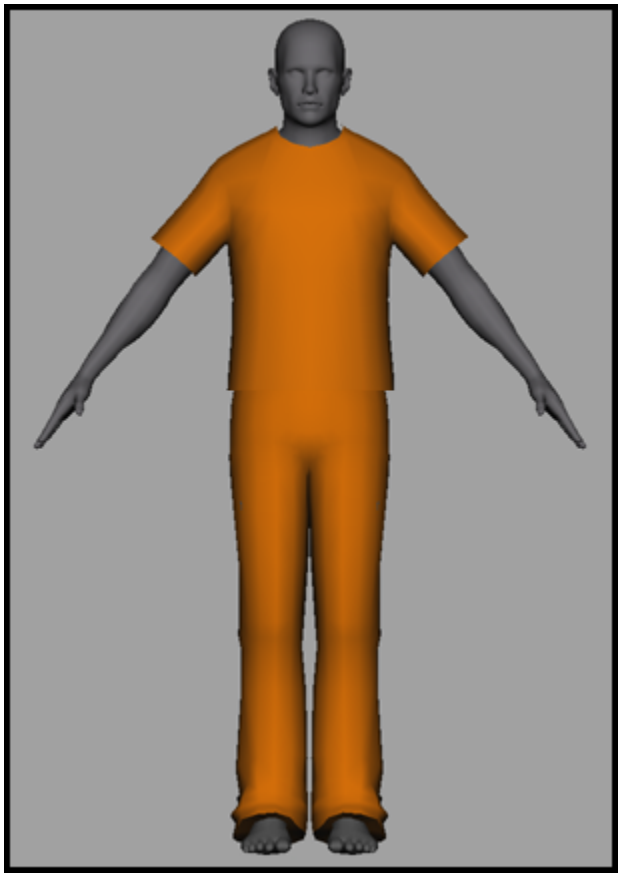
Memory Limits (KB)	Vram	Host	LOD1	LOD2	LOD3	Main
Male	812		68	42	18	2.5
Female	812		68	42	18	2.5

	Male	LOD1	LOD2	LOD3	Female	LOD1	LOD2	LOD3
Polygon Count (recommended)		790	400	132		790	400	132
Mesh Limits		5	3	3		5	3	3
Fat / Thin Blend Shapes		Y	Y	Y		Y	Y	Y

Shaders	Default, Emissive, Blend, Glass, Hair, Skin
Shader FX	Texture Scrolling, Animated Textures
Weighting	Smooth Skin 4 influences.
Joints	Whole Skeleton

Boundary edges between the skin sections will need to have their vertex normals aligned correctly. See the Vertex Normal Tool.

Character Component Torso & Legs



Torso components cover the upper body and arms of the Avatar.

Memory Limits (KB)	Vram	Host	LOD1	LOD2	LOD3	Main
Male	1728		196	134	81	5
Female	2413		196	134	81	5

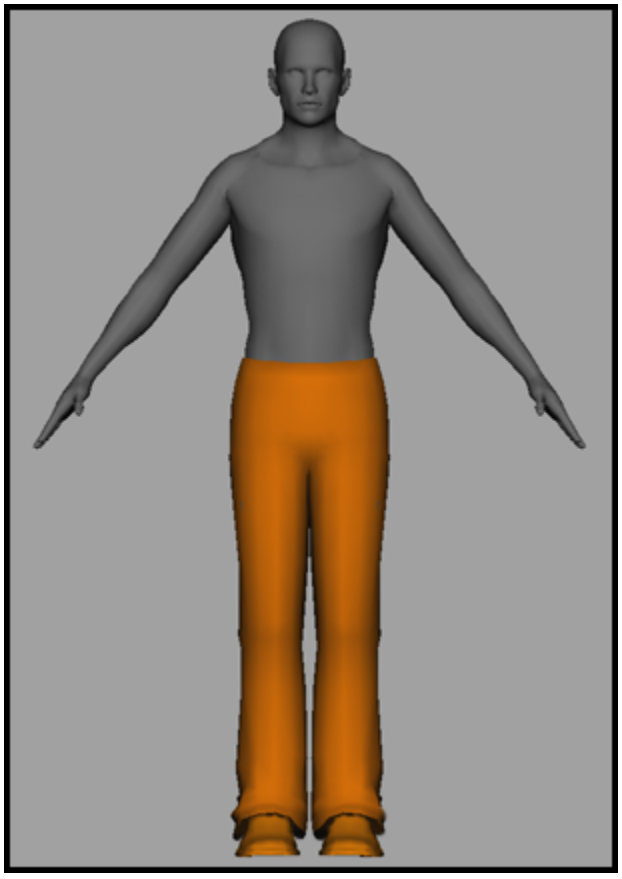
	Male	LOD1	LOD2	LOD3	Female	LOD1	LOD2	LOD3
Polygon Count (recommended)		3515	1930	808		3515	1930	808
Mesh Limits		10	6	6		10	6	6
Fat / Thin Blend Shapes		Y	Y	Y		Y	Y	Y

Shaders	Default, Emissive, Blend, Glass, Hair, Skin
Shader FX	Texture Scrolling, Animated Textures
Weighting	Smooth Skin 4 influences.
Joints	Whole Skeleton

Boundary edges between the skin sections will need to have their vertex normals aligned correctly. See the Vertex Normal Tool.

Budgets are a combination of Torso, Legs.

Character Component Legs & Feet



Torso and Leg components cover the lower body and feet. Composite items are handled as a single component so boundary edges can be less strict.

Memory Limits (KB)	Vram	Host	LOD1	LOD2	LOD3	Main
Male	1495		155	97	56	5
Female	2029		155	97	56	5

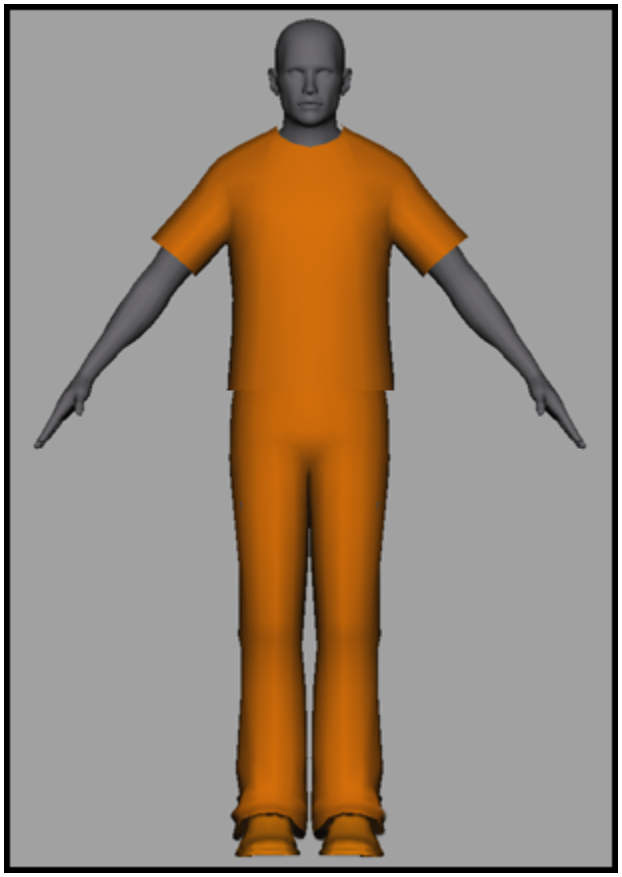
	Male	LOD1	LOD2	LOD3	Female	LOD1	LOD2	LOD3
Polygon Count (recommended)		2305	1230	470		2305	1230	470
Mesh Limits		10	6	6		10	6	6
Fat / Thin Blend Shapes		Y	Y	Y		Y	Y	Y

Shaders	Default, Emissive, Blend, Glass, Hair, Skin
Shader FX	Texture Scrolling, Animated Textures
Weighting	Smooth Skin 4 influences.
Joints	Whole Skeleton

Boundary edges between the skin sections will need to have their vertex normals aligned correctly. See the Vertex Normal Tool.

Budgets are a combination of Legs, Feet.

Character Component Torso & Legs & Feet



Torso Legs and Feet components cover the upper body and arms, legs and feet of the Avatar. Only the hands and head are left out of this.

Memory Limits (KB)	Vram	Host	LOD1	LOD2	LOD3	Main
Male	2540		264	176	99	8
Female	3225		264	176	99	8

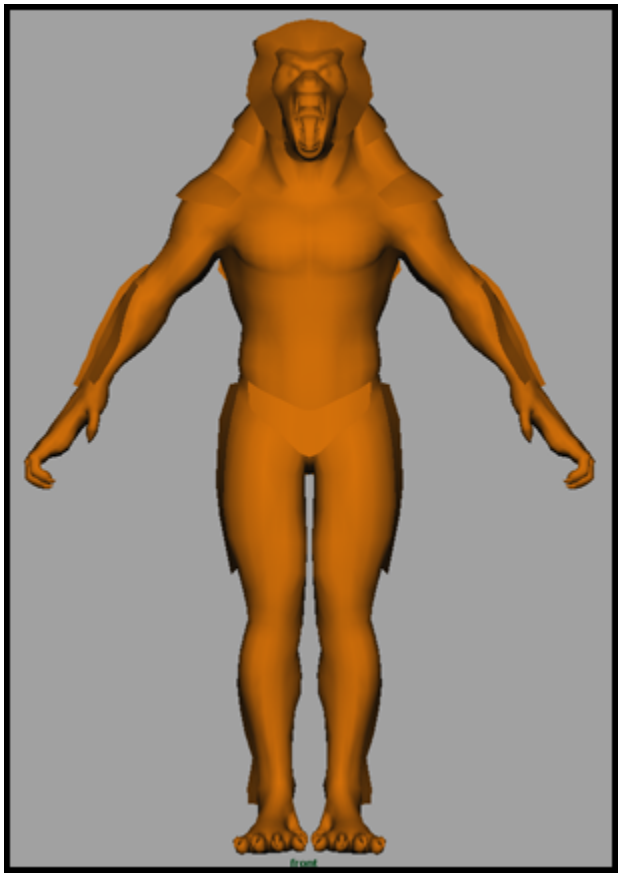
	Male	LOD1	LOD2	LOD3	Female	LOD1	LOD2	LOD3
Polygon Count (recommended)		4305	2330	940		4305	2330	940
Mesh Limits		15	9	9		15	9	9
Fat / Thin Blend Shapes		Y	Y	Y		Y	Y	Y

Shaders	Default, Emissive, Blend, Glass, Hair, Skin
Shader FX	Texture Scrolling, Animated Textures
Weighting	Smooth Skin 4 influences.
Joints	Whole Skeleton

Boundary edges between the skin sections will need to have their vertex normals aligned correctly. See the Vertex Normal Tool.

Budgets are a combination of Torso, Legs, Feet.

Character Component FullBodySuit



The FullBodySuit component covers the whole avatar. All elements of the original avatars mesh are replaced by the FullBodySuit.

Memory Limits (KB)	Vram	Host	LOD1	LOD2	LOD3	Main
Male	5042		915	384	171	31
Female	5096		872	378	178	29

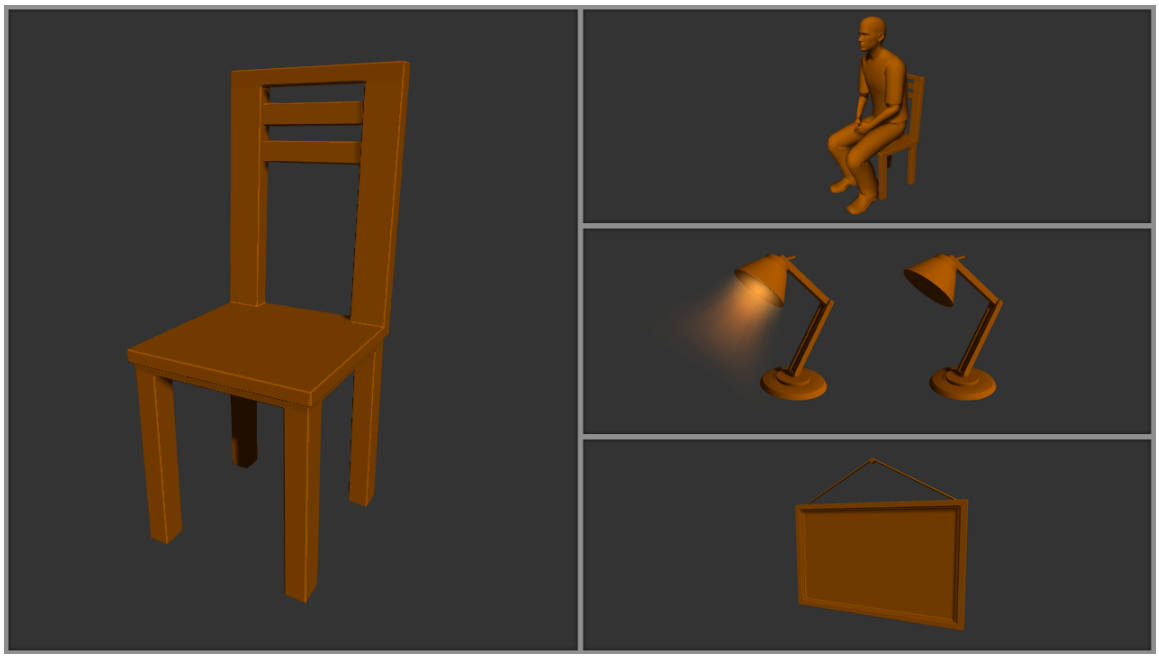
	Male	LOD1	LOD2	LOD3	Female	LOD1	LOD2	LOD3
Polygon Count (recommended)		13350	7020	2366		13350	7020	2366
Mesh Limits		58	33	33		58	33	33
Fat / Thin Blend Shapes		Y	Y	Y		Y	Y	Y

Shaders	Default, Emissive, Blend, Glass, Hair, Skin
Shader FX	Texture Scrolling, Animated Textures
Weighting	Smooth Skin 4 influences.
Joints	Whole Skeleton

Budgets are a combination of Head, Torso, Legs, Feet, Hands, Hair, FacialHair, HeadPhones, JewelLeftEar, JewelRightEar, Glasses.

Furniture Tools

Furniture Models



Furniture models flow the same basic rules as standard models. There are 4 basic types of furniture item.

- General Furniture (a placeable item)
- Seating (object the avatar can sit on)
- Lights (object that has a built-in light and on/off setting)
- Picture Frames / Wall Decorations (object that can be placed on picture hooks on walls and other vertical surrfaces)

General furniture - a placeable item - is a model with collision that can be placed in an apartment from the Furniture Menu. There is no additional interaction for this type of object after it is placed.

Seating - an object the avatar can sit on - is a furniture item that contains a seating locator that, once placed, the avatar can elect to sit on when approached.

Lights - an object with a built-in light and on/off setting - is a furniture item that contains a light locator with a realtime light that can be switched on and off by the user.

Picture frames / Wall Decorations - an object that can be hung on walls and could have pictures swapped out (in the case of picture frames with swappable images) - is a furniture item hung on picture hooks (hooks can be included in Personal Space and Clubhouse scenes). There are two types of picture frame: one has a fixed image as part of the artwork and doesn't swap out, whereas the other allows the image to change (either on a rotating cycle from associated images, or via user selection from the PlayStation HDD). Wall decorations are also available, and since wall decorations don't have picture frame behavior of images that can be swapped out, they can be thought of as general furniture that can be placed on a picture hook (for example crossed halberds).

There are additional items (locators) that can be added to a furniture item in Maya:

- Seating Locator
- Light Locator

These will add extra information on export and will also enable the associated behavior in the client (e.g. avatars will be able to sit on the furniture or switch on/off lights).

By exporting with the Furniture Export Profile, the object for the furniture item will also be created for you.
This Furniture Tools section provides more information on furniture object types.

See the Maya Furniture Scene for detailed information on what is supported or required when exporting furniture items.

Furniture Collision

See Collision For Furniture and Objects.

Furniture Menu Options

The Furniture items you make in Maya will have options during export for you to specify which client menu the item will appear in. All items have to go into at least one menu category, or it will not be selectable.

General furniture - Furniture of the general type can appear under one of these menus in PS Home

- Appliances
- Cubes
- Flooring
- Footstools
- Ornaments
- Storage
- Tables

Seating - Furniture of the seating type can appear under one of these menus in PS Home

- Chairs
- Sofas

Lights - Furniture of the lights type can appear under this menu in Home

- Lamps (desk lights, standard lamps, etc.)

Picture Frames - Furniture of the picture frame or wall decoration type can appear under one of these menus in PS Home

- Picture frames
- Wall Decorations

Furniture Actives and Arcades

Active Furniture

Active Furniture is an object with a scripted component, and this object can be selected from the furniture menu and placed in Personal Spaces or Clubhouse Spaces.

Think of an Active Furniture Object less as furniture and more as a scripted item that can be added to the furniture menu and has furniture behavior for placement and targeting.

Therefore, an Active Furniture Object does not refer to a kind of furniture object, it just means that the Active Item can appear under any of the furniture categories in PS Homes menus. It need not be furniture at all.

It is possible to attach an exported furniture .mdl from Maya to the object, of course, but you are not limited to just furniture art assets - you can use any of the following:

- Furniture
- Model
- Prelit Model
- Animated Model

For more information on the Active Furniture Objects:

Furniture Object Component

Furniture and Decorations

Arcade Cabinet

Arcade Cabinet Objects are a very specific type of furniture. They are essentially a way to place and access an arcade game without building it into a scene.

An arcade game is launched via a screen, and an arcade cabinet is a way to provide an object that houses this screen and can be placed from the furniture menu.

The Arcade Cabinet Object will need a .mdl file that contains the geometry for the cabinet. This can be exported as either a furniture object or a model.

When creating the geometry for this object, you will need to have a backing for the screen. The screen is created by attributes stored in the object, but its advisable to create some geometry to go behind the screen. This geometry will prevent alpha sorting issues on the screen. The backing to the screen is also necessary as the screen is semi-transparent.

It is also a good idea to create a geometry plane in the Maya scene that you can take the coordinates for the screen from. This does not have to be exported - it can be templated and left in the scene for reference.

See Deploying an Arcade Game in Furniture for more information.

Furniture Chairs and Seating

Seat Locators for Furniture in Maya

Seat locators can be added to a furniture object to allow the avatar to sit on the object.

There are three heights to choose from - low, medium and high.

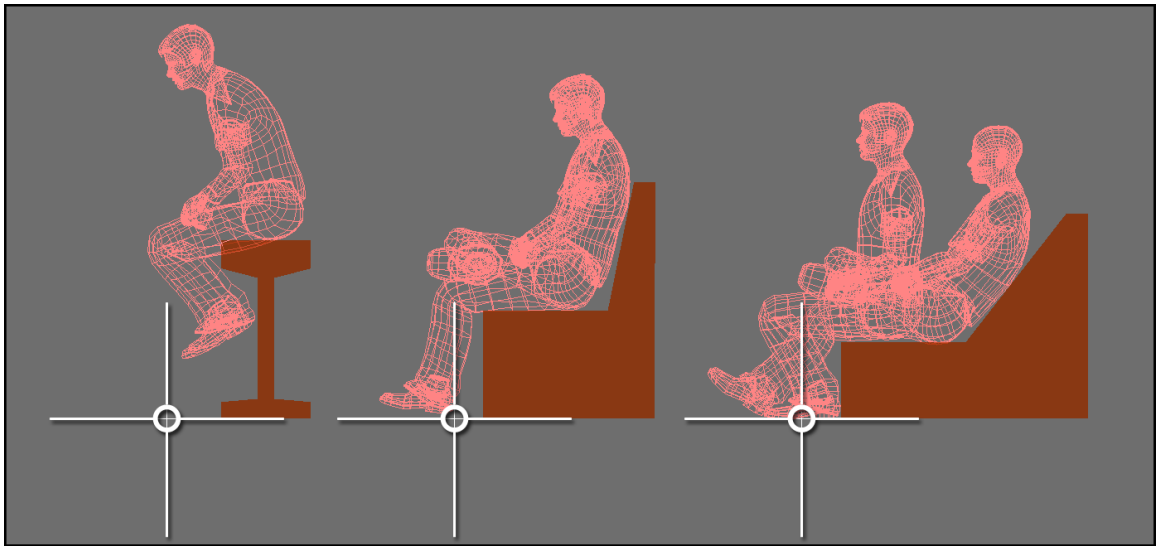
Once a seating locator is added to a furniture object, avatars in the client will be able to target and sit on the object (it can be a chair, a rock, etc.)

To add a seat locator to a furniture object, go to:

Home > Furniture > Add seat locator

Select a Character Seat Locator of the appropriate height from the Furniture Tool Shelf in Maya (Home_Furn). You can choose from the following:

- Low: reclining seat height (0.35 m)
- Medium: standard seat height (0.5 m)
- High: bar stool style seat height (0.7 m)



Multiple Seat Locators

Furniture items can have up to three seats. For furniture that seats more than one, just add more Character Seat Locators.

> ⚠ There is a limit of 3 seating locators per furniture object.

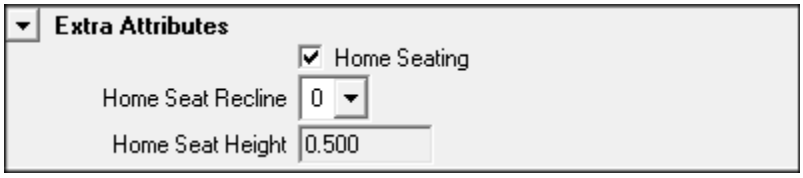
The spacing of the locators are handled within Maya. However, to prevent Avatars clipping into each other, the locators should not overlap.

Recline

When creating your seating locator, you can use the Recline parameter so that the avatar can lean back into a relaxed position (for example on a recliner sofa).

To set the Recline parameter:

1. Select a HomeSeat.
2. In the Attribute Editor, select Extra Attributes.
3. 0 (no recline) is selected in the Home Seat Recline drop-down list by default:



To set up a reclining seat, select 1 from the Home Seat Recline drop-down list.

Seat Collision

Seats can have some additional issues in conjunction with the camera. See Collision For Furniture and Objects for more details on this.

Furniture Lamps and Lights

Lamps

A lamp is an item of furniture that has a switch so the user can turn the light on and off. It is a simple user interaction that allows the user to activate or deactivate a light source and is handled by the client itself, including the button for the user to interact with when they approach the light.

Lamps are created in the same way as other furniture objects, but with a light locator added. This locator will control where the light for the lamp will emit from.

When lamps are embedded in a scene, the interactive functionality (turning them on and off) no longer works.

Light Locator

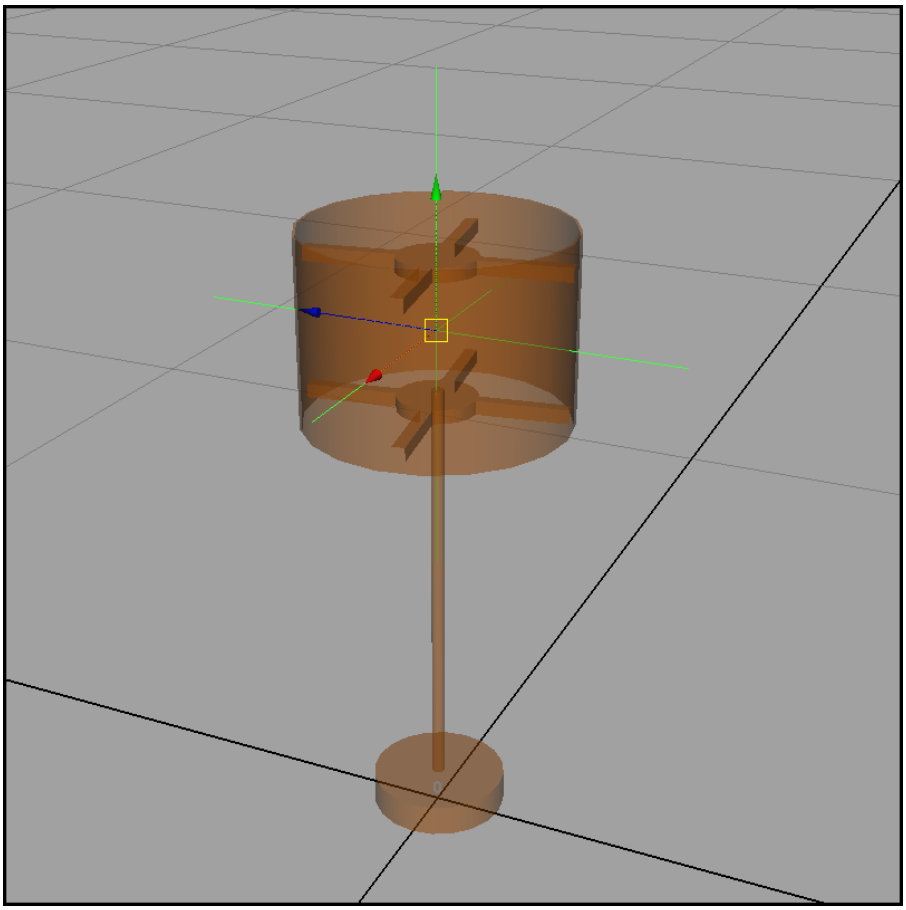
For each lamp you must have 1 (and only one) light locator.

Home > Create Light Locator

The light locator acts like a light bulb, emitting light in all directions.

Place the light locator where you want the light to emit from (i.e. where you would have put a bulb).

You can specify the light mask in the Attribute Editor (more detail on light masks can be seen below) or in the Object Editor (under the resource named light_resource_light_mask_0).



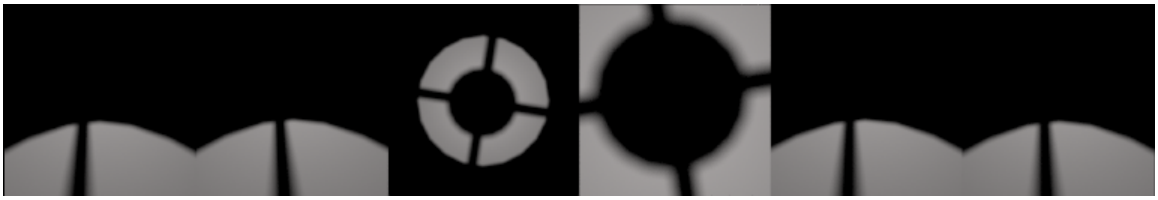
Light Masks

The Dynamic Light in a lamp is a Point Light.

You can assign a light mask to it to control where the light will cast from the lamp.

For example:

A lamp with a shade will need to cast up and down but not from the sides. Because a point light casts in all directions, use a cube map to mask the light.



There are several of ways you can create a cube map to use as a light mask.

Option 1: Inside the Model Viewer

Export the Light fitting or lamp and load up the Model Viewer

Home > Environment > Launch PS3 Model Viewer

Chose the option to hide the grid inside the Model Viewer.

Maneuver the camera so that it is in the position the light will be in and then use the console command envShot to create a cube map.
You can then invert this image in Photoshop to give you the lightmap. You may also have to make further tweaks in Photoshop, ensuring the light and dark areas are correct.

Option 2: Process for Maya (more accurate)

This option gives you a more accurate and precise rendered light mask, but takes longer and requires more familiarity with Maya.

Create a cube with inverted faces and then use the Render to Texture feature in Maya to bake the lighting out onto the faces of the cube.

This will provide you with a mask for your light that can contain complex lighting data and color. The advantage of this approach is the ability to have softer and more subtle lighting.

> To change the color of the light, simply change the white within the mask to a color.

See also:
How to bake out your Light Mask in Maya
Cube Maps in PS Home

Light Object Properties

1. The offset and light mask texture for the light locator is displayed in the Object Editor under Components > light > lights > light(0).
2. The texture used by the light mask for the point light is displayed in the Attribute Editor (the default mask is a solid white texture)

> The values for the point light in a lamp are Attenuation Start = 0, Attenuation End = 2.5 and Attenuation Power = 2. These are not editable.
> For more general information on point lights in PS Home, see Adding Dynamic Lighting.

3. Export the lamp furniture object, ensuring that you check the Update Object Catalog box.

Maya Light Mask Baking

Maya Render To Texture

Maya render to texture feature will allow you to create a lightmap based on the UVs of a mesh. By laying out the UVs from a cube in the corrrect way a cube map can be generated for use as a light mask in Home.

For more information on Maya's render to texture functionality: Maya Help

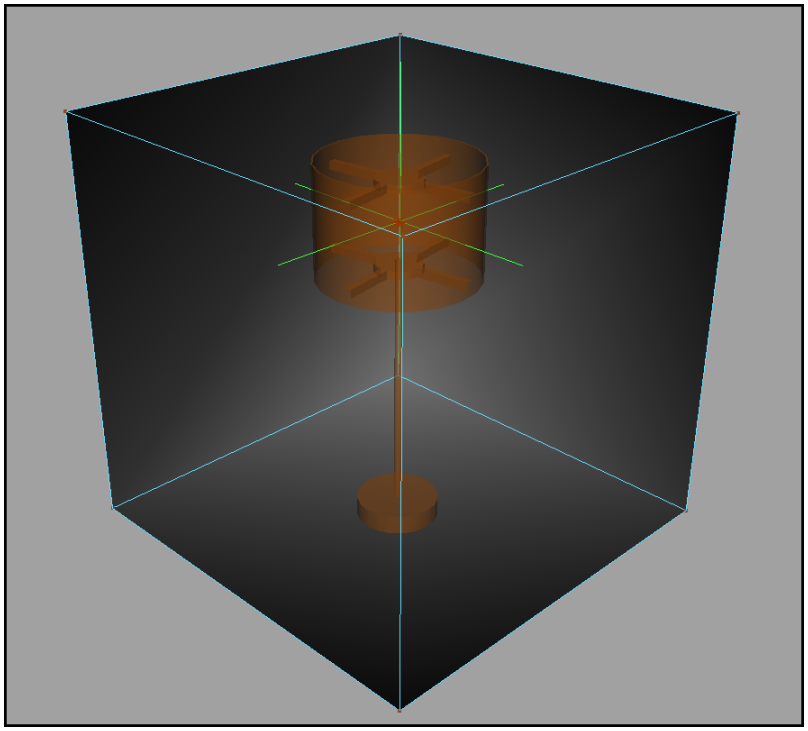
To do this you will need to create a cube and invert the normals of the faces. This will create a cube with all the faces pointing inwards. You can then place the lamp object and lights inside so that they cast onto the faces of the cube.

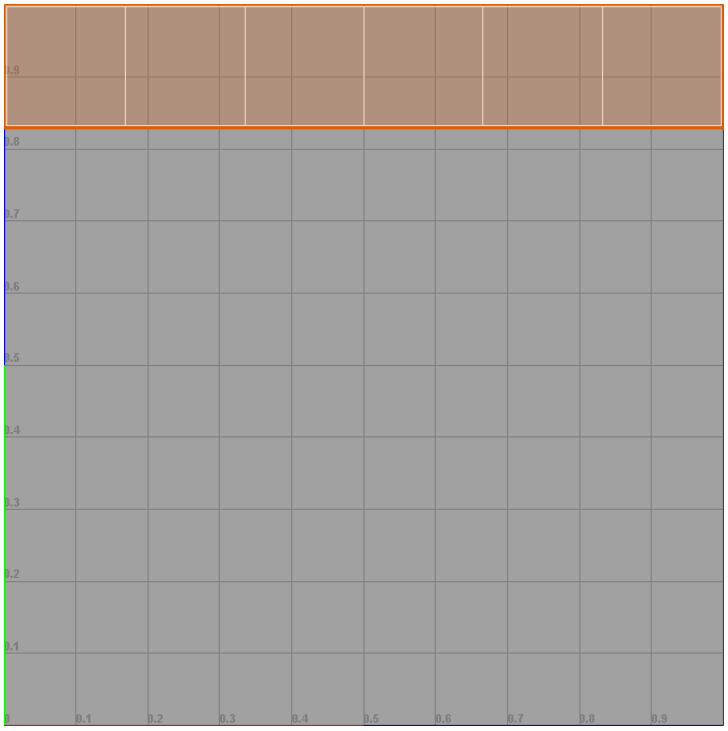
UV Layout

The UVs of the cube will need to laid out and oriented to match Homes cube map format.Below is an example of the Box and the UV layout it uses. The UVs are placed at the top of the 0-1 texture area.

Download Baking Scene

Below you can see an example of the lamp within the cube and how the UVs are laid out in the 0-1 UV area.





For more information on exactly how the UVs will need to be oriented to create the correct Cube Map format for home: Cube Maps

Lights

By baking the lighting to a texture you are able to use any Maya lights you want. It is keep mind that only single point light

will be emitting the light when in game. It is often a good idea to match a Maya point light with the location of the Light locater, this will give you an accurate initial lighting solution.

Baking

To bake the lighting to a texture you will need to use Mayas Render to texture feature. This will work on a selection and you will need to have the cube selected.

Choose Rendering from the Maya menu set dropdown

Then Lighting/Shading > Batch Bake (Mental Ray)

Furniture Picture Frames and Wall Decorations

These are furniture object (picture frames and wall decorations) that can be hung vertically from hooks in spaces.

Wall decorations are simple objects that can be hung on hooks (e.g. a set of crossed hockey sticks, a moose head, a rug).

Picture frames, like wall decorations, can be hung on hooks, but picture frames have an additional function - there is an area where a texture is swapped out and displays images selected by the user from their HDD.

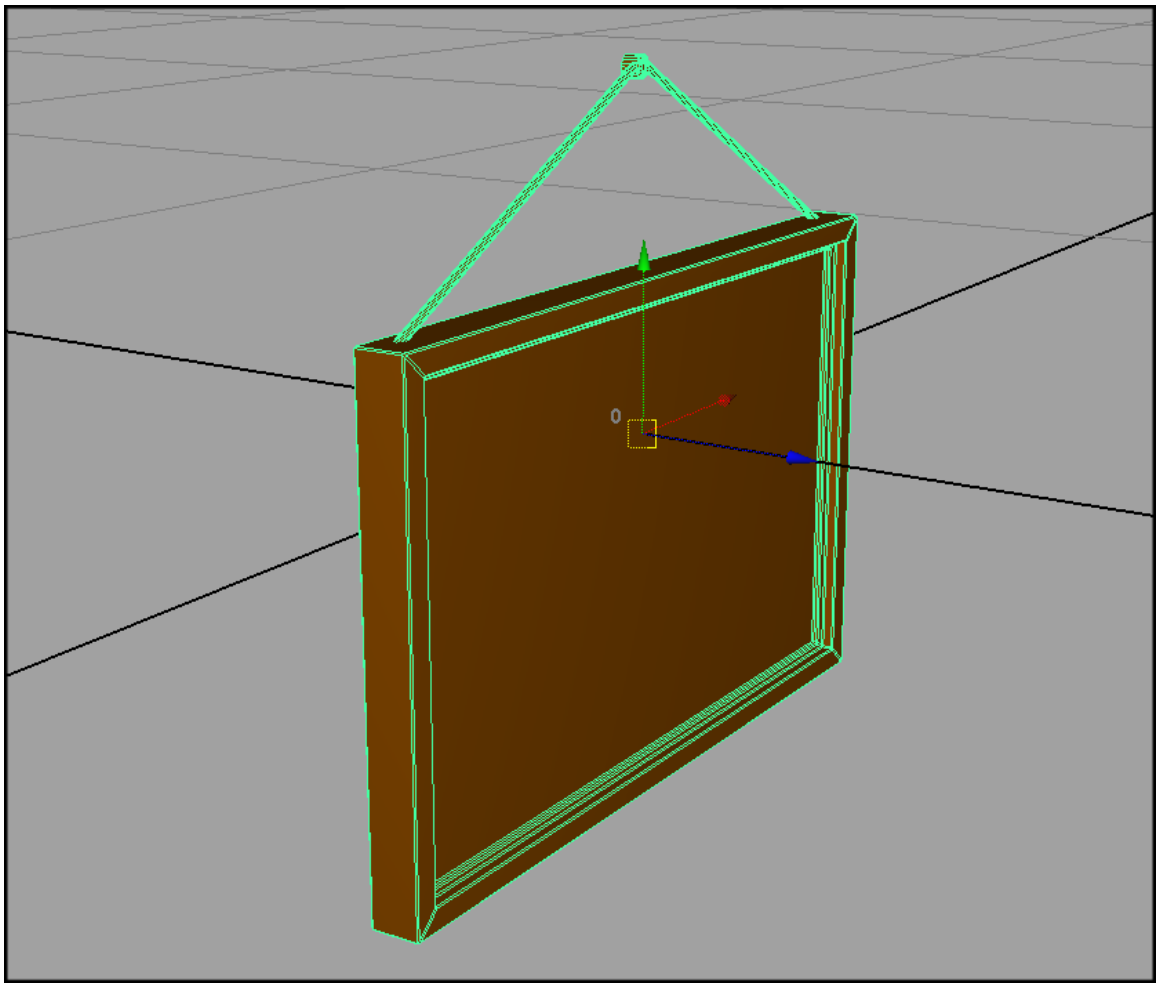
Placement In Maya

Hanging Objects (picture frames and wall decorations) should be placed so that 0,0,0 in Maya is the location where the frame will snap to the picture hook.

- The wall hanging should face Positive Z in Maya

  > Negative Z is the area behind the picture hook, anything behind this point will be inside the wall and will not display.

- Maximum dimensions for picture frames: 2 m (X axis) x 2 m (Y axis) x 0.3 m (Z axis - e.g. protrusion)



Picture Frames

Picture Frames are dynamic because they allow users to load and display pictures from their PlayStation®3 HDD.

Picture frames have an area that is a shader with a default texture as a placeholder. This default texture is marked in the object (within Object Editor) as the texture to replace with user images.

terminology clarification

material mesh shader texture

In Maya, assign a placeholder texture to the geometry shape on which you want to users to place pictures. The area that this texture is assigned to will swap out for the custom image. There are no restrictions on shape or layout, the placeholder material can be applied to any part of the mesh you wish.

To Assign a Placeholder Texture:

1. Create another ATG Material, and select default type.
2. In Material Parameters > Colour Map, choose a placeholder texture. This example uses the Placeholder_c.dds texture in <HDK_INSTALL_DIR>\artsouce\Generic_Shared_Textures:

Once exported this texture will be selected in the object to be replaced by the custom image. See Exporting Picture Frames for more information on where to set the texture that will be replaced.

Wall Decorations

Wall decorations are simply a mesh that is placed on picture hook. There are not special limitations for this mesh. The same rules as above apply for the placement of of the wall decoration and the mesh will need to placed so that it extends into the positive Z axis.

Environment Tools

On this page you can find a list of tools for environments.

Baked Lighting Tools

Maya Lights - Sun, Dome, Maya Lights and Nova

Lighting Set Editor

See also Lightmap UV set 2, including:

- Fix Missing UV set 2
- Texel Scale

Other Environment Tools

Dynamic Reflection Editor

LOD groups

Atgi Locators

Creating Particle Locators

Cloth Simulation for Meshes

See also: Sky Design

Maya Lights

Sun and Dome - Default Environment Lights

The Sun and Dome light are the default lights required for an environment, they provide the base lighting information for the scene and the Sun light position in Maya will directly relate to the position of the sun in the scene. This sun position will also be used for lighting dynamic objects.

The Dome Light will provide a basic ambient light to the baked lighting solution.

> Sun and Dome lights are required before you can export a scene.

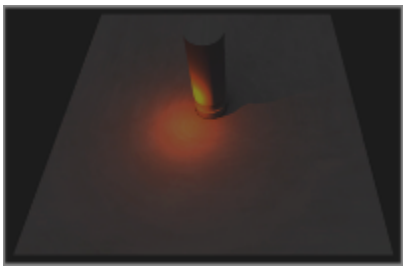
Maya Lights

All of the lights listed below will work when lighting a scene or prelit model in Maya.

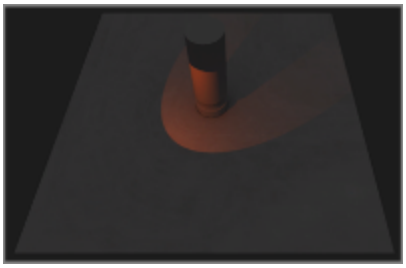
> For more information, see Maya's Online Help on creating all these light types.

Maya Lights:

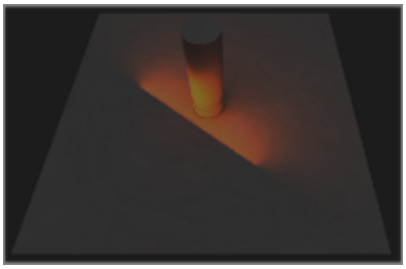
- Point Light
- Spot Light
- Area Light
- Volume Light
- Directional Light*



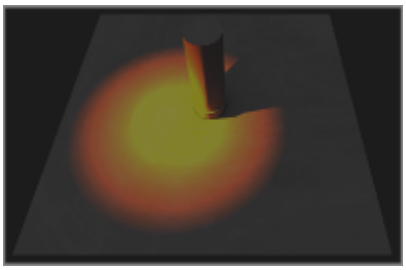
Point Light



Spot Light



Area Light



Volume Light



Directional Light

Nova Lights

There are 2 types of light that can be setup as Nova Lights: Spot and Area.

Most of the options available are for the Area light. The main advantage of the area light is the falloff, which helps to create softer and more realistic lighting.

After running the mel command with a selected Light:

`atgAddNovaLightAttributes;`

The following attributes are added to the selected light. They appear under its shape node in the Extra Attributes tab.

Attribute	Type of Light Effected	Description
ATG Nova Solid Angle	Directional	Gives a directional light scale (in radians) as if it was cast from an infinitely far disc (ie. sun) This softens the edges of the shadows

ATG Nova Spot Light Shape	Spotlight	Changes the shape of the spotlight projection, Square and Circle are the options.
ATG Nova Disk Radius	Spotlight	Represents the radius of a disc-shaped light source in meters. This results in softer shadows, as if cast by a bulb with real dimensions rather than a point. Values of less than 0.5 will often produce very little or no light. Increase the intensity if you plan to have very small lights.
ATG Nova Disable Indirect	All	The light will only cast directly onto surfaces. No bounced light will be calculated. This can be useful with very bright lights to prevent bright bounced light.
ATG Nova Disable Direct	All	The opposite of above. Only the bounced light will be calculated. Can be useful if you want only an ambient light with no direct harsh lighting
ATG Nova Disable Shadow Casting	Delta Lights *	Disables shadows for this light only - NB. also disables indirect lighting * Delta Light: Light from an infinitely small point - either point light, spotlight, or directional light (ie. hard edged shadow lights only)
ATG Nova Light Shape	Area Lights	Used to determine the shape of area lights. Options are quad, disc and sphere. Sphere makes the area light into a point light with scale
ATG Nova Spot Light Shape	Area Lights	This option is not currently used with the home tools
ATG Nova Visible In Camera	Sky Dome Lights	This option is not currently used with the home tools
ATG Nova Indirect Scale		This option is not currently used with the PS Home tools

By adding these attributes to your Spot and Area lights you can significantly change the look of your scene.

Options such as disabling the bounced light can allow you to place lights that simply give an interesting direct illumination without loading your scene with bounced lighting.

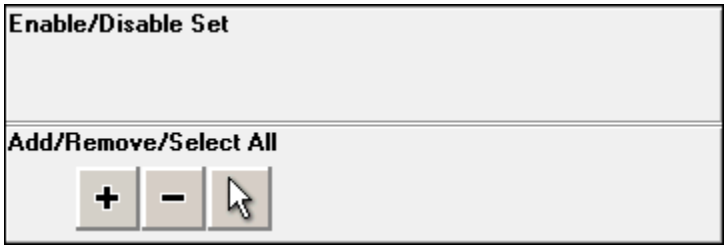
Lighting Set Editor

The Lighting Set Editor lets you to assign geometry to a group that affects how lighting is handled.

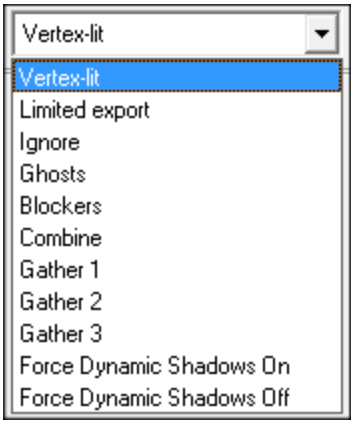
There are cases where a mesh can be in more than one group at a time.

Home > Environment > Lighting Set Editor

Objects are added and removed by selecting them in Maya and then using the requisite button.



Below is the list of light set options:



Set	Description
lighting_blockers	Members of this set do not appear in the final scene but are used in lighting calculations, e.g: bounced light, shadow casting, etc.
lighting_combine	Members of this set are combined for lightmap generation purposes. Normally, each individual mesh in a scene generates its own lightmaps. Using this set lets you have multiple meshes share one lightmap, e.g. for a number of small objects, without having to actually combine them into a single mesh. ⚠ Make sure that when all the member objects are selected, their collective map2 UV layouts do not overlay each other.
lighting_forceDynamicShadowsOff	Objects in this set with a shader that should cast a dynamic shadow are forced not to cast a shadow.
lighting_forceDynamicShadowsOn	Objects in this set are forced to cast a dynamic shadow. This works with all shader types including prelit objects. Prelit objects in this set have both a baked-in shadow and a real time shadow. If the object has a shader that does not cast a shadow, e.g. semitrans and emissive, the shadow is cast from the geometry and not from any alpha component of the shader.
lighting_gatherGroup_n	Where n can be any number or unique name. During final gather calculations that deal with bounced light, certain photon and geometry intersection calculations are bypassed in the interests of speed. This can occasionally cause unwanted results, such as light bleeding or photons becoming trapped between surfaces creating 'glow' artifacts. Assigning meshes with final gather lighting conflicts into different gather groups can be used to resolve these kinds of problems.
lighting_ghosts	Members of this set receive lighting and appear in the scene, but do not cast shadows, e.g. glass panels or windows.
lighting_ignore	Members of this set are completely ignored by Nova but will still appear in scene, and instead take lighting information (if the material assigned requires it) from the light probes which are used to light characters and other dynamic objects in the scene.
lighting_limitOutput	Relighting the entire scene every time you export is both time consuming and unnecessary. If any meshes are added to this group, then Nova only calculates lighting for them, but still use all untemplated scene geometry for those lighting calculations. For example, you may have just moved a tree a little to the left and want to see how its shadow looks now without having to relight the entire scene. Just add the surface that receives the shadow to limitOutput and Nova uses all geometry in its calculations, meaning the tree still throws a shadow. ✓ This functionality is very useful for tweaking small areas of a complex scene.
lighting_vertexLight	Members of this set are vertex lit. One of the fundamental properties of Nova is that all geometry is lightmapped by default. If you want a mesh to use baked vertex lighting instead, then it must be made a member of this set. Vertex lighting is good for things that would gain no benefit from the more time-consuming and resource intensive (at runtime) use of lightmaps. Such objects are typically small, fiddly objects such as the leaves of a tree or vertex dense, organically shaped objects such as stones or rocks. Other types of object that could usefully be vertex lit are those where the lighting is not as important, e.g. glass or distant objects like mountains, etc.
lighting_forceDynamicShadowsOn	Forces members of this set to cast a dynamic shadow (regardless of shader type).

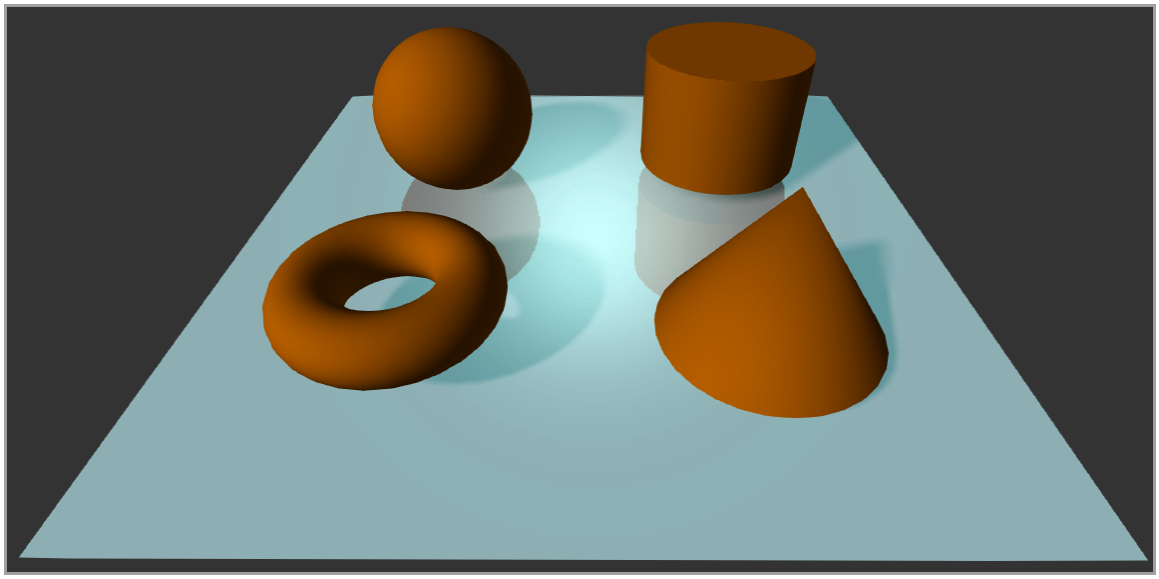
lighting_forceDynamicShadowsOff	Forces members of this set to NOT cast a dynamic shadow (regardless of shader type).

Dynamic Reflection Editor

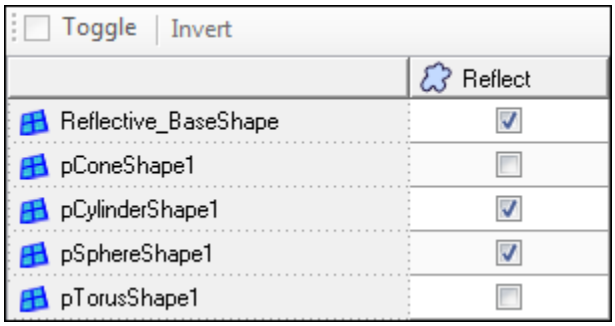
The Dynamic Reflection Editor is used to control what will and will not reflect in a surface that has a shader with Real Time Reflections applied to it.

Home > Environments > Dynamic Reflection Editor

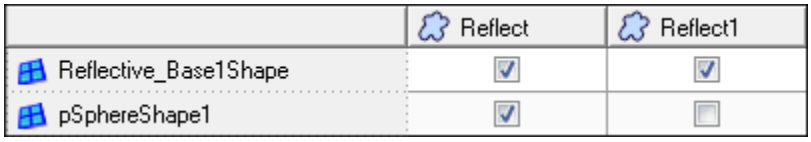
The image below shows the Cone and the Torus not reflecting in the plane while the Sphere and Cylinder do reflect.



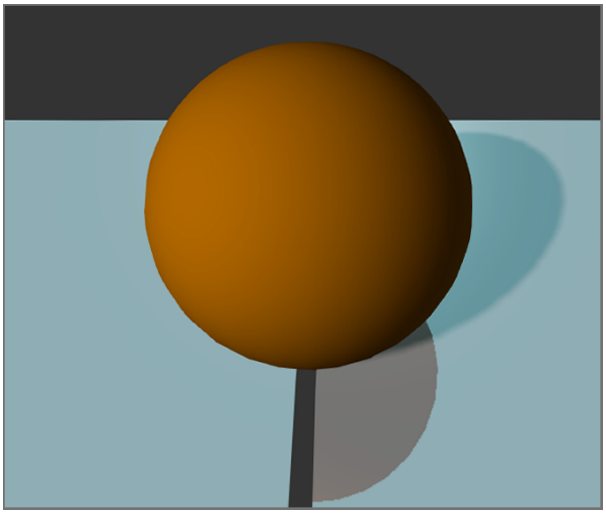
Below you can see the options set in the editor to make this happen. Meshes with the box checked will reflect, meshes with boxes not checked will not reflect.



If there is more that one reflective material, there will be an additional column. You can then set object reflectivity based on material.



The image below show this reflection set for different reflective materials: Reflect is on the Right and Reflect1 on the left. Note the missing reflection on the left-hand surface.



Tool Options

Filter

- Show all - Show all the meshes in the scene
- Show DCC Selection Only - Shows the initial selection when the tool was opened

Live Select

- Meshes - Will select and focus on the mesh in the view port
- Materials - Selects by material type

Options

- Display Full Mesh Names - Displays the full mesh names within the tool

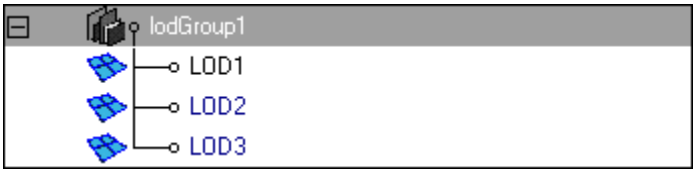
LOD Groups and LODs

Creating LOD groups

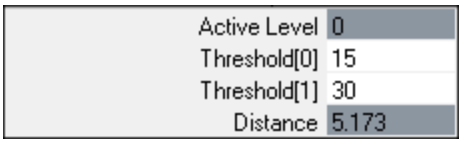
PS Home uses Maya native LOD group system. Select the LOD meshes and then assign them to a LOD group:

Edit > Level of Detail > Group

The LOD group can be viewed in the outliner. The order of the meshes in the group will determine the order in which they swap.



The distance threashold they swap at can be set in the properties for the group.



In this case the Thresholds are 15 and 30. These are the distances in game that the lods will swap at. Therefore LOD1 will be visible till 15 meters, at which point LOD2 will become visible, and LOD2 will be visible until 30 meters, where it swaps to LOD3.

Maya LOD groups

LODs can be used to swap between meshes based on distance from the camera. These meshes can be completely different from each other, they do not require the same material or even lighting model. You can take advantage of this to reduce the total polygon count of the object, the shader count and type, as well as amount of VRAM the textures use.

In the example below you can see the reduction in polygons between 1 to 3. Each LOD mesh also has a different shader and texture applied to it.



LOD Lighting

LODs are subject to the same rules as standard meshes in a scene or in models. The lighting format used will depend on the type of shader applied.

For example: it is possible to have LOD 1 be a prelit object and LOD 2 to be dynamically lit and LOD 3 to be emissive (no lighting).

The ability to have different shaders and textures per LOD allows you to achieve memory and runtime savings in performance.

Atgi Locators

Atgi Locators

When you have exported your Maya assets as `.atgi` files, you can reference them from a Maya scene by placing an Atgi Locator in the scene and specifying the Atgi filename you want to reference.

The advantage of referencing `atgi` files via Atgi Locators is that you can update an `atgi` file, and the updates will be made in all instances where the file is referenced (i.e. in the Atgi Locators).

> ⚠ This is not instancing. It will not save memory. The referenced meshes will all be exported down into a single file as unique geometry.

To export a Maya scene as an `atgi` file that can be referenced as an asset:

Select Home > Environment > Export .atgi

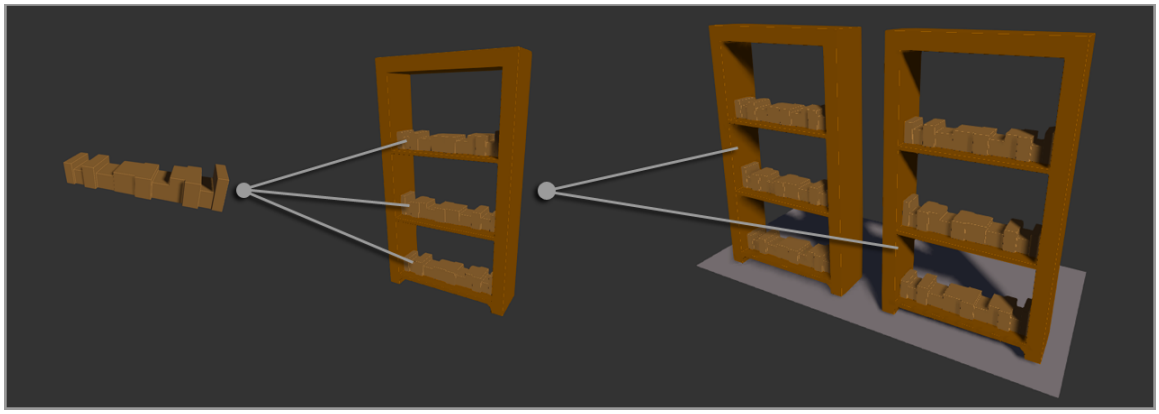
`Atgi` files are located in `<HOME_INSTALL_DIR>\Intermediate`

When nesting objects and using `atgi` locators, you will need to export an `atgi` at each stage.

For example – Bookcase with Books

In the example below there is a bookcase with books in it. The books are referenced into the bookcase scene, and the bookcase is in turn referenced into a final scene.
When updating, for example, the books, you would need to re-export the `atgi` from the books scene as well as the `atgi` from the bookcase scene in order to see the changes in the final scene.

Exporting just the final scene file will not pick up changes made to the first scene file (the book scene).



Creating an Atgi Locator

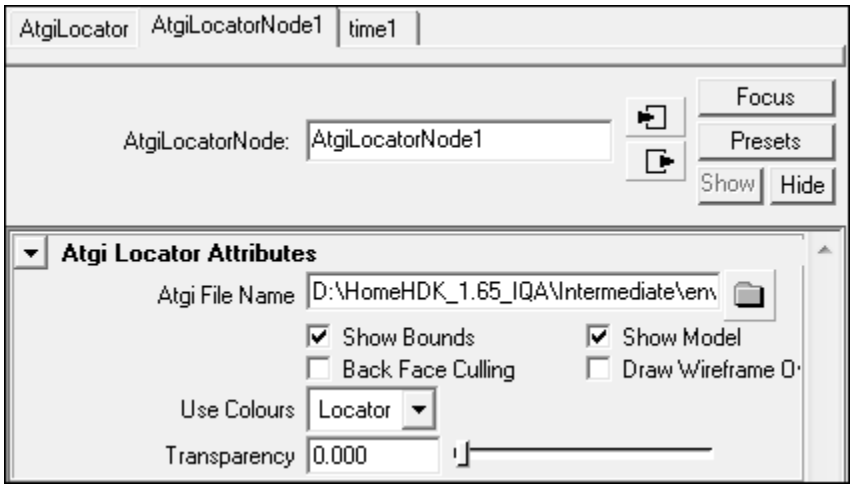
To reference an `atgi` file from a model:

Select Home > Environment > Create Atgi locator:

- With the Atgi Locator selected, open to the Attribute Editor



- Set the `atgi` file under Atgi Locator Attributes > Atgi File Name (`Atgi files are located in <HOME_INSTALL_DIR>\Intermediate`)



Important to note:

- Although the Atgi Export and Atgi Locator options are under the Home > Environment menu, you are not limited to environments when referencing assets. You can reference assets from any model.
- Creating an Atgi from a model creates an Atgi file from the current scene much like an export, but without the lighting processes. This file can then be referenced in another scene using an Atgi Locator.
- Animation is not supported by Atgi locators.
- Atgi locators do not create instanced geometry. They add the referenced geometry to the scene. Adding multiple copies of the same Atgi creates unique geometry for each one.
- Baked Lighting is not taken from the Atgi file. It is re-calculated from the scene the Atgi locator is added to. However, Lights and Lighting settings are carried across from the Atgi file. For example a street lamp with a light placed for the bulb still has a Maya light when the Atgi is referenced into a different scene via an Atgi locator.

Example Usage

You have a library scene. You want to line the walls of the library with 10 bookcases.

You have four options.

Not instanced:

If you want to have something that is non-instanced (for instance if each of the 10 bookcases has to be unique):

- Option 1: Using separate models, model the 10 bookcases into the scene.
- Option 2: Make a bookcase object (for instance a furniture item) and embed the object 10 times in the library scene by adding a scene object component to it (see Embed Objects into Scenes for details).
- Option 3: Make a bookcase model and export it as an atgi file, then place 10 AtgiLocators in the library scene and reference that bookcase atgi file from each of the AtgiLocators.

Instanced:

If you want to have something instanced, use entities called with script:

- Option 4: Create a bookcase entity and call this entity 10 times using a Lua script.

All information is baked into the scene on export. This process therefore exports unique model data for all geometry in the scene. This saves no memory - it is purely a process improvement and a benefit to your workflow. You still pay for the visible meshes in game, but you can reuse the same model file for all 10 bookcases. This saves you effort and allows you to reuse assets more easily.

Creating Particle Locators

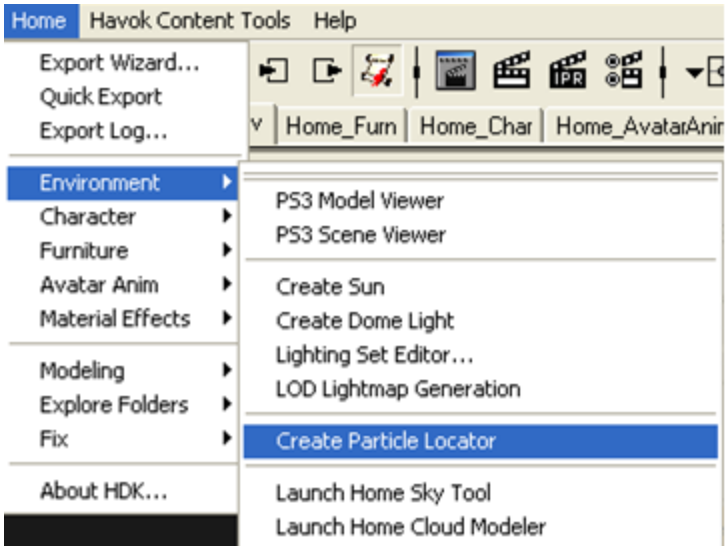
Particle effects for a scene can be waterfalls, raindrops on water surfaces or butterflies. After you have created a particle effect, you can incorporate it into an environment. For information on creating particle effects, see ATG Particle Effects Tool.

You can insert particle effects in an environment in Maya by placing a particle locator in the scene. The locator references the Particle Effect file created using the Particle System Tool. Placing a particle locator in an environment in Maya results in the particle effect being exported as part of the .scproj file.

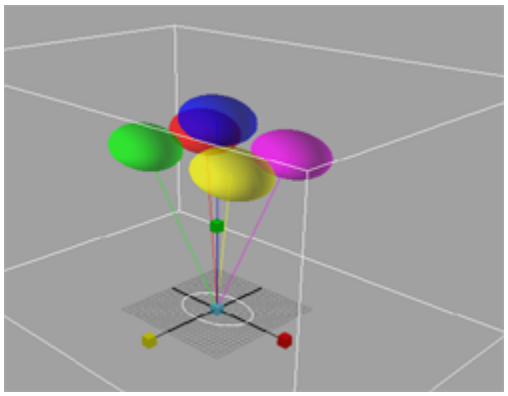
Assigning a Location to a Particle Object

To assign a location to a particle object:

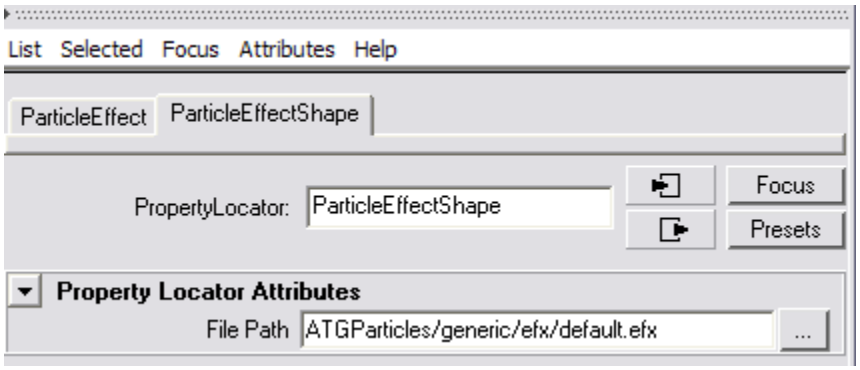
1. Click  in the Home_Env tool shelf or select Home > Environment > Create Particle Locator:



A particle locator appears in your workspace:



2. To modify the particle object's properties, use the following dialog available in the Attribute Editor in the ParticleEffectShape Tab:



3. To select the File Path, click  and select the location of the .efx file you want to use.

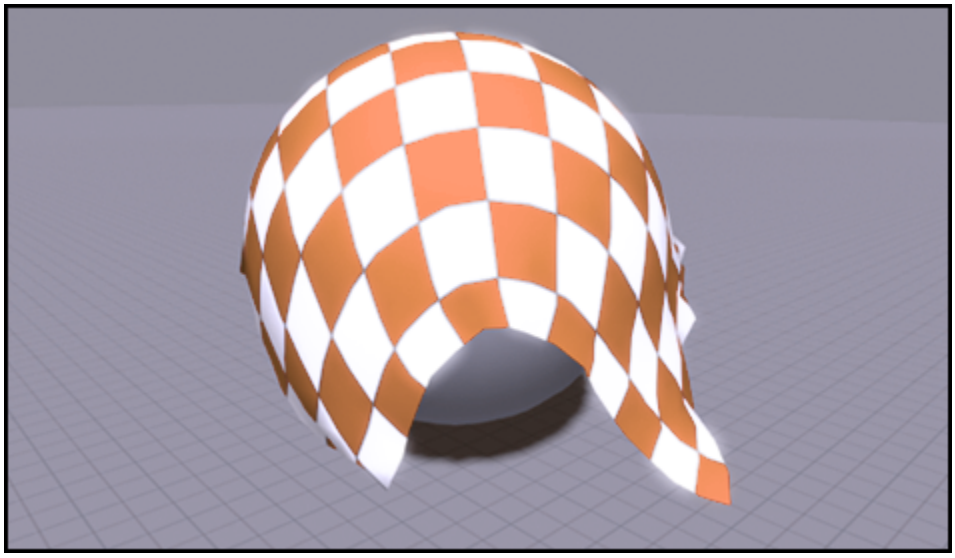
Cloth Simulation for Meshes

Scene meshes, Prelit meshes and Objects can now simulate cloths to a degree. With the cloth effect you can, for example, create flags rippling in the wind, or leaves rustling on trees. The cloth effect is applied to meshes, not shaders, meaning many shader types can be applied to a mesh that has cloth attributes.

> ⚠ The only shaders that do not work with cloth are blend and prelit_blend. For more information see the Shader Reference section.

You can download a Cloth sample from https://home.scedev.net/projects/samples that showcases this feature.

Applying the Cloth Attribute to a Mesh

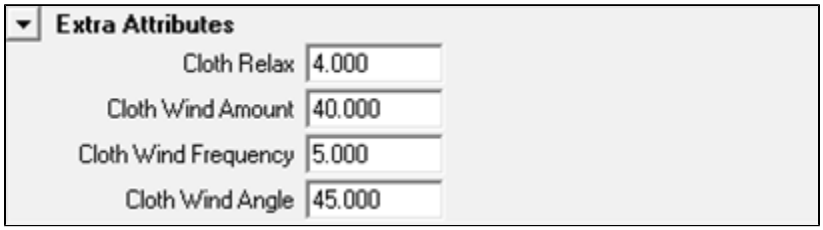


To add the cloth attribute to a mesh in Maya:

1. Select the mesh.
2. Go to Home > Cloth > Add Cloth Setup.
3. Display the Attribute Editor > Extra Attributes and set the cloth attributes.
4. Select Color > Apply Color to apply vertex color. This tells the client how to render the cloth in PS Home.
5. Test the scene in PS Home.

> When using the cloth attribute, you can use only 1 color set on each mesh.

Cloth Attributes

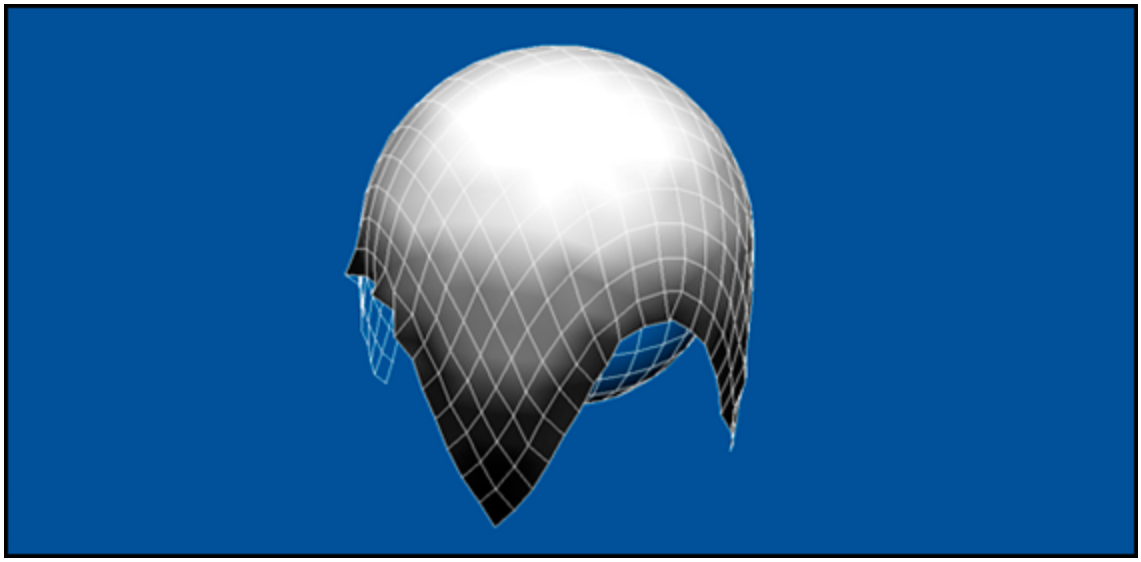


The following attributes are available:

Attribute	Description
Cloth Relax	Adjusts the stiffness/stretchiness of the cloth. This works in conjunction with the vertex color.
Cloth Wind Adjustment	Adjusts the wind speed; i.e. the higher the number the greater the speed, and thus effect.
Cloth Wind Frequency	Adjusts the random movement of the cloth. Increasing the frequency makes the cloth ripple more.
Cloth Wind Angle	Represents the wind direction in degrees. (Each cloth mesh can have a unique wind angle. The wind angle does not need to be the same across all cloth meshes in a scene.)

You can edit and adjust these attributes in real time using console commands. See Live Editing.

Vertex Color



Vertex color tells the cloth how to simulate in PS Home.

The image above shows an example of cloth hanging off on a sphere with the vertex color applied to the mesh.

To apply vertex color in Maya, select Color > Apply Color.

When painting vertex color onto the mesh, the color affects how much the vertices of the mesh are affected by the cloth simulation. White will have no effect and black will be fully affected.



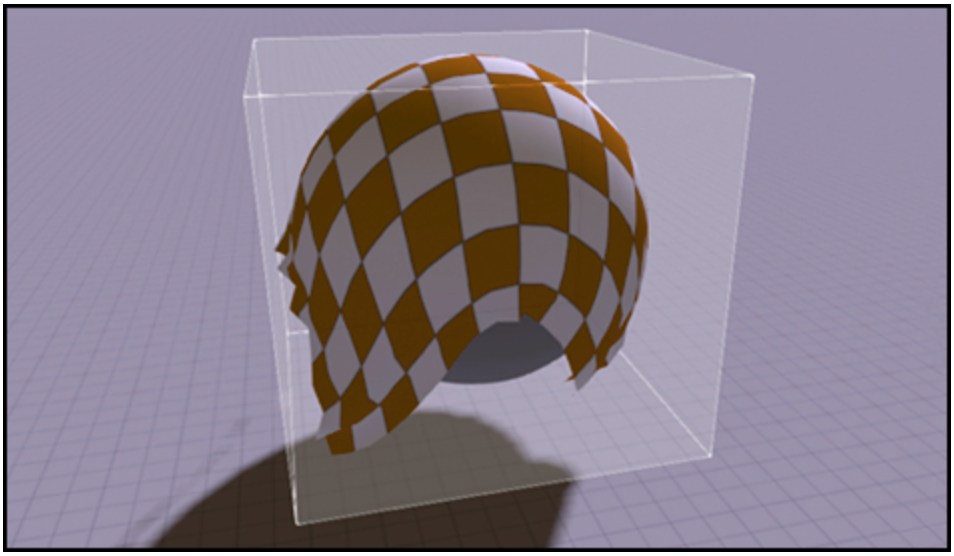
The vertex color works in conjunction with the stiffness of the cloth. For example, if you set Cloth Relax to a very low value (1 or smaller for example) the mesh will be very 'stiff' and the vertex colors have little influence (on black vertices) or no discernible influence (on white vertices) on the cloth effect.

It is often best to use a larger cloth relax and then control how the cloth looks by adjusting the vertex color and wind values.

Only one color set is allowed on each mesh when using the cloth attribute. If there is more than one you will have to remove them.

Duplicating a mesh does not copy vertex colors.

Live Editing



You can edit the cloth attributes added to the mesh in real time in game using the following console commands:

Console Command	Description
ClothSelectNext	Steps through the cloth meshes in the scene. Once selected, a bounding box appears around the cloth object. The remaining console commands will only work when a cloth object is selected by this command, and will only affect the object in the bounding box.
ClothRelax	Sets the cloth relax value.
ClothWindAmount	Sets the cloth wind amount value.
ClothWindFrequency	Sets the cloth wind frequency value.
ClothWindAngle	Sets the cloth wind angle value.

For example:

```
--Select the first mesh cloth object in the scene.
ClothSelectNext

--Change the Cloth Relax value to 3, the Wind Amount to 20, the Wind Frequency to 10, and the Wind
Angle to 90.
ClothRelax 3
ClothWindAmount 20
ClothWindFrequency 10
ClothWingAngle 90
```

⚠ Live editing does not save the values. It only allows you to see how the meshes will render with the given values. When you are finished live editing, make a note of the final values, and then enter them into the meshes in Maya.

Static Lighitng (Prelit Meshes)

Static Lighting

Static Lighting is used for:

- Environments
- Prelit Models

Both environments and prelit models are almost exactly the same and are created in the same way, they differ in how they are finally used within PS Home.

Environment models are assigned to a scene file.

Prelit Models can be used in all the same places as other models, e.g. placed in a scene file or spawned via Lua.

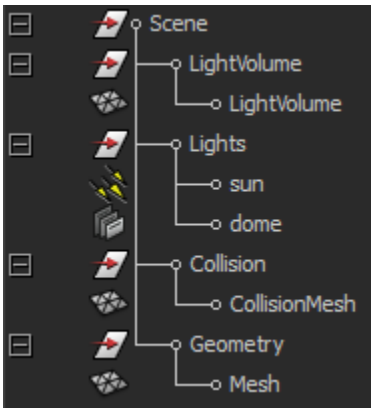
For more detail on these scene types see:

Maya Environment Scene

Maya Scene Prelit Model

Adding Lights for Static Lighting

Maya lights are added to the Lights group within Maya. All prelit objects will need to have a Sun and Dome light in the scene.



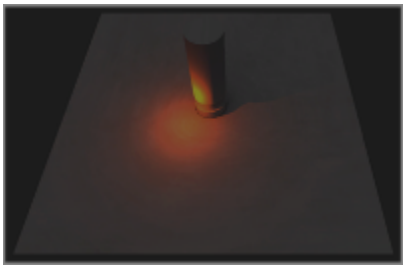
The Sun and Dome can be added using the Home menu or toolbar.



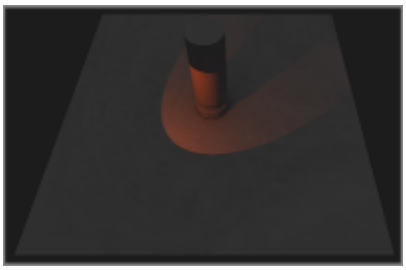
All other lights are added from the default Maya menus as normal.

Create > Lights > Light Type

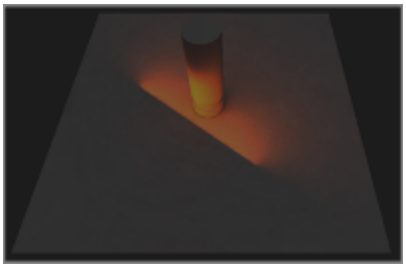
Below is an example of the various Maya lights in PS Home and how they affect the environment.



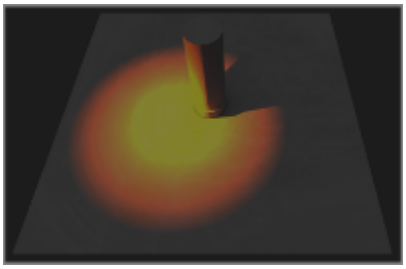
Point Light



Spot Light



Area Light



Volume Light



Directional Light

For more information on the different types of lights you can use in Maya see Maya Lights

> Nova Lights
> Note: Additional Attributes can be added to lights to give better results when exporting. Nova Light Attributes

Setup Meshes for Static Lighting

Meshes with Static Lighting will need to have one of these prelit shaders assigned:

- Prelit Default
- Prelit Glass
- Prelit Blend

In most cases Prelit Default is a good starting point.

Prelit UV Sets

The mesh will also need a 2nd UV set named (map2) for the baking of Lightmaps. These UVs will be used to generate any lightmaps for the model, it is important to lay them out efficiently to get the highest quality of lighting possible.

For more information see UVs and Lightmaps

> Vertex Lighting
> Note: Objects that are vertex lit will still need a UV set 2 although it will not have any effect on the final result of the lighting.

Lightmaps or Vertex Lit

The two options for static lighting are Lightmaps or vertex lighting. In general terms these options break down like so:

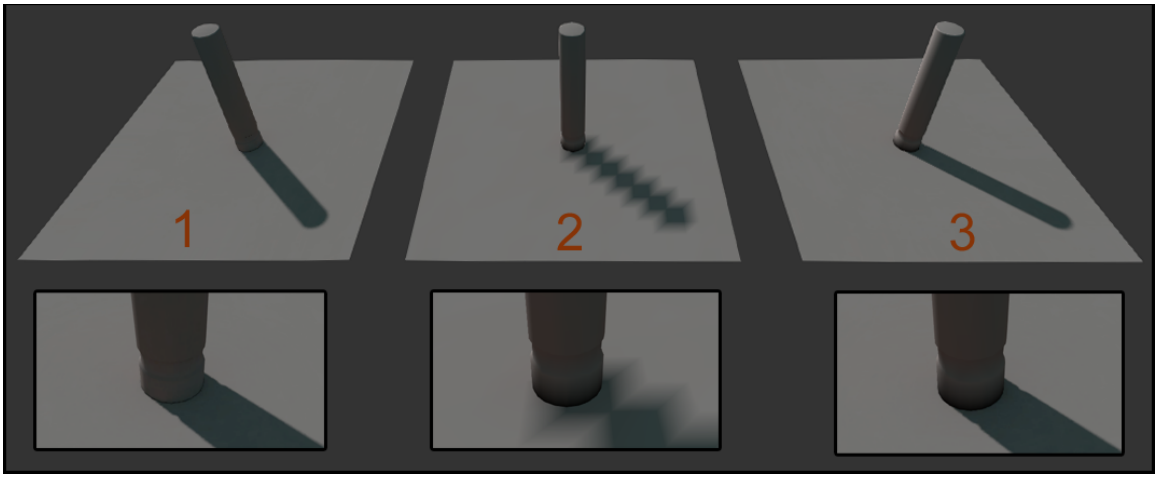
- Lightmaps = More detail but also more Vram usage (dependent on quality settings)
- Vertex Lighting = Less quality (dependent on geometry density) but very cheap and no VRAM usage

The image below highlights the differences in the two options.

1. Both the ground and pillar are using Lightmaps
2. The ground and pillar are both using Vertex Lighting
3. The ground is using Lightmaps and the Pillar is using Vertex Lighting.

Lightmaps are good for surfaces that will receive a shadow and will either need to be very crisp or very detailed, e.g. the shadow from a grating. Vertex Lighting is not very good for this as its resolution is limited by how many polygons and vertices are in the mesh.

What Vertex Lighting is good for is lighting objects that have many vertices or are very small and so will never need to have a complex shadow cast onto them.



The combination of Vertex Lighting (Pillar) and Lightmaps (Ground) in example 3 is a good compromise. The ground will need a high detail smooth shadow being cast from the pillar, while the pillar will need only basic lighting to define its light and dark side and has enough vertices to allow this.

Lighting Optimization

> Lightmaps Or Vertex Lit
> The choice between Lightmaps and Vertex Lighting is the simplest way to optimize your lighting, by using vertex lit objects you will save on the number of lightmaps and so reduce the scene's VRAM usage.
>
> There are a number of additional options available to you to help optimize your scenes Lighting. See Lighting Optimizations.

Adding Meshes to Lighting Sets

Lighting Sets can help to change the way in which objects are lit. For example if you want the mesh to be vertex lit instead of lightmapped you will need to add it to the vertex lighting set.

The light set editor can be found under:

Home > Environment > Lighting Set Editor

See Light Sets for detailed information.

> Light Sets and Qaulity
> There are a number of light sets that will help to improve quality in problem areas. It is worth reading though the available sets carefully to understand where best to use them.

Selecting a Lighting Format – LOM or RNM

All of the options and settings discussed above for lighting are universal. They will work (and in some cases will be required) in both RNM and LOM scenes, However, the final look of the scene can vary a great deal based on the lighting format that is chosen.

There are 2 formats of static lighting within home, LOM and RNM

- Light Occlusion Mapping (LOM)
- Radiosity Normal Map (RNM)

In basic terms, LOM is for exterior scenes where direct sunlight will be visible, RNM is for interior spaces where little or no direct sunlight will be seen.

For more details on LOM and RNM

For examples of LOM and RNM lit scenes

Switching Light Formats
Note: It is possible to export the same scene in LOM and RNM, however this will create extra data in the Build folder. If you want to switch between LOM and RNM or RNM and LOM you should delete the lightmaps folder. This will make sure that you only have lightmaps that are in use by the current lighting format.

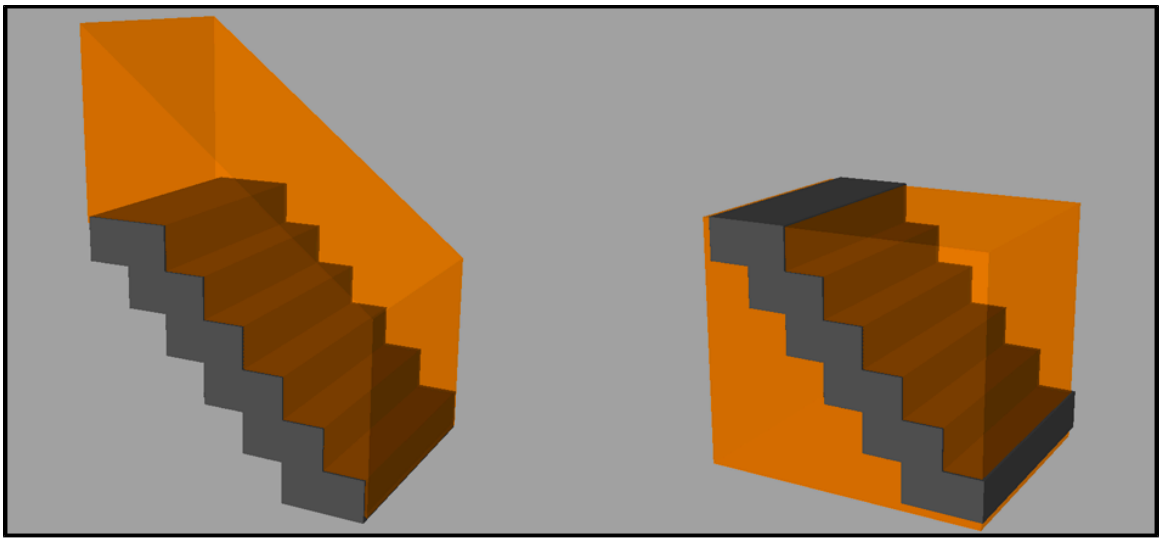
Light Volumes (Light Probes)

Light volumes are placed in the LightVolume group in Maya. The following rules apply

- No ATG shader is needed for the light volume.
- The Light volume will not be display in game.
- Light volumes are calculated at the same time as other baked static lighting.

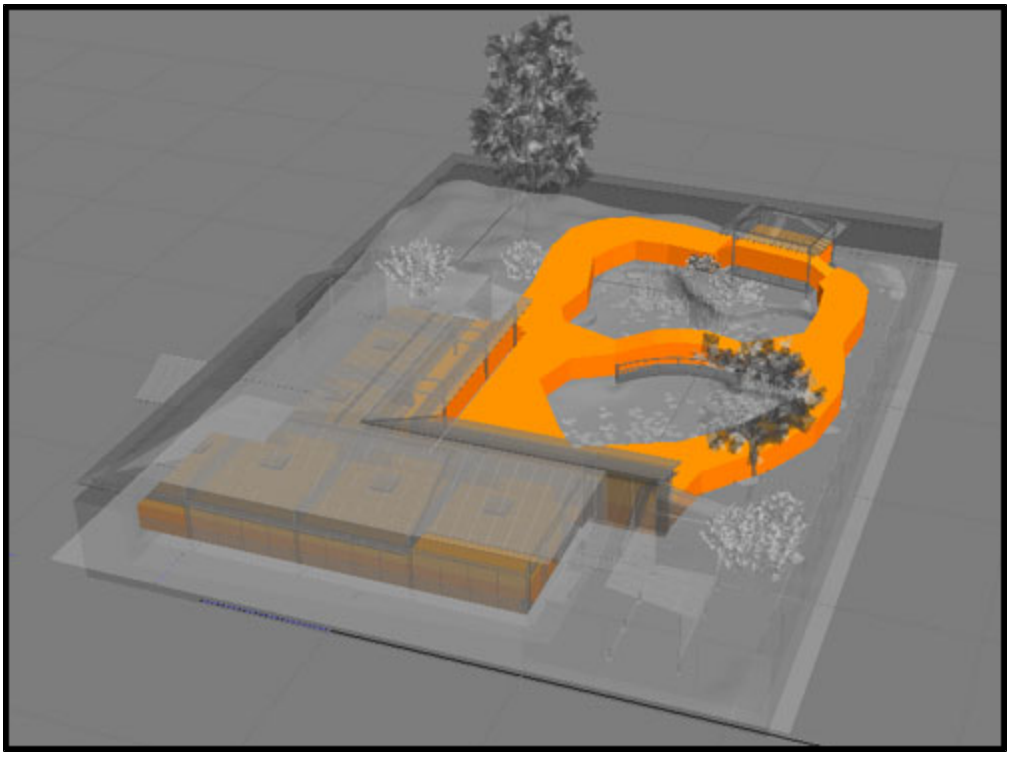
Lights are used for Lightmapping but also for generating light probe information. Light probes are points in space at regular intervals that contain lighting information generated by Nova. This lighting information is the same as that generated to light the static geometry, so that it matches up. Each probe contains directional as well as color and intensity information to determine how brightly a dynamic scene object (such as a character or any other dynamic object) will be lit as it moves around the environment.

You need to specify to the exporter the volume inside which these probes are generated. In other words, you need to indicate the volume inside which characters and other moving objects will be contained. At its most basic a single, two meter deep box, encompassing your entire environment could suffice. However, depending on the design of your scene, this could result in very long export times and be wasteful as lighting data is calculated for areas where the dynamic characters will never go.



For more complex scenes, try and ensure that the light probe generation geometry follows the scene geometry roughly. For instance, just using a flat box on a flight of stairs will cause some probes to be generated under the stairs, and therefore in total darkness. The character may well pick up it's lighting from one of these probes as it walks up these stairs, and turn black.

Below is an example of the correct, economical use of light volume geometry. Shown in orange, it only exists where the character is able to walk - along the paths, and inside the building.



Export Settings

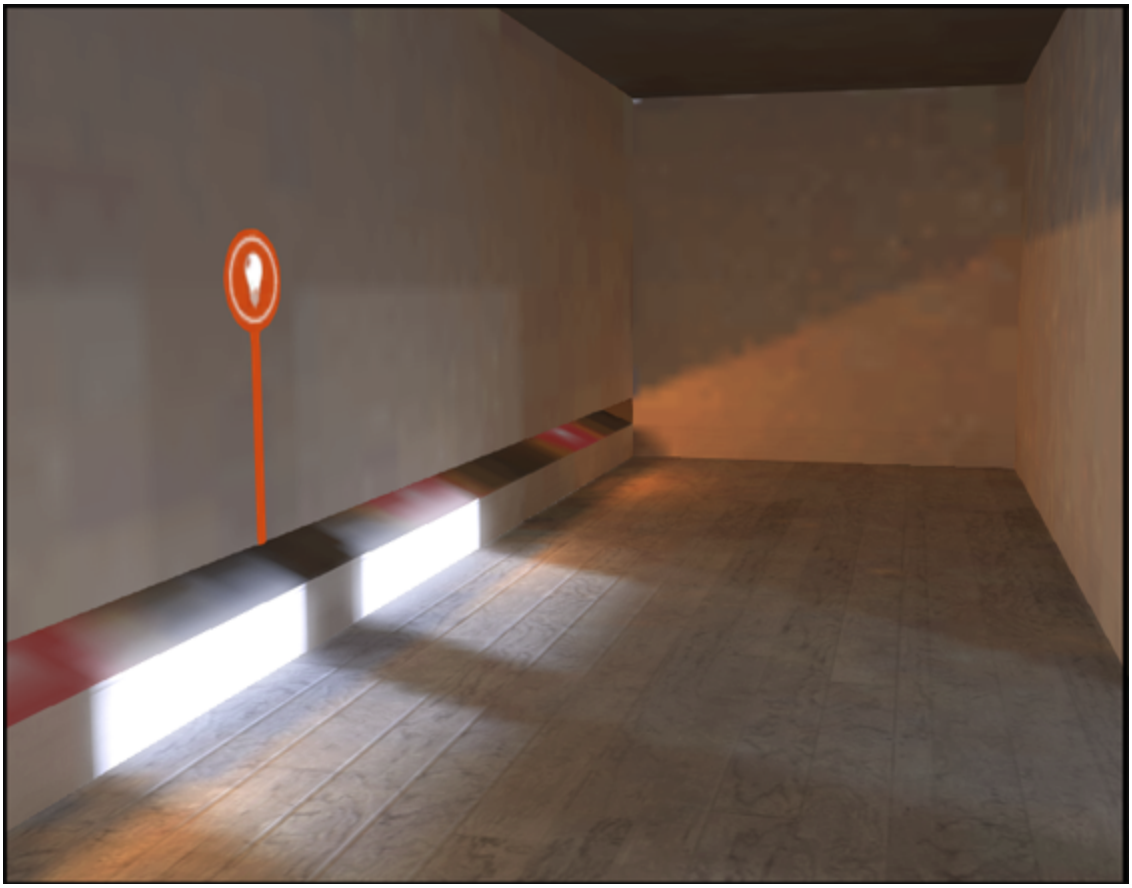
Information on specific static lighting export settings can be found here:

Static Lighting Export Settings

The images below show three ways of setting up lighting for export.

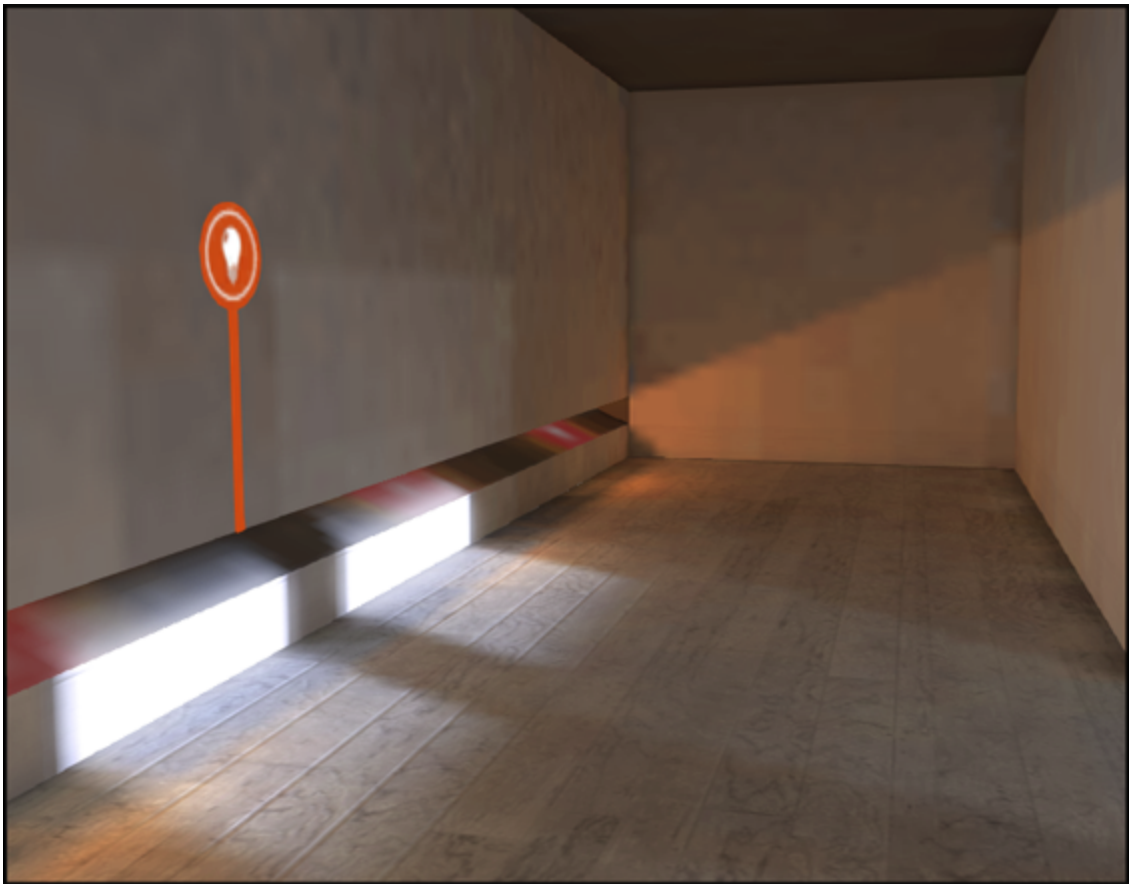
The image below shows a scene lit with Photon Mapping; blotchy light maps are produced. Photon Mapping is also slow when compared with Light Cuts.

Since HDK 1.75 this method is not available; Light Cuts is comprehensively superior.



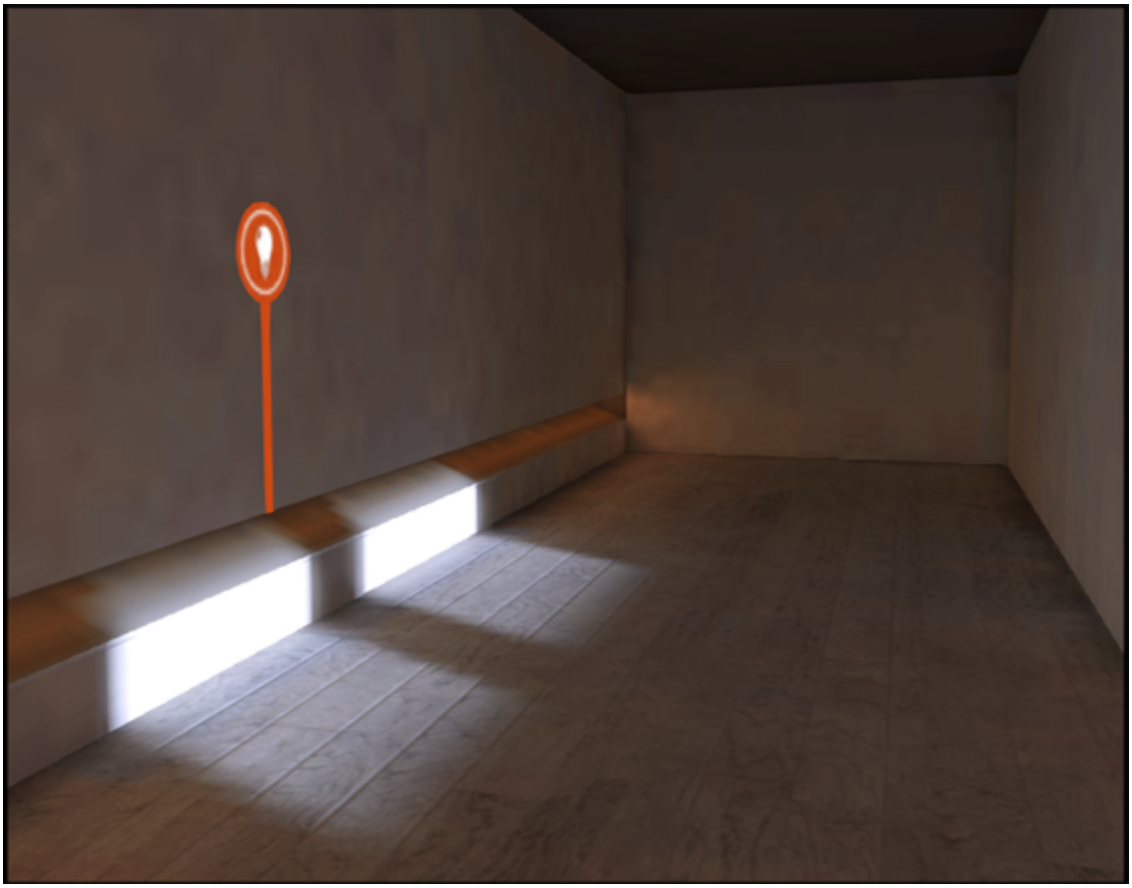
The image below shows the same scene lit with Light Cuts. The light maps are cleaner with fewer artifacts.

The lower lights are still very harsh, however.



The image below shows the same scene as the previous ones, but this time lit using Light Cuts with the addition of Nova attributes on the Lights.

This produces a much softer lighting effect.



Default Light Profiles
Note: To achieve the best final look it is advisable to use Light Cuts and Nova Light attributes on your Maya lights. This will help to improve the quality of the lightmaps but also the simulation of light effects (such as bounced light) to achieve a higher quality result.

Related Subjects

Lightmap UV set 2

Lighting Set Editor

Maya Lights

Lighting Tab

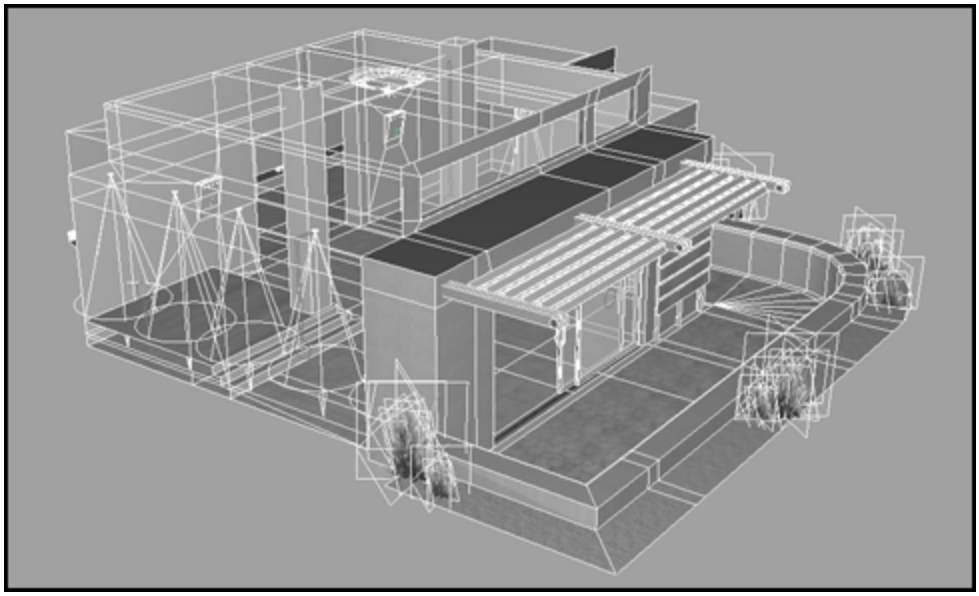
Static Lighting Optimization

See more information on Optimizing Static Lighting Effects

Lighting Optimization

Static Lighting

Lightmap optimization and combining



⚠	WARNING: mesh BalconyObjectShape covers 30.9% of the lightmap texture
⚠	WARNING: mesh ExteriorFrontageShape covers 48.5% of the lightmap texture
⚠	WARNING: mesh DoorFrameDoorShape covers 37.2% of the lightmap texture

Using the Combine light set to combine multiple small object onto a single lightmap can save a great deal of memory, there is no need to have many small lightmaps for smaller objects.

Although the total resolution used may be the same after you have combined the maps together the cost of loading each map will be removed. On a large and complex scene this can save you memory.

Vertex Lighting

For objects that are small of have a dense mesh vertex lighting is an extremely cheap alternative to lightmaps. Not only is vertex lighting cheaper but in some situations it can produce a superior look.

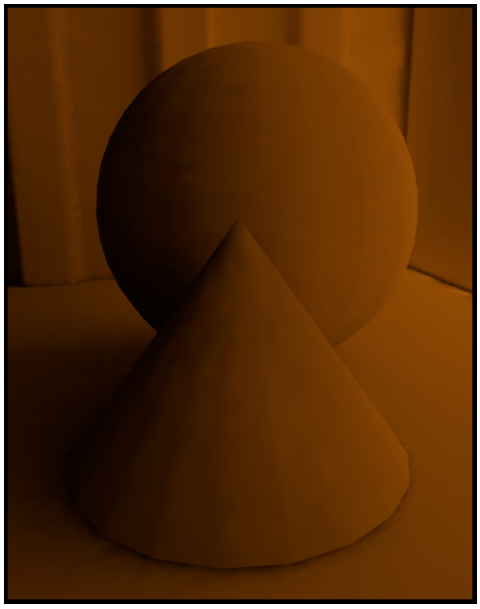
Below is an example of the use of vertex lighting.

The first image shows lightmaps. These have caused a tessellation effect based on the shape of the underlying mesh.

The second is vertex lit. This provides a much smoother result for the rounded shaped. However the cone has not shadowed properly. This is due to the lack of geometry. There is a single vert at that top. This can only be a single color.

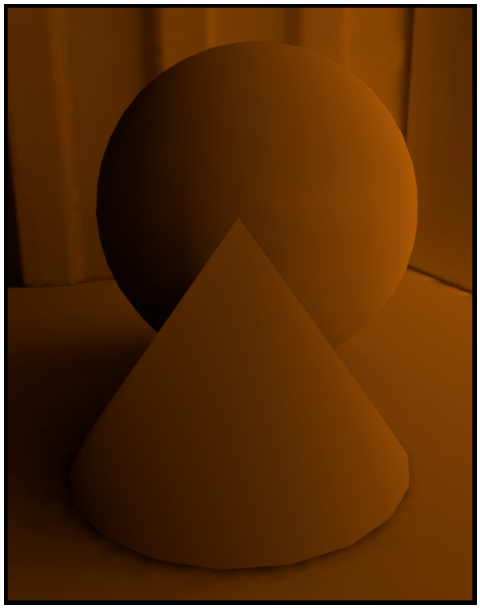
The final image is also vertex lit, additional geometry has been added to improve the shadowing. It can often be cheaper to add a small quantity of geometry and then to vertex light the object in stead of using lightmaps.

LightMap



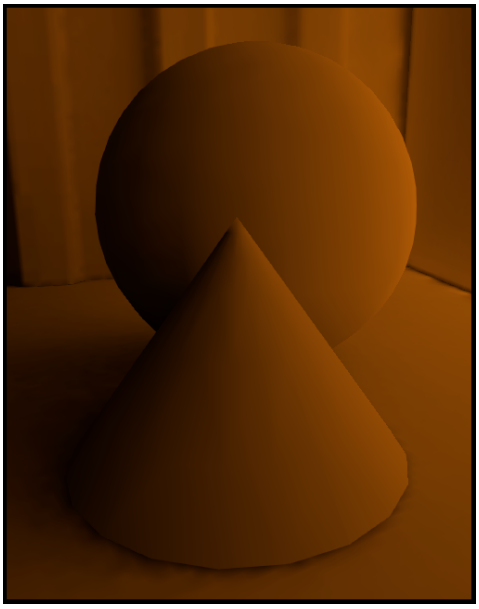
The first image shows lightmaps. These have caused a tessellation effect based on the shape of the underlying mesh.

Vertex Lit



The second is vertex lit. This provides a much smoother result for the rounded shaped. However the cone has not shadowed properly. This is due to the lack of geometry. There is a single vert at that top. This can only be a single color.

Vertex Lit (Extra Geometry)



The final image is also vertex lit, additional geometry has been added to improve the shadowing. It can often be cheaper to add a small quantity of geometry and then to vertex light the object in stead of using lightmaps.

Emissive Shader

The Emissive material does not receive any lighting at all. This can be used to your advantage if you need or have geometry that will not want to be lit or that you want complete control over.

Dynamic Lighting

Dynamic Lighting v Static Lighting

When designing an environment or an object, the choice between static and dynamic lighting may not be as simple as it would initially appear.

While static lighting is generally more efficient than dynamic, there are situations when it may be better to use dynamically lit objects.

Static lighting results in unique geometry with unique lightmaps. This can be expensive in terms of total memory for the package as all geometry becomes uniquely stored data. A form of instancing can be used to help reduce total data for the package.

By creating items such as trees or other props as separate models and lighting them dynamically its possible to then place these objects multiple times in the scene editor. While this won't make any savings in terms of rendered meshes, it will reduce the overall cost of the scene and also make updating artwork simpler.

It can also be worth considering dynamic shadows for collections of small objects that will need detailed shadows. This may seem counter-intuitive, but creating a high detail lightmap for them could be hard, whereas dynamic shadow culling distance can be set. By doing this you can create a situation in which objects very close to you are casting high detail shadows but objects further away have none. By doing this you remove the need to create a large number of high detail lightmaps for small objects.

Lods - Dynamic to Static

Lods with distant baked lods (vertex lit or lightmapped) and close up dynamically lit geometry can be created. This can allow you to have a scene with dynamic elements such as the sun with all the benefits of a statically lit environment.

The main issue with using larger quantities of dynamic objects or scene geometry is the lack of lighting when dynamic shadows are culled at distance.

By adding in Lods with baked lighting at the correct distance its possible to create the illusion of unbroken lighting and hide the switch between dynamic and static lighting. The other optimizations involved with Lods are also gained.

Alpha 1 Bit v semi-trans

When creating objects that will be dynamically lit choosing the correct alpha mode for them is important.

In general terms there are two options:

1. Semi-Trans alpha
   This alpha mode will not cast dynamic shadows if set on a dynamic material. It will often sort incorrectly but is also cheaper to render.
2. 1 Bit alpha
   This alpha mode will cast and receive shadows if set on a dynamic material. This can be very useful for dynamic objects such as plants and trees. This mode also sorts more effectively against other alpha and scene elements. This mode is the most expensive however, especial when multiple layers are put together.

For most situations where a dynamic object is needed 1 Bit alpha is the best choice as it will reflect the correct lighting when dynamic.

Light Sets – Shadow on / Shadow off

It is also possible to force objects to cast or not cast dynamic shadows. This can be used to create interesting situations such as and emissive object that will cast a realtime shadow. It can also be used to help optimize scenes.

For example

A scene that contains grass has the grass set to be 1 Bit alpha.

By making the grass 1 Bit alpha it will be able to receive dynamic shadows but will also cast a dynamic shadow. This can create a lot of lighting and be very expensive.

By setting the shadow casting "off" the effect of having the grass receive shadows is maintained, but the cost of casting shadows is removed.

Light Probe Dynamic Lighting

Dynamic Objects & Light Probes

Dynamic objects (such as characters, furniture and game objects like pool balls or racing cars) take their lighting from a series of light probes. These light probes are regularly spaced points in an artist-defined space that contain directional lighting information. They are not just points of brightness and color – they also impart lighting direction.

> You do not need to set up any additional or special lights to create dynamic light probes. They use exactly the same lights placed in Maya as the baked lighting.

Any dynamic object takes its lighting from the nearest probe, smoothly interpolating between one probe's lighting information and the next as the object moves around the game world. The light probes are generated by SuperNova. They use the same lighting algorithms as those used to generate the baked lighting for static objects, so as to maintain coherent values between the static lighting and that applied to dynamic objects. For this reason, the probes contain indirect as well as direct lighting information.

> In the LOM lighting model, dynamic objects take their direct lighting from the dynamic light, not the probes. Using LOM, the probes provide indirect lighting information and light occlusion data in exactly the same way as the LOM lightmaps do.

You can use Lua to:

- Load or unload light probe volumes
- Use separate light probe volumes in different areas of a scene
- Move light probe volumes around in a scene

The two Lua functions are `Scene.SetLightProbeData()` and `Scene.SetLightProbeTransform()`.

This section explains:

- Loading and unloading light probes
- Moving (transforming) light probes in a scene
- The art tools export process

Loading Light Probe Data and Assigning Slots

To load light probe data, call the function `Scene.SetLightProbeData()`. This function allows you to load light probes and place them into one of nine available slots.

To move the light probe from their original position, use Scene.SetLightProbeTransform().

Position of Light Probes

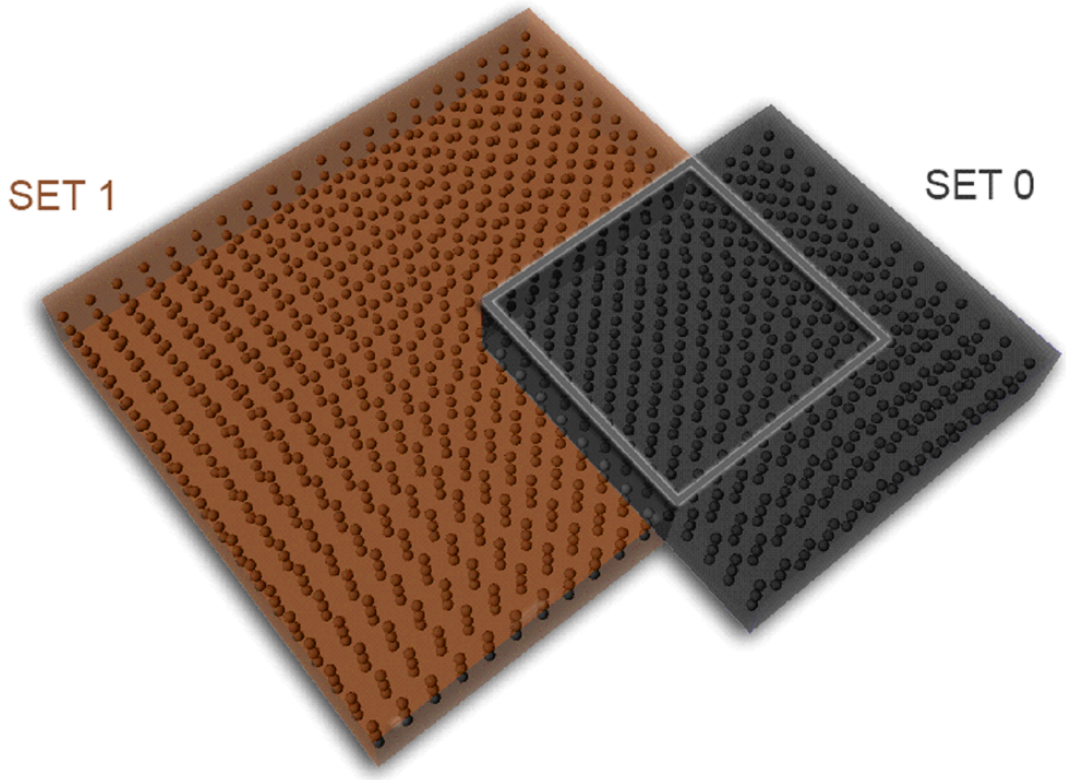
When you load light probes, they are automatically placed in the position in which they were exported. For example, if you export a probe whose center is at (1, 2, 3), it is located at (1, 2, 3) when you load it using Scene.SetLightProbeData().

Light Probe Slots

Light probe volume data is stored in slots. There are 9 slots, numbered 0-8. They are arranged by priority, from highest priority, 0, to lowest priority, 8. When probes intersect, the probe with the highest priority is used in the intersecting area.

Priority

When setting light probe volumes, you are not limited to replacing an existing volume with a new one; you can overlap volumes. In the area where two or more light probes overlap, the probe with the highest priority displays in the overlapping area. This is useful if you have one volume that covers an entire scene, but you want to load or unload a volume in one small area.



In the image above, there are two light probe volumes: the orange volume, which is in slot 1, and the gray volume, which is in slot 0. Where these two volumes overlap, the gray volume is used because it has a higher priority.

Light Probe Transforms

You can move a light probe using the function Scene.SetLightProbeTransform().

When exporting probe data, you can export the volume in its correct world space position. For example, if you export it at (1, 2, 3) in Maya, it loads at (1, 2, 3) in PS Home. However, its origin is always at (0, 0, 0).

Note the correct world position of a light probe before exporting. There is no way to query a light probe's location from the client.

Transforming at Correct World Position

When you transform a light probe that was exported in its correct world space position, its origin is also at the correct world position. Any transformations executed using `Scene.SetLightProbeTransform()` requires an offset.

For example:

1. A light probe is exported at (1, 2, 3).
2. In the client, a script loads the light probe in slot 0.
3. The light probe is loaded into the scene at (1, 2, 3).
4. When loaded, you want to move it to (2, 3, 9). To do so, you must subtract (2, 3, 9) from (1, 2, 3).
5. The script creates a Matrix44 with the value [1, 1, 6].
6. Finally, the script calls `Scene.SetLightProbeTransform( matrix, 0 )` .

The light probe moves from (1, 2, 3) to (2, 3, 9). The `Scene.SetLightProbeTransform()` function always uses (0, 0, 0) as a light probe's origin, which must be accounted for in all transformations.

Transforming from Origin

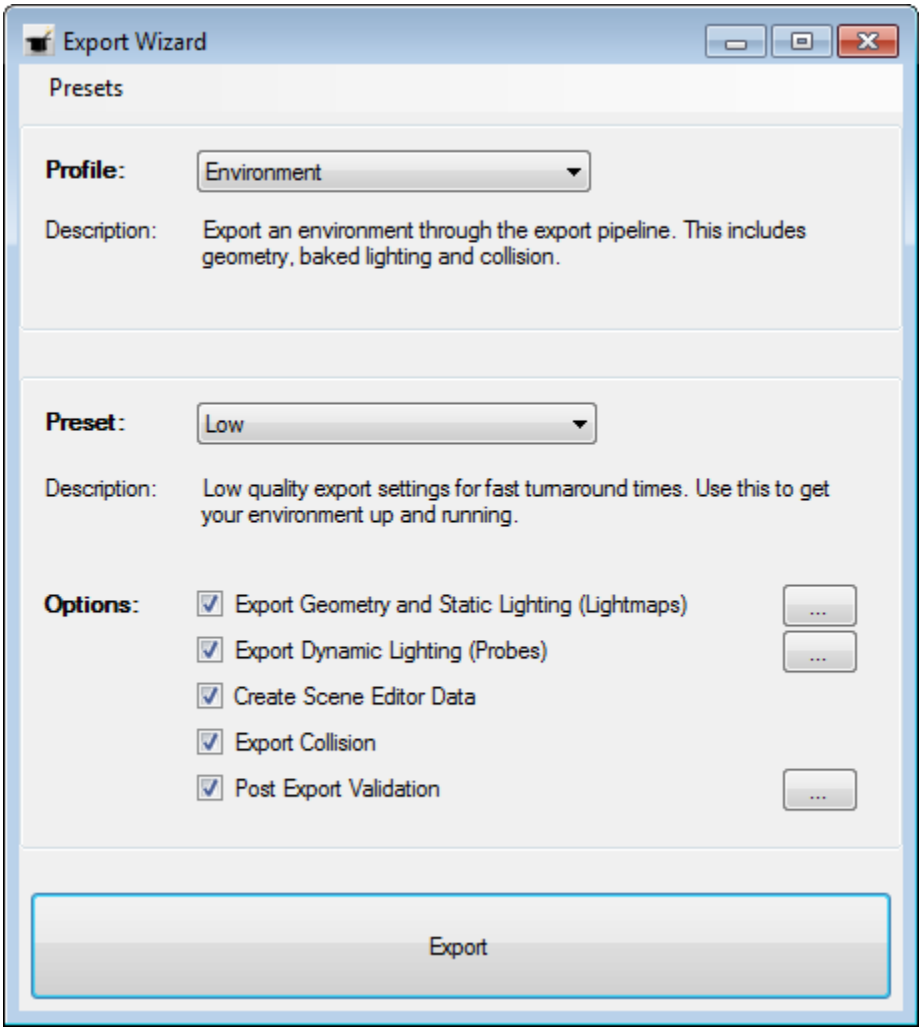
To avoid accounting for the probe's current world position, export the light data from its origin in Maya. The probe file loads at (0, 0, 0) in the client, and no matrix subtraction is required.

> This is more useful for light probes that you know you want to transform.

Exporting Probe Data (Probe Volumes)

To export Probe Volumes:

1. Launch the Export Wizard from the Home drop-down menu or by clicking  .



2. Ensure that only the Export Dynamic Lighting (Probes) box is checked.

3. Click Export. The light probe file is given the name of the scene, so if you export a light probe for the scene Test.ma, then the light probe will be called Test.probe. To create multiple light probes for a scene, create additional scenes in Maya for each probe data set and then only export the probe data.

When creating additional scenes, give the scenes descriptive names.
For example, if you are creating additional light probes for a space called Test Space, name each additional scene 'Test Space_LightProbe1.ma.', when you export the light data, the probe file name is 'Test Space_LightProbe1.probe.'.

Static Lighting Export Settings

Export Settings

Bake Method – LOM / RNM

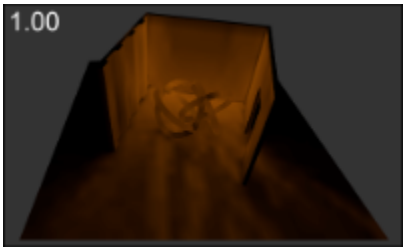
Weather the scene will use LOM or RNM. This will affect the overall lighting mode for the whole scene and dynamically lit objects within it.

More information on LOM and RNM, and examples of LOM and RNM lit scenes are available.

Texel Size

The Texel size will increase the resolution of the light map – if you make the number lower, the detail increases. The default setting for the previous exports is 0.5. For the final quality export this has been changed to 0.025, although you can go lower than this as it will generally not offer you a great deal more detail in the map.

The example below shows the increase in lightmap resolution and the subsequent increase in quality that this brings.



> Vertex Lit
> Note: Texel Size will have no effect on vertex lit geometry. Most of the options here (XXXX where is here? menu? static lighting export settings menu?) will have little or no affect on the outcome of vertex lit geometry, the quality of this lighting is based on the density of the geometry being lit.

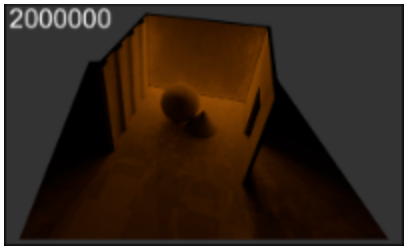
Photons – Direct / Indirect

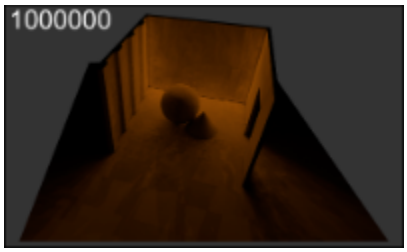
- Direct Photons – Photons that are used to calculate direct lighting. This is light direct from a source light.
- Indirect Photons – Photons used to calculate bounced light. This can be light that is bounced off of a very bright object or around a corner.

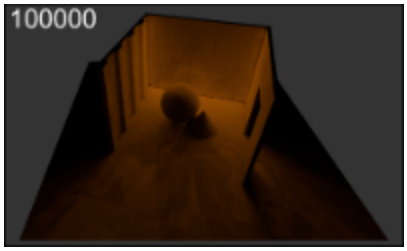
Photons for a scene are split among the lights in the scene. You may need to adjust the photon counts if you have a very large scene or a high number of lights. In most cases the default values will be fine.

- Large scenes have an absolute budget for photons(. XXXX Photons in?) Very large scenes will make for overall poorer lighting quality. Break up the distant geometry into different files and then recombine them in the Scene Editor.

The examples below show varying photon counts. You can see that the higher values are not significantly smoother than the lower. This is because the error threshold is set to high (50) and there are few lights in the scene making the increased photons of little use.







XXXX shouldn't this move to the beginning of the section? if it's legacy, they should just skip it?

> Photon Mapping Options
> Photon Mapping has been removed from HDK 1.75.

Light Cuts Options

Light Cuts is the recommended lighting mode. It will provide the best quality and speed and it fare more efficient than Photon Mapping.

Below is a brief description of the light cuts process that Nova preforms. Its not necessary to understand the complete process but a simple understanding can help you to troubleshoot light problems.

Light Cuts Process:

1. All Maya lights are converted into VPLs (Virtual Point Lights)
2. All VPLs are combined into a Light Tree
3. Cuts are made into this tree to chose clusters of VPLs
4. A Lighting sample is made for each Cut
5. The Lightmap is generated from these samplings.

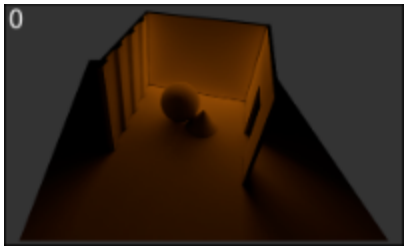
The use of Nova Lights with Light cuts will produce the smoothest look for your scene.

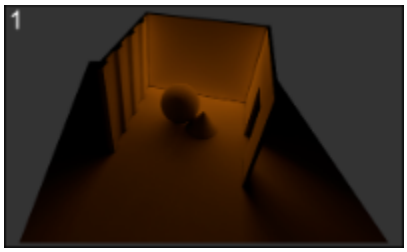
> Nova Lights
> Note: Additional Attributes can be added to lights to give better results when exporting. Nova Light Attributes

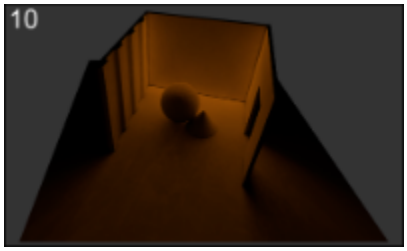
Error Threshold

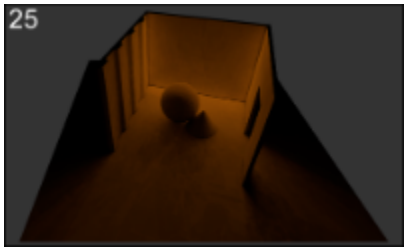
By reducing the error threshold you will increase the render time but should see a small decrease in the amount of noise in the lightmaps. This can sometimes be so small that the added time is not worth the result. It would be best to leave this setting until you are doing the final bake when the small increase in time will not be a problem.

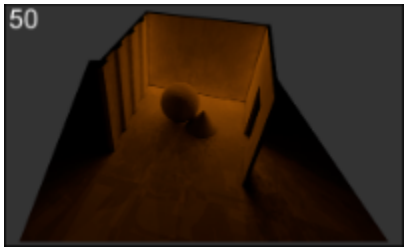
The example below shows the progression from the highest to the lowest error threshold. The quality change can clearly be seen. It can be a good idea to increase the threshold while working on the lighting to help decrease iteration times (setting it to XXXX ? lower/higher value for higher/lower quality).

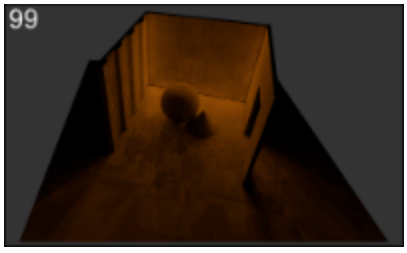












VPL Budget

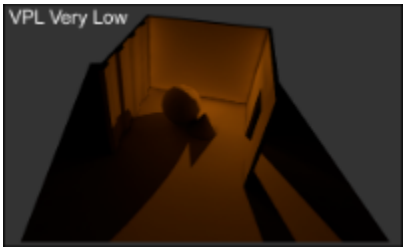
Nova converts all lights in a number of point lights, the VPL budgets is the total number of these point lights that nova can create when exporting your lighting. Increasing the VPL budget will decrease effects like banding, noise and artefact's in your final lightmaps.

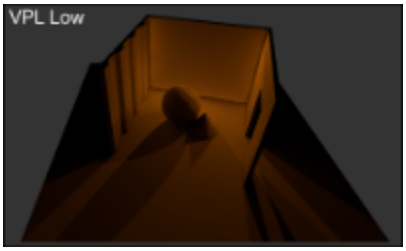
Keep in mind that the different light types in Maya use a different percentage of the VPL budget so having a large number of certain light types will result in you having to increase the budget to get the correct result. Maya Area lights are the most

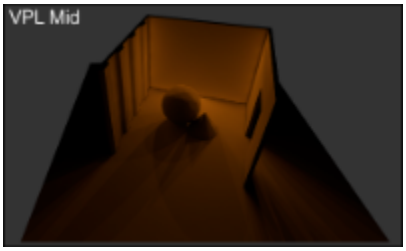
expensive in terms of VPLs.

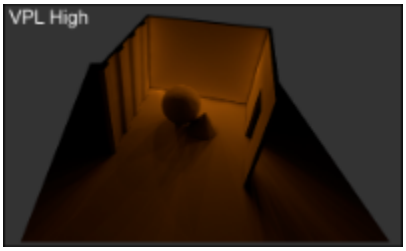
The images below show what can happen in a scene where the VPL budget is too low. The example has a large number of area lights although only one is visible. This light dose not have enough VPLs to simulate correctly.

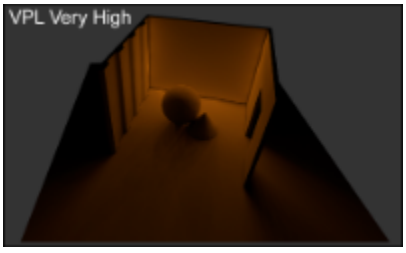
The left most image shows a single light source, the area light has been converted to just 1 VPL as the budget is increased it revives a larger number and so the quality of the lighting is also increased.











Max Cut Size

Max Cut Size controls the the size and number of cuts through the light tree. (explained above) This can affect the quality of the scene and having the value set too high or too low can increase render times.

As a general rule a setting of 1000 will be fine for almost all scenes.

Post Processing

The post processing tab allows you apply affects to the lightmaps once they have been created by Nova. The post processing

options help to blur the maps slightly to break up harsh edges and to soften noise.

The Median Filter can be applied to help smooth out the noise in the lightmaps. Keep in mind however that these effects are applied after the lightmaps have been generated and so in many cases will have a very subtle effect on the final result.

It is much better to tweak the lighting in the scene and Nova export settings to improve lighting quality.

Advanced

Distributed Light Processing

Legacy Mode

Static Lighting RNM and LOM

LOM

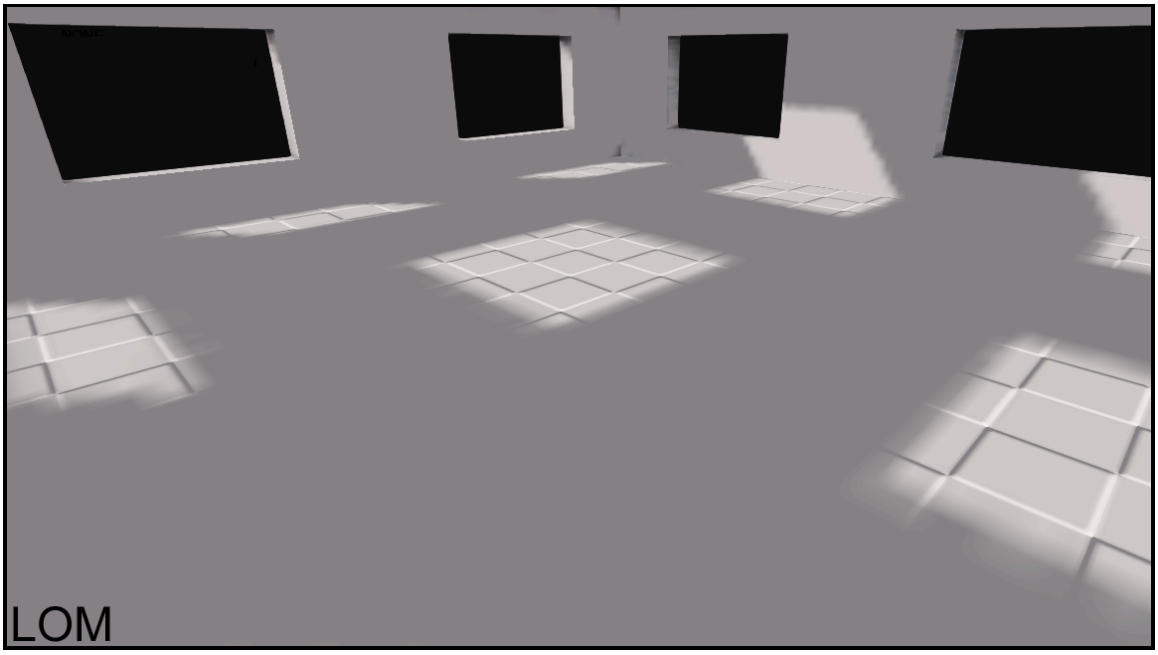
The image below shows a basic scene with the specular overridden to black and exported with LOM.

Positive:

- LOM is best used for outside spaces where lots of direct sunlight can hit surfaces.
- LOM is capable of having long detailed shadows cast by the dynamic sunlight, which can be very important for animated models and Avatars.
- LOM is cheaper than RNM because fewer lightmaps are created.

Negative:

- The main issue with LOM is the lack of normal mapped details on areas that are not in direct sunlight (light from the dynamic sun). Areas outside of direct sunlight can look flat. Note in the image below that no detail has been added to the color map. All the floor detail is handled by the normal map.



RNM

The image below shows exactly the same space as above, a basic scene with the specular overriden to black, but this one is exported with RNM.

Positive:

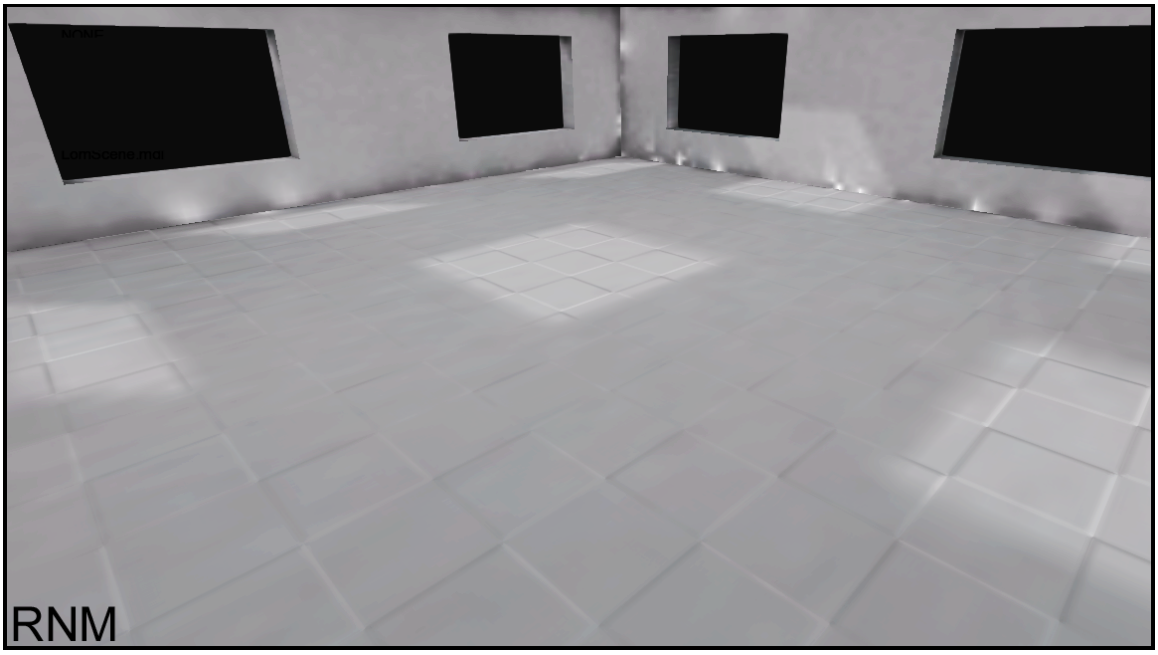
- RNM is designed for inside spaces.
- Compared to LOM, the normal maps are fare more visible (and show greater detail) in areas that are out of direct sunlight.

Adding additional lights to the scene will also affect the viable normal map details.

Negative:

Long shadows cast from dynamic objects can be cut off. RNM uses a different system for lighting dynamic objects, This uses a shadow volume that has a maximum size. If shadows pass beyond this volume they will be clipped and a hard edge will show.

RNM is also more expensive than LOD due to multiple lightmaps being generated.



Shader Reference

Shaders are a way of instructing the GPU to render effects in a specific way.

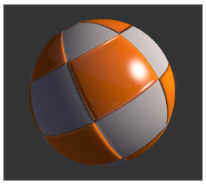
Shaders in PS Home, like Maya shaders, determine what all materials look like in game.

> You cannot use Maya's default shaders - you have to apply one of the ATG shaders from the list below.

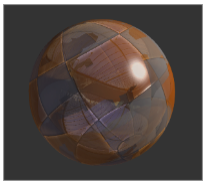
Setting parameters on shaders instructs the GPU to draw them differently.

Basic Shader Types

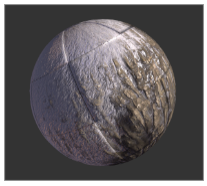
Prelit Shaders



Prelit Default

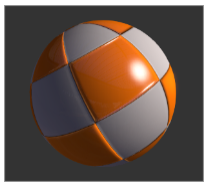


Prelit Glass

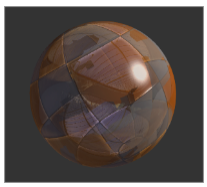


Prelit Blend

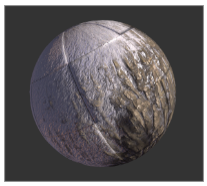
Default Shaders



Default

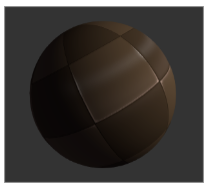


Glass

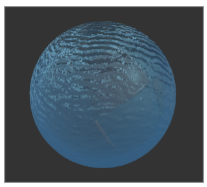


Blend

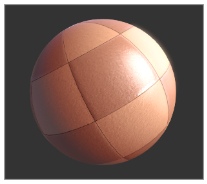
Non Standard



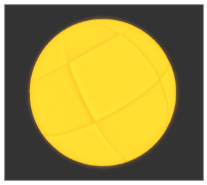
Hair



Water



Skin



Emissive

Shared Shader Attributes

Render State

Texture Scrolling and Flipbook Animation

Tinting

Lighting Attributes on the Lighting Tab

Real Time Reflections

Lua Script ID (Scripted Material Animation)

> For the shaders you can use on cloth simulation, see the Cloth Simulation for Meshes page.

Related Areas to Shaders

Blend Shader Blending

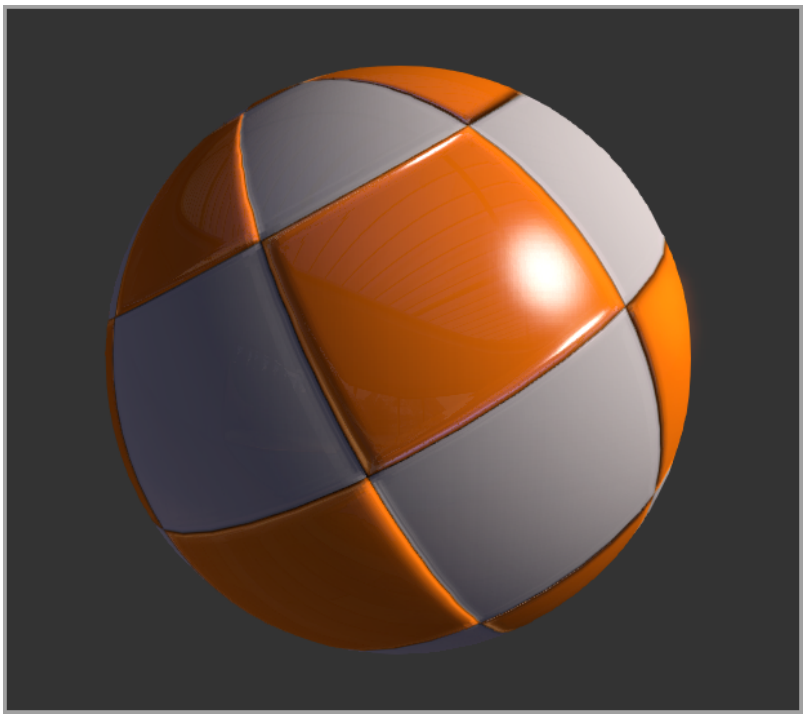
Environment Maps

Lightmap UV set 2

Prelit Default

Family: Prelit Shader

Use: general (no specific purpose)



Prelit Default shader is used by environments and prelit models.

The shader generates Lightmaps for the meshes it is applied to unless the mesh is added to a Lighting Set that prevents this.

Maps:

- Colour Map - With/Without Alpha Channel
- Normal Map
- Specular Map
- Environment Map

Requires a second UV set for Lightmaps

Additional Shader Options:

Tinting

Texture Scrolling

Real Time Reflections

Render State

Lighting Tab

Vertex Channel Usage

The vertex channel is used for Vertex Alpha

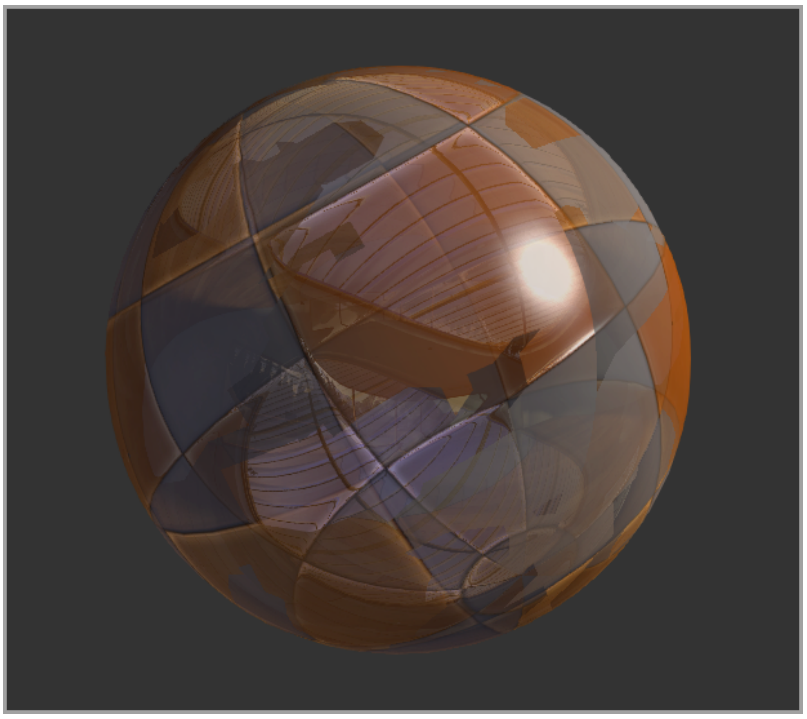
Shader Specific Attributes

None

Prelit Glass

Family: Prelit Shader

Use: for glass or reflective and/or semi-transparent surfaces (e.g. water)



Prelit Glass is for use on transparent and glass-like surfaces.

The main difference between Prelit Glass and Prelit Default are the additional Fresnel attributes. These will help control the specularity and reflectivity of the glass.

The Glass shader also uses a different method for applying specular highlights. So, on a mirror-like surface, only a single point source for the specular will be seen (unlike the default material which would have three sources).

Maps:

- Colour Map
- Normal Map
- Specular Map
- Environment Map

Requires 2nd UV set for Lightmaps

Additional Shader Options:

Tinting

Texture Scrolling

Real Time Reflections

Render State

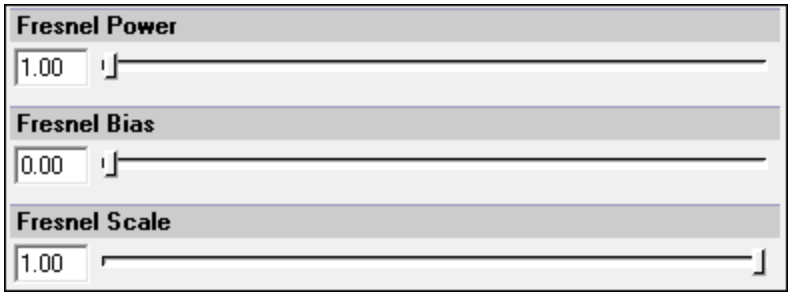
Lighting Tab

Vertex Channel Usage

The vertex channel is used for Vertex Alpha.

Shader Specific Attributes

Fresnel options will cause reflection, specularity, and other attributes to vary according to the viewing angle of a 3D surface.



Fresnel Power - The overall power of the specular reflections in the glass

Fresnel Bias - Increases angle at which the reflection is visible

Fresnel Scale - This intensity of the reflection and the specular highlight

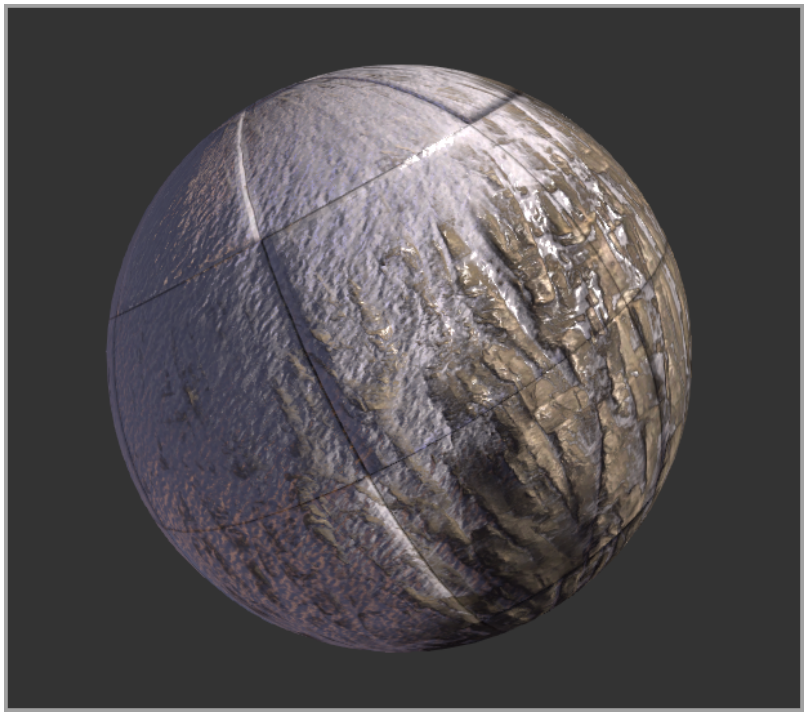
The image below shows from left to right the increase in Fresnel Bias and Fresnel Scale - creating a highly gloss and reflective surface.



Prelit Blend

Family:Prelit Shader

Use: for large areas (e.g. terrain, walls, etc.) that have tiling textures which need to be broken up naturally



Prelit Blend will allow you to blend two sets of texture maps together. This blend is controlled by the alpha channel of the first color map and attributes on the shader its self. Due to the need for two sets of texture maps and the blending effect the shader is more expensive to use than the other types, however there are many situations in which an blend effect can be more efficient that standard materials, terrain for example. Due to the use of the alpha channel for blending this shader does not support alpha texture modes.

For a detailed description, see the Blend Shader Blending process.

Maps:

- Colour Map
- Normal Map
- Specular Map
- Colour2 Map
- Normal2 Map
- Specular2 Map
- Environment Map

Requires 2nd UV set for Lightmaps

Additional Shader Options:

Shader Parameters - Tinting

Texture Scrolling

Render State

Lighting Tab

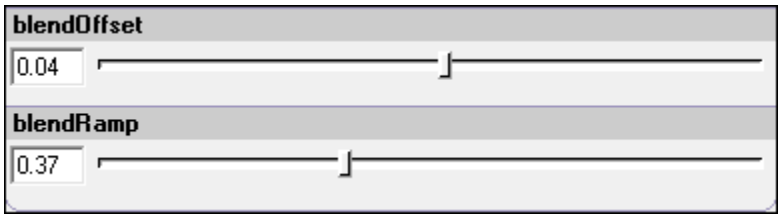
Vertex Channel Usage

The vertex channel is use for Blend Colour (IN Blend Colour)

Shader Specific Attributes

Tweaks

Tweaks will affect the overall look of the blending effect.



Blend Offset

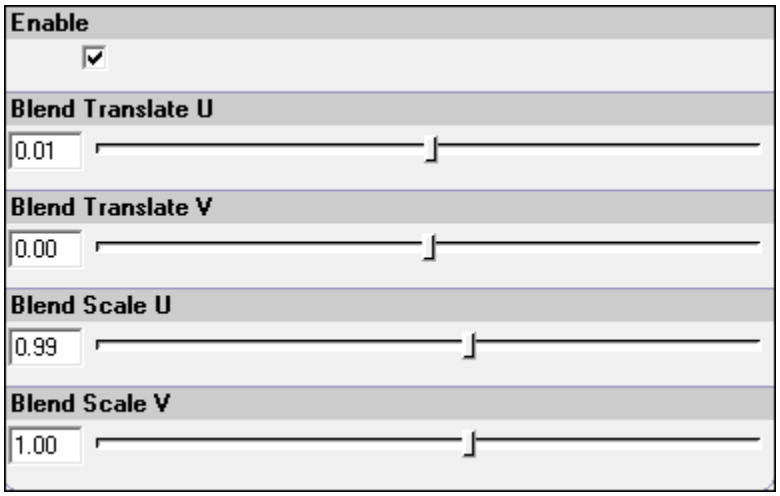
Offsets the vertex color values - moving them all towards either black or white. This has the effect of shifting which areas are blending and by how much. Extreme values will result in no blend.

Blend Ramp

Affects the smoothness of the blend in areas where blending is taking place.

UV Tweaks

The tweaks will only affect the 2nd set of textures. This is to allow for differing resolutions between the two sets.



Blend Translate U & Blend Translate V

Will translate the texture in UV space in the U and V

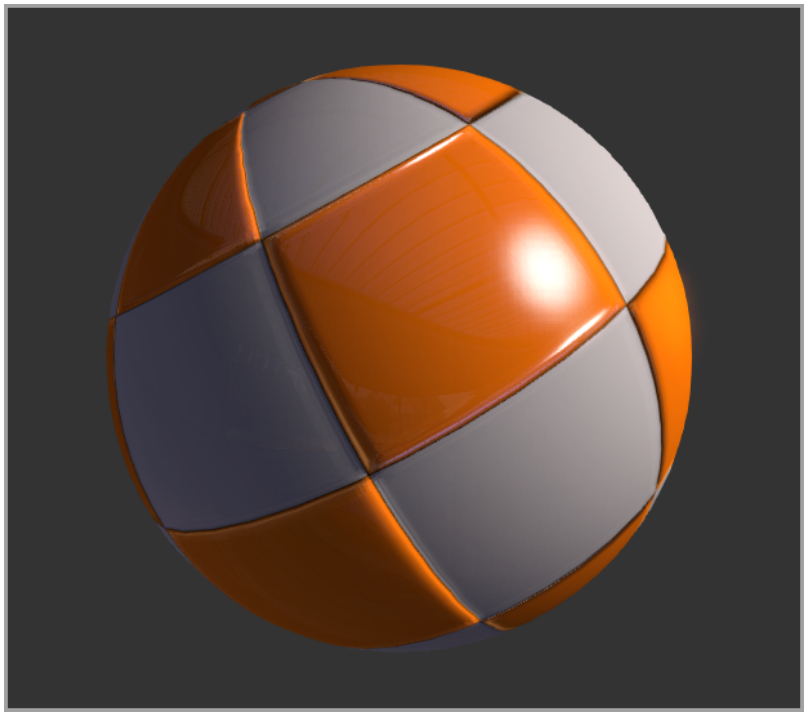
Blend Scale U & Blend Scale V

Will adjust the scale of the texture so allow for a higher tilling rate than the first set.

Default

Family: Default Shader

Use: general (no specific purpose)



Default shader is used by environments and models.

> Meshes with default applied will be lit by the light probes and cast a dynamic shadow where appropriate.

Maps:

- Colour Map - With/Without Alpha Channel
- Normal Map
- Specular Map

Additional Shader Options:

Tinting

Texture Scrolling

Render State

Vertex Channel Usage

The vertex channel is use for Vertex Alpha

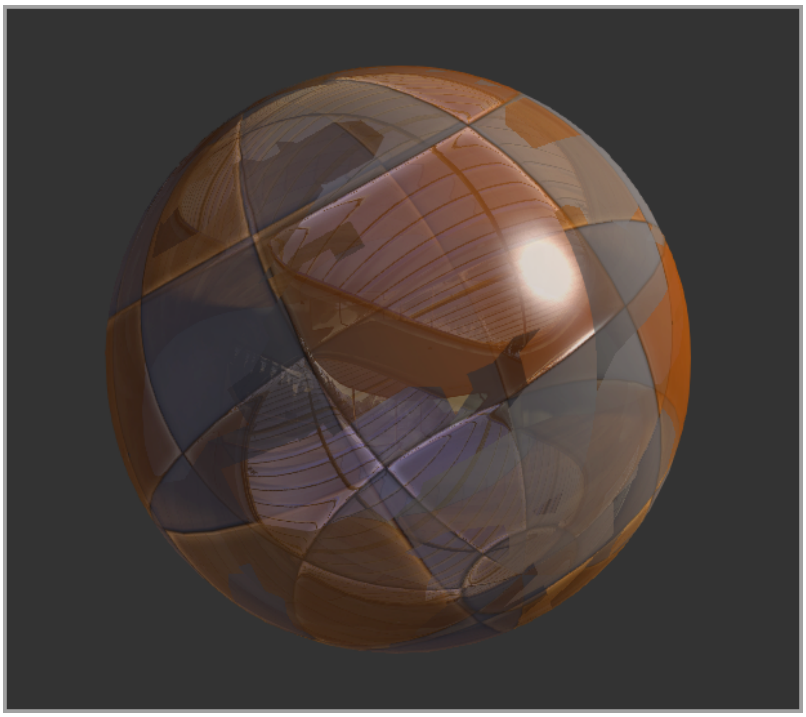
Shader Specific Attributes

None

Glass

Family: Default Shader

Use: for glass or reflective and/or semi-transparent surfaces (e.g. water)



Glass is for use on transparent and glass-like surfaces.

The main difference between Glass and Default are the additional Fresnel attributes. These will help control the specularity and reflectivity of the glass.

The Glass shader also uses a different method for applying specular highlights. So, on a mirror-like surface, only a single point source for the specular will be seen (unlike the default material which would have three sources).

> Meshes with glass applied will be lit by the light probes and cast a dynamic shadow where appropriate.

Maps:

- Colour Map
- Normal Map
- Specular Map

Additional Shader Options:

Tinting

Texture Scrolling

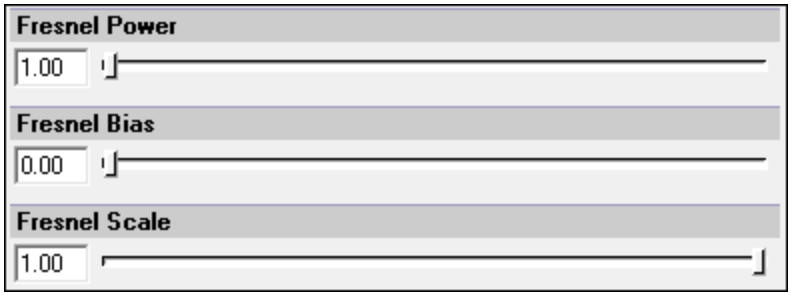
Render State

Vertex Channel Usage

The vertex channel is used for Vertex Alpha.

Shader Specific Attributes

Fresnel options will cause reflection, specularity, and other attributes to vary according to the viewing angle of a 3D surface.



Fresnel Power - The overall power of the specular reflections in the glass

Fresnel Bias - Increases angle at which the reflection is visible

Fresnel Scale - This intensity of the reflection and the specular highlight

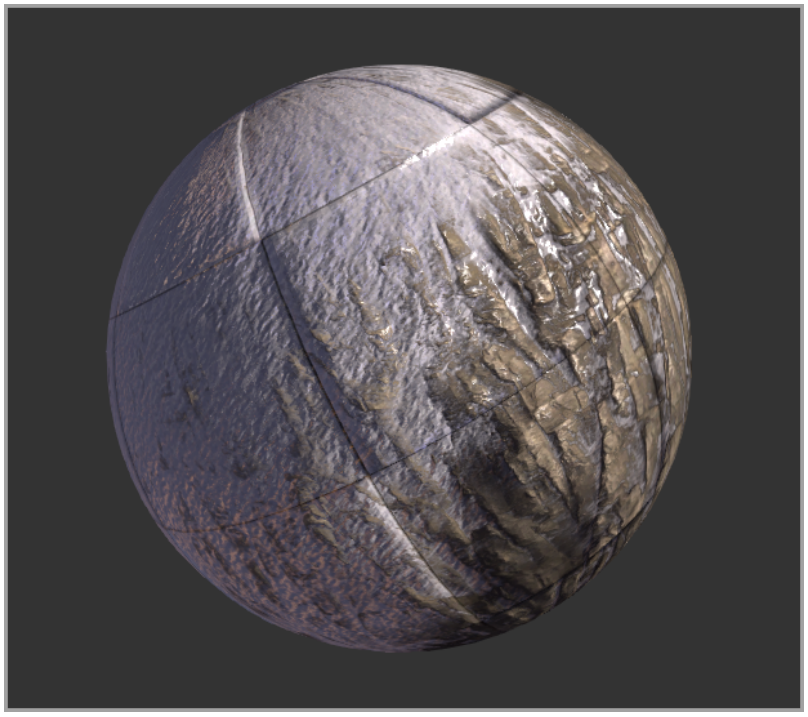
The image below shows from left to right the increase in Fresnel Bias and Fresnel Scale - Creating a highly gloss and reflective surface.



Blend

Family: Default Shader

Use: for large areas (e.g. terrain, walls, etc.) that have tiling textures which need to be broken up naturally



Blend will allow you to blend two sets of texture maps together. This blend is controlled by the alpha channel of the first color map and attributes on the shader its self. Due to the need for two sets of texture maps and the blending effect the shader is more expensive to use than the other types, however there are many situations in which an blend effect can be more efficient that standard materials, terrain for example. Due to the use of the alpha channel for blending this shader does not support alpha texture modes.

Meshes with blend shader will be lit by the light probes and will cast a dynamic shadow where appropriate.

For a detailed description, see the Blend Shader Blending process.

Maps:

- Colour Map
- Normal Map
- Specular Map
- Colour2 Map
- Normal2 Map
- Specular2 Map

Additional Shader Options:

Shader Parameters - Tinting

Texture Scrolling

Render State

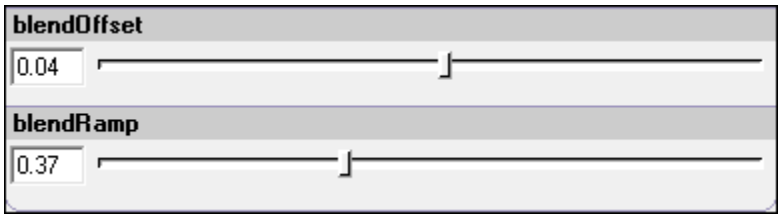
Vertex Channel Usage

The vertex channel is use for Blend Colour (IN Blend Colour)

Shader Specific Attributes

Tweaks

Tweaks will affect the overall look of the blending effect.



Blend Offset

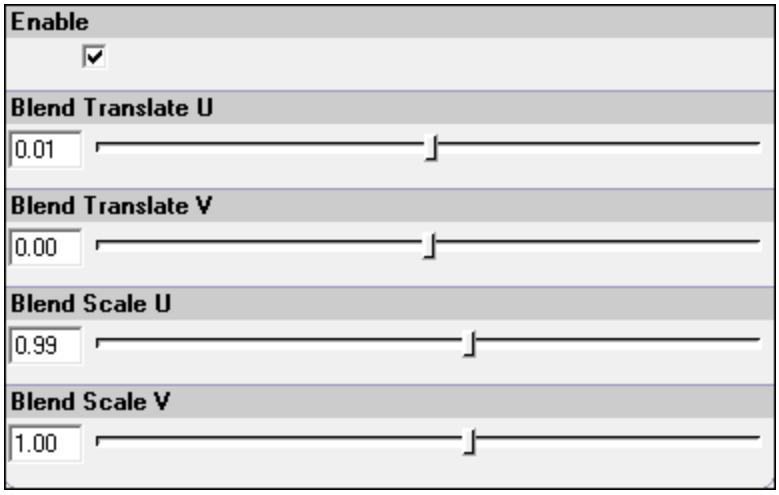
Offsets the vertex color values - moving them all towards either black or white. This has the effect of shifting which areas are blending and by how much. Extreme values will result in no blend.

Blend Ramp

Affects the smoothness of the blend in areas where blending is taking place.

UV Tweaks

The tweaks will only affect the 2nd set of textures. This is to allow for differing resolutions between the two sets.



Blend Translate U & Blend Translate V

Will translate the texture in UV space in the U and V

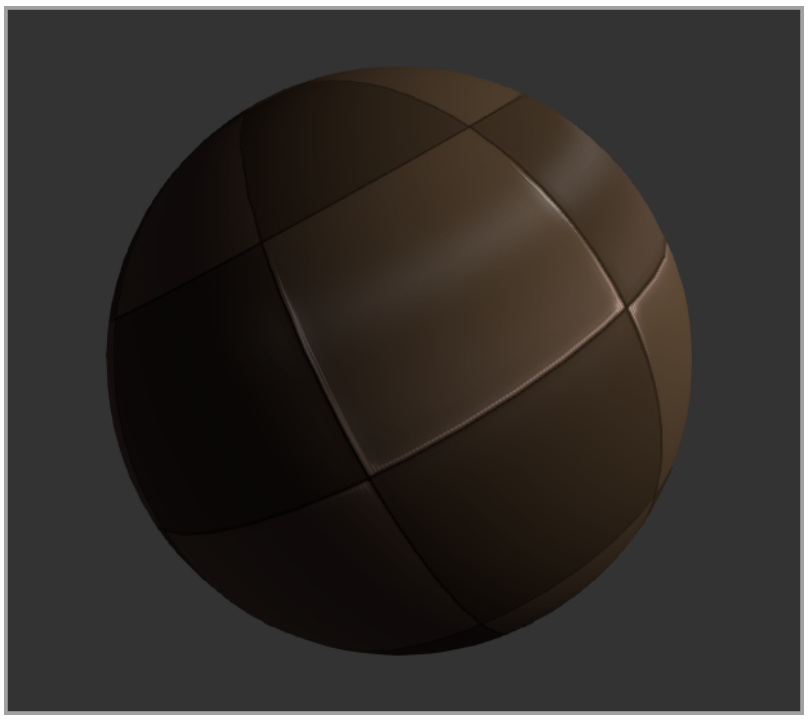
Blend Scale U & Blend Scale V

Will adjust the scale of the texture so allow for a higher tilling rate than the first set.

Hair

Family: Non-standard Shader

Use: for character hair (avatars, or non-avatar models such as animals)



The Hair shader is used for the avatar's hair and head components.

Like the Skin shader, it can change color on the avatar based on the player's selection.

The Hair shader can be use with environments and models as well, but the player won't be able to change color as it does on an avatar.

Hair does not use a Normal Map, instead it has its Alpha channel in a separate map, which allows for a higher resolution Alpha Map without an increased Color Map resolution.

Maps:

- Colour Map
- Alpha Map
- Specular Map

Requires 2nd UV set for Lightmaps

Additional Shader Options:

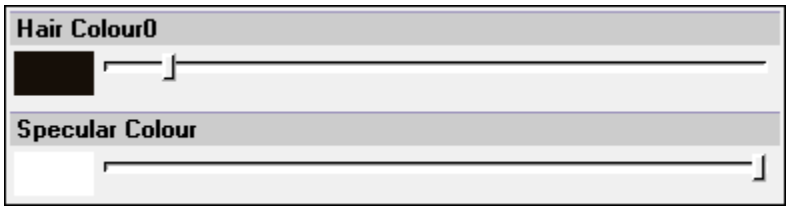
Tinting

Render State

Vertex Channel Usage

The vertex channel is use for Vertex Alpha

Shader Specific Attributes



Hair Colour - Will multiply a color value over the existing color map. This is to represent the swappable color when the shader is used on an avatar. If used on anything other than an avatar it will be as set in Maya.

Specular Colour - Will adjust the native specular values of the hair shader. The specular can also have an additional color set to it.

Character Component - Shading Groups

When using Hair on a character component, the shaders must be attached to a shading group with the folowing name:

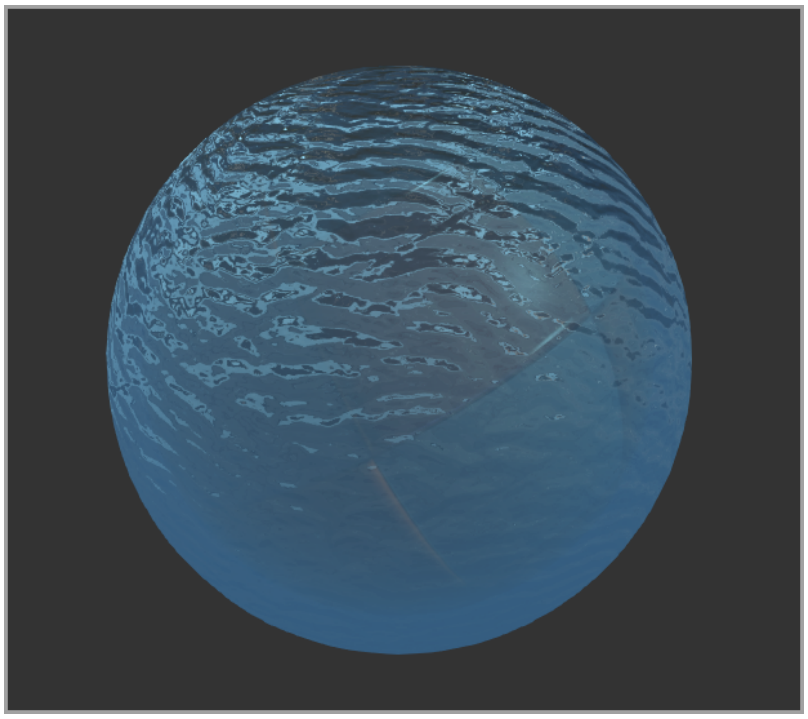
- hairSG - For use on general hair
- facehairSG - For Facial hair

This allows the shader to change color based on the users color selection.

Water

Family: Non-standard Shader

Use: for large bodies of water or other surfaces where waves or ripples are needed - generally just used for water



The Water shader is designed to simulate the moving surface of water.

There are several modes that can be used to give different effects. The main options are:

- Opaque water (a colored surface with moving ripples).
- Transparent surface water (used for shalow water where the surface underneath will be visible)
- Transparent water with Fog (where visibility in the water will be limited to a certain depth before details are fogged out)

With all of these options you can adjust the speed and look of the water surface ripples.

Maps:

- Environment Map
- Normal Map

Requires 2nd UV set for Lightmaps

Additional Shader Options:

Tinting

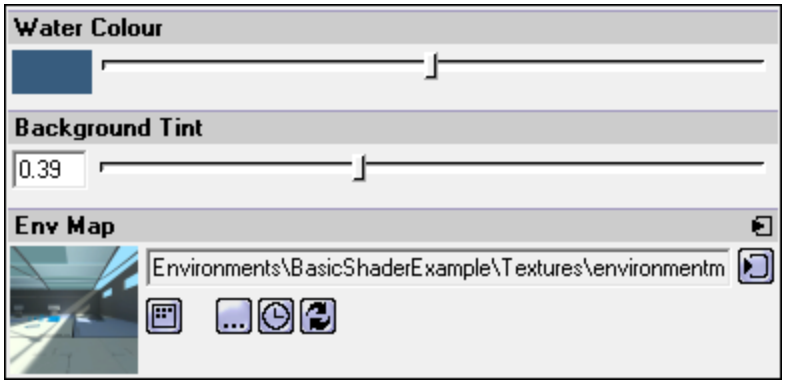
Real Time Reflections

Render State

Vertex Channel Usage

N/A

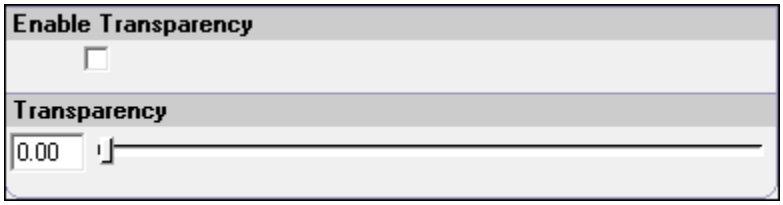
Shader Specific Attributes



Water Colour - The color of the water.

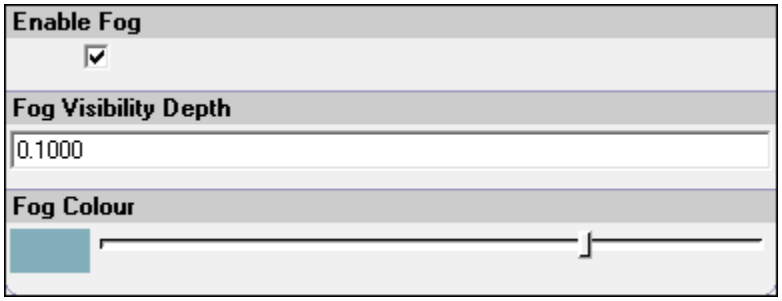
Background Tint - How visible the environment map will be in the water. Not used if real time.

Env Map - The environment map you want to see in the water. Not used if real time.



Enable Transparency - Makes the water transparent. Turn this on and set the semi-trans in the render state to prevent Fog from working.

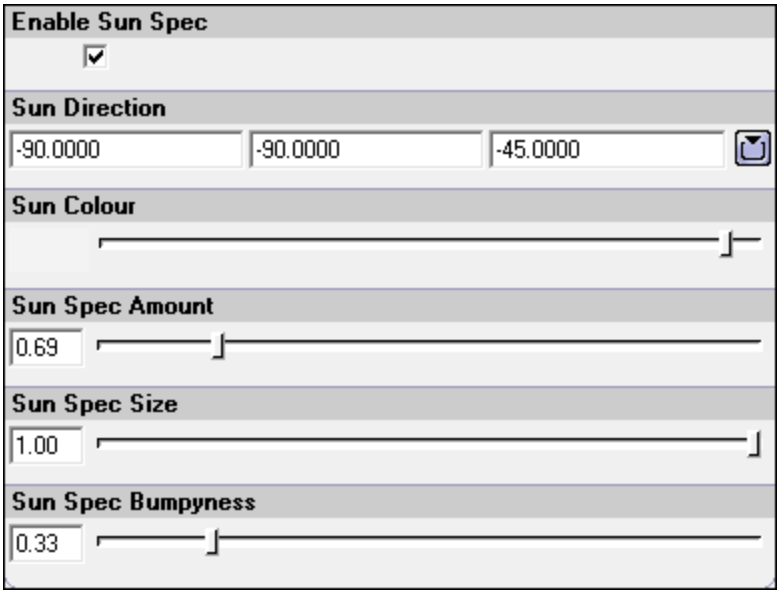
Transparency - This will adjust how transparent the water is.



Enable Fog - Will enable water fog.

Fog Visibility Depth - How deep within the water you can see before the fog will appear. This is based off of the plane of the water.

Fog Colour - The color of the fog under the water.



Enable Sun Spec - Enable the sun specular highlight

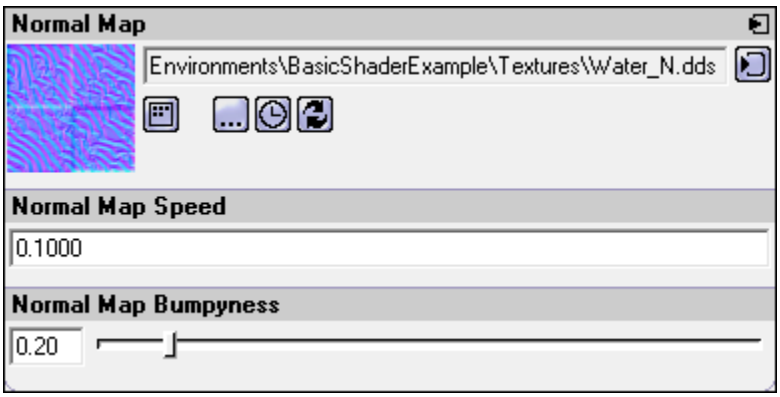
Sun Direction - The direction of the sun XYZ

Sun Colour - The highlight color.

Sun Spec Amount - The intensity of the highlight (a higher value will increase the brightness of the specular effect).

Sun Spec Size - How large the highlight is, how much of the surface of the water it will apply over. This can be more important on shapes other than flat planes.

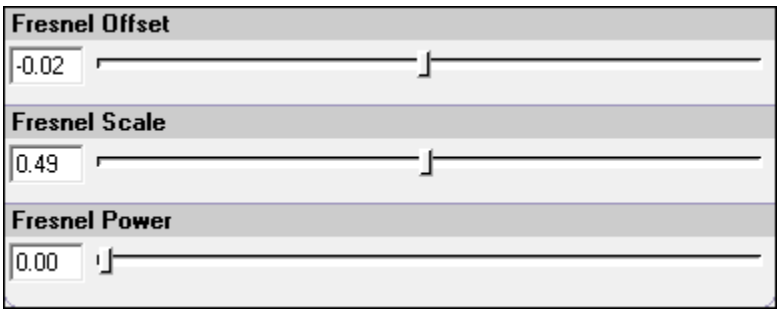
Sun Spec Bumpyness - How much the specular highlight will conform to the normal map. At a low value the highlight will simply sit over the top of the normal map a blob of light, at a high value it will be extremely broken up spread around the normal maps surface. In general a mid range value is best.



Normal Map - The normal map to use.

Normal Map Speed - How fast the UV animation effect should play.

Normal Map Bumpyness - The intensity of the normal map.



Fresnel Offset - How visible the normal map will be based on the angle from the camera. Increasing the value will result in it being highly visible.

Fresnel Scale - Will control the level of reflectivity on the object. The effect will be attenuated by the Fresnel Offset. A low Fresnel Offset will result in little effect from this attribute.

Fresnel Power - How extreme the angle based attenuation will be, a low value will make the overall effect visible from a much wider range of positions. A high value will reduce the viable effect.

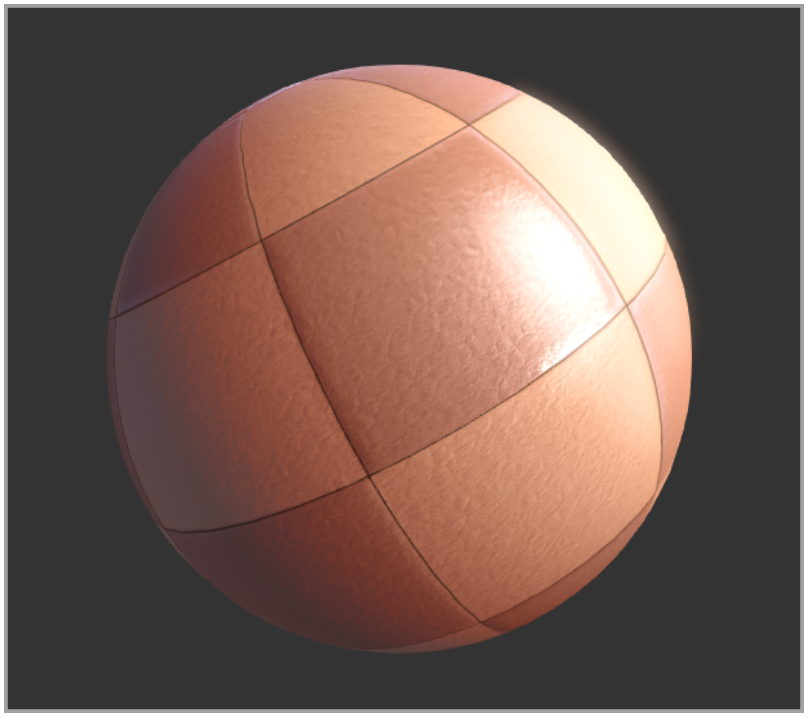
Preview Settings: These settings will control the way in which the water can be previewed in Maya.

They deal with the creation of a fake floor element that can be used to help judge fog visibility.

Skin

Family: Non-standard Shader

Use: for character skin (skin exposed in character components on avatars, or models that are not avatars where you want to apply skin)



The Skin shader is used for the avatar's skin in character components, but it can also be used in environments and on models.

Like the Hair shader, it can change color on the avatar based on the player's selection.

The Skin shader can be use with environments and models as well, but the player won't be able to change color as it does on an avatar.

Skin shaders can be connected to specific shading groups.

If your skin shader in a character component

- is connected to bodySG (for skin on the body of the avatar) or headSG (for Skin on the head of the avatar), it will give the player the ability to adjust skin color.
- is not connected to a skin shading group, you can set the color yourself (but players won't be able to).

Environment skin shaders cannot connect to shading groups.

The shading group names will vary depending on the area where you are applying the shader. Incorrect shading group names will result in the texture either not displaying correctly or custom skin color not working. See the Shader Specific Attributes below.

Maps:

- Colour Map
- Normal Map
- Specular Map

Additional Shader Options:

Tinting

Render State

Vertex Channel Usage

N/A

Shader Specific Attributes

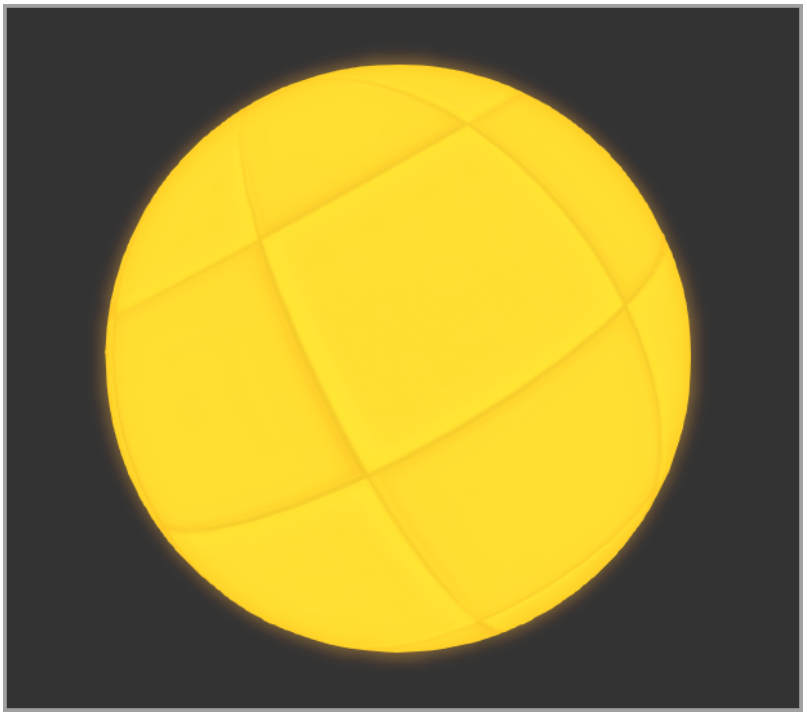
Preview Options: Maya Only

> Character Component – Shading Groups
> When using Skin on a character component, the shaders must be attached to a shading group with the folowing name:
>
> - bodySG – For skin on the body of the avatar
> - headSG – For Skin on the head of the avatar
>
> This allows the player to change the color of the avatar's skin – wherever the skin shader is correctly applied.

Emissive

Family: Non-standard Shader

Use: for glowing objects, screens, lights or effects – anything that glows and receives no lighting from the scene



Use the Emissive shader on surfaces that are glowing or will be emitting light, e.g.: a TV screen or light bulb.

Emissive materials will receive no lighting from the scene and will cast no shadows themselves. They are in essence ignored by the lighting engine entirely. The only map used by emissive textures is an emissive map, a colored map that will dictate which areas are bright or dark.

Maps:

- Emissive Map

Additional Shader Options:

Tinting

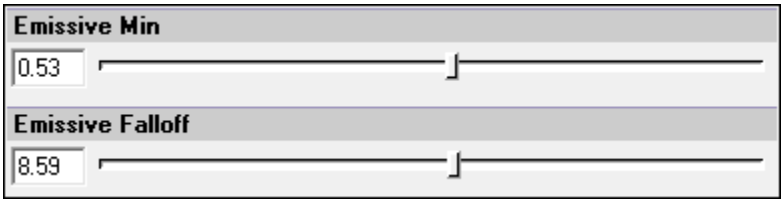
Texture Scrolling

Render State

Vertex Channel Usage

N/A

Shader Specific Attributes



Emissive Min – Will adjust the intensity of the emissive effect. The higher the value the more glow the object will have.

Emissive Falloff – How quickly the emissive glow will dissipate over the surface of the mesh.

Render State

Render State Overview

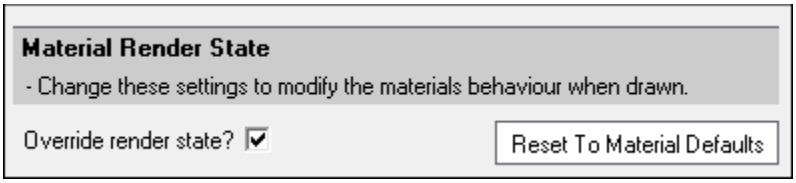
The Render State tab allows various options that control alpha sorting and other shader rendering properties, including:

- Alpha Modes
  - Opaque
  - Semi-transparent
  - Stenciled
- Z Depth
- Enable Backface Culling



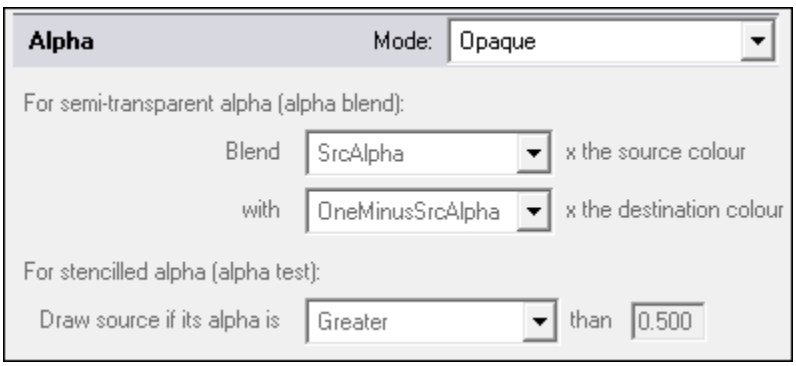
The render state tab is found on the shader.

To enable the editing of the shader render states use the Override render state check box.



Alpha Modes

Alpha modes deal with how alpha is displayed. These settings will work in conjunction with the alpha channel of the color map.



Opaque

This setting means any alpha in the color map is ignored and the material exhibits no transparency. If required, shaders in this mode can cast dynamic shadows.The image above shows the mode which is set to opaque.

Semi-transparent

When rendered, the mesh is blended with the contents of the frame buffer. Semi-trans objects will not cast a dynamic shadow.

How this blend is performed depends on source and destination factors that you can configure using the following drop-down lists. The drop-down list can be seen in the image above.

The equation for the blend is:

```
RGBA written to frame buffer = Source Factor x Source RGBA + Destination Factor x Destination
RGBA
```

Possible source blend factors are:

- One
- Zero
- SrcColor
- OneMinusSrcColor
- SrcAlpha
- OneMinusSrcAlpha
- DstAlpha
- OneMinusDstAlpha
- DstColor
- OneMinusDstColor
- SrcAlphaSaturate

Possible destination blend factors are:

- One
- Zero
- SrcColor
- OneMinusSrcColor
- SrcAlpha
- OneMinusSrcAlpha
- DstAlpha
- OneMinusDstAlpha
- DstColor
- OneMinusDstColor

It can be difficult to visualize exactly what effect each combination of blend factors will have on your material. Consider the following examples.

Example 1

- Source Blend: One
- Destination Blend: Zero
- Result: When the mesh with this material is rendered, it fully overwrites whatever has already been rendered.

Example 2

- Source Blend: One
- Destination Blend: One
- Result: When the mesh with this material is rendered, it is added to whatever is already in the frame buffer. This is also known as additive blending. It is useful for rendering objects that glow.

Example 3

- Source Blend: SrcAlpha
- Destination Blend: OneMinusSrcAlpha
- Result: This is the set of blend factors that is most commonly used for semi-transparent materials like glass. If the alpha in the color map of the material is 0.25, then 25% of the rendered mesh's color is blended with 75% of the color of what is in the frame buffer. In other words, the smaller the alpha value in the source pixel, the more you can 'see through' that mesh.

Stenciled

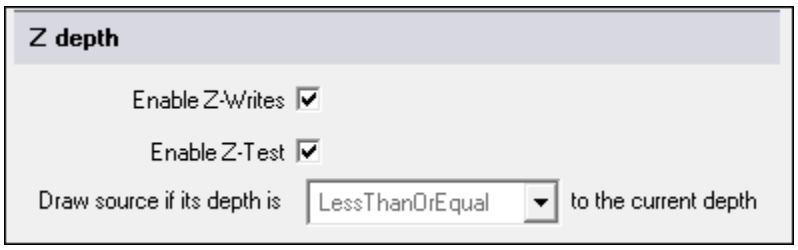
Stenciling is another name for alpha testing. This mode enables you to specify a test that says 'Write this pixel of this mesh to the frame buffer only if the alpha test passes'. The test could be 'Alpha value is greater than 0.5', which would mask out any pixels with a value of less than or equal to 0.5. Stenciled alpha is capable of casting a realtime shadow if the shader allows it.

Possible alpha test functions are:

Less	Draw source if its alpha is less than the specified reference value.
Greater	Draw source if its alpha is greater than the specified reference value.
Equal	Draw source if its alpha is equal to the specified reference value.
NotEqual	Draw source if its alpha is not equal to the specified reference value.
Never	Never draw the source.
LessThanOrEqual	Draw source if its alpha is less than or equal to the specified reference value.
GreaterThanOrEqual	Draw source if its alpha is greater than or equal to the specified reference value.
Always	Always draw the source.

Z Depth

The Z depth option determines whether your mesh gets rendered based on its distance from the viewpoint in relation to other meshes. You control it using the following interface:



Possible depth test functions are:

Less	Draw source if its alpha is less than the specified reference value.
Greater	Draw source if its alpha is greater than the specified reference value.
Equal	Draw source if its alpha is equal to the specified reference value.
NotEqual	Draw source if its alpha is not equal to the specified reference value.
Never	Never draw the source.
LessThanOrEqual	Draw source if its alpha is less than or equal to the specified reference value.
GreaterThanOrEqual	Draw source if its alpha is greater than or equal to the specified reference value.
Always	Always draw the source.

You will almost always want to have Z-test enabled, with a LessThanOrEqual test function set. This means that objects closer to the viewpoint are rendered over those further away.

The Enable Z-Writes check box dictates whether the z-buffer is updated when rendering the mesh to which the material is assigned. You normally leave this function enabled because it allows the rendering engine to give the illusion of depth in your scene, with different objects appearing to be occluding each other based on how far they are from the viewpoint.

Enable Backface Culling



ATGMaterials support a render state named Enable backface culling. If you require double-sided geometry, disable it for each material.

Texture Scrolling and Flipbook Animation

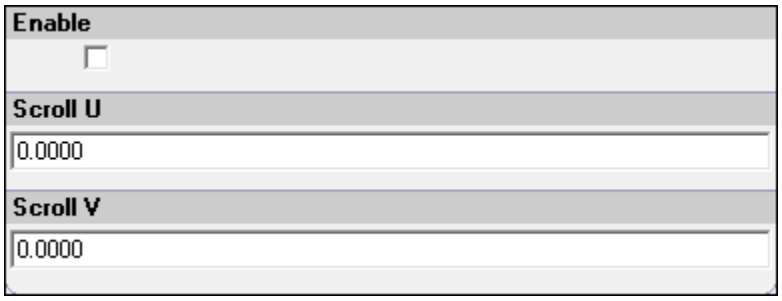
Scrolling UVs

Scrolling can be enabled on certain shader types. This will cause the UVs to animate based on the U and V scroll speed. This value cannot be edited via Lua and is permanent once exported.

Applying Texture Scrolling to a shader makes all maps (color, normal and specular) scroll. This can be previewed by scrubbing on the timeline within Maya. The UVs will scroll at a smooth constant rate.

Scroll U - Can be set to both positive and negative values.

Scroll V - Can be set to both positive and negative values.



Flipbook Animation

Flipbook stlye animation can also be used. Like smooth scrolling it is handled by a shader, however it is handled through the use of custom attributes added in the extra attributes tab.

For the Flipbook to work properly, you must add the following attributes:

- `frameAnimTotalNumFrames`: Total number of frames in the animation.
- `frameAnimFPS`: Sets the frames per second (fps).
- `frameAnimNumX`: Defines the number of frames on your texture in the X-axis.
- `frameAnimNumY`: Defines the number of frames on your texture in the Y-axis
- `frameAnimOffset`: Will offset the starting frame for the flipbook animation (for example, starting with the second frame instead of the first).

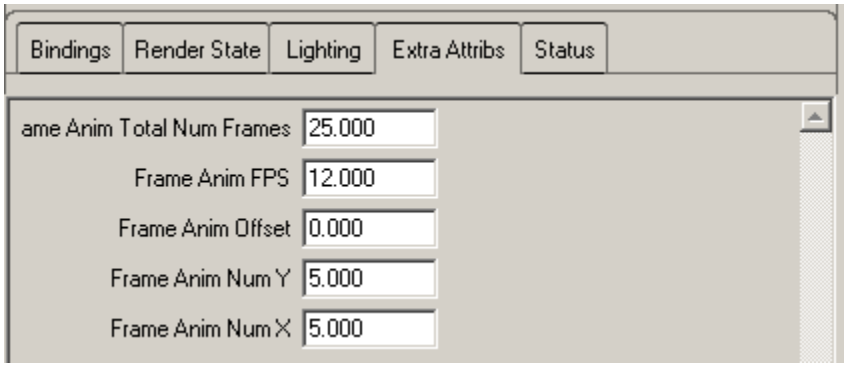
To add an attribute to the shader:

1. Select the shader, and select the Extra Attribs tab:



1. Select Modify > Add Attribute
2. Enter a name in the Long name field
3. Select Float under Data Type
4. Click Add and then the attribute appears under the Extra Attribs tab
5. Click OK to close the Add Attribute dialog when you have finished adding the attributes

The following image shows the Extra Attribs tab with all of the custom attributes added:

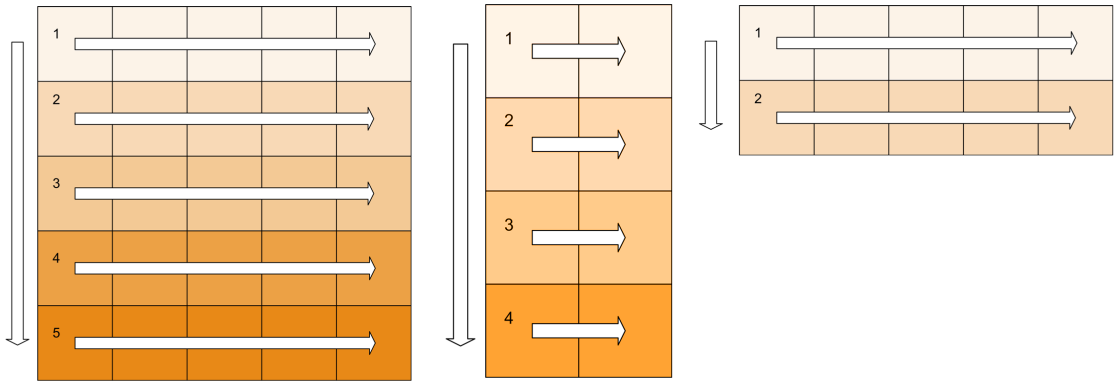


Setting Up Flipbook Texture

The size distribution and number of frames on the texture will need to match the values entered into the attributes.

The way you distribute your individual frames is up to you. For example, you can arrange all of the frames in one long strip (for example, 1x10), in a vertical strip (7x1), in a square (5x5), etc.

The distribution of frames is always read from left to right, top to bottom. There is no set rule to the number of frames, or the grid's distribution within the texture. This is defined in the custom attributes.



The first example above would use the following values. Its is a 5x5 square.

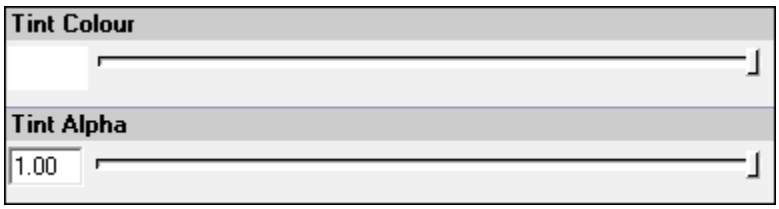
frameAnimTotalNumFrames	25	This block has 25 frames (5x5 grid)
frameAnimFPS	12	This sets the frames per second (fps). 12 fps was used because it gives a smooth animation. The rate is entirely up to you.
frameAnimNumX/ frameAnimNumY	5	Since there are five frames across and down
frameAnimOffset	0	This means that the animation will start with the frame in the first row of the first column

Shader Parameters – Tinting

Tinting

Tinting will add additional color or emissive color to your shader. The color will multiply over the existing maps assigned to the shader.

Enable Tint 



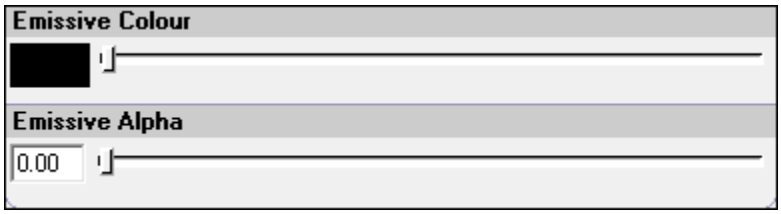
Tint Colour will multiply the set color with the existing color map.

Tint Alpha will act as a global alpha value for the whole mesh, any details added via the alpha channel of the color texture will be used to set the opacity of the mesh. The Tint Alpha will then be aplied over this seting the vasibility of what remains.

(The mesh will need to have its render state set to something other than Opaque for Tint Alpha to have any effect) See render states for more information.

Enable Emissive 

Enableing emissive will overide the shader into an emissive state. The shader will retain the Colour, Normal and Specular but will now also be emissive. This will have the same effect as a standard emissive shader in that it will not recive lighting but it will still cast a shadow if the original shader permitted it.



Emissive Colour will set the color of the emissive override

Emissive Alpha will act to override the existing alpha channel in the texrure map, at 0 the existing alpha channel will be desplayed and at 1 it will be ignored and the mesh will be 100% visable.

(The mesh will need to have its render state set to something other than Opaque for Emissive Alpha to have any effect) See render states for more information.

Both Tint Colour and Tint Alpha can be active at once.

> Tint Values and Lua
> It is also possible to set these values via Lua using **Material.SetVector**
>
> ```
> material:SetVector( "tintClr", Vector4.Create( colour, colour, colour, 1 ) );
> ```
>
> tintClr All materials with tint enabled r,g,b,a
>
> emissiveClr All materials with tint enabled r,g,b,a
>
> Note: You must have the relevant tint enabled in Maya to be able to set it via Lua.

Lighting Tab

Lighting

All the options on the Lighting tab affect only baked lighting - Lightmaps and vertex lighting. All the setting will override a normal function of Nova when preforming lighting.

Diffuse Channel Options
The Diffuse Channel Settings will show in secondary lighting - lighting that has bounced off of the objects the shader is assigned too, it will not affect the object's color directly.

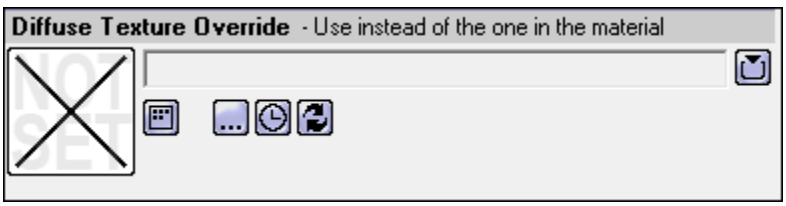
Use Texture (from material or override if set) Set by default will use the diffuse map in lighting calculations.

Use material colour Material color will override the main color map with the map that is set.

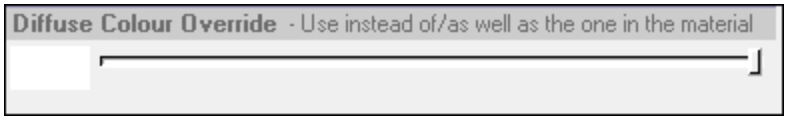
Use Override colour The whole shaders color will be overridden with the set color.

Use vertex colour - This has no effect in the PS Home Toolset.

Diffuse Texture Override
Setting the map above with the option Use material colour - This is a standard dds texture.

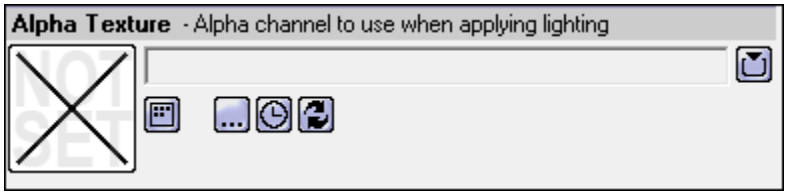


Diffuse Colour Override
Setting the color along with the Use Override Colour to set the color of the shader. This will completely override it is not additive like a tint.



Diffuse Vertex Colour Set – This has no effect in the PS Home Toolset.

Alpha Texture
Setting the Alpha Texture will allow any lightmap shadows to be calculated using the alpha in the given texture. Without this no alpha will be used in the lightmapping process. The texture will have to have an alpha channel to be used. It will not use the color channel. It is entirely valid to set this to the same as the main diffuse map to use the alpha.



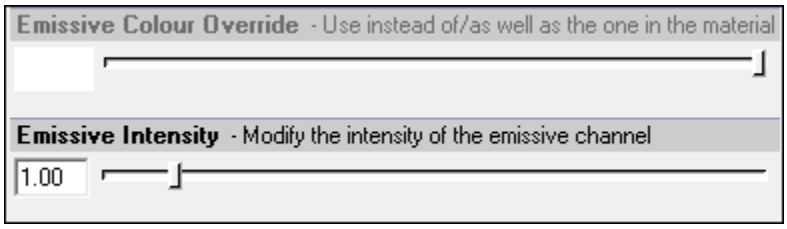
Diffuse/Alpha UV Set – This has no effect in the PS Home Toolset.

Emissive Channel Options

Use Override Colour – This will set the object to behave as if it is an emissive material for the purposes of lighting, the object will emit light. This option will have no affect on emissive materials as they are ignored by the lighting process.

Emissive Texture Override – This has no effect in the PS Home Toolset.

Emissive Colour Override & Emissive Intensity
The color and intensity of the light emission can be set here.



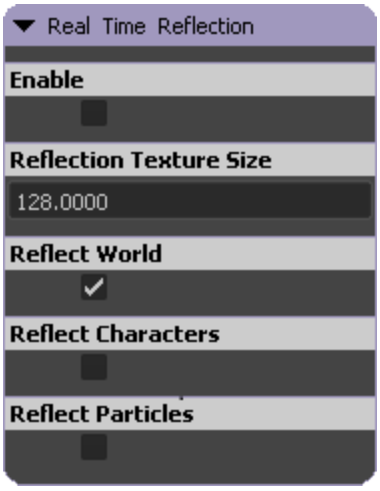
Emissive UV Set – This has no effect in the PS Home Toolset.

Lightmap UV set Override – This has no effect in the PS Home Toolset.

Real Time Reflections

Real Time Reflections

Real time reflections are only available on prelit materials. Ticking Enable will make the shader reflective.

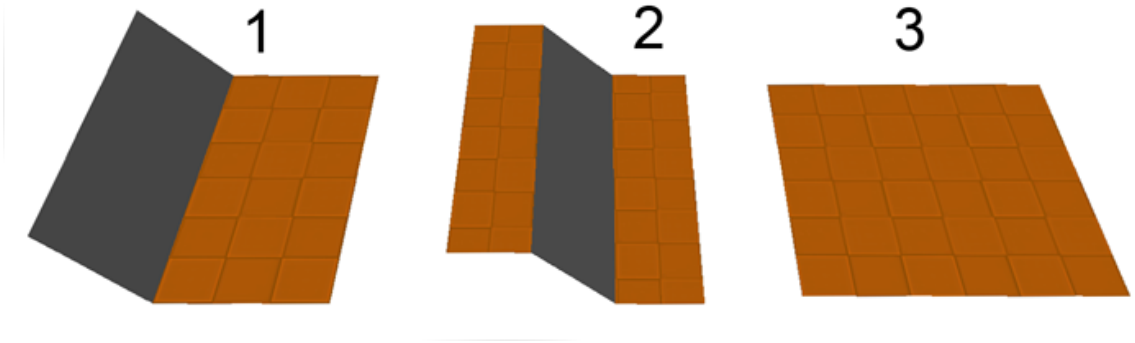


Options available:

Attribute	Description	Default
Enable	Turn on reflections for this shader. If this is not turned on, the other attributes will have no effect.	Off
Reflection Texture Size	Increasing the resolution results in a sharper reflection but uses more VRAM. Must be a power of two (128, 256, and so on).	128
Reflect World	Static environment and dynamic entities are shown in the reflection.	On
Reflect Characters	Avatars are shown in the reflection.	Off
Reflect Particles	Particle effects are shown in the reflection.	Off

Geometry Limitations

Reflective shaders must be applied to a flat plane at a uniform height. If you require multiple planes at different heights a different shader will be needed for each.



Example	Result
1	Will work.
2	Will not work (the reflective shader is applied to polygons at different heights.)
3	Will work.

⚠ If the shader is not applied to a flat plane it will display in unpredictable ways.

Controlling what will Reflect

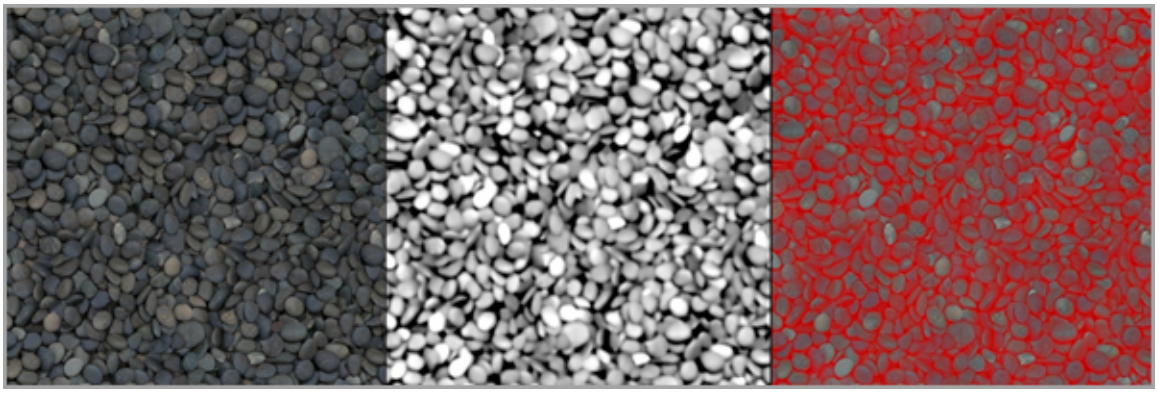
You can also use the Dynamic Reflection Editor to limit what will be reflected by your reflective surfaces.

Blend Shader Blending

Blend Shader Texture Maps

All the texture are as normal with the exception of the first color map. This map must have an alpha channel saved as a DXT5. The alpha channel of this map is used along with the vertex color to control how and where blending will take place within the shader.

The alpha map used is generally a height map, although any map can be used depending on the way you wish to blend the textures. The main reason a height map is often used is that begins the blend at the low points of a normal map giving the effect of building up from within cracks and crevices.



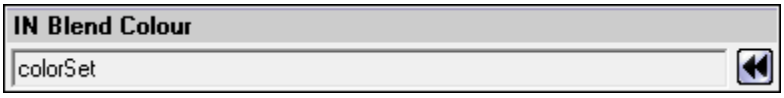
The map above is an example of the heightmap added into the alpha channel. The black and white image in the center is the heightmap.

Any mesh that will use the blend shader will need a color set applied to it. This is a default Maya color set found under the Maya menus:

Colour > Colour Set Editor

This editor will allow you to add a color set to the mesh and define its name.

After you have assigned all the maps, assign the color set to the shader (named BlendSet).



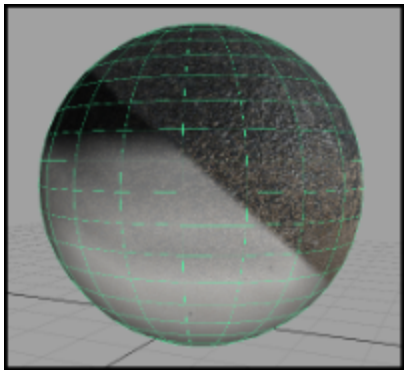
The color set it the used to assign the vertex color that will control how and where the blend will take place on the mesh. All color painting is done via the default Maya vertex color editing and painting tools.

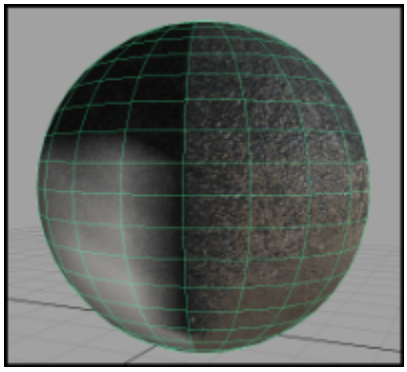
Vertex color applied to the mesh will adjust how much of each of the texture is visible and how much blending will take place based on the height map in the alpha channel. All vertex color should be grascale where Black will show more the 1st set of textures and White will show more of the 2nd. The two images below show this effect where pebbles are set 1 and sand is set 2

As a basic rule:

Black = First Texture Set

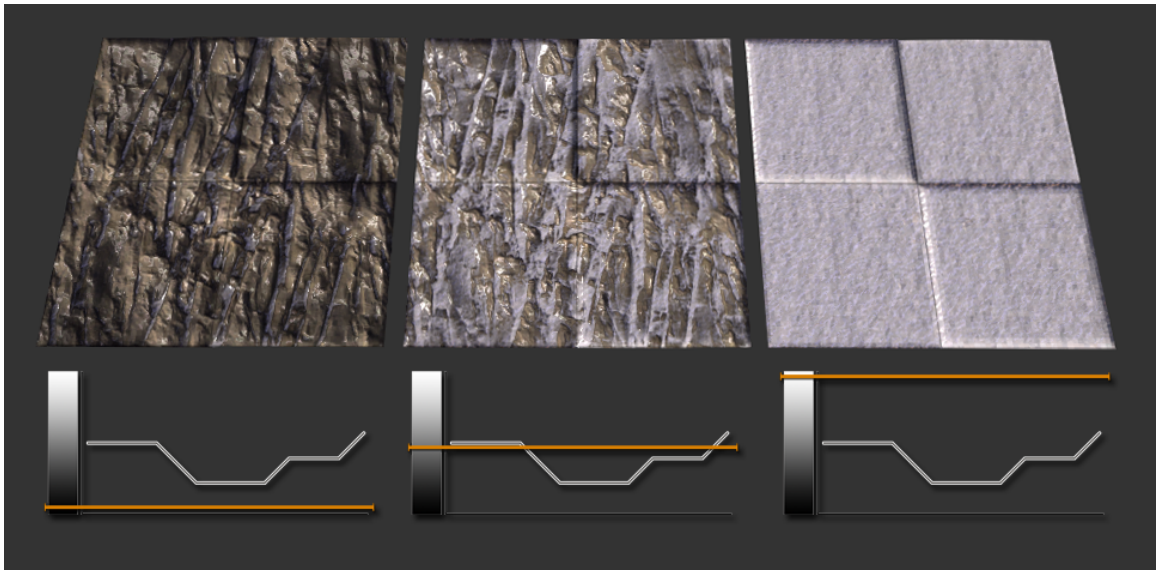
White = Second Texture Set





When painting for the Blend shader, use greyscale values between 0 - 1 only.

The graph below shows the how blending is handled between the vertex color and the height map. The vertex color applied to the vertices is represented by the orange line, The white line on the graph represents the height map in the texture. The texture is blended where the vertex color (orange line) and the color of the height map meet. The quad above each graph shows how this would look using actual textures.



Environment Maps

Environment Map Uses

Environment maps are used as an alternative to a realtime reflection. The environment map is a cube map of the environment. This cube map image is used in place of a true reflection as it still provides a good approximation and is significantly cheaper than a realtime solution. Environment maps are either specified in the shader (For Prelit Shaders) or are added to a scene file where it will become the global map for all other shader types.

Note:

- Environment maps do not reflect - therefore dynamic elements (such as avatars or dynamic sky) will not show up.
- Environment maps can give a less than perfect reflection on very flat large surfaces - they are best used on shaped objects where the exact reflection is not as important as the idea of a reflection to add credibility to the material.
- Environment maps are only available on Prelit shaders.

Generate an Environment Map with a Console Command

The console command `envShot 512` will generate an environment map for you based on the cameras position. Use the free cam to place the camera.

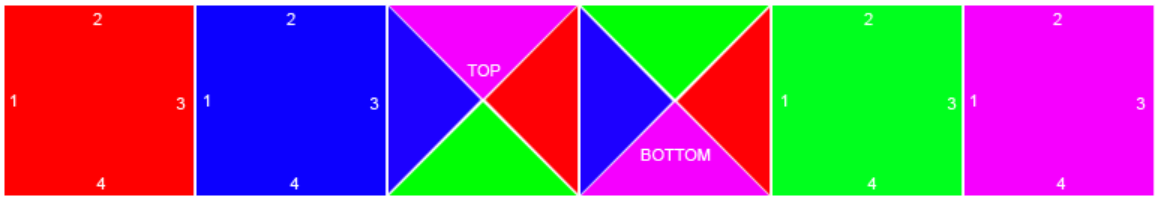
The number after the `envShot` command will represent the resolution of the environment map. In the case above 512 the cube map will be made up of 6 images each 512 x 512 in a strip. The total size for this texture will be 512 x 3072.

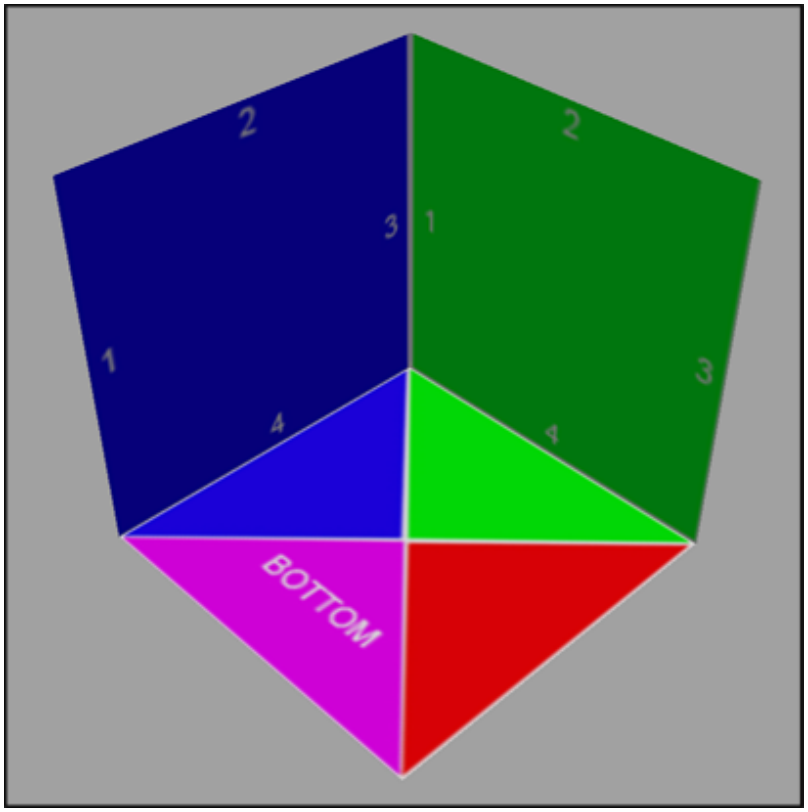
The map will output into the root of the HDK build folder and is saved as a `.tga`, this map will need to be saved as a `.DDS` cube map in Photoshop.

Be sure to select `Cube Map` in the options when saving.

Format Environment Maps by Manual Editing

It is possible to edit or even create a Cube Map by hand. There may be instances where this is preferable for certain effects. Below is an example of the layout for a map.





You can download the texture the above images are based on. By aligning the faces of a cube in Maya using this texture you can see how they will stitch together.

Download Reference Texture

Lightmap UV set 2

Lightmaps for Prelit Materials

All prelit materials will generate a lightmap. This lightmap contains the baked-in lighting information from the scene. Any lights you add in Maya (including the Sun and Dome light) will contribute to this lighting.

Every model that has a prelit shader applied to it will require a 2nd UV set. This can be added in Maya via the UV set editor.

Create UVs > UV set editor

The set should be called map2.

The set needs to be laid out uniquely (no overlapping UVs). If the UVs overlap, your shadows and lighting information will overwrite itself - resulting is incorrect lighting.

For each mesh in question, the size of the lightmap is calculated according to the average surface area of each of the lightmap polygons. So if you have specified a lightmap texel size of 10 cm, the export pipeline will look at the surface area of every polygon on every mesh and average them out. This figure can then be used to decide what size texture will be needed to allow a lightmap texel to be sized at 10 cm in the PS Home client.

Generate Missing map2 (Lighting) UV Sets

This tool will generate a UV set 2 with automatic mapping for any meshes with a prelit material that failed to export because of a missing UV set 2.

Home > Environment > Generate Missing map2 (Lighting) UV Sets

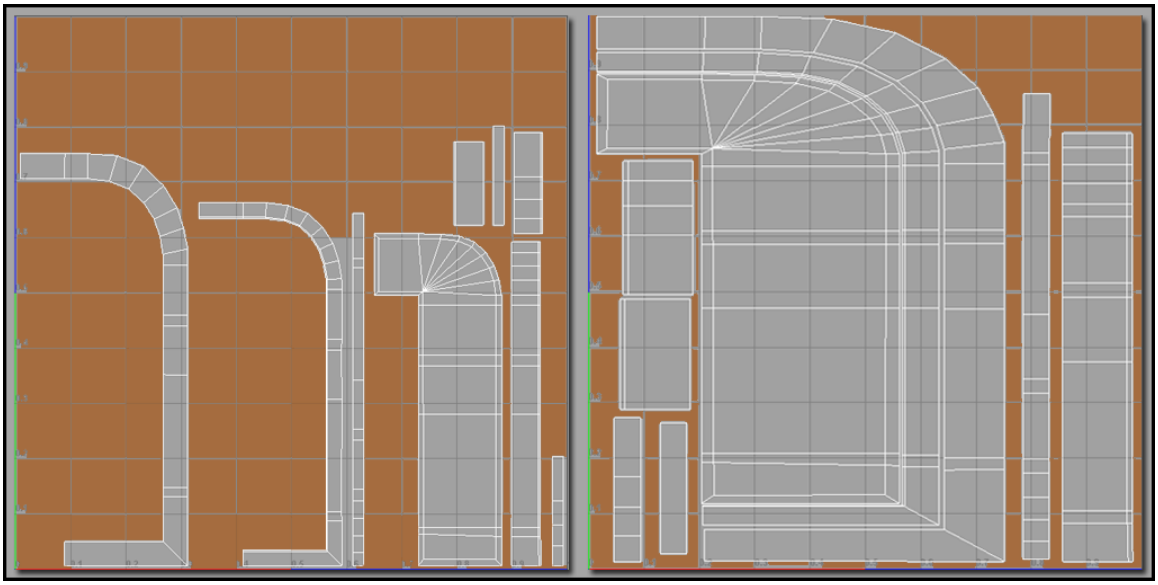
The UVs in this set will need to be edited as they will not be optimal.

This is a useful tool for fixing them automatically so export can continue. This automatic fix will generate a map2 for all meshes that failed - you cannot specify particular ones.

Optimal Lightmaps

The image below shows the difference between an automatically generated UV set 2 (left) and one that has been laid out by hand (right).

In this example the UVs on the left are automatically generated and the UVs on the right have been scaled to fill the available space (which allows for higher resolution lightmaps).



The map on the left has a large amount of wasted space, which can equate to a lot of lost detail.

The map on the right is more optimal, the resolution of the lightmap is determined by the side of the geometry in Maya.

The map must be square - using as much of the space as possible is advised.

It is also advisable to scale the UVs to fit the area. Although having skewed UVs is acceptable, but it is best to skew all the UVs in the same direction.

LightMap Scale
Never change lightmap UVs independently of each other. If there is a UV section that has been scaled independently of other pieces, it will disrupt this calculation and cause the lightmaps to be generated too big or too small.

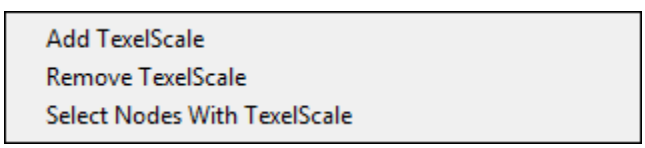
For example:

You have a lightmap UV layout where a UV section has been scaled right down (because, for instance, it's not visible to the player in PS Home). The exporter will always attempt to create a lightmap that will have the specified texel size in the client. Think of the texel size as the maximum size the exporter will allow a lightmap texel to appear. A small, independently scaled UV section will mean the lightmap will have to be enormous to get the texels of that small section to appear at the specified scale in the client.

Texel Scale

Texel scale can be added to a mesh in order to directly control the resolution of the lightmap that is generated.

Home > Environment > Add TexelScale



Once added the Texel Scale attribute will appear under the Extra Attributes tab of the mesh.



This acts like a scale value to the texel size. However, this process assumes that the UV layout is correct and directly proportional to the actual geometry.

To combine multiple small objects' UVs onto a single lightmap can save a great deal of memory, there is no need to have many small light maps for smaller objects see the Lighting Set Editor for more information.

Vertex Alpha

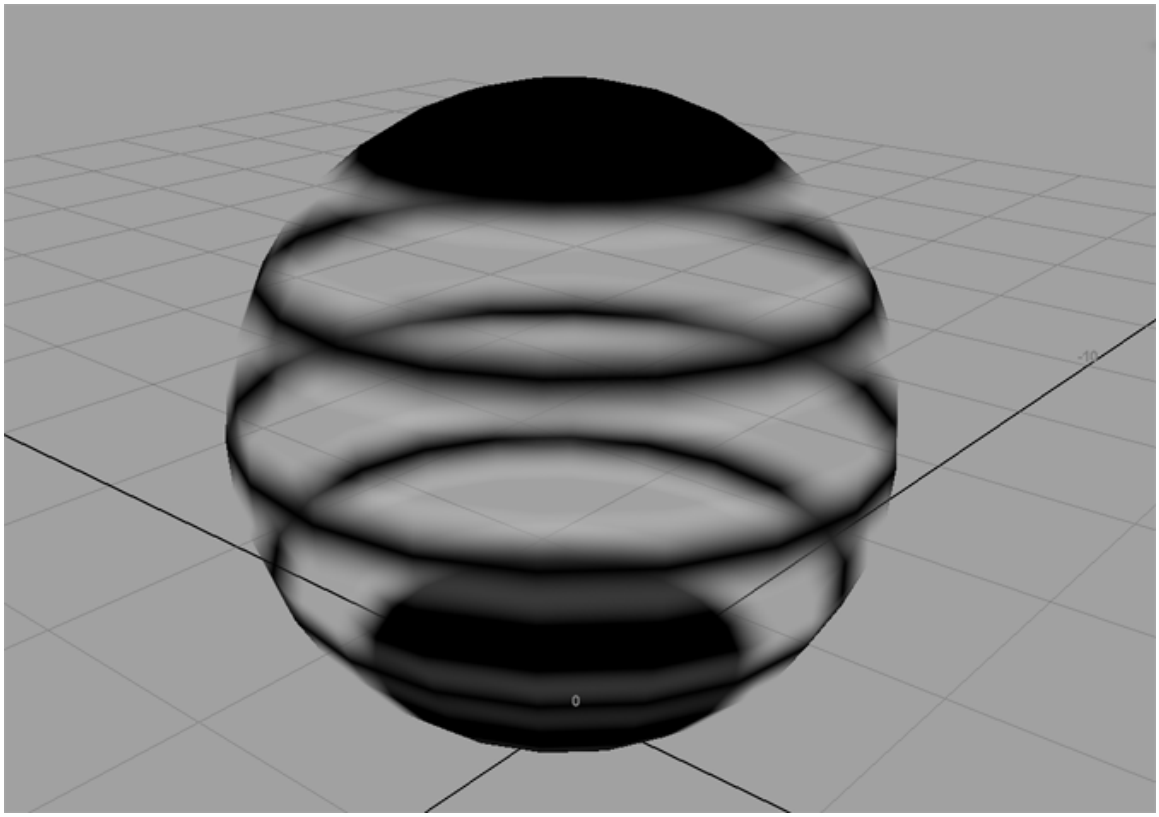
You can apply Vertex Alpha to create transparency on an object that is not based on the texture maps. The following materials cannot use Vertex Alpha:

- Blend
- Prelit - Blend
- Cloth

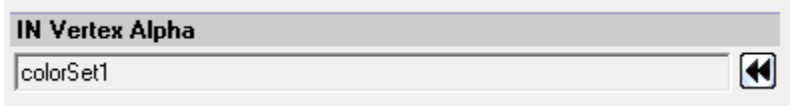
You apply Vertex Alpha using the Maya Color menu by selecting either of the following options:

- Color > Apply Colour
- Color > Paint Vertex Colour Tool

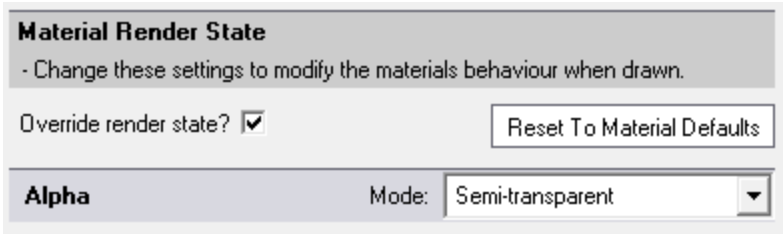
The results of the vertex alpha is visible in the viewport. If you do not see this transparency make sure that the alpha value of the vertex color is set correctly.



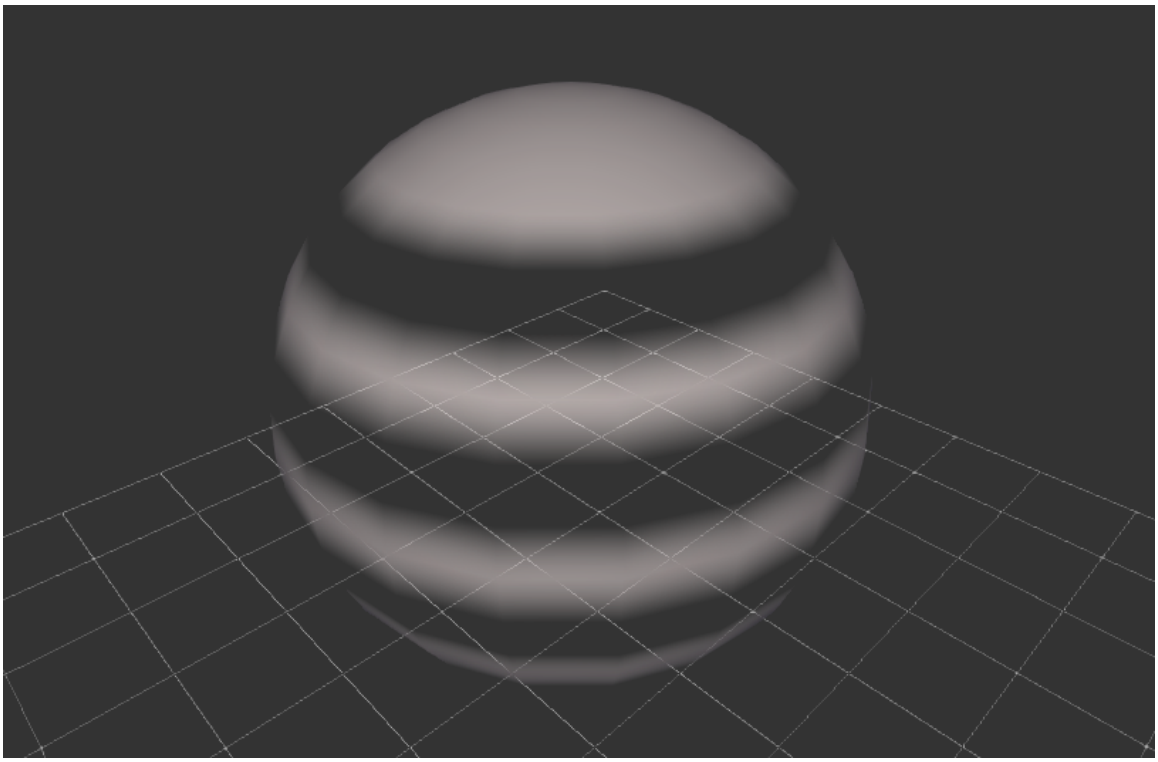
Specify the color set in the shader as shown in the following image:



The shader must also have its blend mode set to semi trans.



The following image shows an example of the final effect.



Collision Tools

Home Collision

Collision in PS Home is the same for environments and furniture. Avatar and companion collision works slightly differently, but is handled automatically in the PS Home client.

Collision is prepared by adding a group node in Maya with the name "Collision". All meshes placed under this group will be treated as Collision.

Collision does not require an ATG shader, it can have any Native Maya shader attached.

Collision Types

Upon export, each Collision mesh created in Maya is further conditioned depending on its associated Collision Type - i.e. Mesh, Hull or Primitive - and has a large impact on the resulting physics simulation.

Type	Description	Export notes	Notes / Examples

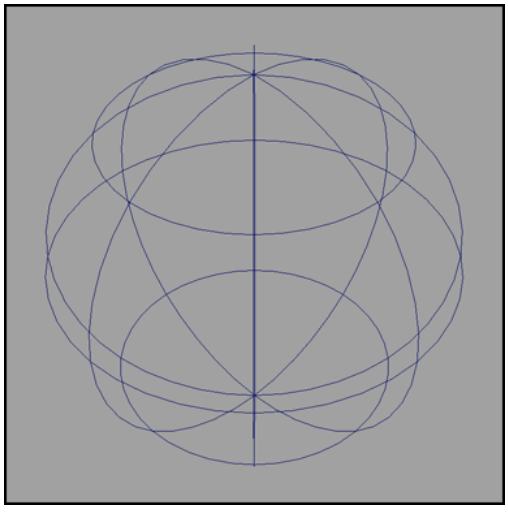
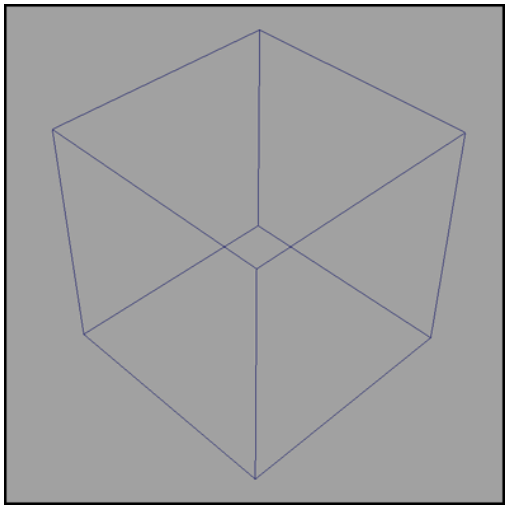
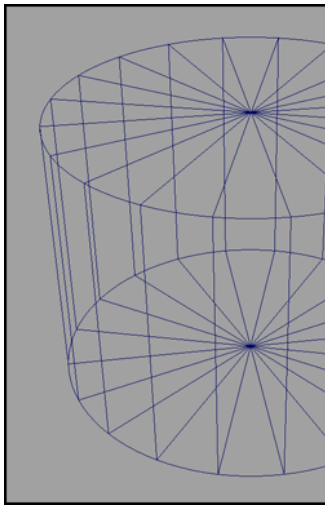
Mesh	Exported collision represents original mesh geometry.	Default for Environment collision meshes	Meshes result in less predictable and accurate simulation results as they are not optimized. Use primitive collision instead wherever possible.
Hull	Converts the collision mesh to a convex hull; this means any hollow areas are filled in.	Default for Furniture collision meshes	Example: This would convert a donut shape to a flat rounded disc. Most Furniture items are hollow, therefore must be broken down into separate primitives/hulls.
Primitive, i.e.: • Sphere • Box • Cylinder • Capsule	Converts the collision mesh to a perfectly formed primitive (simple) shape of your choice.	Must be set via Collision Shape Attribute (see below) to take effect.	Example: Set a Sphere primitive type on a ball to achieve a perfectly round shape during simulation. Without this, the ball will not roll well. Use primitives for best simulation results as they are highly optimized. This often means breaking down a single complex collision mesh into several primitives.

Collision Shape Attribute

Use Collision Shape Attribute to associate a Collision mesh with a primitive type.

This is available under the Home Environment and Home Furniture menus.

The association can be removed from the mesh with Remove Selected Collision Shape Attribute.

Sphere	Box	Cylinder
		
Create Collision Shape Attribute: Sphere	Create Collision Shape Attribute: Box	Create Collision Shape Attribu

Assign Sphere Collision	Assign Cube Collision	Assign Cylinder Collisi

For more information on where to use primitives or mesh collision see: Primitive Collision & Mesh Collision

Debugging primitives
The following information is useful for debugging primitives that are not behaving as you would expect.

Conversion to a primitive follows particular rules:

- The centre point of the mesh defines the centre of the resulting primitive.
- The bounding box of the mesh defines the extents of the resulting primitive.
- The transform of the mesh defines the position and orientation of the resulting primitive. (If you move or rotate a collision primitive - move the transform of the object, not the vertices.)

After the conversion to primitive, vertices have no direct relationship to the collision in game.

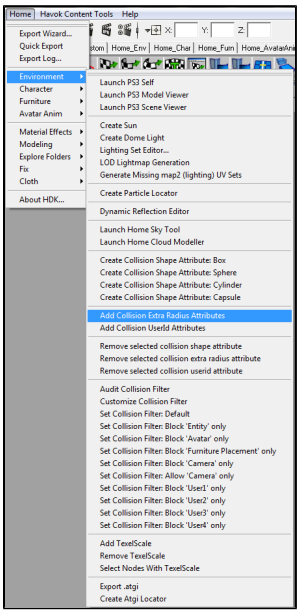
Collision Convex Radius

The default convex radius is 5 cm.

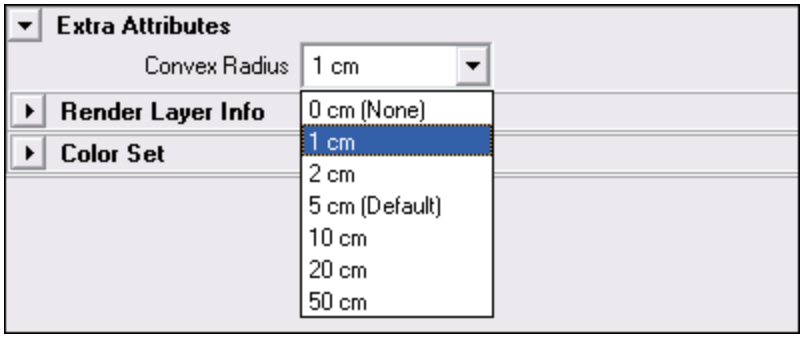
The convex radius allows Havok™ to detect collisions before two collision meshes have intersected, an operation which is simpler for Havok™ to process than dealing with collisions post-intersection.
When items are exported, the collision meshes are automatically shrunk down to a smaller size, but with a convex radius of a specific size around them. For example, a 10 cm^3 cube with a convex radius of 2 cm will be treated by Havok™ as a 6 cm^3 cube (2 cm removed from each side of the cube), but will detect collisions 2 cm earlier. The resultant area covered by the mesh including the convex radius is effectively unchanged, but can be handled by Havok™ much more efficiently.

To set the convex radius, go to:

Home > Enviroment > Add Collision Mesh Attributes



This will add an attribute to the mesh that will appear under the Extra Attribites Tab in the Maya Attribute Editor



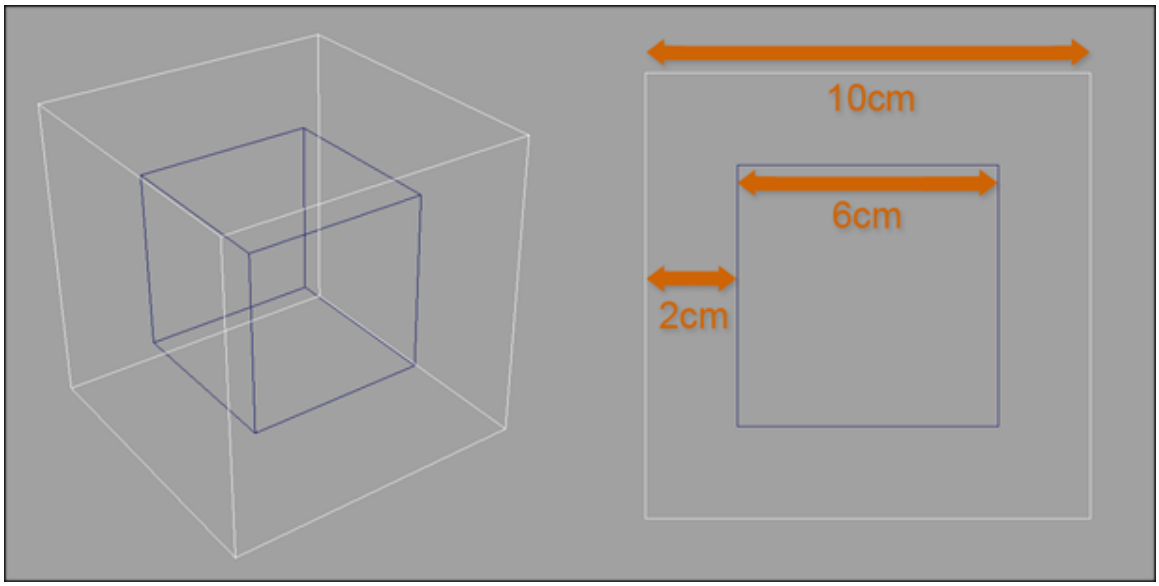
The collision mesh must be bigger than double your convex radius. Therefore, if your convex radius is 5 cm, your mesh must be more than (not equal to) 10 cm. As 5 cm is the default, this setting would need to be changed when creating an item with a radius of 10 cm or less.

For example

When items are exported, the collision meshes are automatically shrunk down to a smaller size, but with a convex radius of a specific size around them as buffer.

Therefore, a 10 cm^3 cube with a convex radius of 2 cm will be treated by Havok™ as a 6 cm^3 cube (2 cm removed from each side of the cube), but will detect collisions 2 cm earlier.

The resulting area covered by the mesh (including the convex radius) is effectively unchanged, but can be handled by Havok™ much more efficiently.



Collision User ID

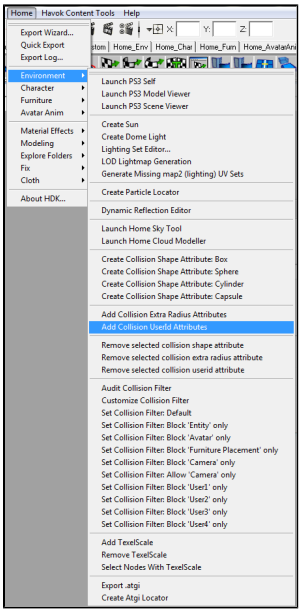
Collision can be assigned an ID (CollisionUserId) in Maya and then retrieved via Lua script.

This makes it possible to identify individual collision meshes by their IDs.

All collision created before HDK 1.50 will have the UserId set to 0 by default

Retrieve your Collision UserId by using:

- a Person (get the ID for the collision the avatar is currently standing on)
- a Raycast (get the ID for the collision the Raycast hits)



This adds an attribute to the mesh that will appear under the Extra Attributes Tab in the Maya attribute editor:



> User ID and Lua
>
> With the Collision UserId added to collision, you can query the value using:
>
> ```
> Person.GetGroundUserId()
> Raycast.GetSurfaceUserId()
> ```
>
> Person - The function `Person.GetGroundUserId()` returns the UserId of the collision that the person is standing on. Remember to call `Person.IsOnGround()` to find out if the using is standing on the ground first.
>
> Raycast- The function `Raycast.GetSurfaceUserId()` returns the UserId of the collision that the Raycast hit.

Collision Filter

For certain meshes in a scene, you can assign what the mesh collides with and which collision layer the mesh is assigned to. You can do this for environment collision meshes (using Maya and the Lua API) and dynamic entities created with the Lua API.

Use Collision Layers to customize how these meshes collide. You can assign a group of these meshes to a layer and then choose what collides with or does not collide with everything on that layer using Collision Filters.

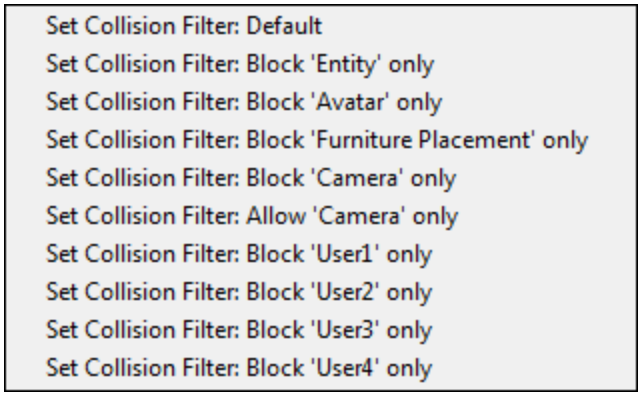
Use Collision Filters to customize the collision for these meshes by deciding which layers it collides with and which ones it does not.

This feature has many possible uses for example, you can limit the area in which furniture items are placed in a personal apartment or create camera collision to control how the camera moves through a space.

The default layer is Scene, there are also User1, User2, User3, and User4. All collision must belong to one of these layers.

There are a number of preset options to determine what the collision will collide with.

Home > Environment



It is alos possible to customize these options with the Customize Collision Filter tool.

Home > Environment > Customize Collision Filter

See Custom Collision Filter tool for details.

Collision Layers and Lua
Entities can be assigned layers and collision filters with Lua.

You can assign the collision layer to an entity using Lua (or query what the setting is) with these functions:

- `Entity.SetCollisionLayer` - Set the layer that the collision entity is a member of - Scene, User1, User2, User3, User4
- `Entity.GetCollisionLayer` - Get the layer that the collision entity is a member of - Scene, User1, User2, User3, User4

You can also set the filters for an entity, choosing to block or allow collision for each layer:

- `Entity.EnableLayerCollision` - Specify a layer and set it to true or false. This enables or disables collision with that layer.
- `Entity.IsLayerCollisionEnabled` - Query whether or not an entity can collide with collision objects of the specified layer.

Use the Scene library to change the default behavior of all the collision on a layer:

- `Scene.SetLayerCollisionDefault` - Enable/disable collision detection between collision objects of the specified collision layers. This does not override the settings of entities already created. It is most useful for defining the default settings applied to entities that are about to be created.
- `Scene.GetLayerCollisionDefault` - Query the default setting for collision between specified layers.

Collision by Object Type

Additional information on the different objects and how they use collision is available here:

Collision for Active Items

Collision for Furniture

Collision for Companions

Collision for Environments

Collision for Avatars

Primitive Collision & Mesh Collision

Maya Meshes and Collision

Geometry in Maya has no effect on collision. The collision is determined by the bounding box of the collision mesh and the attributes set. Choosing how to shape the collision mesh is important, and it is recommended that primitive collision is used rather than mesh collision.

Why Create Primitive Collision?

Primitive collision is used in place of mesh collision for a variety of reasons:

- economical in terms of memory (a smaller .hkx file size)
- more accurate when dealing with dynamic interactions between objects (realistic and robust)

For instance, take a piece of furniture colliding with an environment that contains convex shapes. The inside of a dome created out of mesh collision has an issue when colliding with furniture; it may even be possible to force objects through the collision.

To prevent this you can use multiple primitive collision volumes to make the rough shape of the dome. This makes the collision more realistic and robust.

Primitive Collision in a Space

The main use of primitive collision in your space is for shapes that are complex or convex. Walls and floors are usually already simple enough but better results are often achieved by using primitive collision.

Primitive Collision in Furniture and other Objects

Very few objects should be made up of anything other than primitive collision. Especially as objects can be interacted with by the avatar as well as interact with various environments.

Using Mesh Collision

Mesh collision is only available for environments. All other objects types require that you have primitive collision shapes.

In general it is always best to use primitives for you collision. It is possible however to create meshes for very complex or awkward shapes within an environment. It is possible to use mesh collision for this.

Keep these things in mind when using mesh collision.

> 1. Could this shape be created out of a primitive or a number of primitives?
>
> 2. Will dynamic objects be interacting with the collision (physics objects for example)
>
> 3. Is the mesh collision very dense.

If any of these are true then it would be advisable to use primitives in the place of mesh collision.

Collision For Active Items

Active Items are objects that the user can place within a personal space or clubhouse, in a similar way to placing furniture.

They are furniture items (including furniture collision), but with an additional collision group (SafeVolume) and they allow scripting.

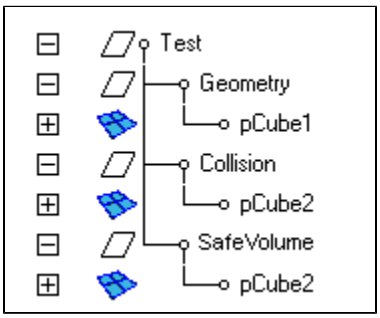
For more information on creating active items, see Active Items.

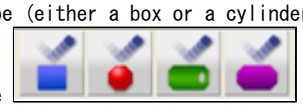
To make active items, export the models (e.g. furniture from Maya, then add a script component using the Object Editor.

Creating a SafeVolume file for your furniture

To create a SafeVolume file for your furniture:

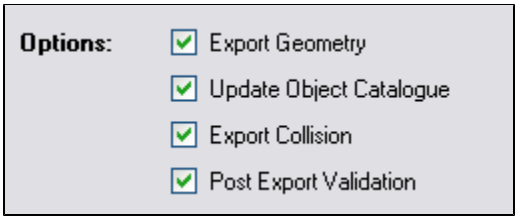
1. In the existing collision group, create a group called SafeVolume.



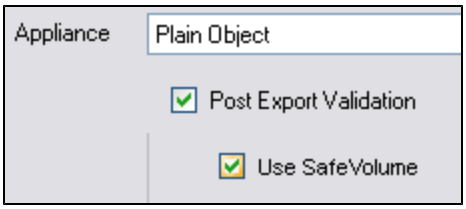
2. Within the SafeVolume group, create a collision mesh using a primitive shape (either a box or a cylinder).
3. Select the active collision mesh and click the appropriate icon for the shape  (or select one of the Create Collision Shape Attribute options from the Home > Furniture menu).
The collision shape is stored as an extra attribute and can be changed by accessing the drop down menu:



4. Export as usual, ensuring that Export Geometry, Export Collision and Update Object Catalogue are selected:



5. During export, the Furniture Object Editor opens (see Validating and Exporting Furniture for more details). Within the model template, ensure that the object information for your active item is set to the appropriate category, the appliance is set to Plain Object and Use SafeVolume is selected:



Click  in the model template to change object information.

A SafeVolume file is added within the active item build folder (with the extension .safevolume.hkx).

Developers can create an additional collision file for an active item that is used specifically for collision when the script component of the object is activated. The additional collision file prevents other users from entering the space used within a mini-game, for instance. When the user places the active item in a scene, the active collision is represented by a transparent box. This visual aid helps the user place the item where there are no obstructions that would interfere with the scripted elements.

Collision For Furniture and Objects

For collision to react properly within PS Home, models require a basic polygonal form to represent the solid surfaces. This should be made up of very simple geometry i.e. boxes. Avoid representing intricate details and curved surfaces.

Use primitive shapes when making furniture collision.

The following image shows a furniture model and its collision mesh. Note that the curved surfaces and details, for example, legs are not modeled individually but have large primitive collision shapes assigned:

Unable to render embedded object: File (0300001A.png) not found.

The collision volume does not require an ATG Shader so leave the objects with the default Maya Shader Material assigned. In addition, collision geometry does not require any particular UV coordinates.

⚠ Note that:

- When you have the simple mesh collision, the base of the furniture and its collision should be at 0 and also the pivot point of all collision objects (including the main collision transform) should be at their local origin 0,0,0. For example the world axis in side view or front view should be treated as the floor.
- Individual collision shapes less than 10 cm thick in any direction may not work robustly in PS Home. Make sure furniture items are bigger than this to ensure accurate collision in PS Home.

Small Objects

If your furniture object is very small (with small collision volumes), it may result in your furniture object not colliding correctly or passing through other objects in the scene.

To help prevent this issue, increase the convex radius so that the initial collision checks detect the object. See also recommended Furniture Dimensions.

Seats and Camera Collision

When seated, sometimes the user view may be partially obstructed (if the camera is forced too close to the avatar and all the user can see is the back of the avatar's head).

When this happens, the avatar is normally hidden automatically by the system. This automatic hiding occurs because the client determines whether the camera is within the avatar's bounding box, and if it is, hides the avatar.

However, for avatars with small bounding boxes, the camera may not be within its boundaries, and will not make the avatar disappear.

It is best to avoid situations where this collision issue may occur.

Keep the following in mind in order to prevent camera issues:

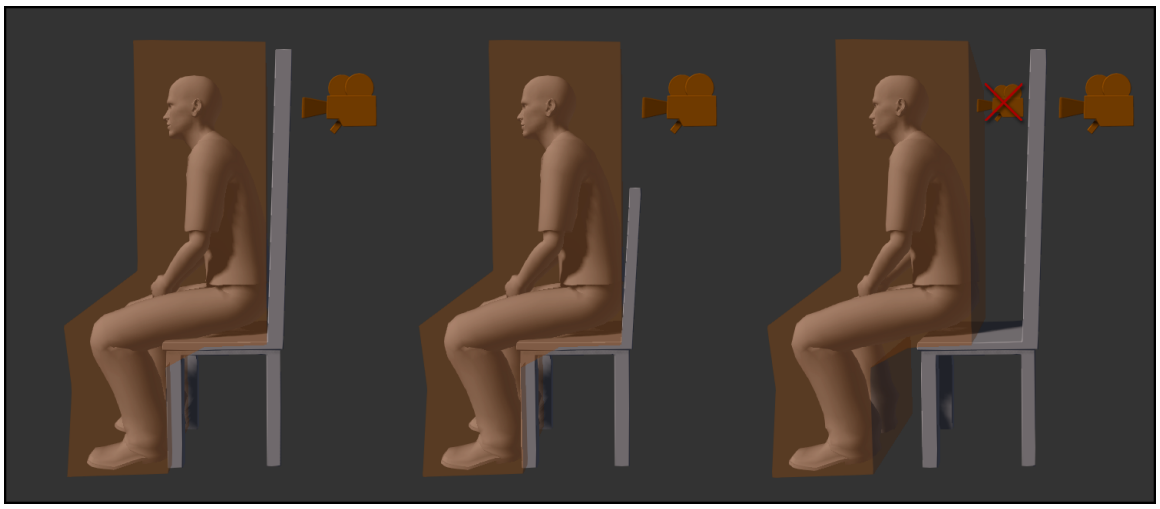
- seat geometry
- avatar clothing (e.g. extremely large clothing)
- seats with an intended focus point (e.g. seats in an auditorium)

Seat Geometry

Seat geometry can affect the camera position, especially when sitting in a high-backed seat (where its top edge meets with the user's neck).

See Dimensions > Seating Height.

Normally, a ray is cast from the avatar's head to the camera's position. If the ray is obstructed - for example by a seat back - then the camera will move itself in front of the blocking geometry.



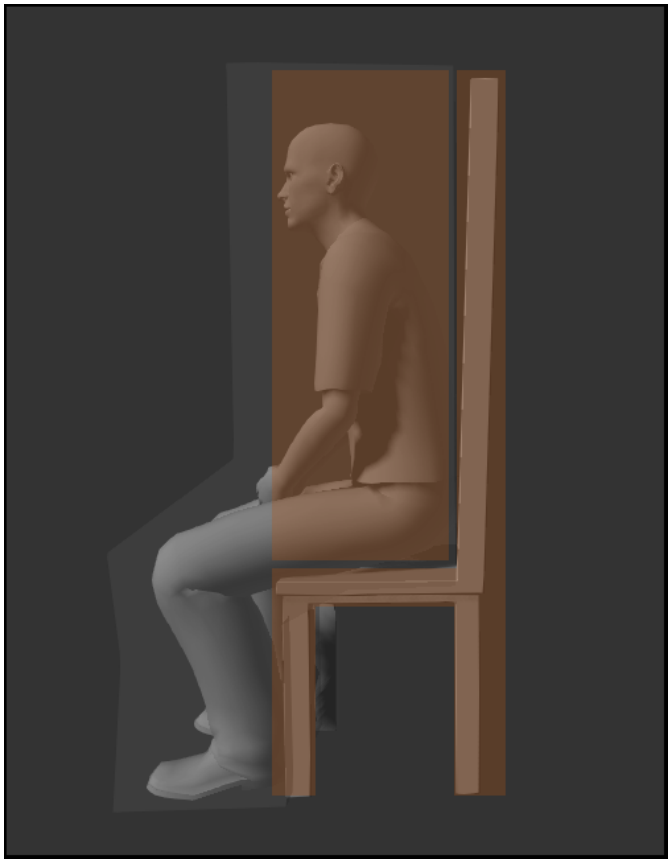
If the camera is then within the bounding box, the avatar will 'disappear', that is, the camera will move in front of the avatar, making the avatar disappear from the user's perspective.

This problem can occur with seats where the back of the seat is well below the avatar's head, the camera is able to move to its normal position without issue.

With tall seats, the collision geometry itself may cause the obstruction. When this happens, the camera moves itself just in front of the geometry. If the avatar's bounding box is small, if the space between the geometry and the avatar is too wide, or if there is a combination of the two, then the camera will stay just in front of the seat's back and right behind the avatar's head, causing a partially obstructed view.

To prevent these kinds of issues extra collision can be added to the seats. When sitting the avatar will ignore the collision of the furniture object.

Additional collision can be added to the seat to help block the camera from problem areas. In the example below an additional block of collision is added to the seating area. This will not affect the avatar sitting but will help to prevent camera issues.



Avatar Clothing

Particularly large items of clothing are problematic when clothing clips with the camera (clothing has no collision so the camera position will not be forced out).
We recommend testing seats with a variety of avatar sizes and clothing, particularly extremely large clothing.

Seats with Focus Points

Environments with a focal point can be more subtable to collision problems on furniture. For example an auditorium with a screen that has a number of seats facing it. In situations like these its best to watch out for collision issues.

Collision For Companions

Companion objects will have a single collision file. This volume is a fixed size that you do not need to edit.

The collision volume is called **sphere.hkx**.

This file is generated when you create the companion object and can be found here:

\build\CcompanionFollowers\<Companion_Name>\collision\sphere.hkx

The enhanced companion collision has a radius of 0.25m and is 0.5m from ground to head. The collision is effectively just a sphere rolling around on the ground (controlling navigation - so not needed for flying pets).

It's not crucial that the enhanced pet fits in the sphere (especially since the name label can be bumped higher).

Ground-based and Flying companions

Collision is the same for both flying and ground-based companions. You do not need to create additional or special collision for them.

Collision For Environments

For collision to react properly within PS Home, models need a basic polygonal form to represent the solid surfaces. Use very simple geometry, such as boxes. Avoid representing intricate details and curved surfaces.

The following image shows an environment model and its collision mesh:

Unable to render embedded object: File (03000045.png) not found.

To calculate the size of the collision in a whole space, add up the file sizes of the collision .hkx files of the scene and all the objects that will run within it (such as Mini-game and arcade game objects, but not furniture items).

The collision volume does not need an ATG Shader. Just leave the objects with the default Maya Shader assigned. In addition, collision geometry does not require any particular UV coordinates.

> ⚠ Please note:
>
> - You must place collision geometry in a collision group named Collision.
> - Apply transforms on collision geometry before exporting to the Scene Editor. Rotation, Scale and Translation properties within the Scene Editor have no effect on the collision mesh.

> ⚠ Please refer to Collision For Avatars for important information regarding remote avatars, and how they interact with the collision in the space.

Using Collision to Seal Off Areas

Some spaces have areas sealed off with collision which the player can unlock through various ways. This could be via attaining achievements in games, or through various commerce systems. Try to avoid creating items which allow players to move to, or view, these places without performing the actions required to gain entry, for example, Active Furniture which moves the player via a script, or allows the player or camera to 'fly' without restriction. We also recommend that you keep private areas of your scenes separate from the main area more securely than just by collision.

See Technical Notes: Do not give the user the ability to move to or view restricted areas of scenes.

Collision For Avatars

When designing environments, scripting interactions or setting up objects for avatars to interact with (such as seats), it is useful to know how avatar collision is handled in PS Home.

Local Avatar Collision

The local avatar is the user's avatar. The local avatar's collision can be thought of as a capsule 1.8 m in height, and with a radius of 0.4 m.

Unable to render embedded object: File (image001.jpg) not found.

Remote Avatar Collision

Remote avatars are all the other avatars in a space with the user's (local) avatar. Remote avatars are positioned using move messages sent by their owners every 2 seconds. They are interpolated over these two seconds to give smooth movement.

> ℹ There is a two second lag between where an avatar moves, and where it appears to be remotely (on someone else's screen).

Unlike the local avatar, remote avatars have no collision capsule. The position of remote avatars is reliant on the move messages received from their owners. However, they are vertically adjusted to stick to the floor (unless they are seated on a chair) to avoid floating above, or merging into, the scene geometry, such as when going up stairs. This is done by firing a collision ray (a thin ray-cast) from above the avatar's feet, to below the avatar's feet. The length of the collision ray is restricted to

prevent the avatar from stepping up too high or falling to low. It's specifically designed to recognize small steps or stairs and to adjust accordingly, but to ignore temporary errors in interpolation where the avatar would otherwise be placed on top of a wall, or on a lower floor in the scene.

> ⚠ If there are any gaps in the floor of a scene, remote avatars can fall through them (although they will usually return to the floor after interpolation). To prevent this problem, pay special attention to make sure that there are no gaps in the floor collision. It's important to understand that this needs to be tested specifically for remote avatars, as the local avatar will not suffer the same problems.

Pushing Away Remote Avatars

Because it's important to always keep the local avatar visible, and not obstructed by remote avatars, no matter where they're positioned, or what they're doing, remote avatars are pushed away from the local avatar, i.e. if any remote avatar gets too close to the local avatar, they can be seen to be pushed away slightly. This is of course only visible to the local player. On the other player's screen, they are unaware of the push, but similarly, they are pushing other avatars away from their local avatar.

So, note that:

> ⚠ The position of the remote avatars can be different from their real position. That is to say, they appear in a position that isn't the same as the actual position seen by them, and other users.

> ℹ When pushed away, the remote avatar will also face the local avatar. However, if a player is blocked or ignored, their avatar will be forced to face directly away from the local avatar.

See also Seating and Collision and Collision For Environments.

Customize Collision Filter

Collision Layers

PS Home uses a fixed number of Collision Layers - with each layer representing a category of collision.

All environment collision is assigned to the Scenelayer

All dynamic entities are assigned to the Entitylayer

Scene:

Environment meshes are assigned to this layer by default. This layer is used for static meshes for normal collision behavior e.g. when an environment is exported without customization, all collision meshes are assigned to the Scenelayer.

Entity:

Dynamic objects are assigned to this layer by default. This layer is used for furniture, entities created with Lua, entity animation models, companion objects and portables. Use this to create collision with dynamic objects e.g. to make a bouncing ball collide with a wall.

Avatar:

All avatars are assigned to this layer. Use this to create collision with avatars e.g. stop avatars from moving into a particular space - a pool of water, the edge of a cliff, etc.

Furniture Placement:

This layer is used while furniture is being placed by a user. Once the furniture has been placed, it is assigned to the Entitylayer. Use this to block the placement of furniture in a particular space.

Camera:

This layer is used by all in-game cameras. Use this to control where the camera can move - and therefore the view a user has of your scene.

User 1-4:

These are customizable layers.

Assign a mesh to one of these layers to group them and customize the collision between objects in your scene.

Assign a mesh to one of these layers to group them and customize the collision between objects in your scene.

- only one mesh? only one mesh per layer?

Assign a mesh to one of these layers to group them and customize the collision between objects in your scene.

- assign a mesh to group them - then you can customize the collision between objects in your scene.
  or
- assign a mesh *and if you *group the assigned meshes you customize collision between objects in your scene.

rephrase sentence to make things clearer)

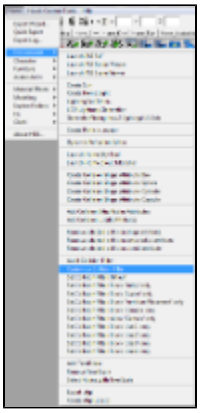
Preset Collision Layers

All of the presets place the collision objects themselves in the Scenelayer.

- Default - Reset the collision to the default and block objects on every layer.
- Block 'Entity' only - Choose to block only objects on the Entitylayer.
- Block 'Avatar' only - Choose to block only objects on the Avatarlayer. Use this, for example, when creating a gangway that stops the avatar but allows the camera to swing out over the edge.
- Block 'Furniture Placement' only - Choose to block only objects on the Furniture Placement layer.
- Block 'Camera' only - Choose to block only objects on the Cameralayer.
- Block 'User 1-4' only - Choose to block only objects on the User 1-4 layer.

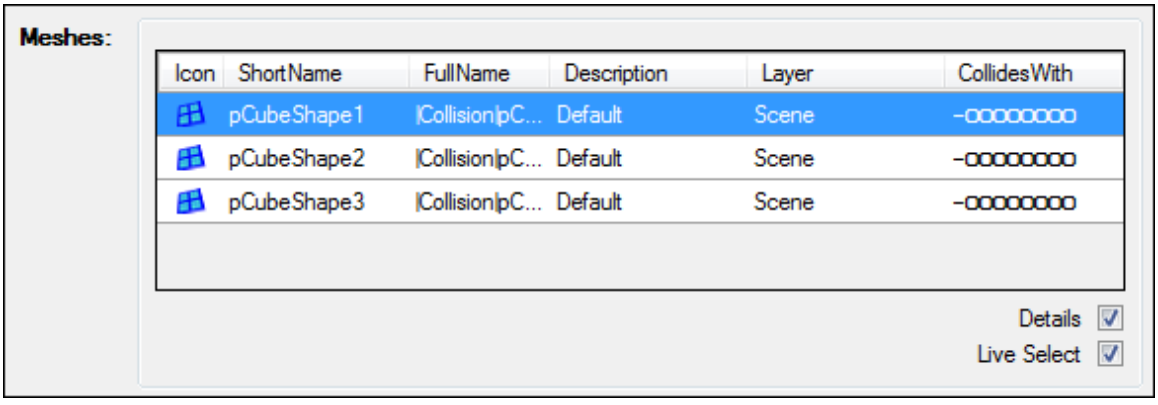
Customize Collision Filter

Home > Environment > Customize Collision Filter



Meshes:

Custom collision filters are applied to a selection of meshes. The tool will allow you to select one of the meshes.

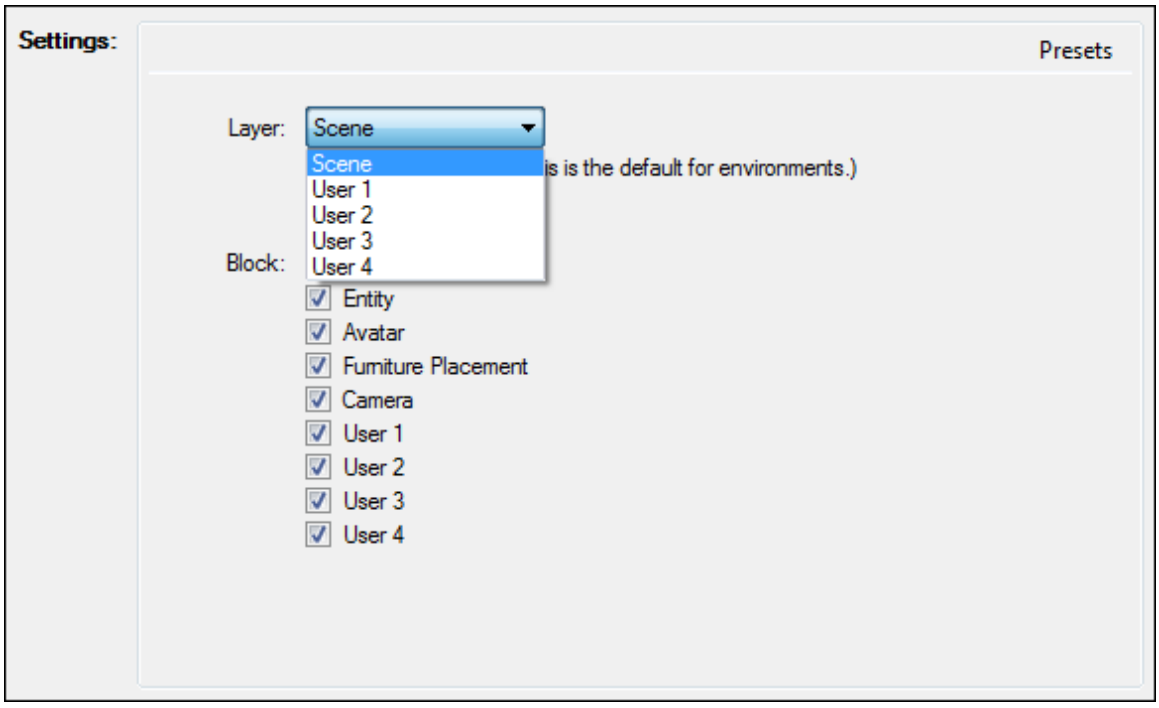


Details shows additional information relating to the collision meshes (such as layer).

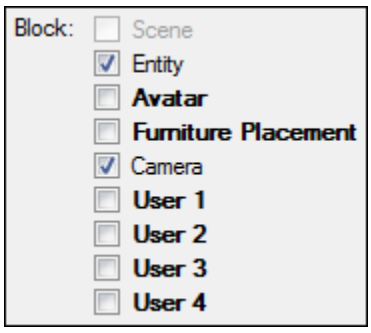
Live Select will highlight the selected mesh in the viewport.

Settings:

From here you can use the dropdown to select which layer the collision mesh should be assigned to.



The check boxes allow you to set items that will collide or not collide with this collision mesh. Checked items will collide.



Audit Collision Filter

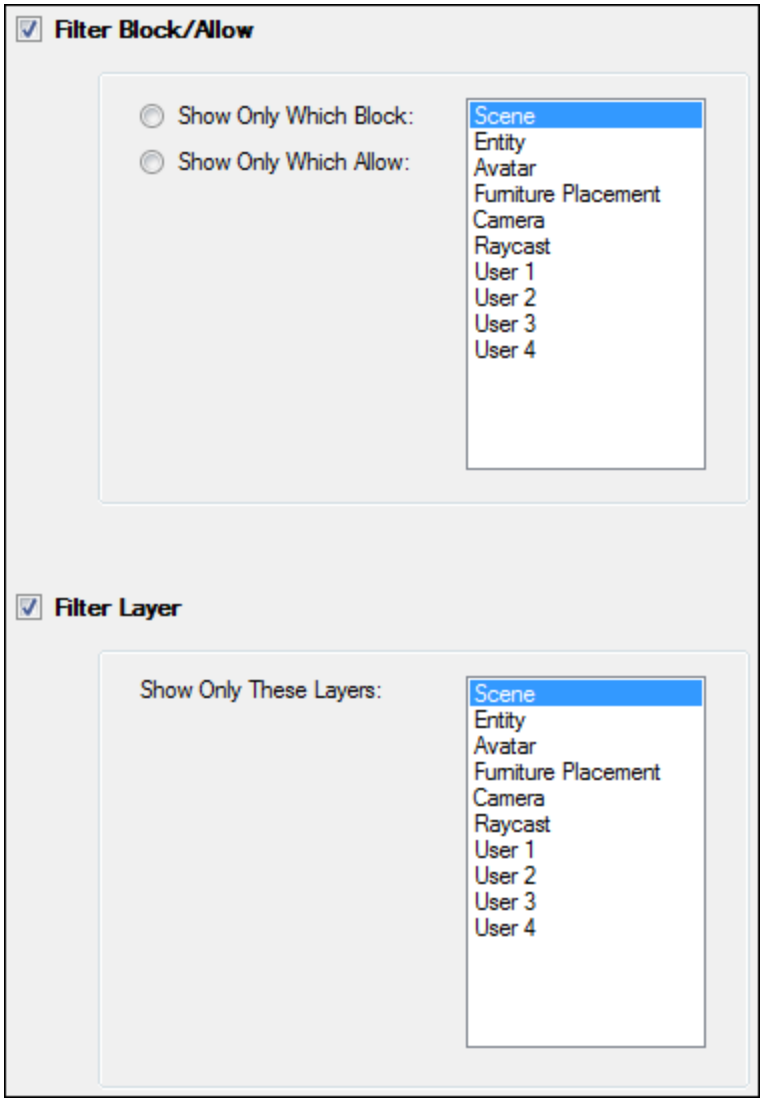
The audit collision filter tool is in essence the same as the tool above except that it is not possible to edit the collisions settings. This tools is just to allow you to view the current settings of collision meshes in the scene.

There are alos additional filtering tools added.

Filter Block/Allow Toggles the ability to filter by what will blocked or allowed by the collision mesh.

Show Only Which Block / Show Only Which Allow You can chose between filtering based on what is allowd to pass though or what is blocked by the collision.

Filter Layer Show only collision that is set to the layer.



Scripting and Debugging Collision

These pages contain information on scripting and debugging collision within home. All collision must be created within Maya, this collision data can then be attached to an Entity using Lua. This can allow you to adjust and change collision for a number of game-play reasons.

For information on debugging collision in game see:

Debugging Collision Items

For tips when scripting collision and the associated Lua commands:

Lua Collision Functions

Lua Collision Functions

Tips for Scripting Collision

Use the following guidelines to help you when scripting collision:

- Simple collision geometry works better in PS Home.
- The Havok™ physics library attempts to model Newtonian physics inside of PS Home. You can interact with it by supplying forces.
- Using complex functions or systems to perform a trivial physical maneuver is likely to be problematic; it is more efficient to use the functionality available using the Havok™ physics system
- Using the existing Havok™ physics system has the advantage of objects moving and behaving in a visually engaging manner without having to constantly update them from Lua. You save a considerable amount of scripting time.
- Ensure your scripting is kept simple. Avoid interfering with the automatic functions within the existing physics system

and avoid superfluous scripting or duplicating behaviors that already exists in the physics system.
- The Havok™ broadphase is a key component of the collision system that optimizes collision checks within a defined area. The broadphase size can be specified in the Scene Editor as a minimum point and a maximum point. See Game/Scene Properties.

Lua Functions

The collision functions allow you to set collision, enable collision layers, and use UserIds to identify where a user is standing or which collision a raycast hit.

```
Entity.EnableCollision
Entity.EnableCollisionCallbacks
Entity.EnableLayerCollision
Entity.GetCollisionLayer
Entity.GetContactCount
Entity.GetContactEntity
Entity.GetContactId
Entity.GetContactPoint
Entity.GetContactRelVelocity
Entity.GetContactUserId
Entity.IsLayerCollisionEnabled
Entity.RegisterCollisionCallback
Entity.SetCollision
Entity.SetCollisionLayer
Entity.SetMaxContacts
Entity.SetRestitutionCoef
Person.GetGroundUserId
Person.IsOnGround
Raycast.GetSurfaceUserId
```

Debugging Collision Items

Collision Items are Havok™ rigid bodies created by entities (e.g. created by scripted objects), furniture, etc. They do not include environment collision.

To debug Collision Items you can use the `ci` command in the debug console.

Use the `ci` command to query and alter Collision Items for debugging purposes only. To alter the collision for your content and achieve a particular behavior in the developed content, use the Lua API.

> In the information about the syntax for the debug command, the "debug item" is the Collision Item which is currently the target of the `ci` command.

The color that a Collision Item is rendered in is significant:

- **Blue** = fixed (not dynamic, cannot move under dynamics)
- **Red** = dynamic (can move) but deactivated (at rest, dynamics not currently being simulated by Havok™)
- **Green** = dynamic and activated (moving/dynamics are currently being simulated)

Syntax

The syntax for the `ci` command is:

```
ci <subcommand> <param1> <param2> ...
```

Vector values should be entered in quotes (single '0 0 0' or double "0 0 0") and the components should be separated by spaces. For example:

```
ci set linvel '0 10 0'

ci set gravity '0 5 0'
```

Sub-commands

Sub-command	What it does

print	prints info about the debug-item
print verbose	prints more info about the debug-item
printall	prints info about all items
printall verbose	prints more info about all items in the system
count	gets how many items there are in the system
null	sets no debug item
first	sets the debug item to the first one in the system
last	sets the debug item to the last one in the system
next	sets the debug item to the next one in the system
prev	sets the debug item to the previous one in the system
get <attributename>	gets the value of the attribute for the debug item
set <attributename> <attributevalue>	sets the value of the attribute for the debug item
setall <attributename> <attributevalue>	sets the value of the attribute for all items in the system

Attributes

Attribute	What it defines
dynamic	boolean - true if the item is dynamic, false if the item is fixed
active	boolean - true if the item is active, false if is deactivated (at rest)
autodeactivate	boolean - true if the item is to be deactivated when it is almost at rest, false otherwise
interpolated	boolean - true if the item's position data is to be interpolated (smoothed), false otherwise
mass	float - the mass of the item
com	vector - local offset of the center of mass of the item
pos	vector - the world position of the item
rot	quaternion - the world rotation of the item
linvel	vector - the linear velocity of the item
angvel	vector - the angular velocity of the item
lindamping	float - the linear damping of the item
angdamping	float - the angular damping of the item
friction	float - the friction coefficient of the item
restitution	float - the restitution (bounce) coefficient of the item
gravity	vector - the gravity acceleration to apply to the item
rendershape	boolean - true if the item's collision shape is to be rendered, false otherwise
rendercontacts	boolean - true if the item's contacts are to be rendered, false otherwise

Corresponding Entity Lua Functions



Some attributes do not yet have specified Entity Lua Setter or Accessor functions. These attributes are shown in cells marked with '-'. Only Lua functions in the Lua API are supported, and where there is no Setter or Accessor function, no substitutes may be used.

Attribute	Entity Lua Setter	Entity Lua Accessor

dynamic	EnableDynamics	-
active	SetDynamicsActive	IsDynamicsActive
autodeactivate	EnableDynamicsAutoDeactivation	IsDynamicsAutoDeactivationEnabled
interpolated	SetInterpolated	IsInterpolated
mass	SetMass	GetMass
com	SetCenterOfMassLocal	GetCenterOfMassLocal
pos	SetPosition	GetPosition
rot	SetRotationQuat	GetRotationQuat
linvel	SetLinearVelocity	GetLinearVelocity
angvel	SetAngularVelocity	GetAngularVelocity
lindamping	SetLinearDamping	GetLinearDamping
angdamping	SetAngularDamping	GetAngularDamping
friction	SetFrictionCoeff	-
restitution	SetRestitutionCoeff	-
gravity	SetGravity	GetGravity
rendershape	DebugSetCollisionVisible	-

Examples

Type `ci next` and `ci prev` into the console to step through each Collision Item in the scene.

Unable to render embedded object: File (collisiondebugcinext.png) not found.

Unable to render embedded object: File (collisiondebugciprev.png) not found.

You can keep the current Collision Item rendered by typing `ci set rendershape true` before stepping to the next item.

Typing `ci setall rendershape true` will render all Collision Items, but be aware that this will likely impact the framerate adversely.

Collision Requirements and Recommendations

For Japanese Live Page
最新の日本語版ページはこちら 「コリジョンの必須要件と推奨要件」

Requirements

All the collision requirements are automatically validated on export in the HDK tools (Maya, Scene Editor and Object Editor).

- All models (e.g. environments or objects) must have at least 1 collision mesh, except character components and animations.
- There should be no collision set for character components or animations.
- Only 1 collision group for objects (environments can have more than 1 collision group).
- Active items must have only 1 SafeVolume group and 1 collision group.

Collision and SafeVolume

- SafeVolume and collision group names are not case sensitive.
- For active items only, SafeVolume must have a collision attribute and it must be only one box or cylinder.

- Make sure that the collision and Safe Volume do not compromise the integrity of the environment or disrupt any user's state. Safe Volume should not prevent a user from moving, or push a user outside of a space. Conversely, set the Safe Volume to make sure that the area that an active needs to operate is not obstructed by environment or placed items.
- Make sure that proper collision and Safe Volume are in place so that the integrity of the script cannot be compromised by users or by furniture placement. For example, if the active item becomes a bowling alley when activated, the collision/Safe Volume should be along the lane so that furniture or users cannot block the bowling ball from going in the gutter.

For more information about SafeVolume, see Collision For Active Items.

Recommendations

- For environments, use a collision file size of 1 MB.
- Collision meshes have pivot points at their local origin.
- All environments and objects must have basic polygonal forms representing solid surfaces simply.
- Use primitive collision in all possible places as a simplified version of the geometry for the collision mesh does not necessarily yield the best results.
- Using the shape of the model to calculate the collision is resource intensive and can result in incorrect collision behavior in PS Home.
- You can place collision or SafeVolume groups in any position in the hierarchy; they do not necessarily have to be at the root.

Tips on Collision for Environments

- Hide areas not meant to be visible and keep the camera in mind. There are several camera views. The chase camera view is the default view when the user moves the avatar around and the user can set it to a close view or a normal view. The user can also move the camera in an orbit around the avatar with the right analogue stick on the controller or is triggered as the default when the user is idle for a while (a mode called Attract Mode Camera). Scripts can also control the camera view (for instance during a Mini-game).

  Keep in mind that there are several possible camera angles when creating collision for a scene, paying particular attention to confined spaces, low ceilings, etc. If the collision is set so the camera is able to view the space from too high up, areas that are "behind the scenes" and should remain hidden will become visible.

- Block off escape routes.

  If the avatar can wander through a wall or off a floor, they will suddenly drop out of the space or get stuck somewhere. The same goes for placing objects in spaces (although furniture can be retrieved as it can be targeted from anywhere, it is not ideal). If you create a space, make sure you prevent this happening. Think of it as making the scene waterproof by encapsulating it in collision meshes.

Tips on Collision for Environments and Objects

- Use primitive collision wherever possible.

  If shapes are simple they should be ok, but if shapes are complex or convex, it is best to use primitive collision to avoid unrealistic and unpredictable behavior. Primitive collision is also more economical in terms of memory.

- Do not make the collision areas too thin.

  Thin collision areas, for instance floors or tables, can cause unpredictable behavior. Work around this by making the collision surface deeper. When objects are dropped onto surfaces with a thinner diameter than the object, the collision may not react as expected.

- Position seating to allow easy access to (and exit from) the seat.
  Ensure that avatars can easily access the seats, and can sit down and stand up without collision restrictions hampering their position or animation fluidity. For more information, see Seating and Collision.

- Try to work with the PS Home physics system, not against it. The less you tweak the physics system (by scripting objects or trying to make objects behave differently to how they were intended), the more likely it is to be compatible with a number of situations and be robust and realistic enough to pass QA.

Optional Home Tools

Custom Scripts for PS Home

Within the Home HDK directory there is a folder that you can add custom tools and scripts to.

By putting these scripts in that folder location they will show up in Maya, in the Home menu under Extra.

Scripts will have to be written in Mel or Python for Maya.

The folder for these scripts can be found here:

<HomeHDK>\dev\ArtTools\Maya\Extra\<Scripts Go Here>

Available Optional Tools

The following optional tool is currently available for PS Home.

Source Animation Reference Tool

Source Animation Reference Tool

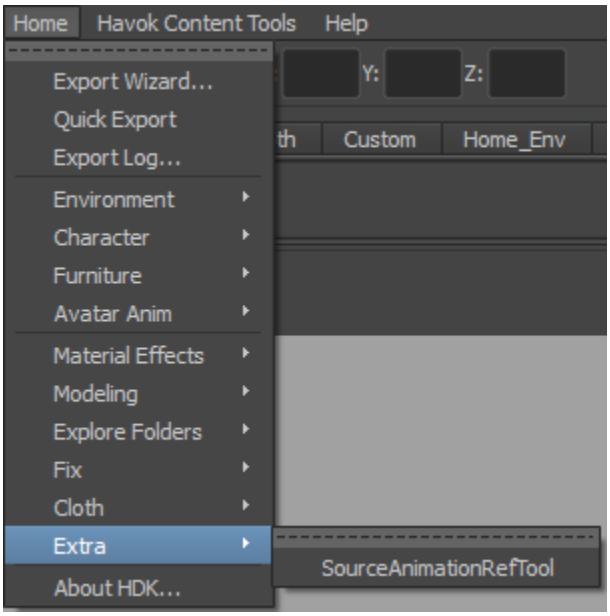
The Source Animation Reference Tool provides access to a selection of animations from Home to help you test character component weighting.

To use the Source Animation Reference Tool, download the SourceAnimationReferenceTool.zip file from https://home.scedev.net/projects/hdk/allreleases and unzip it to your installation directory, i.e. C:\HomeHDK.

Direct download link: https://home.scedev.net/projects/hdk/dl/dl/461/1341/SourceAnimationReferenceTool.zip

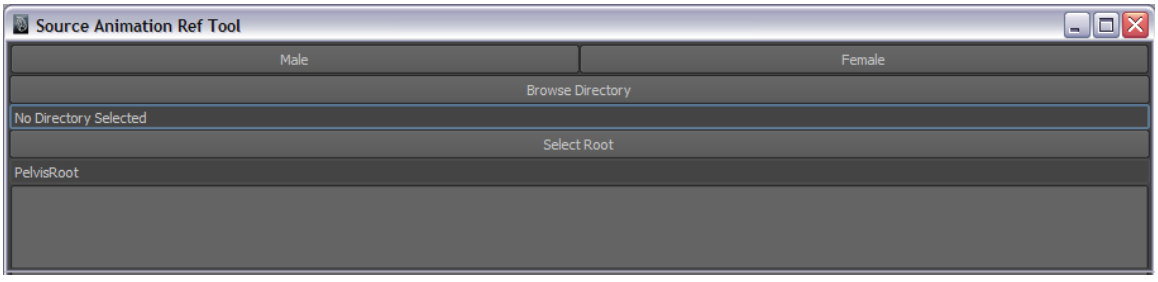
Source Animation Reference Tool in Maya

When the tool is installed, the required menu is displayed in Maya under the Home > Extras menu as shown:



Using the Source Animation Reference Tool

1. When you select Home > Extra > SourceAnimationRefTool, the following dialog opens:



2. To load the animations click the corresponding gender button, i.e. Male or Female.
3. To load an animation from the list, double-click it.

This action replaces the existing animation on the character component rig.

> To revert to the original animation, click the DefaultRestore tab, and double-click the animation you require.

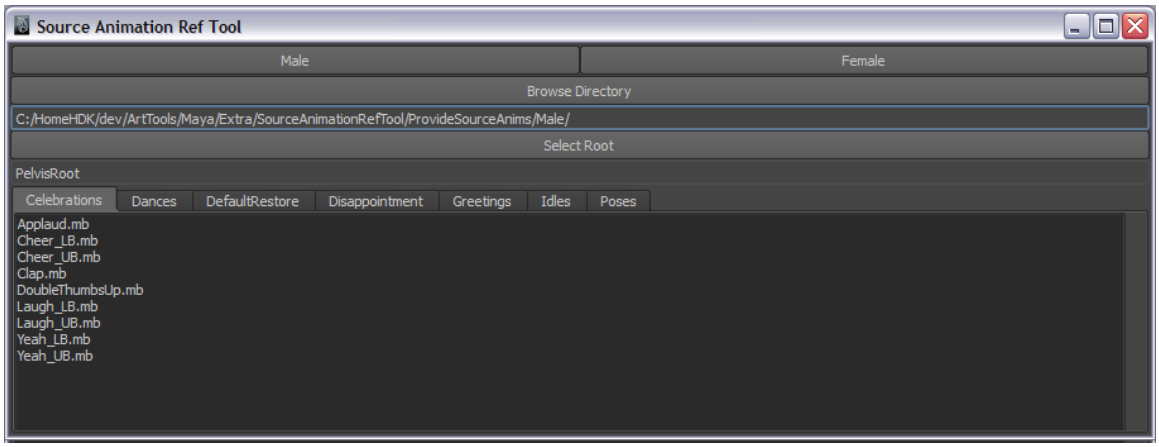
Swapping your Custom Animations onto the Avatar's Rig

You can also swap your own custom animations onto the avatar's rig for testing purposes. To load your own animations into the tool:

Click the Browse Directory button and select the required folder.

The tool finds all the .ma and .mb files with the selected folder and sub folders.

The following image shows an example of the default Male animations being loaded:



> Source Animation Reference Tool also allows you to swap animations onto rigs other than than of the Home Avatar.

By default, PelvisRoot is set as the root joint. To change this, select the new joint from the list and then click the Select Root button. This action ensures that animations not based on the avatar Home rig can also have their animations copied if you wish to use a custom rig and custom animations.

Testing, Validating and Submitting Content

> For Japanese Live Page
> 最新の日本語版ページはこちら 「テスト、検証、コンテンツの提出」

This section contains information and guidance on the tools validations available in the HDK. It also summarizes memory limits for different content types, and any other limitations and recommendations to help you prepare for publication.

It covers how to:

- Plan your content by knowing how to comply with requirements, what the tools check automatically and what you must to profile manually.
- Make sure that objects and scenes conform to the technical requirements - resulting in content that operates successfully in all the possible conditions in PS Home while still maintaining the highest possible quality.
- Use the tools provided in the HDK to help validate and profile content against parameters and constraints.

There are certain limits and restrictions placed on content for PS Home that ensure the content functions properly in all conditions. Some of these limits are checked automatically in the HDK content creation tools. For more information, see HDK Tools Validations. You must test other requirements manually in the runtime using the profiling tools. For more information, see Profiling in the Client.

⚠ You must submit for publication in PS Home only content that has passed all validations during export, packaging and manual profiling.

ℹ The PS Home Technical Requirements Checklist (TRC) values and requirements come from this documentation. When checking the TRC, refer to this documentation.

Content Requirements

For Japanese Live Page
最新の日本語版ページはこちら 「コンテンツの必須要件」

There are various requirements and recommendations to ensure that your content passes QA, publishes successfully, runs optimally and provides good experiences for players.

Before you publish, make sure that your content meets the requirements.

Requirements are based on a number of things, including technical limits (for example, available memory), system requirements (for example, file types), legal obligations (for example, age restrictions), and design rules (for example, differentiating between players and Non-Player Characters (NPC), angle for displaying objects in thumbnails, etc.).

To help you test and validate content before submission, the HDK provides guidance on the types of requirements, automatic validations in the tools, and profiling tools in the client.

Some requirements are tested and validated in the tools. Some can be validated automatically and some have to be tested manually. Automatic validations are done on export and when packaging.

Other requirements can be tested only while playing the content in PS Home, particularly scripted content that is dynamic. The HDK provides various tools to help with profiling content in the client and in the CDEV environment.

We also have guidance for launching and testing your content.

See also:

- Memory Limits
- Models and Meshes
- Dimensions
- Thumbnails
- Age Restrictions
- General Design Guidelines
- Save Data Guidelines
- Non-Player Character Guidelines

Backward Compatibility

We maintain backward compatibility with each new HDK release, unless announced or documented otherwise.

ℹ When you create or modify content, you must follow the requirements as stated in Testing, Validating and Submitting Content and the Technical Requirements Checklist. For example, assets created in HDK 1.32 work in the HDK 1.35 client, but if you modify them using HDK 1.35, you must adhere to the HDK 1.35 rules and requirements.

Regional Requirements

The requirements listed in this documentation are technical requirements, based on technical limits. There are also regional requirements because of regional laws, business operations, and the regional look/presentation of PS Home. Contact your Regional Account Manager for rules, requirements and guidelines specific to your region.

Memory Limits

For Japanese Live Page
最新の日本語版ページはこちら 「メモリ制限」

Memory limits form part of the technical requirements for publishing content in PS Home. Read the information in this section in conjunction with the following sections:

- HDK Tools Validations
- The latest Technical Requirements Checklist (TRC)
- The available Technical Notes

See also:

- Memory Limits for Scripted Objects
- Media Memory Usage
- Memory Limits for Spaces

PS Home Capacity and Resources

Two considerations determine the conditions under which content must operate:

- Capacity (such as how many avatars and objects you can have in a space)
- Resources (such as memory, rendering and processing time, bandwidth use, and frame rate)

PS Home Capacity

PS Home has a finite capacity. The technical requirements in this section outline which limits should be adhered to make sure that content interacts in PS Home under the conditions of interoperability defined by this finite capacity.

PS Home can accommodate the following types of environment:

- Public Spaces
- Public Spaces (no Video)
- Personal Spaces
- Personal Spaces (Video)
- Personal Spaces (no Video)
- Clubhouses
- Game Space 16
- Game Space 32
- Game Space 60
- Video Space

For more information about these spaces, see PS Home Spaces.

Each space allows a certain number of avatars to be present in the space. Each avatar can wear a certain amount of clothing and accessories (character components), and each have one inventory item (such as the bubble machine) equipped. Additionally, avatars can place furniture and active items in their personal spaces and clubhouses.

> ⚠ Resource limits for PS Home (including memory and network bandwidth) are constantly under review. Updates are available on https://home.scedev.net.

Content must be able to work under these conditions. So if an item of clothing is made, it must be able to function in various conditions, in all environments, with the maximum number of avatars and objects present. To help you to make sure that your content will work in these environments, indications of memory, rendering time (GPU), processing time (CPU) and network usage, and parameters, are provided in the documentation and TRCs.

PS Home Resources

PS Home has a certain amount of memory available (Main, Host and VRAM), which is divided into 'slots' that are allocated to various functions. Some of these slots are reserved for functions native to PS Home (for example, communications).

Host RAM includes:

- Model data (and is analogous to the size of any .mdl files)
- Particle data

VRAM includes:

- Textures (which include normal maps and lightmaps, and is analogous to the size of any .dds files, as well as any reflective textures)
- Images
- Videos
- All content available via screens

Main RAM includes everything else, such as:

- Sounds/audio
- Scripts

- Script variables
- Script libraries
- Animation data
- Lighting/Probe data
- Collision data

Resource Memory Overrides

All resource files (except texture, model and particle effects) use MAIN memory by default. The same individual memory restrictions apply to the different pools, but you can choose from which memory pool the object resource is taken. If you want to use HOST memory instead of MAIN memory, you can allocate memory to the specified pool by setting the memory property in the Object Editor.

The exceptions are:

- Models, which are always in HOST. The only thing that is moved to MAIN is the header data for the model.
- Textures, which are always in VRAM. The only thing that is moved to MAIN or HOST is the Header data for the texture.

For more information on how to change the type of memory from main to host, see Adding and Deleting Object Resources.

Animated Textures

Objects that have animated textures have exactly the same memory budgets as objects that use ordinary textures.

> For example
>
> If a furniture item uses animated textures, the animated textures have no more memory allocated than an ordinary texture, and must fit in the overall budget for the furniture item.

Profiling Memory Usage

Some content has a different memory profile in runtime than it does in the tools. This content should not just be automatically validated by the tools, but also profiled to ensure adherence to memory limits.

> For example
>
> Dynamic reflective textures put an additional load on the VRAM that cannot be calculated by the automatic tools validation. Use profiling to calculate resource usage.

Some content, such as Companion objects, uses network replication and must therefore be tested and profiled remotely. See HDK Tools Validations and Profiling in the Client.

Memory Limits for Spaces

> For Japanese Live Page
> 最新の日本語版ページはこちら 「シーンのメモリ制限」

To give a true indication of the resource usage, you must test spaces against memory limits with all the animations, particle effects, scripts and objects associated with the scene active. Objects (such as animated models) that are incorporated directly into a scene prior to packaging may sometimes have no individual limits specified. However, the scene (with these objects incorporated) must conform to the scene limits.

Home Spaces

The following table shows a feature summary for each type of Home space. For HDK validations and memory limit details for Personal and Public Spaces and Clubhouses, see Space Validations.

Feature	Public Space	Public Space (No Video)	Game Space (16 players)	Game Space (32 players)	Game Space (60 players)	Video Space	Arcade Space	Personal Space	Personal Video Space	Personal Space (No Video)	Clubhouse
Capacity	60	60	16	32	60	16	60	12	12	12	32
Soft Capacity[1]	45	45	12	24	45	12	45	N/A	N/A	N/A	N/A

Access Restrictions	None	None	None	None	None	None	None	Owner, Invitee	Owner, Invitee	Owner, Invitee	Club Leader, Members
Portables (Companion Objects)	Y	Y	N	N	N	Y	Y	Y	Y	Y	Y
Scene Editor Screens	Y	N	N	N	N	Y	N	Y	Y	N	Y
Video (including Arcade Games)	Y	N	N	N	N	Y	N	Y	Y	N	Y
720p HD Video	N	N	N	N	N	Y	N	N	Y	N	N
Web Browser	Y	N	N	N	N	N	N	Y	N	N	Y
Adobe® Flash® Player	Y	N	N	N	N	N	N	N	N	N	N
Placeable Furniture	N	N	N	N	N	N	N	Y	N	Y	Y

[1] For 'public spaces, a new instance of the space is generated by the server when all instances have reached their soft capacity.

⚠ Game Spaces and Video Spaces (16 players) cannot be directly accessed from menus (the XMB™, Navigator or Menu Screen) but must be entered via a Public Space or Clubhouse, using any of the Relocate functions. See Relocating to Instances.
Game Spaces (32 players) can be directly accessed from menus (the XMB™, Navigator or Menu Screen).

See also PS Home Spaces and Launch into PS Home from XMB™ or PS3 Title.

ℹ Adobe® Flash® Player is only supported via the web browser. For web pages opened from within PS Home, only Flash 7 is currently supported.

Memory Limits by Scene Type

The following table summarizes maximum memory limits for each Scene Type. For HDK validations and memory limit details for Personal and Public Spaces and Clubhouses, see Space Validations.

	Public Space	Public Space (No Video)	Game Space (16 Players)	Game Space (32 Players)	Game Space (60 Players)	Video Space	Arcade Space	Personal Space	Personal Video Space	Personal Space (No Video)	Clubhouse
MAIN (MB)	12.8	31	43	42	38	7.8	30.8	8.5	7.8	32.6	8.5
HOST (MB)	17.4	17.4	34	30.7	25	31.9	17.4	10	31.9	10	10
VRAM (MB)	74	90	107.3	106.5	105.2	87	74	55.1	87	61.1	55.1
Screen MAIN (MB)	28	0	0	0	0	43	0	28	43	0	28
Screen VRAM (MB)	16	0	0	0	0	16	0	16	16	0	16
Meshes	4096	4096	4096	4096	4096	4096	4096	4096	4096	4096	4096
Speed (Frame rate)	24 ms/f	24 ms/f	24 ms/f	24 ms/f	24 ms/f	24 ms/f	24 ms/f	16 ms/f	16 ms/f	16 ms/f	16 ms/f

ℹ Personal Space and Clubhouse memory limits are for the scene only, and not items that users can place (such as furniture). User items have their own memory limits that are independent of Personal Space or Clubhouse scene memory.

⚠ Note that:

- Scene memory is used by the scene itself, as well as any scripted objects (mini-games, arcade games, environment animation scripts) embedded in the scene.
- Within the VRAM budget for screens there is a 6 MB maximum for displaying images. Should video use more than 10 MB, it reduces the memory available for displaying images (up to the maximum VRAM available for screens).
- The time in which scenes render and process should generally not exceed the limits set. Occasional spikes are unavoidable and are allowed as long as they are not too frequent and sustained.
- Mesh limits do not include 60 avatars, but do include the avatar's menu pad (physical representation of accessing the Menu Screen).

See also:

- HDK Tools Validations
- Scenes
- PS Home Spaces

Memory Limits for Scripted Objects

For Japanese Live Page
最新の日本語版ページはこちら 「スクリプト込みオブジェクトのメモリ制限」

Scripted objects include mini-games, arcade games, game launching scripts and model animation (rigid body animation and joint animation) scripts. Scene memory is used by the scene itself, as well as any scripted objects embedded in the scene.

For validation and memory limits for Active items, see Active Item Validations. For the manual validations required and the memory limits for companion objects, see Profiling Companion Objects.

Object	VRAM	MAIN	HOST	MESHES	OTHER
Includes:	Textures Images Video Content on screens	Sounds /audio Scripts Script variables Script libraries Animation data	Model data Particle data	Meshes in runtime, i.e. mesh-material combinations (in DCC tools)	
Arcade Games in scenes	10 MB	10 MB	-	Must fit within scene limits - e.g. 4,096	⚠ Arcade Games: Limits apply to the arcade game being played in a scene and all the other elements of the scene are active. The Arcade Game must fit within scene limits - including recommended network and rendering/processing rate. For Arcade Games in Furniture, the limits apply to the Arcade Game being played, not the furniture it is associated with.
Arcade Games in furniture	10 MB	10 MB	-	-	See note above about arcade games
Mini-Games in scenes	Must fit within scene budget - e.g. 74 MB (public space) or 54 MB (personal spaces and clubhouses)	Must fit within scene budget- e.g. 12 MB (public space) or 8.5 MB (personal spaces and clubhouses)	Must fit within scene budget- e.g. 17 MB (public space) or 10 MB (personal spaces and clubhouses)	Must fit within scene limits - e.g. 4,096	Must fit within scene limits - e.g. recommended network and rendering/processing rates

Game Launch Objects	1 MB	1 MB	128 KB	N/A	The maximum CPU time: 4 ms/f (recommended 2 ms/f)
Portable Objects: • Companions • Locomotion Objects • Animation Packs	512 KB (local) 256 KB (remote)	256 KB (local) 128 KB (remote)	256 KB (local) 128 KB (remote)	Companions = 3 Locomotion Objects = 6 Animation Packs = 6	Each type of portable template (companions, locomotion objects, animation packs) has compulsory script resources. Each one uses a different amount of memory out of this pool available to portable items. Make sure your total object memory usage does not exceed the total memory allowance for each memory pool for portables. See Profiling Companion Objects for more details. For an estimate of the memory you have available in addition to the template resource usage (i.e. total portables budget minus template resource usage), these are very roughly the amounts: • Companions VRAM / MAIN / HOST = 512 / 150 / 230 KB (local) and 256 / 50 / 115 KB (remote) • Locomotion Objects VRAM / MAIN / HOST = 512 / 150 / 230 KB (local) and 256 / 50 / 115 KB (remote) • Animation Packs VRAM / MAIN / HOST = 512 / 190 / 250 KB (local) and 256 / 100 / 120 KB (remote)

See also:

- Game Components
- Game Launch Objects

Memory Limits for Non-Scripted Objects

For Japanese Live Page
最新の日本語版ページはこちら 「スクリプト無しオブジェクトのメモリ制限」

For memory limits for:

- Clothing: See the Character Components (Clothing) Validations > Memory Limits section
- Custom avatar animation: See the Custom Avatar Animation Validations > Memory Limits section
- Furniture: See the Furniture Validations > Memory Limits section
- Companion objects: See the Profiling Companion Objects > Memory Limits section
- Active items: See the Active Item Validations > Memory Limits section

Memory limits for mini-games, arcade games and game launch objects are listed with the Memory Limits for Scripted Objects.

Models and Meshes

For Japanese Live Page
最新の日本語版ページはこちら 「モデルとメッシュ」

There is a limit on the number of runtime meshes that scenes and objects can have:

- Scenes: 4,096 meshes (including all reflection views, dynamic shadows, dynamic point lights, the avatar's Menu Screen, embedded objects and mini-games)
- Furniture Objects: 5 meshes
- Character Components: 11 meshes (i.e. 5 meshes for LOD1, 3 meshes for LOD2 and 3 meshes for LOD3)

Meshes in Maya and at Runtime

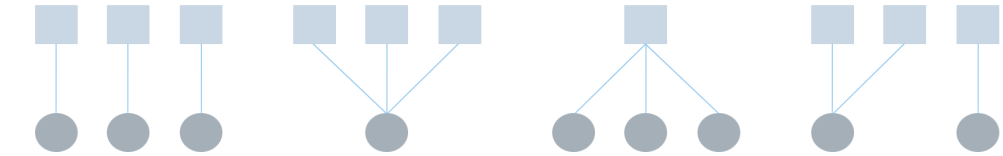
The number of meshes in Maya is different from the number of meshes at runtime in PS Home. This means that the number of meshes must be checked manually using the profiling tools to ensure the limit is not exceeded.

In Maya the mesh count can be considerably lower than in the runtime, because a mesh can contain multiple materials and still be treated as a single mesh, whereas the runtime combines triangles and materials to form meshes. The runtime mesh count also includes reflection views and dynamic shadows which drastically increase the number of meshes.

It can therefore be said that the runtime looks at mesh and material combinations to calculate the total number of meshes. So, three meshes each with their own material will result in exactly the same mesh count at runtime as three meshes sharing a single material.

Example

All of the following material and mesh combinations will result in 3 meshes at runtime:



= meshes
= materials

Meshes Loaded Dynamically in Runtime

The scene mesh count is made up of the number of meshes in the scene's combined models as well as the maximum number of dynamically called models (via scripts such as mini-games).
In the Scene Editor, various models from Maya can be combined in to form a single scene.

In the runtime, additional meshes may be added to a space if scripts call models, for instance if a mini-game loads a set of 3D tokens and a game board.

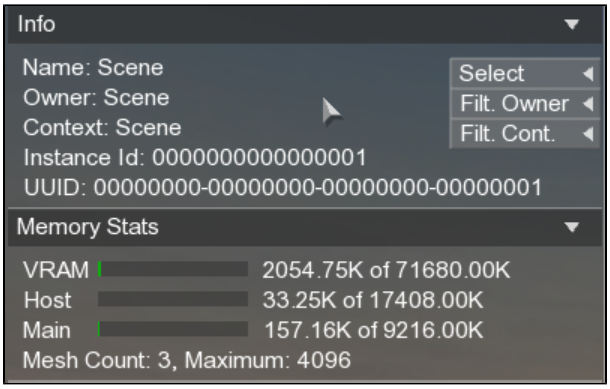
Remember that the scene mesh count must not exceed the maximum of 4,096 at any time, including when it loads additional meshes dynamically.

Counting Meshes in PS Home

Number of Meshes in a Scene

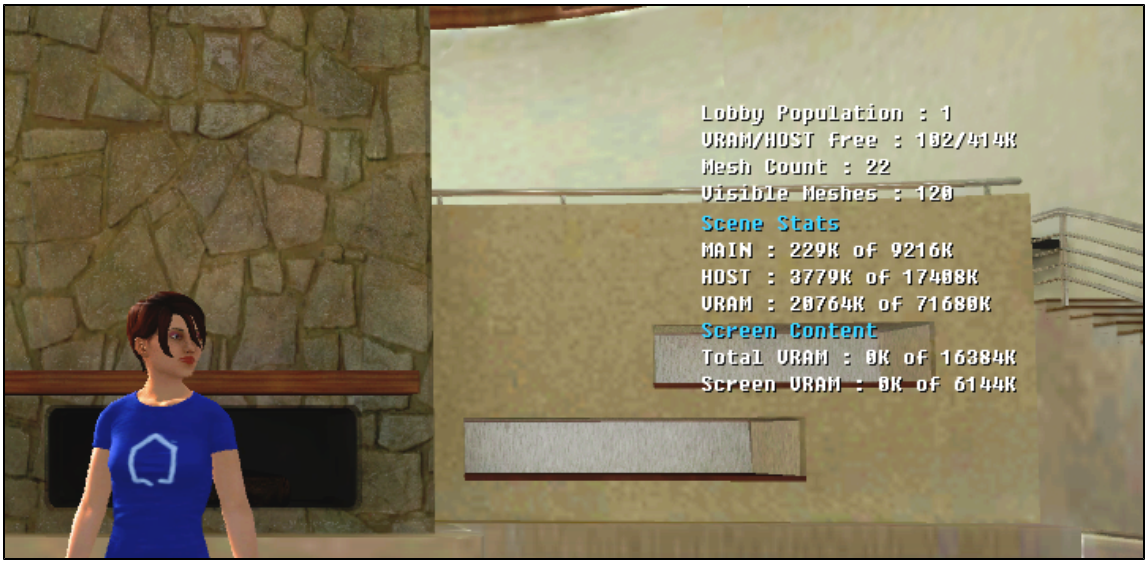
To check that the number of meshes in the scene falls within requirements:

1. Open the Profile GUI/QA Profiler by going to Select > Dev debug > QA Profiler.
2. Select Scene in the Name field.
3. Add the memory stats panel by entering ProfileGui.CreatePanel MemStats. The Mesh Count for the scene is displayed:



QA Testing

When QA test a scene they use the Visible Meshes number in the profiler, and not the Mesh Count in the MemStats panel:



The `Visible Meshes` value shows the total meshes visible to a local user. When QA test scenes, they fill the scene with the maximum number of users for that scene and check the `Visible Meshes` value. If `Visible Meshes` are above 4,096, the scene fails QA.

Number of Meshes in an Object

To check the number of meshes in an object:

1. Open the Profile GUI in the same way as described for scenes.
2. In the `Name field`, select object you want to test.
3. Ensure the number of meshes in the furniture object does not exceed 5, or 11 in a clothing object (5 for LOD1, 3 meshes for LOD2 and 3 meshes for LOD3).

Optimizing Scenes with Meshes

Combining objects together as a single mesh can reduce GPU processing time and improve the performance of the scene.

For example:

- If there are similar objects grouped next to each other (such as hand rail struts on a staircase), they could be rendered as one mesh.
- If there are similar objects grouped far apart (such as trees at opposite ends of a square in a Space), should be rendered as separate meshes so the environment doesn't render objects people cannot actually see

Combining meshes can also decrease rendering performance, as view frustum culling possibilities might be limited when meshes are combined (view frustum culling is applied per mesh bound).

Dimensions

Design all content with the user in mind - from how far they have to walk to whether content is in line of sight for the avatar. Everything also has to fit with other content, content not necessarily made by you.

This means you have to pay attention to dimensions and scale.

For example

If the environment is a series of viewing platforms at the top of the tree canopy in a rainforest, the avatars must be able to move easily between platforms without having to climb down to the forest floor and back up each time. You could create a scripted model that transports avatars direct from one location to another. That way it stays the same space, and is not constrained in size by how far an avatar can walk, but still provides a good experience for players.

There are specific scale and dimensions for environments, components and objects. Make sure that you keep to these guidelines so that your worlds have the correct scale and dimensions for Home avatars to interact with.

Related information:

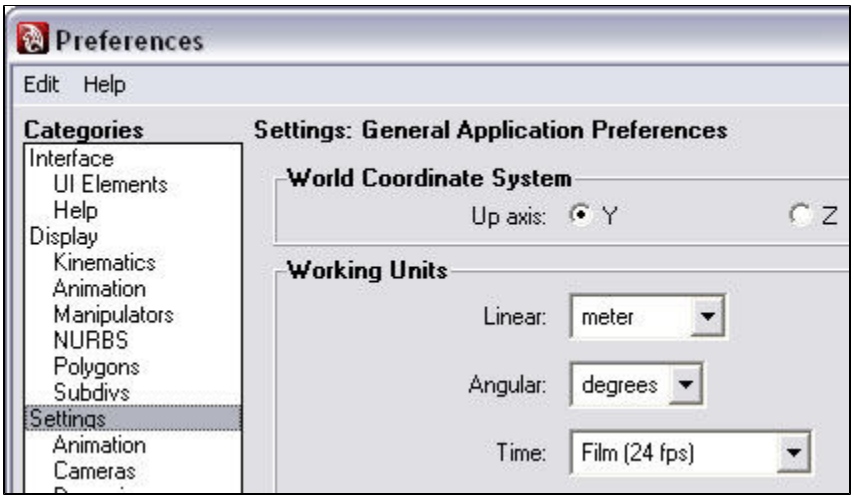
- PS Home Spaces for information on designing spaces

- Environments and Scale for the considerations of scale in environments
- Profiling Character Components and Batch Validator Clothing Limits for information on validating clothing dimensions and recommended clothing dimensions.
- HDK Tools Validations for information on memory limits and other restrictions.

> For a demonstration of what these scales look like in a space, see the Gymnasium sample, available from https://home.scedev.net/projects/samples.

Working Units

Make sure that your Maya Working Units are always set to meters, as shown below:



Avatar Height

The maximum height for an avatar is 196 cm, excluding any hat or hair component.

> Character components contain no collision, so if the hat is very big it might clip through a low ceiling.

Keep avatar height in mind when designing furniture and spaces, to avoid clipping above the avatar's head. Allow at least a 48 cm clearance between the tallest possible avatar and the ceiling height.

Clothing Dimensions

PS Home does not have exact required maximum dimensions for character components. To maximize creativity, the system is intentionally flexible. The suggested limits are guidelines to help you to design suitable character components that fit well into your environment, and are most likely to pass QA.

The recommended limits are considered good guidelines for making character components, but bear in mind that components will be working with other components that were not designed to match each other. In some cases, therefore, keeping within the recommended limits does not guarantee that the components will pass QA or will work well with other components in PS Home.

For example

A shirt with tight sleeves but particularly wide cuffs could intersect with large gloves.

Conversely, in some cases, exceeding the recommended limits can yield content that is worthwhile and may still be passed by QA.

For example

A pair of overly large earrings that intersect through a hairstyle will look perfectly natural in most cases. Or a lion head FullHeadDress could have a mane that extends down an avatar's back. The mane may overlap on a shirt, but if designed well, it can appear to always sit on top of whatever torso component the avatar is wearing.

Ideally, all components should avoid overlapping other slots, so that no component intersection occurs. If they do extend beyond the maximum recommendations, design them so that they cope well with overlapping components, or they might fail QA.

The following tables outlines the recommended character component dimensions. You can profile these components using the `debugClothingLimits` console command.

Node	Description	Sphere Radius (meters)
`debugClothingLimitsHand`	Hand	0.25
`debugClothingLimitsFoot`	Foot	0.25
`debugClothingLimitsLeg`	Leg	0.25
`debugClothingLimitsHead`	Head	0.40
`debugClothingLimitsSpine`	Torso	0.45

The Clothing Dimensions Profiler does not display for the Torso component. No sphere appears around an avatar's torso, but the recommendation is still 45 cm.



For information on profiling clothing dimensions, see Profiling Character Components and Batch Validator Clothing Limits.

Clubhouse Bulletin Screen Dimensions

All clubhouses must have a bulletin screen. The bulletin screen must have a texture set, a screen ID and the height and width specified. The height and width of the message area and the margins must be a specific ratio.

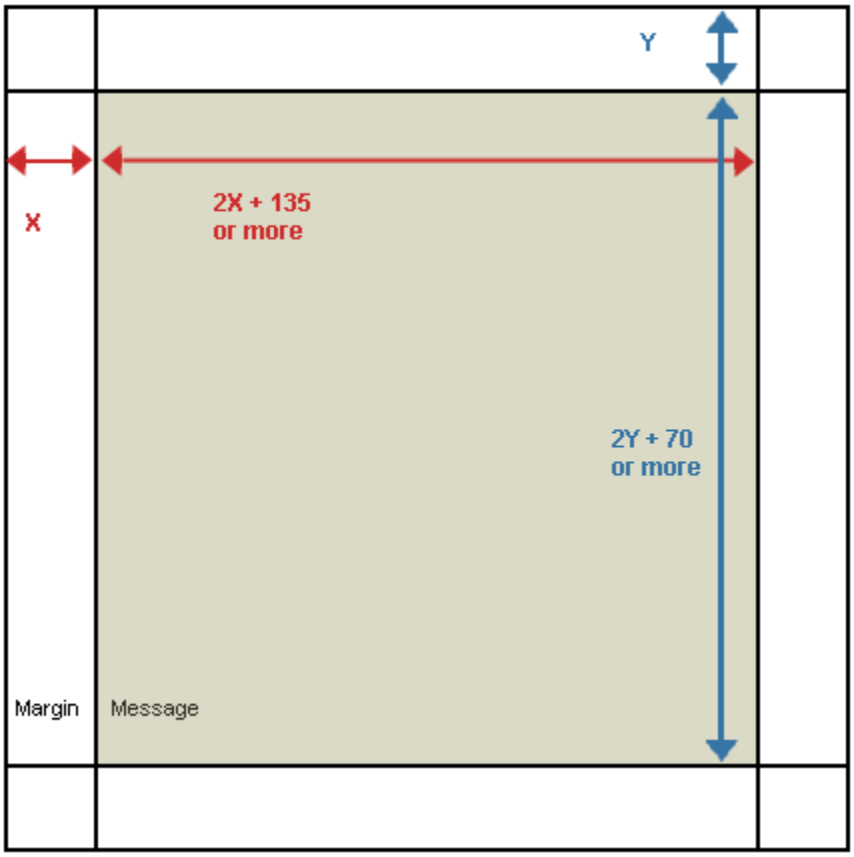
> The Scene Editor automatically validates the dimensions of screens marked as clubhouse bulletin screens.

To work out the appropriate height/width of the message and the margin, use the following formula (where X is width, and Y is

height):

- MessageX cannot be smaller than MessageMarginX*2 + 135
- MessageY cannot be smaller than MessageMarginY*2 + 70

The following image shows a graphic representation of these settings:



For more information on Clubhouses and their bulletin screens, see Clubhouses.

Companion Object Dimensions

The recommended dimension for companion objects are:

Dimension	Value	Description
Height	0.4 m	There is no set height limit, but the Companion Object 'thinks' that it is 0.4 m tall. It calculates how to follow users based on that height.
Width	0.4 m	This is a safe measurement to use for the width, as well as for the overall Companion Object dimensions. Limit the total size to within a 0.4 m radius sphere from the ground. If a Companion Object is any wider, it clips when moving around corners and intricate geometry. If it is any taller, it has unwanted behavior when trying to follow a user.

⚠ Always test Companion objects around obstacles, to make sure that it elegantly navigates around collision.

Companion objects that are very small and likely to fall through collision. If they are very large, they are likely to get stuck, such as in scene collision with low ceilings or narrow spaces. In these cases, they will not pass QA.

Furniture Dimensions

The recommended size for furniture is:

- maximum dimensions: 3 (x) x 2.44 (y) x 4 (z) m
- minimum dimensions: 0.1 (x) x 0.1 (y) x 0.1 (z) m

⚠ Thin surfaces can cause problems with collision and result in small objects falling through.

Picture Frame Dimensions

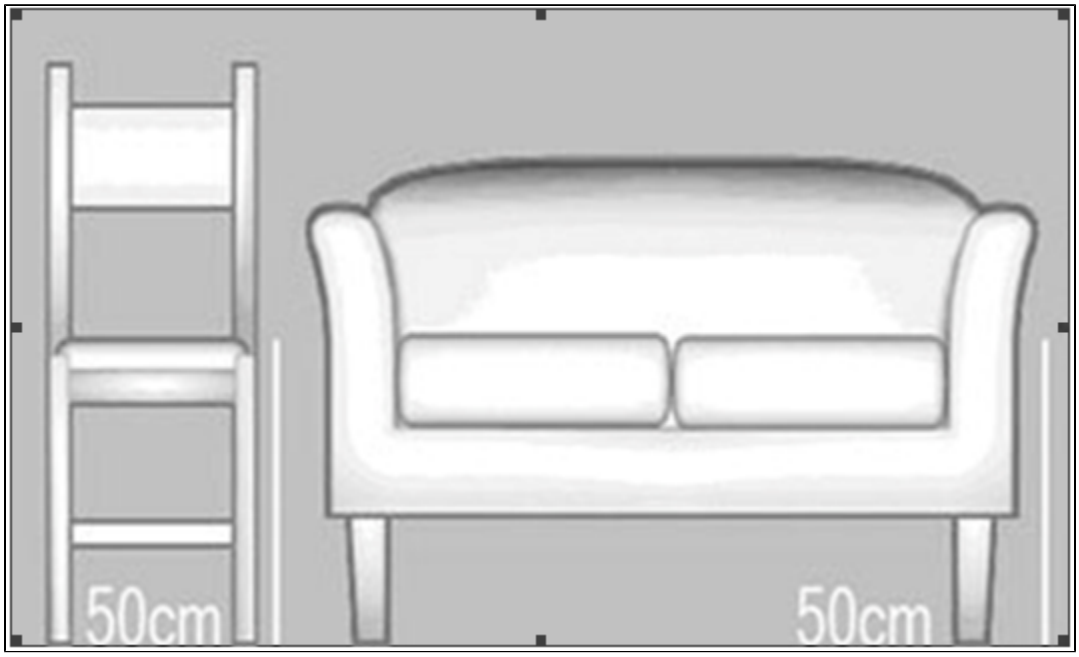
The required maximum dimensions for picture frames is:

- 2 m (X axis) x 2 m (Y axis) x 0.3 m (Z axis - e.g. protrusion)

These dimensions are automatically validated.

Seating Height

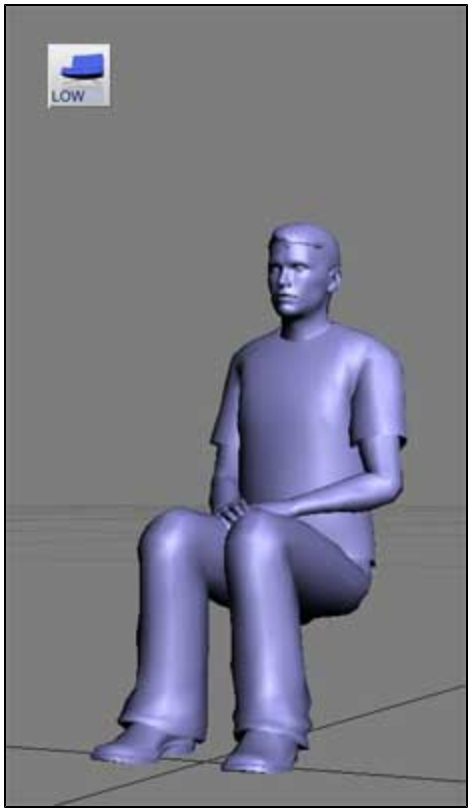
To make sure that the avatar animation for sitting looks realistic, it is important to get the seat height of the character seat locator right:



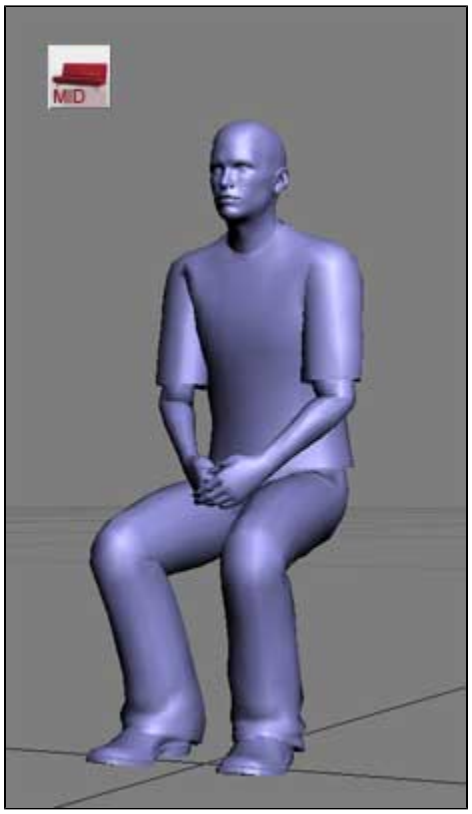
You can choose from three seat heights in the Furniture Tool Shelf in Maya (Home_Furn):

- Low
- Mid (Medium)
- High

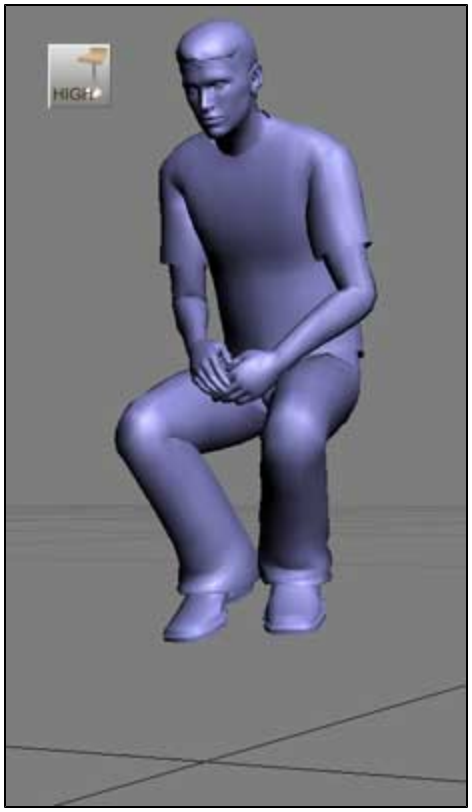
Low - low or reclining seat height (0.35 m)



Medium – standard seat height (0.5 m)



High – bar stool style seat height (0.7 m)



The seat heights are not exact measurements, so place your Character Seat Locators in Maya before modeling your seat. You can then build the geometry as far as possible to fit the avatar pose.

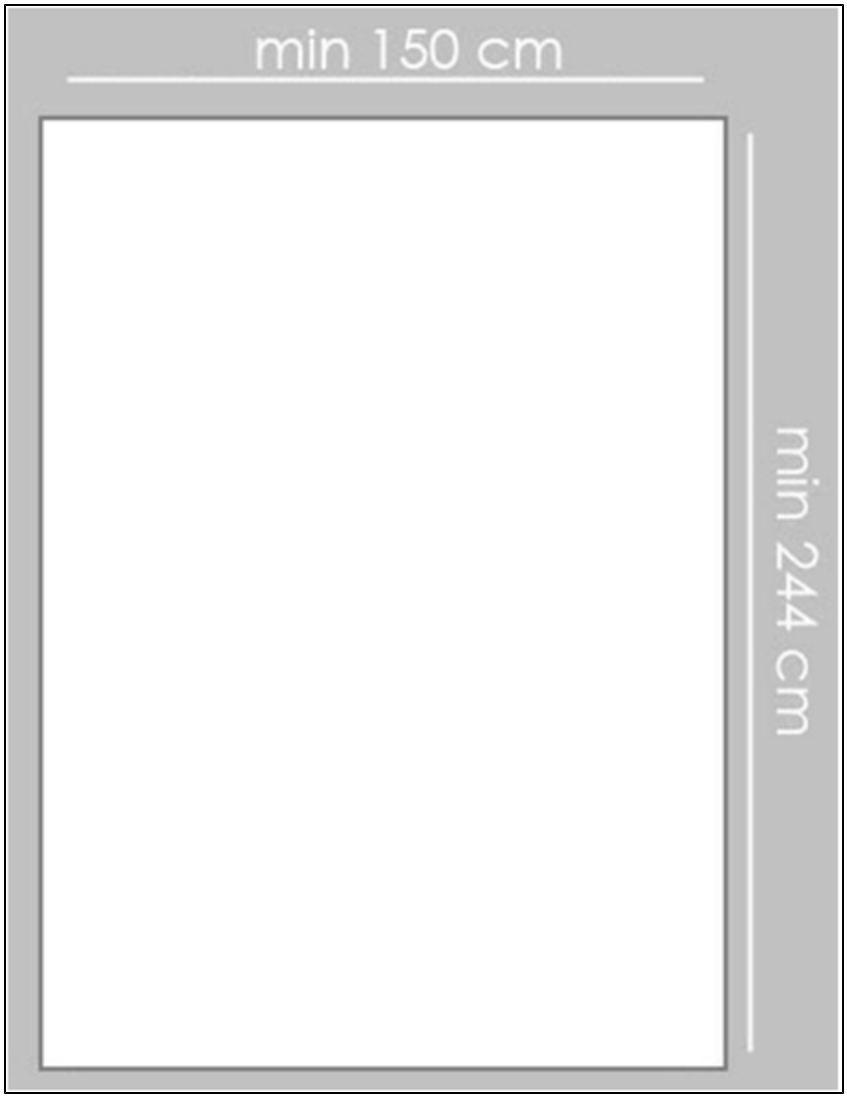
For more information on creating seats, see Creating a Seat.

PS Home Spaces

The recommended size for spaces is automatically validated by the HDK tools at:

- maximum dimensions: 10,000 (x) x 5,000 (y) x 10,000 (z) m
- minimum dimensions: 1.5 (x) x 2.44 (y) x 1.5 (z) m

The minimum size for a space (such as a corridor) is the smallest space between any parts of an environment that still allows two avatars to pass one another without getting stuck.



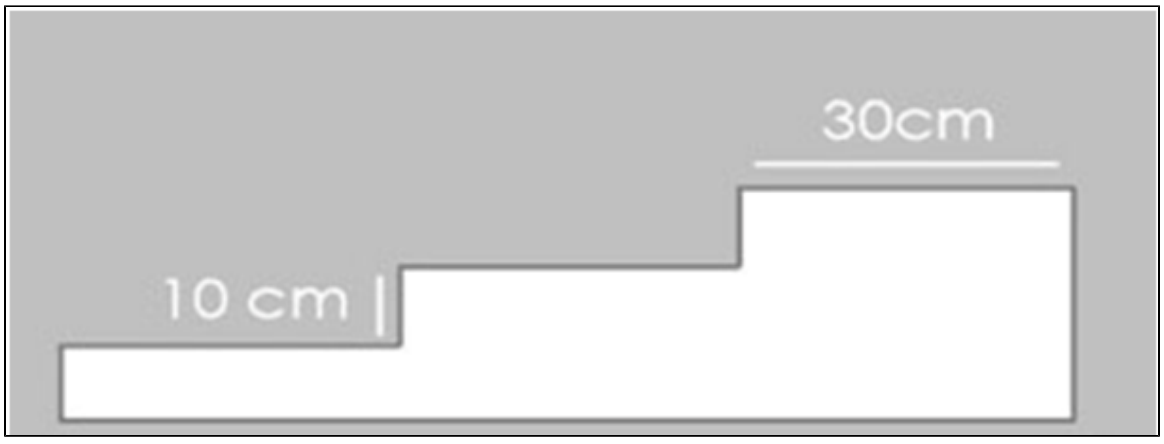
For more guidance on things to consider when designing areas for your players to be in, see PS Home Spaces.

Steps

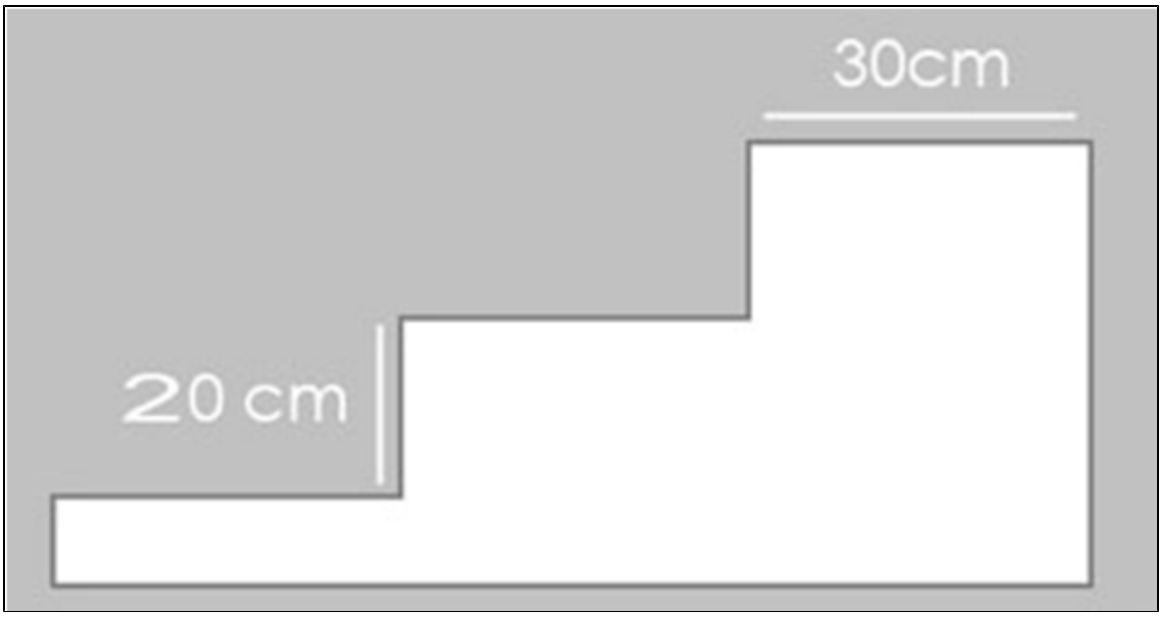
Avatar animations work with two different step pitches. If you do not use these pitches, avatar animation may look inaccurate. The steep pitch works better than the shallow pitch, but be careful not to make the pitch too steep. The collision for stairs is a slope, so avatars may slide down the slope if it is too steep. The steepest pitch you can have is a 2:3 scale, or 33 degrees.

The recommended pitches for steps are:

Shallow pitch:



Steep pitch:

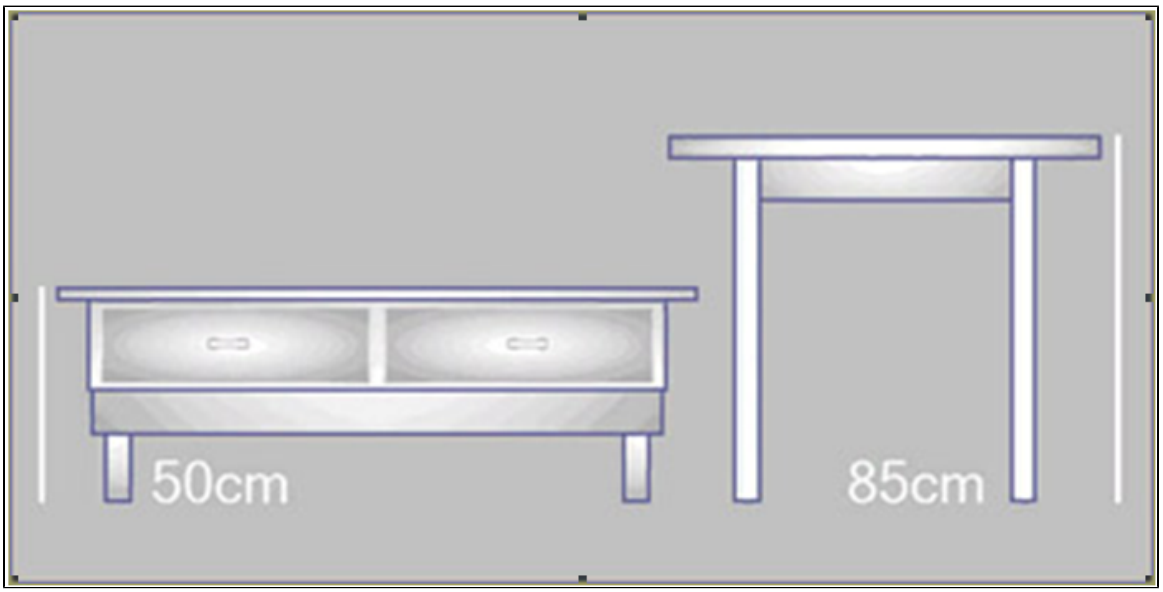


Collision for steps should be a smooth ramp. For example, at 18.5 degrees for shallow pitch stairs, and 33 degrees for steep pitch stairs.

Surface Height

You use surface height for items such as tables and counters.

The recommended height for surfaces for furniture is:



Thumbnails

For Japanese Live Page
最新の日本語版ページはこちら 「サムネイル」

When creating any thumbnail, keep in mind that users select content based on what they see in the thumbnail. Make thumbnails both consistent and illustrative. Strike a balance between consistency (thereby creating a better user interface for users to

navigate), and flexibility, which allows different content to be shown in the best and most accurate way.

For example

Creating a single component thumbnail for a Helghast vest is straightforward. See Clothing Thumbnails for more information.

However, if you are making a special bundle of items to sell, the appropriate thumbnail for the bundle might look different to the standard format of a single item's thumbnail. See Unusual Thumbnails for further guidance.

For a bundle that includes a complete Helghast outfit (with an item for every character component slot), it might be appropriate to show an avatar equipped with all the components, or it might be appropriate to show all the components arranged separately to show they are individual items and not a composite).

On the other hand, if your bundle includes not only this complete Helghast outfit, but also a limited edition furniture and command center Personal Space, it might be more appropriate to have an abstract KillZone® themed image to identify it is a special KillZone® bundle and the details of what the bundle contains are only listed in the metadata description.

The requirements and guidelines help you to standardize the way content in PS Home is displayed.

Requirements

Make sure your thumbnail meets the following requirements:

- Every scene must have a large and a small thumbnail.
- Every object (except arcade games, Game Launch objects, and realtime games/mini-games in scenes) must have a small thumbnail.
- Commercially available content must have large and small thumbnails.
- You must specify a publisher name, or company name for commercially available content.
- For all bundled content, a representation of the fact that it is a bundle and a representative image of the bundled content must be used.
- Thumbnail sizes:
    - Small object thumbnails (128 x 128) include two areas of blank space at the top and bottom, of 13 pixels each (except for small Environment thumbnails, which do not have any blank space, and Navigator thumbnails, which have a blank space at the top and bottom, of 8 pixels each).
    - Large thumbnails (320 x 176) have no blank space reserved and the image can be positioned to make use of the full are of the 320x176 thumbnail area.
    - Thumbnails must be in .png format.

The smaller thumbnails appear in the menus in PS Home (currently on chips as shown below), larger thumbnails appear in the PlayStation®Store and as displayed on the relocation screens for scenes.

⚠ Note that:

- Arcade games, game launch objects and realtime games/mini-games in scenes are not currently visible from the PS Home menu system, but this may change. Thumbnail images in the formats specified are therefore recommended for these objects.
- We recommend that you include small, large and publisher thumbnails for all scenes and all objects.
- Some menus display images showing more of the thumbnail than others. Use the guidelines for the area of blank space (making sure there is no important information and that it is the same color as the rest of the background).

General Rules

- Keep the thumbnail image centered and as big as possible within the frame.
- Avoid cropping the image if possible (in some cases cropping is unavoidable).
- Character components do not cast shadows.



- All other objects (including furniture items) do cast shadows.



- Environments should fill the entire 128x128 (and 320x176) thumbnail images.

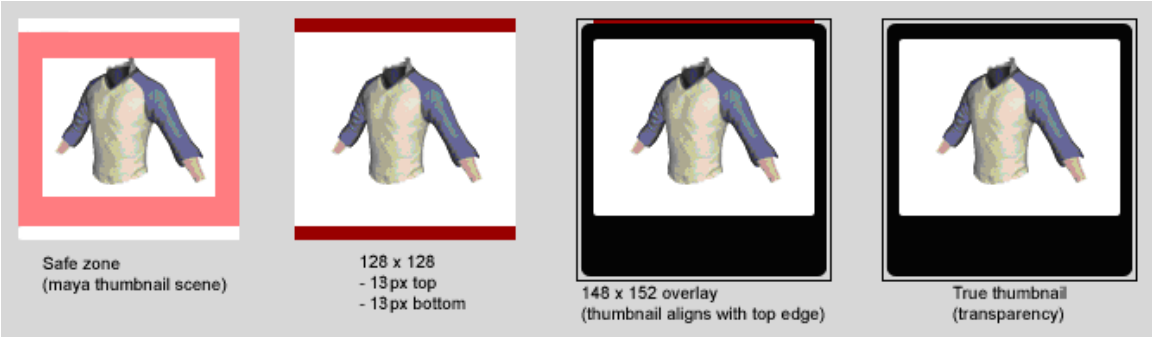


- Do not mislead or falsely represent.

Art Assets

Use the CHIP template provided to make sure that all the Maya generated thumbnails for character components are framed correctly. The chip graphic is a part of the PS Home UI and navigation style and overlays all thumbnail images.

Position your image according to the example below:



For a white dummy head and various other assets for making thumbnails, see the _thumbnail_scene.ma files in the Artsource folder under each component type. This process is described in the workflow sections of the Scene Thumbnails, Furniture Thumbnails and Clothing Thumbnails sections.

Thumbnail Room

You can also use the Thumbnail Room packaged in the HDK to easily create thumbnails for your objects.

To access the thumbnail room from the DEV DEBUG menu, go to Scene List > thumbnailroom.

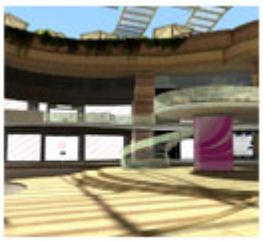
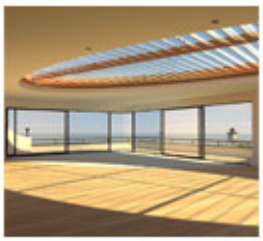
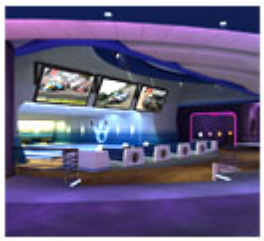
You can use the console command shadowEnable to turn shadows on and off.

Scene Thumbnails

> For Japanese Live Page
> 最新の日本語版ページはこちら 「シーンのサムネイル」

When working with thumbnails, create PS Home environment thumbnails that fill the entire image canvas. For exteriors, use a three-quarter perspective shot wherever possible. For interiors, use a view that shows the scene at its best.

The following examples show both types of thumbnail:

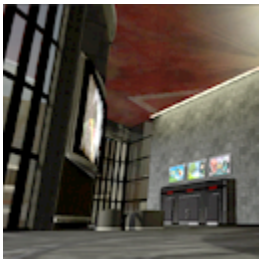











Not all environments have an identical perspective for the small and large thumbnails:




For the larger (320x176) thumbnail image of the environment, the thumbnail is a rectangle and the composition for the smaller square thumbnail may not be suitable. You can take a screenshot from a different angle, providing that the environment is recognizably the same in both perspectives and sizes.

Scene Entitlement Objects

Home spaces as rewards or items that can be purchased require a scene entitlement object to confer ownership on users. This scene entitlement object must have the same thumbnails as the scene it represents.

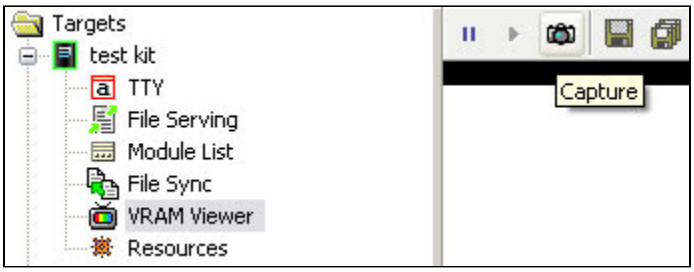
See Scene Entitlement Objects for information on how to create a scene entitlement object.

Creating an Environment Thumbnail

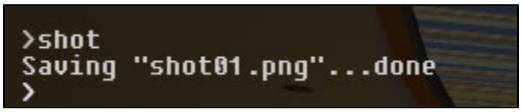
To create an environment thumbnail, you can capture a screenshot from the console and save it to your build folder as follows:

1. Move your Home avatar out of sight so that it is not visible in the shot.
2. Hold down the right stick on the controller to toggle between the game camera and the free camera. Ensure Dip Switch 0 is set to on by typing `setDipSwitch 0 1` in the console command.
3. Move the right stick to compose a good, representative shot of the scene.
4. When in position, get a screen capture using one of the following methods:

    - Using Target Manager Capture: From the Target Manager, take a screen capture of the PlayStation®3 using the VRAM Viewer:

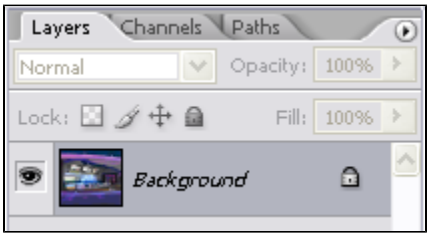
    - Using the Console command shot: Press the SELECT button on the controller to open the debug console. Enter the command `shot` to capture the current screen and automatically save a file named shot01.png to the HDK build folder. The debug console displays updates to reflect the file save:

    > To hide the debug console so that it does not appear in the screenshot, type `consoleNumLines 0` in the console command.

    - Using F4: Press F4 on the keyboard connected to the PlayStation®3. This produces the same result as typing the console command `shot`, but without the debug console being visible. The only problem with this method is that the chat bar appears automatically when you use the keyboard. Position items so that the chat bar does not obscure their view.

5. Load the image into Photoshop to process it. Select the crop tool. Hold down the Shift Key when selecting the area to crop, to keep the selection square. The overall size of the crop is not important at this stage.
6. Select Layer > Flatten Image to reduce the image to a single Photoshop layer:



7. Size the image down to 128 x 128 pixels.
8. Save the thumbnail in .png format.

Navigator Thumbnails

> For Japanese Live Page
> 最新の日本語版ページはこちら 「ナビゲーターのサムネイル」

For the Navigator, you will need to provide the following:

- Thumbnails
- Display Names (localized)
- Descriptions (localized)
- Scene Names
- Regional Availability
- Which stack the chip should be in

See below for more detailed requirements on each item.

Detailed Requirements

Thumbnails

- .png format
- 320x176 pixels (with an 8 pixel transparent border top and one at the bottom)
- Typically this is a cropped screenshot showing off a particularly recognizable part of the space. See the guidelines for Scene Thumbnails for details.

Wherever possible, use a three-quarter perspective shot for exteriors and, for interiors, a view that shows the scene at its best.

Display Name (maximum 32 characters)

Display Names are short phrases indicating the name, function and branding of the space. For example:

- Bowling Alley
- Home Theater
- Red Bull Air Race
- Uncharted™: Sully's Bar

Currently, the name of a scene can be a maximum of 32 characters long.

> ⚠ The Display Name should be the same as that submitted to the Content Delivery System (CDS) when uploading your Scene. See HDK Tools Validations.

Description (maximum 256 characters)

Descriptions are no more than a sentence or two describing a space in more detail and outlining some of the features on offer. For example:

- Bowling Alley: Have some fun and challenge other people at the exciting games on offer in the Bowling Alley.
- Home Theater: Keep up with the latest new releases. Watch game and movie trailers on ten exclusive screens in the Home Theater.
- Red Bull Air Race:The Red Bull Air Race World Series is a motor sport requiring speed, precision and skill. Experience the thrill at Red Bull Home.
- Uncharted™: Sully's Bar: A social gathering spot from the early 20th Century that has somehow weathered decades of storms and conflict. Sully's is a rustic lounge and watering hold set deep in the tropics.

> ⚠ The Description should be the same as that submitted to the CDS when uploading your Scene.

Scene Name

The Scene Name is produced by us at HTS Support after asking developers for the suggested name and a description. This is the Scene Name used when submitting the Scene on the CDS. We produce a unique Scene Name and associated Scene Key - these two are linked to each other.

The Scene Name and Scene Key are obtained by contacting the Regional Support Teams.

Regional Availability

The following is required for every scene:

- Specify the regions the space will be available in (SCEA/SCEE/SCEJ/SCEAsia). Each scene must be available in at least one region up to a maximum of all four. Making a scene available in all four regions means it is global.
- (Scene Chips) Provide localized text for each region - that includes Description and Scene Name (details of each above).

The name and description of each scene must be localized into all of the languages of the regions it is available in. These languages are broken down as follows:

SCEA

- English
- French

SCEE

- English
- French
- Italian

- German
- Spanish

SCEJ

- Japanese

SCEAsia

- English
- Korean
- Traditional Chinese (Taiwan)
- Traditional Chinese (Hong Kong)

All localization should be submitted as a spreadsheet with one row per phrase and one column for each language.

Furniture Thumbnails

For Japanese Live Page
最新の日本語版ページはこちら 「家具のサムネイル」

Requirements

For furniture thumbnails, always use a three-quarter perspective shot.

If possible, always create furniture thumbnail images on a white background (RGB 255,255,255). The shadow fall-off for objects with white backgrounds is to the right, as in the following examples:



If a white background is not appropriate, use a black background (RGB 0,0,0). For objects that have a black background, you do not need a shadow fall-off.

If you feel that a white or black background is not appropriate, please contact your SCE Regional Manager to discuss whether an exception to this rule is possible.

Because all Home assets have ATG Shaders applied, you cannot render these thumbnail views in Maya to obtain the shadowing, so you must create the screenshots on the console. For this purpose, we have provided a template Maya scene in:

```
\build\environments\thumbnailroom\ThumbnailRoom.scene
```

Similar requirements to those for furniture thumbnails apply to the following items:

- Companion objects
- Actives

> If you have difficulty displaying the effects for furniture or active furniture with special functions, you can create a thumbnail that describes the function or load condition, rather than creating a thumbnail for the model itself. For more information, see Actives.

Creating Furniture Thumbnails

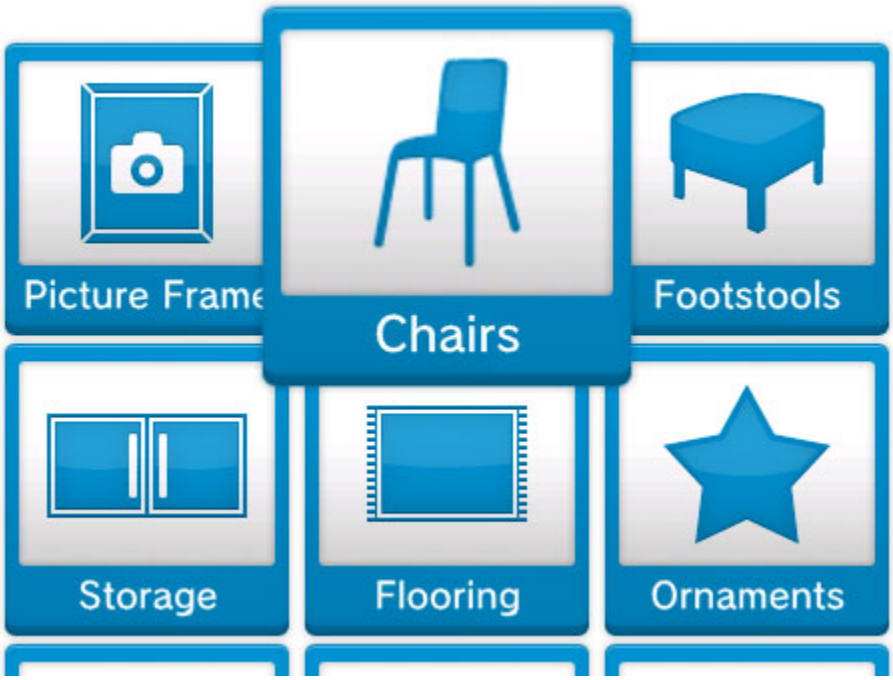
To create furniture item thumbnails, take a screenshot from the console and save it to your PC's build folder.

Taking the Screenshot

1. Run the HomeDeveloper.self using the Target Manager.

   If you want to run offline you must load a specific valid scene using the --scene parameter. For more information, see Launching the Home Developer SELF (although it doesn't matter which scene you use).

2. Select DEV DEBUG > Scene List > thumbnailroom.
3. Press the START button and then select Redecorate.
4. Choose the category of furniture that your item was exported as. This is initially decided in the Object Details dialog before export from Maya.



The chip has the default question mark graphic until it is replaced by the thumbnail you are creating:



5. Use the furniture placement tools to move the item to a position and rotation that produces the required three-quarter view, with its shadow cast in the specified direction:



For smaller objects, you might be able to move the item away from the avatar, rotate it to get the correct shadow direction, then rotate the camera to the appropriate position for the screenshot.

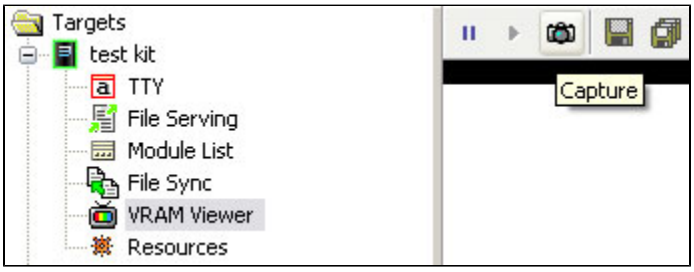
For larger objects, you must place the item in the scene and then use the free camera (by clicking the Right Stick to toggle it on and off) to frame the item appropriately for the screenshot.



6. Choose how you want to capture the image.
    - You can either use the console command `shot`:



    - Or take a screenshot of the PlayStation®3, using the Target Manager VRAM Viewer:



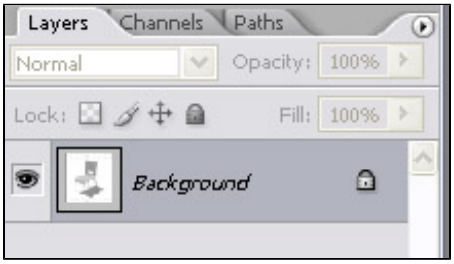
Loading the Image into Photoshop

When you have taken the screenshot, load it into Photoshop, as follows:

1. Load the image into Photoshop.
2. Crop it as closely as possible, but leave enough room for the image to fit comfortably on the thumbnail chip.

Hold down the Shift Key when selecting the area to crop, to keep the selection square (although the precise size is not so important at this stage).

3. Choose Layer > Flatten Image to reduce the image to a single Photoshop Layer:



4. Size the image down to 128 x 128 pixels.
5. Save the thumbnail in .png format.

Clothing Thumbnails

For Japanese Live Page
最新の日本語版ページはこちら 「服のサムネイル」

Requirements

All character components must have the following:

- Always create these thumbnail images on a white background (RGB 255,255,255), but where a white background is not appropriate, you can use a black background (RGB 0,0,0).
- You must always use a three-quarter perspective shot.
- Most of the time a shot focusing on the particular area of the avatar the component is located is appropriate, although in some exceptions a zoomed in view is necessary (for example jewelry, see details below).
- There are no shadows for Character components.
- Keep the view from the front, unless there is specific detail on the rear that requires display, such as a backpack (or clothing details like the spikes and tail in the torso and leg examples below).

If you feel that a white or black background is not appropriate, please contact your SCE Regional Manager to discuss whether an exception to this rule is possible.

If you have difficulty displaying the effects for a character component with special functions or problems loading a character component, you can create a thumbnail that describes the function or load condition, rather than creating a thumbnail for the model itself. For more information, see Actives.

See examples of all the types of character component and further tips below:

Hair and Hats

- white background (RGB 255,255,255)*
- three-quarter perspective
- no shadows

*Where a white background is not appropriate, you can use a black background (RGB 0,0,0).





Torso

- white background (RGB 255,255,255)*
- three-quarter perspective
- no shadows

*Where a white background is not appropriate you can us a black background (RGB 0,0,0).








Hands

- white background (RGB 255,255,255)*
- three-quarter perspective
- no shadows

*Where a white background is not appropriate you can us a black background (RGB 0,0,0).



Legs

- white background (RGB 255,255,255)*
- three-quarter perspective
- no shadows

*Where a white background is not appropriate you can us a black background (RGB 0,0,0).





Feet

- white background (RGB 255,255,255)*
- three-quarter perspective
- no shadows

*Where a white background is not appropriate you can us a black background (RGB 0,0,0).










Accessories

- white background (RGB 255,255,255)*
- three-quarter perspective
- zoomed in to show details
- no shadows

*Where a white background is not appropriate you can us a black background (RGB 0,0,0).

For smaller accessories, a zoomed in view (similar to Jewelry below) is allowed.



Jewelry

- white background (RGB 255,255,255)*
- three-quarter perspective
- zoomed in to show details
- no shadows

*Where a white background is not appropriate you can us a black background (RGB 0,0,0).

Jewelry is often small, so it is advisable to have a zoomed in view.



Legs+Feet (LegsAndFeet)

- white background (RGB 255,255,255)*
- three-quarter perspective
- no shadows

*Where a white background is not appropriate you can us a black background (RGB 0,0,0).



Torso+Legs (TorsoAndLegs)

- white background (RGB 255,255,255)*
- three-quarter perspective
- no shadows

*Where a white background is not appropriate you can us a black background (RGB 0,0,0).



Torso+Legs+Feet (TorsoLegsAndFeet)

- white background (RGB 255,255,255)*
- three-quarter perspective
- no shadows

*Where a white background is not appropriate you can us a black background (RGB 0,0,0).



FullBodySuit

- white background (RGB 255,255,255)*
- three-quarter perspective
- default camera (similar to Torso+Legs+Feet) or a zoomed in perspective, whichever is more appropriate
- no shadows

*Where a white background is not appropriate you can us a black background (RGB 0,0,0).



OR





FullBodySuit with Custom Animations
Some FullBodySuits have custom animations that you may wish to allude to in the thumbnails. As with Companion Objects, a particular pose showing the type of animation of the FullBodySuit may be more appropriate than the rigid rig default.

See Unusual Thumbnails for more general guidance on displaying dynamic behavior.

HeadDress and FullHeadDress

- white background (RGB 255,255,255)*
- three-quarter perspective
- no shadows

*Where a white background is not appropriate you can us a black background (RGB 0,0,0).

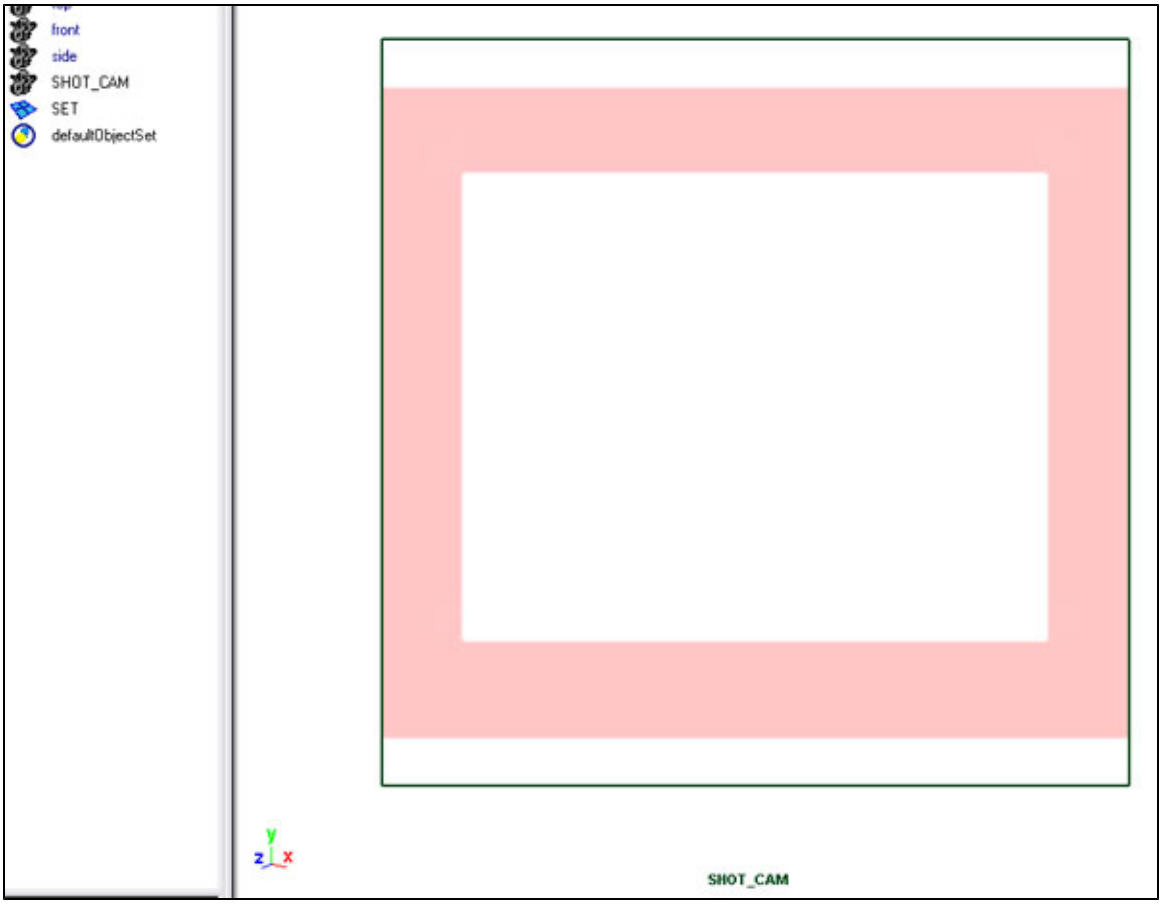


Creating Clothing Thumbnails

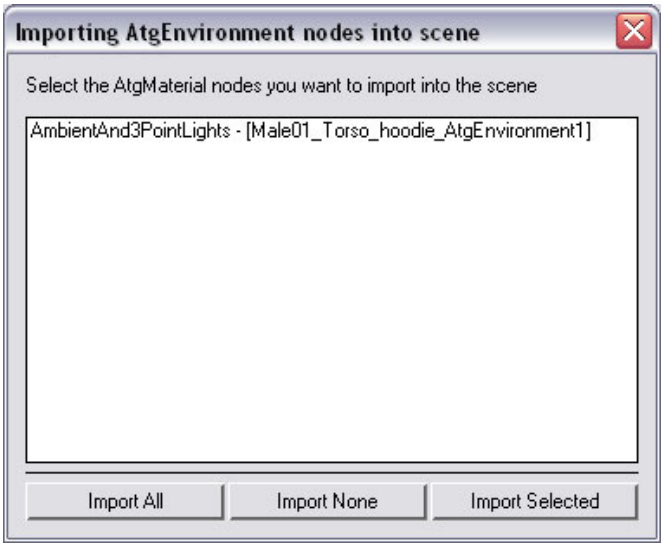
To create clothing thumbnails, a screenshot can be captured on your PC from the DCC tool (Maya) and saved to the build folder as follows:

1. Firstly load the Maya template file for the component type you wish to create a thumbnail for. For example, for a male torso object you would load Male01_Torso_thumbnail_scene.ma from `\artsource\Characters_V2\Male\Male01\Torso`. You will find a similar template file for each class of character component in both the Male and Female artsource branches. For example should you need to create a thumbnail for a female hair asset you would load the Female01_Hair_thumbnail_scene.ma from the `artsource\Characters_V2\Female\Female01\Hair` folder.

   This gives you a chip template object to frame your asset as a thumbnail, and a correctly positioned camera (SHOT_CAM) to provide the optimum point of view:



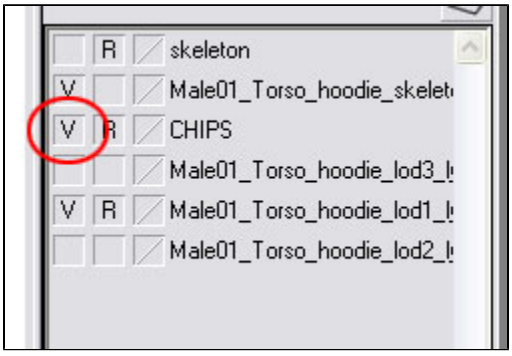
2. Next import your torso asset into Maya using File > Import. If you encounter the Importing nodes into scene dialog (as below), click Import None:



3. You then need to delete everything imported apart from the LOD1 torso object which should be left perfectly framed in the three quarter position:



4. Uncheck visibility (V) for the CHIPS Layer in the Layers panel to make the chip template guide object invisible:



This leaves you with just the framed asset, correctly placed, and the thumbnail border outlined in black:



5. Use the windows Print Screen command to copy and paste the image into Photoshop, then crop it as closely as needed inside the black border. Remember to hold down the Shift Key when selecting the area to crop, to keep the selection square although the precise size is not important at this stage, as you will resize the image in step 7 below.
6. Select Layer > Flatten Image to reduce the image to a single Photoshop Layer:



7. Size the image down to 128 x 128 pixels.
8. Save the thumbnail in .png format to complete the task.

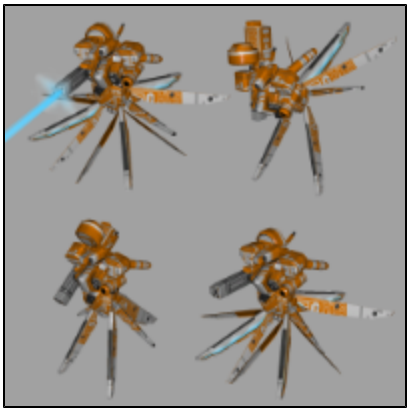
Companion Thumbnails

For Japanese Live Page
最新の日本語版ページはこちら 「コンパニオンオブジェクトのサムネイル」

For all companion objects, follow the requirements for Furniture Thumbnails.

However, if your companion object has a particular animation behavior you wish to display, you may want to capture this in the thumbnail.

Perhaps a particular pose or a combination of poses illustrating the various animations and behaviors would be more appropriate.



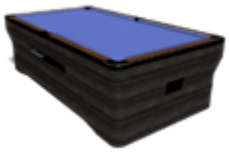
See Unusual Thumbnails for more general guidance on displaying dynamic behavior.

Active Item Thumbnails

> For Japanese Live Page
> 最新の日本語版ページはこちら 「アドバンスド家具のサムネイル」

For actives that are similar to furniture, the thumbnail criteria are the same as those for Furniture Thumbnails.

For example, an active that is a game of chess on a board is amply illustrated by a chessboard object and a game of pool is amply illustrated by a pool table.



If the active's capabilities or features would not be suitably displayed by adhering to the requirements for furniture thumbnails, you can display it slightly more abstractly.

For example, an active that is a tree that changes through four seasons may have a thumbnail image showing a single tree, but with each quarter of the thumbnail depicting one of the states of the tree (spring, summer, autumn, winter).

Another example would be to illustrate some dynamic behavior of the active, for instance a the trajectory of a ball.



See Unusual Thumbnails for more general guidance on displaying dynamic behavior.

Unusual Thumbnails

> For Japanese Live Page
> 最新の日本語版ページはこちら 「その他のサムネイル」

There are several other types of content that are not as narrowly defined as the standard requirements for thumbnails. This other

content usually falls broadly into these categories:

- Content that is a combination of things (for instance bundles or collections).
- Content that has dynamic behavior or changes state.
- Things that do not have a physical manifestation in PS Home (intangible content).

General thumbnail rules still tend to apply (for example, a three quarter perspective). However, there are other considerations that mean a judgment call specific to the contention what is appropriately representative rather than applying a general rule.

The aim is not to mislead users in any way. Therefore, the best advice is to apply common sense and speak to your Home Account Manager if you are in doubt about whether your content will pass Quality Assurance (QA) successfully.

Requirements

Bundles

Bundles of objects must have a representation of the fact that they are bundles and a representative image of the content must be used. In some cases an abstract image would be considered appropriately representative.





Consumable and Rentable Items

If items have a set number of uses before they expire, or a time limit on their use, make this clear to users. This could be in the thumbnail itself, or (in exceptional circumstances) in the description metadata only.

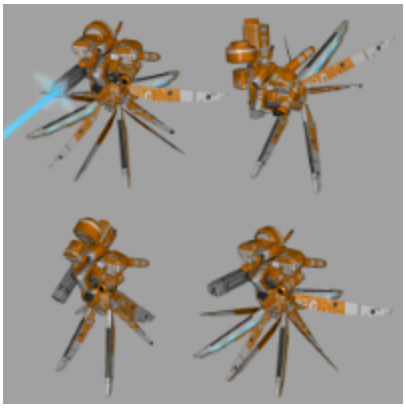


Companion Objects

For all companion objects, follow the requirements for Furniture Thumbnails.

However, if your companion object has a particular animation behavior you wish to display, you may want to capture this in the thumbnail. Perhaps a particular pose or a combination of poses illustrating the various animations and behaviors would be more appropriate.

Alternatively, various images showing a range of movement may be a good indicator of the variations in a companion's routine.



FullBodySuits with Custom Animations

Some FullBodySuits have custom animations that you may wish to allude to in the thumbnails. As with Companion objects, a particular pose showing the type of animation of the FullBodySuit may be more appropriate than the rigid rig default.

Intangible Content

For all content that does not have a tangible or physical manifestation in PS Home, it can be quite difficult to portray in a thumbnail image a concept that is clear to players at a glance.

For instance, there is no rule for how to display these intangibles:

- Unlimited ammo
- Resistance to fire
- Weapons upgrades

Some abstract concepts have a very well established icon that can be associated with it (for instance health), but that is not the case for all intangible content. It is up to you to decide what image depicts the content best and communicates the most accurate information about the content to the player. This thumbnail image could be quite an abstract image.

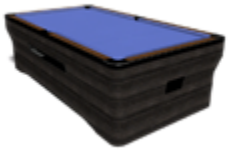
Regardless of what the thumbnail image is, the details in the descriptive metadata of the object would need to make it clear to the user what the object does. Speak to your Home Account Manager if you are in doubt about whether your content will pass QA successfully.

Actives

If the active's capabilities or features would not be suitably displayed by adhering to the requirements for furniture thumbnails, you can display it slightly more abstractly using descriptions that focus on the object's function and features. For example, Character components such as contact lenses appear by zooming into the item itself. You can use a thumbnail picture that shows the avatar wearing the contact lenses to show the effect more accurately as shown in the following example:



The same principle can be applied to furniture with special functions and active furniture. For example, an active that is a game of chess on a board is well described by a chessboard object and a game of pool is easily illustrated by a pool table.



If the active's capabilities or features would not be suitably displayed by adhering to the requirements for furniture thumbnails, you can display it slightly more abstractly. For example, an active that is a tree that changes through four seasons may have a thumbnail image showing a single tree, but with each quarter of the thumbnail depicting one of the states of the tree (spring, summer, autumn, winter). Another example would be to illustrate some dynamic behavior of the active, for instance a the trajectory of a ball.



If you wish to use these thumbnails, please contact your SCE Regional Manager to discuss whether an exception to this rule is possible. However, please note that if the thumbnail is judged inappropriate, we may have to ask you to change the thumbnail.

Age Restrictions

> For Japanese Live Page
> 最新の日本語版ページはこちら 「視聴年齢制限」

To make sure that your content works as expected, you must set the following age restrictions:

- Age rating (if required)
- Parental control level (mandatory)
- Minimum age (mandatory)

If you do not set age restrictions, default values are applied.

Age restriction data must correlate. For example, if you set your content to a minimum age of 5, you must set the parental control Level and age rating to reflect that the content is not accessible to anyone under the age of 5.

Set the minimum age and parental control level (and, where applicable, the age rating) for all regions the content is available in, or localized to, before packaging content for submission. You specify the age restrictions for different regions in the content metadata, using the Object Editor and Scene Editor.

> Edit Localization data outside the Object Editor and Scene Editor, using the master_localisation.xml file. Do not edit any other information (such as age rating or metadata) outside the Editors. Doing so can be very detrimental to your Home content.

Content that is restricted from the user cannot be bought, owned, accessed or received by that user. However, the user can still see restricted content if another avatar has the content on display. For example, if other users are wearing a restricted t-shirt, or place a piece of restricted furniture in a personal space or clubhouse, the restricted user can see them.

Age Rating

Age rating determines only whether an age rating is displayed on-screen against the content. Selecting a specific age rating does not automatically set the minimum age, and it does not determine whether or not the content is access restricted. To restrict access, you must set the minimum age and parental control level.

Age rating is required where content is associated with a game title: for example a "LittleBigPlanet™" t-shirt or a "MotorStorm®" game space. If you set the age rating, you must set it to the equivalent rating or a higher rating than the associated title.

If age rating is applicable, you must obtain it from an official regional body, such as ESRB or PEGI. For content to be released in SCEJ, get the age rating from SCEJ.

> Age rating rules differ from region to region, so contact regional SCE (through your Third-Party Relations Manager or Regional Support) for more information on the requirements for your region.

Parental Control Level

The parental control level uses the PlayStation®3 rating system, which is numbered between 1 and 11. It compares the set rating against the user's console and the current parental control setting.

The following sections outline the parental control settings for each region:

Content targeting SCEJ

1: A (All age groups)

5: B (Ages 12 and over)

7: C (Ages 15 and over)

8: D (Ages 17 and over)

9: Z (Ages 18 and over)

0: Rating Agreement or rating is pending - You cannot use this level for product versions

> SCEJ act as rating body for PS Home content, not CERO, as with game titles for SCEJ.

Content targeting SCE Asia (Korea)

3: Everyone

5: 12 or older

7: 15 or older

9: Not for teens (19 or older)

0: Rating pending - You cannot use this level for product versions

> Korea has its own rating system. For details, contact SCE Asia.

Content targeting SCE Asia (Hong Kong, Taiwan, Singapore, Malaysia, Thailand, Indonesia)

3: Everyone

5: 12 or older

7: 15 or older

9: Not for teens (19 or older)

0: Rating pending - You cannot use this level for product versions

Content targeting SCEA

2: ESRB EC - EARLY CHILDHOOD

3: ESRB E - EVERYONE

4: ESRB E10+ - EVERYONE 10+

5: ESRB T - TEEN

9: ESRB M - MATURE

10: ESRB AO - ADULTS ONLY - SCEA does not allow AO rated content to be published

0: ESRB RP - RATING PENDING - You cannot use this level in content for commercial production

Content targeting SCEE

PEGI Ratings

2: PEGI Rating Age Group 3+

3: PEGI Rating Age Group 7+

5: PEGI Rating Age Group 12+

7: PEGI Rating Age Group 16+

9: PEGI Rating Age Group 18+

There have been changes to the PEGI rating system (icons, descriptors, etc.)

USK Ratings

1: USK 0 (Without age restrictions)

3: USK 6 (Restricted for those below the age of 6)

5: USK 12 (Restricted for those below the age of 12)

7: USK 16 (Restricted for those below the age of 16)

9: USK 18 (Restricted for those below the age of 18)

OFLC Ratings

1: OFLC G (General)

3: OFLC PG (Parental Guidance)

7: OFLC M (Mature)

9: OFLC MA 15+ (Mature Audiences)

The rating board for Australia has changed to ACB. The New Zealand rating board is the OFLC New Zealand.

0: Rating pending - You cannot use this rating in final masters (content) for commercial production.

Contact regional SCE for more information regarding the parental control rating per territory, including information on rating systems that are not described here.

Minimum Age

The minimum age restriction compares a value (age) against the rating on the user's PSN account.

The minimum age that determines whether or not access to content in PS Home is restricted. If you do not set the minimum age, content is not accessible to any user of PS Home.

The age rating is used only to inform the user as to the age level of the content and to determine whether a rating is displayed on-screen against the content. If the age rating is not included, the content is still restricted by the minimum age ("age gating") but a rating is not displayed on-screen.

Making Content Inaccessible for a Region

For regions in which your content must not be displayed, set the age restrictions as follows:

- Age rating: Set the default (blank)
- Minimum age: Set the default (blank) or specify 200+ years

General Design Guidelines

For Japanese Live Page
最新の日本語版ページはこちら 「デザイン全般のガイドライン」

Bear in mind the following constraints when designing your PS Home content.

See also:

- Mini-game Design
- Queueing Best Practice

Interactions

- Interactions in PS Home must be independent of framerate.
- Interactions in PS Home must not be latency-critical.

Scripts Affecting the User State or Content

Avoid functions that might confuse the user, and use functions such as `autojoin` and `player relocation` responsibly. Script functions must not put users into states that they did not choose, or alter their content without their permission. Doing so can disrupt the PS Home user experience, confuse users, or make them feel that they do not have control of the experience.

When deciding to use automatic joining or player relocation, consider the user impact. The following examples are acceptable uses of relocation functions:

- Use the `Person.SetPosition` function to relocate players from a portal to a different area. This function creates behavior that the user is expecting (and would therefore count as user initiated as well).
- Set the join type to `auto` or `minimal` for a mini-game that encompasses the whole space. These join types join the user automatically when they relocate to that space, making their experience of PS Home more fluid.

There are many more instances where you can use these and other functions creatively and sensibly to enhance the user experience.

Avoid the following:

- Do not track or record user activity, as doing so might infringe their privacy rights (see Information Transmission Constraints).
- Do not break the flow of the experience in PS Home by launching a user into a state that is perceptibly different from their current state, and not one that they would expect in the circumstances. For example:
    - Do not suddenly open a web browser if the user does not prompt such an action nor approach any area to initiate this action.
    - Do not launch the user into a game title or relocate the user to an unexpected location that is not immediately obviously connected to the environment or activity they were engaged in prior to relocation.

Contact your Regional Account Manager to discuss cases where you feel a user would benefit from automating an interaction.

Saving and Loading Data

Save and load data sparingly. A script should read from the save file when it is first needed, then cache the information, to make sure that the information is readily available. Reserve accessing the HDD only to make changes to the data.

The following requirements are from the Technical Requirements Checklist (TRC):

- If a load takes over 15 seconds, display a message or animated icon to prevent blank or suspended screens (see TRC R2014).
- Scripted content does not display blank, suspended or unresponsive screens for longer than 15 seconds (see TRC R2015).

Information Transmission Constraints

You must not transmit, track, store or alter sensitive or personal user information without consent. Any such action must be carried out securely.

These constraints affect:

- How you use Lua functions; when using the `HttpPostData` and `Resource` libraries from the Lua API, use the secure https protocol and not the http.
- Policies regarding transmitting information to and from players. For guidelines and details of the SCE policies on information transmission and privacy, contact your Third Party Relations Manager or Regional Support.

Gender

Players may have avatars of either gender. If your content is not clearly associated with one gender (for instance clothing), make sure your content either accommodates both rigs/genders, or warns users that the content is for one of the genders only. See, for example, Configuring Locomotion Objects, which should ideally accommodate both genders.

Save Data Guidelines

For Japanese Live Page
最新の日本語版ページはこちら 「セーブデータガイドライン」

For information on using save data services (storing and retrieving save data online) see Save Data and Save Data Service.

Save Data enables you to save and load data stored online using the Save Data Service.

There are two types of Save Data available by default in PS Home:

- Scene Save Data
- Active Save Data

Scene Save Data

Scene Save Data is available for personal spaces and clubhouses, as well as scene scripts or embedded objects.

Personal spaces and clubhouses have access to Save Data through the `Scene` library. The initial amount of memory available is 4096 bytes, but this size may increase. Always use `Scene.GetMaxSaveDataSize()` to see the maximum amount of Save Data available in a space.

Only the owner of the personal space or clubhouse can use Scene Save Data, and the owner must be present in the space.

Active Save Data

Active Save Data is available only for personal spaces (not for clubhouses).

Active items have access to Save Data through the `Active` Lua library. The amount of memory available is 4096 bytes, but this size may increase. Always use `Active.GetMaxSaveDataSize()` to see the maximum amount of Save Data available to an active.

Only the owner of the personal space can access Active Save Data, and the owner must be present in the personal space.

General Guidelines

The following guidelines apply to both Scene and Active Save Data:

- You do not need to create a UI that informs a user that a script is saving or loading data. The client should handle this automatically.
- Always use the function that returns the maximum amount of Save Data available. Any attempt to save or read beyond the maximum amount will generate an error.
- The Save Data maximum size will never decrease.
- Never rely on Save Data existing or being valid for the script to function correctly. Write your script to handle expected data gracefully, and to ignore bad or non-existent data. This situation can easily arise when multiple scripts are accessing the same Save Data slot.

Non-Player Character Guidelines

For Japanese Live Page
最新の日本語版ページはこちら 「プレイヤー以外のキャラクター（NPC）ガイドライン」

A Non-Player Character (NPC) is a character (human or otherwise) that is part of the PS Home world, either embedded in a scene or in an active item. An NPC can scripted into a mini-game, realtime game, or scene script, either as one aspect of the script or the purpose of it.

For example, an NPC can be:

- Part of a script, such as a realtime game where users collect items in a game space and return them to the NPC.
- Scripted - a sessionless mini-game is embedded in a scene that gives users directions around the scene.

If the NPC does not perform a function of a game, the ideal medium for an NPC in a scene is as a sessionless mini-game. The NPC's actions will be seen only by the local player, and there are no limits to how many users can interact with the NPC at any one time.

As with all content in PS Home, NPCs must not infringe any copyright or other legal restrictions, and they must meet all technical, age and content requirements for the PS Home client and for each region in which they are published.

Visual Guidelines

- NPCs should have a tag indicating that they are NPCs and not a user.
- NPCs should be visually distinct from other avatars. For example, they should wear clothing not available to the user.
- NPCs can talk or yell across an instance. The dialog can appear in a chat box, using the same color as for all system or moderator chat entries. All players in the vicinity can see the communication, but it should be infrequent to avoid spamming the chat log.
- Players can to communicate with the NPC through OSD input and not through the chatbox. Therefore, use chatbox NPC communications for information broadcast to the entire space, and conduct individual conversations between NPCs and players through the OSD. Communicating in this way clearly indicates it is an NPC interaction.

Functional Guidelines

- If an NPC has a specific function, such as to open a door to a new area of the scene, place it in a location related to its function. NPCs do not need to be stationary. You can use NPCs to add dynamics and personality to your scene, but limit their range of responses to their function as much as possible.
- When a player interacts with an NPC it should turn to face the player locally. If the NPC is walking or idle when an interaction is initiated, it should stop and turn to face the initiating player.
- If the NPC is not in a sessionless mini-game, it should be seen globally to turn and face any nearby player who is currently interacting with it. If another player tries to interact with it, the NPC it should turn to face that player locally.
- When an interaction is initiated with an NPC, the ensuing dialog should only be viewable locally to the player.

Audio Guidelines

- If an NPC uses audio for speech, you must localize it for each region in which the NPC is published.
- Use the function `LocalPlayer.GetHomeRegion()` to find their region and to set which audio file to play.
- Use a unique sound effect to indicate that the user initiated an interaction with the NPC.

Profiling in the Client

The HDK comes with tools that allow you to profile your content: scenes, objects, character components, and Lua scripts. These tools show you statistics, tables, graphs and other media that gauge the impact your content has on PS Home (such as rendering time), and what resources your content consumes.

This section provides an overview of the profiling tools available to you in the HDK.



The profiling tools are designed to help you with quality and compliance. However, they are a measure of usage rather than a method for automatically validating whether content is within limits or not. Do not rely on the profiling tools alone. Always check the profiler's limits against the values in the following sources:

- HDK Tools Validations
- Memory Limits
- Launching and Testing Content
- Technical Requirements Checklist

Profiling Tools

You can use the following tools, depending on what type of content you are profiling:

Tool	Description
Profile GUI	Use this GUI to profile your scenes, furniture, mini-games, realtime games, arcade games, and game launching objects placed in your scene. You open it through the Debug Console. See Profile GUI.
Dev Debug Console Commands	Use this console to profile your scenes. This tool provides a number of commands that you can use to retrieve different types of statistic. See Dev Debug Console Commands, Frame Rate and Profiling Particles.

PPU Overview Stats and Profiling	These statistics provide an easy overview of the overall performance of the scene and help you to identify the areas that may be causing frame rate issues. This profiling provides only an approximation of the use. Do not use it for validation or QA approval. See PPU Statistics.
Object Profiler	Also known as the Interactive Object Profiler. You use this tool primarily for profiling scripted objects in your scene, such as arcade games, mini-games and realtime games. It also returns rendering information for your scene and your character. See Profiling Lua Scripts.

Content Viewers

You can load and view content using the following tools:

Tool	Description
Character Viewer	Use the Character Viewer to see how well your character components display during animations. It does not provide statistics, but it does show if your components blend and render well, and fit on avatars of different sizes and in different poses. See Character Viewer.
HDK Browser	Use the HDK Browser to view components in different modes, and to display aspects such as textures, particle effects, and sky. See HDK Browser, Profiling Particles and Sky Profiling.

Profiling Online vs Profling Offline

You can profile online or offline. However, be aware of the factors that contribute to a difference in the network and memory usage reported between online and offline testing.

When an object is packaged, its assets are given unique filenames that contain its object ID. For example, an object's asset `Data\Texture\ground.dds` packages as `0123456-0123456-0123456\Data\Textures\ground.dds`. When running locally, the file `Data\Texture\ground.dds` can be used by multiple objects and the scene. However, `ground.dds` loads only once, even though it is used by many sources. Loading once can result in a smaller memory profile for an object when running locally, compared to when it is packaged and tested online.

Also, when profiling offline, the client cannot take into account factors like network conditions and other users in the same scene.

Because of the packaging of files and the absence of other users and network conditions, use offline profiling only as a guideline when creating content. Online profiling is more accurate, and QA use online when testing content that is submitted to the Content Delivery System (CDS).

Profile GUI

You can use the Profile GUI for profiling both scenes and scene objects (such as furniture and Mini-games).

You can profile scenes and all objects (except character components) in the profiling GUI:

- Scenes
- Mini-game objects
- Arcade game objects
- Game Launch objects
- Furniture
- Active items

When using the Profile GUI, you can use a mouse or your controller to navigate the menus. You can use a mouse to target objects within scenes and to interact with the slide-out menus of the profiler GUI overlay. Either connect a mouse to your PS3 or, if you do not have an additional mouse, you can emulate it using your controller by calling `ProfileGui.ControllerMouseEmu 1` (hold down L1  and move the left stick).

Opening the Profile GUI

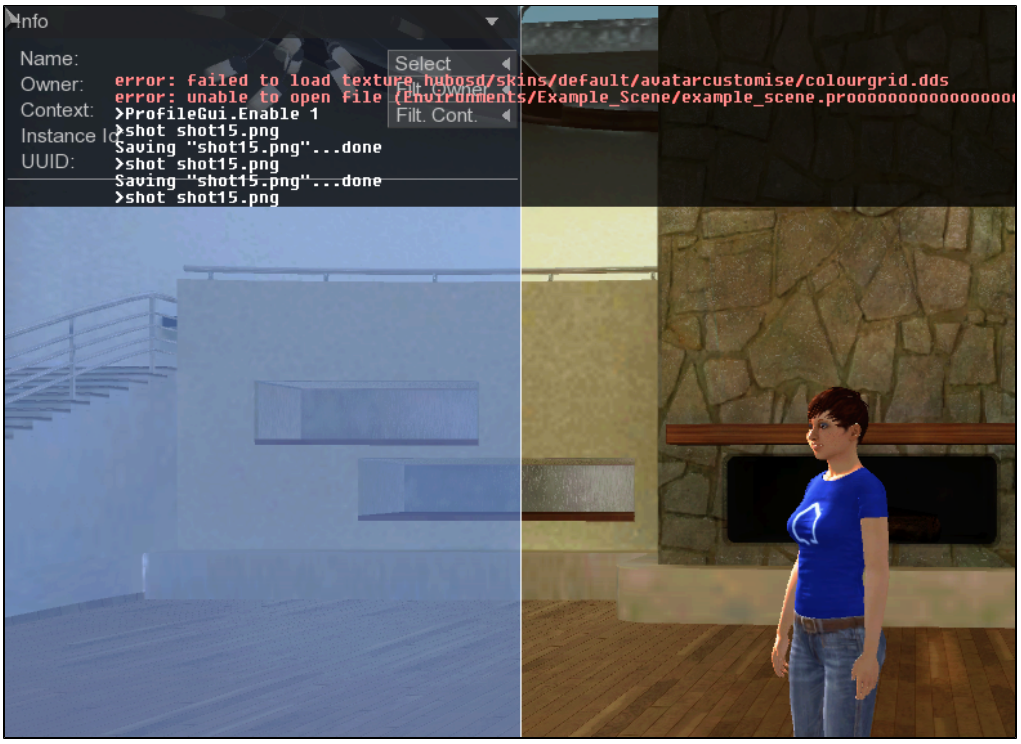
You can open the Profile GUI using the controller or through the console.

Using the Controller

1. Press [SELECT] on your controller.
2. Navigate to DEV DEBUG > QA Profiler.

Through the Console

1. Press [SELECT] on your controller.
2. Navigate to DEV DEBUG > QA Profiler.
3. Type `ProfileGui.Enable 1`.



This command opens a sidebar overlay into which you can call various profiling tools.

Panels

Use the Profile GUI to display various panels in the sidebar:

- Info Panel
- MemStats Panel
- Network Panel
- CPU Panel
- GPU Panel

Each panel contains a variety of information, for example, the Info Panel contains 5 fields of metadata.

To see a full list of the display panel options, run the command `ProfileGui.ListPanels`.

Activating a Panel

To activate a panel:

1. Enter `ProfileGui.CreatePanel < panel name >` into the console, for example, `ProfileGui.CreatePanel Info`.
2. To clear all panels, enter `ProfileGui.ClearPanels`.

Info Panel

Use this panel to retrieve information on the scene, as well as any furniture, mini-games, realtime games, arcade games, actives, and any other components that can be profiled in your scene:



- Select: Initially, this shows every component of the scene from the list above. You can filter this list by owner and context to find information on a particular component in the scene.
- Filt. Owner: Use this option to filter components of the scene by owner. The owner is a property applied to an object or scene in the Object Editor and Scene Editor.
- Filt. Cont.: Use this option to filter components by object (e.g. furniture, mini-game, active item).

Selecting an Object in the Info Panel

To select an object for profiling using the Info Panel:

1. Go to Select > <Object>, where <Object> is the name or ID of the object you want to profile.
2. Activate the CPU Panel.

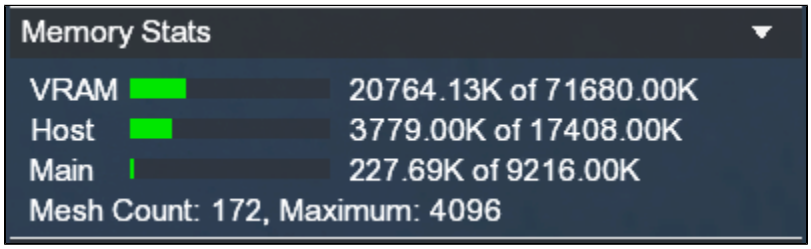
   The object's CPU usage appears on the graph. Furniture items tend to have little impact on your scene. They may appear as only a few bumps on your graph, if at all.

3. To view the legend for the colors, type Objectprofiler 1.

For more information on the Object Profiler, see Profiling Lua Scripts.

MemStats Panel

This panel shows the VRAM, HOST, and MAIN memory use for whatever is selected in the Info panel. It also shows two mesh values:



- Mesh Count: This is the runtime mesh count (i.e. total number of meshes in the scene for the object/scene selected in the Info panel). This is different to the Visible Mesh count.
- Maximum: This is maximum number of meshes. The Mesh Count cannot exceed this number.

These stats are explained in Models and Meshes.

Network Panel

This shows two tables of bandwidth usage and packets sent/received by the scene:



The Send line is yellow. The Receive line is green.

> Packets are packets of data. You can send 10 packets of 1 byte, for example, or 1 packet of 10 bytes. We recommend sending a few big packets, than many smaller ones.

CPU Panel

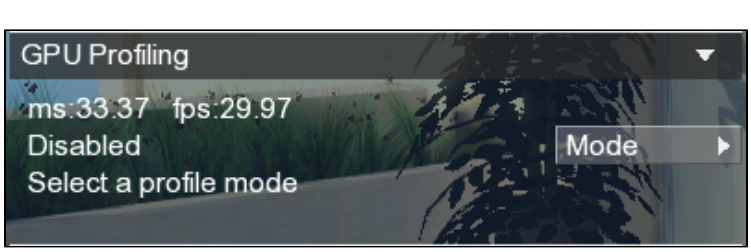
Select an object in the Info panel to display the object's CPU usage in this panel:

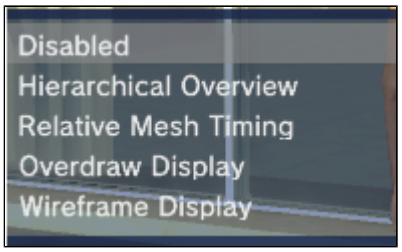


The horizontal axis information is in seconds; the vertical axis is the framerate (ms).

GPU Panel

Use this panel to view your GPU ms rate, and to select different modes for displaying your scene and their GPU rates:





For more information on the frame rate, and limits, in your scene, see Profiling Scenes.

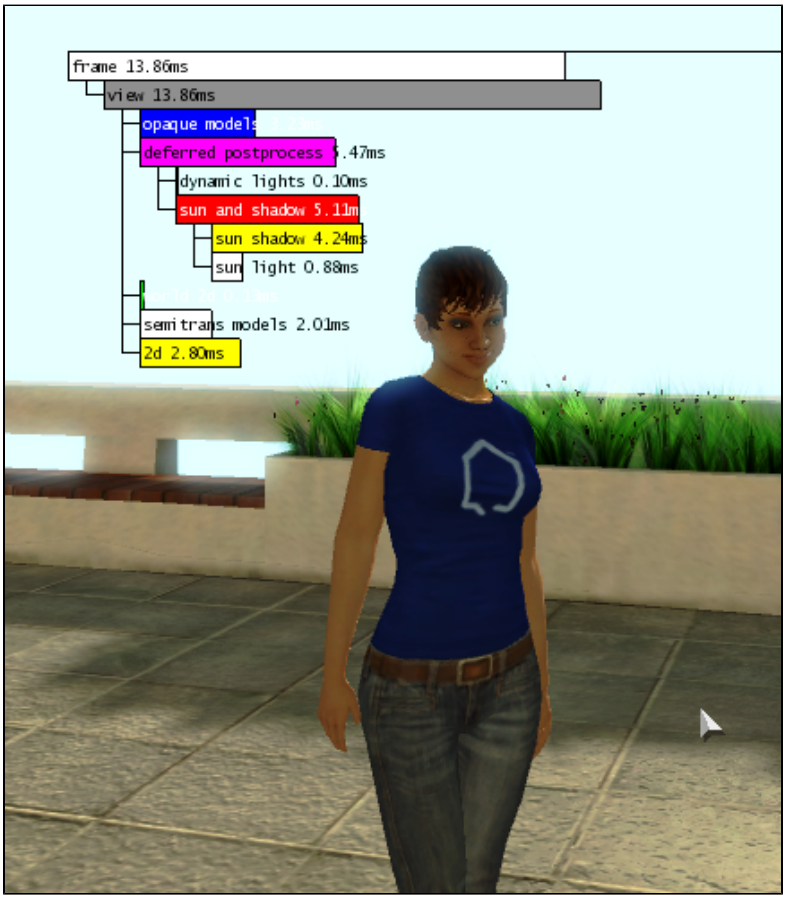
Select Mode to choose one of the GPU Profiling Modes.

Currently, neither the Relative Mesh Timing nor the Overdraw Display modes work correctly. Do not select them.

GPU Profiling Modes

Hierarchical Overview

The Hierarchical Overview displays the contributors to your frame rate. This is an easy way to see what aspects of your scene (for example, dynamic lights, sun light, opaque models) are contributing most to your frame rate:



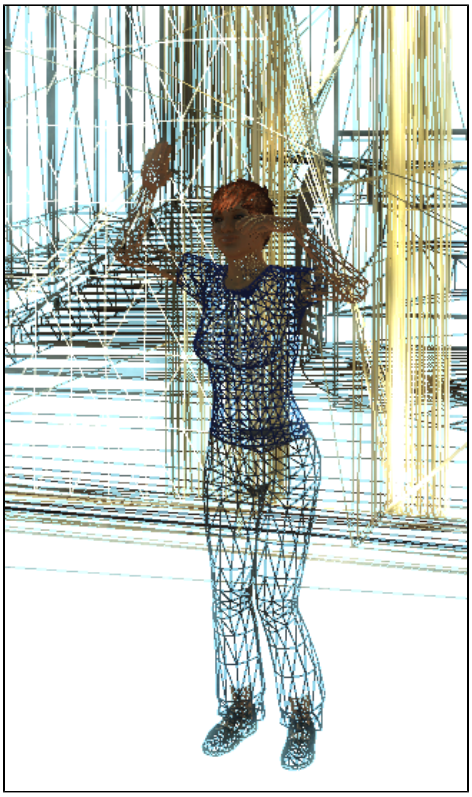
The Hierarchical Overview shows the frame rate of the following items, if they are present in the scene:

- 2D
- debug
- deferred postprocess
- dynamic lights
- fog
- forward opaque models
- opaque models
- opaque models
- particles
- point light
- semitrans models
- sky
- spot light
- sun and shadow
- sun light
- sun shadow
- view
- vistests
- world 2D
- world 2D no fog

The values for certain scene components do not appear in the overview until they have been added to the scene. For example, the bar showing the sky's frame rate does not appear unless you have added an Atmosphere File object and an .atmos file to your scene in the Scene Editor.

Wireframe Display

The Wireframe Display displays the wireframes of your scene.



Character Viewer

You use the Character Viewer to see how well the character components fit with other character components, and what they look like under different conditions.

The Character Viewer enables you to:

- Load character components from HDD.
- Change gender, fatness, LOD.
- Animate an avatar to see how well components blend, and to see if they intersect or clip.
- Swap between textured and wireframe view.
- Fix the camera to a component (even if it is animated), or move the camera freely with your pad. When you fix the camera onto a component, the camera tracks it when you animate the rig.



You cannot use the Character Viewer to check memory usage or any other stats for your character component.

For details on different profiling options, see Profiling in the Client.

Launching the Character Viewer

You can launch the Character Viewer from Maya, the Object Editor, or from the Target Manager.

From Maya

To launch the Character Viewer from Maya:

1. Open or create your character model.
2. Click 

on the Home_Char shelf or select Home > Character > PS3 Character Viewer.

From the Object Editor

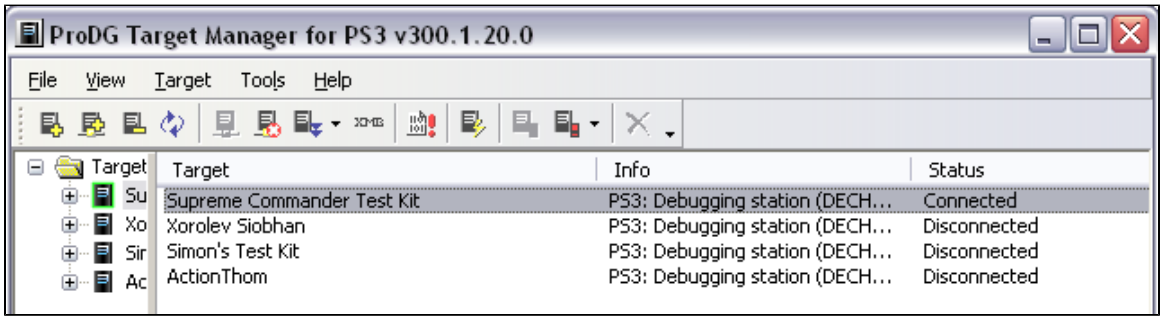
To launch the Character Viewer from the Object Editor:

1. Select the object that you want to preview.
2. Select Object > Preview on PS3.

From the Target Manager

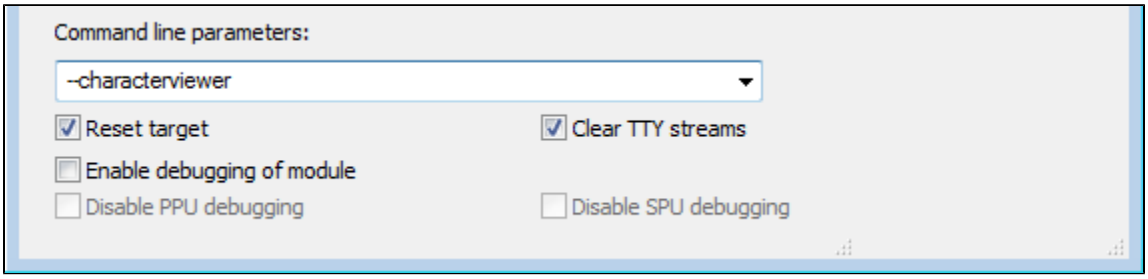
To launch the Character Viewer from Target Manager:

1. Open the Target Manager and connect to your PlayStation®3 kit:



2. Select Target > Load Run Executable or click  in the toolbar.

3. In the Load Module dialog, make sure that you are loading the .self, and enter --characterviewer in the Command line parameters field.



4. After the Character Viewer loads, use the controller to navigate the menus.

Navigating the Character Viewer

When the Character Viewer has loaded, use the controller to navigate the menus. The following table lists the different navigation options.

Option	Description
Menu Cycling	Press the [START] button on the pad to cycle the different options for viewing character components in various animations, poses and with different lighting.
Select Menu Option	Press the PAD ACCEPT button to choose.
Return to Previous Option	Press the PAD DECLINE button to return to the previous option.
Move Up	Press R1 to move the camera up.
Move Down	Press R2 to move the camera down.
Turbo Move	Hold L1 while moving with the Left Analogue Stick, or with R1/R2.
Slow Move	Hold L2 while moving with the Left Analogue Stick, or with R1/R2.
Rotate	Use the Right Analogue Stick to rotate.

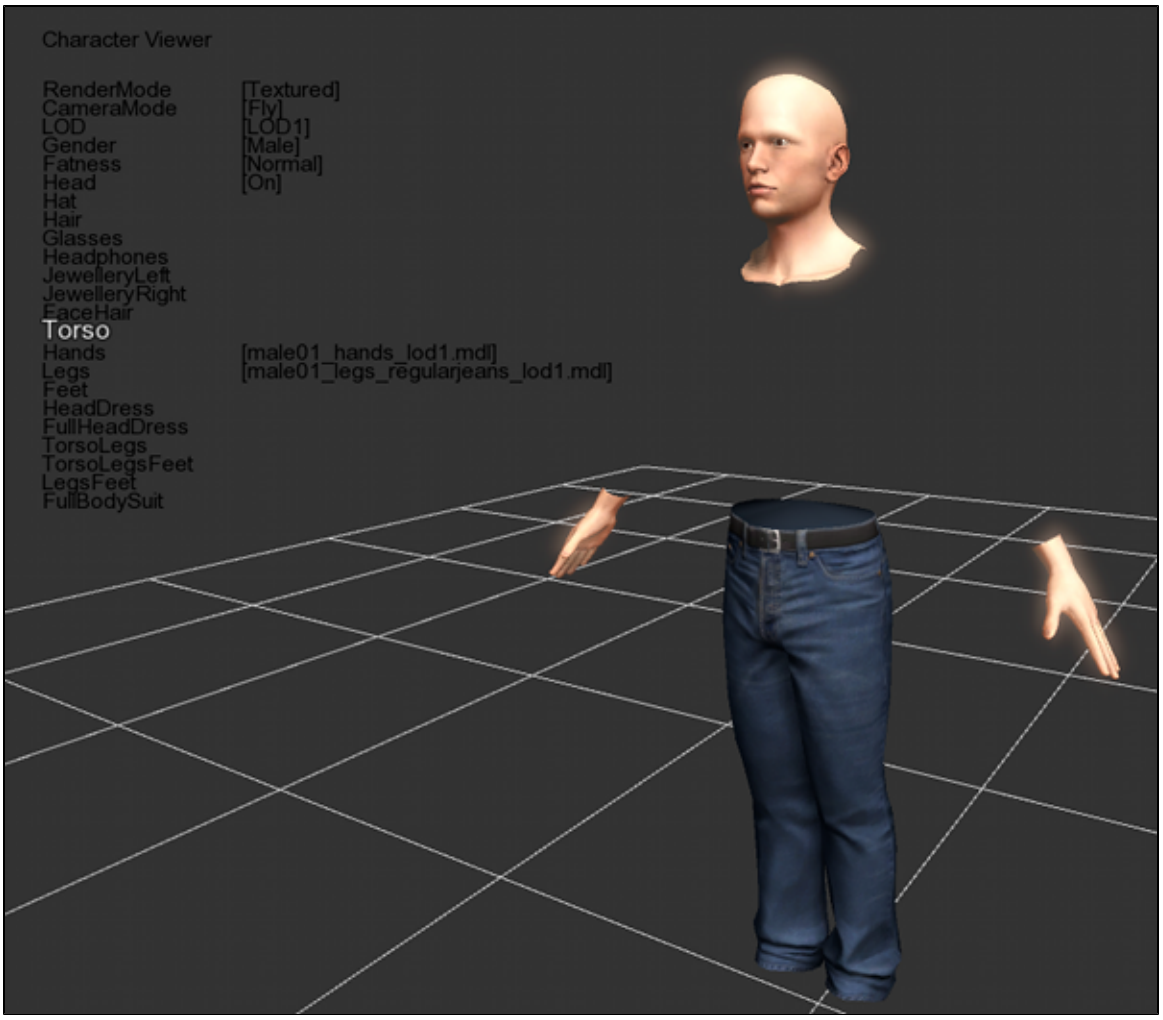
Unlike in PS Home, you can zoom the camera quite close to the avatar.



Character Viewer Functions

Default Character Viewer

- Change gender, fatness (blend shapes) and view all LODs (123)

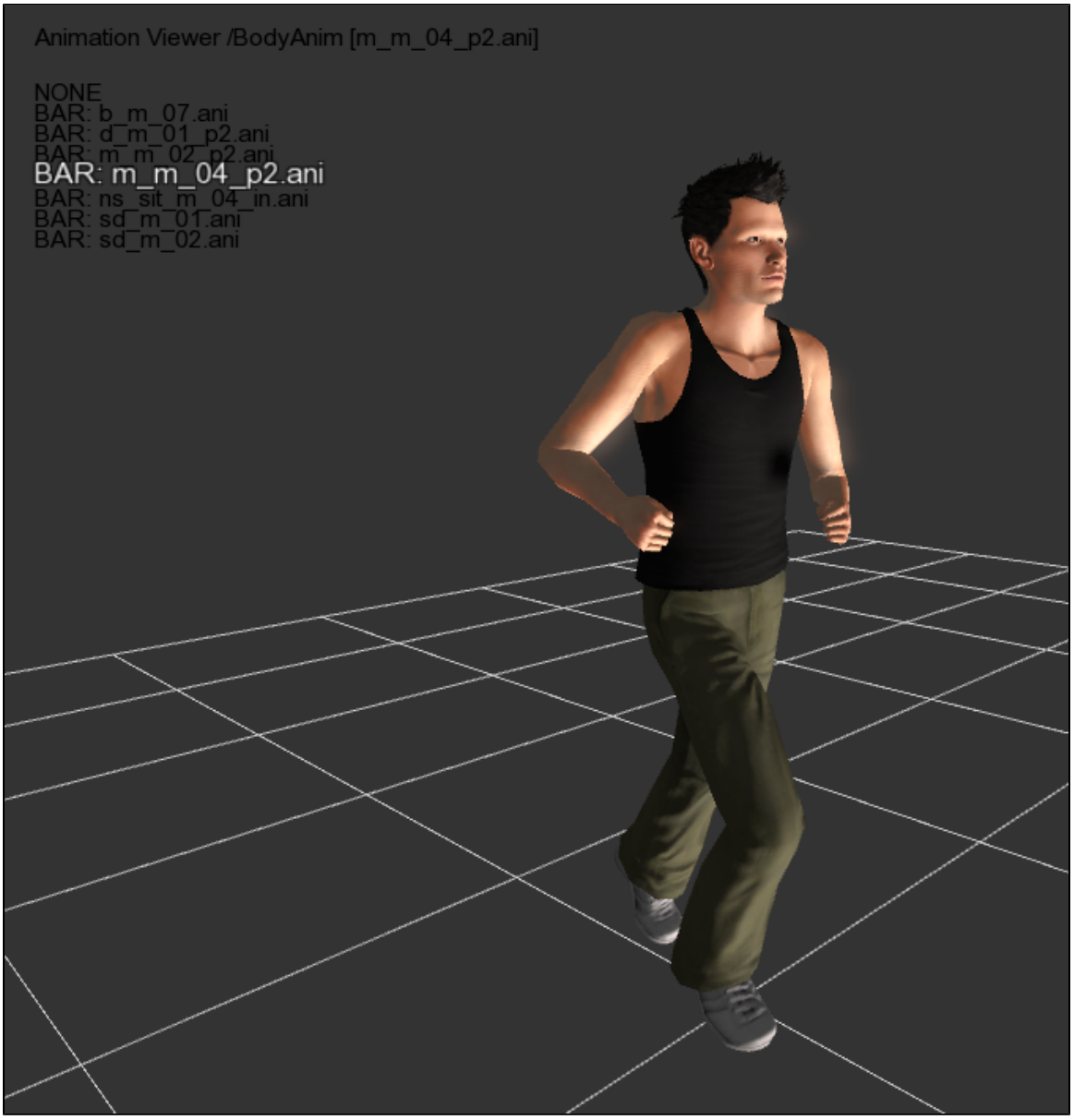


Menu Item	Options (default in **bold**)	Description
RenderMode	Wireframe **Textured (default)**	Renders the model in wireframe or with textures loaded.
Camera Mode	**Fly (default)** Lock To Face Lock At Head Lock At Torso Lock At Left Hand Lock At Right Hand Lock At Pelvis Lock At Feet	Keeps the default user controlled mode (fly) or locks camera view to focus on specific component (even when animating).
LOD	**LOD1 (default)** LOD2 LOD3	Displays components at the level of detail selected.
Gender	**Male (default)** Female	Changes the gender of the character

Fatness	Thin 1.0 Thin 0.8 Thin 0.6 Thin 0.4 Thin 0.2 Thin 0.2 **Normal (default)** Fat 0.2 Fat 0.4 Fat 0.6 Fat 0.8 Fat 1.0	Displays the components at the blend shape selected.
Hat	**None (default)**	Lets you view hat components you have developed.
Hair	**None(default)** male01_hair_additional_05_lod[123].mdl female01_hair_additional_01_lod[123].mdl	Loads example hair or lets you view hair components you have developed.
Glasses	**None (default)**	Lets you view glasses components you have developed.
Accessories	**None (default)**	Lets you view accessories components (such as masks) you have developed.
Jewelry	**None (default)**	Lets you view the jewelry components you have developed.
Face Hair	**None (default)**	Lets you view the facial hair components you have developed.
Torso	**None(default)** male01_torso_vesttop_lod[123].mdl female01_torso_tshirtbabydoll_lod[123].mdl	Loads example torsos or lets you view the torso components you have developed or select an example.
Hands	**None (default)** male01_hands_lod[123].mdl female01_hands_lod[123].mdl	Loads example hands or lets you view the hands/gloves components you have developed.
Legs	**None (default)** male01_legs_khakis_lod[123].mdl female01_legs_trouserscapris_lod[123].mdl	Loads example legs or lets you view the legs components you have developed.
Feet	**None(default)** male01_feet_trainersregular_lod[123].mdl female01_feet_shoesbar_lod[123].mdl	Loads example feet or lets you view the feet/shoes components you have developed.

Animation Viewer

- Animate (to check skin weight).
- Slow down, pause and reset animation.
- Check intersection and clipping (swap out different components to see how well they match with each other and how well they fit the avatar in different animated poses).



Menu Item	Options (default in **bold**)	Description
RenderMode	Wireframe **Textured (default)**	Renders the model in wireframe or with textures loaded.
Camera Mode	**Fly (default)** Lock To Face Lock At Head Lock At Torso Lock At Left Hand Lock At Right Hand Lock At Pelvis Lock At Feet	Keeps the default user controlled mode (fly) or locks camera view to focus on specific component (even when animating).
BodyAnim ResetTime	-	Resets the animation (to the start).

BodyAnim Speed	0.0 0.125 0.25 0.5 **1.0 (default)** 2.0 4.0	Sets the speed of the animation.
BodyAnim	**NONE(default)** B_[MF] 07.ani D [MF] 01_P2.ani M [MF] 02_P2.ani M [MF] 04_P2.ani NS_SIT [MF] 04_IN.ani SD [MF]01.ani SD [MF]_02.ani	Various animations that are useful for checking the skin weighting of components.

Screenshot renderer

Take screenshots with various options of the different views in the Character Viewer.

Menu Item	Options (default in **bold**)	Description
Camera Mode	**Fly (default)** Lock To Face Lock At Head Lock At Torso Lock At Left Hand Lock At Right Hand Lock At Pelvis Lock At Feet	Keeps the default user controlled mode (fly) or locks camera view to focus on specific component (even when animating).
Background Colour	**Grey (default)** Red Green Blue	Changes the background color of the viewer
Take Screenshot	1xSize 1xAA 1xSize 2xAA 2xSize 1xAA 2xSize 2xAA	Takes screenshots and allows for size adjustment (1 or 2 x size and 1 or 2 x AA - anti-aliasing). Screenshots will be stored in the build folder as .png files, e.g. myscreenshot.png

Lighting

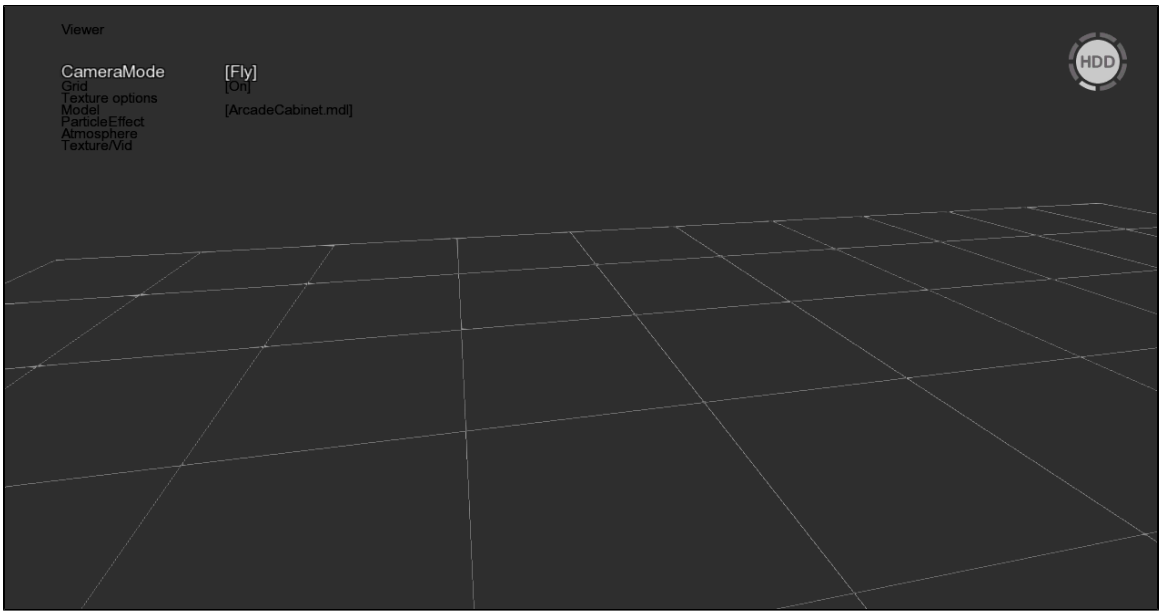
Move directions of lightsource and turn on/off glow and shadows.

Menu Item	Options (default in **bold**)	Description

Camera Mode	**Fly (default)** Lock To Face Lock At Head Lock At Torso Lock At Left Hand Lock At Right Hand Lock At Pelvis Lock At Feet	Keeps the default user controlled mode (fly) or locks camera view to focus on specific component (even when animating).
Render Glow	Off **On (default)**	Toggles glow of the character on/off
Render Shadows	Off **On (default)**	Toggles shadow rendering on/off
SunLight Direction [x,y,z]	-	Move the sunlight source with left stick of the pad.

HDK Browser

The HDK browser enables you to load and view content from your HDD:



Launching the HDK Browser

When launching the HomeDeveloper.self, enter the command `--hdkbrowser`.

Modes

The following table shows the different HDK Browser modes and options:

Browser Option	Description
CameraMode	Choose how fast the camera moves around the space. The options are slow (`SlowFly`), normal (`Fly`), and fast (`FastFly`). You can also choose `Lock to Model`, which locks the camera's perspective on the model.
Grid	Turns the grid on or off.

Texture Options	The options when displaying textures.
View Mode	Switch between alpha only, semi-transparent, color only.
Jpg/Png options	Toggles on and off the following jpg and png options: • Compressed: If the image size should be compressed. • Generate mipmaps: To generate mipmaps for the texture. • Limit image size: Ensures that the texture's width and height are no greater than 512 pixels (ideally used to show user-generated images placed on picture frames). This feature continually halves the image size until both height and width are below 512 pixels. • Make pow 2 wide: Ensures that the texture's width is a power of 2. The advantage of using a power of 2 wide texture is better memory usage. When an image is not a power of 2, the graphics card requires additional texture memory to pad the mipmap data. • Resize down: If you enable Make pow 2 wide, this option specifies whether to round the image size up to the nearest power of 2. When off, it rounds down to the nearest power of 2.
Model	Browse your build folder and choose a model (.mdl file) to view.
Particle Effect	Browse your build folder and choose a particle effect (.efx file) to view.
Atmosphere	Browse your build folder and choose a sky scene (.atmos file) to view.
Texture/Vid	Browse your build folder and choose a texture or video file to view.

Profiling Scenes

You can use a number of tools for profiling scenes:

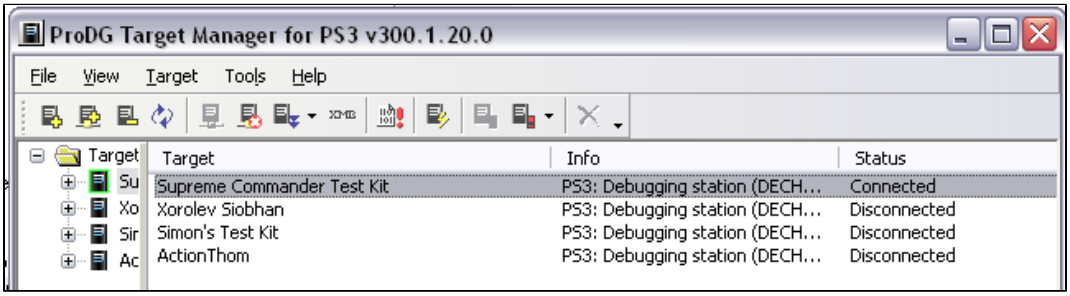
- The Profile GUI.
- The Dev Debug Console Commands, which include commands for showing information about frame rates and particle effects. See Frame Rate and Profiling Particles
- The PPU Statistics window.

For a list of tools that includes profiling other types of content, see Profiling in the Client.

To profile a scene, launch the scene in PS Home, through the Scene Editor. To be able to open a scene in the Scene Editor, you must first create the scene artwork in Maya, then export it. If the export is successful, it generates a .scproj file.

Launching your Scene

1. Open the Target Manager, and connect to your PS3 kit:



2. Open your .scproj file in the Scene Editor.
3. Click  to launch your scene in PS Home:



When you first enter PS Home from the Scene Editor, the screen displays certain critical statistics:



Critical Statistics

General Statistics

- **Lobby Population**: The number of users in the scene.
- **VRAM/HOST free**: Displays the amount of VRAM/HOST memory being used out of the total available.
- **Mesh Count**: Displays the number of meshes in the scene.
- **Visible Meshes**: Displays the number of visible meshes in the scene.

Mesh Count is different to the total Visible Meshes. For information on meshes and limits, see Models and Meshes.

Scene Statistics

- **MAIN**: Displays the amount of MAIN memory being used out of the total available.
- **HOST**: Displays the amount of HOST memory being used out of the total available.
- **VRAM**: Displays the amount of VRAM memory being used out of the total available.

Screen Content

- **Total VRAM**: Total VRAM being used by screens in the scene out of the total available.
- **Screen VRAM**: Total VRAM being used by image screens in the scene out of the total available for image screens.

PPU Statistics

The PPU Overview stats provide an overview of the overall performance of the scene and help you to identify areas that may be causing framerate issues.



These statistics display automatically when you open the console. You turn them on and off using the `EnablePPUOverviewStats` command.

The colored bars indicate the level of performance:

- green bar = within expected range
- orange/yellow bar = high
- red bar = significantly over expected rates

The PPU statistics are divided into the following, main content type categories:

Content Type	Description
Avatars	Update of the local and remote avatars and their animations. This is mostly out of your control, but if the bar goes red regularly it might indicate issues with the scene collision. The PPU measurement should be almost zero until the scene is populated.
LuaGames	The time taken to execute the Lua scripts for mini-games. If the bar is red regularly, look at optimizing your mini-game code and perhaps do less on the 'global'/non-joined update.
LuaScene	Time taken to execute the Lua scene script. If the bar is red regularly you might need to look at optimizing your scene script code. When a bar goes red, it is indicating that this area is using more processor time than one would expect in the normal configurations for PS Home scenes. In some cases this might not be relevant. For example, you may have developed a scene that makes extensive use of the scene script, but does not include mini-games. In this case it is reasonable for the scene-script to be more active.
System	The core client activities. This is mostly out of your control, but it can be affected by using certain Lua functions (such as async commands). This measurement also includes the Havok collision update. You might see this highlighted if a scene is doing a lot of Havok processing.
Lua GC	Garbage collection (includes all scripts, games and scene). This is normally where the most resource is used. If the bar is red, you must optimize your garbage collection in Lua. See Lua Garbage Collection and Managing Garbage Collection. There are also profiling tools and commands available specifically to help you with garbage collection. See Profiling in the Client and Profiling Lua Scripts.
Render	The main, final rendering process. As with Display, this measurement is affected by the number of items to draw in the scene. If the bar is red regularly, consider drawing fewer items, or optimizing/restricting your use of dynamic reflections, shadows and suchlike.

Screens	The time taken to render screens, which includes the time taken to execute the primitive list generated by calls from a renderer attached to a Lua screen, or an HSML layout. The most likely case where the execution of this list will be significant is on screens that generate a lot of primitives through the use of the text. Each character using Home's main font support is drawn using a large number of polygons, and a lot of text can have a significant impact on framerate. Using a font texture and quad draw calls would be a more optimal approach if feasible (It depends on localization and VRAM requirements). The decoding and playback of video on screens takes some processing time, but this is primarily on the PS3's SPUs. It should not appear as a significant factor in the screen profile time.

If the overview profile indicates a problem with your LuaGames, LuaScene or Lua GC, use the Object Profiler to obtain more detailed information on each separate object in the scene running Lua. See Profiling Lua Scripts.

This profiling provides only an approximation of the use. Do not use it for validation or Quality Assurance (QA) approval. In some cases all the stats may become inaccurate. For example, a lot of background thread or HDD activity (such as video or MP3 playback), can slow down all the main thread systems equally and generate spurious red bar results. To make sure your scene will pass QA, test it using different profiling tools and use the automatic validations to make sure that your scene complies with the restrictions and limits set for PS Home content. For more information about validations and limits, see Testing, Validating and Submitting Content.

Dev Debug Console Commands

You can profile your scenes using a number of profiling commands in the Profile GUI.

1. To open the Profile GUI using the controller, press SELECT, select DEV DEBUG and then select QA Profiler.
2. On your controller, press SELECT to open the PlayStation®Home Safe Screen and open the console:



You can use the following profiling commands:

Command	Description
`profileEnable`	Opens a GUI that displays a list of the components of the scene contributing to render and processing rate (measured in milliseconds, ms).
`enableMemoryStats`	Opens a GUI that displays the scene's available and consumed memory for certain properties (for example, HomeString, IO, Network, Video, Sound, Debug).
`enableNetStats`	Opens a GUI that displays the scene's Network Stats (for example, Net Free memory, LibNet Free memory). This also includes network stats for realtime games.
`enableScreenStats`	Opens a GUI that displays the Total video VRAM, total image VRAM and total MAIN memory used by screens the scene.
`enableVideoStats`	Opens a GUI that displays the memory stats for videos in your scene.
`ProfileGui.Enable`	Opens the Profile GUI, a sidebar display into which you can call various profiling tools (for example, cpu, info). This profiler can also be used for objects.

showfps	This Dev Debug command displays a frame rate GUI in the bottom right-hand corner of the screen. See Frame Rate.

Each of these commands takes one of two arguments: 0 for off, and 1 for on. For example:

- To turn on the NetStats, enter enableNetStats 1 into the console.
- To turn off Memory Stats, enter enableMemoryStats 0 into the console.

To obtain the overall GPU and CPU times as well as the scene memory and network statistics, use the following console commands:

- profileEnable 1
- enableMemoryStats 1
- enableNetStats 1

You can also show Lua rendering, Scene CPU, and frame rate.

The profiling tools measure as accurately as possible the resource usage of a scene or object. For many of the statistics that the profiling tools report, they also give the total resources available.

> ⚠ Note that the resource 'budget' is occasionally more generous than the memory limits set for scenes and objects. For example, in HDK v 1.2.3, the total limit of the memory resources for Scenes for the EnableNetStats command is 5 MB. However, the enforced main memory limit for Public Spaces for that version of the HDK is 4 MB, and for Personal Spaces it is 512 KB. Do not use the profiler's maximum limits as a validation tool against your scene's stats. Instead, check the maximum limits in Memory Limits.

Console Commands

profileEnable

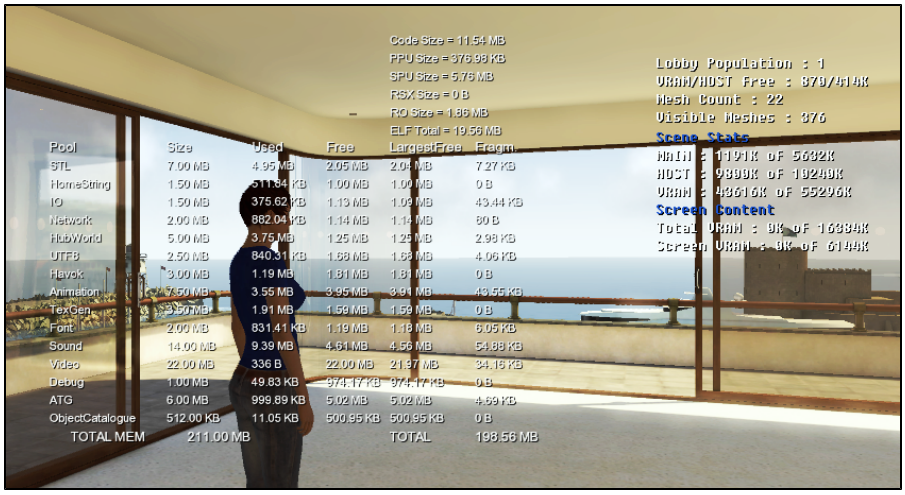


enableMemoryStats

enableMemoryStats has two modes:

- enableMemoryStats 1

  This mode displays all of the memory pools in a table.



The columns are:

- Pool: The memory pool (for example, Network, Screen, Animation).
- Size: The total size of the memory pool.
- Used: The amount of memory used in the pool.
- Free: The total amount free.
- Largest Free and Fragm.: Memory pools can be fragmented. These two columns show the largest fragment of free memory in the memory pool, and the remaining free memory fragment.

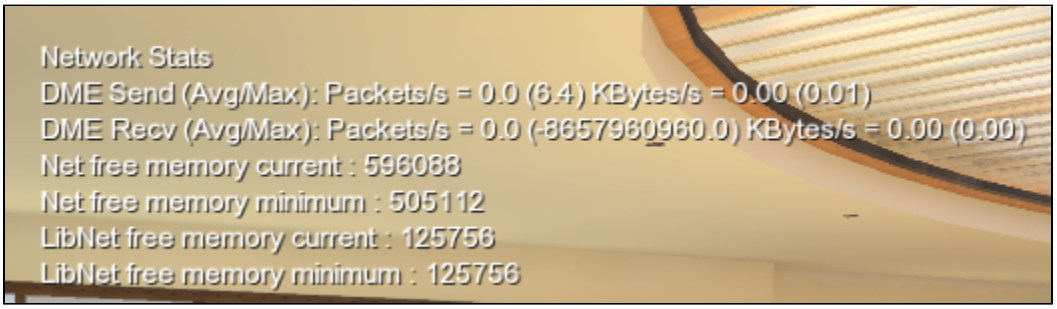
- `enableMemoryStats 2`

  This mode graphically shows the proportion of total memory used in each pool.

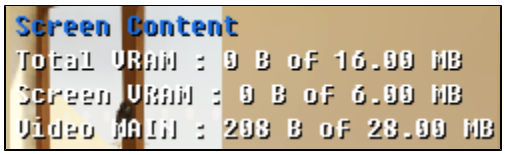


The green bar shows the amount out of the total available. The red bars, on the right, show the total amount of free fragmented memory.

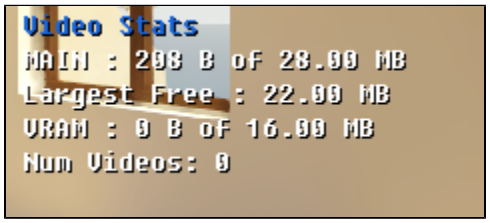
enableNetStats



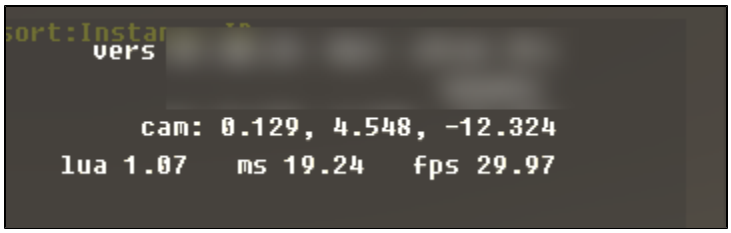
enableScreenStats



enableVideoStats



Lua Rendering, Scene CPU and Frame Rate



With the Console open, set the number of lines in the console to at least 8 lines (use the command `consolenumlines 8`). Information is displayed on the top right side of the screen, giving the current HDK version, camera co-ordinates, and processing speeds.

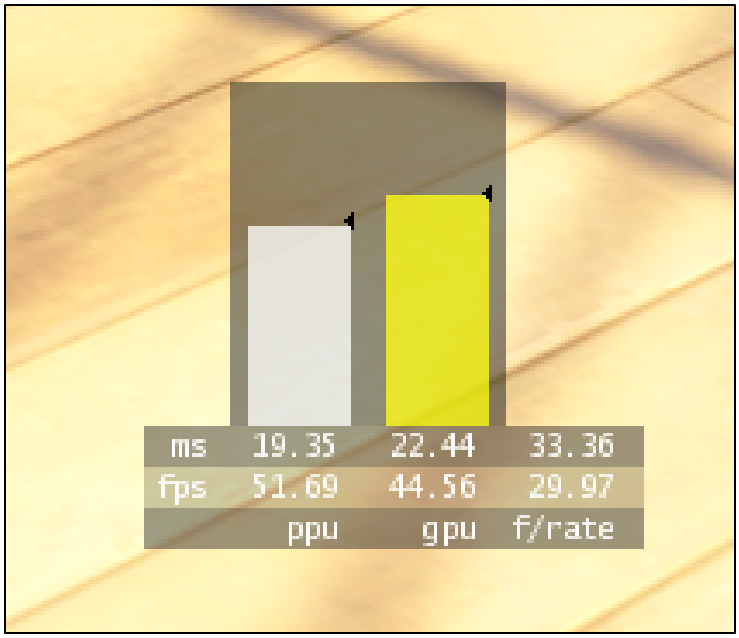
- `vers`: The current version of the HDK.
- `cam`: The camera co-ordinates.
- `lua`: The Lua rendering in ms.
- `ms`: The entire scene's CPU, including Mini-games, scripts and suchlike.
- `fps`: Frame rate of the scene.

See also PPU Statistics and Frame Rate.

Frame Rate

One of the most useful profiling tools is `showfps`. This Debug Console command displays a frame rate GUI in the bottom right corner of the screen. To display the frame rate GUI, type the command `showfps 1`.

The following image shows the frame rate GUI:



The GUI has two bars:

- one displaying the PPU rate
- one displaying the GPU rate

Below the bars is a table showing the numerical values in ms (milliseconds) and fps (frames per second).

For more information on rates of rendering and processing in fps and ms, see Recommended Rates for Rendering and Processing.

See also Profiling Scenes.

Profiling Particles

You can profile the particle effects in you scene using the command `showParticleInfo`. This command opens a table at the bottom of the screen showing all the .efx files in use in the scene.

The following table lists the columns in the particle information table:

Column	Description
name	The filename of the particle effect.
instances	The number of times (instances) that the .efx file is used in the scene.
vram [textures]	These values show the total VRAM memory used by a particle effect, and the number textures it uses.
main [inst]	To render particle effects requires MAIN memory overhead. This means that each particle effect requires MAIN memory to load the file, plus overhead MAIN memory to render it. The values in this column display the total MAIN memory, and the MAIN memory used by the .efx file.
host [inst]	These values show the total HOST memory used by all instances of a particle effect, and the HOST memory used by a single instance of a particle effect.
pushbuf [inst]	To render a particle effect, PS Home sends a series of commands to the GPU. These commands consume memory. The pushbuf value shows the total memory consumed by all instances of a particle effect in order to render. The [inst] value shows the total memory required for a single instance of a particle effect required.
pool	States which memory pool the particle effect is using.

The following sections summarize the Main, Host and Pushbuff values.

Main [inst]



main = overhead + [particle effect]

[inst] = amount of main memory the .efx file uses

overhead = main - [inst]

The overhead value is not shown in the GUI.

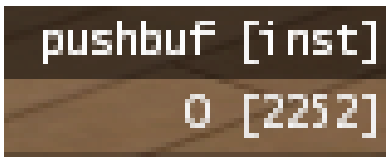
Host [inst]



host = [inst] * instances

[inst] = host memory used by a single instance of a particle effect.

Pushbuff [inst]



pushbuff = [inst] * instances

[inst] = Memory consumed by commands sent to the GPU to render the particle effect.

Profiling Objects

> For Japanese Live Page
> 最新の日本語版ページはこちら 「オブジェクトのプロファイリング」

Profiling determines how resource-intensive the scenes and objects are.

You can use a number of tools for profiling and optimization of objects:

- Clothing Dimensions Profiler: To check that character components fit within the recommended dimensions See Profiling Character Components.
- Object Editor Validator: To validate non-scripted objects. see Individual Non-Scripted Objects.
- Batch Validator: To validate all your Male Clothing, Female Clothing, or Furniture objects at once, instead of validating them individually. See Batch Validator.
- Object Profiler: Scripted objects, such as mini-games, realtime games, and arcade games, must be profiled at runtime using the Object Profiler. For objects with Lua scripted components you can also use the Profile GUI. See Profiling Lua Scripts .

Objects that have a Network component must be profiled both on a local machine and remotely. See Objects with Network Components.

All objects should adhere to the recommended Dimensions.

Maya does not perform automatic validations for Companion objects. You must check all requirements manually. Companion objects use network replication, so they must be profiled by remote users, not by a local user. See Profiling Companion Objects and Objects with Network Components.

Individual Non-Scripted Objects

For Japanese Live Page
最新の日本語版ページはこちら 「スクリプト無しオブジェクトの個別検証」

Use this validation process for furniture, character components, and character components with Script IDs.

Scripted objects, such as realtime and mini-games, cannot be validated by the tools. You must profile them using the Object Profiler. For more information on profiling scripted objects, see Profiling Lua Scripts.

The validation process checks your object in PS Home in offline mode, and generates:

- An XML report
- .jpeg files:
    - Furniture Object Images: a .jpeg file is generated for the top, side and front views.
    - Character Component Images: a .jpeg file is generated for each LOD.
- A .csv file for each object located in your <HDK_INSTALL_DIR>\build\batchvalidator folder.

The results of validating objects individually goes to the same folder as the results of validating multiple objects using the Batch Validator.

Use the Profile GUI to check the memory usage and stats on furniture.

You can check memory usage and statistics on character components only by validating them and then checking the .xml report. To preview your character components, see Character Viewer.

Running the Validator

To run the validator:

1. Open the Object Editor.
2. Select the object that you want to validate.
3. Select Object > Validate on PS3:



4. When the validation is finished, you can open up the Debug Console.

The results are output to either your <HDK_INSTALL_DIR>\build\batchvalidator folder or <USB_ROOT_DIR>. Open the object's XML to view them. The XML is described in detail in Batch Validator Results.

Profiling Character Components

For Japanese Live Page
最新の日本語版ページはこちら 「キャラクタコンポーネントのプロファイリング」

Profiling Clothing

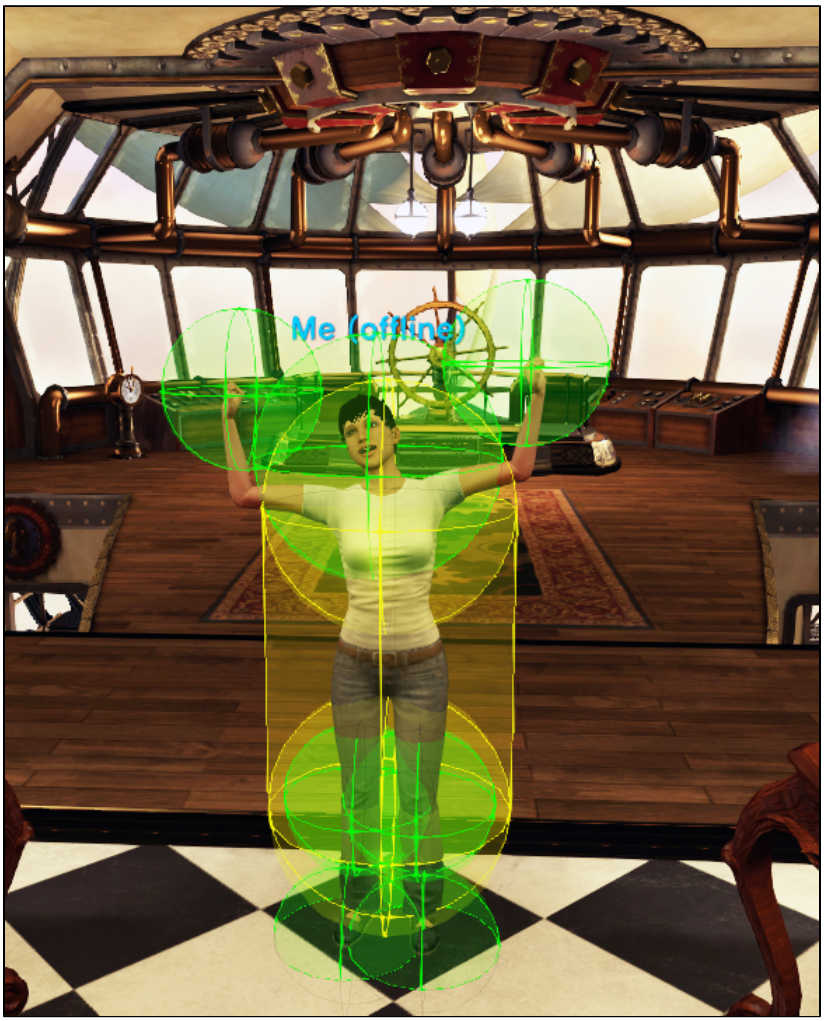
Profiling determines how resource-intensive scenes and objects are. You can use the profiling tools available to profile and optimize scenes and objects.

There are various ways to profile scenes and objects:

- The Object Profiler space profiles memory and mesh counts for furniture and clothing items.
- The Character Viewer lets you view character components on avatars, animate them, select different LODs, and other options.

The Clothing Dimensions Profiler

The Clothing Dimensions Profiler is a tool in the Home .self. You access it by calling the command `debugClothingLimits`. When the profiler opens, it shows green spheres around each individual part of the avatar that can have a character component. You can have spheres around the hands, head, feet, and legs:



These spheres indicate the maximum recommended size for each type of individual character component. It also shows yellow spheres around the avatar's body, which indicate the avatar's collision.

> The spheres do not include composites (such as legs+feet, or FullBodySuit). When profiling composites, toggle the appropriate spheres to check that the entire composite fits within the recommendations for each component.

You use the Clothing Dimensions Profiler to gauge whether character components fit within the recommended dimensions. The recommended dimensions make sure that your components fit within the minimum scene dimensions. If a component does not fit within the recommendations, it will stick outside of the sphere. If your components stay within the spheres, they should not clip through scene geometry.

See also:

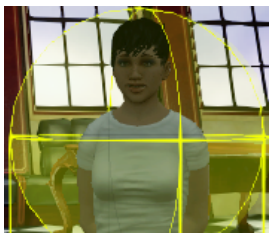
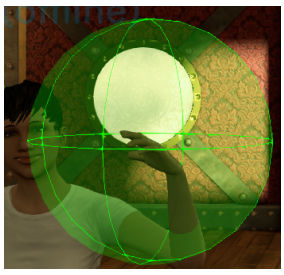
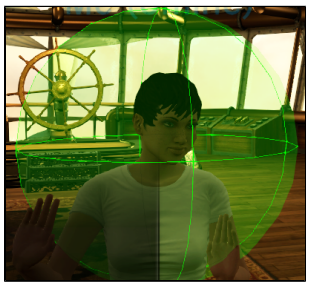
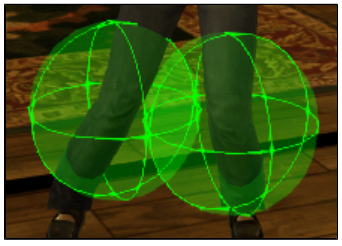
- Dimensions for a full listing of recommended dimensions (including for avatars, clothing, companion objects, and other items that you place in an environment).
- Batch Validator Clothing Limits for information on validating clothing dimensions.
- HDK Tools Validations for information on memory limitations.

Profiling Dimensions Using the Clothing Dimensions Profiler

To profile the dimensions of one or more character components:

1. Launch the .self.
2. Equip the avatar with the character components that you want to profile.
3. Open the Debug Console.
4. When the Clothing Dimensions Profiler is open, check the required components by toggling on/off each sphere around each component, and the collision spheres.

   The following table shows the Dev Debug console commands you can use:

Command	On (1, true)	Off (0, false)
debugClothingLimitsCollision Radius: 0.40 m Height: 1.8 m For reference: Clothing has no collision, so this capsule indicates the boundaries that are used to calculate collision between the avatar and the environment.		
debugClothingLimitsFoot Sphere Radius: 25 cm		
debugClothingLimitsHand Sphere Radius: 25 cm		
debugClothingLimitsHead Sphere Radius: 40 cm		
debugClothingLimitsLeg Sphere Radius: 25 cm		

The following debugClothingLimits commands do not work:

- debugClothingLimitsArm
- debugClothingLimitsHips
- debugClothingLimitsSpine

Profiling Companion Objects

For Japanese Live Page
最新の日本語版ページはこちら 「コンパニオンオブジェクトのプロファイリング」

Profiling of portables - Companion Objects, Animation Packs and Locomotion Objects - works in the same way - follow the instructions for all three types of portable.

Maya does not perform automatic validations for companion objects. You must check all requirements manually. Companion objects use network replication, so they must be profiled by remote users, not by a local user. See Objects with Network Components.

The following table lists the manual validations that are required or recommended for companion objects:

Companion Objects – Manual Validation (Requirements)	Values	Required	Recommend	Check Method
VRAM limit (remote user profiling)	256 KB	x		Profiler
Host memory limit (remote user profiling)	128 KB	x		Profiler
Main memory limit (remote user profiling)	128 KB	x		Profiler
The character limits for names are the same as the limit for object display names. See HDK Tools Validations.		x		Object Editor Validation
Runtime mesh limit	3	x		Profiler
Companion Objects – Manual Validation (Recommendations)				
Triangle limits - recommended maximum	4,000		x	Warning
There is no set height limit, but a companion object 'thinks' that it has collision in a radius of 0.4m from the ground in all directions and calculates how to follow users based on that height/width. If it is any taller, the companion object has unwanted behavior when following a user. Limit a companion object to a 0.4m radius sphere from the ground. If it is any wider, it clips when moving round corners and intricate geometry.	Height: 0.4 m Width: 0.4 m		x	Profiler
CPU time	Maximum 4ms/f Recommended 2 ms/f		x	
Collision	50 KB		x	

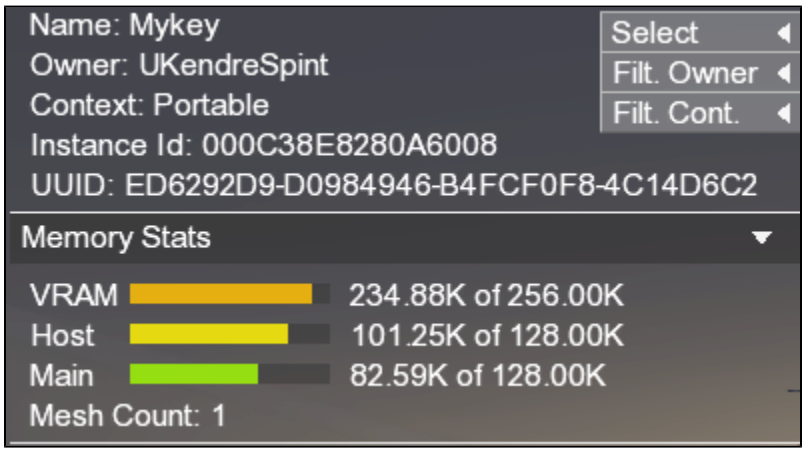
Always test a companion object around obstacles to make sure that it navigates around collision elegantly.

Comparison of Remote and Local Machine Memory Usage

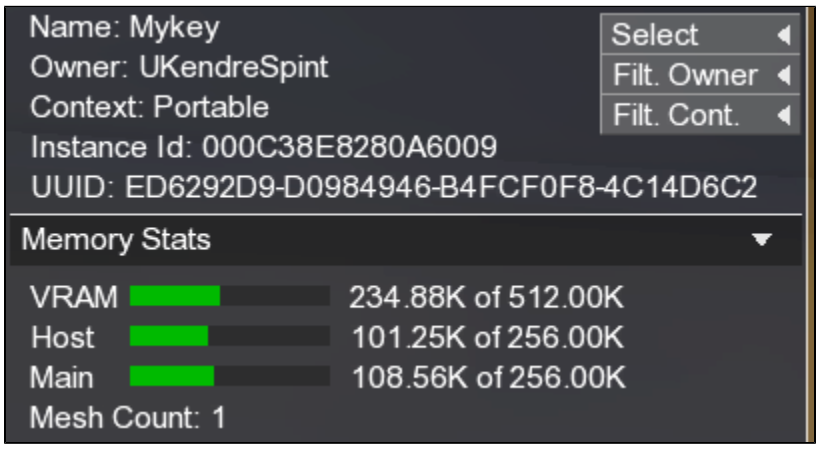
The values in the table show the maximum limits that a remote user should see when they are in the same scene as a local user with an activated companion object. When viewing the profiling tools on a local machine, a companion object's memory limits are double those in the table. These limits are larger on local machines to allow the companion object to load without issue while developing it.

The following images show the differences in memory use and memory limits between a remote machine and a local machine.

Remote Machine Memory



Local Machine Memory



> ⚠ When testing locally, keep in mind that the actual limits you must adhere to are those in the remote machine values shown in the table. These are the limits that Quality Assurance (QA) test against. To profile memory usage for companion objects correctly, you must run the two largest animations to yield the worst case scenario under which the companion object will operate.

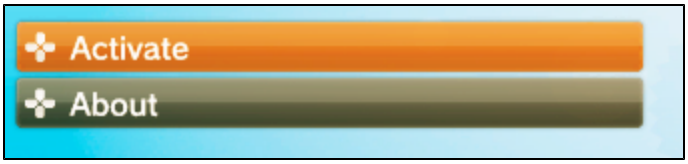
Objects with Network Components

> For Japanese Live Page
> 最新の日本語版ページはこちら 「Networkコンポーネントを持つオブジェクト」

Any object that has a Network component must be profiled both on a local machine and remotely to see that network replication works as intended, and that the object does not go beyond the performance (frame rate and memory) limits on a remote user's machine.

To test objects remotely:

1. Follow the steps in Testing Content in Online Mode to upload your object to the Content Delivery System (CDS), and to download the hubstartup.txt file needed to go online.
2. Launch the Home self on at least two test kits or dev kits with the hubstartup.txt file downloaded from the CDS.
3. Send all users to the same space.
4. On the local machine, press Select > Dev Debug > Object Catalogue Browser > Catalogue.
5. Find the object that you are testing, and add it to your inventory.
6. Activate, launch or place the object:

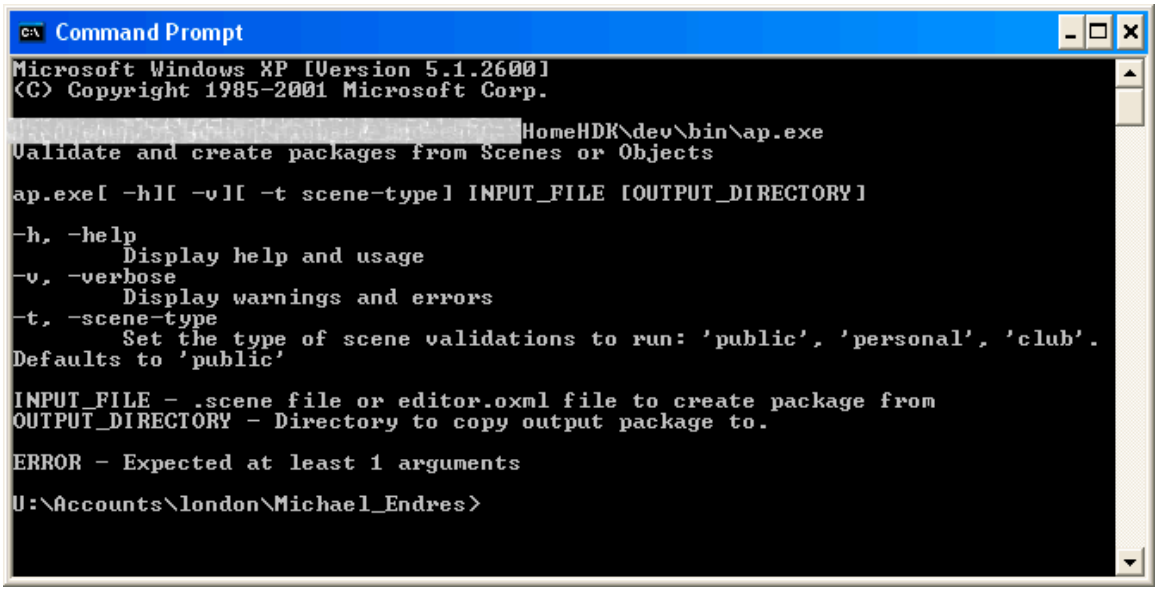


7. On the remote users' machines, open the profiling tools (the Profile GUI, Object Profiler, etc) to see the effects on of the object on the remote users' machines.

Command Line Batch Packaging

The HDK has a command-line tool for batch packaging assets. The tool is located in <HDK_INSTALL_DIR>\dev\bin.

Running the command-line tool displays instructions for its use:



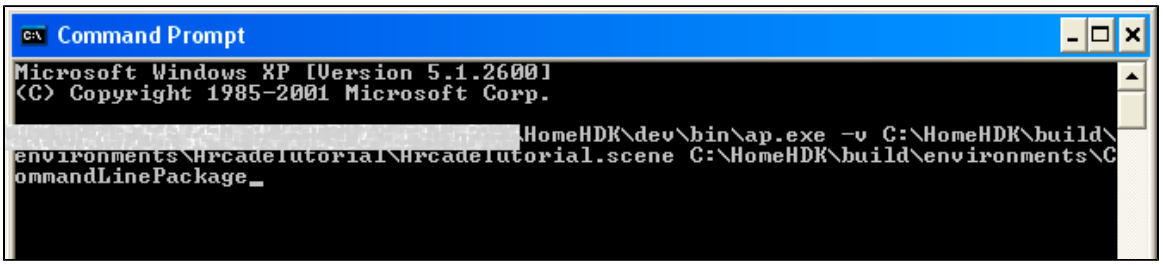
Validate and create packages from scenes or objects:

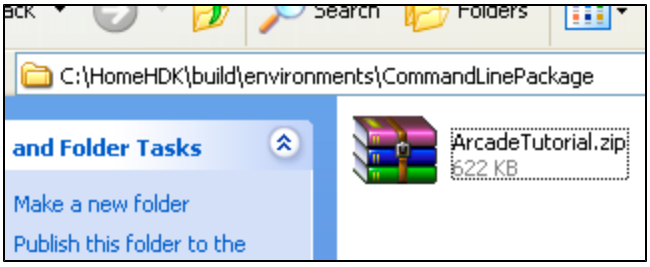
```
ap.exe [ -h] [ -v] [ -t scene-type] INPUT_FILE [OUTPUT_DIRECTORY]
```

- **-h**, **-help**: Display help and usage.
- **-v**, **-verbose**: Display warnings and errors.
- **-t**, **-scene-type**: Set the type of scene validations to run: public, personal, club. Defaults to public.
- **INPUT_FILE**: .scproj file, .scene file or editor.oxml file to create package from.
- **OUTPUT_DIRECTORY**: Directory to copy output package to.

For example, the following example packages the ArcadeTutorial scene, displaying warnings and errors, and putting the zipped package into the folder C:\HomeHDK\build\CommandLinePackage.

```
ap.exe -v -t public C:\HomeHDK\build\environments\ArcadeTutorial\ArcadeTutorial.scene
C:\HomeHDK\build\CommandLinePackage
```





Batch Validator

Use the Batch Validator to validate all your Male Clothing, Female Clothing, or Furniture objects at once, instead of validating individually. SCE QA uses the Batch Validator to test your objects. Make sure you use it before submitting your content for publication through the Content Delivery System (CDS).

Scripted objects, such as realtime and mini-games, cannot be validated by the tools. You must profile them using the Object Profiler. For information on how to profile scripted objects, see Profiling Lua Scripts.

> There is also a command-line tool for batch packaging assets. See Command Line Batch Packaging.

You can run the Batch Validator in online or offline mode:

- If you launch PS Home online and then launch the Batch Validator, it automatically operates in online mode.

  The content that will be validated comes from your sandbox on CDS.

- If you launch PS Home offline and then launch the Batch Validator, it runs in offline mode.

  The content that will be validated comes from the local PC in the folder <HDK_INSTALL_DIR>\build\objects. It will use your offline object catalogue in the same way as PS Home does when running offline.

To load content from a different server, specify the server by using the contentserveroverride command in the hubstartup.txt, for example:



For more information about the hubstartup.txt, see Testing Content in Online Mode.

Setting Up Online Mode

> If you are running Batch Validator offline, or if you are already set up for a specific environment, skip this section and go to Displaying the Menu.

Before you run the Batch Validator in online mode, you must:

1. Upload content to your developer sandbox (see Upload Content and Prepare to Execute Commands below).
2. Execute the necessary console commands (see Execute Console Commands below).
3. Sign into PSN.

Upload Content and Prepare to Execute Commands

For a full explanation of launching in online mode and entering console commands, see The Home Client.

When you have published your content to your developer sandbox using the CDS, you can view it using the Home Developer PKG. To do this, prepare your system by installing the Home Developer PKG on your system using the Install Package Files option on the XMB.

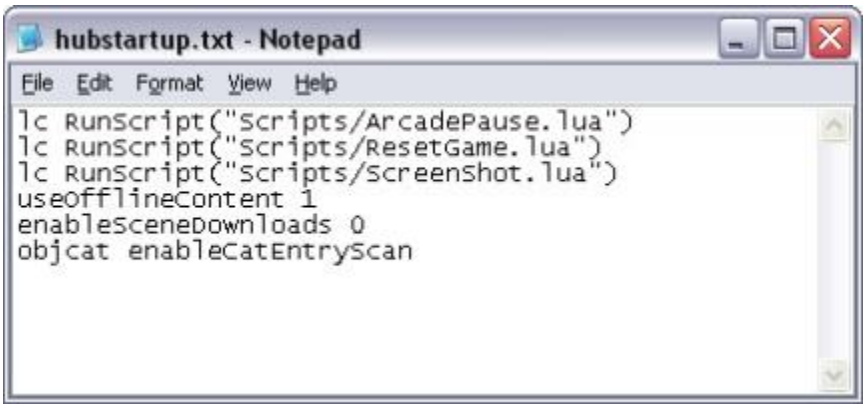
Then log in to the CDS and retrieve your content server override information from the My Details page by clicking the Download hubstartup.txt button. This file contains the console command required to instruct the client to connect to your sandbox. You must enter this command before you connect to PS Home. You can enter it either using a hubstartup.txt file, or on the title screen using the Target Manager.

Execute Console Commands

You can run the hubstartup.txt in the following ways:

- Execute Console Commands from a USB Stick: Insert a USB key loaded with a hubstartup.txt file containing your content server override command in your PS3.
- Execute Console Commands at Boot Time: When PS Home boots up, it loads a text file: <HDK_INSTALL_DIR>\build\Scripts\hubstartup.txt.

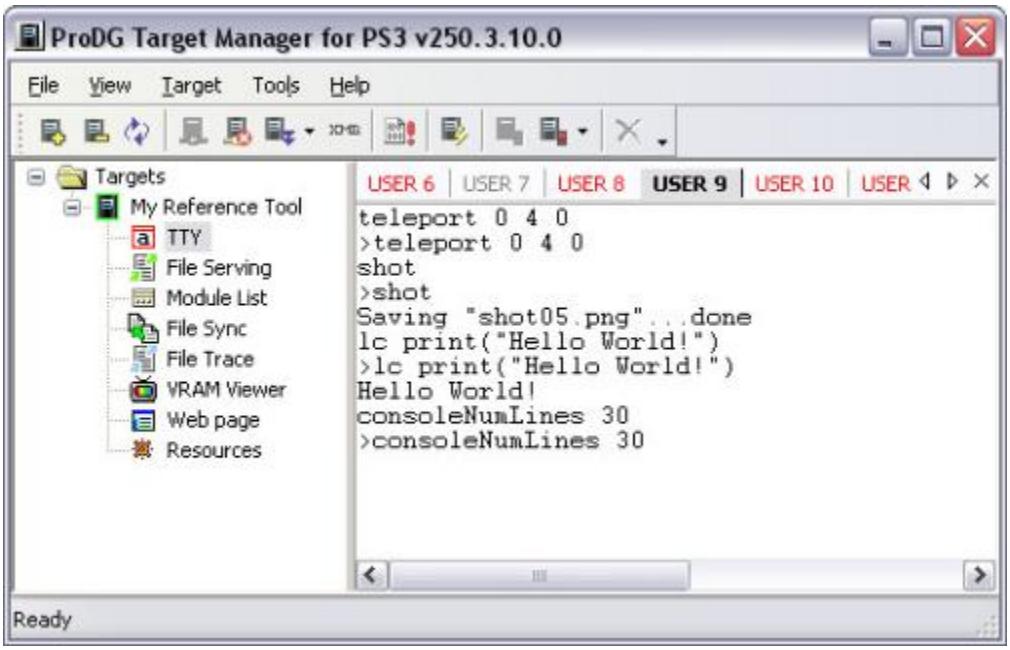
  This file can run commands that affect how PS Home runs. An example hubstartup.txt is included with the HDK which contains the following:



- Executing Console Commands from Target Manager:

  1. From the Targets list in the Target Manager, select the PS3 target you are using.
  2. Expand the item list below that target and select TTY.
  3. In the TTY panel, right-click the USER 9 tab and select Properties. The TTY Channel Properties dialog is displayed.
  4. Select the Echo TTY input to screen option and click OK.

     You can now enter commands directly into the USER 9 tab, as shown below:



> ⚠ The auto-complete function, which you initiate by pressing the TAB key on your keyboard, does not work when using the Target Manager to issue commands.

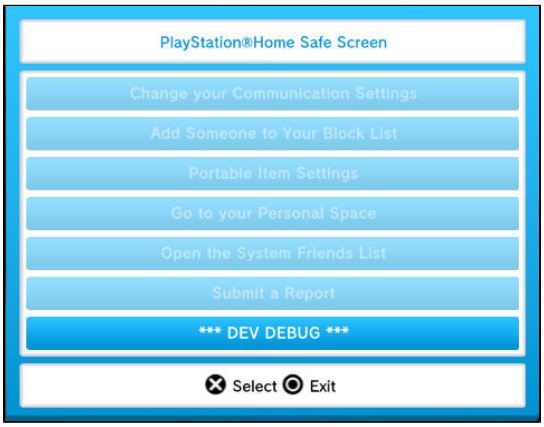
Displaying the Menu

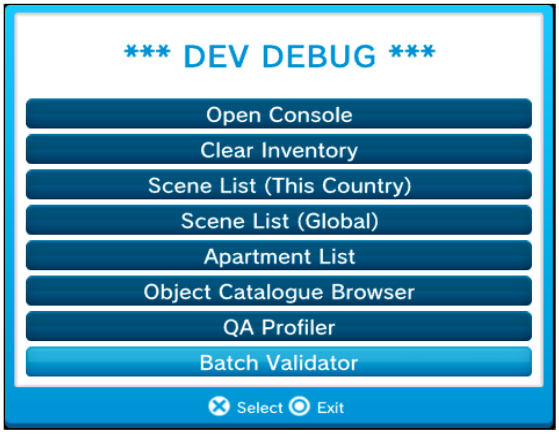
In PS Home, how you display the Batch Validator menu depends on whether you are running development or release mode:

- Press [SELECT] on the controller to open the debug console and type in the Batch Validator command.

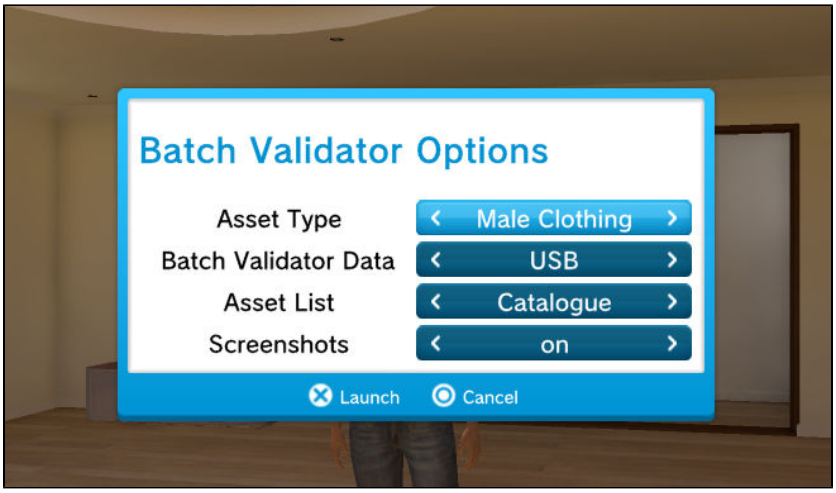
  Or

- Press [SELECT] on the controller to open the Safe Screen and select DEV DEBUG > Batch Validator.





The Batch Validator menu opens:

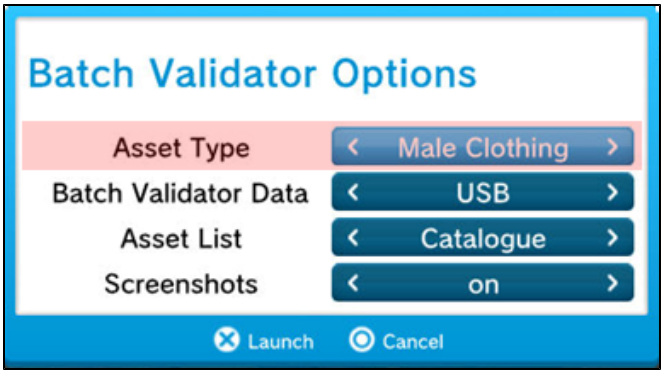


Batch Validator Menu Options

The Batch Validator menu has four options:

- Asset Type: Select which type of content you want to batch validate.
- Batch Validator Data: Select where you want the Batch Validator to save the data.
- Asset List: Select from where you want the Batch Validator to get the objects to validate.
- Screenshots: Select whether you want the batch validator to create screenshots.

Asset Type

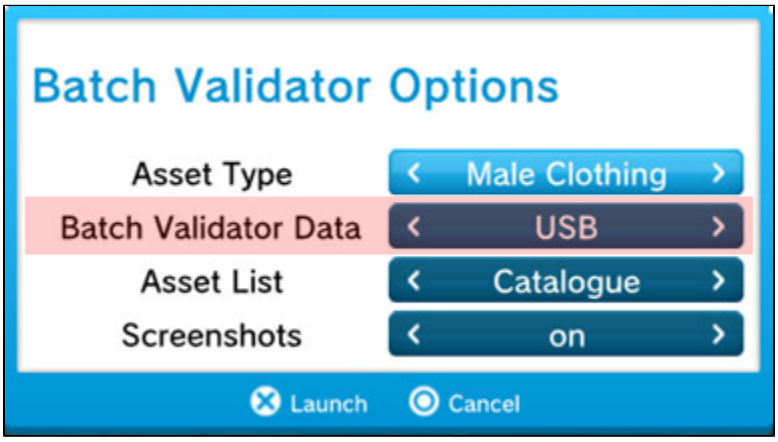


You can validate three types of content (asset types) using the Batch Validator. Each has its own separate folder where the results are stored:

Asset Type	Results Folder Location
Female Clothing	<HDK_INSTALL_DIR>\build\batchvalidator\FemaleClothing
Male Clothing	<HDK_INSTALL_DIR>\build\batchvalidator\MaleClothing
Furniture	<HDK_INSTALL_DIR>\build\batchvalidator\Furniture

> ⚠ If you need to test more than one type of asset (for example, Female and Male Clothing) you must validate each asset type. You must reset the PS3 between validations.

Batch Validator Data



Use this option to choose in which location you want to load and save Batch Validator files.

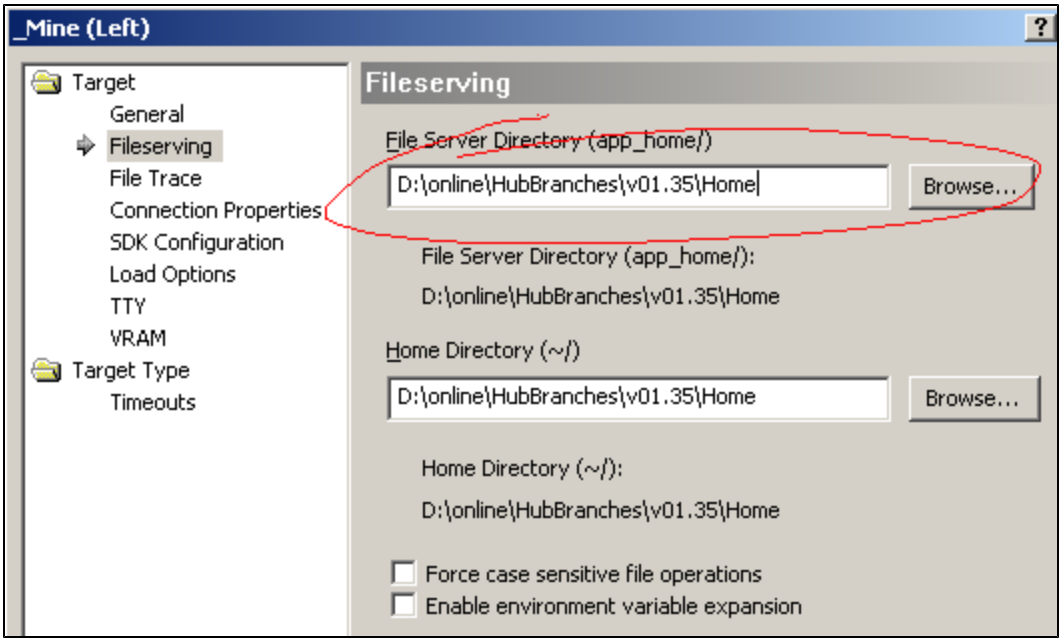
The Batch Validator can create a number of files, so you can choose whether you want to use a USB stick (inserted into the PlayStation®3) or whether to save to your PC. To save files to your PC, you must connect to your PlayStation®3 through the Target Manager so that you can save directly to the File Server Directory.

Batch Validator Data Options

You can choose between USB and PC:

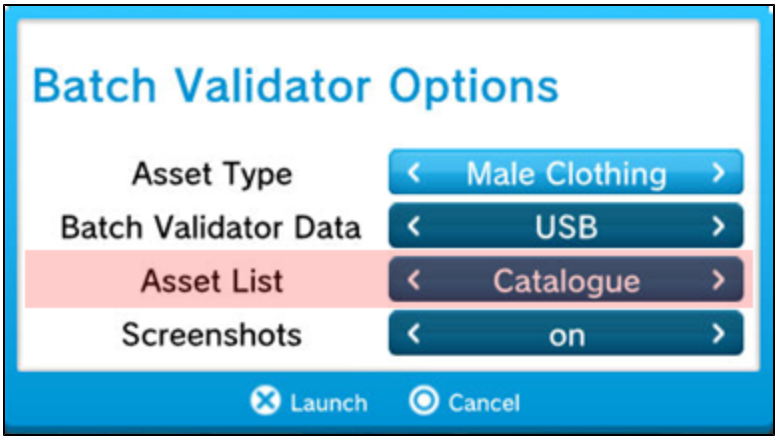
- USB: The data is stored in the folder <USB_ROOT_DIR>:\batchvalidator.
- PC: You must have a Target Manager attached to your PlayStation®3 and the File Server Directory must be set. It then loads and saves data from the Batch Validator folder relative to your File Server Directory on normal builds.

To set the File Server Directory in Target Manager, connect to your PlayStation®3 through the Target Manager, then select Properties > Fileserving.



To check which folder path it is set to, look in the main panel of Target Manager under the File Server Dir column. To set a different path on your PC, use the command `batchValidatorAppPath` to point somewhere else. The default is `app:BatchValidator/`.

Asset List



Use this option to choose where the batch validator gets its input selection from. The options are:

- Catalogue: If you are online, the batch validator goes through the entire catalog on the server. If you are offline, it goes through the locally generated catalog that matches the criteria in the Asset Type.
- File: This option requires a text file named ObjectList.txt in the corresponding folder for the particular asset type. For example, `BatchValidator/MaleClothing/ObjectList.txt`.

The format of the file is an object ID per line. For example:

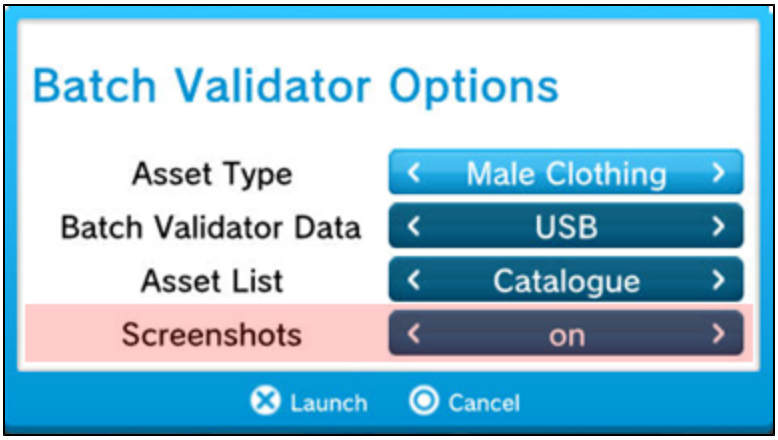
```
68B1A6D2-F8FA4FF8-9102DB85-4B83C9DD

6D308E78-69A24BD1-B1FEB431-9559C662

6D34A1EE-D4BD414D-A74C1EEA-FB61C3E6

6D51D6D8-8691424B-86003B73-5A4483C2
```

Screenshots



You can switch screenshot generation on or off. If it is switched off, the Batch Validator runs quicker. If it is switched on, each object generates three screenshots. For clothing objects, it generates is one screenshot for each LOD. Furniture objects have a screenshot of their front, side and top views.

Launching and Running the Batch Validator

Launching the Batch Validator

When you launch the Batch Validator, it prints its options to the TTY. Example output looks like this:

```
Batch Validator Options

-----------------------

Context: furniture

Path: app:BatchValidator/Furniture/

Online: true

Launched: Menu

Screenshots: false

Asset List: Catalogue Search

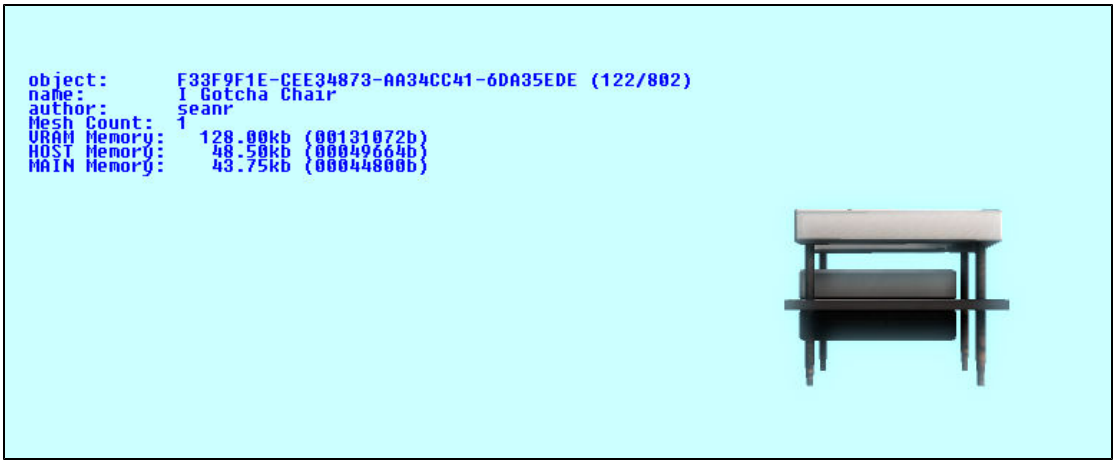
************* Entering Object Mode **************
```

If you are running a catalogue search, it also prints out all the objects it will test, as a file list to the TTY. This list enables you to make an objectlist.txt from the TTY. This is useful if you want to interrupt your validation and start up again from a specific point onwards.

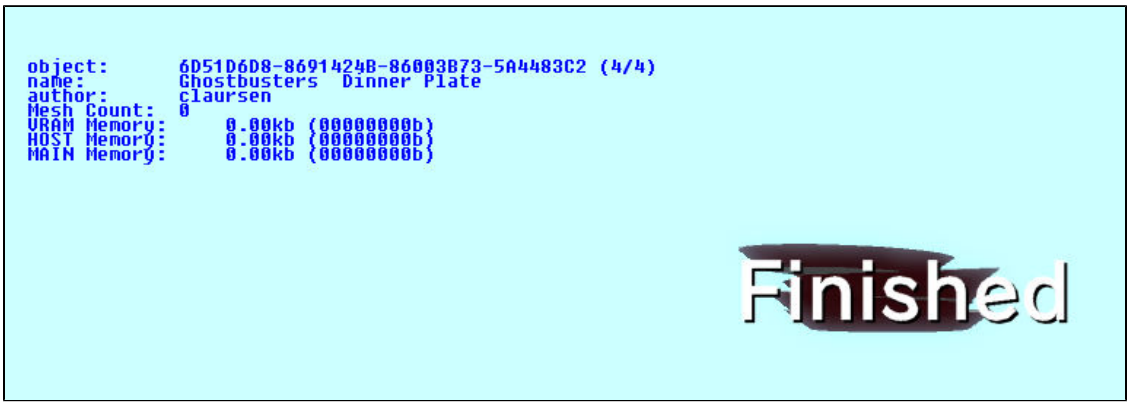
For example, if the batch validation crashes unexpectedly on a particular object, you can copy-and-paste the rest of the objects after the one that crashed from the file list and run it. Alternatively, you can just remove the object causing the crash from the list and rerun it.

Running the Batch Validator

When you launch the Batch Validator, it loads into a new scene. It cycles through the objects you pointed it at, one by one, displaying them to screen. The following image shows the type of information you see:



When it has finished, it displays 'Finished' on the screen, as shown below:



The results are output to your <HDK_INSTALL_DIR>\build\batchvalidator folder, or <USB_ROOT_DIR>. Open the object's XML to view them.

The XML is described in detail in Individual Non-Scripted Object Validation and Batch Validator Results.

Batch Validator Crash Reports

If the Batch Validator crashes, you can view the results in the spreadsheet before the crash occurred. Open the .xml file in Notepad or a similar text editor, and add the text below to the end of the file:

```
</Table>

        <Sorting xmlns="urn:schemas-microsoft-com:office:excel">

            <Sort>Object Id</Sort>

        </Sorting>

    </Worksheet>

</Workbook>
```

Batch Validator Metrics

You use the Batch Validator to validate both male and female clothing items. The Batch Validator iterates through clothing items for a given gender. It produces individual reports, and a single combined table of resource allocation compared to resource limits.

The following table summarizes the Batch Validator metrics:

Metric Name	Description	Notes

Object Id	The GUID for the object being profiled.	
Name	The name of the object.	
Author	The author of the object.	Is retrieved from the object xml.
rig	The rig this clothing items is for.	Presented as GUID Male = 00000000-00000000-00000010-00000000 Female = 00000000-00000000-00000010-00000001.
HDK Version	The HDK version that the object was exported with.	
type	The clothing type of the item. This determines what the limits for the clothing object are. The possible clothing types are: Hair, Hat, Hands, Tors, Legs, Feet, JewelLeftEar, JewelRightEar, Glasses, HeadPhones, FacialHair, LegsAndFeet, TorsoAndLegs, TorsoLegsAndFeet, FullBodySuit, FullHeadDress, HeadDress, JewelBothEars.	Tors is abbreviation for Torso. FacialHair is only applicable to the male rig.
Failure	A failure rating, with 0 being no failure, and increasing with magnitude depending on how close to the resource limits the object comes.	If resource allocation is close to the limit, this increases the failure rating slightly. If the allocation is over the limit for a resource, then the failure rating is increased dramatically. If the object fails to load then the failure rating is bumped up accordingly.
Loaded	Indicates whether the object managed to load successfully.	
LOD 1 VRAM	The video memory allocation in LOD 1.	Prior to HDK 1.60, results were shown for all VRAM allocations at all LODs, but now only the LOD 1 VRAM allocation/limits are shown, as the lower LODs cannot be modified directly, and are a result of the mipmaps used in the textures.
LOD 1 VRAM Limit	The maximum video memory available in LOD 1 for this clothing type.	
Enough mipmaps	Indicates whether or not the textures used have enough mipmaps.	The mipmaps check is necessary because if there are no mipmaps, then the same texture that is used for the high LOD, will be used for the other LODs, and consequently the lower LODs will run out of VRAM. This metric has been added to replace the VRAM metrics at lower LODs that have been removed.
LOD 1 Main	The main memory allocation in LOD 1 (Highest LOD).	
LOD 1 Main Limit	The maximum main memory available in LOD 1 for this clothing type.	
LOD 1 Host	The host memory allocation in LOD 1.	
LOD 1 Host Limit	The maximum host memory available in LOD 1 for this clothing type.	
LOD 1 Mesh count	The number of meshes used in LOD 1.	
LOD 1 Mesh Limit	The number of meshes available in LOD 1.	
LOD 2 Main	The main memory allocation in LOD 2 (Medium LOD).	
LOD 2 Main Limit	The maximum main memory available in LOD 2 for this clothing type.	
LOD 2 Host	The host memory allocation in LOD 2.	
LOD 2 Host Limit	The maximum host memory available in LOD 2 for this clothing type.	
LOD 2 Mesh count	The number of meshes used in LOD 2.	
LOD 2 Mesh Limit	The number of meshes available in LOD 2.	
LOD 3 Main	The main memory allocation in LOD 3 (Low LOD).	

Independent Main	The main memory allocations that are independent of LOD. This deals with allocations made at the object level (i.e. the object definition). This will also include animation allocation data if animations are part of the clothing object (currently only valid for FullBodySuit clothing items).	The independent main allocations are made within the Low LOD main budget, as the Low LOD is always loaded for a given avatar (so the memory is always available).
Total LOD 3 main	The aggregate of LOD 3 Main and independent main allocations.	The LOD 3 main and independent main figures are rounded up to 16 byte boundaries before being aggregated.
LOD 3 Main Limit	The maximum main memory available in for LOD 3 and Independent allocations.	
LOD 3 Host	The host memory allocation in LOD 3.	
Independent Host	This is the allocation made in host memory for things like animations but in host memory as opposed to main.	The independent host allocations are made within the Low LOD Host budget, as the Low LOD is always loaded for a given avatar (so the memory is always available).
Total LOD 3 Host	The aggregate of LOD 3 host and independent host allocations.	The LOD 3 host and Independent host figures are rounded up to 16 byte boundaries before being aggregated.
LOD 3 Host Limit	The maximum host memory available in for LOD 3 and independent allocations.	
LOD 3 Mesh count	The number of meshes used in LOD 3.	
LOD 3 Mesh Limit	The number of meshes available in LOD 3.	
Error Message	Any relevant error messages.	

Batch Validator Results

Individual Non-Scripted Object and Batch Validator Results Files

> The report of the validation for an object creates the exact same files, whether you have validated an individual object (one furniture item or character component), or validated a lot of objects at once (with the Batch Validator).

After 'Finished' is displayed, you have in the results folder:

- 1 .xml file called <OBJECT_ID>.xml
- 3 screenshots per object (if screenshots are turned on)
- 1 .csv (comma separated values) file called objectprofilerrestuls-X.csv
- 1 .xml file called objectprofilerresults-X.xml

> The X in the .xml and .csv files (objectprofilerresults-X.xml and objectprofilerresults-X.csv) starts at 1 and increments by one for each validation session to ensure it does not overwrite previous reports.

The <OBJECT_ID>.xml file reports the basic information on the object. An example of what this file contains is:

```
<objectprofiler>

    <object_id>********-********-********-********</object_id>

    <loaded>true</loaded>

    <name>My Object</name>

    <author>name_surname</author>

    <test_type>furniture</test_type>

    <result>

        <lod>1</lod>

        <memory>

        <host>69632</host>

        <main>50752</main>

        <vram>172672</vram>

        <mesh_count>3</mesh_count>

        </memory>

    </result>

</objectprofiler>
```

Validation Screenshots

If you have the screenshots option turned on for validation, then three screenshots will be created, one for each LOD, as .jpg files.

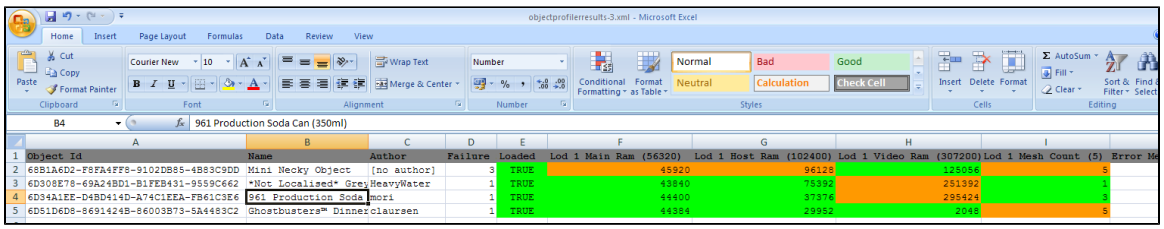
The file format is <Object ID>-<LOD number>.jpg. For example, an object whose Object ID is 00-00-00-00-X will have three images called:

- 00-00-00-00-00-X-1.jpg
- 00-00-00-00-00-X-2.jpg
- 00-00-00-00-00-X-3.jpg

Object Profiler Results XML File

The objectprofilerresults-X.xml file can be opened directly into a Microsoft Excel spreadsheet. The xml file has been created to be used in Microsoft Office Excel specifically, and using other spreadsheet applications may produce unexpected formatting and results.

The report for furniture should look like this:



> The format (columns and colors) of the spreadsheet is the same for individual objects, Male Clothing, and Female Clothing; however, in the case of individual objects, there is only one row with data on it.

The following columns are provided:

Column	Description
Object ID	The object ID for the object that has been validated.
Loaded	True if it managed to load successfully and False if it failed to load (for instance, because of missing files). A cell shows green if it loaded successfully and red if it failed.
Memory (Main, Host, VRAM)	The number of bytes used for each of these memory types. If it is below 80% of the maximum value, the cell is green, if it is 80-100% it is orange and if it is over budget (>100%) then it is red.
Mesh Count	The number of meshes at runtime (which is potentially different from the number of meshes in the DCC tools, as runtime meshes count the number or mesh-material combinations). If it is below 80% of the maximum value, the cell is green. If it is 80-100% it is orange and if it is over budget (>100%) then it is red.
Exceeds Resources	If the object loads successfully, it is given a point of 1 for each Main, Host, VRAM, and mesh count that uses between 80-100% of the resource's total and 100 when it is greater than 100%.
Failure Score	The failure score is calculated in the following way: • Failed to Load: If it failed to load it is given a score of 1000000. • Exceeds Resources: If an objects exceeds a limit (in HOST, VRAM, etc) that property is given a score of 100. • Approaches Resource Limit: If an object uses between 80-100% of the resource's limit, it is given a score of 1 point. For example, the limit for a Female LOD1 Leg Component is 87 KB. If the Batch Validator sees that a Female LOD1 Leg Component is using 80-100% of 87 KB (69.6 -87 KB), then 1 point will be added to the failure score. When objects have a score of 100 or greater in the failure column, the Batch Validator fails them. It is useful to sort the results by the failure score to group all the failed objects together.
Error Message	An error message is given describing why the object failed. Some may not be filled in so it is advisable to save the TTY when running the report.

Using the Batch Validator for Furniture and Clothing

For furniture and clothing (that is, objects that are not scripted and have model files), use the Batch Validator (Static Object Profiler) for profiling.

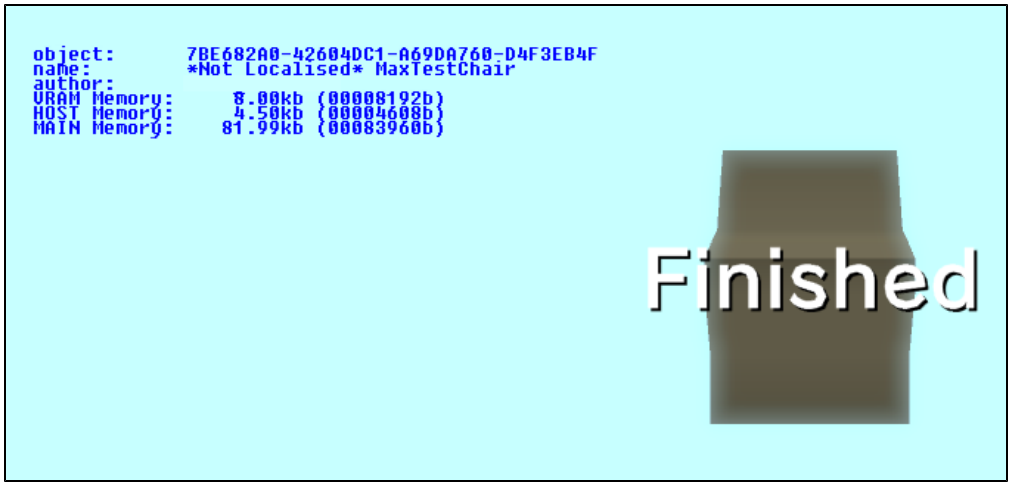
The Static Object Profiler is a space in which you can load objects that are not scripted and get statistics on the resources they use. The Static Object Profiler runs offline. This means it profiles only your local objects.

It establishes how much memory they use, and counts the number of meshes in:

- Furniture (host memory, VRAM, main memory and mesh count).
- Clothing (host memory, VRAM and mesh count).

The profiler space file is located in: <HDK_INSTALL_DIR>\build\batchvalidator

You trigger object profiling by selecting Object > Validate on PS3 in the Object Editor. When you enable the Object Profiler, you see the following type of information:



A message is displayed on the screen when profiling has been completed.

The Object Profiler outputs the following files:

- 3.jpeg files
- A .csv file
- An .xml file

These files display the profile for each LOD. They are stored here: `\build\ObjectProfiler`

The HOST Memory listed on each .jpeg for each LOD should be under the limits specified in Character Components (Clothing).

For more information, see Batch Validator.

Batch Validator Clothing Limits

You use the Batch Validator to validate both male and female clothing items. The Batch Validator iterates through clothing items for a given gender. It produces individual reports, and a single combined table of resource allocation compared to resource limits.

The following tables show the Batch Validator clothing limits for each Level of Detail (LOD). You can obtain these values using the console command `Person.Dbg.ShowLimits`. This command lists the limits for the current debug person. Unless modified explicitly, the debug person is the Local Player.

See also:

- Profiling Character Components for information on profiling using the Clothing Dimensions Profiler.
- Dimensions for recommended dimensions to use.

Male High (1) LOD

Component Type	Main	Host	Mesh	VRAM	Comprised of Atomic Types
Race	5248	392192	8	137216	N/A
Hair	2688	65536	5	1748992	N/A
Hands	2688	98340	5	437120	N/A
Tors	2688	111616	5	1070720	N/A
Legs	2688	89088	5	699392	N/A
Feet	2688	69632	5	831488	N/A
JewelLeftEar	2688	8192	5	14336	N/A
JewelRightEar	2688	8192	5	14336	N/A
Glasses	2688	25600	5	33792	N/A
HeadPhones	2688	22528	5	45056	N/A
FacialHair	2688	49152	5	131328	N/A

Hat	2688	65536	5	1748992	Hair
LegsFeet	5376	158720	10	1530880	Legs Feet
TorsoLegs	5376	200704	10	1770112	Tors Legs
TorsoLegsFeet	8064	270336	15	2601600	Tors Legs Feet
FullBodySuit	32128	940068	58	5163776	Race Hair FacialHair HeadPhones Glasses JewelLeftEar JewelRightEar TorsHands LegsFeet
FullHeadDress	21376	571392	38	2125056	Race Hair FacialHair HeadPhones Glasses JewelLeftEar JewelRightEar
HeadDress	13440	153600	25	1954048	Hair FacialHair HeadPhones JewelLeftEar JewelRightEar
JewelBothEars	5376	16384	10	28672	JewelLeftEar JewelRightEar

Male Medium (2) LOD

Component Type	Main	Host	Mesh	Comprised of Atomic Types
Race	3072	73728	5	N/A
Hair	2688	35840	3	N/A
Hands	2688	53248	3	N/A
Tors	2688	80896	3	N/A
Legs	2688	56320	3	N/A
Feet	2688	43008	3	N/A
JewelLeftEar	2688	4096	3	N/A
JewelRightEar	2688	4096	3	N/A
Glasses	2688	25600	3	N/A
HeadPhones	2688	11264	3	N/A
FacialHair	2688	18432	3	N/A
Hat	2688	35840	3	Hair
LegsFeet	5376	99328	6	Legs Feet
TorsoLegs	5376	137216	6	Tors Legs
TorsoLegsFeet	8064	180224	9	Tors Legs Feet
FullBodySuit	29952	393216	35	Race Hair FacialHair HeadPhones Glasses JewelLeftEar JewelRightEar Tors Hands Legs Feet
FullHeadDress	19200	159744	23	Race Hair FacialHair HeadPhones Glasses JewelLeftEar JewelRightEar
HeadDress	13440	73728	15	Hair FacialHair HeadPhones JewelLeftEar JewelRightEar
JewelBothEars	5376	8192	6	JewelLeftEar JewelRightEar

Male Low (3) LOD

Component Type	Main	Host	Mesh	Comprised of Atomic Types
Race	11904	17408	3	N/A
Hair	11904	22528	3	N/A
Hands	11904	22528	3	N/A
Tors	11904	38912	3	N/A
Legs	11904	30720	3	N/A
Feet	11904	18432	3	N/A
JewelLeftEar	11904	2048	3	N/A

JewelRightEar	11904	2048	3	N/A
Glasses	11904	9216	3	N/A
HeadPhones	11904	6144	3	N/A
FacialHair	11904	5120	3	N/A
Hat	11904	22528	3	Hair
LegsFeet	23808	49152	6	Legs Feet
TorsoLegs	23808	69632	6	Tors Legs
TorsoLegsFeet	35712	88064	9	Tors Legs Feet
FullBodySuit	130944	175104	33	Race Hair FacialHair HeadPhones Glasses JewelLeftEar JewelRightEar Tors Hands Legs Feet
FullHeadDress	83328	64512	21	Race Hair FacialHair HeadPhones Glasses JewelLeftEar JewelRightEar
HeadDress	59520	37888	15	Hair FacialHair HeadPhones JewelLeftEar JewelRightEar
JewelBothEars	23808	4096	6	JewelLeftEar JewelRightEar

Female High (1) LOD

Component Type	Main	Host	Mesh	VRAM	Comprised of Atomic Types
Race	5248	391168	8	146432	N/A
Hair	2688	65536	5	875520	N/A
Hands	2688	104448	5	787456	N/A
Tors	2688	111616	5	1224704	N/A
Legs	2688	89088	5	1246208	N/A
Feet	2688	69632	5	831488	N/A
JewelLeftEar	2688	8192	5	14336	N/A
JewelRightEar	2688	8192	5	14336	N/A
Glasses	2688	24576	5	33792	N/A
HeadPhones	2688	22528	5	44032	N/A
Hat	2688	65536	5	875520	Hair
LegsFeet	5376	158720	10	2077696	Legs Feet
TorsoLegs	5376	200704	10	2470912	Tors Legs
TorsoLegsFeet	8064	270336	15	3302400	Tors Legs Feet
FullBodySuit	29440	894976	53	5218304	Race Hair HeadPhones Glasses JewelLeftEar JewelRightEar Tors Hands Legs Feet
FullHeadDress	18688	520192	33	1128448	Race Hair HeadPhones Glasses JewelLeftEar JewelRightEar
HeadDress	10752	104448	20	948224	HeadPhones JewelLeftEar JewelRightEar
JewelBothEars	5376	16384	10	28672	JewelLeftEar JewelRightEar

Female Medium (2) LOD

Component Type	Main	Host	Mesh	Comprised of Atomic Types
Race	3072	79872	5	N/A
Hair	2688	35840	3	N/A
Hands	2688	57344	3	N/A
Tors	2688	80896	3	N/A
Legs	2688	56320	3	N/A

Feet	2688	43008	3	N/A
JewelLeftEar	2688	4096	3	N/A
JewelRightEar	2688	4096	3	N/A
Glasses	2688	13312	3	N/A
HeadPhones	2688	12288	3	N/A
Hat	2688	35840	3	Hair
LegsFeet	5376	99328	6	Legs Feet
TorsoLegs	5376	137216	6	Tors Legs
TorsoLegsFeet	8064	180224	9	Tors Legs Feet
FullBodySuit	27264	387072	32	Race Hair HeadPhones Glasses JewelLeftEar JewelRightEar Tors Hands Legs Feet
FullHeadDress	16512	149504	20	Race Hair HeadPhones Glasses JewelLeftEar JewelRightEar
HeadDress	10752	56320	12	HeadPhones JewelLeftEar JewelRightEar
JewelBothEars	5376	8192	6	JewelLeftEar JewelRightEar

Female Low (3) LOD

Component Type	Main	Host	Mesh	Comprised of Atomic Types
Race	11904	17408	3	N/A
Hair	11904	22528	3	N/A
Hands	11904	22528	3	N/A
Tors	11904	44032	3	N/A
Legs	11904	38912	3	N/A
Feet	11904	18432	3	N/A
JewelLeftEar	11904	2048	3	N/A
JewelRightEar	11904	2048	3	N/A
Glasses	11904	7168	3	N/A
HeadPhones	11904	7168	3	N/A
Hat	11904	22528	3	Hair
LegsFeet	23808	57344	6	Legs Feet
TorsoLegs	23808	82944	6	Tors Legs
TorsoLegsFeet	35712	101376	9	Tors Legs Feet
FullBodySuit	119040	182272	30	Race Hair HeadPhones Glasses JewelLeftEar JewelRightEar Tors Hands Legs Feet
FullHeadDress	71424	58368	18	Race Hair HeadPhones Glasses JewelLeftEar JewelRightEar
HeadDress	47616	33792	12	Hair HeadPhones JewelLeftEar JewelRightEar
JewelBothEars	23808	4096	6	JewelLeftEar JewelRightEar

HDK Tools Validations

For Japanese Live Page
最新の日本語版ページはこちら 「HDKツールでの検証」

This section directs you to a summary of automatic and manual validations carried out in Maya, the Object Editor and the Scene Editor.

Each topic also includes memory limits and any other requirements that will help your content to pass Quality Assurance (QA).

- Global Validations
- Model Validations, including animated models
- Space Validations
- Character Components (Clothing) Validations
- Custom Avatar Animation Validations
- Furniture Validations
- Active Item Validations
- Object Editor Validations
- Scene Editor Validations

For a summary of memory limits, see Memory Limits.

When your content is ready to submit to the Content Delivery System (CDS), make sure each file size does not exceed 100B. The file size is automatically validated and any files that exceed this limit generate an error.

> When you submit your content, all the automatic tools validations for the type of object or scene are run again.

Automatic Validation in Tools

Automatic validation takes place during the Post Export Validation, which runs when you export content created in Maya. It runs again when you package content in the Scene Editor and Object Editor. If content exceeds the limitations, or has missing content, or does not meet the tool validations in the export and packaging process, you receive one of the following messages:

Message Type	Description	What happens next
Warning	The content exceeds the recommended limits and may not work properly. For example, if you have a sound .bank file larger than 2 MB, you receive a Warning. The .bnk file can be larger than 2 MB, but the warning is in place because larger .bnk files leave you less room to add other types of content to your scene (such as screens). You must test your scene to see if all of your content works as designed with the large .bnk file.	If you receive a Warning, your content still exports and packages successfully, but test your content fully before you send it for publishing.
Error	The content is missing something, or is incorrect. For example, you receive an Error if you try to package an object without thumbnails, or if you try to export a scene without at least one collision mesh.	If you receive an Error, your content does not export or package. Correct the error and try again.
Fatal Error	The content causes a serious error that will crash the Home client. For example, you receive a Fatal Error if you try to package a scene without a spawn point.	If you receive a Fatal Error, your content does not export or package. Correct the error and try again.

> Assets that are very close to the memory limits may fail validation. The margin is about 2-3%.

The art and editor tools packaged in the HDK (Maya, the Scene Editor and Object Editor) provide validation that automatically measures the content against the limits. This automatic validation takes place when exporting or packaging, and produces informative error logs. Using the tools and automatic validation processes automates the majority of the testing for content and helps ensure a lot of the technical requirements are automatically checked and, in some cases, automatically fixed.

The tools check recommendations as well as requirements, resulting in information, warnings or errors on validation. Pay attention to the recommendations, as these aid in content development, but do not confuse recommendations with requirements. Only the requirements are tested by QA and if content does not adhere to requirements, it fails.

For example

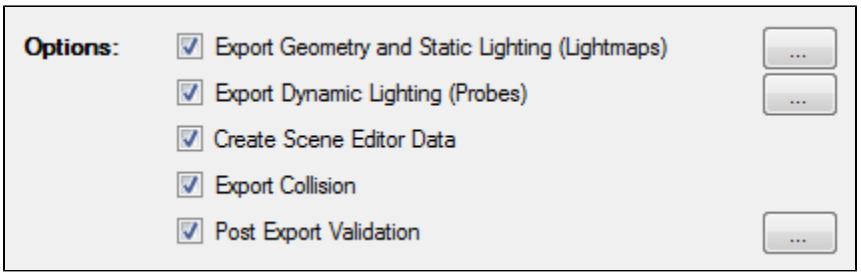
The triangle counts are only a suggestion and not a hard limit or requirement. They are meant to function as a handy indication for artists when planning and developing assets. If they are adhered to strictly, it does not guarantee that content will fulfill the requirements and is no guarantee the content will pass QA. Conversely, if the triangle counts are exceeded it does not prevent content from being published, providing the requirements (e.g. memory limits, etc.) are all met.

Exporting and packaging content using these tools and their validations is a requirement for publication. Knowing what these limits are assists in the planning and design phases of content production. This document lists all the validations that the tools perform. Also see the error logs produced by the tools on export and packaging.

You can export content from Maya with or without Post Export Validation. Exporting content without Post Export Validation selected does not prevent the content working in Scene or Object Editor, but the content is not valid for submission nor publication. Scene Editor and Object Editor, however run all the validations on packaging. Successfully packaged content then requires profiling in runtime. To submit content for publication in PS Home, Post Export Validation must run in Maya, followed by successful packaging in the Editors and thorough manual profiling in the runtime.

Maya

During development of Home content in Maya, you can turn off automatic validation by deselecting the Post Export Validation option. Deselect this option only if you are testing the export to Scene Editor and not packaging the scene for submission to PS Home. After content is finalized, ensure the Post Export Validation option is selected on export.



The Scene Editor and Object Editor

Content that has been exported from Maya can be loaded in the Scene Editor or Object Editor. The Editors automatically validate the content and packaging fails if errors are encountered.

From HDK 1.3.5, when you package content in Scene Editor and Object Editor, the same checks and validations that were run in Maya during Post Export Validation is run on the content.

Packaging

When content is finalized, you can package it and submit it for publication. To finalize content for submission, however, the content must have successfully exported without errors from Maya (with Post Export Validation selected) and the Editors, as well as go through profiling in runtime to ensure all requirements are met.

Global Validations

For Japanese Live Page
最新の日本語版ページはこちら 「全般的な検証」

The following table outlines the requirements to which you must adhere when creating Home content:

Global Manual Validation	Values	Required	Recommended	Check Method
All names (including file names, resource names, asset names, effects names) in scenes must contain only legal characters: a-z, A-Z, 0-9, underscores (_) and hyphens (-).	a-z A-Z 0-9 _ -	x		x
Packaged content (objects and scenes) cannot contain unused files.		x		

Model Validations

For Japanese Live Page
最新の日本語版ページはこちら 「モデルの検証」

This page summarizes both HDK automatic validations and manual validations for models, including animated models. Some validations are required to pass Quality Assurance (QA), other validations are recommended. The validations include memory limits, formatting and any other requirements or recommendations that apply.

In Maya, you can export a 3D shape that has geometry and collision, but is not a public or personal space, a clubhouse, a furniture or active item, a Character component, or an animation. It is a 3D shape that you can use in various ways, such as calling the shape in a mini-game. This 3D shape is referred to generically as a model.

Model files have many applications, so the memory limits and polygon recommendations depend on how they are used. If the model

files are used in a personal space, they contribute to the personal space memory and polygon count. If the model files are used in a furniture, they contribute to the furniture's memory and polygon count.

> ⚠ Models must also adhere to the requirements of whatever is using them. For example, if a mini-game uses a model (as a .mdl resource), the resource contributes to the mini-game's memory usage.

Model Validations

The following table lists the automatic and manual validations that are required or recommended for models.

Models – Automatic Validation (Requirements)	Values	Required	Recommended	Automatically Validated
Geometry meshes have an assigned ATG Material		x		Error
All file references (textures) exist on the local disk		x		Error
There is at least one Geometry mesh in the model		x		Error
The geometry should have no prelit meshes/shaders		x		Error
There is at least one Collision mesh in the model		x		Error
There is only one Collision group in the model		x		Error
All meshes have necessary UV sets for their shader, e.g., the default set requires map1 (or channel1), prelit_default requires both map1 and map2 (or channel1 and channel2)		x		Error
All file references should be under the Artsource folder, but may also be stored in the Build folder		x		Error
Texture paths must contain legal characters only	a-z or A-Z 0-9 _ -	x		Error
Texture sizes are in powers of 2, i.e. 2, 8, 16, 32, 64, 128, 256, ...		x		Error

Models – Automatic Validation (Recommendations)	Values	Required	Recommended	Check Method
Dimensions (Geometry and LightVolume meshes for the entire scene)	Maximum 10,000 x 5,000 x 10,000 m		x	Warning

Models – Manual Validations (Requirements)	Values	Required	Recommended	Check Method
Collision meshes have pivot points at their local origin		x		
Texture size limits: • Maximum: 2048 x 2048 px • Recommended: 1024 x 1024 px	2048 x 2048	x		
Reflection texture size limits: • Maximum: 1024 x 1024 px • Recommended: 512 x 512 px	1024 x 1024	x		

Models – Manual Validations (Recommendations)	Values	Required	Recommended	Check Method
For collision, all environments and objects have basic polygonal forms representing solid surfaces simply. For example, simple curves are included in the shape of the collision mesh, but details such as sofa legs are not.			x	

Animated Models

The following table lists the automatic and manual validations that are required or recommended for animated models.

Animated Models – Automatic Validations (Requirements)	Values	Required	Recommended	Automatically Validated
Geometry meshes have an assigned ATG Material		x		Error
All file references (textures) exist on the local disk		x		Error
There has to be at least one Geometry mesh		x		Error
The geometry should have no prelit meshes/shaders		x		Error
All meshes have necessary UV sets for their shader, e.g., default requires map1 (or channel1), prelit_default requires both map1 and map2 (or channel1 and channel2)		x		Error
All Geometry meshes do not have map2 UV set (this causes animation issues)		x		Error
All animated model scenes must be animated (and not just contain vertex animation)		x		Error
All file references should be under the Artsource folder, but may also be stored in the Build folder		x		Error
Texture paths must contain legal characters only	a-z or A-Z 0-9 _ -	x		Error

Animated Models – Automatic Validations (Recommendations)	Values	Required	Recommended	Check Method
Dimensions (Geometry + LightVolume meshes for entire scene)	Maximum 10,000 x 5,000 x 10,000 m		x	Warning

Animated Models – Manual Validations (Requirements)	Values	Required	Recommended	Check Method
Texture size limits: • Maximum: 2048 x 2048 px • Recommended: 1024 x 1024 px	2048 x 2048	x		
Reflection texture size limits: • Maximum: 1024 x 1024 px • Recommended: 512 x 512 px	1024 x 1024	x		

Animated Models – Manual Validations (Recommendations)	Values	Required	Recommedned	Check Method

Triangle limits for Animated Models (within spaces) - recommended maximum: 100,000 triangles ⚠ Tools give a warning only at 250,000 but the animated model must be placed within a space, which must conform to the limits on that type of space.	100,000		x	

Space Validations

For Japanese Live Page
最新の日本語版ページはこちら 「シーンの検証」

This page summarizes both HDK automatic validations and manual validations for public spaces, personal spaces and clubhouses. Some validations are required to pass QA, other validations are recommended. The validations include memory limits, formatting and any other requirements or recommendations that apply.

Dynamic Reflections

Only scenes can have dynamic reflection mapping. Character components and objects (furniture, active items) cannot have dynamic reflection mapping.

Public Space Validations

Public Spaces – Automatic Validations (Requirements)	Values	Required	Recommended	Automatically Validated
VRAM limit	74 MB	x		Error
Main memory limit	12 MB	x		Error
Host memory limit	17 MB	x		Error
Geometry meshes have an assigned ATG Material		x		Error
No materials are of prelit type. This is an obsolete type replaced by prelit_default		x		Error
All file references (textures) exist on the local disk		x		Error
You must have at least one Geometry mesh		x		Error
Mesh names cannot have spaces or illegal characters	illegal characters .¥/ | :*?"<>			Error
A scene must have at least one LightVolume mesh		x		Error
A scene must have least one prelit mesh (either prelit_default or prelit_glass)		x		Error
A directional light named 'sunShape' exists		x		Error
All meshes have necessary UV sets for their shader. For example, default requires map1 (or channel1), prelit_default requires both map1 and map2 (or channel1 and channel2)		x		Error
All file references should be under the Artsource folder, but may also be stored in the Build folder		x		Error
Texture paths must contain legal characters only	a-z or A-Z 0-9 _ -	x		Error
Texture sizes are in powers of 2, i.e., 2, 8, 16, 32, 64, 128, 256, ...		x		Error

There is at least one Collision mesh in the scene		x		Error
Particle effect files (.efx) are from the Build folder and exist		x		Error
NURBS and SubDiv surfaces are not supported Convert to polygonal geometry before exporting.		x		Error
Runtime mesh limit	4,096	x		Error
Object pre-download limit	8			Error

Public Spaces – Automatic Validations (Recommendations)	Values	Required	Recommended	Check Method
Probe file size recommendation	2 MB		x	Warning
Collision file size recommendation	1 MB		x	Warning
Texture size recommendation	40 MB		x	Warning
Recommended triangle limits for public spaces	250,000		x	Warning
Dimensions (geometry for entire scene): • Maximum: 10,000 (x) x 5,000 (y) x 10,000 (z) m • Min: 1.5 (x) x 2.44 (y) x 1.5 (z) m (the same as corridor minimum dimensions)			x	Warning
Lightmap UVs (map2/channel 2) do not intersect			x	Warning
Lightmap UVs (map2/channel 2) are between 0 and 1			x	Warning
The maximum LightVolume in cubic meters	10,000		x	Warning
Objects in the lighting_ignore Lighting set do not appear in any other lighting sets			x	Warning

Public Spaces – Manual Validations (Requirements)	Values	Required	Recommended	Check Method
Texture size limits: • Maximum: 2048 x 2048 px • Recommended: 1024 x 1024 px	2048x2048	x		
Reflection texture size limits: • Maximum: 1024 x 1024 px • Recommended: 512 x 512 px	1024x1024	x		

Public Spaces – Manual Validations (Recommendations)	Values	Required	Recommended	Check Method
DXT1, DXT3 and DXT5 comparable texture compression is used in DDS image format			x	
A 512 texture set, made up of 512 x 512 px maps, a color map (with alpha channel), a normal map and a specular map (with alpha channel)			x	
Combinations of measures/dimensions are allowed, e.g., 128 x 512, 256 x 64			x	

For collision, all environments and objects should have basic polygonal forms representing solid surfaces simply, e.g., simple curves are included in the shape of the collision mesh, but details such as sofa legs are not			x	
Floor collision at least 1m to prevent small furniture items from falling through			x	

Personal Space Validations

Personal Spaces – Automatic Validations (Requirements)	Values	Required	Recommended	Automatically Validated
VRAM limit	54 MB	x		Error
Main memory limit	8.5 MB	x		Error
Host memory limit	10 MB	x		Error
Geometry meshes have an assigned ATG Material		x		Error
No materials are of prelit type. This is an obsolete type replaced by prelit_default		x		Error
All file references (textures) exist on the local disk		x		Error
There has to be at least one Geometry mesh		x		Error
Mesh names do not have spaces or illegal characters	illegal characters .¥/ | :*?"<>	x		Error
There is at least one LightVolume mesh in the scene		x		Error
There is at least one prelit mesh in the scene (either prelit_default or prelit_glass)		x		Error
A directional light named 'sunShape' exists		x		Error
All meshes have necessary UV sets for their shader, e.g., default requires map1 (or channel1), prelit_default requires both map1 and map2 (or channel1 and channel2)		x		Error
All file references should be under the Artsource folder, but may also be stored in the Build folder		x		Error
Texture paths must contain legal characters only	a-z or A-Z 0-9 _ -	x		Error
Texture sizes are in powers of 2, i.e., 2, 8, 16, 32, 64, 128, 256, ...		x		Error
There is at least one Collision mesh in the scene		x		Error
Particle effect files (.efx) are from the Build folder and exist				Error
NURBs and SubDiv surfaces are not supported Convert to polygonal geometry before exporting.		x		Error
Runtime mesh limit	4,096	x		Error

Personal Spaces – Automatic Validations (Recommendations)	Values	Required	Recommended	Check Method
Probe file size recommendation	2 MB		x	Warning
Collision file size recommendation	1 MB		x	Warning

Texture size recommendation	40 MB		x	Warning
Dimensions (geometry for the entire scene): • Maximum: 10,000 (x) x 5,000 (y) x 10,000 (z) m • Min: 1.5 (x) x 2.44 (y) x 1.5 (z) m (the same as corridor minimum dimensions)			x	Warning
Lightmap UVs (map2/channel 2) do not intersect			x	Warning
Lightmap UVs (map2/channel 2) are between 0 and 1			x	Warning
The maximum LightVolume in cubic meters	10,000		x	Warning
Objects in the lighting_ignore Lighting set do not appear in any other lighting sets			x	Warning

Personal Spaces – Manual Validations (Requirements)	Values	Required	Recommended	Check Method
Texture size limits: • Maximum: 2048 x 2048 px • Recommended: 1024 x 1024 px	2048x2048	x		
Reflection texture size limits: • Maximum: 1024 x 1024 px • Recommended: 512 x 512 px	1024x1024	x		

Personal Spaces – Manual Validations (Recommendations)	Values	Required	Recommended	Check Method
Recommended triangle limits for personal spaces Tools will only give a warning at 250,000.	100,000		x	–
DXT1, DXT3 and DXT5 comparable texture compression is used in DDS image format			x	
A 512 texture set, made up of 512 x 512 px maps, a color map (with alpha channel), a normal map and a specular map (with alpha channel)			x	
Combinations of measures/dimensions are allowed, e.g., 128 x 512, 256 x 64, etc.			x	
For collision, all environments and objects have basic polygonal forms representing solid surfaces simply, e.g., simple curves are included in the shape of the collision mesh, but details such as sofa legs are not			x	
Floor collision at least 1m to prevent small furniture items from falling through			x	

Clubhouse Validations

Clubhouses – Automatic Validations (Requirements)	Values	Required	Recommended	Automatically Validated
VRAM limit	54 MB	x		Error
Main memory limit	8.5 MB	x		Error
Host memory limit	10 MB	x		Error
Geometry meshes have an assigned ATG Material		x		Error

No materials are of prelit type. This is an obsolete type replaced by prelit_default		x		Error
All file references (textures) exist on the local disk		x		Error
There has to be at least one Geometry mesh		x		Error
Mesh names do not have spaces or illegal characters	illegal characters .¥/ | :*?"<>	x		Error
There is at least one LightVolume mesh in the scene		x		Error
There is at least one prelit mesh in the scene (either prelit_default or prelit_glass)		x		Error
A directional light named 'sunShape' exists		x		Error
All meshes have necessary UV sets for their shader, e.g., default requires map1 (or channel1), prelit_default requires both map1 and map2 (or channel1 and channel2)		x		Error
All file references should be under the Artsource folder, but may also be stored in the Build folder		x		Error
Texture paths must contain legal characters only	a-z or A-Z 0-9 _ -	x		Error
Texture sizes are in powers of 2, i.e., 2, 8, 16, 32, 64, 128, 256, ...		x		Error
There is at least one Collision mesh in the scene		x		Error
Particle effect files (.efx) are from the Build folder and exist		x		Error
NURBs and SubDiv Surfaces are not supported Convert to polygonal geometry before exporting.		x		Error
Runtime mesh limit	4,096	x		Error

Clubhouses – Automatic Validations (Recommendations)	Values	Required	Recommended	Check Method
Probe file size recommendation	2 MB		x	Warning
Collision file size recommendation	1 MB		x	Warning
Texture size recommendation	40 MB		x	Warning
Dimensions (geometry for the entire scene): • Maximum: 10,000 (x) x 5,000 (y) x 10,000 (z) • Min: 1.5 (x) x 2.44 (y) x 1.5 (z) m (the same as corridor minimum dimensions)			x	Warning
Lightmap UVs (map2/channel 2) do not intersect			x	Warning
Lightmap UVs (map2/channel 2) are between 0 and 1			x	Warning
Maximum LightVolume in cubic meters	10,000		x	Warning
Objects in the lighting_ignore Lighting set do not appear in any other lighting sets			X	Warning

Clubhouses – Manual Validations (Requirements)	Values	Required	Recommended	Check Method
Texture size limits: • Maximum: 2048 x 2048 px • Recommended: 1024 x 1024 px	2048x2048	x		
Reflection texture size limits: • Maximum: 1024 x 1024 px • Recommended: 512 x 512 px	1024x1024	x		
All clubhouses must have a club bulletin screen		x		
Club bulletin screens must have the appropriate dimensions and screen to message ratio, e.g.: • Screen width and height • Screen message margin • Screen message width and height	MessageX cannot be smaller than MessageMarginX*2 + 135 MessageY cannot be smaller than MessageMarginY*2 + 70	x		
Clubhouses cannot contain picture hooks		x		

Clubhouses – Manual Validations (Recommendations)	Values	Required	Recommended	Check Method
Recommended triangle limits for clubhouses Tools only give a warning at 250,000.	100,000		x	
DXT1, DXT3 and DXT5 comparable texture compression is used in DDS image format			x	
A 512 texture set, made up of 512 x 512 px maps, a color map (with alpha channel), a normal map and a specular map (with alpha channel)			x	
Combinations of measures/dimensions are allowed, e.g., 128 x 512, 256 x 64, etc.			x	
For collision, all environments and objects have basic polygonal forms representing solid surfaces simply, e.g., simple curves are included in the shape of the collision mesh, but details such as sofa legs are not				
Floor collision at least 1m to prevent small furniture items from falling through			x	

Recommended Rates for Rendering and Processing Spaces

Scenes (including all mini-games) should execute and render at a certain speed, when there are the maximum number of characters present and all scene objects are active.

The following table summarizes the rates for the different types of space:

Public Spaces, Game Spaces, Video Space	Personal Spaces	Clubhouses
24 ms/f	16 ms/f	16 ms/f

Sky scenes consume 1.5 – 2.0 ms/f above the recommended rates.

The following table shows the recommended rendering rates for frames. It is better to measure in milliseconds per frame (ms/f), rather than frames per second (fps), because PS Home scenes are locked at 30 frames per second.

Frames Per Second	Milliseconds Per Frame
30 fps	33.33 ms/f
41.67 fps	24 ms/f
62.5 fps	16 ms/f

PS Home tells the PlayStation®3 to render at a rate of 30 frames per second, that is, a new frame every 33.33 ms.

If your scene takes less time than 33.33 ms to render a frame, the PlayStation®3 will lie idle during the time your frame finishes rendering and the PlayStation®3 begins to render the next frame. This means for public scenes running at the recommended 24 ms, the PlayStation®3 will lie idle for (33.33-24) 9.33 ms. The next frame will not start as soon as its predecessor has finished rendering because of the lock at 30 fps.

You can monitor your ms/f rate using the profiling tools (see Profile GUI, or by entering `objectprofiler 1` into the Debug Console). You can also monitor CPU and GPU rates, as well as fps, using the profiler, but fps is locked at 30.

Scripts and other processes affect both CPU and GPU. Whichever processor is slower will bottleneck your rendering speed. For example, if your CPU runs at 1 ms/f, but the GPU runs at 50 ms/f, your scene will run at 50 ms/f.

Character Components (Clothing) Validations

For Japanese Live Page
最新の日本語版ページはこちら 「キャラクタコンポーネント（服）の検証」

Character components are split into different subcategories, including three Levels of Detail (LODs) for Host memory, male and female, and types of component. The HDK Tools automatically validate the following limits for character components:

- VRAM limits
- HOST limits
- Main memory limits
- Mesh limits
- Triangle limits

You can combine certain adjoining character component slots to make a composite character component. For example, you can combine Legs + Feet (LegsAndFeet) components to make a skirt and high boots combination. In this case, the memory limits are the sum of each component used.

Memory Limits

VRAM Limits

The following table lists the required VRAM limits for both atomic and composite character components. The values are for LOD1. The PS Home client creates LODs 2 and 3 automatically based on LOD1.

The Batch Validator automatically checks that the MIP maps are present (on which the lower LODs are calculated). The MIP maps check is necessary because if there are no MIP maps, the same texture that is used for the high LOD is used for the other LODs. As a result, the lower LODs will run out of VRAM.

The VRAM values are automatically validated and, if violated, generate an error.

Character Component VRAM Limits (includes Textures)			
Atomic Character Components	Female	Male	
Head - developers cannot modify the Head slot, but the memory taken up by the Head slot becomes available when creating a Full Head Dress or Full Body Suit character component. The memory limits are therefore for information only.	143	134	KB
FacialHair (Facial Hair)	-	128	KB
Hair (Hairstyle)	855	1708	KB
Hat (Headgear)	855	1708	KB

Hands	769	426	KB
Torso	1196	1045	KB
Legs	1217	683	KB
Feet	812	812	KB
JewelLeftEar (Jewellery (Left ear))	14	14	KB
JewelRightEar (Jewellery (Right ear))	14	14	KB
Glasses (Spectacles)	33	33	KB
HeadPhones (Head Phones)	43	44	KB
Composite Character Components			
LegsFeet (Legs + Feet)	2029	1495	KB
TorsoLegs (Torso + Legs)	2413	1728	KB
TorsoLegsFeet (Torso + Legs + Feet)	3225	2540	KB
JewelBothEars	28	28	KB
FullBodySuit (Head, Torso, Legs, Feet, Hands, Hair, FacialHair, HeadPhones, JewelLeftEar, JewelRightEar, Glasses)	5096	5042	KB
HeadDress (Hair, FacialHair, HeadPhones, JewelLeftEar, JewelRightEar)	926	1908	KB
FullHeadDress (Head, Hair, FacialHair, HeadPhones, JewelLeftEar, JewelRightEar)	1102	2075	KB

Host Limits

The following table list the required Host limits for character components. HOST limits are automatically validated and, if violated, generate an error.

Host Limits – Character Components (includes Model data)				
Female Atomic	LOD1	LOD2	LOD3	
Head – the memory taken up by the Head slot becomes available when creating a Full Head Dress or Full Body Suit character component	380	78	17	KB
Hair (Hairstyle)	64	35	22	KB
Hat (Headgear)	64	35	22	KB
Hands	102	56	22	KB
Torso	109	79	43	KB
Legs	87	55	38	KB
Feet	68	42	18	KB
JewelLeftEar (Jewellery (Left ear))	8	4	2	KB
JewelRightEar (Jewellery (Right ear))	8	4	2	KB
Glasses (Spectacles)	24	13	7	KB
HeadPhones (Head Phones)	22	12	7	KB
Female Composites				
LegsFeet (Legs + Feet)	155	97	56	KB
TorsoLegs (Torso + Legs)	196	134	81	KB
TorsoLegsFeet (Torso + Legs + Feet)	264	176	99	KB
JewelBothEars	16	8	4	KB
FullBodySuit (Full Body Suit)	872	378	178	KB
HeadDress (Head Dress)	102	55	33	KB

FullHeadDress (Full Head Dress)	506	146	57	KB
Male Atomic	LOD1	LOD2	LOD3	
Head: (see Female Atomic above. Only applicable for Full Head Dress and Full Body Suit composites)	380	72	17	KB
FacialHair (Facial Hair)	48	18	5	KB
Hair (Hairstyle)	64	35	22	KB
Hat (Headgear)	64	35	22	KB
Hands	96	52	22	KB
Torso	109	79	38	KB
Legs	87	55	30	KB
Feet	68	42	18	KB
JewelLeftEar (Jewellery (Left ear))	8	4	2	KB
JewelRightEar (Jewellery (Right ear))	8	4	2	KB
Glasses (Spectacles)	25	12	9	KB
HeadPhones (Head Phones)	22	11	6	KB
Male Composites				
LegsFeet (Legs + Feet)	155	97	48	KB
TorsoLegs (Torso + Legs)	196	134	68	KB
TorsoLegsFeet (Torso + Legs + Feet)	264	176	86	KB
JewelBothEars	16	8	4	KB
FullBodySuit (Full Body Suit)	915	384	171	KB
HeadDress (Head Dress)	150	72	37	KB
Full Head Dress (FullHeadDress)	555	156	63	KB

Main Memory Limits

> ⚠ Main memory limits are required only for Full Body Suits. You must adhere to Main memory limits for Full Body Suits with custom avatar animations.

If a character component is over the Main memory limit, the Batch Validator report shows it in red. Although it is not required for all character components, it is strongly recommended that Main memory be within the limits indicated. Exceeding these limits can cause problems with the content in the live environment and adversely affect the user experience. If your content exceeds Main limits memory, try reducing the number of meshes, materials, or the size of the filenames used.

The following table lists different factors that can affect main memory - because the system automatically uses the following amount of memory.

Main Limits - Character Components	Female	Male	
Head (only applicable for Full Head Dress and Full Body Suit composites)	5	5	KB
FacialHair (Facial Hair)	-	2.5	KB
Hairstyle (Hair)/Hat (Headgear)	2.5	2.5	KB
Torso	2.5	2.5	KB
Legs	2.5	2.5	KB
Feet	2.5	2.5	KB
Hands	2.5	2.5	KB
Glasses (Spectacles)	2.5	2.5	KB
HeadPhones (Head Phones)	2.5	2.5	KB

JewelLeftEar (Jewellery (Left ear))	2.5	2.5	KB
JewelRightEar (Jewellery (Right ear))	2.5	2.5	KB
Composites			
LegsFeet (Legs + Feet)	5	5	KB
TorsoLegs (Torso + Legs)	5	5	KB
TorsoLegsFeet (Torso + Legs + Feet)	8	8	KB
JewelBothEars	5	5	KB
FullBodySuit (Full Body Suit)	29	31	KB
HeadDress (Head Dress)	11	13	KB
Full Head Dress (FullHeadDress)	19	21	KB

Mesh Limits

The following table lists Mesh limits for atomic and composite character components.

Mesh Limits	MALE			FEMALE		
Component	LOD1	LOD2	LOD3	LOD1	LOD2	LOD3
Head (only applicable for Full Head Dress and Full Body Suit composites)	8	3	3	8	3	3
FacialHair (Facial Hair)	5	3	3	–	–	–
Hairstyle (Hair)/Hat (Headgear)	5	3	3	5	3	3
Torso	5	3	3	5	3	3
Legs	5	3	3	5	3	3
Feet	5	3	3	5	3	3
Hands	5	3	3	5	3	3
Glasses (Spectacles)	5	3	3	5	3	3
HeadPhones (Head Phones)	5	3	3	5	3	3
JewelLeftEar (Jewellery (Left ear))	5	3	3	5	3	3
JewelRightEar (Jewellery (Right ear))	5	3	3	5	3	3
Composites						
LegsFeet (Legs + Feet)	10	6	6	10	6	6
TorsoLegs (Torso + Legs)	10	6	6	10	6	6
TorsoLegsFeet (Torso + Legs + Feet)	15	9	9	15	9	9
JewelBothEars	10	6	6	10	6	6
FullBodySuit (Full Body Suit)	58	33	33	58	33	33
HeadDress (Head Dress)	25	15	15	25	15	15
FullHeadDress (Full Head Dress)	38	21	21	38	21	21



For composite character components, the mesh limits are still per component. Just take into account how many components the composite spans, and multiply the number of meshes by that amount.

Triangle Limits

The following table lists recommended triangle limits. Triangle limits are for guidance only. They are not indicative of memory usage.

Component	LOD1	LOD2	LOD3
Head - developers cannot modify the Head slot, but the memory taken up by the Head slot becomes available when creating a Full Head Dress or Full Body Suit character component. The triangle limits for Head are therefore for information only.	2700	1360	140
FacialHair (Facial Hair)	700	350	150
Hairstyle (Hair)/Hat (Headgear)	1400	810	426
Torso	2000	1100	470
Legs	1515	830	338
Feet	790	400	132
Hands	2500	1220	390
Glasses (Spectacles)	600	300	150
HeadPhones (Head Phones)	600	300	70
JewelLeftEar (Jewellery (Left ear))	250	175	50
JewelRightEar (Jewellery (Right ear))	250	175	50
Composites			
LegsFeet (Legs + Feet)	2305	1230	470
TorsoLegs (Torso + Legs)	3515	1930	808
TorsoLegsFeet (Torso + Legs + Feet)	4305	2330	940
FullBodySuit (Full Body Suit)	13350	7020	2366
HeadDress (Head Dress)	3200	1810	746
FullHeadDress (Full Head Dress)	3800	2110	896

General Validations

The following table lists some general automatic validations for character components.

Character Components (Clothing) – Automatic Validations (Requirements)	Values	Required	Recommended	Automatically Validated
There can be no geometry with a water shader assigned		x		Error
Geometry must be skinned		x		Error
Geometry meshes have an assigned ATG Material		x		Error
The object and all its associated resources exist and can be loaded		x		Error
The object is designated a valid clothing object (has a valid Rig ID and Component ID)		x		Error
All file references (textures) exist on the local disk		x		Error
There has to be at least one Geometry mesh		x		Error
The geometry should have no prelit meshes/shaders		x		Error
All meshes have necessary UV sets for their shader, e.g., default requires map1 (or channel1), prelit_default requires both map1 and map2 (or channel1 and channel2)		x		Error
Hair material must have a Shading Group of hairSG or facehairSG		x		Error
Skin material must have a Shading Group of bodySG or headSG		x		Error
All components require 3 Levels Of Detail (LODs)		x		Error
Only one skeleton root may exist		x		Error
All file references should be under the Artsource folder, but may also be stored in the Build folder		x		Error

Texture paths must contain legal characters only	a-z or A-Z 0-9 _ -	x		Error
Character Components (Clothing) – Automatic Validations (Recommendations)				
Dimensions (geometry for entire character component) For individual component limits and how to profile them, see Profiling.	Maximum 5 (x) x 5 (y) x 5 (z) m		x	Warning
Runtime mesh limit: • LOD1: 5 • LOD2: 3 • LOD3: 3 For composite character components, the mesh limits are still per component. Just take into account how many components the composite spans, and multiply the number of meshes by that amount. See the table of Mesh Limits for all character components and composites above.	LOD1: 5 LOD2: 3 LOD3: 3	x		Error

Character Component Textures

MIP Maps

MIP maps (also known as mipmaps) for Character component textures are required, but it is strongly recommended that all textures should have MIP maps.

Maya automatically checks for fully MIP mapped textures on export of a character component. If MIP mapped textures are missing, the tools returns an error.

Dimensions

The following table lists the required dimensions for character textures. These are manual validations.

Character Components (Clothing) – Manual Validations						
Head (only applicable for Full Head Dress and Full Body Suit composites)	C	512x512	N	512x512	S	512x512
FacialHair (Facial Hair)	C	128x128	A	128x128	S	64x64
Hairstyle (Hair)/Hat (Headgear)	C	512x512	A	512x512	S	256x256
Torso	C	512x512	N	512x512	S	256x256
Legs	C	512x512	N	512x512	S	256x256
Feet	C	256x256	N	256x256	S	128x128
Hands	C	256x256	N	256x256	S	128x128
Accessories, such as Glasses, HeadPhones, JewelLeftEar, JewelRightEar	C	128x128	N	128x128	S	64x64

Custom Avatar Animation Validations

For Japanese Live Page
最新の日本語版ページはこちら 「カスタムアバターアニメーションの検証」

The following table lists the automatic validations that are required for custom avatar animation.

Custom Avatar Animation – Automatic Validations (Requirements)	Values	Required	Recommended	Automatically Validated

There must be animation (and not just vertex animation)		x		Error
Only one skeleton root may exist		x		Error

Custom avatar animations can be scripted into the following types of content:

- as part of a mini-game in a scene or in furniture (e.g. an active item)
- as part of a FullBodySuit

> See also the requirements for adding and removing repertoires to a Mini-game or Realtime game.

Memory Limits

As the animation is a resource in the object, it inherits the object's memory limits. For example, if it is scripted to be part of an active, it must fit into the memory constraints of actives. You must therefore profile the custom avatar animation according to the memory limits of those objects.

The following table outlines the memory limits to profile against:

Custom Avatar Animations	VRAM	MAIN	HOST
In mini-games in scenes			
Mini-games in scenes use the scene's memory. See Memory Limits by Scene Type. As mini-games must fit within the overall scene budget for memory, the custom avatar animation that is part of the mini-game must also fit within the overall scene.			
In mini-games in furniture			
Actives	2 MB	1 MB	1 MB
In composite character components			
FullBodySuits (see Female and Male versions below)			
Female	5096 KB	64 KB	LOD1: 872 KB LOD2: 378 KB LOD3: 178 KB
Male	5042 KB	64 KB	LOD1: 915 KB LOD2: 384 KB LOD3: 171 KB

> You must adhere to the main memory limits for FullBodySuits with custom avatar animations. Main memory for composite character components is reported for LOD1 in the Batch Validator. As long as this value is below the limits specified, the FullBodySuit should have the correct LOD2 and LOD3 memory usage.

Furniture Validations

> For Japanese Live Page
> 最新の日本語版ページはこちら 「家具アイテムの検証」

This page summarizes both HDK automatic validations and manual validations for furniture. Some validations are required to pass Quality Assurance (QA), other validations are recommended. The validations include memory limits, formatting and any other requirements or recommendations that apply.

You cannot script furniture objects unless they are defined as an Active Item. Standard furniture objects do not, therefore, have a main memory requirement and developers do not have much control over how much memory is used from this pool. For Active Item memory requirements, see Active Item Validations.

The following table lists the automatic and manual validations that are required or recommended for furniture:

Automatic Validations

Furniture – Automatic Validations (Requirements)	Values	Required	Recommended	Automatically Validated
VRAM limit (includes textures)	300 KB	x		Error
Host memory limit (includes model data)	100 KB	x		Error
MAIN memory limit	48 KB	x		Error
Geometry meshes have an assigned ATG Material		x		Error
The object and all its associated resources exist and can be loaded		x		Error
The object is designated a furniture object (has an Furniture component, an appliance ID, category ID, model reference and collision reference)		x		Error
All file references (textures) exist on the local disk		x		Error
There are no LightVolume meshes		x		Error
There has to be at least one Geometry mesh		x		Error
There is at least one Collision mesh		x		Error
There is only one Collision group		x		Error
The geometry should have no prelit meshes/shaders		x		Error
All meshes have necessary UV sets for their shader, e.g., default requires map1 (or channel1), prelit_default requires both map1 and map2 (or channel1 and channel2)		x		Error
All file references should be under the Artsource folder, but may also be stored in the Build folder		x		Error
Texture paths must contain legal characters only	a-z or A-Z 0-9 _ -	x		Error
Maximum number of seats is 3	3	x		Error
Maximum number of lights is 1	1	x		Error
Runtime mesh limit	5	x		Error
Picture Frame dimensions (geometry for picture frames)	Maximum 2 x 2 x 0.03 m	x		Error
Furniture – Automatic Validations (Recommendations)				
Recommended maximum triangle limits	4,000		x	Warning

Manual Validations

Furniture – Manual Validations (Requirements)	Values	Required	Recommend	Check Method
For picture frames, screens must be placed in the positive Z-axis.		x		
Furniture – Manual Validations (Recommendations)				
For picture frames, place all geometry you want users to see in the positive Z-axis			x	

For collision, all environments and objects have basic polygonal forms representing solid surfaces simply. For example, simple curves are included in the shape of the collision mesh, but details such as sofa legs are not.			x	
Furniture may have environment maps, but it is sensible to avoid the use of dynamic reflective surfaces.			x	
Recommended dimensions (geometry for furniture)	Max 3 (x) x 2.44 (y) x 4 (z) Min 0.1 (x) x 0.1 (y) x 0.1 (z) m		x	

Active Item Validations

For Japanese Live Page
最新の日本語版ページはこちら 「アドバンスド家具の検証」

⚠ Limits specified apply to the entire active item (including model and textures, dynamically called or otherwise). See Lua Scripting for more details on active items.

The following table lists the automatic and manual validations that are required or recommended for active items:

Active Items – Automatic Validation (Requirements)	Values	Required	Recommended	Automatically Validated
VRAM limit. Includes: • Textures • Images • Video • Content on screens	2 MB	x		Error
Host memory limit. Includes model and particle data	1 MB	x		Error
Main memory limit. Includes: • Sounds/audio • Scripts • Script variables • Script libraries • Animation data	1 MB	x		Error
Geometry meshes have an assigned ATG Material		x		Error
The object and all its associated resources exist and can be loaded		x		Error
The object is designated an active object (has an Active Item/Furniture component, an appliance ID, category ID, model reference and collision reference)		x		Error
All file references (textures) exist on the local disk		x		Error
There has to be at least one Geometry mesh		x		Error
There is at least one Collision mesh in the scene		x		Error
There is only one Collision group in the scene		x		Error
There is only one Safe Volume (Active Collision) group in the scene		x		Error
Safe Volume (Active Collision) consists of only one box or cylinder		x		Error
The geometry should have no prelit meshes/shaders		x		Error
All meshes have necessary UV set(s) for their shader, e.g.:'default' requires map1 (or channel1), prelit_default requires both map1 and map2 (or channel1 and channel2)		x		Error
All file references should be under the Artsource folder, but may also be stored in the Build folder		x		Error

Texture paths must contain legal characters only	a-z or A-Z 0-9 _ -	x		Error
Maximum number of seats is 3	3	x		Error
Maximum number of lights is 1	1	x		Error
Runtime mesh limit	50	x		Profiler
Active Items – Automatic Validation (Recommendations)				
Recommended maximum triangle limits	4,000		x	Warning

Active Items – Manual Validation (Requirements)	Values	Required	Recommended	Check Method
Active items can only have Lua screens		x		
Active item video screens must be created via a user event (e.g. choosing to play the video)		x		
Active Item video screens cannot play in the background (i.e. they should not be on and always playing video)		x		
If (and only if) you are using the `System.VideoSystemLock()` function: If `System.VideoSystemLock()` returns `false`, do not attempt to play a video. Inform the user that the video cannot be played at that time because either another active item is using the video or because screens that are a part of the scene are playing video.		x		
Active Items – Manual Validation (Recommendations)				
Do not embed active items in scenes.			x	
For collision, all environments and objects have basic polygonal forms representing solid surfaces simply. For example, simple curves are included in the shape of the collision mesh, but details such as sofa legs are not.	Recommended: 50 KB		x	
Furniture may have environment maps, but it is sensible to avoid the use of dynamic reflective surfaces.			x	
Recommended dimensions (Geometry for furniture)	Max 3 (x) x 2.44 (y) x 4 (z) Min 0.1 (x) x 0.1 (y) x 0.1 (z) m		x	

Execution and Render Speeds

Active items should execute and render at a certain speed, when there are the maximum number of characters present and all scene objects are active.

The following table shows the rates for idle and active states:

State	PPU	GPU
Idle state - Deployed in a scene	2 ms/f	N/A*
Active state - User(s) interacting with Active	4 ms/f	N/A*

*There are no GPU rate limits for Active items.

Object Editor Validations

For Japanese Live Page
最新の日本語版ページはこちら 「Object Editorの検証」

This page summarizes the HDK automatic validations and manual validations for the Object Editor. Some validations are required to pass Quality Assurance (QA), other validations are recommended. The validations include memory limits, formatting and any other requirements or recommendations that apply.

Object Editor Validations – Automatic Validations	Values	Required	Recommended	Automatically Validated
Age Rating				
Age Rating must be specified (see Preparing Objects for Packaging)		x		Error
Object Metadata				
A component must contain header data: the object must have one Header node, a localization file and content must be localized for all supported regions		x		Error
All items that are commercially available must have a [#Legal] tag and the legal_tag property in the object's Metadata		x		Fatal Error
If you change the object Metadata, you must save the object again to update the object's odc file		x		Error
If you change the object Metadata, you must save the object again to update the object's catalog entry file		x		Error
Template Objects				
Only objects created from templates can have locked resources		x		Error
Objects created from templates must have the template in the template folder (HDK root/dev/templates)		x		Error
Template objects must have all their associated locked components and resources present		x		Error
Locked components, resources or item types must not be modified		x		Error
Template objects must have a valid ID, matching what is expected by the object's template (including no ID if that is the object template's expectation)		x		Warning
Object Resources				
All files referenced must exist		x		Fatal Error
Resource paths must not contain any illegal characters or spaces	a-z or A-Z 0-9 . ¥ / - : _	x		Fatal Error
A file path must not exceed the maximum number of characters	256	x		Fatal Error
Resource overrides can only be applied to resources of the same type as those being overridden		x		Fatal Error
Scripted Assets and Resources must have the Protected property set correctly. All Lua files must have the Protected property set to True.		x		Fatal Error
All assets that have a .lua or .luac extension must have their Type property set to lua		x		Fatal Error
For arcade games, the Game Spawner component must have its default_screen_texture property set		x		Error
Furniture components must have a valid resource		x		Error
Object Description				

All objects must have a Name and Description in the Object Description file (.odc). Commerce items must also have a Publisher name (company name) and a Product Description (productLongDescription).		x		Error
Object thumbnails (Large, Small and Publisher thumbnails) must be included in the Object Description File (.odc). Small thumbnails are required for furniture, actives, character components, portables (companion objects) . It is recommended that you include all thumbnails (Large, Small and Publisher thumbnails). See Preparing Objects for Packaging.		x		Error
Information about the object must not exceed the maximum number of characters. Character limits are in bytes, which is the UTF8-encoded string size for the field.	Author: Author information is not included in the object packaging, so there is no memory limit. Be reasonable with what you enter. Object display names: 128 bytes Object descriptions: 4,096 bytes Publisher name (Can also appear as 'Company Name'): 128 bytes Entitlement ID: 32 bytes Category ID: 57 bytes Product ID: 49 bytes Element field: 40 bytes Thumbnail Publisher/Small/Large image URL: 256 bytes each Communication ID: 12 characters (Game Launch components only)	x		Fatal Error
Object Components				
Furniture: maximum number of seats	3	x		Fatal Error
All seats in furniture items must have a valid position		x		Error
All lamps must have a Light component		x		Fatal Error
All Light components must contain lights		x		Fatal Error
Furniture with a Game Spawner mini-game component must have an Active Item component		x		Fatal Error
Picture frame Furniture components that support textures being replaced must have their texture_to_replace property set		x		Fatal Error
An object with a Game Spawner component must also have a Furniture component		x		Fatal Error
An object with a RealTime Game component must also have a Lua Environment component		x		Error
An object with a RealTime Game component must also have either an Active Item or a Scene Object component		x		Error
Component Combinations				
The Object Editor checks that object components are not mixed in an invalid way. For details of which components you cannot combine, see Invalid Component Combinations.		x		Fatal Error
Lua Environment Components				
The debug property must be set to False before submitting the object to QA.		x		Warning
Game Launch Components				
The communication_id property of the Game Launch component cannot exceed 12 characters	12	x		Fatal Error
Arcade Game Components				

Arcade Games cannot use HOST memory (the resource memory cannot be set to HOST)		x		Error
Scene Object (Embedded Object) Components				
All numeric_parameters added must have their default_value property set		x		Error
The default_value property set for numeric_parameters added to the Scene object must be less than the max_value property set, e.g.: default_value < max_value	default_value < max_value	x		Error
If max_value and min_value properties are set for numeric_parameters, the max_value must be greater than min_value, and min_value must be less than max_value, e.g.: min_value < max_value	min_value < max_value	x		Error
All enum_parameters must have at least one value set in their values property (e.g. set the list of values which the enum can be set to).		x		Error
All enum_parameters must have a default_value which is contained within its values property		x		Error
Companion Objects (Portable Objects)				
No portable component can be used unless the object is created from a template that contains a portable component		x		Error
MiniGame Components				
The network_session property within the MiniGame component is set to False, but a Network component has been found in the object, only MiniGame components with network_session set to True should have a Network component.		x		Error
The network_session property within the MiniGame component is set to False but the session_size property is not set to 1.		x		Error
RealTime Game Components				
The RealTime Game component's network_definition_file property has not been set		x		Error
The RealTime Game component's max_players property is set to an invalid value, max players should be set to a value between 2 and 16	2 - 16	x		Error
The RealTime Game component's num_tokens property is set to an invalid value. num_tokens must not exceed 256.	< 256	x		Error
Sessionless Mini-game - Manual Validations (Requirements)				
• No network communication (i.e. communication between local player and remote user) is permitted within the mini-game. • No access to Netproperty bags, Network component, OutboundMessage and ReceivedMessage Lua functions. • Sessionless mini-games must be embedded objects.		x		Error

Invalid Component Combinations

The following table lists object components that you cannot combine. If you do, you get a fatal error.

Component	Invalid combination
Active Item components	Do not combine any of the following components with an Active Item component: • Arcade Game component • World Map/Navigator component • Scene Entitlement component • Targetable component
Clothing components	Do not combine any of the following components with a clothing component: • Active Item component • Arcade Game component • Camera component • Scene Object component • Entity component • Event Timer component • Furniture component • Game Launch component • Game Spawner component • Light component • Locomotion component • Locomotion Part component • Lua Environment component • MiniGame component • Network component • OSD component • Pad component • Particles component • Portable component • Renderer component • Repertoire component • Rewards component • Rig component • Screen component • System component • Targetable component
Furniture components	Do not combine a Targetable component with a Furniture component unless an Active Item component exists
MiniGame components	Do not combine any of the following components with a MiniGame component: • Arcade Pame component • Portable component • RealTime Game component
Portable components	Do not combine any of the following components with a Portable component: • Arcade Game component • Scene Object component • Furniture component • MiniGame component
Arcade Game component	Do not combine any of the following components with an Arcade Game component: • Furniture component • Portable component

Furniture and Game Spawner component	Do not combine any of the following components with a Furniture and Game Spawner component unless an Active Item component exists: • Camera component • Event Timer component • Game Launch component • Light component • Locomotion component • Locomotion Part component • Lua Environment component • Network component • OSD component • Pad component • Particles component • Portable component • Renderer component • Repertoire component • Rewards component • Rig component • Screen component • System component

Manual Validations

The Publisher Name (company name) is also required in the .odc for all items that will be available commercially, but it is not automatically validated. You must manually check that the company name has been added to your object.

Scene Editor Validations

> For Japanese Live Page
> 最新の日本語版ページはこちら 「Scene Editorの検証」

This section outlines the automatic validations required or recommended for successful Quality Assurance (QA), and any manual checks you need to carry out.

The following table lists the automatic and manual validations that are required or recommended in the Scene Editor.

Scene Validations – Automatic Validation	Values	Required	Recommended	Automatically Validated
Scenes must contain at least one Spawn Point		x		Fatal Error
Scenes must contain only one scene Script File		x		Warning
Age Rating				
Age Rating must be specified (see Preparing Scenes for Packaging)		x		Error
Environment Maps				
Scenes must contain an Environment Map		x		Error
All Environment Maps must reference assets		x		Fatal Error
Atmosphere Files				
Scenes can contain only one Atmosphere File		x		Fatal Error
All Atmosphere Files must reference assets		x		Fatal Error
Arcade Games and Mini-Games				
Arcade Games and Mini-Games must have a game object set		x		Fatal Error
Arcade Game and Mini-Game Game ID properties must be in a valid format, for example, xxxxxxxx-xxxxxxxx-xxxxxxxx-xxxxxxxx		x		Fatal Error
Arcade Games must have a valid screen_texture set		x		Fatal Error
Seating Areas				

Seating Areas must contain Seats		x		Fatal Error
Particle Effects				
Particle Effects must have a filename		x		Error
Particle Effects cannot be attached as the child of an entity - this feature is unsupported		x		Fatal Error
The Particle Effects efx_filename does not have a value		x		Warning
Screens				
Screen Links File must have a file reference set, e.g. the object must reference a screenlinks .xml file.		x		Error
Screen Links File is required for scenes containing one or more screens (required to set the content that the screen displays)			x	Warning
Screens must have a value for the Screen ID property		x		Error
Screens must have a valid quad shape (else the screen will have an unexpected shape in the client). All screens that are not quads need to be geometry screens.		x		Error
All materials referenced by screens must be available in the scene		x		Error
If a scene contains a screen that plays sound (for example, a video screen), there must be a Sound Zone		x		Warning
Scene Description				
All scenes must have a display name and description in the Scene Description .sdc file (see Preparing Scenes for Packaging)		x		Error
Information about the scene must not exceed the maximum number of characters (character limit in bytes, which is the UTF8-encoded string size for the field): • Scene display names: 128 bytes • Scene descriptions: 4,096 bytes • Publisher name (company name): 128 bytes • Thumbnail Publisher/Small/Large image URL: 200 bytes		x		Fatal Error
Both small and large thumbnails for scenes must be included in the Scene Description .sdc file. Including all thumbnails (large, small and Publisher thumbnails) is recommended. (see Preparing Scenes for Packaging)		x		Error
Sound Banks				
Maximum Sound Bank per scene	1	x		Fatal Error
The Sound Bank must have a file reference set, for example, all Sound Bank objects must reference a sound bank (.bnk) file		x		Fatal Error
Sound asset cannot be a .bank file. These files are unsupported, please use the .bnk file		x		Fatal Error
All sounds referenced in Point Sounds, Ambient Sounds and Sound Zones must be in a sound bank that is referenced in the scene		x		Fatal Error
The Sound Bank must reference a .bnk file		x		Error
A .bank file with the same name as the referenced .bnk file must exist in the same directory		x		Error
Maximum number of sounds (sound streams or sounds in a sound bank) playing in a scene at any one time	18	x		Error
It is recommended that .bnk files are kept below the size specified, but the required limit is that sound must fit within the scene memory budget	2 MB		x	Warning
Sound Zones				
Maximum sound zones per scene	5	x		Fatal Error

Models				
All Models must have valid geometry references set		x		Fatal Error
All scenes must contain a Model file		x		Fatal Error
Collision				
You cannot have more than one Collision file in the scene	1	x		Fatal Error
Embedded Objects				
You cannot have embedded furniture objects in personal spaces or clubhouses		x		Fatal Error
Embedded objects must be valid and have the correct value		x		Fatal Error
Clubhouses				
All Clubhouses must have a Club Bulletin Screen		x		Error
Club Bulletin Screens must have a texture set		x		Error
Club Bulletin Screens must have the appropriate dimensions and screen to message ratio. For example: • Screen width and height • Screen message margin • Screen message width and height	MessageX cannot be smaller than MessageMarginX*2 + 135 MessageY cannot be smaller than MessageMarginY*2 + 70	x		Error
Assets				
There can be no illegal characters or whitespace in resource paths	a-z or A-Z 0-9 . ¥ / - : _	x		Fatal Error
Scripted Assets must have the Protected property set correctly. All Lua files must have the Protected property set to True (the Protected box is checked)		x		Fatal Error
You must not set the Protected property to True (the Protected box is unchecked) for the following assets: • model • texture • particle • collision • anim • skeleton • soundbank • soundstream • repertoire • clothing_actions • clothing_manifest • rig_settings • control_rig • emotes		x		Fatal Error
All assets that have a .lua or .luac extension must have their Script Type property set to lua		x		Fatal Error
All Script objects must have an associated script file		x		Fatal Error
Triggers				
A Change Event Trigger set to CommercePoint must have a destination. The trigger should contain a CommercePoint action with a URL and the Parameter 1 property should contain the address of the Commerce Point file.		x		Fatal Error
A trigger set to ChangeLobby must contain a destination. Its Parameter 1 property should contain the scene name to move to.		x		Fatal Error
A trigger set to ChangeSceneMessage must contain a destination. Its Parameter 1 property should contain the scene name to move to		x		Fatal Error

Triggers must have its Action Name property set.		x		Fatal Error
Entities				
Geometry assets (that are referenced by an entity) must have the Scripted property set to True		x		Error

Manual Validations

The Publisher Name (company name) is also required in the .sdc, but it is not automatically validated. You must manually check that the company name has been added to your scene (see Preparing Scenes for Packaging).

Submission Testing and the CDS

Publishing Content to Online Environments

Once content has been appropriately packaged and tested and is ready for publication, it can be submitted to the CDS (Content Delivery System), and be uploaded into an online environment. In the online environment, the content will pass through all the stages of testing and Quality Assurance (QA), including testing by the company you hire to test your work.

> Note that for final submissions of content to be published live in PS Home (i.e. after content has passed QA), packaged content must not contain any debug functions.

Scene Keys, Company Keys and Online/Offline Testing

Any scene may be developed offline (i.e. local to the development kit and machine), but the moment it needs to be viewed online (i.e. support multiple users who access the content from the Home servers), it requires approval confirmation in the form of a scene key which is a secure ID generated by the PS Home team for you. From HDK 1.2.1 onwards, you also require a unique scene name. We assign this to you at the same time we send back your scene key. By viewing online, you make contact with PS Home's test servers and we check the validation details.

Before you begin uploading your assets through the CDS, you can test your scene and assets. Anyone who shares the same asset files and scenelist.xml file on their local system can view the same scene or object as you if you meet the criteria described below. However, once the submission process commences, you need to have the same hubstartup.txt file to ensure you are all pointing at the same test servers. In the latter case the assets are online, in the former case you are all viewing local files across your network, using the Home servers only for validation.

By applying for a scene key, you gain access to the Home test environment, where you can share your scenes using the latest version of the PS Home platform. Once you have been given this key, anybody using that as well as the same company key can meet up. So you can use it to meet your work colleagues within your scene and test multi-player interaction, such as trying out arcade games, join in mini-games with 2+ players, test video support, share content across an entire worldwide company.

For more information on launching with a scene key, see Launching a Scene in PS Home.

The scene key is obtained by contacting your Regional Support Team. We will send you your scene key which we generate using a secure encryption tool to ensure your key is unique. We will also generate a unique scene name which is the official name you can then use if you are referencing it from other scenes.

> If you have scenes developed prior to HDK 1.2.1, you must apply for a new scene key as previous scene keys will not work.

The company key is generated at the end of the installation process of the HDK. The first developer in your company who installs the HDK will be prompted to generate a company key using the Company Key Manager. Once they have done this, they should distribute this to the others in the company who need it. Your company key will be placed in the clientuid.xml in your HDK install folder. If you wish to access the Company Key Manager again, you can select it from your Home Development Kit list of applications.

> The clientuid.xml file does nothing. It only contains your company key for reference. It is likely that this file will be removed from the HDK in the future.

Until you receive your unique scene name, you need to use temporary placeholders for spawn point redirecting, etc.

Process Overview

1. Develop your scene offline using the HDK.
2. When ready to test scene online, apply for a scene name and scene key.

3. With the scene name and scene key, upload your scene into your sandbox on CDS.
4. Test your content online.
5. Once ready for QA and publishing in the client, submit a request for your scene to go to QA.

Testing Content Online

You have two options for testing your content online:

- You can launch the Home .self using the Target Manager and the hubstartup.txt file created after uploading to CDS. For more information, see Testing Content in Online Mode.
- You can launch the scene online through the Scene Editor. See Launching a Scene in PS Home.

Submitting to QA

To submit your content to your regional QA, use the My Submissions link on the CDS website. For more information on using the CDS, see https://cds.home.scedev.net.

More about the Scene Key and Scene Name

When packaging a scene for submission to the CDS in the Scene Editor, you will be asked for a display name and description. These are entirely separate from the above process, and are used inside PS Home itself to display information to those using it on the Navigator.

The scene key and scene name are produced by us at HTS Support after asking developers for the suggested name and a description. We produce a unique scene name and associated scene key that are linked to each other. So you must provide both when you use the CDS.

The scene name is only used in a very few places:

1. The Scene List.
2. The function `LocalPlayer.Relocate`.
3. A Change Scene trigger placed in your scene.

So it's not visible to the end users of PS Home and is not used as the actual public display names of scenes.

Therefore, when we ask for a suggested scene name and description, we're hoping to return a unique scene name almost identical to the one the developer has suggested. We will access up to 20 characters, without spaces, and then add the randomized 8 digits to the end so it's always unique.

You must keep your scene name the same. You can submit a changed scene through the CDS without renaming it. If you do rename it, you have created a new scene and must receive a new scene key to associate with it.

We ask that you suggest a scene name that you will be able to recognize and do not use generic terms such as "test" and "home". For example:

Item	Description
Suggested Scene Name	MyCompLaunchpadApt
Description	This is our apartment with a launchpad special feature.
*Scene Key	1823966b-13cc-5573-e398-c504319ff85e
*Final scene name	MyCompLaunchpadApt_1234_HHY7

*These items are provided by your SCE Regional Manager.

As you can see, we do not need regional or company information.

We could standardize this, but we do not feel that there is a requirement to do so, due to the limited use of the scene name.

We do request that:

- The suggested scene name must not be a generic testing name such as "Test Scene". It should describe the scene in some basic way.
- It cannot have spaces in the name, i.e. "MyCompSpace" is correct, "My Comp Space" is incorrect.
- It must be 20 characters or fewer.
- It must not be changed just because the scene has been changed.
- The scene description should be not more than a short sentence describing the scene in simple terms (for example, "A desert island personal space with a hotel complex").

Submission FAQs

Where do I switch between offline and online builds?

You toggle between offline and online builds through the Scene Editor. The Scene Editor offers two online modes:

- Home: A personal apartment with options to place furniture and customize avatars and view your character clothing and accessories.
- PublicLobby: a shared default space that others with the correct access information can join.

Do I need to go online to test my Lua, furniture or clothing objects, and do I need a Unique ID for them?

No and Yes. When you export your object using Maya, a Home model file (.mdl) is created containing the geometry and a matching Havok file (.hkx) is created containing the collision data. If you then use the Object Editor to view your object, you will see that a unique Object ID is created for you by the HDK.

You do not need to go online to test your objects. From HDK 1.2 onwards, there are profiling tools available to help with the optimization of scenes and objects. Profiling determines how resource-intensive the scenes and objects are. For more information on the static object profiling of furniture and clothing, interactive object profiling of Lua scripted objects and scene profiling, see Profiling in the Client. Also see Lua Scripting and Scene Editor.

Do I need to always have the latest HDK for my scenes and objects to work?

Content that was produced and packaged using a previous version of the HDK should always be compatible with future releases, though it is always worth testing your content with new PS Home clients as they become available to ensure there are no unforeseen compatibility issues.

Submitting scenes and objects to PS Home

For submitting the assets you wish to make available to PS Home, we provide a web-browser-based, secure Content Delivery System (CDS). The CDS is the online interface for managing Quality Assurance (QA) and publication for PS Home. It automates many of the processes of validating and requesting publication for assets (environments and objects) developed for PS Home (including a package size limit of 100 MB). We integrate with Open ID to authorize access to developers using their DevNet account details.

The CDS validates assets submitted online, provides online QA environments with access restrictions for teams to work on, manages the communications between development teams and the QA teams and provides the interface for scheduling asset publication.

The CDS was upgraded to version 1.1 in November 2008 to support additional features and security enhancements. This version offers a new test area for developers to upload to. So you need to apply for new scene keys and re-upload all previously uploaded online assets.

How do I find the CDS and what do I need ready to access it?

You must have a PlayStation®3 Developer Network (DevNet) account and a company key

For scene submission, you need a scene key for each scene (which you can request from https://ps3.scedev.net or https://home.scedev.net), as well as your unique scene name.

For object submission, you only need an object ID (automatically created by the Object Editor).

The file size of the content package you want to upload should be no bigger than 100 MB (the CDS generates an error if your package exceeds this, as the upload is likely to fail if this size is exceeded. To submit a package larger than the maximum specified, contact SCE in advance.).

Using TTY for Debugging

This section covers how to use the TTY channels in Target Manager to debug your content.

Debugging Basics

Whenever you encounter an unexpected error or behavior in the client, check the console and TTY output for information about the error. Most errors in the TTY output are highlighted in red. Out Of Memory errors are clearly marked, for example:

```
**********************************************************************
***  Memory Error: 20
***  Maximum pool or pager size reached, address 0x00000000, size 0x00000033.
***  Pool: HomeString
***  Core   Largest:       16   Free:    76664  Total:   786432
***
***  Requested: 51
**********************************************************************
```

You can improve your chances of finding errors by increasing your TTY window's buffer size. To do this:

1. Right-click the TTY Channel panel that you want to effect and select Properties.
   The TTY Channel Properties dialog is displayed.

2. In Buffer Settings, change the Size (KB) value to 1024 or higher, and click OK.
   This allows more output to be printed before the old output is over-written.

If you can't pinpoint the problem, re-run your scene and attempt to reproduce the issue, but just before you reach the point at which the error occurs right-click in the TTY window and select Clear. This will clear the window and narrow down where the error might be, allowing you to find it more easily. If you are still unable to find the cause, then open a DevNet Support Ticket and include the TTY output.

TTY Channels

Target Manager has 6 default channels: All, TM, KERNEL, PPU, PPU (STDERR), SPU. There are also User TTY channels which contain all the Home Client specific output.

The Client channels are defined as follows:

Channel	System
USER 1	Storage Trunk
USER 2	Http
USER 3	Network
USER 4	Rewards
USER 5	Audio
USER 6	Script
USER 7	Video
USER 8	Memory
USER 9	Console
USER 10	Messages
USER 11	Character/UI
USER 12	Profile
USER 13	Debug

Renaming the TTY Channels

To rename the TTY channels:

1. Right-click the tab you require and select Properties.
   The TTY Channel Properties dialog opens.
2. Type the new name in the Name field and click OK.

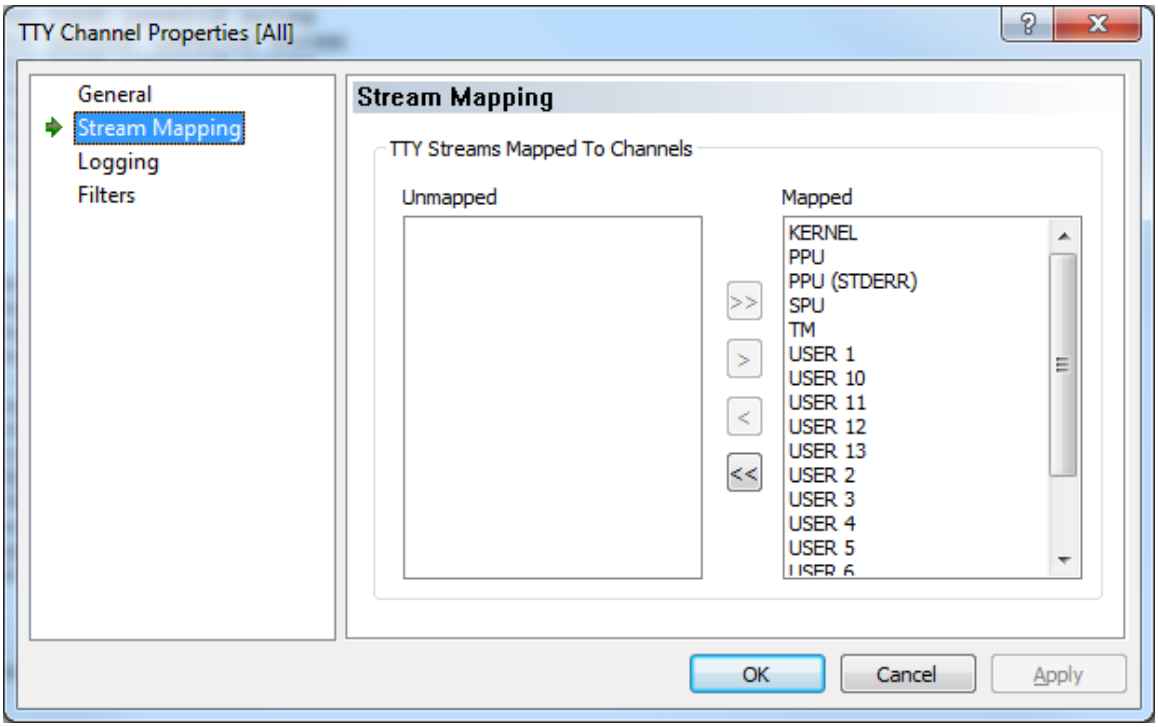
The Script panel (User 6) is very useful as it shows only the print statements from your scripts and your Lua errors.

Filtering the Content of the All Channel

If you know what the User channels are, you can filter the All tab so that it excludes channels you do not require.

To filter the content of the All Channel:

1. Right-click All and select Properties.
   The TTY Channel Properties dialog opens.



2. Select the Stream Mapping tab
3. Move any channels that you don't want to see from Mapped to Unmapped so that the output from that channel is not displayed in the All channel.
4. Click OK.

TTY Tips and Tricks

Echo TTY Input to Screen

On the TTY Channel Properties dialog, the General tab has an Echo TTY input to screen option. This option is particularly useful on the Channel User 9 Console panel as it allows you to type console commands directly into the window. So if you want add objects into your inventory using inv adduserobj, you can simply copy and paste it into the User 9 window.

Color Code your Lua Output

You can print to the console from your Lua scripts in different colors using ASCII encoding. For example, the print statement below prints your statement in white on a black background:

```
print('\27\[37;40mWhite text on a black background' )
```

The color codes are as follows:

Foreground colour

¥27[30m Black
¥27[31m Red
¥27[32m Green
¥27[33m Yellow
¥27[34m Blue
¥27[35m Magenta
¥27[36m Cyan
¥27[37m White

Background colour

¥27[40m Black
¥27[41m Red
¥27[42m Green
¥27[43m Yellow
¥27[44m Blue

¥27[45m Magenta
¥27[46m Cyan
¥27[47m White

Launching and Testing Content

This section covers the process for launching and testing scenes and objects using the tools, the conditions the scene or object should be operating in to test, e.g. profiling should take place while all content that could be active simultaneously is active - running scripts and animations and calling embedded and dynamically loaded content, etc.

This section also gives general tips and console commands to help test and optimize the content.

For more information on profiling methods, see Profiling in the Client.

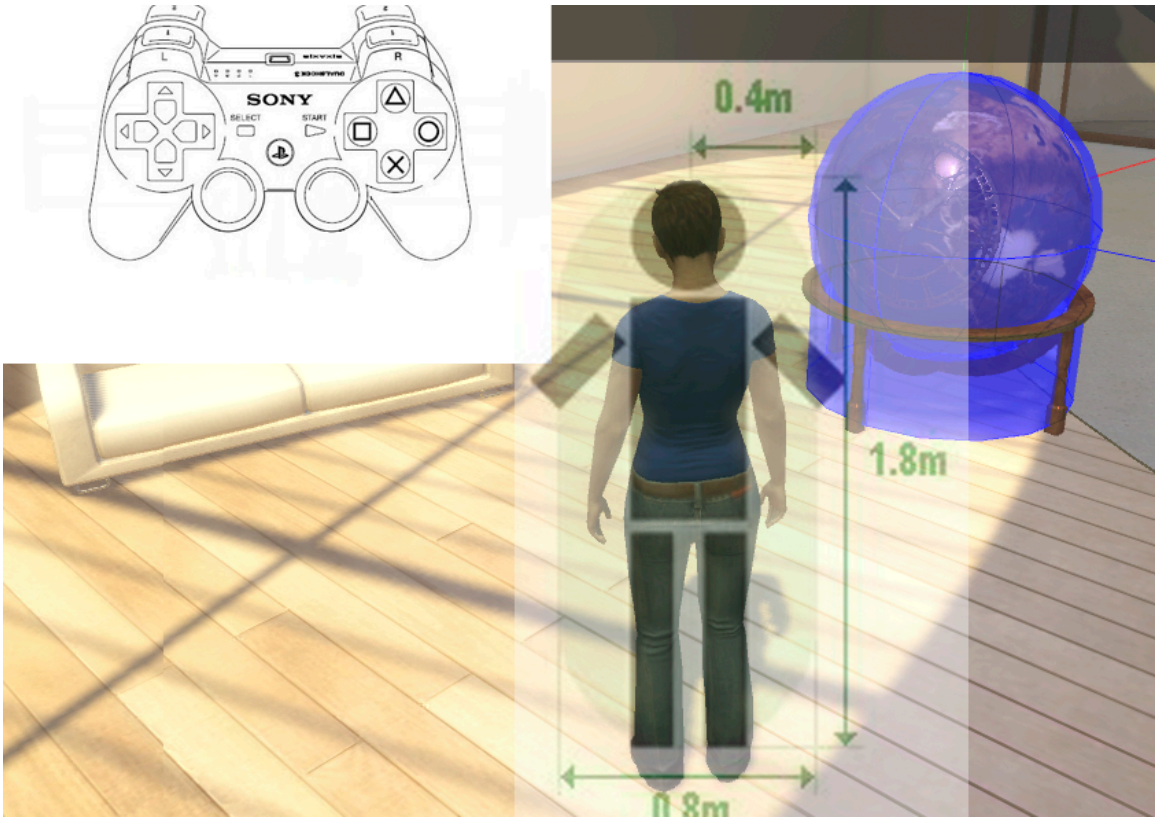
The following table summarizes the various options for launching content in PS Home:

Task	How
Launch scene from Maya	See Previewing Your Work.
Launch scene from the Scene Editor	See Launching a Scene in PS Home.
Launch objects from the Object Editor	See Profiling to use the Static Object Profiler and Preparing Objects for Packaging.
Launch scenes and objects online and offline from the Target Manager	See Profiling Scenes, Launching a Scene in PS Home and Object Editor. There are also several command line parameters that you can use to load PS Home in certain modes, as well as viewers that allow you to browse, view and manipulate objects in PS Home from your HDD. This is designed to make it easier to profile and quality assure your content and to help you check that everything works as expected in the runtime. You can load scenes and objects into PS Home using command line parameters via the hubstartup.txt file before you launch PS Home, or in PS Home itself using console commands in the debug console. See Objects with Network Components, and Lua Scripting.
Profile scenes and objects with Debug Console commands	You can profile scenes and objects in PS Home using console commands in the debug console (or in the User 9 section of the Target Manager). For a full list of these console commands, see Profiling Lua Scripts and Testing Content in Online Mode. You can also profile scenes and objects using the Profile GUI that can be enabled with a debug console command `ProfileGui.Enable 1`. For more information, see Profiling in the Client.
Add objects to the inventory in PS Home	See Managing the User Inventory. For Game Launch objects and Home Rewards, see the relevant sections in Game Launch Objects and PS Home Rewards.

Systems and Services

This section explains the PS Home systems, PSN systems, and communications between them.

See also System Information for details on scripting and System functionality.



Collision in PS Home

What has Collision in PS Home?

The following items have collision:

- Environments
- Furniture
- Physics objects: these are any object you have created with collision - for example, an active item or a model called by a Mini-game with collision properties specified

The following items do not have collision:

- Character components
- Animations, including:
    - Animated models
    - Particle effects and sky design effects
    - Avatar animations

For more information on collision and targeting, see Targeting System.

For Information on creating collision in Maya, see Collision Tools

What Interacts with Collision?

It is not just the avatar that interacts with the collision in the scene. Collision must make sure that everything that exists (or could exist) in the scene is restricted to the scene boundaries. This rule applies not just avatars, but to the camera and all objects, including furniture, active items, and objects called in by scripts.

Exporting collision data using Havok

PS Home Art Tools use Havok™ to export collision data directly from Maya, to world-ready format. When an export takes place, these meshes are automatically converted to collision nodes (and back again). This means that the collision is made up of normal meshes.

The collision mesh is, for the most part, a simplified version of the art work. For example, this is the collision mesh for an example environment:

Unable to render embedded object: File (03000045.png) not found.

However, it is much better to use as much primitive collision as possible.

Various places will have simple approximations of the original shape (for instance, collision for steps should be done as a slope).

Unable to render embedded object: File (0200003D.jpg) not found.

Camera and Collision

The camera obeys the collision mesh. So in an environment, the collision is normally raised in areas where the camera should not be moving outside of that specified area. Notice the raised collision on the environment sides to prevent the camera moving through the glass.

Unable to render embedded object: File (03000045cropped.png) not found.

Dimensions and Collision

There are certain standard dimensions in PS Home that you must conform to. If you ignore these dimensions, the way the avatars interact with collision looks unrealistic or might prove problematic.

Collision for steps are developed as slopes. Avatars walking up and down steps is an approximation, and there are predetermined step sizes that should be followed to ensure the default step-climbing animation in avatars looks realistic.

All heights and dimensions that the character or camera can collide with must conform with current PS Home scale conventions. For more detail on these please refer to Environments and Scale and Dimensions in the General Design Guidelines section.

How to make Geometry into a Collision Mesh

Place the geometry into your Collision group, where it will be automatically templated.

Unable to render embedded object: File (03000047.png) not found.

Important Information

There must be at least one collision mesh (in the Collision subgroup). You can use the following:

- Primitive meshes
- Arbitrary convex meshes - but keep them simple
- Concave meshes - but avoid if possible

Collision meshes do not need any ATG Material applied to them, nor do they require any particular UV mapping.

Primitive meshes are recommended for collision. If you do not specify a collision attribute, the collision object is exported as a convex hull or mesh, depending on the geometry of the collision object. The collision object only exports as a primitive object when it has a collision attribute.

After export, if you see an error such as the one shown below, you can ignore it. It is only a side-effect of the underlying collision exporting.

Unable to render embedded object: File (03000048.png) not found.

Seating and Collision

Ensure that avatars can easily access the seats, and can sit down and stand up without collision restrictions hampering their position or animation fluidity.

Remember that there are 3 seating heights - see Dimensions.

When the local avatar sits down on a seat, collision is turned off. This is so that the avatar can successfully reposition itself to where the seat is without being pushed around by the collision of the seat, or surrounding collision geometry.

This means however, that when the avatar stands up from the seat, and collision is turned back on, it needs to do a few checks to make sure it can stand up properly without the collision pushing it into inappropriate positions.

This could make the movement or animations look clunky, but also may result in the avatar being forced on top of the seat when standing, or more importantly, into collision areas that it can't escape from.

Standing Up

When standing up, it is important to make sure the collision on the seat is recognized, so that the avatar can be placed far enough out from the seat, and that there is sufficient room in front of the seat to stand up into.

- avatar's seated position

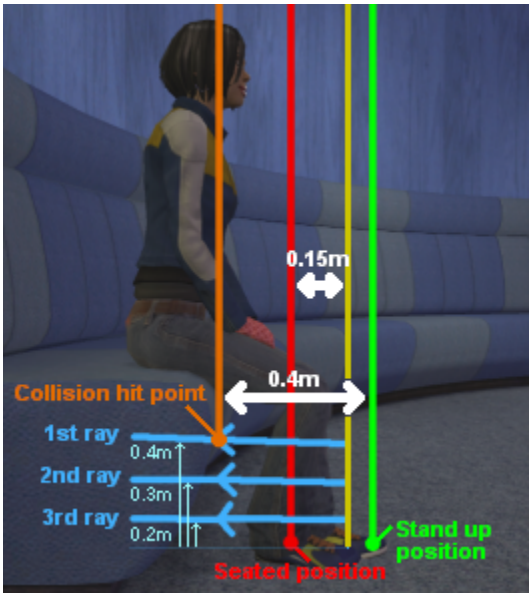
- collision hit point

- collision ray origin

- avatar's standing position

- rays fired to detect seat collision

To do this, collision rays are fired from 0.15 m (the yellow line) in front of the avatar's seated position (which is at the avatar's feet, shown as a red line).



The rays are fired in the order shown (in blue). When any of them hit collision, the seat is registered as found (if the first ray hits, the other two need not be fired, and if the second ray hits, the third need not be fired. In this picture, the first ray hits the seat). When the seat is found, the position of the collision is taken (shown in orange).

The new stand up position is then calculated by taking this collision point and moving it 0.4 m, which is the radius of the local avatar's collision capsule (see Avatar Collision) out from the seat, and aligning the vertical position to stand the avatar on the floor (shown in green).

It is therefore important to always design seating areas and chairs in a scene with these measurements in mind.

If the system determines that it either cannot find the seat, or that there is insufficient room to stand up into, it takes the position the avatar was at when the user selected to sit on the seat, and snaps the avatar to that position. It also snaps the animation to standing. This is a safety net if the seat is positioned awkwardly, so if this happens, consider how well designed your seat is. It may be that snapping the position and animation is preferred, which is valid, but always make sure this is intentional.

It's vitally important to check that the avatar sits down and stands up from every seat correctly. Check that it looks correct visually (in terms of animations and movement), and check that the avatar isn't placed in an awkward position in terms of collision after standing up from the seat. Check they're not trapped, or positioned outside of the intended collision area.

PS Home Physics

PS Home uses the Havok™ System for the physics. Havok™ is an industry standard rigid body dynamics package that simulates rigid body dynamics in realtime. Simulations a wide range of physical phenomena, including:

- Forces and torques
- Collision detection and response
- Friction and restitution

Working with the physics system in PS Home relies on some existing knowledge of physics engines. If you are unfamiliar with physics for games, there are many sources where you can learn more about how rigid body physics works and the forces, torques and friction involved.

The Havok™ System gives you significant flexibility in development. However, customizing the physics to work around present limitations or behavior is not recommended. It is better that you rely on the native collision system and adhere closely to the restrictions and recommendations, to ensure system stability, backward compatibility and inter-operability with various conditions. We recommend that you keep your implementations simple; it is easier to work with the native physics in PS Home.

For guidelines on scripting collision, see Scripting and Debugging Collision.

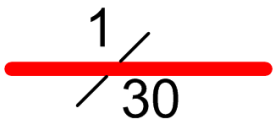
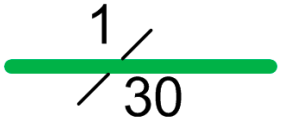
Client Update Rates

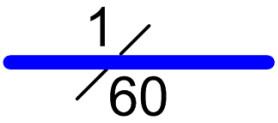
Client Update Rates

The PS Home client runs three systems in the background:

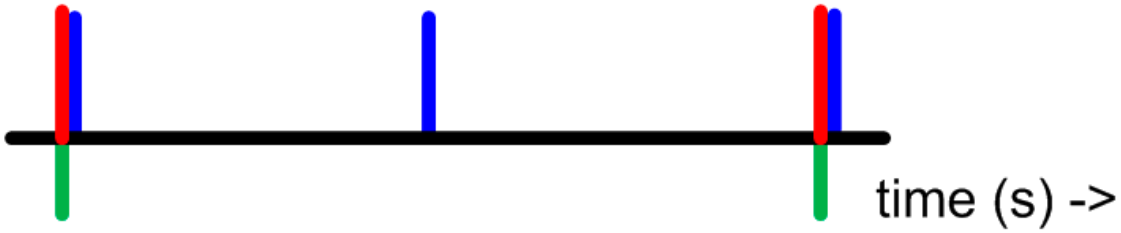
- Normal Game Loop
- Havok system
- Game Render system

These systems each have their own update rates. The Normal Game and Game Render update at the same frequency. The Havok system updates more frequently, but it can become out of sync with the other two system, depending on network and performance conditions.

	Normal Game (Script)	The Lua script and script logic updates. Update Rate: Under ideal conditions, 1/30 ms. The rate is affected by game performace; if the game lags updates will be less frequent.
	Game Render	Rigid body updates (transformation of rigid bodies). Update Rate: Same as Normal Game Updates

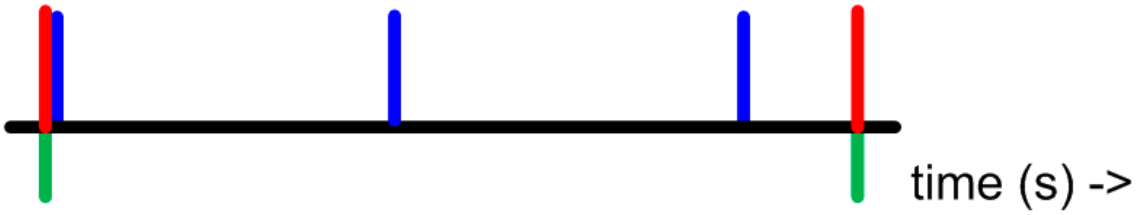
	Havok	Havok engine updates Update Rate: 1/60 ms
	Force	This is the force applied to an entity.

Normal Update Rate



Under good network and performance conditions, there are two Havok updates for every Script and Render update (i.e. 2 blue lines for every 1 red and green line).

Lagging Update Rate



Under poor network and performance conditions, the Script and Render updates can lag. Havok updates are unaffected by performance conditions, which means that there can be more than 2 Havok updates for every Script and Render update.

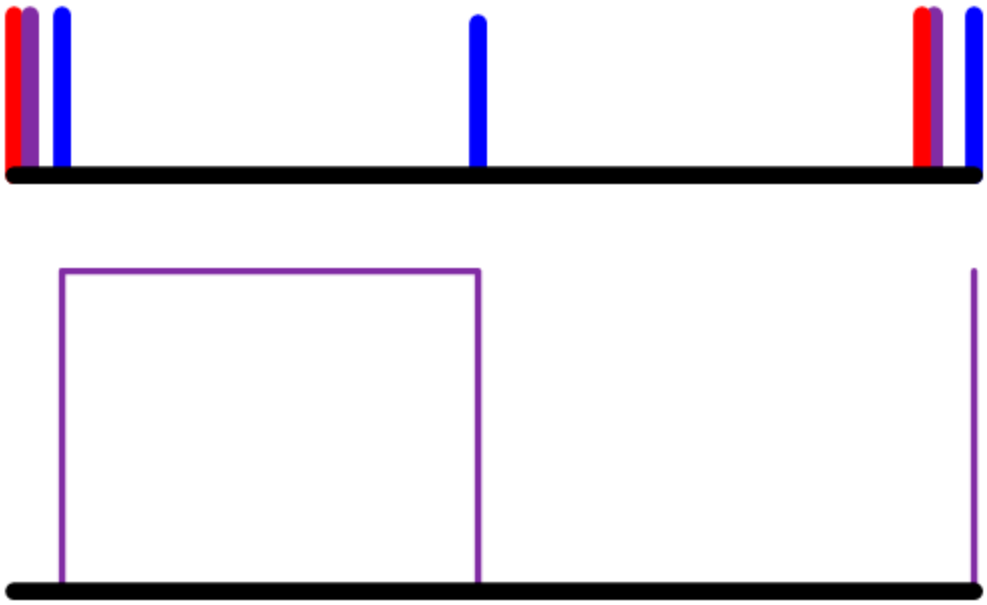
Issues with Differing Update Rates

Two issues can arise from the out-of-sync Havok update rates:

- Pulse Forces: Forces can only be applied at a Script update, and they only last until the next Havok update.
- Out of Date Physics Updates: Rigid body transformations (green line) use the most recent Havok updates (blue line). There can be a gap between a Havok update and the rigid body transformation, meaning physics transformations can be up to 1/60th a second out-of-date.

Pulse Forces

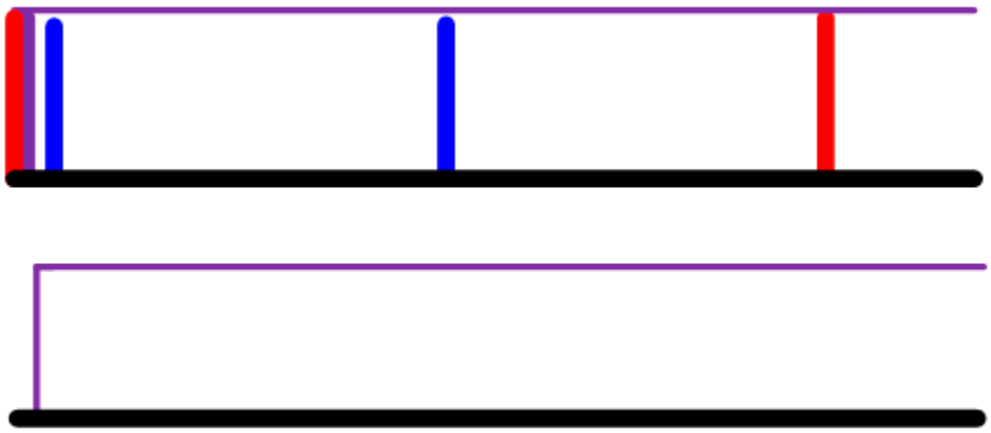
The Pulse Force Problem



1. A script tells the client to apply a force (purple line) to a rigid body.
2. The force is applied at the subsequent Havok update, and lasts only until the next Havok update.
3. Because there are at least 2 Havok updates per 1 Script update, no force is applied at any of the subsequent Havok updates.

The result is that rigid bodies' movements pulse.

The Solution



Use the function `Entity.CreateForceAction()`.

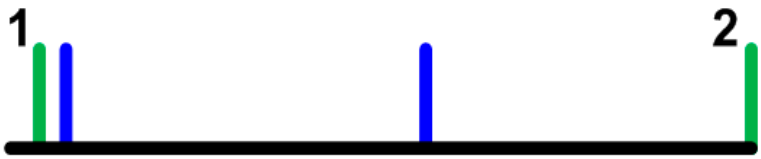
This function creates a force that is applied to an rigid body (entity) until the script destroys the action force.

Related functions:

- `Entity.CreatePointForceAction()`
- `Entity.CreateTorqueAction()()`.

Out of Date Physics Updates

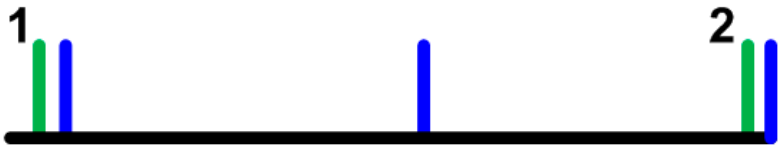
The Out of Date Problem



1. A rigid body is rendered (at step 1).
2. There are at least two Havok updates between Game Render updates.
3. Only the most recent Havok update (the second blue line) is applied to the rigid body at the next Game Render update.

The result is that the physics transform is about 1/60th of a second out-of-date. (The gap between the second blue line and step 2 is about 1/60th of a second.)

The Solution



Use the function `Entity.EnableInterpolateMode()`.

This function does two things:

1. The client uses the two most recent Havok updates to calculate the Havok update one step ahead of time. So by the time the client reaches step 2, it already knows what the next Havok update will be.
2. Using the most the recent and the future Havok update, the client interpolates the transform.

The client applies the interpolated Havok transform at the next Game Render update (step 2). Physics transformations will not always be linear, which is what this function assumes, but the interpolated transform is more accurate than the 1/60th out-of-date transform.

Physics Functions

The HDK gives you, the developer, control over the Havok physics engine. The physics functions allow you to apply forces torque and impulses to entities, set gravity and mass, apply linear damping, and even set whether an entity is affected by the Havok physics engine (`Entity.SetDynamicsActive()`).

```
Entity.ApplyAngularImpulse
Entity.ApplyImpulse
Entity.CreateForceAction
Entity.CreateForceAction
Entity.CreatePointForceAction
Entity.CreateTorqueAction
Entity.DestroyAction
Entity.EnableAction
Entity.EnableDyanmics
Entity.EnableDynamicsAutoDeactivation
Entity.EnableInterpolatedMode
Entity.EnableKeepUpright
Entity.GetAngularDamping
Entity.GetAngularVelocity
Entity.GetCenterOfMassLocal
Entity.GetCenterOfMassWorld
Entity.GetContactRelVelocity
Entity.GetGravity
Entity.GetLinearDamping
Entity.GetLinearVelocity
Entity.GetMass
Entity.GetPointVelocity
Entity.IsActionEnabled
```

```
Entity.IsDynamicsActive
Entity.IsDynamicsAutoDeactivationEnabled
Entity.IsInterpolatedModeEnabled
Entity.IsKeepUprightEnabled
Entity.LookAtWorld
Entity.SetActionForce
Entity.SetActionPosition
Entity.SetActionTorque
Entity.SetAngularDamping
Entity.SetAngulaVelocity
Entity.SetCenterOfMassLocal
Entity.SetDynamicsActive
Entity.SetFrictionCoef
Entity.SetGravity
Entity.SetLinearDamping
Entity.SetLinearVelocity
Entity.SetMass
Entity.SetRestitutionCoef
```

Chat System in PS Home

The chat system in PS Home has two methods of communication:

- Voice chat: (headphones and microphone) A user can be in one voice channel at a time. They can listen and broadcast to this channel.
- Text chat: (keyboard) A user can receive text chat from all channels but must select a channel to broadcast to.

> Users can have only one Personal Voice Chat running at a time. However, they can switch voice channel from Personal Voice Chat to Group or Club voice without having to end the Personal Voice Chat session.

PS Home uses a number of different communication channels. Each channel supports both text and voice, apart from the Local channel in public spaces, which has voice disabled. See Channels.

Users can switch voice channel and text channel independently, so they can be in different channels for voice and text at the same time. For example, they can broadcast text on the Local channel while speaking to their Club on voice comms.

> If a user is in a group and a game, they can switch between the two voice channels.

Chat Log Tabs

The chat log has the following tabs:

- A Text tab, and a Voice tab. The separate tabs allow users to manage where they are broadcasting the different types of chat to, and to keep track of all the channels that are available.
- An Alerts tab, which displays useful information and messages to the user, such as rewards they have been given, notifications of invites to chat or to join a club, and suchlike.

Different colors make it easy to identify different channels.

Chat Log Modes

The chat log has three modes:

Mode	Appearance	User Ability
Maximized	Fully open	Read and Write
Minimized	Partially open	Read
Closed	Cannot be seen	None

To check a user's chat log mode, use `System.GetChatLogMode()`.

Changing a User's Chat Log Mode

You can change a user's chat log mode using `System.SetChatLogMode()`. Changing the mode can be useful, for example, in a game that has a cut-scene. When the cut scene begins, but before the cut-scene users can chat, the chat log mode is set to 'closed.'

When the cut-scene finishes, the script changes the chat log mode to 'minimized.'

> This function is intended to free-up the lower left-hand corner to improve the look and feel of a game. It is not intended to prevent users from chatting. Attempts to do so can easily be circumvented by users, by sending PSN messages, for example. Do not build a game that relies on users not chatting.

Controls

You can use the following controls:

Control	Action
[L3] or D-pad	Access the chat log channels (text and voice chat).
[TRIANGLE]	Open the VKB to display the message to the default channel, usually public.
Menu Screen	For more options and settings, such as: • Use of pop-up color in chat Log entries • Text font and size in Chat Log • Automatic fade timing • Prefix display next to each line, including channel identifier and time stamp

Finding People

When the chat log is open, users can display a list of everyone who is in their current channel. They can see where the other users are in PS Home, and then interact with them.

To obtain further information and options on each chat participant, the user can select the Options Menu item from the chat log.

Channels

The following channels are available to users, each of which has Text Chat and Voice Chat. All valid channels are available to use at any time, and the user is able to Text Chat in one channel, and Voice chat in another if so desired:

- Local: Channel containing everyone near the user (persistent).
- Group: Channel containing everyone in the same Group from anywhere in PS Home (channel exists while the Group exists).
- Club: Channel containing everyone in the Club from anywhere in Home. This channel is persistent. The user must select one Club to receive and broadcast to.
- Activity(Game Channel - Mini-Games or Realtime Games): Channel for every user playing the specific game (channel exists while the game session exists).
- Personal: Channel for sending and receiving communications with one specified person, which is not visible/audible to anyone else.

Local Channel

Everyone in the same scene as the user is available on the Local Channel, if within range (8 meters).

> Local Voice Chat is disabled in Public Spaces.

Group Channel

Everyone who is a member of your Group is on your Group channel. You can chat with all other members who are joined to that Group.

You can access this channel from anywhere (any instance of any type of scene), and no matter what you're doing (e.g. while joined to a Mini-game).

Club Channel

Like the Group channel, everyone who is a member of your Club is on your Club channel. You can chat with all other members who are joined to that Club.

You can access this channel from anywhere (any instance of any type of scene), and no matter what you're doing (e.g. while joined to a Mini-game).

If you belong to more than one Club, when you open the Club channel it automatically opens the channel of the default Club. You

can change the Club channel to a different Club (or set a different default) from the Chat Log options menu.

Activity Channel (Game Channel – Mini-games or Realtime Game)

Users can:

- Chat to anyone in their Mini-game or Realtime game.
- Or chat to just their Team. (Teams are optionally implemented by the game's Lua script, so their availability and construction are on a game to game basis).

Everyone who has joined the same instance of the Mini-game or Realtime game is on that user's Activity channel.

The Activity channel defaults to Off, but can be activated using the following Lua function:

`ActivityChannel.Initialize()`: Toggle Activity Chat Channel on/off.

Customizing the Activity Channel

The activity channel allows the game (Mini-game or Realtime game) it is associated with to customize its behavior. For example, the game can:

- Add and dynamically update a suffix on the name of the channel, using `ActivityChannel.SetTitle`.
- Set permissions for who each user in the channel can communicate with and via which method (Text or Voice), using `ActivityChannel.SetCommunicationPermissions()`. This can be used to create 'Teams'.
- Limit the results to display only those users who have some relationship with one another, when users choose to "List Users" or view who is in a voice channel on the voice tab.

Channel Suffix

The Activity channel has a custom suffix on its name that can be set by the game and dynamically updated for each user while they remain in the channel. The custom suffix is displayed in the tab header and Chat Log option menu. It takes the following form:

`Activity (<suffix>)`

For example

```
Activity (Chess)
Activity (Pool)
Activity (Pool - Team A)
```

Each user participating in a given game can potentially see a different suffix for the channel depending on what the game is trying to communicate to them.

Permissions (Creating Teams)

> ⚠ In Mini-games and Realtime games, you can determine who is allowed to chat via the Activity channel, using the function `ActivityChannel.SetCommunicationPermissions()`. However, users can set their own Preferences for chat, which override the system settings.

The game can specify for each user in the Activity channel which other users in the channel they can speak to and hear (for example, both text and voice). This allows the game complete flexibility for how it relates the users in the channel to each other. The table of relationships between users can be updated dynamically by the game at any time.

Example

For example

There are three users in the activity channel: User1, User2 and User3. For each user in the channel, there is a table that contains information on their chat relationship with everyone else:

User1's table:

	Send – text	Send – voice	Receive – text	Receive – voice
User2	Yes	Yes	Yes	Yes
User3	No	No	No	No

User2's table:

	Send – text	Send – voice	Receive – text	Receive – voice
User1	Yes	Yes	Yes	Yes
User3	No	No	No	No

User3's table:

	Send – text	Send – voice	Receive – text	Receive – voice
User1	No	No	No	No
User2	No	No	No	No

In this example, User1 and User2 are on the same "team" and can communicate freely. User3 is on a different "team" and cannot communicate with the others at all.

When the activity channel has its users listed either through the Text tab "List Users" option or through the list on the Voice tab, the names displayed are restricted to those that have some relationship with the local user. The game can dynamically update the relationships and thus change who is displayed on the appropriate channel lists.

Example continued...

Extending the previous example, each user sees the following:

User1's list

User1 has a relationship with User2, so can see the following people in the activity channel:

- User1
- User2

User2's list

User2 is on the same team as User1, so sees the same list:

- User1
- User2

User3's list

User3 has no relationship with the other two users in the channel, so does not see any other users in there:

- User3

Personal Channel

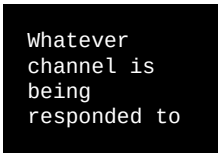
With the Personal Channel, the Text and Voice components work a bit differently from how they do in other channels.

- Personal Text Chat can be used to communicate with a chosen player, with a one-off message. The Users then return to the channel they were previously in.
- Personal Voice Chat is a bit like making a phone call. A user has to request a Personal chat from another chosen user, and that user has to accept the request.

Users can use the Personal channel with anyone from anywhere, not just friends or people in their vicinity, as long as they are in Home.

Slash Commands and String Formats for Channels

Channel	Color	Slash commands	Message String format	Example
Local		`l`	`l [string]`	/l hey everyone! check out my new sneakers
		`/local`	`/local [string]`	/l hey everyone! check out my new sneakers

Group		/g	/g [string]	/g why don't we go check out the new FPS game?
		/group	/group [string]	/group why don't we go check out the new FPS game?
Club		/c	/c [string]	/c wow did anyone see that new high score?
		/club	/club [string]	/club wow did anyone see that new high score?
Game/Activity		/ga	/ga [string]	/ga it's your move
		/game	/game [string]	/game it's your move
		/a	/a [string]	/a it's your move
		/activity	/activity [string]	/activity it's your move
Personal		/p	/p <online ID> [string]	/p JaneDoe123 hey alan, i like your new dress lol
		/pm	/pm <online ID> [string]	/pm JaneDoe123 hey alan, i like your new dress lol
		/personal	/personal <online ID> [string]	/personal JaneDoe123 hey alan, i like your new dress lol
		/personalmessage	/personalmessage <online ID> [string]	/personalmessage JaneDoe123 hey alan, i like your new dress lol
Reply		/r	/r [string]	/r i hope you sent this as a private message or the whole space just heard me say you don't suck at racing
		/reply	/reply [string]	/reply i hope you sent this as a private message or the whole space just heard me say you don't suck at racing

Network Platform

PlayStation®Home provides a secure, encrypted means of communicating with the PlayStation®Network, so that you can access users' rankings, trophies and game title data, and authenticate users' access to your external server. These features are typical in PlayStation®3 games, and now in PlayStation®Home through a Unified Asynchronous Command Lua Interface (AsyncCommand).

The interface is designed to be easy to understand and use, with a standard format for using NP functions. The NP functions enable you to perform a number of actions. For example, you can host data on external servers, access PSN, create user-specific persistence within your space, and link your PlayStation®3 games to your PS Home spaces and Mini-games.

Service Prerequisites

PlayStation®3 developer - To use NP Services, you need to be a PlayStation®3 developer with access to https://ps3.scedev.net/. If you are not a PlayStation®3 developer, speak to your Home Account Manager for further information.

Communication ID, Service ID, and passphrase (NP Ticketing does not need a passphrase) - These are available only on request for PlayStation®3 developers, through https://ps3.scedev.net/. If you are using pre-existing IDs, they must be updated for use with PS Home.

Each of the NP services (such as NP Ticketing, NP TUS) have their own prerequisites. Speak to your Home Account Manager for more information.

Connecting the Home client to PSN

For information on connecting the Home client to the PSN/NP (PlayStation®Network) servers, see Configuring the Network.

NP Lua API

PS Home's Network Platform (NP) functionality comes from the PlayStation®3 Network Platform SDK. We provide only the means to use existing NP functionality. For more information on NP, see https://ps3.scedev.net/.

To use NP Services, you must be a PlayStation®3 developer with access to https://ps3.scedev.net/. If you are not a PlayStation®3 developer, speak to your Home Account Manager for further information.



Many of the NP Service functions require the use of the `BigInt` API where the arguments require `BigInt` values. See the Lua API Reference for more details.



NP Ranking and NP Title User Storage can be PSN server heavy. Although NP Ticketing relies on your own server, it too can be taxing on PSN servers when checking tickets. Use it sparingly.

There are currently five command types for `AsyncCommand`:

- NP Entitlements
- NP Ranking
- NP Ticketing
- NP Title User Storage
- NP Trophy

NP Entitlements

NP Entitlementsis a read/write command type. It provides an interface to the entitlements stored on a user's PlayStation®Network account. Home content can use this API to retrieve information on what the player has previously purchased, either from commerce points within PS Home or from the the PlayStation®Store.

For example, an arcade scene has a number of arcade cabinets that each require 1 credit to be played. A commerce point inside of the scene sells credits to users. When a user joins an arcade cabinet, a credit is consumed.

NP Ranking

NP Ranking is a read/write command type. It allows access to users' scores for a PlayStation®3 title in a scene. The title can be used to display the rankings of all users in a space on a leaderboard, or to give the highest ranking user a special item. You can also record scores to your title's leaderboards using this API.

For example, pull list of a user's set of scores on a Mini-game in a title that you also use in PS Home. If that user beats their own score, then you can write the new high score to the PSN server.

NP Ticketing

NP Ticketing is used for authenticating external servers. This is read-only.

For an external web server to be able to authenticate a PlayStation®3 client as a legitimate PSN/PlayStation®3 user, the client must submit a ticket.

The ticket is downloaded by the client from the PSN server and then uploaded to the external server as part of the authentication handshake.

When the client is authenticated, users can use features on the external server such as leaderboards, forums, or streaming video.

For example, you host a server that has a live forum on it, and you want to use this space to ask users what they want to see in your space, to tell them what is coming, and to stream videos from the server to your scene to show them what is in store. You use NP Ticketing to authenticate each user so that they can access and post to the forums from your scene, and to choose what videos to stream.

NP Title User Storage

NP Title User Storage is a read-only API that allows for communication between user, PS Home and PlayStation®3 titles (for example, unlocking an item in PS Home after completing a quest in a title). This is read-only.

> This can only be used if a user already has storage space on a title's Title User Storage server.

The command type allows PS Home to read specified user's (and virtual user's) data from the PSN server.

For example, the user L33t has unlocked the Coriolis Effect in your title. To make PS Home more dynamic for L33t, you create Mini-game that uses NP TUS to import L33t's character's stats from your title, allowing him to play as his character against other users in the Mini-game.

NP Trophy

NP Trophy allows you to query a list of trophies a user has earned for your titles. This is read-only.

For example, you create a furniture item that users who have unlocked all of the trophies from your title receive when they enter your scene.

Using the Async Library

The AsyncCommand functions are created and executed, and some have results returned by passing the command as a parameter. NP Ticketing is slightly different to the other NP APIs, and has its own process. See NP Ticketing.

Example 1:

```
ticketCmd = AsyncCommand.Create( AsyncCommands.TicketRequest )
```

Where the Async Command `TicketRequest` is a parameter of `AsyncCommand.Create`.

Example 2:

```
AsyncCommand.Execute ( ticketCmd, ...)
```

Where `ticketCmd` is a parameter of `AsyncCommand.Execute`.

Scripting an Async Command

To use NP commands, follow this general format:

1. If the functionality requires a context, create a context using `AsyncContext.Create()`
2. Create a command using `AsyncCommand.Create()`
3. Execute the command with `AsyncCommand.Execute()`
4. For certain NP APIs, get results of a command with `AsyncCommand.GetResults()`.
5. Check results with `if(command:IsFinished())`, and `if(command:Succeeded())`
6. If required, process the results, for example, `ipairs(resultsArray)`
7. Destroy context and command by setting them to `nil` (i.e. `<context>=nil`).

How the general format looks in script:

```
Context = AsyncContext.Create( AsyncContexts.Type )

Command = AsyncCommand.Create( AsyncCommands.Type )

Command:Execute( parameter_01, ...., parameter_0x)

if ( Command:IsFinished and Command:Succeeded ) then


    Results = Command:GetResults( parameter_001, ..., parameter_00x )

    -- Do something with the results.


end

Context = nil

Command = nil
```

For a full list of functions, see the Lua API Reference.

AsyncCommand Messages

To add AsyncCommand messages, use the command ASYNC_CMD_PRINT.

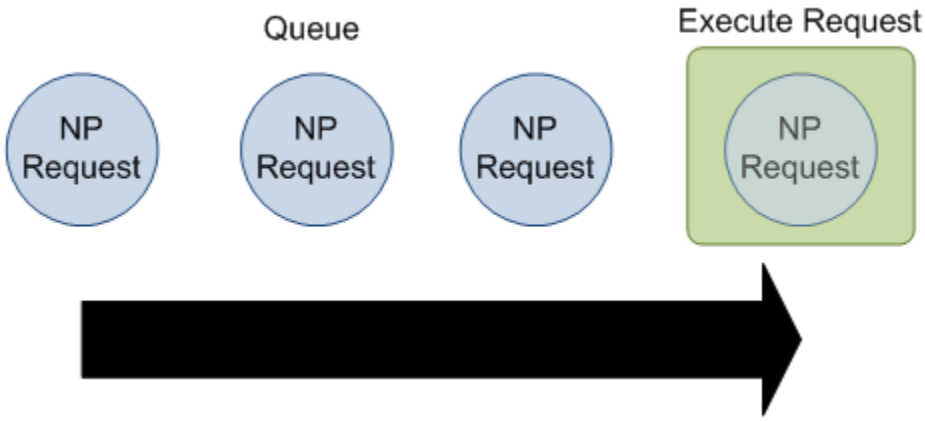
For example:

```
ASYNC_CMD_PRINT( kAsyncComdLevelDebug, "%x is spamming", this )
```

By default, the level of spamming is set to kAsyncCmdLevelError, and you can change this value with the cvar asyncCmdPrintLevel.
cvar asyncCmdPrintLevel values:

- 0 – error
- 1 – warnings and errors
- 2 – warnings, errors, and messages

NP Request Queue



To prevent an overload of the PSN servers, PS Home has a throttling system that queues NP requests.

NP Ranking and NP Title User Storage Queue

PS Home allows you to access NP Ranking and NP TUS services through scripts, which can be found in actives, Mini-games, Realtime games, arcade games and scenes. This means there could potentially be more than 60 instances of scripts accessing NP Ranking and

NP TUS. That amount of traffic could cripple both the PSN servers and PS Home.

Each source of NP requests (e.g. a Mini-game) is restricted to one request every 5 minutes, and when necessary the client throttles these requests to reduce the load on NP servers. This restriction does not apply to NP Entitlement requests.

The NP requests enter a throttling queue and execute sequentially, not simultaneously.

NP Trophy Queue

NP Trophy requests enter a separate queue. Unlike with NP Ranking and NP Title User Storage, NP Trophy accesses the user's HDD, which means that the requests are executed and processed faster.

NP Ticketing Queue

NP Ticketing and NP Entitlement requests enter a separate queue.

AsyncCommand Packets in the Throttling System

The NP Throttling system enters each individual AsyncCommand list into a queue. You cannot divide your NP Service request into several commands to reduce the size of the data you send or receive per request because of the possibility of time-outs between requests.

For example, if you need to write 10 KB worth of data using the NP Ranking APIs to your title's PSN Server, you must send the data through one request, and not through, for example, 5 requests of 2 KB each.



See PS Home network limits (Send: (2 KB/s); Receive (4 KB/s)) in Memory Limits.

NP Entitlements

This read/write API allows you to check and consume a user's entitlements, which are stored on their PlayStation®Network account. Entitlements are an object/status pair stating whether the user has access to an object, and what type of access. See the Entitlement Types section below. Home content can use the NP Entitlement API to retrieve this information on purchases the user made from either a Commerce Point or on the PlayStation®Network.

The NP service provides a user's entitlements as part of an NP ticket that identifies that user. This is different to the NP Ticketing system, which authenticates a Home user with an external server.

To check an entitlement you need a Service ID and an Entitlement ID. To do anything else, you need an Object ID as well.

Check User's Inventory

Before you check a user's entitlements or object consumption, check that the object is in the user's inventory.

Prerequisites:

- Object ID

To check if a commerce object is in the user's inventory:

1. Create an AsyncContext object, specifying AsnycContexts.Inventory as the type.
2. Create an AsyncCommand object of type AsyncCommands.InvIsInInventory.
3. Execute the AsyncCommand specifying the AsyncContext and the Object ID.
4. Get the results from the AsyncCommand object. The result will be either true if it is in their inventory, and false otherwise.

For example:

```
invAsyncContext = AsyncContext.Create( AsyncContexts.Inventory )

invAsyncIsIn = AsyncCommand.Create( AsyncCommands.InvIsInInventory )

invAsyncIsIn:Execute( invAsyncContext, "0000-0000-0000-0000" )

result = invAsyncIsIn:GetResult()
```

Why Use NP Entitlements

Use the NP Entitlement API to check a user's entitlements, that is, if a user owns, is currently renting, or can consume an

object.

- If a user is renting an object, you can check when the rental period expires.
- If a user can consume an object, you can check how many units the user has, and how many units have been consumed.

The NP Entitlement system is useful only for commerce objects (i.e. objects that are purchased at a Commerce Point or in the PlayStation®Store). See Commerce in PS Home for more information on selling content.

> ⚠ Do not use the NP Entitlement API for reward objects. The API returns useful values only for commerce objects.

There are four NP Entitlement Async commands:

Command	Description
`EntitlementRequest`	Checks the entitlement type and entitlement properties on one or more commerce objects.
`InvIsInInventory`	Checks that a specific commerce object is in the user's inventory.
`InvGetConsumableCount`	Checks the quantity of remaining charges (units) a user has on a Consumable Object.
`InvConsume`	for consuming a unit from a Consumable Object.

Be aware that each time you execute an Async command, you enter it into the throttling system. NP Entitlement requests enter the same queue as NP Ticketing requests. See NP Request Queue for more information.

Entitlement Types

An entitlement can be one of three types:

- Permanent: An entitlement of this type is permanently owned by the user.
- Rental: An entitlement of this type can be used by the user for a period of time.
- Consumable: An entitlement of this type can be used by the user a certain number of times.

Entitlement Properties

Entitlements have certain properties, depending on their type:

- `CreationDate`: a UNIX timestamp representing when the entitlement was created.
- `ExpiryDate`: (rental only) A UNIX timestamp representing when the entitlement will expire.
- `RemainingCount`: (consumable only) The number of units remaining in the entitlement.
- `ConsumedCount`: (consumable only) The number of units consumed overall.

	Type	CreationDate	ExpiryDate	RemainingCount	ConsumedCount
Permanent	"permanent"	Y			
Rental	"rental"	Y	Y		
Consumable	"consumable"	Y		Y	Y

Retrieving a User's Entitlements

`AsyncCommands.EntitlementRequest` returns a table containing the entitlement type and any relevant properties of an object.

Prerequisites:

- Service ID
- Entitlement ID

To retrieve a user's entitlements for one or more objects:

1. Create an AsyncCommand object of type `AsyncCommands.EntitlementRequest`.
2. Execute the AsyncCommand object, specifying the service ID whose entitlements you want to check.
3. Use `AsyncCommand.GetResults()` on the AsyncCommand object to retrieve a table containing the user's entitlements. When using this command, you can enter the Entitlement ID of each object that you want to check in a Lua table. If the user has no entitlements, `AsyncCommand.GetResuts()` will return nil.

If you need to check whether a user has a specific entitlement, call AsyncCommands.InvIsInInventory().

For example:

```
EntReq = AsyncCommand.Create( AsyncCommands.EntitlementRequest )

EntReq:Execute( "XXX1234-XXXX12345_00" )

entitlementIds = { "0000-0000-0000_00" , "0000-0000-0000_01", " "0000-0000-0000_02" }

results = EntReq:GetResutls( entitlementIds )
```

Consumable Objects

Consumable Objects are objects that can be used only a finite number of times. You can specify how usage is measured, for example, charges, credits, tickets.

A Consumable Object can be consumed:

- By itself: For example, an active item water cannon that can be shot 5 times
- By other objects: For example, a raffle ticket that has five charges. In a carnival space, each Realtime games costs 1 raffle ticket, allowing a user to play Realtime games 5 times in the space.

When the charges on a Consumable Object reach zero, the Consumable Object is removed from the user's inventory.

Consumable Objects are managed through the entitlements system using the Network Platform Entitlements API.

See also Types of Commerce.

Checking Charges on Consumable Units

Prerequisites:

- Object ID

To check the amount of remaining charges on Consumable Object:

1. Create an AsyncContext object, specifying AsyncContexts.Inventory as the type.
2. Create an AsyncCommand object of type AsyncCommands.InvGetConsumableCount.
3. Execute the AsyncCommand, specifying the AsyncContext and the Object ID.
4. Get the results from the AsyncCommand object. If the object is a Consumable Object and the user owns it, then the function will return the number of units remaining. Otherwise, the function returns -1.

For example:

```
invAsyncContext = AsyncContext.Create( AsyncContexts.Inventory )

invGetConsCount = AsyncCommand.Create( AsyncCommands.InvGetConsumableCount )

invGetConsCount: Execute( invAsyncContext, "0000-0000-0000-0000" )

numConsCount = invGetConsCount:GetResults()
```

Consuming Units

Prerequisites:

- Object ID

To consume a unit from a Consumable Object:

1. Create an AsyncContext object, specifying AsyncContext.Inventory as the type.
2. Create an AsyncCommand object of type AsyncCommands.InvConsume.
3. Execute the AsyncCommand, specifying the AsyncContext, Object ID, and the number of units to consume.
4. Get the results from the AsyncCommand object. If successful, this function will return the number of units consumed. Otherwise, the function returns -1.

For example:

```
invAsyncContext = AsyncContexts.Create( Inventory )

invAsyncCons = AsyncCommand.Create( AsyncCommands.InvConsume )

invAsyncCons: Execute( invAsyncContext, "0000-0000-0000-0000", 5 )

numConsumed = invAsyncCons:GetResults()
```

NP Ranking

This is a read/write API that allows you to access any user's leaderboard rankings for a particular title, and write scores to them.

With this information, you can display ranking information in your scene. For example, you can:

- Make your scene react to the ranking information, giving the highest ranking user a special item or effect
- Write the latest score achieved while in PS Home to a leaderboard
- Use a leaderboard from your title specifically for your PS Home Realtime game

Leaderboards

You use leaderboards to rate a user's score on a particular PlayStation®3 title against all the other users who have played that title while signed into PSN.

You create scoreboards through PlayStation®3 title development, not through PS Home. For more information, go to the PlayStation®3 Developer Network.

> Leaderboards have a maximum size of 64 bytes, so users may not get a score on the leaderboard, and are not guaranteed game information.

Scoreboard Behavior

This information applies to the creation and use of scoreboards, and not to PS Home itself, but you should be aware of these restrictions before using the NP Ranking AsyncCommand.

- The lag time between registering a score and the score appearing on the scoreboard is usually between 5-10 minutes. It can be an hour or more if the scoreboard has not migrated to the new PSN servers or is being manually operated on.
- The rank is determined at the exact moment of score registration, which is registered as a temporary rank. (Use the NP Ranking API to make it permanent.)
- Do not access PSN servers more than once every 5 minutes. If you access them too frequently, you can slow down the servers.

Player Character ID

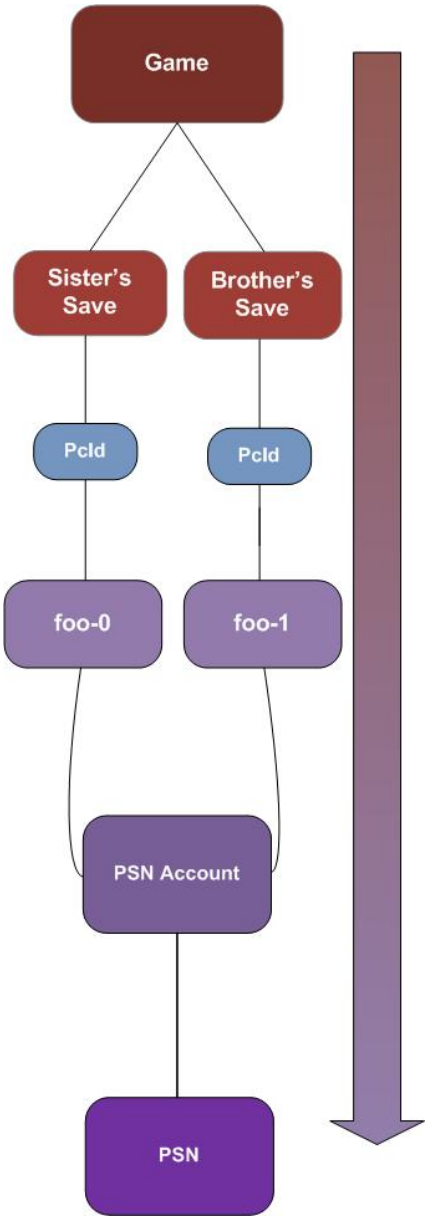
The player character ID (PcId) is a unique ID used by the NP score ranking utility. It is used in PlayStation®3 titles and PS Home when recording scores for games that support multiple players on a single PlayStation®3 title, or PS Home game. The PcId allows a title to store up to 10 unique saves per user PSN account on PSN.

Attach a value (from 0 to 9) to the NP ID for each PcId. The total number of PcIds you can create per player is 10.

Example:

A brother and sister play a game, while sharing the same PSN account. They each have their own save for the game. The game uses the PcId to call the sister's save 'foo-0', and the brother's save 'foo-1'. It stores the data from the saves on the PSN server for the game title.

The sister then signs into PS Home, and the Realtime game that she plays uses `ScoreGetRankingByNpIdPcId`, which allows her to choose her save and upload her ranking information.



Asynchronous Command Lua Interface

To create a context for NP Ranking commands:

`local passphrase = Resource.Find( 'passphrase' ) --passphrase which is stored as an encrypted file (See Protected property in the Object Editor section for information on encrypted files.)`

```
local context= AsyncContext.Create(AsyncLibs.ScoreRanking, "NPWR99999", passphrase)
```

To destroy the context:

```
<context> =nil
```

Available AsyncCommands

The `ScoreGetBoardInfo` command displays scoreboard information in a table, with the following fields:

- `number rankLimit`
- `number updateMode`
- `number sortMode`

- number gameDataLimits
- number gameDataMaxSize

The following commands retrieve ranking information of target players by NpIds, or by a particular PcId:

- ScoreGetRankingByNpId
- ScoreGetRankingByNpIdPcId

The following command registers a score to a scoreboard and enables you to obtain a temporary rank for that score:

- ScoreRecordScore

> A temporary rank is the rank at the moment a score is registered. If a higher score is registered before the next official ranking sort by the server, ranking is created with a lower rank than that of the temporary rank. When two players register the same score, they are given the same temporary rank.

The following command retrieves ranking information in the specified serial rank range:

- ScoreGetRankingByRange

NP Ticketing

You can host services on external servers that users can access from PS Home, using NP Ticketing to authenticate the exchange.

For an external web server to be able to authenticate a PlayStation®3 client as legitimate, the client must submit a ticket. The client downloads the ticket from the PSN server and then uploadd it to the external server as part of the authentication handshake. For more information, including how the Ticket Checker Module works and requesting Service IDs, see PlayStation®3 Developer Network.

When the client is authenticated, users can access features on the external server such as leaderboards, videos, or forums.

> There are no specific functions in the PS Home Lua API like there are for the PlayStation®3 SDK that retrieve individual attributes from an NP Ticket.

How NP Ticketing Works

1. Scripted content in PS Home requests a ticket from the PSN to validate accessing a server. The script requests a ticket from the PSN verifying the client.
2. The ticket is forwarded to the server (which must already be integrated with the NP TCM or sent over HTTPS).
3. When the ticket is validated, the services on the website become available to the user in PS Home.

> PS Home offers developers a server to test sending and checking an NP Ticket. See NP Ticketing Test Server.

For more information on using and setting up the TCM, go to the PlayStation®3 Developer Network.

How to Use NP Ticketing

Before creating an NP Ticket request in your script, make sure that you have:

- A Service ID (if you do not already have a Service ID, request one from your Regional Account Manager through the SCE Developer Network)
- A website hosting your service

To create a script that handles the NP Ticket:

1. Create a MemoryContainer large enough to contain your ticket, and create the HttpPostData you will wrap it in.
2. Create an HttpPostData object.
    - Add any fields to it you may need
    - Add a data header to the HttpPostData
3. Send a ticket request using the same MemoryContainer that the HttpPostData object is using. (The native side will work out how to store the ticket in the container.)
    - Wait for the ticket request to complete.
4. Finalize the HttpPostData field.

- Add any additional fields you might need.
- Finalize the `HttpPostData` object.

5. Use `Resource.Request` request to the server that will receive the ticket, passing in the `HttpPostData` object.

Lua Libraries Used

NP Ticketing uses the following Lua libraries:

- `MemoryContainer`
- `Base64`
- `HttpPostData`
- `Resource`
- `AsyncCommand`
- `Xml`

For a full list of the Lua Libraries and their functions, see the Lua API Reference.

NP Ticketing Error Codes

The following list shows NP Ticketing Error Codes from the `Execute.AsyncCommands.TicketRequest` function. If the `Execute()` call or the command fails, `AsyncCommand.GetResultCode()` will return one of the following errors:

- `-1` - A non-specific general error.
- `-2` - Invalid argument passed into `Execute()`.
- `-3` - This object already has a request in progress.
- `-4` - The ticket system is currently busy, please wait a couple of frames and try calling `Execute()` again.
- `-5` - The ticket is too large to fit within the supplied memory container.

NP Ticketing Test Server

PS Home has a server that you can use to test sending an NP ticket request.

You can test the server by adding a Lua script to a Mini-game, and then activate it online with the Home client. The NP Ticketing process is as follows:

1. Activate the object.
2. Request a ticket.
3. Post the ticket to the server.
4. Receive an xml/json response.
5. Parse the xml/json file and print its contents.

> There is an example Mini-game that you can use in PS Home to test NP Ticketing. See Example - NP Ticketing Mini-game and the NP Ticketing sample on https://home.scedev.net/projects/samples.

The NP Ticketing Test Server

- The server has a set of RESTful web services to interact with.
- All requests must be https.
- All tickets are assumed to be from service ID `EP9000-NPXX00473_00`. This is a test service ID and can be used only in SP-INT.

ASCI URL and Details

https://samples.hdk.scee.net/NPTicketing/get_ticket_data.format

Formats: xml, json

HTTPS Method

POST

Parameters: `ticket` - The ticket data as multi-part form data

Base64 URL and Details

https://samples.hdk.scee.net/NPTicketing/get_ticket_data_base64.format

Formats: xml, json

HTTPS Method

POST

Parameters: ticket - The ticket data as multi-part form data

Sample Responses

XML

An xml response for both functions looks something like this:

```
<npticket>

    <id>--enter your id here--</id>
    <Environment>Dev</Environment>
    <Issued>10/09/2009 13:04:44</Issued>
    <Expires>10/09/2009 13:14:44</Expires>
    <ServiceID>EP9000-NPXX00473_00</ServiceID>
    <Region>gb</Region>
    <Language>1</Language>
    <entitlements />

</npticket>
```

JSON

A json response for both functions looks something like this:

```
{"id":"<enter your id here>",
"Environment":"Dev",
"Issued":"10\/09\/2009 13:04:44",
"Expires":"10\/09\/2009 13:04:44",
"ServiceId":"EP9000-NPXX00473_00",
"Region":"gb","Language":1,
"Entitlements":[]}
```

NP Title User Storage

NP Title User Storage (NP TUS) is a read-only API that allows access to user's data on a PlayStation®3 title through PSN. You are allowed to read the users' PSN file on a particular title. With NP TUS you can, for example, unlock content in your space in PS Home when a user achieves a given milestone in your title.

Each title has 32 64-bit integers (TUS variables) and binary data slots (TUS data) of up to 1 MB per user. Each title is also allowed up to 8 virtual users, each of which have 128 64-bit integers and binary data up to 2 MB. This data is stored on the PSN server.

To use NP Title User Storage, a user must already have storage allocated on the title's Title User Storage server. For a user to have storage, they must:

- Play the title, while signed into the PSN
- Do something that would cause the title to access the server, such as earn a trophy, or by playing online



If a user does not have data on the Title User Storage server, PS Home cannot access TUS for the user for that title. You must write your code to handle these users (See TRC for NP).

Requesting Portions of TUS Data

TUS data can be as large as 1 MB, but this may not always be the case. You can request any size of data, but it can only be from the beginning of the data. For example, if the data you want is in the first 5KB, you can just download 5KB.

You cannot request 5KB of data starting from an offset of 512KB. To get that, you must download 512KB + 5KB.

Available Commands

- TusGetData
- TusGetDataVUser
- TusGetMultiSlotDataStatus
- TusGetMultiSlotDataStatusVUser
- TusGetMultiSlotVar
- TusGetMultiSlotVarVUser
- TusGetMultiUserDataStatus
- TusGetMultiUserDataStatusVUser
- TusGetMultiUserVar
- TusGetMultiUserVarVUser

NP Trophy

PlayStation®3 trophies are awarded while the user is playing a title. When a trophy is awarded, it is saved to the user's HDD. The trophies can be synced to the NP Trophy PSN server.

You can use the NP Trophy API to query a user's trophies saved to their HDD. You can obtain only the unlocked status of a trophy from this API. It does not return trophy images and names. To display the images and names of trophies, you must package the images, and use the Lua script.

With NP Trophies you can, display users' trophy information on leaderboards, or unlock features within PS Home (such as a Mini-game for those players who have earned a particular trophy.) You cannot unlock trophies in PS Home because the NP Trophy API is a read-only API.

Available Commands

The following command is available for NP Trophies:

TrophyGetUnlockStates

NP Best Practices

All APIs

Cache NP Data

After first accessing NP data, cache the data instead of accessing NP Services each time you want to use the same data.

Check the Status of an NP Request Before Sending Another

With the NP Throttling queue, some requests may take longer than expected to save or return data.

Write your code to handle repeat NP Service requests elegantly. For example, make sure that it checks the status of existing NP Service requests (for example, to see if they timed out), so that your content does not spam PSN servers, or fill the NP Throttling queue with the same request.

Assume NP Requests May Fail

Assume that any NP request may fail permanently, that is, even on retrying. These circumstances can arise, for example, when the NP server you are trying to access is undergoing maintenance.

Use Network Messages to Synchronize

Use network messages to synchronize if the data extracted from NP is small and easily replicable, rather than performing an NP service query for each user.

For example, if a user's active item accesses NP services when it is first deployed, use network messages to give the other clients the data, instead of requiring them to issue an NP request.

> For active items and Mini-games, use Network Property Bags wherever possible. They allow you to easily share common data between users over a network. See the Lua API Reference.

Run NP Commands in the Background

Always run NP commands in the background so that they do not affect the user experience. Do not make the retrieval or submission of an NP command pivotal to your PS Home content, if the NP service fails or the request takes a long time to complete.

For example, let users play through a Mini-game session before showing the Mini-game's scoreboard, so that the NP service has time to run, process, and return values in the background.

Other

Be aware of what data you need, and how much you request. Requests for large amounts of data can significantly slow the PSN servers and PS Home.

NP Ranking

- Do not obtain all the possible scoreboards simultaneously. Only request those that are necessary for the operation.
- You can retrieve scoreboard information in the background during gameplay, but leave an appropriate amount of time between each successive command to prevent spamming the NP service.
- The lag time between registering a score and the score appearing on the scoreboard is usually between 5-10 minutes. It can be an hour or more, so design your code to handle these extreme delays.

NP Restrictions

Network Platform APIs

- NP IDs (Service IDs, Entitlement IDs, Product IDs and User IDs) should not be exposed to the users.
- Do not divide the amount of data you want into packets. If you need 1 MB of data, do not execute 10 NP requests of 102.4 KB each. Request the full 1 MB of data.
- NP requests need to be activated by the user. For example, joining a Mini-game.
- The following restrictions exclude NP Entitlements:
    - PS Home content should not rely on NP data. For example, a Mini-game should not depend on receiving NP Ranking or NP TUS data before letting you interact with it. The delay between sending a request for data and receiving it can be several minutes (or a response may never arrive), during which time users may think that the game is broken, and leave.
    - Do not send a unique NP request more frequently than once every 5 minutes. Cache as much information as possible to avoid unnecessary access to the server.

Network Platform Ranking

- Do not obtain all the possible scoreboards simultaneously. Request only those that are necessary for the operation.
- Scores can be posted only once every 5 minutes.

Network Platform Ticketing and Entitlements

- Only one source (such as a Mini-game) can make NP Ticketing and NP Entitlement requests in a scene.

NP Ticketing Only

- The ticket must be submitted to an external server over an SSL connection (such as HTTPS), or with a server that has been integrated with the NP Ticket Checker Module (TCM).
- NP TCM is not provided by PS Home or through the HDK. You can find documentation and support for it through the PS3 Developer Network.
- With NP TCM, you cannot specify a cookie parameter in the ticket request.
- The maximum size of ticket data is 64 KB.

NP Entitlements Only

- When consuming an item, content must check that results of an `InvConsume` request using `GetResults`. Do not assume that successfully executing an `InvConsume` request means that requested amount of units were consumed.

- Make users fully aware if they are about to consume an item, and how much they will consume.

> Also see Information Transmission limits and policies in General Design Guidelines and Distributed Memory Engine Limits.
>
> PlayStation Developer Network Guidelines on NP can be found here
> https://ps3.scedev.net/docs/ps3-en,NP_Ranking-Overview/

Example – NP Ticketing Mini-game

This example provides a sample set-up for your script to run a mini-game in PS Home that checks a ticket on a test NP server.

You can download the NP Ticketing sample from https://home.scedev.net/projects/samples.

The sample uses the following Lua components in a mini-game:

- OSD (On Screen Display) script
- NP Ticketing script
- main script
- Util (Utilities) script

It also includes two XML files, `osd_interface` and `types`, which are both for OSD. For more information on using OSD, see On Screen Display (OSD) Lua Library.

When you activate the NP Ticketing mini-game in PS Home, you can do the following:

- Request a ticket from the NP Ticketing service, and use it to communicate with the test NP server.
- Switch between standard ASCII or Base64 encoding for requests to the server.
- Switch between receiving XML or JSON responses from the test server.
- Acknowledge an error response from the test server (for XML responses only).
- Parse XML/JSON responses from the server into a format readable by the user.
- Demonstrate how to wait for responses without freezing the client.
- Demonstrate the entire process to the player in realtime via an OSD.
- Use caching for holding a blank NP request. (The sample does not need to request a new NP ticket every time the [CROSS] button is pressed.)

The script:

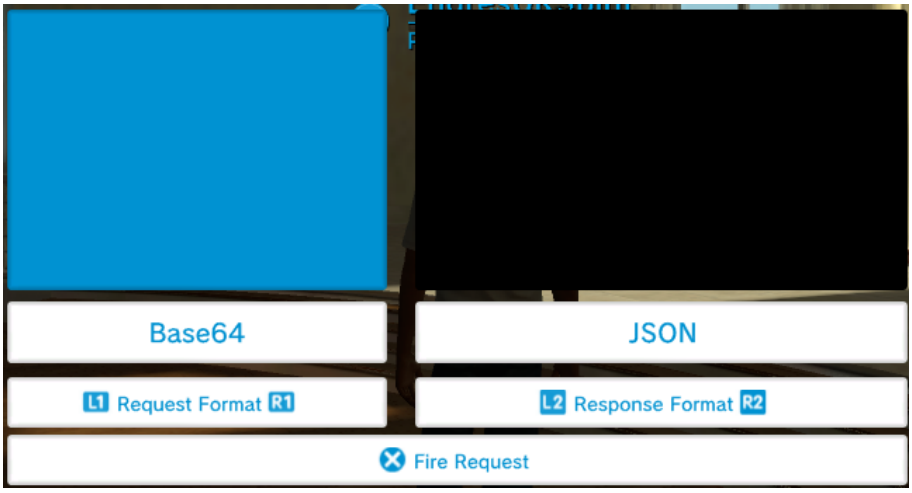
- Allows for timeouts for NP and resource requests. A request is given five seconds before it is canceled.
- Has a 'lifetime' counter that keeps track of whether the ticket in a cached request is still valid or has expired.
- Can handle failed requests.

> Note that:
>
> - The sample ignores new requests when one is already in progress. If you press the [CROSS] button before the result of a current request shows up on the OSD, nothing happens.
> - The sample code allows you only to send a ticket and receive the response. It does not acknowledge an error response from the test server when using JSON responses.

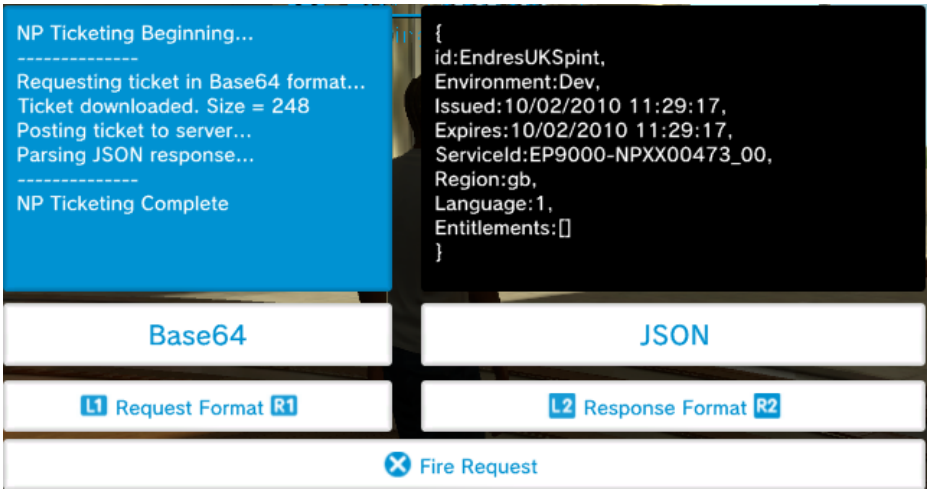
> Do not use this code for your live content. Use it only to test sending a ticket on our server to see how it works, without the need to set up your own server.

The following screenshots show what the example script produces in PS Home.

When first activated, you see the following:



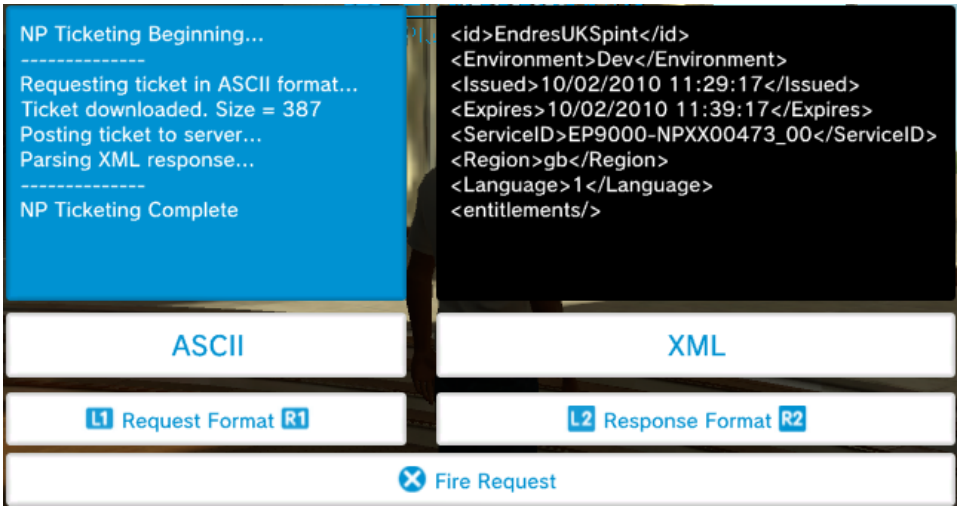
If you choose Base64 and JSON, you see:



If you choose Base64 and XML, you see:



If you choose ASCII XML, you see:



If you choose ASCII Base64, you see:



Launch into PS Home from XMB™ or PS3 Title

You can launch into a specific scene (or even a specific spawn point) in PS Home from outside of PS Home.

> For example
>
> Launch PS Homefrom the XMB™:
>
> Regional SCE put a PS Home space called "Easter Bunny Hunt - Open Season" on the What's New section on the XMB™. When someone selects it from the XMB™, they are taken straight into the space, rather than having to find the
>
> "Easter Bunny Hunt - Open Season" space in the Navigator after launching into PS Home.
>
> Return to PS Home from a game title:
>
> A PS3 title ("Awesome Dodo Adventures") supporting game launching from PS Home can make use of the same mechanism. PS Home launches the title "Awesome Dodo Adventures", when user selects to return to PS Home they are directed to different spaces depending on whether they won or lost.

Speak to regional SCE if you want to use this feature in your content.

You can use this feature if you want to Launch into PS Home from XMB™ into a specific PS Home space (or even a specific spawn point).

If you have used game launching in PS Home to launch the player into a PS3 title, you can use the Game Return feature to return the player (and return data) to PS Home from the title. For more information, see Launch into PS Home from PS3 Title - Game Return.

Game Spaces and Video Spaces (16 players) cannot be directly accessed from menus (the XMB™, Navigator or Menu Screen) but must be entered via a Public Space or Clubhouse, using any of the Relocate functions. See Relocating to Instances.
Game Spaces (32 players) can be directly accessed from menus (the XMB™, Navigator or Menu Screen).

See also PS Home Spaces and Memory Limits for Spaces.

Returning to PS Home from PS3 Title – Game Return

If you have used game launching in PS Home to launch the player into a PS3 title, you can use this feature (Game Return) to return the player (and return data) to PS Home from the title.

How do I return a player from my game?

You have the option of supporting game launching in your PS3 title, and if you want to you can submit a game launch object to PS Home.

It is possible to make use of the Game Return functionality without having specific support for PS Homegame launching added to the PS3 title. For more information, see Game Launch Objects.

When the title is launched, it should check for the return value of `CELL_GAME_GAMETYPE_HOME` from the `cellGameGetBootGameInfo` function to determine if the title was launched from PS Home.

If the title was indeed launched from PS Home, once the user has completed their game, the title writes a text file to the PS3 HDD containing details to be returned to PS Home. The title will need to prompt the user to return to PS Home through the [PS button] option.

If valid, the details written in the text file will be used to direct the user to a specific PS Home space.

Additionally, when the player arrives at the space, you have the option of including scripted functionality that uses the information passed in through the text file. This information can be accessed using the Lua function `Scene.GetLaunchParam`.

What should be in the text file?

The file should be called `PSHOMELAUNCH.TXT` and written to the path provided by `cellGameGetHomeLaunchOptionPath()`, for example:

```
ret = cellGameGetHomeLaunchOptionPath(gamePath, gamePathPersonal ) ;

if ( ret == CELL_GAME_RET_OK )

{

sprintf( returnFilePath, "%s/PSHOMELAUNCH.TXT", gamePath );

}
```

The text file should contain &-separated key/value pairs, for example:

```
version=1.0&scene_name=MYHOMESPACE_ID_181D_4365&spawn_id=blah&opt1=1&opt2=3.14
```

Here is a list of the keys currently supported and their meanings:

Keys	Meanings	From the example
version <optional>	1.0 To handle version difference in the future. If omitted, the version will be 1.0.	`version=1.0`
scene_name	String A unique ID used in PS Home which indicates 1 specific Space.	`scene_name=MYHOMESPACE_ID_181D_4365`

spawn_id<optional>	String Which Spawn point to go in the Scene. If not specified, the default spawn point will be selected.	spawn_id=blah
opt1<optional>	Enumeration of optional parameter. Game Specific data can be listed.	opt1=1
opt16 <optional>	Optional parameters can be added.	opt2=3.14

Limits

There can be up to 16 options specified. Each option value can be up to 128 bytes long.

The entire text file should always be smaller than 4 KB.

For example

Example flow:

1. User launches from PS Home into PS3 title and plays game, scores 4500 points.
2. Title writes out PSHOMELAUNCH.TXT file containing:

   ```
   version=1.0&scene_name=MYHOMESPACE_ID_181D_4365&opt1=4500
   ```

3. Prompts user to return to Home which the user does. He returns directly to the MYHOMESPACE scene
4. In that scene the script calls:

   ```
   score = Scene.GetLaunchParam( 1 )
   ```

5. and reacts to the score returned from the game (in this case 4500 points).

Other Considerations

At the moment there is no guarantee that multiple users returning to PS Home from a single game launch session will return to the same instance of a space.

Launch into PS Home from XMB™

You can use this feature if you want to launch from What's New in the XMB™ into a specific PS Home space (or even a specific spawn point).

Speak to regional SCE if you want to use this feature in your content.

Your Regional Account Manager can help you arrange to launch from What's New in the XMB™.

Contact them and provide them with the following:

- Scene Name
- Spawn ID (optional)
- repeatable parameters (optional)

Testing Game Return

When the user launches into your PlayStation®3 title from PS Home, you can detect the event and modify your title's startup appropriately. This is called Game Launch. For more information, see Game Launch Objects.

After Game Launching, you can write data to the PlayStation®3 HDD which can be used by PS Home (and your Home content) to change the behavior when the user chooses to return to PS Home via the XMB™. You could, for instance, send the winners of the game to one space, and the losers to another. You could also use optional parameters in the return data to have results shown in the Home space; a script can read these optional parameters by calling `Scene.GetLaunchParam( index )`. This is called Game Return. For more information, see Launch into PS Home from XMB™ or PS3 Title.

You can also test the Game Return data by launching PS Home from the XMB and using a USB stick.

Test Game Return

1. Write the Game Return .txt file, and edit the hubstartup.txt file.
2. Place the Game Return .txt file and hubstartup.txt file in the root directory of a USB stick.

3. Launch PS Home from the XMB™.

Game Return text file

The .txt file on the USB stick is the same as the one written to the PlayStation®3 HDD for Game Return. For more information, see Launch into PS Home from XMB™ or PS3 Title.

File name: The file can be any name. The name cannot have spaces. It must have a .txt extension. For example: PSHOMELAUNCH.TXT, and must be all caps.

Format: The Game Return .txt file should contain &-separated key/value pairs, for example:

version=1.0&scene_name=MYHOMESPACE_ID_181D_4365&spawn_id=blah&opt1=1&opt2=3.14

Here is a list of the keys currently supported and their meanings:

Keys	Meanings	From the example
version <optional>	To handle version difference in the future. If omitted, the version will be 1.0.	version=1.0
scene_name	String A unique ID used in PS Home which indicates 1 specific Space.	scene_name=MYHOMESPACE_ID_181D_4365
spawn_id<optional>	String Which Spawn point to go in the Scene. If not specified, the default spawn point will be selected.	spawn_id=blah
opt1<optional>	Enumeration of optional parameter. Game Specific data can be listed.	opt1=1
opt16 <optional>	Optional parameters can be added.	opt2=3.14

Limits

There can be up to 16 options specified. Each option value can be up to 128 bytes long.

The entire text file should always be smaller than 4 KB.

Special Characters

To use special characters, use the Escape Character.

For example, if you want to pass the value 'a&b' in an opt parameter, use: opt1=a%26b

%26 is the Escape Character for the ampersand.

> We recommend avoiding using special characters in arguments.

Example File	Effect
scene_name=Cinema	Launches user at default spawn point in scene Cinema.
Verdana	Launches user at spawn point 'blah' in scene Cinema. Version is marked as 1.0.
scene_name=Cinema&spawn_id=blah&opt1=1&opt2=3.14	Launches user at spawn point 'blah' in scene Cinema. A script in the scene checks to see if values are passed in parameters opt1 and opt2 by calling Scene.GetLaunchParam(index), and then uses these values.

Hubstartup.txt File

Write the following in the hubstartup.txt file:

```
GameReturnTestLocal usb:<GAME_RETURN.txt>
```

For example

```
GameReturnTestLocal_usb:PSHOMELAUNCH.txt
```

The DUALSHOCK®3 Controller

The PlayStation®3 system includes a wirelessDUALSHOCK®3 Controller.



©2006 Sony Computer Entertainment Inc. All rights reserved.
Design and specifications are subject to change without notice.

The wireless controller for the PlayStation®3 system provides the most intuitive game play experience, with pressure sensors in each action button and the inclusion of the highly sensitive SIXAXIS™ motion sensing technology. Each hit, crash and explosion is more realistic when the user feels the rumble in the palm of their hand. It can even detect natural movements for realtime and high precision interactive play, acting as a natural extension of the user's body.

DUALSHOCK®3 utilizes Bluetooth technology for wireless game play, and the controller's USB cable to seamlessly and automatically charge the controller through the PlayStation®3 at any time.

Handling User Input

Setting Up Controllers

To make controller functions accessible, the controllers must be configured for use. Before calling any Lua functions from the Pad library, the Pad component must be added to your scripted object and the Pad library must be loaded using the function call LoadLibrary("Pad"). When this is complete, up to four controllers can be accessed by first creating instances of a specified number of controllers and then assigning each to a variable.

```
Pad.CreatePads(2)
player1 = Pad.GetPad(1)
player2 = Pad.GetPad(2)
```

Reserving Pad Buttons

When the controllers are available, you can start reserving and assigning controller buttons. When a button is reserved, it is assigned a user-definable name that can be used to represent a game function as opposed to a button label. For example:

```
jump = 1
run = 2
fire = 3
xMovement = 4

Pad.Reserve( player1, PAD_CROSS, jump, false, false )
Pad.Reserve( player1, PAD_SQUARE, run, false, false )
Pad.Reserve( player1, PAD_R2_SHOULDER, fire, true, false )
Pad.Reserve( player1, PAD_JOY_LEFT_X, xMovement, false, false )
```

The Pad.Reserve function takes the following arguments:

1. A Pad instance.
2. One of 26 button identifiers: these identifiers cover all available controller functions, including the analog sticks, motion sensor functions, DUALSHOCK®3 vibration etc.
3. actionId - The user defined action ID to map the button ID to. You can enter any number between 0-50.
4. A boolean to determine if the Pad.WasJustPressed function repeatedly returns true if a button is held down. (A bit like 'key repeat' on a keyboard).
5. A boolean to determine whether the input event should 'drop through'. This is very rarely used, and should be used with caution. Having this set will mean that specific lower priority input handlers within PS Home will also receive the input event. E.g. if the left joystick is reserved, and allowed to drop through, you'll receive the input event, but will still be able to use the left joystick to move your avatar.

When all the required functions are reserved, the controller is ready to be accessed. When reserved controller functions are no longer needed, they can be released using the Pad.UnReserve function, which allows the controller functions to be used for other purposes.

Two special ButtonIds exist, PAD_ACCEPT and PAD_DECLINE, which, when reserved, use either the Cross or Circle button depending on the console region. PAD_CROSS and PAD_CIRCLE use the Cross button and Circle button regardless of region. So be very careful to pick the correct ButtonIds.



There are two ways to exit something in PS Home: START, and PAD_DECLINE.
START is used, for example, in Mini-games, and PAD_DECLINE is used for features such as the Wardrobe.

If you reserve a pad button, but then you want to reserve the button with a different action, you need to unreserve the button first. Otherwise, the second reserve call is ignored, and the initial reservation persists.

It is safe to unreserve a button that isn't reserved. The call is ignored.

If you want to relinquish a previous reserved action ID, use the Pad.UnReserve function. If you want to relinquish all reserved action IDs, use the Pad.UnReserveAll function.

Controller Functions

Handling Button Presses

There are four functions which detect button presses:

- Pad.WasJustPressed returns true if a given button was just pressed down (or held, if 'key repeat' is set).
- Pad.WasJustReleased returns true if a given button was just released.
- Pad.IsHeld returns true if a button has been pressed, but has not yet been released.
- Pad.WasAnyJustPressed returns true if any button was just pressed down.

These functions can be called inside a scripted object using the OnLocalPlayerUpdateGameplay callback function to detect button presses, for example:

```
function OnLocalPlayerUpdateGamePlay()
   if( Pad.WasJustPressed( player1, jump ) then
      -- Code to make player jump
   end
   if( Pad.IsHeld( player1, run ) then
      -- Code to make player run
   end
end
```

Analog Features

Every button on the controller is pressure sensitive (except for the L3, R3, Select, Start and 'PS' buttons) and can return an analog pressure reading via the `Pad.GetAnalogExtent` function. This function is used to access the analog reading from all analog features of the controller, including buttons, analog sticks and motion sensor function. The values returned depend on the type of feature accessed:

- Buttons return a value between 0 (not pressed) and 1 (fully depressed)
- D-Pad directions return a value between 0 (not pressed) and 1 (fully depressed)
- Analog sticks return a value between -1 and 1, separated into X and Y components.
- Motion sensor functions return values between -1 and 1.

Example:

```
if( Pad.IsHeld( player1, fire ) then
      -- Shoot at a rate depending on how hard the fire button is pressed
      Shoot( Pad.GetAnalogExtent( player1, fire ) )
end

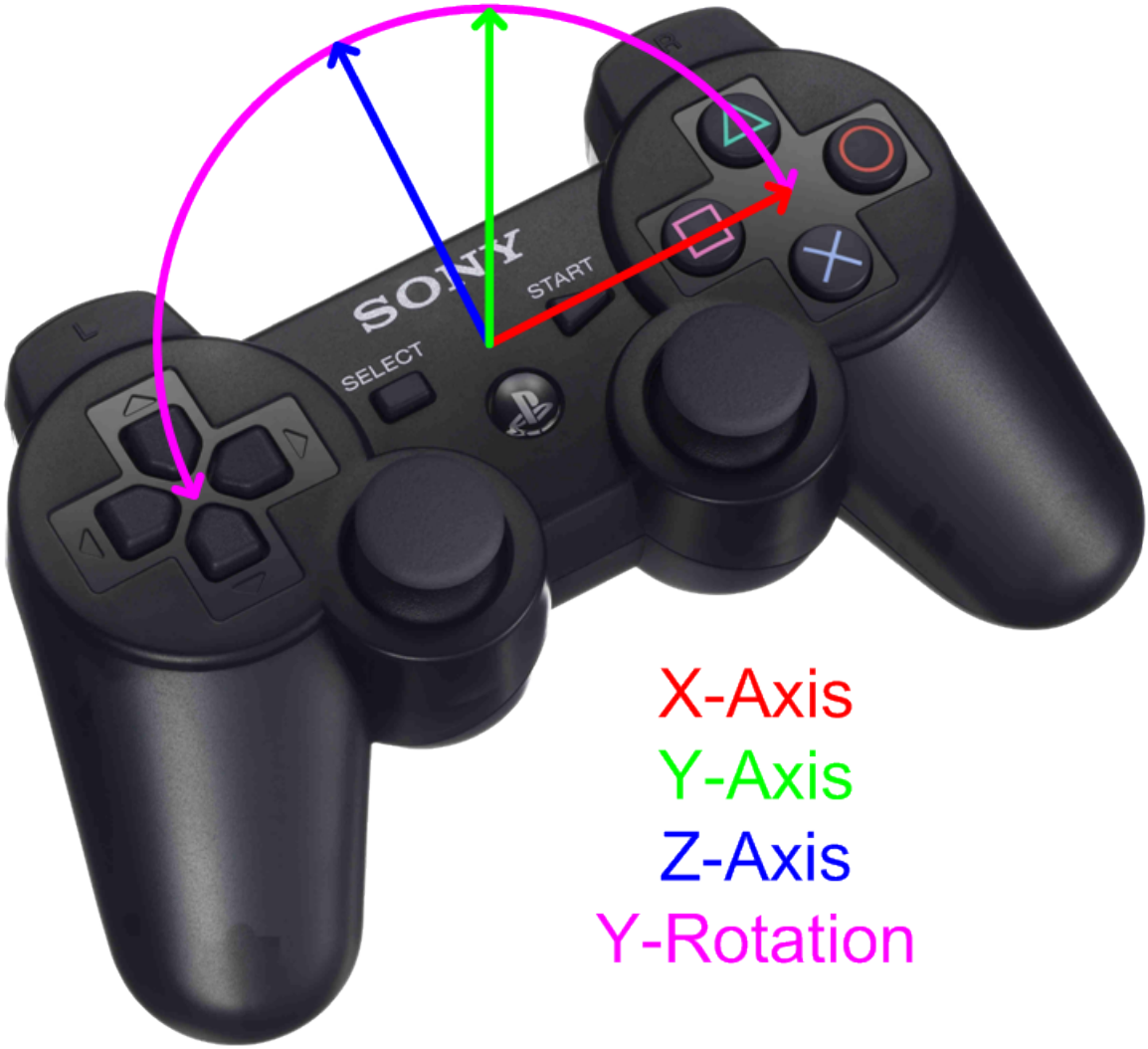
-- Move player at speed depending on analog stick
MovePlayer( Pad.GetAnalogExtent( player1, xMovement ) )
```

Motion Sensor Functions

The controller features four sensors to detect motion and tilt, three accelerometers (one for each axis) and an angular velocity sensor (gyro) that detects yaw (rotation on the Y-axis). All four sensors can be reserved and accessed in the same way that any analog button can by calling the `Pad.GetAnalogExtent` function. The analog values reported during motion (when holding the controller flat) are as follows:

- X-Axis: Moving the controller to the right produces a negative acceleration, moving left results in a positive acceleration.
- Y-Axis: Moving the controller upwards produces a positive acceleration, moving downwards results in a negative acceleration.
- Z-Axis: Pushing the controller away from you produces a positive acceleration, pulling it towards you results in a negative acceleration.
- Y-Rotation: Rotating the controller clockwise produces a positive acceleration, rotating anti-clockwise results in a negative acceleration.

X-Rotation (Pitch) and Z-Rotation (Roll) can be calculated using the Z-Axis acceleration and X-Axis acceleration sensors respectively.



You can obtain sensor information using the `Pad.GetSensorForwards`, `Pad.GetSensorRight` and `Pad.GetSensorUp` functions. These functions return a vector of their respective directions; you can use these functions to create effects such as rotating objects as the user rotates the controller, for example:

```
tiltCube = Entity.Create()
tiltCube:SetModel( "cube" )
tiltCube:SetPosition( Vector4.Create( 0, 1, 0 ))
target = Vector4.Create()
target:Add(tiltCube:GetPosition(), Pad.GetSensorForwards(player1))
tiltCube:LookAt(target, Pad.GetSensorUp(player1))
```

Be aware that gravity causes a constant acceleration on the controller's sensors. When the controller lies flat, there is an acceleration of around 0.22 on the Y-Axis. When the controller is rotated 90° on the Z-Axis, this acceleration is transferred to the X-Axis and when rotated 90° X-Axis, it is transferred to the Z-Axis. An acceleration between 0 and 0.22 is recorded on all axes if the controller is held at an angle that is not parallel to any axis.

Vibration

The `Pad.SetActuator` function produces controller vibration when used in conjunction with a DUALSHOCK®3 controller. The wireless controller features two actuators, one small and one large that can be configured independently. These actuators must be reserved using `Pad.Reserve` as a button or sensor would. Each actuator can run at a given percentage of its full power, for example, the following code will run the controller's small actuator at 80% of its full power.

```
smallVibration = 5
Pad.Reserve( player1, PAD_ACTUATOR_SMALL, smallVibration)
Pad.SetActuator( player1, smallVibration, 0.8 )
```

The Pad API Sample

The PAD API sample demonstrates all of these features and is available from https://home.scedev.net/projects/samples.

The game displays an on-screen representation of each connected controller and tints all analog buttons red to reflect the analog force exerted on the button. Analog stick movement is reflected by a small red circle that moves around the stick as it is moved. The small and large actuators are assigned to the L2 and R2 triggers respectively, and will vibrate with an intensity related to the analog extent reading of the triggers. Raw motion sensor readings for controller 1 are displayed on the left along with the maximum accelerations recorded since the Mini-game started. These values can be reset by pressing the cross button. Three cubes are used to represent potential uses for the sensors:

- The left cube displays controller rotation using the `tiltCube:LookAt(target, Pad.GetSensorUp(player1))` function.
- The right cube displays Y-rotation readings from the angular velocity sensor.
- The centre cube represents the current acceleration exerted on the cube.



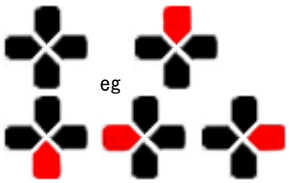
Pad Icons in Localization

When displaying text in PlayStation®Home, it is often necessary to indicate Control Pad buttons with icons, for example, in a legend. Text always needs to be translated into the correct language, and consequently the text is always obtained from the localization system. The text in the files loaded into this system can use the following tags to denote the icons.



©2006 Sony Computer Entertainment Inc. All rights reserved.
Design and specifications are subject to change without notice.

Button	Icon on the Controller	Action in PS Home
[ACCEPT]	or	The [ACCEPT] button is the cross or circle button depending on region.
[DECLINE]	or	The [DECLINE] button is the circle or cross button, depending on region.
[CROSS]	or	Deprecated. This will do the same as [ACCEPT].
[CIRCLE]	or	Deprecated. This will do the same as [DECLINE].
[ALL_REGION_CROSS]		This will be converted into the Cross icon (regardless of regional settings that swap the cross and circle button usage).
[ALL_REGION_CIRCLE]		This will be converted into the Circle icon (regardless of regional settings that swap the cross and circle button usage).
[SQUARE]		This will be converted into the Square icon.
[TRIANGLE]		This will be converted into the Triangle icon.
[DOWN]		This will be converted into the D-Pad icon, with the down button highlighted.
[LEFT]		This will be converted into the D-Pad icon, with the left button highlighted.
[RIGHT]		This will be converted into the D-Pad icon, with the right button highlighted.

[UP]		This will be converted into the D-Pad icon, with the up button highlighted.
[UP][DOWN]		This will be converted into the D-Pad icon, with the up and down buttons highlighted.
[LEFT][RIGHT]		This will be converted into the D-Pad icon, with the left and right buttons highlighted.
[UP][DOWN][LEFT][RIGHT]	 eg	This will be converted into the D-Pad icon, with all the buttons highlighted.
[L1]		This will be converted into the L1 shoulder button icon.
[L2]		This will be converted into the L2 shoulder button icon.
[L3]		This will be converted into the L3 (left stick click) icon.
[LEFTSTICK]		This will be converted into the Left Stick icon.
[R1]		This will be converted into the R1 shoulder button icon.
[R2]		This will be converted into the R2 shoulder button icon.
[R3]		This will be converted into the R3 (right stick click) icon.
[RIGHTSTICK]		This will be converted into the Right Stick icon.
[SELECT]		This will be converted into the Select Button icon.
[START]		This will be converted into the Start Button icon.
[MOTIONSENSOR]		This will be converted into the Motion Sensor icon.

For example:

In Localization Text	On-Screen Appearance
Press the [TRIANGLE] button	Press the △ button

Pad Icons in Localization and Reserving Pad buttons in Lua Script

> The localization icon tag IDs differ slightly in naming convention from the IDs used to reserve pad buttons in Lua script.

Reserving button IDs in Lua script:

PAD_ACCEPT= CROSS or CIRCLE (depending on region).
PAD_DECLINE= CROSS or CIRCLE (depending on region).
PAD_CROSS= CROSS (all regions).
PAD_CIRCLE= CIRCLE (all regions).

Button icon tag IDs in localization:

[ACCEPT]= CROSS or CIRCLE (depending on region).
[DECLINE]= CROSS or CIRCLE (depending on region).
[CROSS] = Deprecated. This will do the same as [ACCEPT].
[CIRCLE] = Deprecated. This will do the same as [DECLINE].
[ALL_REGION_CROSS]= CROSS (all regions).
[ALL_REGION_CIRCLE]= CIRCLE (all regions).

OSD (On Screen Display) Lua Library

You use the OSD library to create on-screen dialog boxes and menus that are in the same style as the PS Home user interface. Typical uses for OSD include:

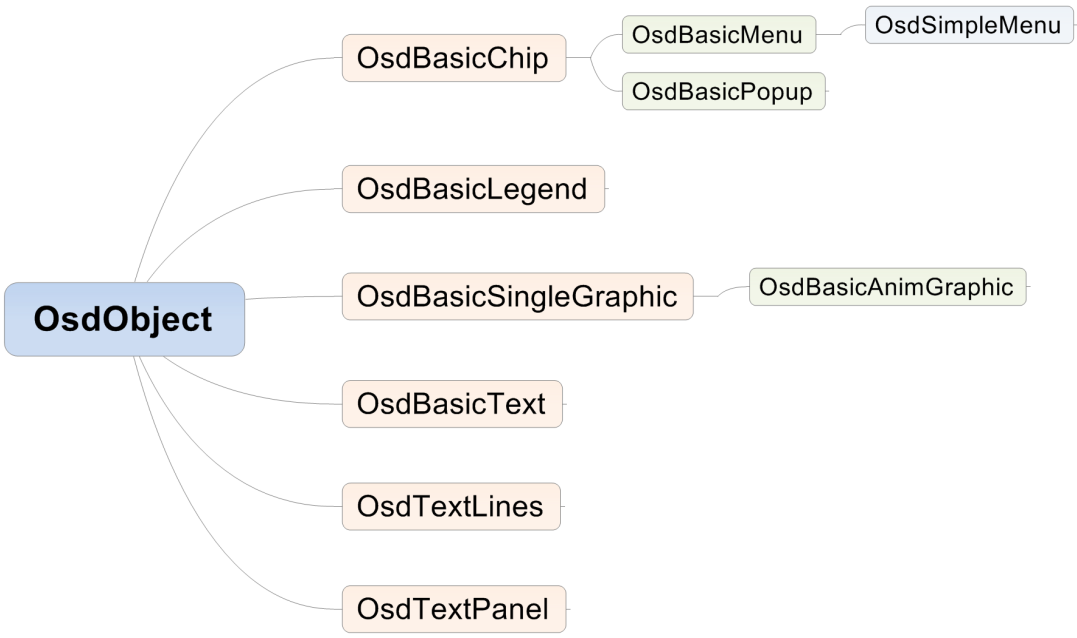
- Event notifications
- Displaying game-related information (such as scores)
- Menu systems

OsdObject Hierarchy

When you use the OSD library, you create OSD elements. These elements have their own functions.

All OSD elements are derived from the `OsdObject`. The `OsdObject` is itself an OSD element. It contains functions and controls common to all of the OSD elements, such as position, size and visibility.

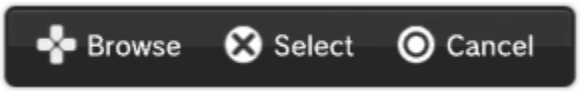
OSD elements are arranged in a hierarchy that allows child elements to use the functions and features of their parents. The following diagram shows the OSD element hierarchy:



Some parent OSD functions are blocked from being used by their children. They are blocked to maintain consistency. For example, you cannot use `OsdBasicChip.CreateContentObject()` with an `OsdBasicPopup`, as doing so places content in an area that should remain blank in an `OsdBasicPopup`. However, in some cases it is required to use parent functions; `OsdBasicPopup` has no function of its own to control its size, so in this case you need to use functions such as `OsdBasicChip.SetSize()`.

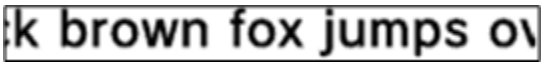
OSD Elements

OsdBasicLegend



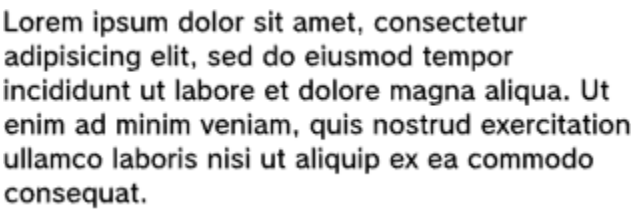
You normally use the `OsdBasicLegend` element to display controller information that is relevant to the current situation. You should add individually each piece of information that needs to be displayed on the legend. If the legend is too small to display all the information, it automatically rotates in the information. By default, the `OsdBasicLegend` appears centered at the bottom of the screen.

OsdBasicText



The `OsdBasicText` element displays a single line of text. You can clip the visible area of the text and then set the text to scroll along the visible area.

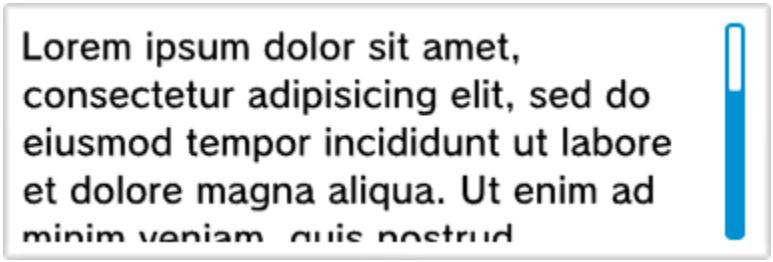
OsdTextLines



`OsdTextLines` displays text that wraps when it reaches a specific width. The text can have custom line spacing applied and can

be aligned to the left, right or center.

OsdTextPanel



An `OsdTextPanel` displays wrapped text in a scrollable text box.

OsdBasicSingleGraphic

An `OsdBasicSingleGraphic` displays up to two static textures (foreground and background) and can use the texture's alpha channel to integrate the user's selected color scheme. Black areas of the texture's alpha channel are fully transparent and white areas are displayed using the color scheme. As a result, images with no color channels (alpha-only) use only the user color scheme. For example, the following two images show foreground and background textures using only the alpha channel.

Samples of these alpha-only textures are included in the OSD API sample which you can download from https://home.scedev.net/projects/samples.



You use alpha-only images display a user's selected color, and when they are used together, they produce the following image:



The colors can be customized by applying tints of various colors. In addition to a range of standard colors, you can use the following special color types:

- `OsdColor.Foreground` is the user's selected color.
- `OsdColor.HighlightFg` is a lighter shade of the user's selected color.
- `OsdColor.Background` is the standard OSD background color (white).
- `OsdColor.OppositeBg` is the opposite of the standard OSD background color (black).
- `OsdColor.Normal` reverts the default settings.

The following table shows examples of these settings in use:

Front Texture Tint	Background Texture Tint	Result
`OsdColor.Background`	`OsdColor.Foreground`	
`OsdColor.OppositeBg`	`OsdColor.Normal`	

OsdColor.Red	OsdColor.Grey	

OsdBasicAnimGraphic



OsdBasicAnimGraphic derives from OsdBasicSingleGraphic and therefore supports all the features of OsdBasicSingleGraphic. In addition, it supports animation.

OsdBasicChip

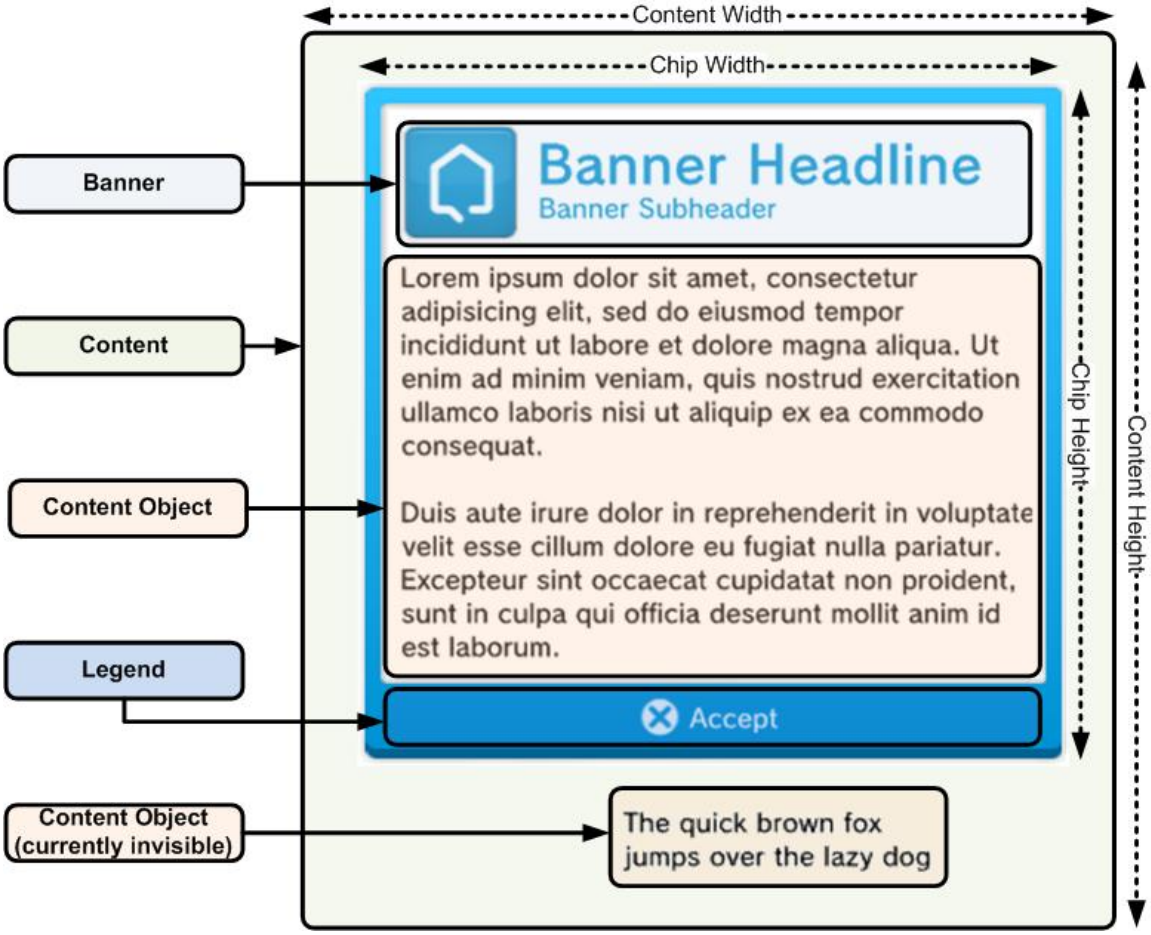


The OsdBasicChip combines several smaller OSD elements into one object. It arranges several default elements in a consistent way and contains an area on which to display custom content. The size of the visible area where custom content is displayed is determined by the size of the chip.

OsdBasicChip Components

The OsdBasicChip comprises the following elements:

- Banner Icon: A graphic that appears in the top left corner of the chip. Both foreground and background textures are supported, but unlike OsdBasicGraphic, you cannot change the colors.
- Content: A canvas area that contains all the content that needs to be displayed. The area of the content area can be much larger than the size of the chip itself, but only content that is currently inside the main area of the chip is displayed. The content area can change position in order to bring other content into view.
- Content Object: You can create any OSD element as a child element of the content area. This element is known as a Content Object. You can manipulate content objects in the same way as standalone OSD elements.
- Legend: The legend of an OsdBasicChip supports a similar range of features to an OsdBasicLegend, but is integrated into the bottom of the chip itself as opposed to being a separate element.



OsdBasicPopup



The `OsdBasicPopup` derives from `OsdBasicChip` and enables you to create simple text-only dialog boxes without having to fully specify an `OsdBasicChip`. For example, you might want to display a notification for a short period. An `OsdBasicPopup` contains a title (up to two lines) and a message body. The height of the `OsdBasicPopup` automatically expands to fit whatever text the message body contains.

OsdBasicMenu

The `OsdBasicMenu` also derives from `OsdBasicChip` and enables you to create a menu system with options arranged vertically. Each option added to the list can be any type of OSD element, or a custom OSD element.



The OsdBasicMenu enables you to specify how the look of the menu should change when users interacted with it using the functions OsdBasicMenu.SetOnSetItemSelectedCallback() and OsdBasicMenu.SetOnSetItemEnabledCallback(). In the above example, the highlighted item is changed to display blue text and the colors of its icon are reversed.

OsdSimpleMenu



OsdSimpleMenu derives from OsdBasicMenu and enables you to create a menu more quickly and easily, but at the cost of some flexibility. Unlike OsdBasicMenu, which can have any OSD element added to its menu structure, an OsdSimpleMenu can have only text-based items of a predefined style.

Creating OSD Elements

OSD Elements can be created using the HDK Lua API. OSD Elements are created in a hierarchical structure. When the OSD system is initialized, an abstract top-level element is created known as the root object. A handle for this element can be accessed using the function Osd.GetRoot(). Any OSD element can be created as children of this root object.

⚠ Child objects inherit position alignment (OsdObject.SetPosition()) from their parent, but do not inherit rotation (OsdObject.SetAngle()).

The following is an example of an OSD element being created as a child of the OSD root (in this case, the element is an OsdBasicPopup), followed by an explanation of the example.

```
MyOsdRoot = Osd.GetRoot()

MyPopup = MyOsdRoot:CreateChildObject( "OsdBasicPopup", "mypopup1" )

OsdBasicPopup.SetTitle( MyPopup, "Popup Title" )

OsdBasicPopup.SetMessage( MyPopup, "Message" )

OsdObject.SetWidth( MyPopup, 300 )

OsdObject.SetPosition( MyPopup, OsdAlign.Center, 0, 0, OsdAlign.Center )

OsdObject.Activate( MyPopup )
```

Getting the OSD Root

```
MyOsdRoot = Osd.GetRoot()
```

This line retrieves a handle on the root OSD element and assigns it to MyOsdRoot.

Creating a New Child

```
MyPopup = MyOsdRoot:CreateChildObject( "OsdBasicPopup", "mypopup1" )
```

A new child of MyOsdRoot is created; this child is of type OsdBasicPopup. The variable MyPopup stores a handle for the new OSD element. The name "mypopup1" is the name given to the object in the OSD hierarchy; this allows the handle to be retrieved at any point by calling OsdObject.FindObject(MyOsdRoot, "mypopup1").

Setting Properties

```
OsdBasicPopup.SetTitle( MyPopup, "Popup Title" )

OsdBasicPopup.SetMessage( MyPopup, "Message" )

OsdObject.SetWidth( MyPopup, 300 )
```

These lines set the title, message and width of the object. In this case, the height is left undefined as an OsdBasicPopup automatically adjusts its height to fit its content. Note that the SetWidth() function is inherited from OsdObject. The colon operator can be employed to simplify this, for example MyPopup:SetWidth(300) is equivalent to OsdObject.SetWidth(MyPopup, 300).

Setting the Position

```
OsdObject.SetPosition( MyPopup, OsdAlign.Center, 0, 0, OsdAlign.Center )
```

This function positions an OSD element using a specified anchor point at a given offset to the position of its parent. This anchor point is often referred to as a local-space align point.



In the above picture, the image is the parent element of the text. The following function could be used to set the position of the text:

```
OsdObject.SetPosition( MyText, OsdAlign.Right, 10, 0, OsdAlign.Left )
```

In this example, the parent-space align point is represented by the orange circle and the local-space align point is represented by the green circle. The local-space align point is being offset by 10 pixels on the x-axis from the parent-space align point.

Displaying the Element

```
OsdObject.Activate( MyPopup )
```

This causes the element to be faded into view. To fade the element out of view, call `OsdObject.Deactivate()`. Again, `MyPopup:Activate()` is equally valid.

Blocked OSD Functions

Certain `OsdObject` parent functions are blocked from being used by child OSD objects to maintain functionality of the OSD library and consistency in the OSD output in PS Home.

Below is the format for presenting the blocked OSD functions.

Child Library - The child library that is blocked from using a certain parent function and parent attribute.

`Blocked Function` - The function that the child library is blocked from using.

`Blocked Attribute` (if applicable) - The attribute in the blocked function that the child is prevented from using.

`Alternative Function` (if applicable) - An alternative function that you can use if you want to set other attributes that are controlled by the blocked function.

OsdBasicLabel

`OsdObject.SetHeight` - Labels have a fixed height.

`height` - Labels have a fixed height.

OsdBasicLegend

`OsdObject.SetWidth`- Legend width is set automatically, see `Alternative` functions below.

`font` - Labels have a fixed font.

`topbottomborder` - Labels have a fixed height.

`Alternative: OsdBasicLegend.SetMinWidth, OsdBasicLegend.SetMaxWidth, OsdBasicLegend.SetFixedWidth`

OsdBasicMenu

`OsdBasicChip.CreateContentObject`

`Alternative: OsdBasicMenu.AddItem`

OsdBasicPopup

OsdBasicChip.SetBannerHeadline

OsdBasicPopup.SetTitle

OsdBasicChip.SetBannerSubHeader

OsdBasicPopup.SetMessage

OsdBasicText

OsdObject.SetHeight - Height is determined by the font.

height - Height is determined by the font.

OsdTextLines

OsdObject.SetHeight - Height is determined by the font, number of lines, and line spacing.

OsdSimpleMenu

OsdBasicMenu.AddItem

Alternative: OsdSimpleMenu.AddItem

Create Custom OSD Elements with XML

You can use XML to define the layout of a custom OSD Element. This allows you to combine several OSD elements into one object.

Defining a custom OSD element

Here is an example of a custom object:

```
<OSD_OBJECT_TYPE>

    <OSD_OBJECT type="OsdObject" origin="topleft" xpos="0" ypos="0" align="center">

        <OSD_OBJECT name="icon" type="OsdBasicSingleGraphic" origin="topleft" xpos="0" ypos="0"
align="topleft" color="normal"/>

        <OSD_OBJECT name="title" type="OsdBasicText" text="test" origin="topleft" xpos="75" ypos="24"
align="left" font="18reg" color="black"/>

        <OSD_OBJECT name="description" type="OsdBasicText" text="" origin="topleft" xpos="75"
ypos="40" align="left" font="13reg" color="black"/>

    </OSD_OBJECT>

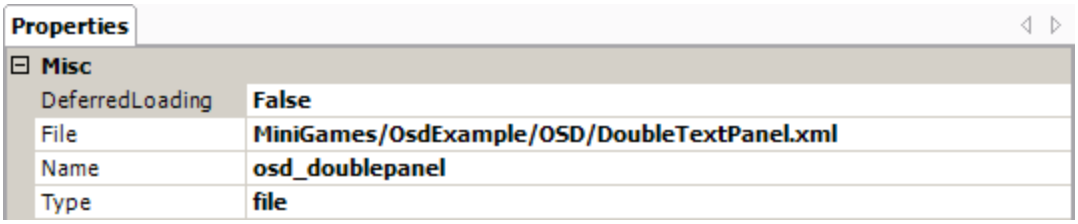
</OSD_OBJECT_TYPE>
```

In the second line, a new OSD Object type is defined; this is the basis for all custom OSD objects. The origin attribute defines the anchor point for positioning the object and the align attribute defines where the positioning origin is in the screen coordinates.

Inside this object type, three elements are defined as children of it, an OsdBasicText element that will be used as a heading, and then two different text panels positioned side-by-side. See OSD Elements for a list of all available XML attributes for each OSD element.

Adding a custom OSD element to an object

Once you have a custom object defined, it must be added to an object (Mini-game, Game Launching Object, etc.) in the Object Editor. The XML file that defines this object should be added to the object as a resource and given a name, for example, osd_doublepanel.



Once this resource is defined inside the object, it must then be referenced in an OSD XML types file. This file links the resource in the object to a Lua type that can be scripted, for example:

```
<OSD_OBJECT_TYPES>

    <OSD_OBJECT_TYPE name="DoubleTextPanel" xml="resource:osd_doublepanel"/>

</OSD_OBJECT_TYPES>
```

This file must also be added to the object as a resource using the Object Editor. Once this file is added as a resource, configure the OSD component of the object to reference this OSD XML types file. Once this is done, the custom OSD object can be used in a script. It is created in the same way as any other OSD element, for example:

```
MyDoublePanel = MyOsdRoot:CreateChildObject("DoubleTextPanel", "mydoublepanel")
```

Once this is done, the component parts of the object can be edited by first finding the element within the object (using the name attribute from the XML), and then setting the text, for example:

```
MyDoublePanel:FindObject("heading"):SetText(Me:GetLocalizedText("DoublePanelHeading"))
```

Other OSD functions can be called on any of your elements by first using `FindObject()` and then calling the function.

Troubleshooting OSD Elements

When troubleshooting OSD elements, the console command `showOsdObjectBounds 1` can be very useful in solving the problem. This command draws boxes around each of the components of an OSD element.



An example of `showOsdObjectBounds` in use.

Common OSD Issues

The OSD element is not displayed

Solution: Ensure that all properties required to view the element are set. Not all properties have default values, for example, if a font and font color is not set, the text will not be displayed. If an element has no width, it may not be displayed.

Adding a legend generates an error

Solution: Ensure that the legend object is wide enough to display the longest legend in each language that needs to be supported. Ensure that legend items are added to the legend individually and not as one long entry. This allows the entries to be spaced correctly and means that they can be cycled through multiple 'pages' if there are too many to fit at the same time. Also ensure that the width of the legend object is set before adding legend items.

OSD Reference

Lua Enumeration to XML String Mappings

Lua functions which take an enumeration as a parameter are mapped to a string value when creating custom OSD elements with XML. The following table shows these mappings.

Lua Enumeration	XML String
OsdAlign.TopLeft	"topleft"
OsdAlign.Top	"top"
OsdAlign.TopRight	"topright"
OsdAlign.Left	"left"
OsdAlign.Center	"center"
OsdAlign.Right	"right"
OsdAlign.BottomLeft	"bottomleft"
OsdAlign.Bottom	"bottom"
OsdAlign.BottomRight	"bottomright"
OsdAlign.LeftRight	"leftright"
OsdAlpha.Normal	"normal"
OsdAlpha.Ninety	"ninety"
OsdAlpha.ThreeQuarter	"threequarter"
OsdAlpha.Half	"half"
OsdAlpha.Quarter	"quarter"
OsdAlpha.Eighth	"eighth"
OsdAlpha.Sixteenth	"sixteenth"
OsdColor.Foreground	"fg"
OsdColor.HighlightFg	"hifg"
OsdColor.Background	"bg"
OsdColor.OppositeBg	"oppbg"
OsdColor.Normal	"normal"
OsdColor.Black	"black"
OsdColor.White	"white"
OsdColor.Grey	"grey"
OsdColor.Green	"green"
OsdColor.Yellow	"yellow"
OsdColor.Red	"red"
OsdMotion.None	"none"
OsdMotion.Snap	"snap"
OsdMotion.ScaleDist	"approach"
OsdMotion.Spring	"spring"
OsdMotion.ConstSpeed	"smooth"

OsdObject

Attribute	Description
type	The OSD element you are creating, e.g. "BasicSingleGraphic"
isactive	If set to true, the element will be activated by default. Equivalent to OsdObject.Activate().
origin	The parent space align point for positioning the element, equivalent to the alignPointOrigin argument of OsdObject.SetParentSpaceAlignPoint().
xpos	The x-axis offset from the parent space align point, equivalent to the alignPointOffsetX argument of OsdObject.SetParentSpaceAlignPoint().
ypos	The y-axis offset from the parent space align point, equivalent to the alignPointOffsetY argument of OsdObject.SetParentSpaceAlignPoint().
width	The width of the element, equivalent to OsdObject.SetWidth().
height	The height of the element, equivalent to OsdObject.SetHeight().
align	The local space align point for positioning the element, equivalent to OsdObject.SetLocalSpaceAlignPoint.
alpha	Sets the alpha transparency level of the element, can be any floating point decimal between 0 and 1.
color	The affected part of this attribute depends on the element being created, but is typically used to specify text color, equivalent to OsdObject.SetColorType().
angle	Sets a rotation angle in degrees, currently only supported by OsdBasicSingleGraphic().
intangible	If set to be true, the element becomes intangible: it has no size, position etc, but child elements can be created beneath it. This is useful when arranging elements in order to create groups. The OSD Root is anintangible OSD element.
autosetvisible	If this is set to be true the object will be set visible/invisible when its parent is set visible/invisible.
uid	Sets the unique ID of the OSD element, equivalent to OsdObject.SetUniqueId().
deactivatedscale	Sets a scale value for the element to be scaled to as the element is deactivated.

OsdBasicText

Derives from: OsdObject

Attribute	Description
text	The text displayed in the element, equivalent to OsdBasicText.SetText().
font	The font of the element, equivalent to OsdBasicText.SetFont().
fitwidth	Determines whether the width of the element is automatically set to the width of the text.
minclip	The "left" clipping value, equivalent to the minClip argument of OsdBasicText.SetClipping(). Clipping is relative to local space align point.
maxclip	The "right" clipping value, equivalent to the maxClip argument of OsdBasicText.SetClipping(). Clipping is relative to local space align point.
scrolltype	Sets the scrolling behavior of the element, equivalent to the scrollType argument of OsdBasicText.SetScrolling().
scrollspeed	Sets the scrolling speed of the element, equivalent to the scrollSpeed argument of OsdBasicText.SetScrolling().
scrolldelay	Sets a delay before the element begins to scroll, equivalent to the scrollDelay argument of OsdBasicText.SetScrolling().

OsdBasicSingleGraphic

Derives from: OsdObject

Attribute	Description
reversecolors	Changes the background color (bg) to the foreground color (fg) and vice versa.

width	Fixes the size of the graphic to the specified width.
height	Fixes the size of the graphic to the specified height.
aspectlocked	Locks the aspect ratio of the graphic.
foregroundalpha	Alpha transparency of the foreground graphic, a floating point value between 0 and 1.
renderactualsize	If set to be true, ignores width/height settings and draws the graphic at 100% scale.

OsdBasicAnimGraphic

Derives from: OsdBasicSingleGraphic

Attribute	Description
fps	Sets the frame rate of the animated graphic, equivalent to OsdBasicAnimGraphic.SetFrameRate.

OsdBasicChip

Derives from: OsdObject

Attribute	Description
size	Sets the chip to a preset size, equivalent to OsdBasicChip.SetSize().
height	Fixes the size of the chip to the specified height.
contentwidth	Fixes the size of the chip's content object to the specified width.
contentheight	Fixes the size of the chip's content object to the specified height.
nobanner	If set to be true, removes the banner from the OsdBasicChip, equivalent to OsdBasicChip.SetBannerWantVisible().

OsdBasicPopup

Derives from: OsdBasicChip

Attributes: None

OsdTextLines

Derives from: OsdObject

Attribute	Description
font	The font of the element, equivalent to OsdTextLines.SetFont().
linespacing	Sets line spacing, equivalent to OsdTextLines.SetLineSpacing().
justify	Aligns text to the left, right or center, equivalent to OsdTextLines.SetJustification().
ignoretails	If set to true, the descenders of characters are ignored when calculating the height of the text.
ignoreaccents	If set to be true, the diacritical marks of characters are ignored when calculating the height of the text.

OsdTextPanel

Derives from: OsdObject

Attribute	Description
font	The font of the element, equivalent to OsdTextPanel.SetFont().
textborder	Specifies the size of the text border in pixels, equivalent to OsdTextPanel.SetTextBorder().
linespacing	Specifies the line spacing of the panel, equivalent to OsdTextPanel.SetLineSpacing().
justify	Aligns text to the left, right or center, equivalent to OsdTextLines.SetJustification().

showpanel	Equivalent to OsdTextPanel.SetPanelVisible().
fitheight	Sets the height of the panel to fit its content, equivalent to OsdTextPanel.SetFitHeightToText().
textcolor	Sets the color of the text, equivalent to OsdTextPanel.SetTextColorType().
scroll	Sets the scrolling behavior for the element, equivalent to OsdTextPanel.SetScrollType().

OsdBasicMenu

Derives from: OsdBasicChip

Attributes: None

OsdSimpleMenu

Derives from: OsdBasicMenu

Attributes: None

OsdBasicLegend

Derives from: OsdBasicLabel

Attribute	Description
font	The font of the element, equivalent to OsdTextPanel.SetFont().
topbottomborder	Specifies the size of the legend border along the top and bottom of the element.
leftrightborder	Specifies the size of the legend border along the left and right of the element.

PS Home Rewards

The Home Reward system allows developers to grant special PS Home objects to users. A Home Reward can be any object that can be published in the live environment, such as furniture, scene entitlements, clothing, and Companion Objects.

You can grant a Home Reward in different situations, such as:

- When a user accomplishes a specific task or goal in an external title. For example, completing the title, finding a hidden area, or just playing the title.
- When a user accomplishes a specific task or goal in PS Home. For example, visiting a PS Home space, or playing or completing a Mini-game in a PS Home space.

The following image shows an example of a Mini-game in progress:



Granting Rewards from an External Title

1. External Title writes a REWARDS.XML file to the PlayStation®3 HDD.
2. Each time the user logs into PS Home, the client searches the PlayStation®3 HDD for newREWARDS.XML files.
3. If any new REWARDS.XML files are found, it parses the file storing the reward information in the client, and deletes the REWARDS.XML.
4. (Automatic rewards only) The reward is immediately added to the user's inventory.
5. (Lua Reward) Lua script needs to issue the reward to the user by calling `Reward.CollectTicket()`.

> ⚠ In cases where you issue a Lua Reward, advise the user within the title of the steps they must take to receive the reward. For example, if a user earns a reward while playing an external title but must go to a certain space in PS Home to receive the reward in their inventory, they need to know. If the external title does not inform the user that they that space before to receive the reward, they will not know why the item does not appear in their inventory after logging into PS Home.

Granting Rewards from PS Home

1. Lua script writes a reward ticket by calling `Reward.AddTicket()`.
2. (Automatic rewards only) The reward is immediately added to the user's inventory.
3. (Lua Reward) Lua script then issues the reward to the user by calling `Reward.CollectTicket()`.

> ⚠ If you issue a Lua Reward, advise the user within the title of the steps they must take to receive the reward.

Guidelines

- An object cannot be both a reward and an object available for purchase in a commerce point, in the same region. For example, if a shirt is a reward item in SCEE, it cannot also be a purchase item in SCEE. It can, however, be a reward in SCEE, and available for purchase in SCEAsia.
- For external title rewards:
    - You can grant a maximum of five rewards per brand or title. For further clarification on this figure, contact your Home Business Manager.
    - The age rating of a reward must not exceed the age rating of the game.
- When developing and testing Home Rewards, you must run PS Home in Online Mode. You can control this through the Scene Editor's Launch Options dialog, or by launching HomeDeveloper.self with the `--sceneId` command line option with the Target Manager.
- Do not issue rewards for items that are not yet released in PS Home.

Rewards Entitlement ID

> It is assumed that you have read the Object Editor documentation and you have an understanding of how objects are built.

For items that are themselves rewarded, you must configure an entitlement_id in the metadata of the object. This entitlement ID specifies (per region, as there may be different requirements regionally) whether the object is awarded automatically (AUTOMATIC_REWARD) or is rewarded and must be redeemed by further action in PS Home (LUA_REWARD).

The process follows two stages: the issue of a reward ticket and the exchange of this ticket for a reward. The automatic reward function completes these two stages immediately, whereas the Lua reward function completes the second stage (the exchange of the ticket for the reward) upon further user action (for example, going to a specific space to collect the free t-shirt for completing all the levels in an arcade game).

> Keep in mind that the automatic reward function should be used only after careful consideration. To make the best use of travel to various locations in PS Home and to avoid users feeling 'spammed', there are several instances where delaying exchanging tickets for rewards is preferable to automatic redemption of rewards.

Awarding Reward Tickets

The Home Rewards system maintains a list of reward 'tickets'. These tickets indicate to Home client the list of Reward items that the player has access to. Each ticket contains the unique Home ObjectID specifying the item to be awarded (which may be a clothing item or a furniture item or indeed any new object types that are made available in the future).

There are two methods for adding items to the Reward Ticket list:

- From an external title, by writing out a REWARDS.XML file to the PlayStation®3's HDD, within a folder relating to the external application's TitleID.

  > The REWARDS.XML filename must be in upper case.

- Through a LUA script call within Home client.

Awarding Tickets from a Title

When a title awards the user a Home Reward, it should write out a file called REWARDS.XML (note that the upper case of this name is mandatory). This file must be written to the directory on the PlayStation®3's hard drive returned by the following function:

```
#include <sysutil/sysutil_game_exec.h>

int cellGameGetHomeDataExportPath(

      char *exportPath

)
```

For more information on this function, see the Cell SDK documentation.

The export path should be of the following form:

dev_hdd0/game/[HOME_TITLE_ID]DATA/USRDIR/IMPORT/[LOCAL_USER_ID]/[TITLE_ID]

Where:

- HOME_TITLE_ID is the title ID of Home.
- LOCAL_USER_ID is the unique identifier associated with the user to whom the reward was granted.
- TITLE_ID is the title ID of the title calling the above cell function.

As you can see from the format of the above path, each title is responsible for managing its own REWARDS.XML files (where each user can be allocated one per title). Therefore, it is not possible for titles to overwrite each others' files.

⚠ Please note:

- Do not export any other files to this export path. Only export the REWARDS.XML file.
- The function cellGameGetHomeDataExportPath only behaves as described above when it is called from an executable run from an installed package. If the function is called from an executable that was launched from Target Manager, the returned path will instead be of the following form: dev_hdd0/game/[HOME_TITLE_ID]DATA/USRDIR/IMPORT/[LOCAL_USER_ID] This is because the trailing title ID is queried from the executable's package and a SELF run via Target Manager has no package. During development, you are advised to append your title ID to the returned path. Once packaged, your executable can use the returned path unmodified.
- Any information of Home rewards must be prevented from being transferred between users (via savedata for example).
- If REWARDS.XML cannot be output because cellGameGetHomeDataExportPath() returns an error or there is not enough available space, display an appropriate error message. (See PlayStation®3 TRC number 34.)

To open a REWARDS.XML file for writing, you would do something similar to the following:

```
char rewardsFilename[CELL_GAME_HDDGAMEPATH_SIZE + 12];
int ret = cellGameGetHomeDataExportPath(rewardsFilename);
if (ret != CELL_HDDGAME_RET_OK)
{
      printf("Error - Could not obtain export path.\n " );
      return false;
}
strcat( rewardsFilename, "/REWARDS.XML" );
int fd;
CellFsErrno err = cellFsOpen( rewardsFilename,
      CELL_FS_O_WRONLY|CELL_FS_O_CREAT, &fd, NULL, 0 );
if (err != CELL_FS_SUCCEEDED)
{
      printf("Error - Could not open file to write.\n " );
      return false;
}
```

Specification of REWARDS.XML

The REWARDS.XML that your title writes should adhere to the following rules:

- It must contain <REWARDS> and </REWARDS> tags.
- The VERSION attribute of the <REWARDS> tag should be ="1".
- Each reward ticket must have <REWARDTICKET> and </REWARDTICKET> tags.
- Any reward tickets without at least one <OBJECTID> child element will be ignored.
- Do not use any other tags.

An example REWARDS.XML is shown below:

```
<REWARDS VERSION="1">

   <REWARDTICKET>

      <OBJECTID>749BAC2A-CE7042D6-871380C5-038FE826</OBJECTID>

   </REWARDTICKET>

   <REWARDTICKET>

      <OBJECTID>54097ED3-97EE400D-9600271A-263D093C</OBJECTID>

   </REWARDTICKET>

</REWARDS>
```

What Happens when Home Client Reads Your REWARDS.XML

When Home client boots, it scans the PlayStation®3's HDD for REWARDS.XML files. If it detects a new REWARDS.XML file, it will

parse and then delete the file. Each reward ticket parsed will be added to a separate internal list stored by Home client.

> If Home client encounters a REWARDS.XML file that is larger than 50KB in size, it will not be parsed, but will be immediately deleted instead. Since you are limited in the number of rewards you can grant a Home user, it is highly unlikely this eventuality will ever occur.

There may be cases when the external title needs to add to an existing REWARDS.XML (i.e. if the user is granted Home Rewards in multiple sessions of the title, without running PS Home in between). The external title should correctly read and amend an existing REWARDS.XML to the correct format in those cases.

Awarding Tickets from PS Home

The Lua call for adding 'Reward Tickets' is:

```
Rewards.AddTicket( string objectID )
```

for example:

```
Rewards.AddTicket( "11F21642-974E4FD5-BC4E1EE7-2B28A905" )
```

This system allows PS Home Mini-games, Realtime games, arcade games, other objects or scenes to award rewards to the player.

For example, get a high score on an arcade game and receive a new Personal Space in return.

For example, visit a scene and receive a free clothing item.

Awarding Tickets for Unpublished Content

Be wary of publishing content that gives rewards for content that has not been published. The reward object cannot be issued if it does not exist in the client. The client's reward ticket list is regularly overwritten, which means that after the reward object has been published, the user will then need to repeat the action/event that espoused the reward ticket.

How Can You Reward Unpublished Content

This can happen if a PlayStation®3 title is released before the reward objects are published in the Home client.

It can also occur with Home content and Home reward objects. For example, even if the two are submitted simultaneously the content may make it to the live client, yet the reward object may not at the same time if it does not meet QA requirements.

What Happens to Unpublished Reward Tickets

For PlayStation®3 titles:

PS Home searches the HDD for REWARDS.XML files when it starts. It passes the REWARDS.XML data to the reward ticket list in the Home client, deleting the REWARDS.XML files in the process. It then gives the user the rewards. The next time the user receives a reward ticket, either in PS Home or in a title, the reward ticket list is overwritten, deleting all record that the user should receive the unpublished reward once it is released into the client.

The user will then need to repeat whatever action/event they did in the PlayStation®3 title to earn the reward ticket, and thus the reward.

For Rewards Issued by Home Content:

When the user earns the reward in PS Home, for example, by completing a Realtime game, for an unpublished item, the reward ticket is saved on the client's reward ticket list; however, the next time that the user earns a reward, the reward ticket list is overwritten, deleting any record that the user was supposed to have received the reward once it is published in the live client.

The user will then need to repeat whatever action/event they did in PS Home to earn the reward ticket, and thus the reward.

Workaround

Unfortunately, there is no easy workaround that allows you to store reward tickets for unpublished items.

You can use the NP Ticketing service to store the IDs of those users who received the reward before it is published on an external server. This, however, is not ideal, and can be messy:

- Even after the reward has been published, the deserving user will not automatically receive the reward.
- Instead, the user will need to access content in PS Home that accesses your server to query if the user received the reward ticket, and then reissue the ticket.

- After the reward has been published, the need to check servers to see if a user has received the reward ticket but not the reward will diminish, eventually becoming a needless check and waste of resources.

> ⚠ Avoid releasing content that issues reward tickets before the reward has been published in PS Home.

Collecting Reward Tickets

Awarded tickets need to be converted into Home objects added to the user's inventory. For example, if a ticket corresponding to a t-shirt is collected, the t-shirt is immediately available in the user's wardrobe.

The User Inventory

The inventory lists the Home objects that are owned by the user. Reward tickets can only be successfully granted to the user if the corresponding Home object is not already in the user's inventory. There are a number of Dev Debug Console commands which allow you to inspect and manipulate the inventory:

Command	Description
`inv list user`	List all object IDs of objects in the user's inventory
`inv adduserobj <OBJECTID>`	Add the specified object to the user's inventory
`inv removeuserobj <OBJECTID>`	Remove the specified object from the user's inventory
`inv clearuser`	Remove all objects from the user's inventory

> ⚠ You will not be able to use `adduserobj` in the hubstartup.txt in Target Manager.

For more information on using the Dev Debug Console and for a reference of useful commands, see The Debug Console.

Home rewards will appear in the user's inventory, as well in the user's Menu Screen > Options > Personal > My Rewards. Reward objects (including Scene Entitlement Objects) have the same name, description, publisher, etc. requirements as any other object.

The Test Inventory

For development purposes, the HDK provides something called the test inventory. This is an XML file - `<HDK_INSTALL_DIR>\build\TestInventory.xml`. When exporting items such as clothing, they will be added to the test inventory. When HomeDeveloper.self is run, it will read the test inventory and add the items it finds to the user's inventory. If you want the item of clothing to be a reward, though, you may not want this behavior. To avoid the object being added to the user inventory, you can either remove it using the commands above, or you can edit TestInventory.xml and remove the corresponding object ID.

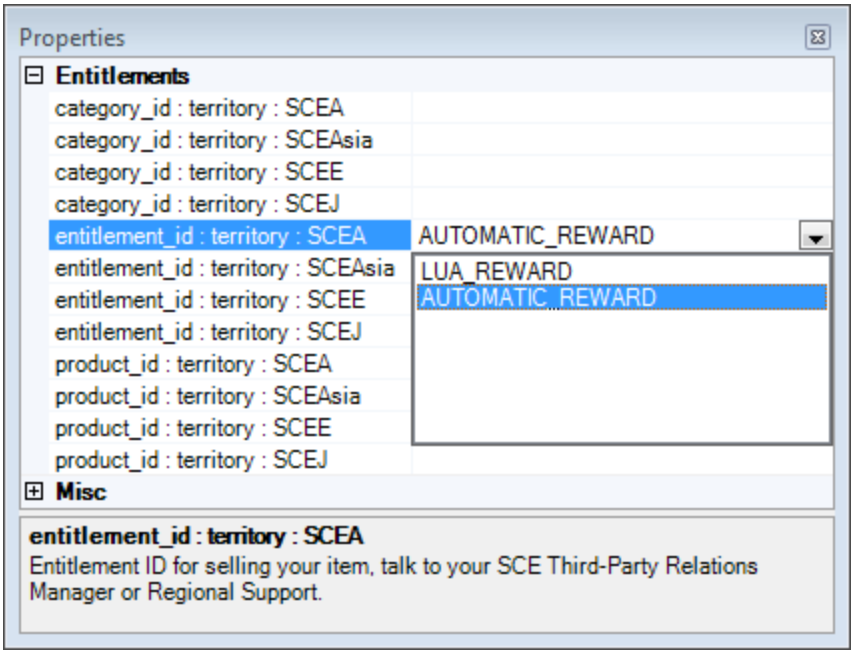
Collection Methods

> ⚠ For a reward ticket to be successfully collected, you need to run your scene in online mode.

There are two methods for converting reward tickets into Home objects:

- Automatic collection
- Lua scripted collection

The method for collecting an item is specified on a per-region basis in the object's properties. Select the reward object in the Object Editor and view its metadata properties in the Properties panel. You can determine how the object is collected using the `entitlement_id` fields in the Entitlements section:



Items that can be awarded cannot be available for purchase.

> The entitlement_id region used is always the region of the PSN account - traveling to a different region's lobby does not affect how and if particular items can be awarded.

Automatic Rewards

When a reward ticket links to an object which has this entitlement ID, the item is automatically added to the user's inventory without any further intervention. Obviously, use of this entitlement ID needs to be carefully considered to avoid too many items being added to the user's inventory and being considered 'spam'.

The script should be:

```
--Give Reward Item

if Rewards.AddTicket(objId) then
....
```

Errors in Manually Collecting Automatic Rewards

Do not call Rewards.CollectTicket() after granting an Automatic Reward to a user. This will result in the following error:

```
WARNING: RewardTicket collection attempted with invalid method
```

Lua Rewards

Having this entitlement ID means that the item is collected only when a Lua script (either running on a scene or as part of another object) calls a Lua function to award the item.

> If an external server is used to host reward information, it must be a secure server using HTTPS.

Here is a simple script that awards an item when the user walks up to a particular screen in the world:

```
pos = LocalPerson.GetPosition()
screen = Scene.FindScreen( "AwardVideoScreen" )
screenPos = screen:GetCenter()
screenToPlayer = Vector4.Create()
screenToPlayer:Subtract( screenPos, pos )
-- Get distance of player from video screen
dist = screenToPlayer:Length3()
-- check distance of player from screen
if ( dist < 12.5 ) then
      if ( hasAwardedItem == 0 ) then

         hasAwardedItem = 1
         url = "https://myserver.com/ItemAwardedPage.hsml"
         screen:SetContentSource( url )

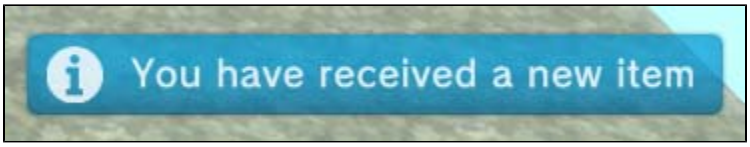
         objectID = "00000000-000000001-00001000-00001235"
         Rewards.CollectTicket( objectID )
      end
end
```

The `Rewards.CollectTicket` call would do nothing unless both:

1. The user already has the corresponding ticket in the Reward Ticket list
2. The object's metadata properties has an appropriate `Entitlement_id` field with `LUA_REWARD` selected.
3. The user already has the object.

Repeated calls to an item already in the user's inventory would be ignored.

On successfully collecting a reward, PS Home notifies the user with the following message:



The newly awarded Home object should now be accessible. If the object was an item of clothing, it will now appear in the Wardrobe. If the object was an item of furniture, it will now appear in the furniture browser.

Commerce in PS Home

Content in PS Home is sold through commerce points which are virtual stores. PS Home users are already able to purchase items from the shopping mall through various commerce points, such as the Threads store within the shopping mall. You can make your content available for sale and use a commerce point to sell content from your own branded scene.

There are regional variations in commerce that you should talk to your Home Account Manager about. This section covers the basic requirements, but the exact process is different for each region. If you want to use commerce in more than one region then you need to fulfill the requirements of each region.

You can develop the following content for sale in PS Home:

Content	Description
Objects	Clothing, furniture, active items, mini-games, realtime games within a scene or attached to active items, arcade-games as furniture, companion objects.
Rental and Consumables	Objects that have a limited use time or amount of uses. They can be any of the scripted objects above, or they can be stand-alone objects that are redeemed for access to scripted Home content.
Scenes	Personal spaces, clubhouses.
PSN products	Game titles.
Bundles	Items grouped together and sold as one commerce item.

There is content available for sale within PS Home that can only be developed internally. We will be making more content available for sale within commerce points in future releases.

Developing and Publishing Commerce Content

> ⚠ Due to increased internal security, you must make a request to your region to access commerce items in your sandbox. To do this, raise an issue and specify the Sign-in ID and IP address range for those who wish to access commerce items.

When you have created objects, the publishing process can take around eight weeks to complete, including all Quality Assurance (QA). The time required depends on the nature of your content, and on regional requirements. Your content requires functional, localization and commerce testing, as well as time for the retail data to be set up in PS Home and PlayStation®Network.

To develop content for sale in PS Home:

1. Discuss your requirements and the content you want to sell with your Home Account Manager. For example, discuss:

    - The number of objects and scenes (personal spaces) you want to develop.
    - The type of content and how it should be grouped with other content. Grouping determines how the user selects your content in the PlayStation®Store.

      You can group items as a bundle to be sold together as one commerce item, or sold separately but listed in a group (such as a chip that represents a single costume: when the user clicks on the chip they can view and purchase components of that costume).

You can also design full outfits. See Composite Character Components.

1.
    - Commerce point in which the content is sold (for example, threads in the PS Home shopping mall or your own commerce point - see below).
    - Commerce point in which the content is sold (for example, threads in the PS Home shopping mall or your own commerce point - see below).
    - Regions and languages in which the content is sold.
    - If any objects are to be rentable (they expire after a certain time) or consumable (they can only be used a set number of times - see below).
    - We supply a Service ID on request to all registered PlayStation®3 and PS Home developers for each region. If you do not already have one, you must obtain a new one. This is used for financial tracking and can take up to a week to set up. For 3rd parties there may be some documentation to be completed as part of this process. Contact your PS Home Business Manager for details.
    - Broad pricing needs to be agreed. Discuss with your PS Home Business Manager.

2. Develop content and upload it to your sandbox for your QA process using the Content Delivery System (CDS). Each object/scene needs the following:

    - Thumbnails
    - Video (if included)
    - Age rating (not applicable for all SCEE commerce items, but necessary for scenes). SCEE require the Minimum Age field in the Object Age Rating dialog of the Object Editor to be completed (otherwise the object won't export).
    - Localization

3. Send a list of your new objects to your Home Account Manager with the following information:

    - Object ID (created when you export objects)
    - Product name in all supported languages (e.g. EFIGS)
    - Age rating (not applicable for all SCEE commerce items, but necessary for scenes)
    - Localization

4. Bundles require a dummy commerce object (or Scene Entitlement Objects for spaces) associated with them. This enables them to be purchased from a commerce point. For bundles, you need to create an object with no model file (so it is not added to the user's inventory).

    > ℹ Commerce objects display a 3D preview of the model for the user when they view that product on the commerce point. If the object has no model file (for example, a dummy object for a bundle) then the thumbnail is displayed instead.

5. To create a dummy object for a bundle:

    a. Create a new object in the Object Editor.
    b. Do not add any resources or components, but leave the object blank.
    c. Add thumbnails, entitlement data, localization information and age ratings as you do for a normal object. The description, information and thumbnails should be for the bundle and inform the user about the content contained in the bundle (including how many items are included and the type of objects).
    d. Save the object.

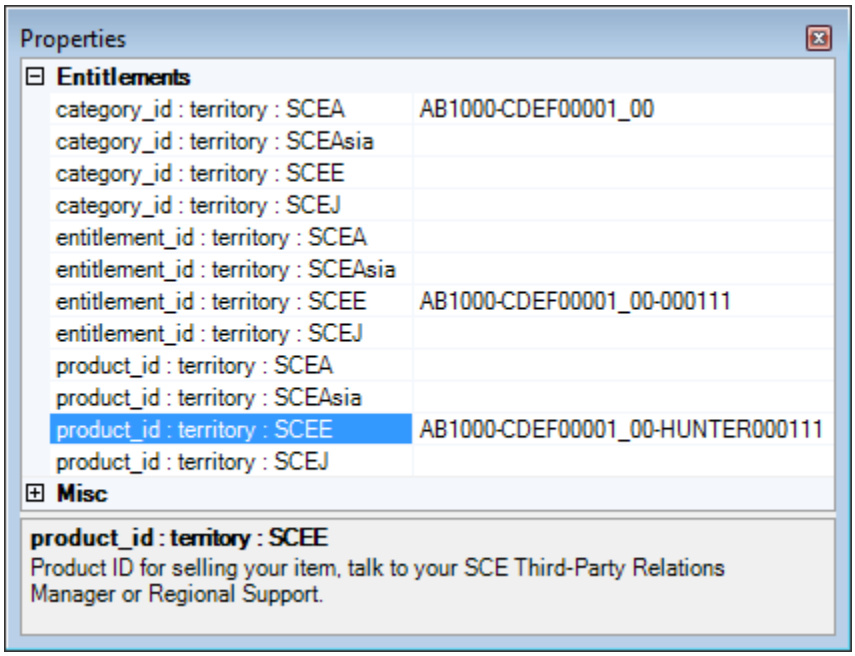
> For spaces, create a scene entitlement object, (see Scene Entitlement Objects) and include the [#Legal] line (see below)

6. If you are developing an object, contact regional SCE to obtain the following for your content:

    - Category ID
    - Entitlement ID
    - Product ID

    Objects sold in more than one region require category, Entitlement ID and Product IDs for each region.

7. Use the Object Editor to add the category ID, Entitlement ID and Product ID within the metadata of each object:



> Your object description text must contain the following tag at the end of the text (this is case sensitive): [#Legal]. This tag ensures that the correct regional legal text is displayed at the end of your description within the PlayStation®Store. The legal text is not displayed elsewhere, after the item is purchased.

Without the [#Legal] text in your object, the content will fail QA and will not be published.

For scenes, add the description to the Scene Entitlement Object, not the scene itself. Scenes are not sold directly to users, so scene descriptions do not appear in the PlayStation®Store.

1. If you are developing a scene, raise an issue with regional support to request the following:

    - Scene Key
    - Scene Name

2. Resubmit your content to your sandbox (using the CDS) and test it against your test commerce point to ensure it can be purchased successfully.

3. For SCEE developers, the commerce point should be tested locally.

    Use the console command `useOfflineContent 1` to enable the use of the items in your `OfflineContent` folder in your HDK installation folder.

    Use the debug console command `Homestore.Open <XML filename>` to open the test commerce point.

    The commerce point XML must be located in your build folder:

    `<HDK_Install_Dir>\build\OfflineContent\CommercePoints`

All your objects must already be set up in the PlayStation®Network at this point in order to test the purchasing of them.

4. For all developers apart from SCEE, the commerce point is tested online - your Home Account Manager will provide details.
5. When you are satisfied with the functionality and quality of your items, follow the standard procedure for publishing. Once the content is approved for publishing and goes live, it becomes available for sale at the commerce point you requested (for example, threads or your own commerce point - see below).

> PlayStation®Store data is region-specific and is generated by each target region. To release items to a new region after an initial release, the items need to be republished according to the CDS guidelines (for example, selecting all the relevant regional boxes). It is therefore better to agree with the regions in which you wish to sell early on, so that all this data can be entered together.

Creating a Commerce Point

For Japanese Live Page
最新の日本語版ページはこちら 「コマースポイントの作成」

The Home Account Manager creates the commerce point XML file for you, based on the information you provide about the content you want to sell. This includes a list of items to sell, the grouping of those items, the video (if included) and the background image for the commerce point. You also need to decide how the commerce point is opened and then implement it within your scene or object (if needed).

You can open a commerce point in one of the following ways:

- By adding a trigger to a scene

  If you already have a published scene or have one in development, the trigger is added in the Scene Editor and all you need is a URL to your commerce point. The trigger handles all user interaction and is best used when you want to add a commerce point for one region only.

- Using the Lua API function **`System.CommercePointOpen`**

  Use the function in a Lua script (along with related functions) to open the commerce point. The function can be used within a Lua script in the following:

  - Mini-game/realtime game as an embedded object
  - Mini-game object added to a scene
  - active (with mini-game/realtime game component)

    This is best used when you want to make the commerce point available in different regions (because you need a separate commerce point for each region).

    > A MiniGame or Realtime Game component is required when using this function to provide a user interface.

- From the Navigator

  Enter the commerce point directly from the Navigator. Contact your Home Account Manager to discuss this option.

PlayStation®Store Logo

When providing access to the PS Home commerce system (and therefore to the PlayStation®Store) the area where the commerce system can be accessed must display the PlayStation®Store logo. This applies to any PS Home content, including commerce points in spaces, actives, mini-games, realtime games, game launch objects.

The logo should be placed within the physical 3D space (not in the interface) and can be integrated into the specific context, as long as it is still recognizable as the PlayStation®Store logo (for example, big enough, not obscured, prominent enough, etc.).

For example:

- In a space with a medieval castle theme, an area in the space where the commerce point can be accessed could be indicated by a tapestry hanging on the wall depicting an embroidered and stylized PlayStation®Store logo or a heraldic device on the wall with a stylized PlayStation®Store logo.
- In a custom Lua game launching object that has its own user interface menu system, an option on the menu where the commerce point can be accessed could be indicated by a PlayStation®Store logo.
- In an active item with a shopping catalogue theme, where interacting with the active item will initiate the PS Home commerce system, the chair itself could have the PlayStation®Store logo as part of the upholstery design.

Contact your Home Account Manager for information on the PlayStation®Store logo data. The logo files are available on the PS Home Developer Network or the PS3 Developer Network in the package SCE_logo.zip.

Developing a Commerce Point

To develop your own commerce point and make content available for sale:

1. Discuss your requirements and the content you wish to sell with your Home Account Manager. You need to agree on the following:
    - Legal requirements
    - Number and type of objects and scenes (apartments) for sale
    - Regions and languages in which the commerce point is available

    > If you want to make the commerce point available in more than one region, then it is best to use the Lua function `System.CommercePointOpen`. It is, however, possible to open a different commerce point for each region with a trigger by creating a separate scene for each region. Set the trigger (that points to the commerce point) to the relevant region in each scene - see below for details on making the trigger.

    > Because the only difference within the scene is the trigger, you can simply make a copy of the scene for each region and change the trigger. However, you will need a different scene key and scene name for each scene.

2. Send a list of the objects and scenes that are to be sold from your commerce point (where appropriate) to your Home Account Manager. This should be a list of object/scene IDs and object/scene names and any grouping required. You also need to send the thumbnails for branding and categorization within the commerce point.
3. If you want to have a unique image in the background of the PlayStation®Store, or define the color of the chip border/text then send the following to your Home Account Manager:
    - Background image: This should be a maximum of 720 pixels height by 1280 pixels width in resolution and either .png, .jpg or .DDS format. Ensure there is nothing vital on the left and right edges of the image because users with a Standard Definition video resolution will have those edges cropped. If using DDS, ensure they are generated without any mipmaps, as this increases the client memory usage and may cause a significant memory issue.
    - OSD border/text color: Choose the color of the border of the chip in which the item is displayed in the PlayStation®Store - this also changes the color of the text. The following options are available:
        - Grey
        - Black
        - White
        - Green
        - Red
        - Yellow

Adding a Trigger to a Scene

To add a commerce point trigger to a scene and allow users to access the commerce point by targeting a trigger, follow these steps:

1. Develop your scene and upload to your sandbox for your QA process using the Content Delivery System (CDS). The scene must have the following:
    - Thumbnails
    - Age rating for each region (not applicable for SCEE)
    - Localization
    - Path and filename for the commerce point XML file. For the commerce point to work you must reference the XML file in the scene. Obtain this information from the Home Account Manager.
    - Scene key and scene name for each scene. Raise an issue with regional support to request them.

2. Contact your Home Account Manager to obtain the following details for your content:
    - Path and filename for the commerce point XML file. Reference this in the scene so that the commerce point functions correctly.
    - Scene key and scene name for each scene.

3. In the Scene Editor, add either the Change Event Trigger or Change Event Box Trigger object to your scene. This sets the area within which a user accesses your commerce point.
    For more information on Targeting, see Targeting System.
    For more information about Trigger properties, see Scene, Object and Asset Properties > Trigger Properties.

    Change the trigger object's properties as follows:
    - Select CommercePoint from the Action Name drop-down list.
    - Enter the path and filename for the commerce point XML file (as provided by your Home Account Manager), in the

Parameter 1 field, for example XFORM_Lounge_2_9KEL_1N8L/A/CPAXFL00.xml.

4. Resubmit your content to your sandbox (using the CDS) and test that objects can be purchased from the commerce point.

5. For SCEE developers, test the commerce point locally.
   - Use the console command useOfflineContent 1 to use the items in your OfflineContent folder in your HDK installation folder.
   - Use the debug console command Homestore.Open <XML filename> to open the test commerce point.
   - The commerce point XML must be located in your build folder: <HDK_Install_Dir>\build\OfflineContent\CommercePoints

6. For all developers apart from SCEE, the commerce point is tested online - your Home Account Manager can provide details. Follow the standard procedure for submitting content to PS Home. After the content is approved for publishing and goes live, your commerce point is available to users.

Open a Commerce Point within a Lua Script

To open a commerce point within a Lua script, first contact your Home Account Manager to obtain the following information for your content:

- Path and filename for the commerce point XML file for each valid region. For the commerce point to work, you must reference the XML file in the function.

Within your Lua script, include the following:

- Define the URL for each valid region.
- Find the territory of the PSN account for the local player.
- Check that there is a valid commerce point for the local player's territory.
- Check that the commerce point is not already running.
- Launch the commerce point using the appropriate URL.

The following sample code shows how you can complete these checks and launch a valid commerce point.

You can reuse this example in its entirety by replacing the URLs with your own. In this example there is no commerce point for SCEASia so it has been set to nil. In your script, use nil for any regions that do not have a valid URL.

```
tab =   {   SCEE    = "XFORM_Lounge_2_9KEL_1N8L/E/CPEXFL00.xml",
            SCEA    = "XFORM_Lounge_2_9KEL_1N8L/A/CPEXFL00.xml",
            SCEJ    = "XFORM_Lounge_2_9KEL_1N8L/J/CPEXFL00.xml",
            SCEAsia = nil
        }

LoadLibrary("System")
LoadLibrary("MiniGame")
LoadLibrary("LocalPlayer")
LoadLibrary("Object")

MiniGame.DisableWelcomeMessages()

local territory = LocalPlayer.GetAccountTerritory()
local commercePoint = nil
if territory then
    commercePoint = tab[territory]
    if commercePoint then
      Object.GetMe():EnableTargeting(true)
    else
      print("Sorry this is only available to the following regions:")
      for i,v in pairs(tab) do print end
    end
else
    print("I can't find your territory ID, this is probably because you are offline")
end

function OnLocalPlayerJoin()
    print("Opening Commerce point: "..commercePoint)
    local result = System.CommercePointOpen(commercePoint)
    if result == -1 then
      print("Commerce point: "..commercePoint .. " is already running.")
    end
    MiniGame.LocalPlayerLeaveGame()
end

function OnCanLocalPlayerJoin()
    return commercePoint ~= nil
end
```



To test just this script, you can use it within an embedded object or a mini-game object added to one of your test scenes.



If you open a commerce point in your script and you want to call a function when the user has left the commerce point (the commerce point is closed), use the `System.CommercePointIsClosed` function to check that the commerce point has closed.



Commerce points in Active Items that do not have a mini-game or realtime game component are not supported. Do not use the `System.CommercePointOpen()` function in any actives.

Testing Online

To use the Lua function `LocalPlayer.GetAccountTerritory`, the user must be online. Unlike most commerce point testing, you cannot check Lua scripts that use this function locally.

To testing scripts online that use `System.CommercePointOpen` you must also have a valid commerce point online for the required regions. Contact your Home Account Manager for more details.

Types of Commerce

Quick Purchase supports all types of commerce.

PSN Products

You can request that existing PSN products should be made available through a commerce point. This allows users to go seamlessly from experiencing PS Home content to buying the related PSN Product (for example, a game title).

If you have them available, simply send the following to your Home Account Manager:

- Product ID
- Category ID

Permanent Object

A permanent object exists in a user's inventory permanently. When a user acquires a permanent object, the user owns that object, unlike consumable and rental objects which exist in a user's inventory for a limited amount of uses or time.

Permanent Object Limits and Restrictions

Any type of Home content can be sold in a commerce point as a permanent object. Scenes cannot be sold directly, but scene access objects can.

Consumable Object

Consumable objects are objects that can be used only a finite number of times. When the charges on a consumable object reach zero, the consumable object is removed from the user's inventory.

Scripting Consumable Objects

There are two ways to consume a unit from a consumable object:

- Using the Lua AsyncCommands library
- Using the Lua Commerce library

The Commerce library synchronously consumes a unit from a consumable object, meaning that all other script processes will wait for a commerce request to complete. The AsyncCommands library is part of the Network Platform (NP) of functions. It asynchronously executes requests, and has read/write access to user entitlements. For more information on consumable objects and using the AsyncCommand function, see NP Entitlements.

We recommend using the AsyncCommands library rather than the Commerce library.

Consumable Objects Limits and Restrictions

- Scenes cannot be consumable objects.
- Only scripted objects can use consume units from a consumable object.
- When an object consumes a unit, check that the user received what they expected to receive (access to a game, an object, etc).
- The relationship between pricing and the integer (used to set the usage) needs to be communicated to your Home Account Manager when setting up a consumable item. For example, $10.00 purchases 10 units and one usage consumes 2 units - this would mean that the object expires after 5 uses.
- If a user is purchasing a consumable object, they must be made aware that it is a consumable object and how many units come with the purchase.
- If a user is about to consume a unit from a consumable object they must be made aware of the cost and how many units they have remaining in that consumable object.

Rental Items

Rental objects are similar to consumable objects, but instead of having charges, a rental object is valid and in a user's inventory, for a limited amount of time. The expiry of rental items is handled by the PlayStation®Store.

In addition to the usual information and process of creating a commerce item, also provide the expiry duration to your Home Account Manager to create a rental object.

Rental Items Limits and Restrictions

- Scenes cannot be rental objects.
- If a user is purchasing a rental object, they must be made aware that it is a rental object and when it expires.

Quick Purchase

Quick Purchase enables users to purchase an object faster and more easily than through the full commerce point system.

To speed up the purchasing process, you can:

- Script objects to bypass most of the Commerce Point steps, taking the users straight to checkout without needing to first exit the scripted situation (such as a game). Users arrive at the checkout with one or more objects already added to their basket. Quick Purchase supports merchandising models like freemium or micro-payments.

> ⚠ Scripts that use Quick Purchase must ensure that the PlayStation®Store logo and prices for the items being Quick Purchased are displayed.

- Select a minimal subset of service IDs, rather than selecting all of them whenever a purchase is made. To use this option, you must set up the products and Stock-Keeping Units (SKUs) so that only entitlements under the same service ID are granted when purchasing a product.

With Quick Purchase, only the object name is displayed in the checkout basket. You must provide players with full information on any object they are purchasing, if they want it. To make sure that players can get information about exactly what they are purchasing, follow these guidelines:

- Provide a full object description.
- Make it clear to the user whether the item is consumable or rentable.
- Explain exactly what users are purchasing: state whether the item is the object they selected for Quick Purchase, or whether the object just allows them to purchase related things (like more levels to a game).
- Make sure that the appropriate legal text is displayed prior to displaying the commerce point. Normally this includes the PlayStation®Store logo and how much items cost.

> ℹ The following Lua functions may be useful when using Quick Purchase:
>
> - Resource.GetSystemResource - Retrieves the PS Store logo.
> - AsyncCommand AsyncCommand.Create(AsyncCommands.CommerceGetProductInfo) - Retrieves product information

The technical requirements for quick purchase (including rentable and consumable items) are:

- PS Store Logo
- Publisher (used to be Maker Name)
- Item Description
- A thumbnail
- Price
- Legal text

> ℹ If you are planning to use the Quick Purchase, contact your regional SCE for more detailed guidelines and advice on restrictions.

Example

The following example describes a scenario where you are making a quest game and uses the Quick Purchase on two types of content:

- The first type of content is for several bottles of potion that the player can buy throughout the levels. Each potion can be used five times.
- The second type is a special scripted familiar that is accessible after completing level 1. This scripted familiar also allows players to purchase levels 3-10.

In both cases, you must make sure that the player can Quick Purchase the objects in as few steps as possible, but you must also add an area that provides more information about what they are buying should they want to it.

For both the potions bottles and the scripted familiar, you must provide a description of the type of content that will be bought and any legal text required:

- For the potion bottles, you must tell players that they are buying a potion bottle that can be used only five times.
- For the scripted familiar, you must tell them that they are buying further levels of the quest game (levels 3-10), which they then own outright,. You might also tell them that if they do not purchase the game just yet, they can summon the scripted familiar anytime in the game and still be able to purchase the full quest game.

The following code provides a sample Quick Purchase script:

```
checkoutCommand = AsyncCommand.Create( AsyncCommands.CommercePointAuto )

if ( not checkoutCommand:Execute( "TestCommercePoint.xml", false, { "TestNodeId" } ) )


      print( 'Commerce system busy' ) - retry here if desired


end

if (checkoutCommand:IsFinished() and checkoutCommand:Succeeded()) then


      if (checkoutCommand:GetResults() == AsyncCommercePointResult.CommercePointOk) then


            print("Checkout success!")


      else if (checkoutCommand:GetResults() ==
      AsyncCommercePointResult.CommercePointCancelledByUser) then


            print("User cancelled the checkout process.")


      end


end
```

Get Price

A useful Lua function for obtaining the price of an item is AsyncCommands.CommerceGetPrice.

```
boolean AsyncCommand.Execute( AsyncCommand getPriceCmd, string|table itemId )
string, number AsyncCommand.GetResults( AsyncCommand getPriceCmd )
```

Either an object ID or category ID/product ID pair is accepted.

The result is both a localized and formatted string representing the price (for display) and a numerical value (for sorting, comparison etc).

Get Thumbnails and other Scene Information

There are some useful Lua functions that get information about scenes, for instance a Lua call to retrieve the URL of a specific scene's large thumbnail image, and the description text, maker name and other relevant information associated with it - the async command SceneInfoRetrieve:

```
AsyncCommand Create( AsyncCommands.SceneInfoRetrieve )
boolean Execute( AsyncCommand cmd, string sceneId )
table GetResults( AsyncCommand cmd)
```

Result table:

```
string description    -- localised scene description
string thumbnail      -- direct URL to large-format scene thumbnail
string author         -- localised author (maker) name
```

Get PS Store Logo

The current store logo is loaded into memory during bootup. The logo is available to scripts and can be called using the GetSystemResource function in the Resource lilbrary, which takes a constant value identifying the resource to retrieve.

To get the PS store logo you would call it like this:

```
resource = Resource.GetSystemResource( SysResources.TexPsStoreLogo )
```

Troubleshooting Commerce Content

Entitlement ID

- A missing or invalid Entitlement ID removes the object from PS Home.
- If the Entitlement ID has a valid format but identifies the asset incorrectly (for example, it corresponds to another object), users will not be able to purchase the desired product.

  Testing: Purchase an item and then test if it is correctly added to your inventory. Also, re-enter the commerce point; it should indicate that the item has already been purchased.

Product ID

- An object with missing or invalid Product ID will appear in a commerce point (i.e. users are able to select it), but there will be an error when PS Home tries to obtain its price. For this reason, users will not be able to add the object to the shopping basket.
- If the Product ID has a valid format but identifies the object incorrectly (for example, it corresponds to another object), users will not be able to purchase the desired product.

  Testing: Make sure that the item name is correct during the checkout process.

Age Ratings

- Incorrect age rating will result in the object appearing, but users being unable to add the items to their shopping baskets and purchase them

  Testing: Test with different age restricted accounts or use `overrideAgeControlAge` or `overrideAgeControlParentalLevel` in your hubstartup.txt file to test your scene with a specific age or parental control level. Include the command with an integer value for the age you want the scene to use, for example:

  ```
  overrideAgeControlAge 15
  ```

  For more information on using the hubstartup.txt, see Executing Console Commands at Boot Time.

Save Data Service

For Japanese Live Page
最新の日本語版ページはこちら 「セーブデータサービス」

PS Home Save Data Service

HDK users are now able to interact with the Save Data service via some new Lua async commands and some debug console commands.
The save data service provides the facility for data files to be stored and retreived at a later date.
There can be one data file per object per user.

See also Save Data Guidelines

Lua Async Commands

- **AsyncCommands.SaveDataSave**
- **AsyncCommands.SaveDataLoad**

Please see the AsyncCommand library in the Lua API Reference for more detailed information.

Console Commands

Console Commands	What they do
SaveData.DownloadFiles <directory path> <object ID 1> <object ID 2> ...	Will download all save data files requested to the directory given by the path. Each file will be named <object ID>.dat.

SaveData.DownloadFilesBulk <directory path> <path to bulk list file>	Will download all save data files listed in the bulk list file to the directory given by the path. The bulk list file should be a text file containing one object ID per line.
SaveData.DeleteFiles <object ID 1> <object ID 2> ...	Will delete all save data files listed as arguments.
SaveData.UploadFiles <object ID 1> <path 1> <object ID 2> <path 2> ...	Will upload files to save data slots with specified IDs, so file at "path 1" will be uploaded to save data slot for "object ID 1", and so on.

SaveData.UploadFilesBulk <path to bulk list file>	Will upload all save data files listed in the bulk list file. The bulk list file should be a text file with an object ID and file path on each line, in that order, separated by a space.
	
	For example:
	
	`38E56317-051E4CB0-AC119B66-2290243F`
	`app:38E56317-051E4CB0-AC119B66-2290243F.dat`
	`00000000-00000000-00000000-00000003`
	`app:00000000-00000000-00000000-00000003.dat`

All file paths used with the above commands support the prefixes usb, app, host, and so on.

Save Data Limits

- The maximum size of a save data file is 4096 bytes.
- Each object or scene (i.e. GUID) can read from /write to SaveData at most every 30 seconds.

> ⚠ Writing to a Save Data file for a user counts as a 'read' for the purposes of 'read maximum rate' above - i.e. if you write a Save Data file, it also counts as a read)

- The following uses of Save Data are not allowed: storage of executable information, shell scripts, Lua scripts, scripting information in any current or future recognised scripting language, any usual disclaimers.

Lua API Reference

Lua API is the full scripting cookbook which serves as reference for all of your library and function needs.

Download the CHM from SCEDev or from the download area of this page (see the paperclip just under the title).

See a list of new and changed functions in What's New in Lua.

> ℹ The functions will soon be incorporated into the wiki itself - right here in this section! Check the Recent Changes in the HDK Help Wiki for updates.

> ⚠ Errors with CHM
>
> If your CHM doesn't open or display properly, it may be due to Windows security.
>
> The error is caused by a Windows security patch that prevents CHM files from being seen properly displayed initially when downloaded online or shared on a network.
>
> To fix:
> 1. Right-click on the CHM > Properties.
> 2. Select 'Unblock.'
> 3. Open the .CHM.

Collapse all
Expand all Collapse all

Misc

Const

Brief

A variety of useful constant values.

Definition

Arguments

None

Return Values

Description

The available constant values are:
MinNum - The smallest positive normalized number. Note that this is a very small positive number, not the most negative normalized number. For this, use -Const.MaxNum.
MaxNum - The largest positive normalized number.
MaxContigInt - The largest representable contiguous integer, that is the integer x such that x, x - 1, x - 2, ..., 0 are all exactly representable as a Lua numbers, but x + 1 is not.

Examples

See Also

None

CreateTable

Brief

Create an empty Lua table pre-allocated to the specified size.

Definition

```
table CreateTable( number arraySize = 0, number recSize = 0 )
```

Arguments

arraySize - The number of array elements to allocate space for.recSize - The number of records (map elements) to allocate space for.

Return Values

An empty table, with space for arraySize array elements and recSize record (map) elements.

Description

Lua doesn't provide a native function to create a table pre-allocated to a certain size, which means that tables without an initializer list perform O(logN) allocations when inserting N elements. This function will create a table with sufficient space for a requested number of array and record elements. The size of the table will however be 0, as it has no elements (only the memory has been pre-allocated).

Examples

```
local cache = CreateTable( 100 )            -- create an empty table with space for 100 array elements
local dictionary = CreateTable( 0, 50 )     -- create an empty table with space for 50 records
local pointless = CreateTable()             -- default of 0, 0 is same as { }
print(#cache, #dictionary, #pointless)      -- prints "0 0 0"
```

See Also

None

DebugGetProfileTime

Brief

Return a debug timing value.

Definition

```
number DebugGetProfileTime()
```

Arguments

None

Return Values

Current time in seconds.

Description

Returns a timing value in seconds.

Examples

```

```

See Also

None

GetDeltaTime

Brief

Returns the time in seconds between this frame and the last frame.

Definition

```
number GetDeltaTime()
```

Arguments

None

Return Values

The time in seconds between this frame and the last frame.

Description

Returns the time in seconds between this frame and the last frame.

Examples

```
print( "Delta time is " .. GetDeltaTime() )
```

See Also

- GetInstanceTime

GetInstanceTime

Brief

Returns the current instance's time in milliseconds.

Definition

```
number, number GetInstanceTime()
```

Arguments

None

Return Values

The current instance time in milliseconds and nil if not in an instance.

Description

This returns the current world instance's time in milliseconds. The instance time starts counting when the instance was created and wraps almost every 50 days. Due to the precision of a lua number the time is split into two values. The bottom value contains 3 bytes and the top half contains the most significant 2 bytes. The majority of applications will not require the top number. The Home client must be run in online mode in order for this function to work as above. If the client is running offline for testing, the time since startup will be returned, in the same format. The following only applies to the online mode. Instance time is not designed for tracking real-world time with 100% accuracy due to latency between clients. It can slow down, speed up and even run backwards and is best used as a mechanism to sync short-term events between clients (e.g. launch a firework in 500ms and explode it 1200ms after that). This creates the effect of events occurring at the same time as far as the player is concerned, regardless of lag. If you need a mechanism to sync over longer periods of time, use Time.SetFromLocal. But note that keeping it updated by adding deltas to it will also cause it to eventually drift out of sync due to a build up of tiny errors. Long term syncing of events between clients should use the NP time with the Lua script checking it once every thirty seconds or so. Once the desired time is near, switch to another timer that is better suited for short time measurements.

Examples

```
b,t = GetInstanceTime()
print( string.format( "The instance time in hex is %#.2x%.6x", t, b ) )
```

See Also

- GetDeltaTime

GetThreadId

Brief

Returns the thread ID for the Lua environment.

Definition

```
number GetThreadId()
```

Arguments

None

Return Values

The thread ID.

Description

Returns the thread ID for the Lua environment.

Examples

```
print( "My id is " .. GetThreadId() )
```

See Also

None

IsPaused

Brief

Returns true if the environment is paused and false otherwise.

Definition

```
boolean IsPaused()
```

Arguments

None

Return Values

true if the environment is paused and false otherwise.

Description

Returns true if the environment is paused and false otherwise.

Examples

```
print( "I am paused = " .. IsPaused() )
```

See Also

- Pause
- Resume

KillThread

Brief

Kills the specified thread.

Definition

```
KillThread( number threadId )
```

Arguments

threadId - The thread to kill.

Return Values

None

Description

Kills the specified thread when the thread has yeilded or teminated.

Examples

```
KillThread( GetThreadId() )
print( "This will still print" )
Sleep( 0.0 )
```

See Also

None

LoadLibrary

Brief

Loads the specified library.

Definition

```
LoadLibrary( string name )
```

Arguments

name - Name of the library to load.

Return Values

None

Description

Loads the specified library. You may not have permision for specific libraries as components can grant library permission.

Examples

```
LoadLibrary( "Vector4" )
```

See Also

None

Pause

Brief

Pauses the Lua environment.

Definition

```
Pause()
```

Arguments

None

Return Values

None

Description

Pauses the Lua environment. Threads are suspended and update functions are not called.

Examples

```
Pause()
```

See Also

- Resume
- IsPaused

Resume

Brief

Resumes a paused Lua environment.

Definition

```
Resume()
```

Arguments

None

Return Values

None

Description

Resumes a paused Lua environment.

Examples

```
Resume()
```

See Also

- Pause

Sleep

Brief

Sleeps the current thread for a specified amount of time.

Definition

```
Sleep( number timeToSleep )
```

Arguments

timeToSleep - The time to sleep in seconds.

Return Values

None

Description

Sleeps the current thread for a specified amount of time. Only works for child threads.

Examples

```
function Hello()
    while true do
        print( "hello" )
        Sleep( 10.0 )
    end
end

SpawnFunction( Hello )
```

See Also

- SpawnFunction

SpawnFunction

Brief

Spawns the specified function in a child Lua thread.

Definition

```
number SpawnFunction( function functionName )
```

Arguments

functionName - The name of the function to spawn.

Return Values

The child thread ID value, or -1 on failure.

Description

Spawns the specified function in a child Lua thread. The function must sleep or yield otherwise it will execute forever.

Examples

```
function Hello()
    while true do
        print( "hello" )
        Sleep( 10.0 )
    end
end

SpawnFunction( Hello )
```

See Also

- Sleep

Type

Brief

Returns the type of the Home specific object.

Definition

```
string Type( userdata object )
```

Arguments

object - A Home specific type.

Return Values

The name of the type.

Description

Returns the type of the Home specific object. Similar to the Lua native type() command, except that where Lua's type() function will only return "userdata" for all API types, the Type function will return a string representing the API object type.

Examples

```
v = Vector4.Create()
print( type( v ) ) -- Prints "userdata"
print( Type( v ) ) -- Prints "Vector4"
```

See Also

None

Active

Active.GetEntity

Brief

Returns the entity for the active item model.

Definition

```
Entity Active.GetEntity()
```

Arguments

None

Return Values

The entity for the active item model.

Description

Returns the entity for the active item model so other entities can attach themselves to it or query it such as its position.

Examples

```
e = Active.GetEntity()
```

See Also

- Entity.GetPosition

Active.GetMaxSaveDataSize

Brief

Returns the maximum size for the save data allowed for this object.

Definition

```
number Active.GetMaxSaveDataSize()
```

Arguments

None

Return Values

The maximum size in bytes.

Description

Gets the maximum size in bytes of the save data for this object. Any attempt to save or read beyond this will generate an error. This value is guaranteed not to decrease between revisions, and will be at least 4096 bytes.

Examples

```
local maxSize = Active.GetMaxSaveDataSize()
```

See Also

None

Active.IsMoving

Brief

Returns true if the active item is currently being moved.

Definition

```
boolean Active.IsMoving()
```

Arguments

None

Return Values

Returns true if the active item is being moved, or false if it has been placed.

Description

Indicates whether the active item is moving or placed.

Examples

```
myActiveAnimationEnabled = Active.IsMoving()
```

See Also

None

Active.IsSafeVolumeIntersected

Brief

Returns true if the active item's safe volume is currently obstructed by another furniture item.

Definition

```
Entity Active.IsSafeVolumeIntersected()
```

Arguments

None

Return Values

True if the active item's safe volume is obstructed, or false if not.

Description

Indicates whether the active item's safe volume is currently intersected by any furniture items.

Examples

```
g_miniGameDisabled = Active.IsSafeVolumeIntersected()
```

See Also

None

Active.IsSaveDataAccess

Brief

Returns true if this instance is allowed to save data and false otherwise.

Definition

```
boolean Active.IsSaveDataAccess()
```

Arguments

None

Return Values

true if this instance is allowed to save data and false otherwise.

Description

The only instance that can save data is the owner of the apartment.

Examples

```
if Active.IsSaveDataAccess() == true then
    mc = MemoryContainer.Create( Active.GetMaxSaveDataSize() )
    Active.LoadData( mc )
end
```

See Also

None

Active.IsUpright

Brief

Returns true if the active item is currently upright.

Definition

```
boolean Active.IsUpright()
```

Arguments

None

Return Values

Returns true if the active item's safe volume is upright, or false if not.

Description

Indicates whether the active item is currently upright, that is, if it is considered to be in the correct orientation to be usable.

Examples

```
g_miniGameDisabled = not Active.IsUpright()
```

See Also

None

Active.LoadData

Brief

Copies the Active's save data into the supplied memory container.

Definition

```
void Active.LoadData(MemoryContainer memContainer, int size = Active.GetMaxSaveDataSize(), int offset
= 0)
```

Arguments

memContainer - The memory container to load into.size - Number of bytes to obtain.offset - Offset in bytes into the memory container at which to store the data.

Return Values

None.

Description

All actives have an allocated save data area which will be persistent across loads of the same object instance. Each instance has a separate save data area. Note that an active should not rely on specific data being present in the save data (as the user may delete their Home save data at any time) and thus actives should be able to recognise and handle the case where no save data is present. When an Active is created all of its save data is set to zero.

Examples

```
local memContainer = MemoryContainer.Create(Active.GetMaxSaveDataSize())
Active.LoadData(memContainer)
if (memContainer:GetUint16(0) == 48879) then        -- 0xBEEF
    print('Save data is valid: found the beef')
    g_mySize = memContainer:GetFloat32(4)
end
```

See Also

None

Active.SaveData

Brief

Copies the data from the supplied memory container into the Active's save data area and requests it to save.

Definition

```
Active.SaveData(MemoryContainer memContainer, number size = Active.GetMaxSaveDataSize(), int offset =
0)
```

Arguments

memContainer - The memory container to obtain the data from.size - Number of bytes to obtain.offset - Offset in bytes into the memory container at which to start reading data.

Return Values

None.

Description

All actives have an allocated save data area which will be persistent across loads of the same object instance. Each instance has a separate save data area. Note that an active should not rely on specific data being present in the save data (as the user may delete their Home save data at any time) and thus actives should be able to recognize and handle the case where no save data is

present.
Note: If size is not specified and your memory container is bigger than the maximum save data size an error will occur to catch accidental data truncation.

Examples

```
local memContainer = MemoryContainer.Create(8)
memContainer:SetUint16(0, 48879)        -- 0xBEEF, tags our data as valid
memContainer:SetFloat32(4, mySize)      -- save our current size so we can keep growing next time
Active.SaveData(memContainer, 8)
```

See Also

None

ActivityChannel

ActivityChannel.GetConnectionState

Brief

Used to determine the connection state of the Activity Channel.

Definition

```
ActivityChannelState ActivityChannel.GetConnectionState()
```

Arguments

None

Return Values

The connection state of the Activity Channel.
One of the following status codes:
ActivityChannelState.Disconnected - disconnected from Activity Channel.
ActivityChannelState.Connecting - connecting to Activity Channel.
ActivityChannelState.Connected - connected to Activity Channel.
ActivityChannelState.Disconnecting - disconnecting from Activity Channel.

Description

Sometimes it is useful to know whether the Activity Channel is connected or not, or indeed if the channel is in the process of connecting or disconnecting, e.g. for telling the user when they are able to chat, or ensuring an RTGame can be successfully started (which requires the Activity Channel to be disconnected).

Examples

```
if ActivityChannel.GetConnectionState() == ActivityChannelState.Disconnected then
    StartRTGame()
end
```

See Also

- ActivityChannel.Initialize

ActivityChannel.Initialize

Brief

Used to initialize or shut down the Chat Log Activity Channel.

Definition

```
ActivityChannel.Initialize( boolean value = true )
```

Arguments

value - true to initialize the activity channel, or false stop the activity channel.

Return Values

None.

Description

While players are joined to the game they can communicate privately via the Activity Channel, if initialized with this function. By default, an Activity Channel will not be initialized, and therefore will not be available. In all cases a channel is only created if the minigame/RTGame has a session size of 2 or more, and at least 2 people must be joined to the game in order to initialize the channel. This function should be called during initialization of the mini-game, before joining the game.

Examples

```
ActivityChannel.Initialize()
```

See Also

None

ActivityChannel.SetCommunicationPermissions

Brief

Set communication permissions for who each remote user in the activity channel (text and/or voice).

Definition

```
ActivityChannel.SetCommunicationPermissions( Person otherPerson, boolean isAllowedReceiveTextFrom,
boolean isAllowedReceiveVoiceFrom, boolean isAllowedSendTextTo, boolean isAllowedSendVoiceTo )
```

Arguments

remotePerson - remote person to set activity channel permissions onisAllowedReceiveTextFrom - whether we receive text from this remote personisAllowedReceiveVoiceFrom - whether we receive voice from this remote personisAllowedSendTextTo - currently ignoredisAllowedSendVoiceTo - whether we can send voice to this remote person

Return Values

None.

Description

This allows developers to set communications permissions between the local person and a remote person. Params isAllowedSendTextTo and isAllowedSendVoiceTo are used to determine whether the local user is able to send text or speech to another user, and affect whether a person will be visible in the chat-log. Note though that the client does not currently block outgoing communication on text, only incoming. The permissions are not replicated and so must be done on each client.

Examples

```
if (inHotSeat) then
    ActivityChannel.SetCommunicationPermissions( otherPlayerInQuizGame, false, false, false, false )
end
```

See Also

- ActivityChannel.Initialize

ActivityChannel.SetTitle

Brief

Set a descriptive title to the activity chat channel.

Definition

```
ActivityChannel.SetTitle( string channelTitle = nil )
```

Arguments

channelTitle - New channel title (if null or none supplied will default to default name)

Return Values

None.

Description

Lets people personalize their chat channel. Must be set after initializing the channel.

Examples

```
ActivityChannel.Initialize( );
ActivityChannel.SetTitle( "Team A" );
```

See Also

- ActivityChannel.Initialize

Animation

Animation.Create

Brief

Create an uninitialized animation object.

Definition

```
Animation Animation.Create()
```

Arguments

None

Return Values

An uninitialized animation object.

Description

Create a new uninitialized animation object. Can be passed to Entity.GetActiveAnim() to avoid uncontrolled memory allocation. No other Animation functions are valid on this instance until it is initialized with an active animation.

Examples

```
local anim = Animation.Create()
myEntity:GetActiveAnim(index, anim)
print(anim:GetDuration())
```

See Also

- Entity.GetActiveAnim

Animation.GetBlendState

Brief

Access the current blend state of the active animation.

Definition

```
AnimBlendState Animation.GetBlendState(Animation anim)
```

Arguments

anim - active animation to query

Return Values

An AnimBlendState value:
AnimBlendState.Zero - animation has blended out to zero.
AnimBlendState.In - animation is being blended in.
AnimBlendState.Out - animation is being blended out.
AnimBlendState.Playing - animation is playing at full blend weighting

Description

Returns the current blend state of an active animation

Examples

```
if (anim:GetBlendState() == AnimBlendState.Playing) then
    print("Anim is playing at max blend")
end
```

See Also

- Entity.BlendAnimIn
- Entity.BlendAnimOut
- Animation.GetBlendWeight
- Animation.GetBlendTime

Animation.GetBlendTime

Brief

Access the time within the ease in/out period.

Definition

```
number Animation.GetBlendTime( Animation anim )
```

Arguments

anim - active animation to query

Return Values

The time within the ease in/out period

Description

Access the time within the ease in/out period.

Examples

```
if (anim:GetBlendTime() == 0) then
    print("Blend hasn't started yet")
end
```

See Also

- Entity.BlendAnimIn
- Entity.BlendAnimOut
- Animation.GetBlendWeight
- Animation.GetBlendState

Animation.GetBlendWeight

Brief

Access the current blending weight of the specified active animation.

Definition

```
number Animation.GetBlendWeight(Animation anim)
```

Arguments

anim - active animation to query

Return Values

The current blending weight of the active animation, in the range [0, 1]

Description

Returns the current blending weight of the specified active animation.

Examples

```
local w = anim:GetBlendWeight()
if (w > 0) then
    print("Animation is blended in!")
end
```

See Also

- Entity.BlendAnimIn
- Entity.BlendAnimOut

- Animation.GetBlendTime
- Animation.GetBlendState

Animation.GetDuration

Brief

Get the duration of the animation in seconds.

Definition

```
number Animation.GetDuration( Animation anim )
```

Arguments

anim - A valid active animation.

Return Values

Duration of the animation in seconds.

Description

Get the duration of the animation in seconds.

Examples

```
local duration = anim:GetDuration()
```

See Also

- Animation.GetTime

Animation.GetName

Brief

Access the name of the animation resource being played.

Definition

```
string Animation.GetName( anim )
```

Arguments

anim - the active animation to be queried

Return Values

The name of the animation resource that is being played by the active animation, or nil if no active anim.

Description

Access the name of the animation resource being played by the active animation.

Examples

```
local name = anim:GetName()
```

See Also

None

Animation.GetPlaybackRate

Brief

Returns the current playback rate of the animation

Definition

```
number Animation.GetPlaybackRate(Animation anim)
```

Arguments

anim - the animation of which to query the playback rate

Return Values

the playback rate of the animation

Description

The value returned is a multiple of the normal playback rate, so 1 is normal rate, and -1 is reversed

Examples

```
number animSpeed = Animation.GetPlaybackRate(anim)
```

See Also

- Animation.SetPlaybackRate

Animation.GetTime

Brief

Get the current playback time within a specified animation.

Definition

```
number Animation.GetTime( Animation anim )
```

Arguments

anim - A valid active animation.

Return Values

Current playback time in seconds.

Description

Get the current animation playback time. This is the time index within the animation that the animation is currently located at.

Examples

```
local currAnimTime = anim:GetTime()
if (currAnimTime > 5.0) then
    ChangeAnim()
end
```

See Also

- Animation.SetTime

Animation.IsValid

Brief

Returns true if the animation object is valid.

Definition

```
boolean Animation.IsValid( Animation anim )
```

Arguments

anim - The animation object.

Return Values

True if the animation object is valid, false otherwise.

Description

When using the function variants that modify existing animation objects rather than return new instances, if an object has no active animation to return then the animation object will become invalid to reflect this. A script should check this before using the animation object, similarly to checking for nil when using the original "return new instance" version.

Examples

```
local anim = Animation.Create()
myEntity:GetActiveAnim(index, anim)
if (anim:IsValid()) then
    print("Entity has an active anim")
end
```

See Also

- Animation.Create
- Entity.GetActiveAnim

Animation.SetBlendWeight

Brief

Sets the blend weight of the specified active animation.

Definition

```
Animation.SetBlendWeight(Animation anim, number weight)
```

Arguments

anim - active animation to set weighting of.weight - the desired weighting for the animation (mut be between 0 and 1).

Return Values

None.

Description

If an animation is currently blending in or out, then setting the blend weight directly will stop the automatic blending, and leave the animations weighting to the one set through lua. Animation.GetBlendTime will return 0, and Animation.GetBlendState will return AnimBlendState.Playing. Use this function to manually control the animation blend weights without using the automatic blending.

Examples

```
anim1:SetBlendWeight(0.5)
```

See Also

- Entity.BlendAnimIn
- Entity.BlendAnimOut
- Entity.PlayAnim
- Animation.GetBlendTime
- Animation.GetBlendState

Animation.SetLoopCallback

Brief

Define a Lua function to be called when an animation loops or finishes.

Definition

```
Animation.SetLoopCallback(function callback, userData = nil)
```

Arguments

callback - The Lua function to call. Should be a globally-scoped function.userData - An object to pass to the callback function. Can be any valid Lua or API type.

Return Values

None.

Description

If a script wishes to trigger a new anim or perform other processing as soon as an existing anim finishes or loops, it can register a callback to occur on this event using this function. PS Home will call the specified function at the exact time of finishing or looping, prior to applying the animation to the skeleton. The callbacks are only valid while the Animation object on which they were registered exists; if the Animation object is garbage collected the callbacks will be deactivated. The animation object must be valid in order to set a callback.

Examples

```
entity = Entity.Create()
walkToDoorAnim = Animation.Create()

entity:SetModel("Model")
entity:SetSkeleton("Skeleton")

entity:BlendAnimIn("WalkToDoor", false, 0, AnimBlendType.Linear, 0.2, 0, walkToDoorAnim)

-- set up a callback to occur at end of WalkToDoor, and make it pass us walkToDoorAnim as userdata
walkToDoorAnim:SetLoopCallback(AnimCallback, walkToDoorAnim)

function AnimCallback(anim)
    if (anim == walkToDoorAnim) then
        entity:BlendAnimOut(walkToDoorAnim, AnimBlendType.Linear, 0.1)
        entity:BlendAnimIn("OpenDoor", false, 0, AnimBlendType.Linear, 0.1)
        -- if you want to register a new callback here, make sure the anim object lives beyond the
callback scope
    end
end
```

See Also

- Entity.BlendAnimIn

Animation.SetPlaybackRate

Brief

Sets the playback rate of an animation

Definition

```
Animation.SetPlaybackRate(Animation anim, number pbRate)
```

Arguments

anim - animation on which to set the playback ratepbRate - playback rate to set (can be zero, effectively pausing the anim). The playback rate is a multiplier, so 1 is normal speed, 0.5 is half speed
e, and -1 is reverse, and so on.

Return Values

none

Description

If the playback rate is set such that the animation will loop more than once in a single frame, any loop callbacks will only be called once.

Examples

```
Animation.SetPlaybackRate(reverseAnim, -1)
```

See Also

- Animation.GetPlaybackRate
- Entity.PlayAnim
- Entity.BlendAnimIn

Animation.SetTime

Brief

Set the current playback time within the specified animation.

Definition

```
Animation.SetTime( Animation anim, number time )
```

Arguments

anim - A valid active animation.time - Playback time to set in seconds.

Return Values

None

Description

Set the current animation playback time.

Examples

```
anim:SetTime(0.0)   -- set animation back to beginning
```

See Also

- Animation.GetTime

ArcadeGame

ArcadeGame.EnableMlaa

Brief

Enable or disable the MLAA (morphological anti-aliasing) process.

Definition

```
ArcadeGame.EnableMlaa(boolean enable)
```

Arguments

enable - True to enable or false to disable.

Return Values

None.

Description

PS Home uses the MLAA technique to perform anti-aliasing, and the implementation of this technique introduces one frame of rendering latency into the rendering pipeline. If the arcade game wishes to reduce latency this function can be used to disable the MLAA to remove this additional frame delay. Ensure that you re-enable it upon game exit.

Examples

```
ArcadeGame.EnableMlaa(false)
```

See Also

- ArcadeGame.EnableSceneRender
- ArcadeGame.EnableVsync

ArcadeGame.EnableSceneRender

Brief

Enable or disable the render of the 3D scene.

Definition

```
ArcadeGame.EnableSceneRender(boolean enable)
```

Arguments

enable - True to enable or false to disable.

Return Values

None.

Description

When playing an arcade game the current 3D scene is invisible, so it may be advantageous to a script to disable the render to reclaim some performance for the arcade game. Ensure that you re-enable it upon game exit.

Examples

```
ArcadeGame.EnableSceneRender(false)
```

See Also

- ArcadeGame.EnableMlaa
- ArcadeGame.EnableVsync

ArcadeGame.EnableVsync

Brief

Enable or disable the standard vsync timing.

Definition

```
ArcadeGame.EnableVsync(boolean enable)
```

Arguments

enable - True to enable or false to disable.

Return Values

None.

Description

By default PS Home enables vertical syncing (vsync) which limits the frame rate to 30 frames per second (fps). This function can be used to disable this and allow the arcade game to run at 60 fps, which will make the game feel more reactive and be more suitable for fast "twitch" style games.

Examples

```
ArcadeGame.EnableVsync(false)
```

See Also

- ArcadeGame.EnableSceneRender
- ArcadeGame.EnableMlaa

ArcadeGame.GetScreen

Brief

Returns the Arcade Game's screen.

Definition

```
Screen ArcadeGame.GetScreen()
```

Arguments

None

Return Values

The Arcade Game's screen.

Description

Returns the active Arcade Game's screen.

Examples

```
screen   = ArcadeGame.GetScreen()
renderer = Renderer.Create( Vector4.Create( 0, 0 ), Screen.GetSize( screen ) )
Renderer.SetTarget( renderer, screen )
```

See Also

- Renderer.Create
- Renderer.SetTarget
- Screen.GetSize
- Vector4.Create

ArcadeGame.LeaveGame

Brief

Leave an Arcade Game.

Definition

```
ArcadeGame.LeaveGame()
```

Arguments

None

Return Values

None.

Description

Tells the local player to leave the Arcade Game (if one is being played).

Examples

```
if (wantQuit) then
    ArcadeGame.LeaveGame()
end
```

See Also

None

AsyncCommand

AsyncCommand.Cancel

Brief

Cancels an in progress command.

Definition

```
boolean AsyncCommand.Cancel( AsyncCommand asyncCmd )
```

Arguments

asyncCmd - async command to cancel.

Return Values

It returns true if the command was successfully canceled and false otherwise.

Description

Cancels an executing command.

Examples

```
local result = AsyncCommand.Cancel( asyncCmd )
```

See Also

- AsyncCommand.Create
- AsyncCommand.Execute

AsyncCommand.Create

Brief

Create a new asynchronous command.

Definition

```
AsyncCommand.Create(commandType, ...)
```

Arguments

commandType - a valid command type value. See specific AsyncCommand.Create functions for valid command types.

Return Values

AsyncCommand object on success, nil on failure.

Description

Creates a new asynchronous command with the specified parameters, if any. If nil is returned and there is no associated Lua error provided, then this indicates that the specified command type is unavailable. If the command type is unavailable, the script should make no further attempts to create it.

Examples

```
ticketCmd = AsyncCommand.Create( AsyncCommands.TicketRequest )
```

See Also

- AsyncCommand.Create
- AsyncCommand.IsFinished
- AsyncCommand.GetErrorCode

AsyncCommand.Execute

Brief

Marks an asynchronous command for execution.

Definition

```
boolean AsyncCommand.Execute( asyncCmd, ...)
boolean AsyncCommand.Execute( AsyncCommand command, string serviceId, MemoryContainer container )
```

Arguments

asyncCmd - command to executeserviceId - The service id to request a ticket forcontainer - The container to store the ticket in

Return Values

false if command could not be executed, true if command successfully marked for execution.

Description

Submits the command to be executed. Note that success does not indicate that the command has finished, nor that it must successfully complete, rather that the command is in progress and will at some later stage either return successfully or with an error. The user should poll the command for completion, either within a coroutine or the game update loop. For the period between executing the command and receiving the result, the command is considered in progress. In this period the user can request cancellation of the command; this may or may not succeed depending on the execution stage of the command, and so should not be relied upon explicitly. No upper time limit is provided for the completion of command execution, nor are are any guarantees provided that commands will execute in the order they were submitted; in other words, a script should not rely on either a specific execution time or order. If required, execution order can be implemented by the script by waiting for preceding commands to finish before issuing dependent commands.

If Execute returns false, then this is most likely due to invalid or missing arguments passed to the function, and will be indicated by a Lua error - check the documentation for the command. If there is no Lua error, then the command execution has

failed due to the NP services system being currently too busy to accept another command. In this case, wait some time and try to execute the command again, and repeat if necessary. This situation should be rare and transient, and only occur under very high load.

Examples

```
local container = MemoryContainer.Create( 1024 )
local ticketCmd = AsyncCommand.Create( AsyncCommands.TicketRequest )
AsyncCommand:Execute( ticketCmd, "XX1234-XXXX12345_00", container )
```

See Also

- AsyncCommand.Create
- AsyncCommand.IsFinished
- AsyncCommand.IsInProgress
- AsyncCommand.Cancel

AsyncCommand.GetErrorCode

Brief

Indicates the command result code if the command failed.

Definition

```
number AsyncCommand.GetErrorCode(asyncCmd)
```

Arguments

asyncCmd - async command to query.

Return Values

A result code indicating the error that occurred.
The result returned can be one of the following:
AsyncErr.Unspecified = 1
AsyncErr.System = 2
AsyncErr.Fatal = 3
AsyncErr.ServerError = 4
AsyncErr.SystemUnavailable = 5
AsyncErr.Aborted = 6
AsyncErr.PermissionDenied = 7
AsyncErr.LimitExceeded = 8
AsyncErr.AuthenticationFail = 10
AsyncErr.NotFound = 11
AsyncErr.Blacklisted = 12
AsyncErr.AlreadyExists = 13
AsyncErr.Timeout = 14
AsyncErr.Busy = 15
AsyncErr.Censored = 16

AsyncErrRanking.ScoreNotFound = 101
AsyncErrRanking.BoardNotFound = 102
AsyncErrRanking.InvalidScore = 103
AsyncErrRanking.NotBestScore = 105

AsyncErrTrophy.UnknownTitle = 300
AsyncErrTrophy.InvalidCommId = 301

Description

The result code is only valid if the command failed; the code should not expect any specific result code for a pending or successful command. Result codes will be fixed, in that the same error will always produce the same result code.

Examples

```
if (not asyncCmd:Succeeded()) then
    print('Async command failed with result: ' .. asyncCmd:GetErrorCode())
end
```

See Also

- AsyncCommand.Succeeded

AsyncCommand.GetNpErrorCode

Brief

Gets the internal result code in the event of a command error result.

Definition

```
string AsyncCommand.GetNpErrorCode(asyncCmd)
```

Arguments

asyncCmd - async command to query.

Return Values

A result code indicating the error that occurred.

Description

The result code is only valid if the command failed; the code should not expect any specific result code for a pending or successful command. Result codes will be fixed, in that the same error will always produce the same result code. This command can be used to identify the cause of an error more exactly.

Examples

```
if (not asyncCmd:Succeeded()) then
    print('DEBUG: Async command failed with result: ' .. asyncCmd:GetNpErrorCode())
end
```

See Also

- AsyncCommand.Succeeded
- AsyncCommand.GetErrorCode

AsyncCommand.GetResultCount

Brief

Indicates the number of results available.

Definition

```
number AsyncCommand.GetResultCount( AsyncCommand asyncCmd )
```

Arguments

asyncCmd - async command to query.

Return Values

The number of results available.

Description

Returns the number of results available from the most recent execution of the command.

Examples

```
print(AsyncCommand.GetResultCount( asyncCmd ) .. ' results.')
```

See Also

AsyncCommand.GetResults

Brief

Access the results of the command.

Definition

```
value|table|array AsyncCommand.GetResults(AsyncCommand asyncCmd)
```

Arguments

asyncCmd - async command to query.

Return Values

A command-dependent value or table of values representing the results of the command.

Description

If the command has not successfully completed, nil will be returned. The exact format of the results is command-dependent, see the documentation for individual commands.

Examples

```
if (asyncCmd:IsFinished() and asyncCmd:Succeeded()) then
    print('Command results were: ', asyncCmd:GetResults())
end
```

See Also

- AsyncCommand.Succeeded

AsyncCommand.IsFinished

Brief

Indicates if the command has completed, either successfully or with an error.

Definition

```
boolean AsyncCommand.IsFinished(asyncCmd)
```

Arguments

asyncCmd - async command to query.

Return Values

true if the command has completed, false if not submitted or still in progress.

Description

Note that if a command has not yet been executed (only created), both IsInProgress and IsFinished will return false. This function will still return true if the command finishes with an error; to determine whether the command succeeded or failed query AsyncCommand.Succeeded().

Examples

```
if (asyncCmd:IsFinished() and asyncCmd:Succeeded()) then
    print('Results were: ', asyncCmd:GetResult())
end
```

See Also

- AsyncCommand.IsInProgress
- AsyncCommand.Succeeded
- AsyncCommand.GetErrorCode

AsyncCommand.IsInProgress

Brief

Indicates if the command has been submitted but has not yet completed.

Definition

```
boolean AsyncCommand.IsInProgress(asyncCmd)
```

Arguments

asyncCmd - async command to query.

Return Values

Returns true if command has been submitted and is not yet complete, or false otherwise.

Description

Note that if a command has not yet been executed (only created), both IsInProgress and IsFinished will return false.

Examples

```
if (asyncCmd:IsInProgress()) then
    print('Processing...')
end
```

See Also

- AsyncCommand.IsFinished

AsyncCommand.Succeeded

Brief

Indicates if the command succeeded or failed.

Definition

```
boolean AsyncCommand.Succeeded(asyncCmd)
```

Arguments

asyncCmd - async command to query.

Return Values

Returns true if the command is complete and succeeded, false otherwise.

Description

If the command has not been submitted or is still in progress, false will be returned. This should not be confused with the command failing, which can only be determined once the command is finished.

Examples

```
if (not asyncCmd:Succeeded()) then
    print('Async command failed with result: ', asyncCmd:GetErrorCode())
end
```

See Also

- AsyncCommand.GetErrorCode
- AsyncCommand.GetResults

AsyncContext

AsyncContext.Create

Brief

Creates a new asynchronous execution context.

Definition

```
AsyncContext AsyncContext.Create(AsyncContexts type, ...)
```

Arguments

type - the type of context to create. Can be one of the following:
AsyncContexts.TitleUserStorage
AsyncContexts.Trophy
AsyncContexts.ScoreRanking
AsyncContexts.Inventory
AsyncContexts.ObjectCatalog
AsyncContexts.Stream
AsyncContexts.SaveData

Return Values

The new async execution context, or nil if creation failed.

Description

Most async commands require the creation of an execution context in order to run. The context will typically specify details of the title and/or user the script wishes to target with the command. Note: It is good practice to nil your context if you will not use it again for the life of the script.

Examples

```
-- this creates a context to access the current user's TUS data for (non-existent) title NPWR99999
local passphrase = Resource.Find('passphrase')
local context = AsyncContext.Create(AsyncContexts.TitleUserStorage, "NPWR99999", passphrase)
```

See Also

- AsyncCommand.Create

Atmos

Atmos.GetParam

Brief

Reads a parameter from the scene atmosphere.

Definition

```
boolean|number|string|Vector4 Atmos.GetParam( string paramName )
boolean|number|string|Vector4 Atmos.GetParam( string paramName, number index )
```

Arguments

paramName - name of parameter to read. index - index of parameter. (if required)

Return Values

nil if parameter does not exist in scene. If successful the value of the parameter is returned.

Description

This function tries to retrieve a parameter value from the scene atmosphere, e.g. BASICSKY_HORIZONCOLOUR. If the parameter does not exist in the current atmosphere, nil is returned. The type of values returned back depend on the type of parameter. All parameter names and types can be found in the SkyTool documentation. Some parameters require an index because more than one can exist in a scene, e.g. SPRITECLOUD_TINTCOLOUR.

Examples

```
horizonColor = Atmos.GetParam( "BASICSKY_HORIZONCOLOUR" )           --  horizonColor will become a
Vector4
numSpriteClouds = Atmos.GetParam( "NUM_SPRITECLOUDS" )              --  numSpriteClouds will become a
number
spriteCloudTint = Atmos.GetParam( "SPRITECLOUD_TINTCOLOUR", 4 )     --  spriteCloudTint will become a
Vector4
```

See Also

None

Atmos.SetParam

Brief

Sets a parameter to the scene atmosphere

Definition

```
boolean Atmos.SetParam( string paramName, boolean|number|string|Vector4 paramVal )
boolean Atmos.SetParam( string paramName, number index, boolean|number|string|Vector4 paramVal )
```

Arguments

paramName - name of parameter to be set.index - index of parameter. (if required)paramVal - value of parameter to be set.

Return Values

false if parameter does not exist in scene, or if the type given is incorrect. true if parameter set correctly.

Description

This function sets a parameter in the scene atmosphere, e.g. BASICSKY_HORIZONCOLOUR. If the parameter does not exist in the current atmosphere, false is returned. The type of value to assign depends on the type of parameter. If the type given is incorrect, false is returned. All parameter names and types can be found in the SkyTool documentation. Access to the Atmos library is only granted for scene objects such as mini games and embedded objects. Some parameters require an index because more than one can exist in a scene, e.g. SPRITECLOUD_TINTCOLOUR.

Examples

```
Atmos.SetParam( "BASICSKY_HORIZONCOLOUR", horizonColor );           --  horizonColor is a Vector4
Atmos.SetParam( "EXPHEIGHTFOG_FOGDENSITY", fogDensity );            --  fogDensity is a number
Atmos.SetParam( "SPRITECLOUD_TINTCOLOUR", 4, spriteCloudTint );     --  spriteCloudTint is a Vector4
```

See Also

None

Base64

Base64.Decode

Brief

Decodes the base64 contents of a memory container.

Definition

```
number Base64.Decode( MemoryContainer out, MemoryContainer in, number offset = -1, number size = -1 )
```

Arguments

out - The memory container to write the decoded data to.in - The memory container to read the base64 data from.offset - The offset of the data in the input memory container. Defaults to -1.size - The size of data to read from the input memory container. Defaults to -1.

Return Values

The number of bytes written to the output memory container or a negative number on error.
-1 - Invalid arg
-2 - Not enough space in the target memory container

Description

Decodes the base64 contents of a memory container. Data is always written to the output container as a stream. If no size or offset are supplied (or are set to -1), all available data from the input memory container will be encoded. This function will not alter the input memory container in anyway.

Examples

```
local output = MemoryContainer.Create( 1024 )
local numBytesDecoded = Base64.Decode( output, input )
```

See Also

- Base64.Encode
- Base64.DecodeFromString

Base64.DecodeFromString

Brief

Decodes the base64 contents of a string.

Definition

```
number Base64.DecodeFromString( MemoryContainer out, string in )
```

Arguments

out - The memory container to write the decoded data to.in - The string containing the base64 data.

Return Values

The number of bytes written to the output memory container or a negative number on error.
-1 - Invalid arg
-2 - Not enough space in the target memory container

Description

Decodes the base64 string and writes the decoded data to the output memory container. Data is always written to the output container as a stream.

Examples

```
local output = MemoryContainer.Create( 1024 )
local numBytesDecoded = Base64.DecodeFromString( output, 'QmFzZTY0IHRlc3QgMw==' )
print( MemoryContainer.GetString( output, 0 ) )
```

See Also

- Base64.Encode
- Base64.Decode

Base64.Encode

Brief

Encodes the contents of a memory container into base64.

Definition

```
number Base64.Encode( MemoryContainer out, MemoryContainer in, number offset = -1, number size = -1 )
```

Arguments

out - The memory container to write the encoded data to.in - The memory container to read the original data from.offset - The offset of the data in the input memory container.size - The size of data to read from the input memory container.

Return Values

The number of bytes written to the output memory container or a negative number on error.
-1 - Invalid arg
-2 - Not enough space in the target memory container

Description

Encodes the contents of a memory container into base64. Data is always written to the output container as a stream. If no size or offset are supplied (or are set to -1), all available data from the input memory container will be encoded. This function will not alter the input memory container in anyway.

Examples

```
local input = MemoryContainer.Create( 1024 )
local output = MemoryContainer.Create( 1024 )
MemoryContainer.SetString( input, 0, 'Base64 test 1' )
local numBytesEncoded = Base64.Encode( output, input )
```

See Also

- Base64.Decode
- Base64.DecodeFromString

BasicGenx

BasicGenx.AddAttribute

Brief

Adds an attribute to the current element.

Definition

```
number BasicGenx.AddAttribute( BasicGenx writer, string name, string|boolean|number value )
```

Arguments

writer - The instance of the XML writer.name - The name of the new attribute.value - A string, boolean or number to be used as the attribute value.

Return Values

0 on success, any other value is an error. BasicGenx error codes are:
1 - Bad UTF-8
2 - Non-XML character
3 - Bad name
4 - Memory allocation failed
5 - Bad namespace name
6 - Internal error
7 - Duplicate prefix
8 - Call out of sequence
9 - No start tag for EndElement call
10 - I/O error
11 - Missing attribute value
12 - Malformed comment body
13 - ?> in Processing Instruction
14 - Target of Processing Instruction matches [xX][mM][lL]
15 - Same attribute specified more than once
16 - Attribute cannot be in default namespace
17 - Namespace declared twice with different prefixes on one element
18 - Default namespace declared on an element which is in no namespace

Description

Adds an attribute to the current element. If the value passed is a boolean and it is set to true, then "true" is written, otherwise, "false" is written. If the value passed is a number, it will always be written in floating point representation even if it is a whole number. In order to write a number in integer format, it is necessary to use BasicGenx.AddAttributeInt. A string value is written as specified. Passing nil as the value parameter will result in the string "nil" being written.

Examples

```
-- Example 1: Write "Hello World" to the XML
stringValue = "John Smith"
result = BasicGenx.AddAttribute( writer, "playerName", stringValue )

-- Example 2: Write "true" to the XML
booleanValue = true
result = BasicGenx.AddAttribute( writer, "enableStats", booleanValue )

-- Example 3: Write "100.000000" to the XML
numberValue = 100
result = BasicGenx.AddAttribute( writer, "percentComplete", numberValue )
```

See Also

- BasicGenx.AddAttributeInt

BasicGenx.AddAttributeInt

Brief

Adds a new attribute to the current XML element with integer formatting.

Definition

```
number BasicGenx.AddAttributeInt( BasicGenx writer, string name, number value )
```

Arguments

writer - The instance of the XML writer.name - The name of the new attribute.value - The value of the attribute.

Return Values

0 on success, any other value is an error. BasicGenx error codes are:
1 - Bad UTF-8
2 - Non-XML character
3 - Bad name
4 - Memory allocation failed
5 - Bad namespace name
6 - Internal error
7 - Duplicate prefix
8 - Call out of sequence
9 - No start tag for EndElement call
10 - I/O error
11 - Missing attribute value
12 - Malformed comment body
13 - ?> in Processing Instruction
14 - Target of Processing Instruction matches [xX][mM][lL]
15 - Same attribute specified more than once
16 - Attribute cannot be in default namespace
17 - Namespace declared twice with different prefixes on one element
18 - Default namespace declared on an element which is in no namespace

Description

Adds a new attribute to the current XML element with integer formatting. The value is written specifically as an integer, as opposed to BasicGenx.AddAttribute which writes numbers in floating point format. If the specified number is not an integer, it is truncated in some non-specified way.

Examples

```
local result = BasicGenx.AddAttributeInt( writer, "numItems", 10 )
```

See Also

- BasicGenx.AddAttribute

BasicGenx.AddText

Brief

Sets the value of an XML element.

Definition

```
number BasicGenx.AddText( BasicGenx writer, string|boolean|number value )
```

Arguments

writer - The instance of the XML writer.value - A string, boolean or number to set as the value in the XML element.

Return Values

0 on success, any other value is an error. BasicGenx error codes are:
1 - Bad UTF-8
2 - Non-XML character
3 - Bad name
4 - Memory allocation failed
5 - Bad namespace name
6 - Internal error
7 - Duplicate prefix
8 - Call out of sequence
9 - No start tag for EndElement call
10 - I/O error
11 - Missing attribute value
12 - Malformed comment body
13 - ?> in Processing Instruction
14 - Target of Processing Instruction matches [xX][mM][lL]
15 - Same attribute specified more than once
16 - Attribute cannot be in default namespace
17 - Namespace declared twice with different prefixes on one element
18 - Default namespace declared on an element which is in no namespace

Description

Sets the value of an XML element. If the value passed is a boolean and it is set to true, then "true" is written, otherwise, "false" is written. If the value passed is a number, it will always be written in floating point representation even if it is a whole number. In order to write a number in integer format, it is necessary to use BasicGenx.AddTextAsInt. A string value is written as specified. Passing nil as the value parameter will result in the string "nil" being written.

Examples

```
-- Example 1: Write "Hello World" to the XML
stringValue = "Hello World"
result = BasicGenx.AddText( writer, stringValue )

-- Example 2: Write "true" to the XML
booleanValue = true
result = BasicGenx.AddText( writer, booleanValue )

-- Example 3: Write "10.000000" to the XML
numberValue = 10
result = BasicGenx.AddText( writer, numberValue )
```

See Also

- BasicGenx.AddTextAsInt

BasicGenx.AddTextAsInt

Brief

Sets the value of an XML element with integer formatting.

Definition

```
number BasicGenx.AddTextAsInt( BasicGenx writer, number value )
```

Arguments

writer - The instance of the XML writer.value - The value to set in the XML element.

Return Values

0 on success, any other value is an error. BasicGenx error codes are:
1 - Bad UTF-8
2 - Non-XML character
3 - Bad name
4 - Memory allocation failed
5 - Bad namespace name
6 - Internal error
7 - Duplicate prefix
8 - Call out of sequence
9 - No start tag for EndElement call
10 - I/O error
11 - Missing attribute value
12 - Malformed comment body
13 - ?> in Processing Instruction
14 - Target of Processing Instruction matches [xX][mM][lL]
15 - Same attribute specified more than once
16 - Attribute cannot be in default namespace
17 - Namespace declared twice with different prefixes on one element
18 - Default namespace declared on an element which is in no namespace

Description

Sets the value of an XML element with integer formatting. The value is written specifically as an integer, as opposed to BasicGenx.AddText which writes numbers in floating point format. If the specified number is not an integer, it is truncated in some non-specified way.

Examples

```
-- Example: Write "10" to the XML
numberValue = 10
result = BasicGenx.AddTextAsInt( writer, numberValue )
```

See Also

- BasicGenx.AddText

BasicGenx.CheckText

Brief

Scan a string and report problems.

Definition

```
number BasicGenx.CheckText( BasicGenx writer, string text )
```

Arguments

writer - The instance of the XML writer.text - The string to check.

Return Values

0 on success, any other value is an error. BasicGenx error codes are:
1 - Bad UTF-8
2 - Non-XML character
3 - Bad name
4 - Memory allocation failed
5 - Bad namespace name
6 - Internal error
7 - Duplicate prefix
8 - Call out of sequence
9 - No start tag for EndElement call
10 - I/O error
11 - Missing attribute value
12 - Malformed comment body
13 - ?> in Processing Instruction
14 - Target of Processing Instruction matches [xX][mM][lL]
15 - Same attribute specified more than once
16 - Attribute cannot be in default namespace
17 - Namespace declared twice with different prefixes on one element
18 - Default namespace declared on an element which is in no namespace

Description

Scan a string to be written by BasicGenx and report problems with UTF-8 encoding or non-XML characters.

Examples

```
local result = BasicGenx.CheckText( writer, "My test text" )
```

See Also

None

BasicGenx.Create

Brief

Creates a new XML writer.

Definition

```
BasicGenx.Create( MemoryContainer container )
```

Arguments

container - The memory container to write the XML data to.

Return Values

A new instance of an XML writer.

Description

Creates a new XML writer. The memory container is written to as if it were a stream so it can be used to write data to an existing container.

Examples

```
local container = MemoryContainer.Create( 4096 )
local writer = BasicGenx.Create( container )
```

See Also

- MemoryContainer.Create

BasicGenx.EndDocument

Brief

Ends writing an XML document.

Definition

```
BasicGenx.EndDocument( BasicGenx writer )
```

Arguments

writer - The instance of the XML writer.

Return Values

None.

Description

Ends writing an XML document.

Examples

```
local container = MemoryContainer.Create( 4096 )
local writer = BasicGenx.Create( container )
BasicGenx.StartDocument( writer )

-- Write XML element hierarchy here...

BasicGenx.EndDocument( writer )
```

See Also

- BasicGenx.StartDocument

BasicGenx.EndElement

Brief

Closes an XML element.

Definition

```
number BasicGenx.EndElement( BasicGenx writer )
```

Arguments

writer - The instance of the XML writer.

Return Values

0 on success, any other value is an error. BasicGenx error codes are:
1 - Bad UTF-8
2 - Non-XML character
3 - Bad name
4 - Memory allocation failed
5 - Bad namespace name
6 - Internal error
7 - Duplicate prefix
8 - Call out of sequence
9 - No start tag for EndElement call
10 - I/O error
11 - Missing attribute value
12 - Malformed comment body
13 - ?> in Processing Instruction
14 - Target of Processing Instruction matches [xX][mM][lL]
15 - Same attribute specified more than once
16 - Attribute cannot be in default namespace
17 - Namespace declared twice with different prefixes on one element
18 - Default namespace declared on an element which is in no namespace

Description

Closes an XML element which must have previously been opened with BasicGenx.StartElement.

Examples

```
local result = BasicGenx.EndElement( writer )
```

See Also

- BasicGenx.StartElement

BasicGenx.StartDocument

Brief

Starts writing an XML document.

Definition

```
BasicGenx.StartDocument( BasicGenx writer )
```

Arguments

writer - The instance of the XML writer.

Return Values

None.

Description

Starts writing an XML document.

Examples

```
local container = MemoryContainer.Create( 4096 )
local writer = BasicGenx.Create( container )
BasicGenx.StartDocument( writer )

-- Write XML element hierarchy here...

BasicGenx.EndDocument( writer )
```

See Also

- BasicGenx.Create
- BasicGenx.EndDocument
- MemoryContainer.Create

BasicGenx.StartElement

Brief

Starts writing an XML element.

Definition

```
number BasicGenx.StartElement( BasicGenx writer, string name )
```

Arguments

writer - The instance of the XML writer.name - The name of the new XML element.

Return Values

0 on success, any other value is an error. BasicGenx error codes are:
1 - Bad UTF-8
2 - Non-XML character
3 - Bad name
4 - Memory allocation failed
5 - Bad namespace name
6 - Internal error
7 - Duplicate prefix
8 - Call out of sequence
9 - No start tag for EndElement call
10 - I/O error
11 - Missing attribute value
12 - Malformed comment body
13 - ?> in Processing Instruction
14 - Target of Processing Instruction matches [xX][mM][lL]
15 - Same attribute specified more than once
16 - Attribute cannot be in default namespace
17 - Namespace declared twice with different prefixes on one element
18 - Default namespace declared on an element which is in no namespace

Description

Starts writing an XML element. It is valid to nest calls to BasicGenx.StartElement.

Examples

```
local result = BasicGenx.StartElement( writer, "XML" )
```

See Also

- BasicGenx.EndElement

BigInt

BigInt.Add

Brief

Add the specified value to a BigInt.

Definition

```
BigInt.Add(BigInt dst, BigInt|number|string src1, BigInt|number|string src2)
```

Arguments

dst - the BigInt to modifysrc1 - the first/left sourcesrc2 - the second/right source

Return Values

Description

Adds the specified values to the specified BigInt. dst = src1 + src2.

Examples

```
bigInt1:Add(bigInt1, 32)
bigInt2:Add(bigInt1, "0x393823")
bigInt3:Add("723523", bigInt3)
bigInt4:Add(bigInt3, bigInt3)
```

See Also

- BigInt.Create

BigInt.And

Brief

Perform a bitwise AND operation with a BigInt and a specified value.

Definition

```
BigInt.And(BigInt dst, BigInt|number|string src1, BigInt|number|string src2)
```

Arguments

dst - the BigInt to modifysrc1 - the first/left sourcesrc2 - the second/right source

Return Values

Description

Perform a bitwise AND operation with a BigInt and a specified value. String values may be decimal, or if prefixed with 0x or 0X, hexadecimal. The string will be checked to ensure it is a valid number and fits within the BigInt's precision limit.

Examples

```
bigInt1:And(32)
bigInt2:And("0x8000")
bigInt3:And("128")
bigInt4:And(bigInt3)
```

See Also

- BigInt.Create

BigInt.AsHexString

Brief

Generates a hexadecimal string representation of the BigInt.

Definition

```
string BigInt.AsHexString(BigInt i)
```

Arguments

i - a BigInt

Return Values

A string containing the BigInt value in hexadecimal.

Description

Generates a hexadecimal string representation of the BigInt. The string will contain only hexadecimal digits (0-9, A-F) with no prefix (0x or similar).

Examples

```
local i = BigInt.Create(64, "ABCDEF01234456789")
print(i:AsHexString())      -- prints "ABCDEF01234456789"
```

See Also

- BigInt.Create

BigInt.Create

Brief

Create a signed big integer with the specified number of bits.

Definition

```
BigInt BigInt.Create(number numberOfBits, number|BigInt|string initialValue = 0 )
```

Arguments

numberOfBits - size of integer in bits
initialValue - value to initialise returned big integer in either number, string or BigInt.

Return Values

A big integer with the specified size, or nil on failure.

Description

Create a big integer with the specified number of bits of precision. At present only 32 and 64 bit integers are supported. Note that these are not "native" types and therefore have a significant performance penalty, and should only be used where necessary. Normal integer arithmetic should be done using the built-in Lua number type. String values may be either decimal or, if prefixed with 0x or 0X, hexadecimal. The string will be checked to ensure it is a valid number and fits within the BigInt's precision limit.

Examples

```
local bigInt = BigInt.Create(64, "0x1234567890ABCDEF")
local bigInt2 = BigInt.Create(64, "1234567890123456")
local bigInt3 = BigInt.Create(32, 123321)
local bigInt4 = BigInt.Create(32, bigInt3)
```

See Also

- BigInt.Set

BigInt.Divide

Brief

Divide a BigInt by a specified value.

Definition

```
BigInt.Divide(BigInt dst, BigInt|number|string src1, BigInt|number|string src2)
```

Arguments

dst - the BigInt to modifysrc1 - the first/left sourcesrc2 - the second/right source

Return Values

Description

Divide the specified BigInt by the specified value. String values may be decimal, or if prefixed with 0x or 0X, hexadecimal. The string will be checked to ensure it is a valid number and fits within the BigInt's precision limit.

Examples

```
bigInt1:Divide(32)
bigInt2:Divide("0x393823")
bigInt3:Divide("723523")
bigInt4:Divide(bigInt3)
```

See Also

- BigInt.Create

BigInt.GetBit

Brief

Returns the value of the specified bit.

Definition

```
boolean BigInt.GetBit( BigInt i, number bitIndex )
```

Arguments

i - a BigIntbitIndex - Which bit to get the value of. bitIndex starts at 0.

Return Values

Returns the value of the specified bit.

Description

Returns the value of the specified bit.

Examples

```
local i = BigInt.Create( 64, -1 )
print( 'top bit is ', BigInt.GetBit(i, 63 ) )
```

See Also

- BigInt.Create
- BigInt.SetBit

BigInt.Multiply

Brief

Multiply the specified value with a BigInt.

Definition

```
BigInt.Multiply(BigInt dst, BigInt|number|string src1, BigInt|number|string src2)
```

Arguments

dst - the BigInt to modifysrc1 - the first/left sourcesrc2 - the second/right source

Return Values

Description

Multiply the specified value with the specified BigInt. String values may be decimal, or if prefixed with 0x or 0X, hexadecimal. The string will be checked to ensure it is a valid number and fits within the BigInt's precision limit.

Examples

```
bigInt1:Multiply(32)
bigInt2:Multiply("0x393823")
bigInt3:Multiply("723523")
bigInt4:Multiply(bigInt3)
```

See Also

- BigInt.Create

BigInt.Negate

Brief

Negate a BigInt.

Definition

```
BigInt.Negate(BigInt i)
```

Arguments

i - the BigInt to negate

Return Values

Description

Negate the specified BigInt, that is, if it's current value is x, set it to -x.

Examples

```
bigInt1:Negate()
```

See Also

- BigInt.Create

BigInt.Not

Brief

Perform a bitwise NOT operation on the specified BigInt.

Definition

```
BigInt.Not(BigInt i)
```

Arguments

i - the BigInt on which to perform bitwise NOT

Return Values

Description

Perform a bitwise NOT operation on the specified BigInt.

Examples

```
bigInt1:Not()
```

See Also

- BigInt.Create

BigInt.Or

Brief

Perform a bitwise OR operation with a BigInt and a specified value.

Definition

```
BigInt.Or(BigInt dst, BigInt|number|string src1, BigInt|number|string src2)
```

Arguments

dst - the BigInt to modifysrc1 - the first/left sourcesrc2 - the second/right source

Return Values

Description

Perform a bitwise OR operation with a BigInt and a specified value. String values may be decimal, or if prefixed with 0x or 0X, hexadecimal. The string will be checked to ensure it is a valid number and fits within the BigInt's precision limit.

Examples

```
bigInt1:Or(32)
bigInt2:Or("0x8000")
bigInt3:Or("128")
bigInt4:Or(bigInt3)
```

See Also

- BigInt.Create

BigInt.Set

Brief

Set a big integer to the specified value

Definition

```
BigInt.Set(BigInt|number|string value)
```

Arguments

value - value to set

Return Values

Description

Set a big integer to the specified value. String values may be either decimal or, if prefixed with 0x or 0X, hexadecimal. The string will be checked to ensure it is a valid number and fits within the BigInt's precision limit.

Examples

```
bigInt:Set("0x1234567890ABCDEF")
bigInt2:Set("1234567890123456")
bigInt3:Set(123321)
bigInt4:Set(bigInt3)
```

See Also

- BigInt.Create

BigInt.SetBit

Brief

Sets the the specified bit to the specified value.

Definition

```
boolean BigInt.SetBit( BigInt i, number bitIndex, boolean value )
```

Arguments

i - a BigIntbitIndex - Which bit to get the value of. bitIndex starts at 0.value - The value to set the bit to, true is on and false is off.

Return Values

None

Description

Sets the the specified bit to the specified value.

Examples

```
local i = BigInt.Create( 64, 1234 )
BigInt.SetBit( i, 63, true ) ) -- Make it negative.
```

See Also

- BigInt.Create
- BigInt.GetBit

BigInt.SetZero

Brief

Set the specified BigInt to zero.

Definition

```
BigInt.SetZero(BigInt i)
```

Arguments

i - the BigInt to set to zero

Return Values

Description

Set the specified BigInt to zero.

Examples

```
bigInt1:SetZero()
```

See Also

- BigInt.Create

BigInt.ShiftLeft

Brief

Bit shifts the source integer left by the specified amount of bits and saves in the destination.

Definition

```
BigInt.ShiftLeft( BigInt dst, BigInt|number|string src, BigInt|number|string shift )
```

Arguments

dst - The BigInt to save the result to.src - The value to shift.shift - the amount to shift by.

Return Values

None

Description

Bit shifts the source integer left by the specified amount of bits and saves the result in the destination.

Examples

```
dst = BigInt.Create( 64 )
src = BigInt.Create( 64, "0xFF0000" )
BigInt.ShiftLeft( dst, src, 5 )
```

See Also

- BigInt.Create
- BigInt.ShiftRight
- BigInt.GetBit
- BigInt.SetBit

BigInt.ShiftRight

Brief

Bit shifts the source integer right by the specified amount of bits and saves the result in the destination.

Definition

```
BigInt.ShiftRight( BigInt dst, BigInt|number|string src, BigInt|number|string shift )
```

Arguments

dst - The BigInt to save the result to.src - The value to shift.shift - the amount to shift by.

Return Values

None

Description

Bit shifts the source integer right by the specified amount of bits and saves the result in the destination.

Examples

```
dst = BigInt.Create( 64 )
src = BigInt.Create( 64, "0xFF0000" )
BigInt.ShiftRight( dst, src, 5 )
```

See Also

- BigInt.Create
- BigInt.ShiftRight
- BigInt.GetBit
- BigInt.SetBit

BigInt.Subtract

Brief

Subtract the specified value from a BigInt.

Definition

```
BigInt.Subtract(BigInt dst, BigInt|number|string src1, BigInt|number|string src2)
```

Arguments

dst - the BigInt to modifysrc1 - the first/left sourcesrc2 - the second/right source

Return Values

Description

Subtract the specified value from the specified BigInt. String values may be decimal, or if prefixed with 0x or 0X, hexadecimal. The string will be checked to ensure it is a valid number and fits within the BigInt's precision limit.

Examples

```
bigInt1:Subtract(32)
bigInt2:Subtract("0x393823")
bigInt3:Subtract("723523")
bigInt4:Subtract(bigInt3)
```

See Also

- BigInt.Create

BigInt.ToNumber

Brief

Converts a BigInt to a native Lua number value if possible.

Definition

```
number BigInt.ToNumber( BigInt i )
```

Arguments

i - a BigInt

Return Values

The Lua number value equivalent to the BigInt.

Description

Converts a BigInt to a native Lua number value if possible. If the BigInt is out of range, a Lua error will occur.

Examples

```
local i = BigInt.Create( 32, 1234 )
print(i:ToNumber() + 1)     -- prints 1235
i:Set("121223523262")
--print(i:ToNumber())       this will error
```

See Also

- BigInt.Create

BigInt.Xor

Brief

Perform a bitwise XOR operation with a BigInt and a specified value.

Definition

```
BigInt.Xor(BigInt dst, BigInt|number|string src1, BigInt|number|string src2)
```

Arguments

dst - the BigInt to modifysrc1 - the first/left sourcesrc2 - the second/right source

Return Values

Description

Perform a bitwise XOR operation with a BigInt and a specified value. String values may be decimal, or if prefixed with 0x or 0X, hexadecimal. The string will be checked to ensure it is a valid number and fits within the BigInt's precision limit.

Examples

```
bigInt1:Xor(32)
bigInt2:Xor("0x8000")
bigInt3:Xor("128")
bigInt4:Xor(bigInt3)
```

See Also

- BigInt.Create

Camera

Camera.Activate

Brief

Activates the specified camera.

Definition

```
Camera.Activate( Camera camera, number transitionTime = 0.0 )
```

Arguments

camera - The camera to activate.transitionTime - The time taken to smoothly move from the previously active camera's position, to

the
specified camera's position.

Return Values

None.

Description

Activates the specified camera.

Examples

```
Camera.Activate( gameCamera, 1.0 )
```

See Also

- Camera.Deactivate
- Camera.IsActivated

Camera.AttachToParent

Brief

Links the camera to the given entity so that the camera inherits the entity's transform.

Definition

```
Camera.AttachToParent( Camera camera, Entity parent )
```

Arguments

camera - The camera to update.parent - The entity to attach this camera to, or nil to detach the camera.

Return Values

None.

Description

Links the camera to the given entity so that the camera inherits the entity's transform. This will override any values set via Camera.SetPosition, or Camera.SetLookAtPosition until the connection is cleared by passing in nil as the parent. If the entity is destroyed then the camera will be unparented. It is not possible at present to specify a local transform offset from the entity.

Examples

```
Camera.AttachToParent( gameCamera, myEntity ) -- connect the camera to myEntity
Camera.AttachToParent( gameCamera, nil ) -- detach the camera from any connected entity
```

See Also

None

Camera.Create

Brief

Creates a new 3D camera.

Definition

```
Camera Camera.Create( string constructionMethod = nil )
```

Arguments

constructionMethod - How to construct the camera, either nil or "current".

Return Values

Returns a Camera instance.

Description

Creates a new 3D camera. This camera is, by default, positioned at the origin (0, 0, 0). If constructionMethod is "current" it creates a new camera with the properties of the currently active camera.

Examples

```
gameCamera = Camera.Create( "current" )
Camera.SetPosition( gameCamera, Camera.GetPosition() + Vector4.Create( 0, 1, 0 ) )
```

See Also

- Camera.Activate

Camera.Deactivate

Brief

Deactivates the specified camera.

Definition

```
Camera.Deactivate( Camera camera, number transitionTime = 0.0 )
```

Arguments

camera - The camera to deactivate.transitionTime - The time taken to smoothly move from the specified camera's position back to the
previously active camera's position.

Return Values

None.

Description

Deactivates the specified camera.

Examples

```
if Camera.IsActivated( gameCamera ) then
    Camera.Deactivate( gameCamera, 1.0 )
end
```

See Also

- Camera.Activate
- Camera.IsActivated

Camera.EnableAvatarCameraSync

Brief

Tell the normal Avatar camera to face the same direction as a deactivating custom camera, minimizing the camera transition.

Definition

```
Camera.EnableAvatarCameraSync()
```

Arguments

None

Return Values

Description

When enabled, the normal Avatar Camera will sync with any active custom camera to face the same direction. This simply means that when the custom camera is deactivated, the normal Avatar Camera will be facing the same direction, and the magnitude of the transition between the two will be minimized. This defaults to on. It's rare, and generally not advised to disable this.

Examples

```
Camera.EnableAvatarCameraSync(false)
```

See Also

None

Camera.GetInverseViewMatrix

Brief

Get the view to world transform for the currently active camera.

Definition

```
Camera.GetInverseViewMatrix(Matrix44 matrix)
```

Arguments

matrix - The matrix to set.

Return Values

None.

Description

Retrieves the view to world transform for the currently active camera. This can be considered as the world transform of the camera object itself.

Examples

```
local invViewMat = Matrix44.Create()
Camera.GetInverseViewMatrix(invViewMat)
```

See Also

- Camera.GetViewMatrix
- Camera.GetProjectionMatrix

Camera.GetLookAtPosition

Brief

Retrieves the camera's world space target position.

Definition

```
Vector4 Camera.GetLookAtPosition( Camera camera )
Camera.GetLookAtPosition( Camera camera, Vector4 outPos )
Vector4 Camera.GetLookAtPosition()
Camera.GetLookAtPosition( Vector4 outPos )
```

Arguments

camera - The camera to query.outPos - (out) The vector to receive the target position.

Return Values

Returns a Vector4 specifying the world space target position of the camera.

Description

Retrieves the camera's world space target position. Only the X, Y and Z components of the Vector4 are valid. If no camera is specified it uses the currently active camera.

Examples

```
camtarget = Camera.GetLookAtPosition( gameCamera )
```

See Also

- Camera.SetLookAtPosition

Camera.GetPosition

Brief

Retrieves the camera's world space position.

Definition

```
Vector4 Camera.GetPosition( Camera camera )
Camera.GetPosition( Camera camera, Vector4 outPos )
Vector4 Camera.GetPosition()
Camera.GetPosition( Vector4 outPos )
```

Arguments

camera - The camera to query.outPos - (out) The vector to receive the camera position.

Return Values

A Vector4 specifying the world space position of the camera.

Description

Retrieves the camera's world space position. Only the X, Y and Z components of the Vector4 are valid. If no camera is specified

it uses the currently active camera. Note that inactive cameras are not updated, and therefore if a position is requested for an inactive camera the most recent position available will be returned. For the currently active camera the position will match the translation of the inverse view matrix.

Examples

```
campos = Camera.GetPosition( gameCamera )
```

See Also

- Camera.SetPosition
- Camera.GetInverseViewMatrix

Camera.GetProjectionMatrix

Brief

Get the projection matrix for the currently active camera.

Definition

```
Camera.GetProjectionMatrix(Matrix44 matrix)
```

Arguments

matrix - The matrix to set.

Return Values

None.

Description

Retrieves the projection transform for the currently active camera. A script should not assume a particular projection transform format. The projection matrix is used to transform a point from view to clip space.

Examples

```
local projMat = Matrix44.Create()
Camera.GetProjectionMatrix(projMat)
```

See Also

- Camera.GetViewMatrix
- Camera.GetInverseViewMatrix

Camera.GetViewMatrix

Brief

Get the world to view transform for the currently active camera.

Definition

```
Camera.GetViewMatrix(Matrix44 matrix)
```

Arguments

matrix - The matrix to set.

Return Values

None.

Description

This matrix will transform from world space to view space.

Examples

```
local worldToViewMat = Matrix44.Create()
Camera.GetViewMatrix(worldToViewMat)
```

See Also

- Camera.GetInverseViewMatrix
- Camera.GetProjectionMatrix

Camera.GetViewPortFOV

Brief

Retrieves the vertical field of view for the specified camera.

Definition

```
number Camera.GetViewPortFOV( Camera camera )
number Camera.GetViewPortFOV()
```

Arguments

camera - The camera to query.

Return Values

The vertical field of view for the camera, specified in degrees.

Description

Retrieves the vertical field of view (FOV) for the specified camera. If no camera is specified it uses the currently active camera.

Examples

```
fov = Camera.GetViewPortFOV( gameCamera )
```

See Also

- Camera.SetViewPortFOV

Camera.GetViewPortZFar

Brief

Retrieves the camera's far clip plane.

Definition

```
number Camera.GetViewPortZFar( Camera camera )
number Camera.GetViewPortZFar()
```

Arguments

camera - The camera to query.

Return Values

The far clip plane of the camera in units of meters.

Description

Retrieves the camera's far clip plane. Units are specified in meters. The default far clip plane for a newly created camera is 4000.0. If no camera is specified it uses the currently active camera.

Examples

```
farz = Camera.GetViewPortZFar( gameCamera )
```

See Also

- Camera.SetViewPortZFar

Camera.GetViewPortZNear

Brief

Retrieves the camera's near clip plane.

Definition

```
number Camera.GetViewPortZNear( Camera camera )
number Camera.GetViewPortZNear()
```

Arguments

camera - The camera to query.

Return Values

The near clip plane of the camera in units of meters.

Description

Retrieves the camera's near clip plane. Units are specified in metres. The default near clip plane for a newly created camera is 0.1. If no camera is specified it uses the currently active camera.

Examples

```
nearz = Camera.GetViewPortZNear( gameCamera )
```

See Also

- Camera.SetViewPortZNear

Camera.IsActivated

Brief

Determines whether the specified camera is currently active.

Definition

```
boolean Camera.IsActivated( Camera camera )
```

Arguments

camera - The camera to query.

Return Values

true if the specified camera is activated, false otherwise.

Description

Determines whether the specified camera is currently active.

Examples

```
if Camera.IsActivated( gameCamera ) then
    Camera.Dectivate( gameCamera, 1.0 )
end
```

See Also

- Camera.Activate
- Camera.Deactivate

Camera.SetLookAtPosition

Brief

Sets the camera's world space target position.

Definition

```
Camera.SetLookAtPosition( Camera camera, Vector4 target )
```

Arguments

camera - The camera to update.target - Vector4 specifying the world space target position for the camera.

Return Values

None.

Description

Sets the camera's world space target position. Only the X, Y and Z components of the Vector4 are used to position the camera. The W component is ignored.

Examples

```
Camera.SetLookAtPosition( gameCamera, Vector4.Create( 0, 10, 0, 0 ) )
```

See Also

- Camera.GetLookAtPosition

Camera.SetPosition

Brief

Sets the camera's world space position.

Definition

```
Camera.SetPosition( Camera camera, Vector4 position )
```

Arguments

camera - The camera to update.position - Vector4 specifying the world space position of the camera.

Return Values

None.

Description

Sets the camera's world space position. Only the X, Y and Z components of the Vector4 are used to position the camera. The W component is ignored.

Examples

```
Camera.SetPosition( gameCamera, Vector4.Create( 0, 10, 0, 0 ) )
```

See Also

- Camera.GetPosition

Camera.SetViewPortAspect

Brief

Sets the aspect ratio for the specified camera's viewport.

Definition

```
Camera.SetViewPortAspect( Camera camera, number aspect )
```

Arguments

camera - The camera to update.aspect - aspect ratio for the camera's viewport (width / height).

Return Values

None.

Description

Sets the aspect ratio for the specified camera's viewport.

Examples

```
Camera.SetViewPortAspect( gameCamera, 16.0 / 9.0 )
```

See Also

Camera.SetViewPortFOV

Brief

Sets the vertical field of view of the specified camera.

Definition

```
Camera.SetViewPortFOV( Camera camera, number fov )
```

Arguments

camera - The camera to update. fov - Vertical field of view for the camera, specified in degrees.

Return Values

None.

Description

Sets the vertical field of view of the specified camera in degrees.

Examples

```
Camera.SetViewPortFOV( gameCamera, 60.0 )
```

See Also

- Camera.GetViewPortFOV

Camera.SetViewPortZFar

Brief

Sets the camera's far clip plane.

Definition

```
Camera.SetViewPortZFar( Camera camera, number zfar )
```

Arguments

camera - The camera to update. znear - The far clip plane of the camera in units of metres.

Return Values

None.

Description

Sets the camera's far clip plane. Units are specified in metres. The default far clip plane for a newly created camera is 4000.0.

Examples

```
Camera.SetViewPortZFar( gameCamera, 5000.0 )
```

See Also

- Camera.GetViewPortZFar

Camera.SetViewPortZNear

Brief

Sets the camera's near clip plane.

Definition

```
Camera.SetViewPortZNear( Camera camera, number znear )
```

Arguments

camera - The camera to update.znear - The near clip plane of the camera in units of meters.

Return Values

None.

Description

Sets the camera's near clip plane. Units are specified in metres. The default near clip plane for a newly created camera is 0.1. The minimum value that the function will accept is 0.001.

Examples

```
Camera.SetViewPortZNear( gameCamera, 0.2 )
```

See Also

- Camera.GetViewPortZNear

Cloth

Cloth.SetWindAngle

Brief

Set the wind angle for all cloth meshes.

Definition

```
Cloth.SetWindAngle(number angle)
```

Arguments

angle - The wind angle in degrees.

Return Values

None.

Description

Set the angle for the wind simulation applied to cloth meshes. The angle is measured such that 0 degrees points down the positive X axis, and moves anti-clockwise so that 90 is +Y, 180 is -X and 270 is -Y. All cloth meshes have the same wind angle.

Examples

```
local cloth = myEntity:GetCloth()
if (cloth) then
    cloth:SetWindAngle(135)
end
```

See Also

- Entity.GetCloth

Clubs

Clubs.GetClubCount

Brief

Returns the number of clubs the current user is associated with.

Definition

```
number Clubs.GetClubCount()
```

Arguments

None

Return Values

The number of clubs.

Description

Returns The number of clubs the current user belongs to, or is associated with via invites or requests. Users are listed as "members" of clubs that they are currently invited to or have requested membership to, even if the invite or request has not been accepted. In this case, the member's status field will indicate this.

Examples

```
local numClubs = Clubs.GetClubCount()
print('The user belongs to ' .. numClubs .. ' clubs')
```

See Also

Clubs.GetClubInfo

Brief

Returns the full data relating to a specified club to which the user belongs.

Definition

```
table Clubs.GetClubInfo(number index | string clubId)
```

Arguments

index - The index of which club to get information from. Indicies start at 1.clubId - The club ID to get club information from. This can be retrieved from Clubs.GetClubList.

Return Values

Information about the specified club in the form of a table, or nil if the clubId is not accessible to this user.

Description

Retrieves the full club data for a specified club that the user belongs to. The format of the table is:
{
string id,
string name,
string tag,
number memberCount,
ClubRole role,
ClubMemberStatus status,
Time dateCreated,
string description
}

The role can be one of the following values:
ClubRole.Unknown
ClubRole.NonMember
ClubRole.Member
ClubRole.Subleader
ClubRole.Leader

The status can be one of the following values:
ClubMemberStatus.Member
ClubMemberStatus.Invited
ClubMemberStatus.Requested
The dateCreated is the date/time of the club creation, in server time in UTC. The status and role are for the current user. The club membership data may be a maximum of 1 hour out of date, as it is based on a local cache of the PSN club data, which is periodically refreshed. All data is read only. Only club IDs obtained from Clubs.GetClubList() are valid, any others will generate a Lua error.

Examples

```
local clubs = Clubs.GetClubList()
print('User is in ' .. #clubs .. ' clubs:')
for i, club in ipairs(clubs) do
    local info = Clubs.GetClubInfo(club.id)
    print('Club ' .. club.name .. ' description: ' .. info.description)
end
```

See Also

- Clubs.GetClubList
- Clubs.GetMemberList
- Clubs.GetMemberInfo

Clubs.GetClubList

Brief

Returns the list of Home clubs to which the current user belongs.

Definition

```
table[] Clubs.GetClubList()
```

Arguments

None

Return Values

The information as an array (integer-indexed table) of tables.

Description

Obtains the list of Home clubs to which the current user belongs, returned as an array of tables. Each entry in the array is of the following form:
{
string id,
string name,
string tag,
number memberCount,
ClubRole role,
ClubMemberStatus status
}

The role can be one of the following values:
ClubRole.Unknown
ClubRole.NonMember
ClubRole.Member
ClubRole.Subleader
ClubRole.Leader

The status can be one of the following values:
ClubMemberStatus.Member
ClubMemberStatus.Invited
ClubMemberStatus.Requested
The club membership data may be a maximum of 1 hour out of date, as it is based on a local cache of the PSN club data, which is periodically refreshed. All data is read only. The status and role are for the current user.

Examples

```
local clubs = Clubs.GetClubList()
print('User is in ' .. #clubs .. ' clubs:')
for i, club in ipairs(clubs) do
    print('Club ' .. club.id .. ' ' .. club.name)
end
```

See Also

- Clubs.GetClubInfo
- Clubs.GetMemberList
- Clubs.GetMemberInfo

Clubs.GetInstanceMemberList

Brief

Returns the list of members of a specified club that are within the current instance.

Definition

```
table[] Clubs.GetInstanceMemberList(number index | string clubId)
```

Arguments

index - The index of which club to get information from. Indicies start at 1.clubId - The club ID to get club information from. This can be retrieved from Clubs.GetClubList.

Return Values

The information as an array (integer-indexed table) of tables, or nil if the clubId is not accessible to this user.

Description

Obtains the list of club members within the current instance at the time of calling, returned as an array of tables. The local player is implicit and is not included in the list. Each entry in the arry is of the following form:
{

Person person,
ClubRole role,
ClubMemberStatus status
}

The role can be one of the following values:
ClubRole.Unknown
ClubRole.NonMember
ClubRole.Member
ClubRole.Subleader
ClubRole.Leader

The status can be one of the following values:
ClubMemberStatus.Member
ClubMemberStatus.Invited
ClubMemberStatus.Requested
The member list data may be a maximum of 1 hour out of date, as it is based on a local cache of the PSN club data, which is periodically refreshed. All data is read only. Only club IDs obtained from Clubs.GetClubList() are valid, any others will generate a Lua error. Note that users are listed as "members" of clubs that they are currently invited to or have requested membership to, even if the invite or request has not been accepted. In this case, the member's status field will indicate this. Only users who are full members of the specified club will be able to retrieve the member list, for invited or pending users this function will return nil.

Examples

```
local clubs = Clubs.GetClubList()
local members = Clubs.GetInstanceMemberList(clubs[1].id)
for j, member in ipairs(members) do
    print('  ' .. member.user:GetName())
end
```

See Also

- Clubs.GetClubList
- Clubs.GetMemberInfo
- Clubs.GetMemberList

Clubs.GetMemberCount

Brief

Obtain the number of members in a given club.

Definition

```
number Clubs.GetMemberCount(number index | string clubId)
```

Arguments

index - club indexclubId - club ID obtained from GetClubList

Return Values

The number of members in the club, or nil if the clubId is not accessible to this user.

Description

If the user is not a full member of the club, this function will return the number of full members of the club, not including any pending or invited members. If the user is a full member of the club it returns the total number of users that are associated with the club. This includes users that have been invited or have requested to join, but are not full members.

Examples

```
if (Clubs.GetMemberCount(0) == 32) then
    print("Club 0 is full!")
end
```

See Also

- Clubs.GetClubList

Clubs.GetMemberInfo

Brief

Returns the member data, if any, relating to a specified user/person in a club the they belongs to.

Definition

```
table Clubs.GetMemberInfo(string clubId, User|Person user)
```

Arguments

clubId - the ID of the club to retrieve the member info foruser - a User or Person object identifying the user to retrieve membership info for

Return Values

The information as a table or nil if the user is not associated with the club, or nil if the clubId is not accessible to this user.

Description

Returns the member data, if any, relating to a specified user in a club the user belongs to. If the user is a member of the club, a table of the following form is returned:
{
ClubRole role,
ClubMemberStatus status
}

The role can be one of the following values:
ClubRole.Unknown
ClubRole.NonMember
ClubRole.Member
ClubRole.Subleader
ClubRole.Leader

The status can be one of the following values:
ClubMemberStatus.Member
ClubMemberStatus.Invited
ClubMemberStatus.Requested
The club membership data may be a maximum of 1 hour out of date, as it is based on a local cache of the PSN club data, which is periodically refreshed. All data is read only. Only club IDs obtained from Clubs.GetClubList() are valid, any others will generate a Lua error. Only users who are full members of the specified club will be able to retrieve the member info, for invited or pending users this function will return nil.

Examples

```
local clubs = Clubs.GetClubList()
if (#clubs > 0) then
    local friends = LocalPlayer.GetFriends()
    print('The following of your friends also belong to the ' .. clubs[1].name .. ' club:')
    for i, friend in ipairs(friends) do
        local memInfo = Clubs.GetMemberInfo(clubs[1].id, friend)
        if (memInfo ~= nil and memInfo.status == ClubMemberStatus.Member) then
            print('  ' .. friend:GetName())
        end
    end
end
```

See Also

Clubs.GetMemberList

Brief

Returns the member list of a specified club to which the current user belongs.

Definition

```
table[] Clubs.GetMemberList(number index | string clubId)
```

Arguments

None

Return Values

The information as an array (integer-indexed table) of tables, , or nil if the clubId is not accessible to this user.

Description

Obtains the member list of a club to which the current user belongs, returned as an array of tables. Each entry in the array is of the following form:
{
User user,
ClubRole role,
ClubMemberStatus status
}

The role can be one of the following values:
ClubRole.Unknown
ClubRole.NonMember
ClubRole.Member
ClubRole.Subleader
ClubRole.Leader

The status can be one of the following values:
ClubMemberStatus.Member
ClubMemberStatus.Invited
ClubMemberStatus.Requested
The member list data may be a maximum of 1 hour out of date, as it is based on a local cache of the PSN club data, which is periodically refreshed. All data is read only. Only club IDs obtained from Clubs.GetClubList() are valid, any others will generate a Lua error. Note that users are listed as "members" of clubs that they are currently invited to or have requested membership to, even if the invite or request has not been accepted. In this case, the member's status field will indicate this. Only users who are full members of the specified club will be able to retrieve the member list, for invited or pending users this function will return nil.

Examples

```
local clubs = Clubs.GetClubList()
for i, club in ipairs(clubs) do
    print('Members of ' .. club.name .. ':')
    local members = Clubs.GetMemberList(club.id)
    for j, member in ipairs(members) do
        print('  ' .. member.user:GetName())
    end
end
```

See Also

- Clubs.GetClubList
- Clubs.GetClubInfo
- Clubs.GetMemberInfo

Clubs.IsRefreshed

Brief

Returns true if the club data is up-to-date and false otherwise.

Definition

```
boolean Clubs.IsRefreshed()
```

Arguments

None

Return Values

true if the club data is up-to-date, false if data is not up-to-date or is currently refreshing.

Description

Checks if the club data is up-to-date and ready to be accessed. If a refresh has been requested, this function will return false, and if the data is out-of-date it will also return false.

Examples

```
if (Clubs.RequestRefresh()) then
    state = RefreshState.Refreshing     -- custom user "enum"
end
if (state == RefreshState.Refreshing and Clubs.IsRefreshed()) then
    print('Club data up to date')
    state = RefreshState.Idle
end
```

See Also

- Clubs.RequestRefresh

Clubs.IsRefreshing

Brief

Returns true if the club data is currently refreshing.

Definition

```
boolean Clubs.IsRefreshing()
```

Arguments

None

Return Values

true if the club data is refreshing, false if the club data system is idle.

Description

Checks if the club data is currently refreshing. If a refresh has been requested, this function will return true until the data is full refreshed. The old club data is still valid while the refresh is in progress.

Examples

```
if (Clubs.IsRefreshing()) then
    ShowRefreshingClubsIcon()       -- user function
end
```

See Also

- Clubs.RequestRefresh
- Clubs.IsRefreshed

Clubs.RequestRefresh

Brief

Request an update of the club data.

Definition

```
boolean Clubs.RequestRefresh()
```

Arguments

None

Return Values

true if the refresh was started, false if the club data is already up-to-date.

Description

Requests a refresh of the local club data cache. If false is returned, the club data is already up-to-date and can be used by the script. The current club data remains accessible while the refresh is taking place and is atomically replaced once complete, so scripts can rely on the data being consistent during the refresh.

Examples

```
if (Clubs.RequestRefresh()) then
    state = RefreshState.Refreshing     -- custom user "enum"
end
if (state == RefreshState.Refreshing and Clubs.IsRefreshed()) then
    print('Club data up to date')
    state = RefreshState.Idle
end
```

See Also

- Clubs.IsRefreshed

Collision

Collision.Create

Brief

Creates a new trigger volume.

Definition

```
TriggerVolume Collision.CreateTriggerVolume( string triggerVolumeName, number maxOverlaps, enum
shapeType, number shapeDim1, ... )
TriggerVolume Collision.CreateTriggerVolume( string triggerVolumeName, number maxOverlaps, string
shapeResourceName )
TriggerVolume Collision.CreateTriggerVolume( string triggerVolumeName, number maxOverlaps, Resource
shapeResource )
```

Arguments

triggerVolumeName - The name to give to the trigger volume.maxOverlaps - The maximum number of overlaps to process.shapeType - The TriggerVolumeShape enum defining the shape of the trigger volume.shapeDimN - The Nth dimension of the shape of the trigger volume.shapeResourceName - The name of the collision resource representing the shape of the trigger volume.shapeResource - The collision resource representing the shape of the trigger volume.

Return Values

A new TriggerVolume instance.

Description

Creates a new trigger volume. Valid values for shapeType are:
TriggerVolumeShape.Sphere
TriggerVolumeShape.Box
TriggerVolumeShape.Cylinder
TriggerVolumeShape.Capsule

Examples

```
function MyTriggerVolumeCallback1( triggerVolume, state, count )
    if state == TriggerVolumeOverlapState.Enter then
        print( count .. " new overlaps for trigger volume " .. tostring( triggerVolume ) .. " this
frame" )
    elseif state == TriggerVolumeOverlapState.Exit then
        print( count .. " old overlaps for trigger volume " .. tostring( triggerVolume ) .. " this
frame" )
    end
end

function MyTriggerVolumeCallback2( triggerVolume, state, count )
    for i = 1, count do
        local   overlapType = TriggerVolume.GetOverlapType( triggerVolume, i )
        local   overlapData = TriggerVolume.GetOverlapData( triggerVolume, i )
        print( "trigger volume " .. tostring( triggerVolume ) .. " overlap " .. tostring( i ) .. ":
type=" .. tostring( overlapType ) .. ", object=" .. tostring( overlapData.object ) )
    end
end

triggerVolumes = {}
triggerVolumes[1] = Collision.CreateTriggerVolume( "simpleTriggerVolume1", 16,
TriggerVolumeShape.Sphere, 1.0 )
triggerVolumes[2] = Collision.CreateTriggerVolume( "simpleTriggerVolume2", 16, TriggerVolumeShape.Box,
3.0, 2.0, 4.0 )
triggerVolumes[3] = Collision.CreateTriggerVolume( "resourceTriggerVolume1", 16,
"kTriggerVolumeShapeResource1" )
triggerVolumes[4] = Collision.CreateTriggerVolume( "resourceTriggerVolume2", 16, Resource.Find(
"kTriggerVolumeShapeResource2" ) )

TriggerVolume.SetLocalMatrix( triggerVolumes[1], triggerVolumeTransformMatrix[1] )
TriggerVolume.SetLocalMatrix( triggerVolumes[2], TriggerVolume.GetLocalMatrix( triggerVolumes[1] ) )
TriggerVolume.SetPosition( triggerVolumes[3], triggerVolumePosition[1] )
TriggerVolume.SetPosition( triggerVolumes[4], TriggerVolume.GetPosition( triggerVolumes[3] ) )

for i = 1, #triggerVolumes do
    TriggerVolume.AddCallback( triggerVolumes[i], MyTriggerVolumeCallback1 )
    TriggerVolume.AddCallback( triggerVolumes[i], MyTriggerVolumeCallback2 )
    if TriggerVolume.IsLayerOverlapsEnabled( triggerVolumes[i], CollisionLayer.Scene ) then
        TriggerVolume.SetLayerOverlapsEnabled( triggerVolumes[i], CollisionLayer.Scene, false )
    end
    if not TriggerVolume.IsEnabled( triggerVolumes[i] ) then
        TriggerVolume.SetEnabled( triggerVolumes[i], true )
    end
end

TriggerVolume.RemoveCallback( triggerVolumes[3], MyTriggerVolumeCallback2 )
TriggerVolume.RemoveAllCallbacks( triggerVolumes[4] )

local   numOverlaps = TriggerVolume.GetOverlapCount( triggerVolumes[1] )

for i = 1, numOverlaps do
    local   overlapType = TriggerVolume.GetOverlapType( triggerVolumes[1], i )
    local   overlapData = TriggerVolume.GetOverlapData( triggerVolumes[1], i )
    print( "trigger volume " .. tostring( triggerVolumes[1] ) .. " overlap " .. tostring( i ) .. ":
type=" .. tostring( overlapType ) .. ", object=" .. tostring( overlapData.object ) )
end
```

See Also

None

Collision.CreateLinearCast

Brief

Create a linear cast object of a specified shape.

Definition

```
LinearCast Collision.CreateLinearCast(ShapeType|string|Resource shape, ..., number maxOverlaps = 1)
```

Arguments

shape - Either a ShapeType enum value, or a string or resource representing a collision shape.
If a ShapeType is specified, the value defines the immediately following parameters:
ShapeType.Sphere: number radius - sphere radius.
ShapeType.Box: Vector4 halfExtents - X, Y, Z half-extents.
ShapeType.Capsule: Vector4 point1, Vector4 point2, number radius - capsule points and radius.maxOverlaps - Maximum number of overlaps to store, must be at least 1.

Return Values

A LinearCast object if successful, or nil otherwise.

Description

A linear cast will sweep a shape through a physics world and return collision items that overlap the swept volume. Hits will be stored until maxOverlaps hits have been recorded, at which point the sweet will stop. More complex shapes will be less efficient to sweep through the world.

Examples

```
cast = Collision.CreateLinearCast(ShapeType.Box, Vector4.Create(0.2, 0.8, 0.2), 5)      -- create cast
of box of half-extents [0.2, 0.8, 0.2], 5 max overlaps
rot = Matrix44.Create()
rot:SetRotationXyz(0, 45, 0)
cast:SetRotation(rot)
cast:Start(Vector4.Create(0,0,0), Vector4.Create(5,0,0))
while (not cast:IsReady()) do
    coroutine.yield()
end
local v = Vector4.Create()
for i = 1, cast:GetHitCount() do
    cast:GetHitPoint(i, v)
    print("Hit point:", v)
    cast:GetHitNormal(i, v)
    print("Hit normal:", v)
    local type = cast:GetHitType(i)
    print("Hit type:", type)
    print("Hit distance:", cast:GetHitDistance(i))
    local data = cast:GetHitData(i)
    if (type == HitType.Entity) then
        -- data is an entity
    elseif (type == HitType.Person) then
        -- data is a person
    elseif (type == HitType.Furniture) then
        -- data is a table (see GetHitData)
    elseif (type == HitType.Scene) then
        -- data is user ID of scene feature hit
    else
        -- data is nil - no further info available
    end
end
```

See Also

- LinearCast.Start

Collision.CreateRaycast

Brief

Creates a new Raycast object.

Definition

```
Raycast Collision.CreateRaycast()
```

Arguments

None

Return Values

A new Raycast instance.

Description

Creates a new Raycast object.

Examples

```
raycast = Collision.CreateRaycast()
```

See Also

- Raycast.Start

Debug

Debug.ArcadeGameGetActive

Brief

Returns the object of the currently active arcade game.

Definition

```
Object Debug.ArcadeGameGetActive()
```

Arguments

None

Return Values

The object of the currently active arcade game and nil if there is not one.

Description

Returns the object of the currently active arcade game and nil if there is not one.

Examples

```
arcadeGame = Debug.ArcadeGameGetActive()
print( "The active game is " .. Object.GetObjectId( arcadeGame ) )
```

See Also

- Object.GetObjectId

Debug.ArcadeGameLoad

Brief

Loads the specified arcade game.

Definition

```
Debug.ArcadeGameLoad( string objectId )
```

Arguments

objectId - The object id of the arcade game.

Return Values

None

Description

Loads the specified arcade game.

Examples

```
-- The object ID of the game as shown in Object Editor
myGame = "4626CD25-2B27433A-983EB874-7061BBAB"
Debug.ArcadeGameLoad( myGame )
```

See Also

None

Debug.ArcadeGameReset

Brief

Resets the currently playing arcade game.

Definition

```
Debug.ArcadeGameReset()
```

Arguments

None

Return Values

None

Description

Resets the currently playing arcade game. As part of the reset process, the arcade game object is completely reloaded, including all script resources. This removes the need for reloading the Home runtime during the development of an arcade game. Typically, this function is called from the debug console (via the 'lc' command), or from a script run from the console via the RunScript function.

Examples

```
function ResetArcadeGameThread()
    while ( true ) do
        if Debug.KeyIsPressed( KEY_F2 ) then
            Debug.ArcadeGameReset()
        end
    end
end

ResetArcadeGameThreadId = SpawnFunction( ResetArcadeGameThread )
```

See Also

- Debug.KeyIsPressed
- Debug.MiniGameReset
- Debug.SceneReset
- Debug.RunScript
- SpawnFunction

Debug.ArcadeGameSetBorderColor

Brief

Sets the color of the arcade game's border when running it fullscreen.

Definition

```
Debug.ArcadeGameSetBorderColor( Vector4 rgb )
```

Arguments

rgb - The color to set the border to.

Return Values

None

Description

Sets the color of the arcade game's left and right borders when running it in fullscreen mode (using the --game command line argument for HomeDeveloper.self). The screen space of a Home arcade game is fixed to 1024x720. However, the screen space of the entire display device is 1280x720 which leaves vertical borders on the left and right hand sides of the screen. By default, the border color is set to magenta.

Examples

```
-- Set border color to green
Debug.ArcadeGameSetBorderColor( Vector4.Create( 0, 1, 0 ) )

-- Set border color to magenta, the default color
Debug.ArcadeGameSetBorderColor( Vector4.Create( 1, 0, 1 ) )
```

See Also

- Vector4.Create

Debug.DebuggerRegister

Brief

Registers the Lua environment with the debugger.

Definition

```
Debug.DebuggerRegister()
```

Arguments

None

Return Values

None

Description

Registers the Lua environment with the debugger. When registered with the debugger you can debug the Lua script using SLED, the HDK's Lua debugger.

Examples

```
Debug.DebuggerRegister()
Debug.DebuggerSetupProject()
```

See Also

- Debug.DebuggerSetupProject
- Debug.DebuggerUnRegister

Debug.DebuggerSetupProject

Brief

Sets up the project for the debugger before connecting to the debugger.

Definition

```
Debug.DebuggerSetupProject( string projectName = "Home" )
```

Arguments

projectName - The name of the debugger project.

Return Values

None

Description

Sets up the project for the debugger before connecting to the debugger. All the files opened in the registered Lua environments are added to the project at the time the function is called.

Examples

```
Debug.DebuggerRegister()
Debug.DebuggerSetupProject()
```

See Also

- Debug.DebuggerRegister
- Debug.DebuggerUnRegister

Debug.DebuggerUnRegister

Brief

Unregisters the Lua environment with the debugger.

Definition

```
Debug.DebuggerUnRegister()
```

Arguments

None

Return Values

None

Description

Unregisters the Lua environment with the debugger. By default the environment is unregistered.

Examples

```
Debug.DebuggerUnRegister()
```

See Also

- Debug.DebuggerRegister
- Debug.DebuggerSetupProject

Debug.DrawBox3d

Brief

Draws a 3-dimensional box in the scene.

Definition

```
Debug.DrawBox3d( Matrix44 transform, Vector4 rgba )
```

Arguments

transform - Transform of box.rgba - The color of the box.

Return Values

None

Description

Draws a 3-dimensional box in the scene. For debug purposes only. The origin is at the center of the box.

Examples

```
matrix = Matrix44.Create()
Matrix44.SetScale( matrix, 1, 0.4, 3, 1 )
Matrix44.SetRotationXyz( matrix, 0, 30, 18, 0 )
Matrix44.SetTranslation( matrix, 0, 2, 0, 0 )
Debug.DrawBox3d( matrix, Vector4.Create( 1, 0, 1, 0.5 ) )
```

See Also

- Debug.DrawSphere3d
- Debug.DrawLine3d
- Debug.DrawCone3d
- Debug.DrawCylinder3d

Debug.DrawCone3d

Brief

Draws a 3-dimensional Cone in the scene.

Definition

```
Debug.DrawCone3d( Matrix44 transform, Vector4 rgba )
```

Arguments

transform - Transform of Cone.rgba - The color of the cone.

Return Values

None

Description

Draws a 3-dimensional Cone in the scene. For debug purposes only. The origin is the top of the cone and the cone points down the Z axis.

Examples

```
matrix = Matrix44.Create()
Matrix44.SetScale( matrix, 1, 0.4, 3, 1 )
Matrix44.SetRotationXyz( matrix, 0, 30, 18, 0 )
Matrix44.SetTranslation( matrix, 0, 2, 0, 0 )
Debug.DrawCone3d( matrix, Vector4.Create( 1, 0, 1, 0.5 ) )
```

See Also

- Debug.DrawSphere3d
- Debug.DrawLine3d
- Debug.DrawBox3d
- Debug.DrawCylinder3d

Debug.DrawCylinder3d

Brief

Draws a 3-dimensional Cylinder in the scene.

Definition

```
Debug.DrawCylinder3d( Matrix44 transform, Vector4 rgba )
Debug.DrawCylinder3d( Vector4 point0, Vector4 point1, number radius, Vector4 rgba )
```

Arguments

transform - Transform of Cylinder.rgba - The color of the cylinder.point0 - One end of the cylinder axispoint1 - Opposite end of the cylinder axisradius - Radius of the cylinder

Return Values

None

Description

Draws a 3-dimensional Cylinder in the scene. For debug purposes only. The origin is the center of its first side.

Examples

```
matrix = Matrix44.Create()
Matrix44.SetScale( matrix, 1, 0.4, 3, 1 )
Matrix44.SetRotationXyz( matrix, 0, 30, 18, 0 )
Matrix44.SetTranslation( matrix, 0, 2, 0, 0 )
Debug.DrawCylinder3d( matrix, Vector4.Create( 1, 0, 1, 0.5 ) )
```

See Also

- Debug.DrawSphere3d
- Debug.DrawLine3d
- Debug.DrawCone3d
- Debug.DrawBox3d

Debug.DrawLine3d

Brief

Draws a 3-dimensional line in the scene.

Definition

```
Debug.DrawLine3d( Vector4 start, Vector4 end, Vector4 rgba )
```

Arguments

start - The first point of the line.end - The last point of the line.rgba - The color of the line.

Return Values

None

Description

Draws a 3-dimensional line in the scene. For debug purposes only.

Examples

```
Debug.DrawLine3d( Vector4.Create( 0, 0, 0 ), Vector4.Create( 10, 10, 0 ), Vector4.Create( 1, 0, 0, 1 )
)
```

See Also

- Debug.DrawRect
- Debug.DrawText

Debug.DrawQuad3d

Brief

Draws a 3-dimensional quad in the scene.

Definition

```
Debug.DrawQuad3d( Vector4 v1, Vector4 v2, Vector4 v3, Vector4 v4, Vector4 rgba, boolean isWireFrame =
false )
```

Arguments

v1 - First quad vertex.v2 - Second quad vertex.v3 - Third quad vertex.v4 - Fourth quad vertex.rgba - The color of the quad.isWireFrame - Render quad wire-frame or solid (default).

Return Values

None

Description

Draws a 3-dimensional quad in the scene. For debug purposes only.

Examples

```
-- Draw a solid red quad
v1 = Vector4.Create( 0, 0, 0 )
v2 = Vector4.Create( 10, 0, 0 )
v3 = Vector4.Create( 10, 10, 0 )
v4 = Vector4.Create( 0, 10, 0 )
rgba = Vector4.Create( 1, 0, 0, 1 )
Debug.DrawQuad3d( v1, v2, v3, v4, rgba, false )
```

See Also

- Debug.DrawSphere3d
- Debug.DrawLine3d
- Debug.DrawCone3d
- Debug.DrawBox3d
- Debug.DrawCylinder3d
- Debug.DrawTriangle3d

Debug.DrawRect

Brief

Debug function used to display a quad on the 2d overlay.

Definition

```
Debug.DrawRect( number x, number y, number w, number h, rgba = [1, 1, 1, 1] )
```

Arguments

x - The x-component of the top left screen space coordinate of the quad.y - The y-component of the top left screen space coordinate of the quad.w - The width in pixels of the quad.h - The height in pixels of the quad.rgba - The color for the rectangle.

Return Values

None

Description

This function is available to aid debugging, allowing the LUA script to display untextured quads on the main 2d overlay.

Examples

```
-- Render a 100x100 square in the top left corner of the screen
Debug.DrawRect( 0, 0, 100, 100 )
```

See Also

- Debug.DrawLine3d
- Debug.DrawText

Debug.DrawSphere3d

Brief

Draws a 3-dimensional Sphere in the scene.

Definition

```
Debug.DrawSphere3d( Matrix44 transform, Vector4 rgba )
Debug.DrawSphere3d( Vector4 center, number radius, Vector4 rgba )
```

Arguments

transform - Transform of the sphere.center - Center point of the sphere.radius - Sphere radius.rgba - The color of the sphere.

Return Values

None

Description

Draws a 3-dimensional Sphere in the scene. For debug purposes only. The origin is the center of the sphere.

Examples

```
matrix = Matrix44.Create()
Matrix44.SetScale( matrix, 1, 0.4, 3, 1 )
Matrix44.SetRotationXyz( matrix, 0, 30, 18, 0 )
Matrix44.SetTranslation( matrix, 0, 2, 0, 0 )
Debug.DrawSphere3d( matrix, Vector4.Create( 1, 0, 1, 0.5 ) )
```

See Also

- Debug.DrawLine3d
- Debug.DrawCone3d
- Debug.DrawBox3d
- Debug.DrawCylinder3d

Debug.DrawText

Brief

Debug function used to display text on the 2d overlay

Definition

```
Debug.DrawText( number x, number y, string message )
```

Arguments

x - Screen space x-coordinate marking the start position of the string.y - Screen space y-coordinate marking the start position of the string.message - Text to display.

Return Values

None

Description

This function is available to aid debugging, allowing the LUA script to display text on the main 2d overlay. The rendered text is near-white with a black drop shadow.

Examples

```
Debug.DrawText(200, 200, "Hello World!" )
```

See Also

- Debug.DrawLine3d
- Debug.DrawRect

Debug.DrawTriangle3d

Brief

Draws a 3-dimensional triangle in the scene.

Definition

```
Debug.DrawTriangle3d( Vector4 v1, Vector4 v2, Vector4 v3, Vector4 rgba, boolean isWireFrame = false )
```

Arguments

v1 - First triangle vertex.v2 - Second triangle vertex.v3 - Third triangle vertex.rgba - The color of the triangle.isWireFrame - Render triangle wire-frame or solid (default).

Return Values

None

Description

Draws a 3-dimensional triangle in the scene. For debug purposes only.

Examples

```
-- Draw a solid red triangle
v1 = Vector4.Create( 0, 0, 0 )
v2 = Vector4.Create( 10, 0, 0 )
v3 = Vector4.Create( 0, 10, 0 )
rgba = Vector4.Create( 1, 0, 0, 1 )
Debug.DrawTriangle3d( v1, v2, v3, rgba, false )
```

See Also

- Debug.DrawSphere3d
- Debug.DrawLine3d
- Debug.DrawCone3d
- Debug.DrawBox3d
- Debug.DrawCylinder3d
- Debug.DrawQuad3d

Debug.GetDevMode

Brief

Returns the user defined dev mode.

Definition

```
string Debug.GetDevMode()
```

Arguments

None

Return Values

The user defined dev mode.

Description

Returns the user defined dev mode specified with the command line argument --devmode such as --devmode lua_debug.

Examples

```
function Update()
    if ( Debug.GetDevMode() == "lua_debug" ) then
        print( "a = " .. a .. " b = " .. b )
    end
end
```

See Also

None

Debug.GetStack

Brief

Returns the stack as a string.

Definition

```
string Debug.GetStack()
```

Arguments

None

Return Values

The stack as a string.

Description

Returns the stack as a string. For debug purposes only.

Examples

```
print( "The stack at this point is " .. Debug.GetStack() )
```

See Also

None

Debug.GetThreadIds

Brief

Returns a table of thread IDs for all the Lua threads.

Definition

```
number[] Debug.GetThreadIds()
```

Arguments

None

Return Values

A table of numbers representing thread IDs.

Description

Returns a table of thread IDs for all the Lua threads.

Examples

```
ids = Debug.GetThreadIds()
```

See Also

None

Debug.GetWorldToScreen

Brief

Converts a 3d world position to a screen position.

Definition

```
Vector4 Debug.GetWorldToScreen( Vector4 worldPosition )
```

Arguments

worldPosition - The 3d position in the world.

Return Values

The screen space coordinate in the range [0,0] -> [1280,720].

Description

Converts a 3d world position to a screen position.

Examples

```
screenPos = Debug.GetWorldToScreen( worldPos )
```

See Also

None

Debug.KeyIsHeld

Brief

Returns true if the specified key is held and false otherwise.

Definition

```
boolean Debug.KeyIsHeld( number key )
```

Arguments

key - The key to test. Can be one of the following:

Function keys:
KEY_F1 to KEY_F24

Alpabetic keys:
KEY_A to KEY_Z

Numeric keys:
KEY_0 to KEY_9

Number pad numeric keys:
KEY_NUM0 to KEY_NUM9

Miscellaneous keys:
KEY_ESCAPE, KEY_GRAVE, KEY_MINUS, KEY_EQUALS,
KEY_BACK, KEY_TAB, KEY_LEFTBRACKET, KEY_RIGHTBRACKET,
KEY_RETURN, KEY_CAPSLOCK, KEY_SEMICOLON, KEY_QUOTE,
KEY_CONSOLE, KEY_LEFTSHIFT, KEY_BACKSLASH, KEY_COMMA,
KEY_PERIOD, KEY_FORWARDSLASH, KEY_RIGHTSHIFT,
KEY_LEFTCONTROL, KEY_LEFTWINDOWS, KEY_LEFTALT,
KEY_SPACE, KEY_RIGHTALT, KEY_RIGHTWINDOWS,
KEY_RIGHTCONTROL, KEY_PRINTSCREEN, KEY_SCROLLLOCK,
KEY_PAUSE, KEY_HOME, KEY_PAGEUP, KEY_END, KEY_PAGEDOWN,

KEY_DELETE, KEY_INSERT, KEY_UP, KEY_LEFT, KEY_DOWN,
KEY_RIGHT, KEY_NUMLOCK, KEY_NUMDIV, KEY_NUMMUL,
KEY_NUMSUB, KEY_NUMADD, KEY_NUMENTER, KEY_NUMDEL,
KEY_APP, KEY_POWER, KEY_NUMEQUALS, KEY_EXECUTE,
KEY_HELP, KEY_MENU, KEY_SELECT, KEY_STOP1, KEY_AGAIN,
KEY_UNDO, KEY_CUT, KEY_COPY, KEY_PASTE, KEY_FIND,
KEY_MUTE1, KEY_VOLUMEUP1, KEY_VOLUMEDOWN1,
KEY_NEXTTRACK, KEY_PREVTRACK, KEY_STOP2, KEY_PLAY,
KEY_MUTE2, KEY_VOLUMEUP2, KEY_VOLUMEDOWN2

Return Values

true if the specified key is held and false otherwise.

Description

Returns true if the specified key is held and false otherwise.

Examples

```
print( "Escape held = " .. Debug.KeyIsHeld( KEY_ESCAPE ) )
```

See Also

- Debug.KeyIsPressed
- Debug.KeyIsReleased

Debug.KeyIsPressed

Brief

Returns true if the specified key is pressed and false otherwise.

Definition

```
boolean Debug.KeyIsPressed( number key )
```

Arguments

key - The key to test. Can be one of the following:

Function keys:
KEY_F1 to KEY_F24

Alpabetic keys:
KEY_A to KEY_Z

Numeric keys:
KEY_0 to KEY_9

Number pad numeric keys:
KEY_NUM0 to KEY_NUM9

Miscellaneous keys:
KEY_ESCAPE, KEY_GRAVE, KEY_MINUS, KEY_EQUALS,
KEY_BACK, KEY_TAB, KEY_LEFTBRACKET, KEY_RIGHTBRACKET,
KEY_RETURN, KEY_CAPSLOCK, KEY_SEMICOLON, KEY_QUOTE,
KEY_CONSOLE, KEY_LEFTSHIFT, KEY_BACKSLASH, KEY_COMMA,
KEY_PERIOD, KEY_FORWARDSLASH, KEY_RIGHTSHIFT,
KEY_LEFTCONTROL, KEY_LEFTWINDOWS, KEY_LEFTALT,
KEY_SPACE, KEY_RIGHTALT, KEY_RIGHTWINDOWS,
KEY_RIGHTCONTROL, KEY_PRINTSCREEN, KEY_SCROLLLOCK,
KEY_PAUSE, KEY_HOME, KEY_PAGEUP, KEY_END, KEY_PAGEDOWN,
KEY_DELETE, KEY_INSERT, KEY_UP, KEY_LEFT, KEY_DOWN,
KEY_RIGHT, KEY_NUMLOCK, KEY_NUMDIV, KEY_NUMMUL,
KEY_NUMSUB, KEY_NUMADD, KEY_NUMENTER, KEY_NUMDEL,
KEY_APP, KEY_POWER, KEY_NUMEQUALS, KEY_EXECUTE,
KEY_HELP, KEY_MENU, KEY_SELECT, KEY_STOP1, KEY_AGAIN,

KEY_UNDO, KEY_CUT, KEY_COPY, KEY_PASTE, KEY_FIND,
KEY_MUTE1, KEY_VOLUMEUP1, KEY_VOLUMEDOWN1,
KEY_NEXTTRACK, KEY_PREVTRACK, KEY_STOP2, KEY_PLAY,
KEY_MUTE2, KEY_VOLUMEUP2, KEY_VOLUMEDOWN2

Return Values

true if the specified key is pressed and false otherwise.

Description

Returns true if the specified key is pressed and false otherwise. Note, the keyboard is set to text entry mode and key repeat will kick in if a key is held down for too long. Use Debug.SetKeyMode to change this behaviour.

Examples

```
print( "Escape pressed = " .. Debug.KeyIsPressed( KEY_ESCAPE ) )
```

See Also

- Debug.KeyIsHeld
- Debug.KeyIsReleased

Debug.KeyIsReleased

Brief

Returns true if the specified key is released and false otherwise.

Definition

```
boolean Debug.KeyIsReleased( number key )
```

Arguments

key - The key to test. Can be one of the following:

Function keys:
KEY_F1 to KEY_F24

Alpabetic keys:
KEY_A to KEY_Z

Numeric keys:
KEY_0 to KEY_9

Number pad numeric keys:
KEY_NUM0 to KEY_NUM9

Miscellaneous keys:
KEY_ESCAPE, KEY_GRAVE, KEY_MINUS, KEY_EQUALS,
KEY_BACK, KEY_TAB, KEY_LEFTBRACKET, KEY_RIGHTBRACKET,
KEY_RETURN, KEY_CAPSLOCK, KEY_SEMICOLON, KEY_QUOTE,
KEY_CONSOLE, KEY_LEFTSHIFT, KEY_BACKSLASH, KEY_COMMA,
KEY_PERIOD, KEY_FORWARDSLASH, KEY_RIGHTSHIFT,
KEY_LEFTCONTROL, KEY_LEFTWINDOWS, KEY_LEFTALT,
KEY_SPACE, KEY_RIGHTALT, KEY_RIGHTWINDOWS,
KEY_RIGHTCONTROL, KEY_PRINTSCREEN, KEY_SCROLLLOCK,
KEY_PAUSE, KEY_HOME, KEY_PAGEUP, KEY_END, KEY_PAGEDOWN,
KEY_DELETE, KEY_INSERT, KEY_UP, KEY_LEFT, KEY_DOWN,
KEY_RIGHT, KEY_NUMLOCK, KEY_NUMDIV, KEY_NUMMUL,
KEY_NUMSUB, KEY_NUMADD, KEY_NUMENTER, KEY_NUMDEL,
KEY_APP, KEY_POWER, KEY_NUMEQUALS, KEY_EXECUTE,
KEY_HELP, KEY_MENU, KEY_SELECT, KEY_STOP1, KEY_AGAIN,
KEY_UNDO, KEY_CUT, KEY_COPY, KEY_PASTE, KEY_FIND,
KEY_MUTE1, KEY_VOLUMEUP1, KEY_VOLUMEDOWN1,
KEY_NEXTTRACK, KEY_PREVTRACK, KEY_STOP2, KEY_PLAY,
KEY_MUTE2, KEY_VOLUMEUP2, KEY_VOLUMEDOWN2

Return Values

true if the specified key is released and false otherwise.

Description

Returns true if the specified key is released and false otherwise.

Examples

```
print( "Escape released = " .. Debug.KeyIsReleased( KEY_ESCAPE ) )
```

See Also

- Debug.KeyIsHeld
- Debug.KeyIsPressed

Debug.MemGetLargestBlock

Brief

Retrieves the largest block available within the budget.

Definition

```
Debug.MemGetLargestBlock( MemoryType memoryType = MemoryType.Main )
```

Arguments

memoryType - The memory type to check for this budget. Can be one of the following:

Memory Type:
MemoryType.Main
MemoryType.Host
MemoryType.Vram

Return Values

Size in kilobytes (KB) or 0 if there was an error.

Description

Gets the largest available memory block. This isn't guaranteed to be the size of an allocatable block due to alignment restrictions and headers used within the memory system. This is intended to be used as an indication of fragmentation within the budget.

Examples

```
block = Debug.MemGetLargestBlock( MemoryType.Main )
```

See Also

- Debug.MemGetTotal
- Debug.MemGetUsed

Debug.MemGetTotal

Brief

Retrieves the total size of the budget the object is using.

Definition

```
Debug.MemGetTotal( MemoryType memoryType = MemoryType.Main )
```

Arguments

memoryType - The memory type to check for this budget. Can be one of the following:

Memory Type:
MemoryType.Main
MemoryType.Host
MemoryType.Vram

Return Values

Size in kilobytes (KB) or 0 if there was an error.

Description

Gets the total size of the budget for the specified memory type. Defaults to main memory.

Examples

```
total = Debug.MemGetTotal( MemoryType.Main )
```

See Also

- Debug.MemGetUsed
- Debug.MemGetLargestBlock

Debug.MemGetUsed

Brief

Retrieves the amount of memory currently being used.

Definition

```
Debug.MemGetUsed( MemoryType memoryType = MemoryType.Main )
```

Arguments

memoryType - The memory type to check for this budget. Can be one of the following:

Memory Type:
MemoryType.Main
MemoryType.Host
MemoryType.Vram

Return Values

Size in kilobytes (KB) or 0 if there was an error.

Description

Gets the amount of used memory.

Examples

```
used = Debug.MemGetUsed( MemoryType.Main )
```

See Also

- Debug.MemGetTotal
- Debug.MemGetLargestBlock

Debug.MiniGameGetActive

Brief

Returns the mini game the user is joined to and nil if there is not one.

Definition

```
Object Debug.MiniGameGetActive()
```

Arguments

None

Return Values

The mini game the user is joined to and nil if there is not one.

Description

Returns the mini game the user is joined to and nil if there is not one.

Examples

```
miniGame = Debug.MiniGameGetActive()
print( "The active game is " .. Object.GetObjectId( miniGame ) )
```

See Also

- Object.GetObjectId

Debug.MiniGameGetClosest

Brief

Returns the closest mini game and nil if there is not one.

Definition

```
Object Debug.MiniGameGetClosest( boolean onlyJoinable = false )
```

Arguments

onlyJoinable - If true it only returns the closest joinable game.

Return Values

The closest mini game and nil if there is not one.

Description

Returns the closest mini game and nil if there is not one.

Examples

```
miniGame = Debug.MiniGameGetClosest()
print( "The closest game is " .. Object.GetObjectId( miniGame ) )
```

See Also

- Object.GetObjectId

Debug.MiniGameReset

Brief

Resets the closest mini-game to the player's avatar.

Definition

```
Debug.MiniGameReset( string resetType = nil )
```

Arguments

resetType - Can be 'all' or an object ID to reset. If nil it resets the closest game.

Return Values

None

Description

Resets the closest mini-game to the player's avatar. As part of the reset process, the mini-game object is completely reloaded, including all script resources. This removes the need for reloading the Home runtime during the development of a mini-game. Typically, this function is called from the debug console (via the 'lc' command), or from a script run from the console via the RunScript function.

Examples

```
function ResetMiniGameThread()
    while ( true ) do
        if Debug.KeyIsPressed( KEY_F3 ) then
            Debug.MiniGameReset()
        end
    end
end

ResetMiniGameThreadId = SpawnFunction( ResetMiniGameThread )
```

See Also

- Debug.ArcadeGameReset
- Debug.KeyIsPressed
- Debug.SceneReset
- Debug.RunScript
- SpawnFunction

Debug.MouseGetVelocityX

Brief

Returns the horizontal velocity of the mouse.

Definition

```
number Debug.MouseGetVelocityX()
```

Arguments

None

Return Values

The horizontal velocity of the mouse.

Description

Returns the horizontal velocity of the mouse. If the mouse if dragged to the left, it will return a negative number and if it is dragged to the right, it will return a positive number.

Examples

```
mouseX = Debug.MouseGetVelocityX()
```

See Also

- Debug.MouseGetVelocityY
- Debug.MouseIsButtonHeld

Debug.MouseGetVelocityY

Brief

Returns the vertical velocity of the mouse.

Definition

```
number Debug.MouseGetVelocityY()
```

Arguments

None

Return Values

The vertical velocity of the mouse.

Description

Returns the vertical velocity of the mouse. If the mouse is pushed forwards, it will return a negative number and if it is pulled backwards, it will return a positive number.

Examples

```
mouseY = Debug.MouseGetVelocityY()
```

See Also

- Debug.MouseGetVelocityX
- Debug.MouseIsButtonHeld

Debug.MouseIsButtonHeld

Brief

Returns true if the button is held and false otherwise.

Definition

```
boolean Debug.MouseIsButtonHeld( number button )
```

Arguments

button - The button to check if it is held. Can be one of the following:
MOUSE_BUTTON_LEFT
MOUSE_BUTTON_MIDDLE
MOUSE_BUTTON_RIGHT

Return Values

true if the button is held and false otherwise.

Description

Returns true if the button is held and false otherwise.

Examples

```
if ( Debug.MouseIsButtonHeld( MOUSE_BUTTON_RIGHT ) ) then
    print( "Right mouse button held" )
end
```

See Also

- Debug.MouseGetVelocityX
- Debug.MouseGetVelocityY

Debug.MouseIsButtonPressed

Brief

Returns true if the button is pressed and false otherwise.

Definition

```
boolean Debug.MouseIsButtonPressed( number button )
```

Arguments

button - The button to check if it is pressed. Can be one of the following:
MOUSE_BUTTON_LEFT
MOUSE_BUTTON_MIDDLE
MOUSE_BUTTON_RIGHT

Return Values

true if the button is pressed and false otherwise.

Description

Returns true if the button is pressed and false otherwise.

Examples

```
if Debug.MouseIsButtonPressed( MOUSE_BUTTON_RIGHT ) then
    print( "Right mouse button pressed" )
end
```

See Also

- Debug.MouseGetVelocityX
- Debug.MouseGetVelocityY

Debug.MouseIsButtonReleased

Brief

Returns true if the button is released and false otherwise.

Definition

```
boolean Debug.MouseIsButtonReleased( number button )
```

Arguments

button - The button to check if it is released. Can be one of the following:
MOUSE_BUTTON_LEFT
MOUSE_BUTTON_MIDDLE
MOUSE_BUTTON_RIGHT

Return Values

true if the button is released and false otherwise.

Description

Returns true if the button is released and false otherwise.

Examples

```
if Debug.MouseIsButtonReleased( MOUSE_BUTTON_RIGHT ) then
    print( "Right mouse button released" )
end
```

See Also

- Debug.MouseGetVelocityX
- Debug.MouseGetVelocityY

Debug.RunScript

Brief

Loads and runs the specified file.

Definition

```
boolean Debug.RunScript( string fileName )
```

Arguments

fileName - The name of the file to load.

Return Values

None

Description

Loads and runs the specified file. The path is relative to the build folder. If you are running scripts in objects then you must have the Lua files as lua resources and run them with Resource.Run instead.

Examples

```
Debug.RunScript( "myscripts/jumpthread.lua" )
```

See Also

- Resource.Run

Debug.SceneIsReady

Brief

Returns true if the scene is ready and false otherwise.

Definition

```
boolean Debug.SceneIsReady()
```

Arguments

None

Return Values

true if the scene is ready and false otherwise.

Description

Returns true if the scene is ready and false otherwise.

Examples

```
if Debug.SceneIsReady() == true then
    game = Debug.MiniGameGetClosest()
end
```

See Also

None

Debug.SceneReset

Brief

Resets the scene object for dynamic and scripted content.

Definition

```
Debug.SceneReset()
```

Arguments

None

Return Values

None

Description

Resets the scene object for dynamic and scripted content such as scene entity animations and the scene script.

Examples

```
Debug.SceneReset()
```

See Also

- Debug.ArcadeGameReset
- Debug.MiniGameReset

Debug.ScreenSetContent

Brief

Sets a content source or sources on the screen.

Definition

```
Debug.ScreenSetContent( Screen screen, string[] contentList )
Debug.ScreenSetContent( Screen screen, string content, number contentIndex = 1 )
```

Arguments

screen - The screen to update.contentList - An array of strings specifying multiple content sources.content - A string for the content source.contentIndex - The index to set the content to. Ranges from 1 to the number of content sources. Defaults to 1.

Return Values

None.

Description

Sets a content source or sources on the screen. The source string will be a path relative to the HDK's build root directory. Note that paths relative to the HDK's build root directory can only be used for testing/development purposes. Scripts uploaded to a content server must reference content served over HTTP and use the Screen.SetContent function.

Examples

```
-- Example1: Testing a video served from the host PC's hard drive for debug
localSource = "localvideo/testvideo.mp4"
Debug.ScreenSetContent( myScreen1, localSource )

-- Example 2: Play back a video served from an HTTP server
onlineSource = "http://mycontentserver.com/onlinevideo.mp4"
Screen.SetContent( myScreen2, onlineSource )
```

See Also

- Screen.SetContent

Debug.SetErrorCallback

Brief

Specify a function to be called in the event of a script error.

Definition

```
Debug.SetErrorCallback(string callbackFuncName)
```

Arguments

callbackFuncName - Name of global function to call when script environment errors.

Return Values

None.

Description

Sets a function to be called in the event of a script error. During development often a script will contain errors which cause the script to be shut down, and if the script has modified global state it is not given a chance to clean up these changes, which can make re-running the fixed script problematic. If this function is called, it will set the name of a global script function which will be called when the script environment errors, and can be used to perform this clean up. It is only necessary to call this function once upon script initialization, it will be retained until the environment is destroyed. Note that this function is not available outside of developer builds and is not a substitute for proper script clean up in the normal game situations (player leaving game, object destroyed, etc).

Examples

```
Debug.SetErrorCallback("MyErrorFunc")

function MyErrorFunc()
    myCustomCamera:Deactivate()
    myCustomCamera = nil
end
```

See Also

None

Debug.System

Brief

Runs the command string in the console window.

Definition

```
Debug.System( string command )
```

Arguments

command - The command to run.

Return Values

None

Description

Runs the command string in the console window.

Examples

```
Debug.System( "lc print( 'Hello' )" )
```

See Also

None

Entity

Entity.ApplyAngularImpulse

Brief

Applies an angular impulse to the entity.

Definition

```
Entity.ApplyAngularImpulse( Entity entity, Vector4 impulse )
```

Arguments

entity - The entity to update. impulse - World space vector specifying angular impulse to be applied to the entity.

Return Values

None.

Description

Applies an angular impulse to the entity around the center of mass, creating angular velocity.

Examples

```
-- Apply angular impulse around Y axis
Entity.ApplyAngularImpulse( entity, Vector4.Create( 0, 0.1, 0.0) )
```

See Also

None

Entity.ApplyImpulse

Brief

Applies an impulse around a point.

Definition

```
Entity.ApplyImpulse( Entity entity, Vector4 impulse )
Entity.ApplyImpulse( Entity entity, Vector4 impulse, Vector4 point )
```

Arguments

entity - The entity to update.impulse - World space vector specifying impulse (or force) to be applied to the entity.point - World space vector specifying the point around which the impulse acts.

Return Values

None.

Description

Applies an impulse around a point. If no point is specified, the center of mass of the entity's rigid body is used.

Examples

```
-- Apply an impulse around the entity's center of mass
Entity.ApplyImpulse( entity, Vector4.Create( 5, 10, 5 ) )

-- Apply an impulse around the origin of world space
Entity.ApplyImpulse( entity, Vector4.Create( 0, 10, 0 ), Vector4.Create( 0, 0, 0 ) )
```

See Also

None

Entity.AttachPersonLabel

Brief

Attach a person label to the entity.

Definition

```
Entity.AttachPersonLabel(Entity entity, Person person, Vector4 offset)
```

Arguments

person - The person to take the label from.offset - The offset of the label from the entity's position. (Optional. Defaults to 0, 0, 0).

Return Values

Description

This function will detach the person label from the given person, and attach it to the entity instead. Any number of person labels can be attached to the entity.

Examples

```
Entity.AttachPersonLabel(entity, person, labelOffset)
```

See Also

- Entity.DetachPersonLabel

Entity.AttachToBone

Brief

Attach the entity to the specified bone of its parent.

Definition

```
Entity.AttachToBone( Entity entity, string boneName )
```

Arguments

entity - The entity to update.boneName - The name of the bone to attach to or nil to clear it. Can be one of the following:
"pelvis"
"hips"
"leftleg"
"leftfoot"
"rightleg"
"rightfoot"
"spine"
"neck"
"head"
"leftshoulder"
"leftarm"
"leftforearm"
"lefthand"
"rightshoulder"
"rightarm"
"rightforearm"
"righthand"

Return Values

None.

Description

Attach the entity to the specified bone of its parent. In addition to specifying the bone to which the entity is to be attached, it is also necessary to specify the person to which the bone belongs. This is done by calling Entity.AttachToParent and supplying the desired person.

Examples

```
entity = Entity.Create()
Entity.AttachToParent( entity, LocalPlayer.GetPerson() )
Entity.AttachToBone( entity, "head" )
```

See Also

- Entity.Create
- Entity.AttachToParent
- Entity.SetAttachType

Entity.AttachToParent

Brief

Attaches the entity to the specified parent.

Definition

```
Entity.AttachToParent( Entity child, Entity parent, string boneName = nil )
Entity.AttachToParent( Entity child, Person parent, string boneName = nil )
```

Arguments

entity - The entity to update.parent - The parent to attach the entity to or nil to remove it.boneName - The name of the bone to attach to, if any. For a person, it must be from the following list:
"pelvis"
"hips"
"leftleg"

"leftfoot"
"rightleg"
"rightfoot"
"spine"
"neck"
"head"
"leftshoulder"
"leftarm"
"leftforearm"
"lefthand"
"rightshoulder"
"rightarm"
"rightforearm"
"righthand"

Return Values

None.

Description

Attaches the entity to the specified parent. The parent's world matrix will be applied to the entity so the entity's position, rotation and scale are relative to the parent. If a bone is specified, the entity will be attached to the specified bone in the parent's skeleton.

Examples

```
-- Parent an entity to an entity
entity1 = Entity.Create()
parent = Entity.Create()
Entity.AttachToParent( entity1, parent );

-- Parent an entity to a person's right hand
entity2 = Entity.Create()
person = Person.FindInInstance( personId )
Entity.AttachToParent( entity2, person, "righthand" )
```

See Also

- Entity.Create
- Person.FindInInstance
- Entity.SetAttachType

Entity.BlendAnimIn

Brief

Blends an animation into, and alongside the currently playing animations.

Definition

```
Entity.BlendAnimIn( Entity entity, string resourceName, boolean isLooping = true, number time = 0,
number easeType = AnimBlendType.Linear, number blendDuration = 0.25, number priority = 0, Animation
anim = nil)
Entity.BlendAnimIn( Entity entity, Resource animResource, boolean isLooping = true, number time = 0,
number easeType = AnimBlendType.Linear, number blendDuration = 0.25, number priority = 0, Animation
anim = nil)
```

Arguments

entity - entity to blend animation ontoresourceName - the name of the animation resource to blend inanimResource - the animation resource to blend inisLooping - flag indicating if the animation played should loop. (default is true)time - point in at which the animation should start playing from (default is 0.0)easeType - the easing method to be used. The values used are in the enumerated type AnimBlendType (default value is AnimBlendType.Linear)blendDuration - the period in seconds, over which the blend factor will increase from 0.0 to 1.0 (default is 0.25)priority - The priority affects where in the animation list an animation

will be played. The higher the number, the later it will be played.
If a prior animation is being played at the same priority, then the new animation will be placed after the old animation in the list. (default is 0)anim - if supplied, the animation handle is set to the newly created active animation

Return Values

None.

Description

Values for the ease type are provided by the enumeration AnimBlendType:
AnimBlendType.None
AnimBlendType.Linear
AnimBlendType.SCurve
AnimBlendType.Accelerate
AnimBlendType.Decelerate

Examples

```
Entity.BlendAnimIn( entity, "myAnim" )
Entity.BlendAnimIn( entity, "myWalk", true, 0.0, AnimBlendType.Linear, 0.25, 0, animHandleToSave)
```

See Also

- Entity.GetIndexOfAnim
- Entity.GetActiveAnim

Entity.BlendAnimOut

Brief

Blends the specified active animation out.

Definition

```
Entity.BlendAnimOut(Entity entity, Animation anim, number easeType = AnimBlendType.Linear, number
blendDuration = 0.25)
```

Arguments

entity - entity to blend anim out fromanim - animation to blend outeaseType - the blend curve to use when blending out, default is AnimBlendType.LinearblendDuration - the duration over which the animation is blended out, default is 0.25

Return Values

None.

Description

Begins to blend out a currently active animation. To check the progress of the blend, use Animation.GetBlendState and/or Animation.GetBlendTime. An invalid animation handle will have no effect.

Examples

```
local animIndex = Entity.GetIndexOfAnim( entity, "idleAnim" )
local hAnim = Entity.GetActiveAnim( entity, animIndex )
Entity.BlendAnimOut( entity, hAnim, AnimBlendType.Linear, 0.25 )
```

See Also

- Entity.BlendAnimIn
- Entity.GetIndexOfAnim

- Entity.GetActiveAnim
- Entity.PlayAnim
- Animation.GetBlendWeight

Entity.Copy

Brief

Clone an entity's state to another entity.

Definition

```
Entity.Copy(Entity dest, Entity source)
```

Arguments

dest - Destination entity to modify.source - Source entity to copy.

Return Values

None.

Description

Copies all state from the source entity to the destination entity, leaving the source entity unchanged. The only state that will not be copied is transform state (position and orientation), as having two identical entities in the same place may cause issues with collision and other systems. The destination entity is still a different entity (it will not compare equal and will have a different unique ID) but will have the same state as the source entity. Similarly to Reset, the hierarchy state of the destination entity will not be modified; if the script wishes to detach the entity and its children this must be done explicitly.

Examples

```
myDestEntity:Copy(mySourceEntity)
```

See Also

- Entity.Reset

Entity.Create

Brief

Creates a new entity.

Definition

```
Entity Entity.Create()
```

Arguments

None

Return Values

A new Entity instance.

Description

Creates a new entity.

Examples

```
entity = Entity.Create()
```

See Also

None

Entity.CreateForceAction

Brief

Creates a new 'force' action for an entity.

Definition

```
number Entity.CreateForceAction( Entity entity, boolean autoEnable = false )
```

Arguments

entity - The entity to which the action will belong.autoEnable - true to automatically enable the action, false otherwise.

Return Values

A numerical action ID.

Description

Creates a new 'force' action for an entity. If autoEnable is false, the action can be enabled with Entity.EnableAction. This action applies a force to an entity every physics update. This function is only available if the entity has a collision item.

Examples

```
entity = Entity.Create()
e:SetCollision("kCollisionResource")
e:EnableDynamics()
actionId = Entity.CreateForceAction( e, false )
f = Vector4.Create( 10.0, 0.0, 0.0 )
Entity.EnableAction( e, actionId )
Entity.SetActionForce( e, actionId, f )
```

See Also

- Entity.CreateTorqueAction
- Entity.CreatePointForceAction
- Entity.EnableAction
- Entity.SetActionForce

Entity.CreatePointForceAction

Brief

Creates a new 'point-force' action for an entity.

Definition

```
number Entity.CreatePointForceAction( Entity entity, boolean localSpace = true, boolean autoEnable =
false )
```

Arguments

entity - The entity to which the action will belonglocalSpace - true to use a local-space point, false to use a world-space point.autoEnable - true to automatically enable the action, false otherwise.

Return Values

A numerical action ID.

Description

Creates a new 'point-force' action for an entity. If autoEnable is false, the action can be enabled with Entity.EnableAction. This action applies a force at a given position to an entity. The position can a local-space point or a world-space point. This function is only available if the entity has a collision item.

Examples

```
entity = Entity.Create()
e:SetCollision("kCollisionResource")
e:EnableDynamics()
actionId = Entity.CreatePointForceAction( e, true, false )
f = Vector4.Create( 10.0, 0.0, 0.0 )
p = Vector4.Create( 0.0, 1.0, 0.0 )
Entity.EnableAction( e, actionId )
Entity.SetActionForce( e, actionId, f )
Entity.SetActionPosition( e, actionId, p )
```

See Also

- Entity.CreateTorqueAction
- Entity.CreateForceAction
- Entity.SetActionPosition

Entity.CreateTorqueAction

Brief

Creates a new 'torque' action for an entity.

Definition

```
number Entity.CreateTorqueAction( Entity entity, boolean autoEnable = false )
```

Arguments

entity - The entity to which the action will belong.autoEnable - true to automatically enable the action, false otherwise.

Return Values

A numerical action ID.

Description

Creates a new 'torque' action for an entity. If autoEnable is false, the action can be enabled with Entity.EnableAction. This action applies a torque to an entity every physics update. This function is only available if the entity has a collision item.

Examples

```
entity = Entity.Create()
e:SetCollision("kCollisionResource")
e:EnableDynamics()
actionId = Entity.CreateTorqueAction( e, false )
t = Vector4.Create( 10.0, 0.0, 0.0 )
Entity.EnableAction( e, actionId )
Entity.SetActionTorque( e, actionId, t )
```

See Also

- Entity.CreateForceAction
- Entity.CreatePointForceAction
- Entity.EnableAction
- Entity.SetActionTorque

Entity.DebugSetAxesVisible

Brief

Sets the entity to show its axis or not.

Definition

```
Entity.DebugSetAxesVisible( Entity entity, boolean value, boolean recursive, number size = 1)
```

Arguments

entity - The entity to update.value - If true the entity shows its axis and if false it hides it.recursive - If true the entity also sets the values of all the children below it.size - The size of the line drawn to represent the axis (in meters).

Return Values

None.

Description

Sets the entity to show its axis or not. Entities default to not show their axis when created. Note that this function is provided for debugging purposes and should not be used in a finalized script.

Examples

```
entity = Entity.Create()
Entity.DebugSetAxesVisible( entity, true, true, 1.0 )
```

See Also

- Entity.DebugSetHierarchyVisible
- Entity.DebugSetCollisionVisible

Entity.DebugSetCollisionVisible

Brief

Sets the entity to show its collision or not.

Definition

```
Entity.DebugSetCollisionVisible( Entity entity, boolean value, boolean recursive )
```

Arguments

entity - The entity to update.value - If true the entity shows its collision shape.recursive - If true the entity also sets the values of all the children below it.

Return Values

None.

Description

Sets the entity to show its collision or not. Entities default to not show their collision when created. Note that this function is provided for debugging purposes and should not be used in a finalized script.

Examples

```
entity = Entity.Create()
Entity.DebugSetCollisionVisible( entity, true, true )
```

See Also

- Entity.DebugSetAxesVisible
- Entity.DebugSetHierarchyVisible

Entity.DebugSetHierarchyVisible

Brief

Sets the entity to show its hierarchy or not.

Definition

```
Entity.DebugSetHierarchyVisible( Entity entity, boolean value, boolean recursive )
```

Arguments

entity - The entity to update.value - If true the entity shows its hierarchy with its immediate children and if false it hides it.recursive - If true the entity also sets the values of all the children below it.

Return Values

None.

Description

Sets the entity to show its hierarchy or not. Entities default to not show their hierarchy when created. Note that this function is provided for debugging purposes and should not be used in a finalized script.

Examples

```
entity = Entity.Create()
Entity.DebugSetHierarchyVisible( entity, true, true )
```

See Also

- Entity.DebugSetAxesVisible
- Entity.DebugSetCollisionVisible

Entity.DestroyAction

Brief

Destroy an entity action.

Definition

```
Entity.DestroyAction( Entity entity, number actionId )
```

Arguments

entity - The entity to which the action belongsactionId - The numerical action ID, obtained at action creation time

Return Values

None.

Description

Destroy an entity action. It is good practice to destroy all actions when no longer required, however the system will also automatically destroy them if the target entity is destroyed. This function is only available if the entity has a collision item.

Examples

```
entity = Entity.Create()
e:SetCollision("kCollisionResource")
e:EnableDynamics()
actionId = Entity.CreatePointForceAction( e, false )
f = Vector4.Create( 10.0, 0.0, 0.0 )
p = Vector4.Create( 0.0, 1.0, 0.0 )
Entity.EnableAction( e, actionId )
Entity.SetActionForce( e, actionId, f )
Entity.SetActionPosition( e, actionId, p )
...
Entity.DestroyAction( e, actionId )
```

See Also

- Entity.CreateForceAction
- Entity.CreateTorqueAction

Entity.DetachPersonLabel

Brief

Detaches a previously attached person label from the entity.

Definition

```
Entity.DetachPersonLabel(Entity entity, Person person)
```

Arguments

person - The person the label was taken from.

Return Values

Description

This function will detach the person label previously attached to the entity, and will re-attach it to the person it was taken from.

Examples

```
Entity.DetachPersonLabel(entity, person)
```

See Also

- Entity.AttachPersonLabel

Entity.EnableAction

Brief

Make entity action take effect.

Definition

```
Entity.EnableAction( Entity entity, number actionId, boolean enable = true )
```

Arguments

entity - The entity to which the action belongsactionId - The numerical action IDenable - true to enable the action, false otherwise

Return Values

None.

Description

Make entity action take effect. A disabled action will not be applied to the entity. This function is only available if the entity has a collision item.

Examples

```
entity = Entity.Create()
e:SetCollision("kCollisionResource")
e:EnableDynamics()
actionId = Entity.CreatePointForceAction( e, false )
f = Vector4.Create( 10.0, 0.0, 0.0 )
p = Vector4.Create( 0.0, 1.0, 0.0 )
Entity.EnableAction( e, actionId, true )
Entity.SetActionForce( e, actionId, f )
Entity.SetActionPosition( e, actionId, p )
...
Entity.DestroyAction( e, actionId )
```

See Also

- Entity.IsActionEnabled

Entity.EnableCollision

Brief

Enables/disables collision on the specified entity.

Definition

```
Entity.EnableCollision( Entity entity, boolean enable = true, Phys3dWorld world = nil )
```

Arguments

entity - The entity to update.enable - true to enable collision, false otherwise (defaults to true if not specified).world - The Phys3dWorld to add to the entity to, or nil to add to the default world.

Return Values

None.

Description

If collision is enabled for the specified entity, the local player's avatar and other physical entities can collide with it, thereby preventing penetrations. If collision is disabled, the local player's avatar and other physical entities will be able to pass through this entity. Note that the collision world specified in this call will override the collision world specified to SetCollision, if any; specifying nil here will cause the item to be removed from the world specified in SetCollision and added to the default Phys3dWorld. Limitation: Although collision can be enabled for translated and/or rotated entities, scaling is not supported.

Examples

```
-- Turn off collisions
Entity.EnableCollision( entity, false )

-- Turn on collisions
Entity.EnableCollision( entity )
```

See Also

- Entity.EnableDynamics
- Phys3dWorld.Create

Entity.EnableCollisionCallbacks

Brief

Enable or disable collision callbacks for the specified entity.

Definition

```
Entity.EnableCollisionCallbacks(Entity entity, boolean enable)
```

Arguments

entity - the entity to enable/disable collision callbacks forenable - enable or disable

Return Values

None.

Description

Enable or disable collision callbacks for the specified entity. Note that in order to receive callbacks, you must register your interest in the particular type(s) of collision event you want to receive, using Entity.RegisterCollisionCallback(). You must also set the maximum contacts using Entity.SetMaxContacts(). Finally, the entity must have collision set and enabled for the callback(s) to occur. The entity need not have dynamics enabled as static entities will also report collisions.

Examples

```
-- set up a collision data buffer
Entity.SetMaxContacts( 64 )

foo = Entity.Create()
foo:SetModel("model")
foo:SetCollision("collision")
foo:SetPosition(Vector4.Create(0, 5, 0))
foo:SetVisible(true)
foo:EnableCollision(true)
foo:EnableDynamics(true)
foo:EnableCollisionCallbacks(true)
foo:RegisterCollisionCallback(EntityCollision.ContactAdded, ContactAddedCallback, foo)
```

See Also

- Entity.RegisterCollisionCallback
- Entity.SetMaxContacts
- Entity.SetCollision
- Entity.EnableCollision

Entity.EnableDynamics

Brief

Enables/disables dynamics on the specified entity.

Definition

```
Entity.EnableDynamics( Entity entity, boolean enable = true )
```

Arguments

entity - The entity to update.enable - true to enable dynamics, false otherwise (defaults to true if not specified).

Return Values

None.

Description

If dynamics is enabled, the rigid body attached to the entity will be activated in PS Home's physics simulation. If dynamics is disabled, the entity will become fixed in its current position and will no longer be affected by the simulation. Note that collision must be enabled on the entity if dynamics is to be enabled. Attempts to set absolute translations or rotations on an entity that has dynamics enabled will be ignored. For this to succeed, dynamics must be temporarily disabled while the transformation is explicitly applied to the entity. Limitation: This function must not be called for entities that have been scaled in any way.

Examples

```
-- Turn off dynamics
Entity.EnableDynamics( entity, false )

-- Turn on dynamics (collision must be enabled as well)
Entity.EnableCollision( entity )
Entity.EnableDynamics( entity )
```

See Also

- Entity.EnableCollision

Entity.EnableDynamicsAutoDeactivation

Brief

Enable/disable auto-deactivation of dynamics on the specified entity.

Definition

```
Entity.EnableDynamicsAutoDeactivation( Entity entity, boolean enable = true )
```

Arguments

entity - The entity to update. enable - true to enable auto-deactivation, false otherwise (defaults to true if not specified).

Return Values

None.

Description

Dynamics must be enabled with Entity.EnableDynamics() for the activated/deactivated state to take effect. An Entity with activated dynamics is one that is having it's motion updated. If dynamics are deactivated, Home's physics engine will stop updating the Entity's motion, until it is activated again. Entities can be activated by another call to Entity.SetDynamicsActive or by the application of forces/torques/impulses or by coming into contact with other activated Entities. Entities can be deactivated with Entity.SetDynamicsActive or by allowing auto-deactivation which will deactivate the Entity if it has not been moving for a while. Auto-deactivation can be enabled/disabled with this function.

Examples

```
-- Disable auto-deactivation
Entity.EnableDynamicsAutoDeactivation( entity, false )

-- Enable auto-deactivation
Entity.EnableCollision( entity )
Entity.EnableDynamics( entity )
Entity.SetDynamicsActive( entity )
Entity.EnableDynamicsAutoDeactivation( entity )
```

See Also

- Entity.EnableDynamics
- Entity.SetDynamicsActive
- Entity.IsDynamicsAutoDeactivationEnabled

Entity.EnableInterpolatedMode

Brief

Enable/disable interpolated mode.

Definition

```
Entity.EnableInterpolatedMode( Entity entity, boolean enable )
```

Arguments

entity - The entity. enable - true to enable interpolated mode, false otherwise.

Return Values

None

Description

Enable/disable interpolated mode. Interpolated mode can help smooth the motion of dynamic entities, however there is an extra processing overhead.

Examples

```
Entity.EnableInterpolatedMode( entity, true )
```

See Also

- Entity.IsInterpolatedModeEnabled

Entity.EnableKeepUpright

Brief

Set whether or not to keep the entity upright.

Definition

```
Entity.EnableKeepUpright( Entity entity, boolean keepUpright = true, number softness = 0.0 )
```

Arguments

entity - entity to be affected.keepUpright - true to keep the entity upright, false otherwisesoftness - the softness of the constraint: 0 is strongest, 1 is weakest.

Return Values

None.

Description

Adds a dynamic constraint to try to keep the entity upright. The softness value is only necessary when enabling the keep upright constraint.

Examples

```
e = Entity.Create()
e:SetCollision("kCollisionResource")
e:EnableCollision()
e:EnableDynamics()
e:EnableKeepUpright( true, 0.1 )
```

See Also

- Entity.IsKeepUprightEnabled

Entity.EnableLayerCollision

Brief

Set whether or not an entity can collide with collision of the specified layer.

Definition

```
Entity.EnableLayerCollision( Entity entity, enum layer, boolean enable )
```

Arguments

entity - The entity.layer - The collision layer (a CollisionLayer enum).enable - true to enabled collision, false otherwise.

Return Values

None.

Description

Set whether or not an entity can collide with collision of the specified layer. Changes will only take effect for new collisions/overlaps. If two objects are already colliding (or very close), disabling their collisions via this function will not take effect until they separate. Also, if two objects are overlapping but do not currently collide with each other (due to disabled layer collisions), then enabling their collision via this function will not take effect until they separate.

Examples

```
Entity.EnableLayerCollision( entity1, Entity.GetCollisionLayer( entity2 ), false )
```

See Also

- Entity.IsLayerCollisionEnabled

Entity.EvaluateAnims

Brief

Evaluates all current animation(s) active on the entity, and updates the skeleton.

Definition

```
Entity.EvaluateAnims( Entity entity )
```

Arguments

entity - The entity to update.

Return Values

None.

Description

In the general case, changes made to the animation state on an entity will be reflected in the next frame. In certain situations, a script may wish to modify the animation list and have it immediately reflected in the current frame. In this event, the script can call this function, and PS Home will re-evaluate the entire animation blend list on the entity, and apply the results to the entity's skeleton. Use this function sparingly, as this evaluation is synchronous and significantly slower than the core animation update. Generally the single-frame delay in the normal case can be hidden with appropriate blending, rather than relying on this function.

Examples

```
entity = Entity.Create()
Entity.SetModel( entity, "mymodel" )
Entity.SetSkeleton( entity, "myskel" )
Entity.PlayAnim( entity, "myanim" )
...
function OnUpdate()
    if (playInstantDeathAnim) then
        local animIndex = Entity.GetIndexOfAnim( entity, "myanim" )
        local anim = Entity.GetActiveAnim( entity, animIndex )
        Entity.BlendAnimOut( entity, anim, AnimBlendType.Linear, 0.1 )
        Entity.BlendAnimIn( entity, "instantDeath", false, 0.0, AnimBlendType.Linear, 0.1, 0)
        Entity.EvaluateAnims( entity )
    end
end
```

See Also

None

Entity.FindMaterials

Brief

Get an array of materials on an entity's model.

Definition

```
Material[] Entity.FindMaterials( Entity entity, string materialId=nil)
```

Arguments

entity - The entity to search on.materialId - The material's scriptId as set in max/maya, or nil to get all materials on the entity

Return Values

An array of Materials that matches the given materialId, or all materials on the entity if no id is given.

Description

Gets an array of Materials that matches the given materialId, or all materials on the entity if no id is given.

Examples

```
entity = Entity.Create()
Entity.SetModel( entity, "mymodel" )
allMaterials = Entity.FindMaterials(entity)
print( "Found " .. #allMaterials .. " materials." );
for i, v in ipairs(array) do
    print( tostring( v ) )
end
```

See Also

- Material.GetParam
- Material.SetAlphaRef
- Material.SetBlendMode
- Material.SetMatrix
- Material.SetTexture

- Material.SetVector

Entity.GetActiveAnim

Brief

Returns a handle to a currently active animation

Definition

```
Animation Entity.GetActiveAnim( Entity entity, number index )
Entity.GetActiveAnim( Entity entity, number index, Animation anim )
```

Arguments

entity - the entity to accessindex - the index of the animation handle (within the entity's animation list) to retrieve.anim - the animation object to set, instead of returning a new one

Return Values

If animation is not specified as a parameter, either a handle to the animation, or nil if no current anim.
If an animation is specified as a parameter, there is no return value.

Description

Get a handle to the current active animation. Note that if there is no current active animation and the user has specified an animation object to set, the animation object will be marked as invalid. To check this, use Animation.IsValid.

Examples

```
local animIndex = entity:GetIndexOfAnim("myAnim")
local anim = entity:GetActiveAnim( animIndex )
```

See Also

- Animation.IsValid
- Animation.Create

Entity.GetAngularDamping

Brief

Gets the angular damping of the collision resource attached to the entity.

Definition

```
number Entity.GetAngularDamping( Entity entity )
```

Arguments

entity - The entity to query.

Return Values

The angular damping for the entity.

Description

Gets the angular damping of the collision resource attached to the entity. This function is only valid if the entity has a collision item.

Examples

```
local damping = Entity.GetAngularDamping( entity )
```

See Also

- Entity.EnableCollision
- Entity.EnableDynamics
- Entity.SetCollision
- Entity.SetAngularDamping

Entity.GetAngularVelocity

Brief

Returns the angular velocity around the center of mass of the entity.

Definition

```
Vector4 Entity.GetAngularVelocity( Entity entity )
Entity.GetAngularVelocity( Entity entity, Vector4 outVec )
```

Arguments

entity - The entity to query.outVec - The Vector4 to store the result to.

Return Values

The entity's current angular velocity.

Description

Returns the angular velocity around the center of mass of the entity's rigid body in world space. The W component of the returned vector is zero. If outVec is specified it stores the result in there instead of returning a new Vector4.

Examples

```
-- Add the entity to the physics simulation
Entity.EnableCollision( entity, true )
Entity.EnableDynamics( entity, true )

-- Query the angular velocity
angularVel = Entity.GetAngularVelocity( entity )
```

See Also

- Entity.GetLinearVelocity

Entity.GetBoneCount

Brief

Get the number of bones in the entity's skeleton.

Definition

```
number Entity.GetBoneCount(Entity entity)
```

Arguments

entity - Entity to query.

Return Values

Number of bones in the skeleton, or 0 if the entity has no skeleton attached.

Description

Retrieves the number of bones/joints in the entity's skeleton. If no skeleton is present, 0 is returned.

Examples

```
local boneCount = myEntity:GetBoneCount()
```

See Also

- Entity.GetBoneIndex

Entity.GetBoneIndex

Brief

Find the index of a bone in the entity's skeleton.

Definition

```
number Entity.GetBoneIndex(Entity entity, string boneName)
```

Arguments

entity - Entity to query.boneName - Name of bone to locate.

Return Values

The index of the bone in the skeleton (in range 1 - GetBoneCount()), or 0 if not found.

Description

Retrieves the index of the specified bone/joint in the entity's skeleton. The index is 1-based. If the bone cannot be found, or there is no skeleton attached, 0 is returned.

Examples

```
local leftHandIndex = myEntity:GetBoneIndex("lefthand")
```

See Also

- Entity.GetBoneCount
- Entity.GetBoneLocalMatrix
- Entity.GetBoneWorldMatrix

Entity.GetBoneLocalMatrix

Brief

Get the local transform matrix for a bone in the skeleton.

Definition

```
Entity.GetBoneLocalMatrix(Entity entity, string|number bone, Matrix44 result)
```

Arguments

entity - Entity to query.bone - Bone name or index.result (out) - Matrix44 value to store result into.

Return Values

None.

Description

Retrieve the local-space transform matrix for a specified bone in the skeleton. If the bone is not found an error will result. Retrieving the matrix by index will be more efficient if done more than once.

Examples

```
local mat = Matrix44.Create()
myEntity:GetBoneLocalMatrix("lefthand", mat)
-- or
local leftHandIndex = myEntity:GetBoneIndex("lefthand")
myEntity:GetBoneLocalMatrix(leftHandIndex, mat)
```

See Also

- Entity.GetBoneWorldMatrix
- Entity.GetBoneIndex

Entity.GetBoneWorldMatrix

Brief

Get the world transform matrix for a bone in the skeleton.

Definition

```
Entity.GetBoneWorldMatrix(Entity entity, string|number bone, Matrix44 result)
```

Arguments

entity - Entity to query.bone - Bone name or index.result (out) - Matrix44 value to store result into.

Return Values

None.

Description

Retrieve the world-space transform matrix for a specified bone in the skeleton. If the bone is not found an error will result. Retrieving the matrix by index will be more efficient if done more than once.

Examples

```
local mat = Matrix44.Create()
myEntity:GetBoneWorldMatrix("lefthand", mat)
-- or
local leftHandIndex = myEntity:GetBoneIndex("lefthand")
myEntity:GetBoneWorldMatrix(leftHandIndex, mat)
```

See Also

- Entity.GetBoneLocalMatrix
- Entity.GetBoneIndex

Entity.GetCenterOfMassLocal

Brief

Returns the position of the entity's center of mass in local space.

Definition

```
Vector4 Entity.GetCenterOfMassLocal( Entity entity )
Entity.GetCenterOfMassLocal( Entity entity, Vector4 outVec )
```

Arguments

entity - The entity to query.outVec - The Vector4 to store the result to.

Return Values

The position of the entity's center of mass in local space.

Description

Returns the position of the entity's center of mass in local space. This may be offset from the coordinate origin of the entity depending on its physics configuration. If outVec is specified it is used to store the result instead of returning a new Vector4. This function is only available if the entity has a collision item.

Examples

```
print( "Entity COM = " .. Entity.GetCenterOfMassLocal( entity ) )
```

See Also

- Entity.EnableDynamics
- Entity.GetCenterOfMassLocal

Entity.GetCenterOfMassWorld

Brief

Returns the position of the entity's center of mass in world space.

Definition

```
Vector4 Entity.GetCenterOfMassWorld( Entity entity )
Entity.GetCenterOfMassWorld( Entity entity, Vector4 outVec )
```

Arguments

entity - The entity to query.outVec - The Vector4 to store the result to.

Return Values

The position of the entity's center of mass in world space.

Description

Returns the position of the entity's center of mass in world space. This may differ from the position of the entity depending on its physics configuration. If outVec is specified it stores the result in there instead of returning a new Vector4.

Examples

```
print( "Entity COM = " .. Entity.GetCenterOfMassWorld( entity ) )
```

See Also

- Entity.EnableDynamics

Entity.GetChildren

Brief

Get an array of the child entities of a specified entity.

Definition

```
table Entity.GetChildren(Entity entity)
```

Arguments

entity - entity to obtain children from.

Return Values

An array of the child entities of the specified entity.

Description

If an entity has had children attached using Entity.AttachToParent, this function will return an array of all the child entities, in the order in which they were attached. If there are no children, an empty table will be returned.

Examples

```
parent = Entity.Create()
child1 = Entity.Create()
child2 = Entity.Create()
child1:AttachToParent(parent)
child2:AttachToParent(parent)
children = parent:GetChildren()
print(#children)                -- prints 2, children[1] == child1 and children[2] == child2
print(#child1:GetChildren())    -- prints 0, child1 has no children
```

See Also

- Entity.AttachToParent
- Entity.GetParent
- Entity.GetSiblings

Entity.GetCloth

Brief

Retrieve a handle to the entity's cloth meshes, if any.

Definition

```
Cloth Entity.GetCloth(Entity entity)
```

Arguments

entity - Entity to query.

Return Values

A Cloth object, or nil if the entity has no model or model contains no cloth elements.

Description

If the entity's model contains cloth elements, this function can be used to obtain a handle to the cloth elements to alter their parameters. All cloth elements in the model are modified when using the cloth handle.

Examples

```
myEntity:SetModel("ModelWithCloth")
local cloth = myEntity:GetCloth()
if (cloth) then
    cloth:SetWindAngle(135)
end
```

See Also

- Cloth.SetWindAngle

Entity.GetCollisionLayer

Brief

Query the collision layer of an entity.

Definition

```
enum Entity.GetCollisionLayer( Entity entity )
```

Arguments

entity - The entity.

Return Values

The collision layer (a CollisionLayer enum).

Description

Query the collision layer of an entity.

Examples

```
print( "collisionLayer="..Entity.GetCollisionLayer( entity )
```

See Also

- Entity.SetCollisionLayer

Entity.GetContactCount

Brief

Retrieve the number of contacts passed to a collision callback.

Definition

```
number Entity.GetContactCount()
```

Arguments

None

Return Values

Number of contacts passed to collision callback.

Description

Retrieve the number of contacts available in a collision callback. This function is only valid inside a collision callback.

Examples

```
function ContactAddedCallback(entity)
    local numContacts = Entity.GetContactCount()
    print(numContacts .. ' contacts added')
end
```

See Also

- Entity.GetContactPoint
- Entity.GetContactEntity
- Entity.GetContactRelVelocity
- Entity.GetContactId

Entity.GetContactEntity

Brief

Retrieve the other entity that has been contacted in a collision.

Definition

```
Entity Entity.GetContactEntity(number index)
```

Arguments

index - index of the contact point

Return Values

The contacted entity, or nil in a collision with the world.

Description

Retrieve the other entity contacted in a collision, that is, the entity that is not reporting the collision. In a collision with world geometry or another physics object not owned by an Entity, nil is returned. It is important to realise that the entity returned is not the exact same entity instance from Lua's perspective. It will compare equal using == or ~=, and all API functions are available, but if used as a table key it will not match the "original" entity. If you wish to match the entities using a table, it is recommended to insert them using a unique key based on the entity itself. Note that this function is only valid from within a collision callback, and will not be available for ContactRemoved callbacks.

Examples

```
function ContactAddedCallback(entity)
    local numContacts = Entity.GetContactCount()
    for i = 1, numContacts do
        local entity = Entity.GetContactEntity(i)
        if (entity == nil) then
            print("I've hit the world!\n")
        end
    end
end
```

See Also

- Entity.GetContactCount
- Entity.GetContactPoint
- Entity.GetContactRelVelocity
- Entity.GetContactId

Entity.GetContactId

Brief

Retrieve the unique contact ID.

Definition

```
number Entity.GetContactId(number index)
```

Arguments

index - index of the contact point

Return Values

The contact ID.

Description

Retrieve the unique ID number assigned the contact. This number will be preserved across multiple frames and so can be used to track individual contacts over the duration of the collision. Do not make any assumptions about the ID numbers returned. Note that this function is only valid from within a collision callback.

Examples

```
function ContactProcessCallback(entity)
    local numContacts = Entity.GetContactCount()
    for i = 1, numContacts do
        local id = Entity.GetContactId(i)
        UpdateContactTracker(id)                -- do something with contacts being tracked
    end
end
```

See Also

Entity.GetContactPoint

Brief

Retrieve the position and/or normal of the specified contact.

Definition

```
Entity.GetContactPoint(number index, Vector4 position = nil, Vector4 normal = nil)
```

Arguments

index - index of the contact pointposition - Vector4 to store the position into, or nil if unwantednormal - Vector4 to store the normal into, or nil if unwanted

Return Values

None.

Description

Retrieve the position and/or normal of a contact, specified by an index which must be between 1 and the contact count. Passing nil to position and/or normal will skip retrieval of that value. Both values will be given in world space. Note that this function is only valid from within a collision callback, and will not be available for ContactRemoved callbacks.

Examples

```
function ContactAddedCallback(entity)
    local numContacts = Entity.GetContactCount()
    local pos = Vector4.Create()
    local norm = Vector4.Create()
    for i = 1, numContacts do
        Entity.GetContactPoint(i, pos, norm)
        SpawnImpactEffect(pos, norm)            -- user function to spawn an impact particle effect
    end
end
```

See Also

- Entity.GetContactCount
- Entity.GetContactEntity
- Entity.GetContactRelVelocity
- Entity.GetContactId

Entity.GetContactRelVelocity

Brief

Retrieve the relative velocity of the colliding objects.

Definition

```
number Entity.GetContactRelVelocity(number index)
```

Arguments

index - index of the contact point

Return Values

The relative velocity, projected onto the contact normal.

Description

Retrieve the relative velocity of the colliding objects, projected onto the contact normal. Note that the value may be negative or positive. Note also that this function is only valid from within a collision callback, and only in reponse to the ContactAdded event.

Examples

```
function ContactAddedCallback(entity)
    local numContacts = Entity.GetContactCount()
    local pos = Vector4.Create()
    local norm = Vector4.Create()
    for i = 1, numContacts do
        local relVel = math.abs(Entity.GetContactRelVelocity(i))
        Entity.GetContactPoint(i, pos, norm)
        if (relVel > 10) then
            SpawnBigParticle(pos, norm)
        else
            SpawnSmallParticle(pos, norm)
        end
    end
end
```

See Also

Entity.GetContactUserId

Brief

Retrieve the user ID of the part of a party (entity or collision geometry) involved in a contact.

Definition

```
number, number Entity.GetContactUserId(number contactIndex)
number Entity.GetContactUserId(number contactIndex, number partyIndex)
```

Arguments

contactIndex - index of the contact pointpartyIndex - index of the party; 1 or 2

Return Values

In the first usage, both user IDs in order, or the single user ID requested in the second usage. If a value is nil, no user ID is available.

Description

Retrieve the user ID(s) of the part of a party (entity or collision geometry) involved in a contact. Note that this function is only valid from within a collision callback. Note that there are two collision items involved in a contact, so there are two corresponding user IDs. The user IDs are ordered such that party 1 is the entity registered for collision callbacks, and party 2 is the other collision item.

Examples

```
function ContactProcessCallback(entity)
    local numContacts = Entity.GetContactCount()
    for i = 1, numContacts do
        local idA = Entity.GetContactUserId(i, 1)
        local idB = Entity.GetContactUserId(i, 2)
        -- or local idA, idB = Entity.GetContactUserId(i)
        if idA == 100 then
            print("UserID 100 has been hit!")
        end
    end
end
```

See Also

Entity.GetCullDistance

Brief

Get the cull distance for an entity.

Definition

```
number Entity.GetCullDistance(Entity entity)
```

Arguments

entity - The entity to modify.

Return Values

The cull distance of the entity, or 0 if disabled.

Description

Get the cull distance of an entity set with SetCullDistance.

Examples

```
if (myEntity:GetCullDistance() > 500) then
    print("This entity has a very large cull distance!")
end
```

See Also

- Entity.SetCullDistance

Entity.GetGravity

Brief

Query the gravity vector of the entity.

Definition

```
Vector4 Entity.GetGravity( Entity entity )
Entity.GetGravity( Entity entity, Vector4 gravityVectorOut )
```

Arguments

entity - entity to be queried.gravityVectorOut - gravity vector storage.

Return Values

The gravity vector

Description

Query the gravity vector of the entity. This function is only valid if the entity has a collision item.

Examples

```
-- first form
print( Entity.GetGravity( entity ) )

-- second form
local   grav = Vector4.Create()
Entity.GetGravity( entity, grav )
print( grav )
```

See Also

- Entity.SetGravity

Entity.GetIndexOfAnim

Brief

Find the index of the active animation that is using the specified animation resource.

Definition

```
number Entity.GetIndexOfAnim( Entity entity, string resourceName, number startIndex = 1 )
```

Arguments

entity - the entity to queryresourceName - the name of the animation we are looking forstartIndex - index to begin search at

Return Values

The index of the animation being queried, or nil if not found.

Description

The start index is provided so that multiple searches can be made if the same resorce is being played multiple times.

Examples

```
local animIndex = Entity.GetIndexOfAnim( myEntity, "myAnim" )
```

See Also

- Entity.GetActiveAnim
- Entity.BlendAnimIn
- Entity.BlendAnimOut
- Entity.StopAnim

Entity.GetLinearDamping

Brief

Gets the linear damping of the collision resource attached to the entity.

Definition

```
number Entity.GetLinearDamping( Entity entity )
```

Arguments

entity - The entity to query.

Return Values

The linear damping for the entity.

Description

Gets the linear damping of the collision resource attached to the entity. This function is only valid if the entity has a collision item.

Examples

```
local damping = Entity.GetLinearDamping( entity )
```

See Also

- Entity.EnableCollision
- Entity.EnableDynamics
- Entity.SetCollision
- Entity.SetLinearDamping

Entity.GetLinearVelocity

Brief

Returns the linear velocity of the entity.

Definition

```
Vector4 Entity.GetLinearVelocity( Entity entity )
Entity.GetLinearVelocity( Entity entity, Vector4 outVec )
```

Arguments

entity - The entity to query.outVec - The vector to write the result to.

Return Values

The entity's current linear velocity.

Description

Returns the linear velocity of the center of mass of the entity's rigid body in world space. The W component of the returned vector is zero. If outVec is specified it stores the result in there instead of returning a new Vector4.

Examples

```
-- Add the entity to the physics simulation
Entity.EnableCollision( entity, true )
Entity.EnableDynamics( entity, true )

-- Query the linear velocity
linearVel = Entity.GetLinearVelocity( entity )
```

See Also

- Entity.GetAngularVelocity

Entity.GetLocalBounds

Brief

Retrieve the local-space bounds of an entity's model.

Definition

```
Entity.GetLocalBounds(Entity entity, Vector4 min, Vector4 max)
```

Arguments

entity - The entity to query.min (out) - Vector4 to receive the minimum extents.max (out) - Vector4 to receive the maximum extents.

Return Values

None.

Description

This function will retrieve the local-space bounds of the entity's model. If no model is set then an error will be raised. The returned vectors indicate the minimum and maximum extents on each axis in local space, and define an axis-aligned bounding box. The W components will be 1.

Examples

```
Vector4 min = Vector4.Create()
Vector4 max = Vector4.Create()
myEntity:GetLocalBounds(min, max)
local center = (min + max) * 0.5
local extents = max - center
```

See Also

- Entity.GetWorldBounds

Entity.GetLocalMatrix

Brief

Returns the current local transformation matrix of the entity.

Definition

```
Matrix44 Entity.GetLocalMatrix( Entity entity )
Entity.GetLocalMatrix( Entity entity, Matrix44 outMat )
```

Arguments

entity - The entity to query.outMat - The Matrix44 to store the result to.

Return Values

The current local matrix of the entity.

Description

Returns the current local transformation matrix of the entity. This matrix is relative to the coordinate system of the entity's parent, if one exists. If the entity has no parent, the matrix is relative to the world space coordinate system. If outMat is specified it stores the result in there instead of returning a new Matrix44.

Examples

```
entity = Entity.Create()
mat = Entity.GetLocalMatrix( entity )
print( mat )
```

See Also

- Entity.Create

Entity.GetMass

Brief

Returns the mass of the collision resource attached to the entity.

Definition

```
number Entity.GetMass( Entity entity )
```

Arguments

entity - The entity to query.

Return Values

The mass of the collision resource attached to the entity.

Description

Gets the mass of the rigid body that is attached to the entity. This function can only be called if the entity has a collision resource.

Examples

```
local mass = Entity.GetMass( entity )
```

See Also

- Entity.EnableCollision
- Entity.EnableDynamics
- Entity.SetCollision
- Entity.SetMass

Entity.GetNumActiveAnims

Brief

Returns the number of active animations currently being played on the specified entity.

Definition

```
number Entity.GetNumActiveAnims( Entity entity )
```

Arguments

entity - the entity to query

Return Values

The number of active anims running.

Description

Returns the number of active animations currently being played on the specified entity.

Examples

```
local numAnims = Entity.GetNumActiveAnims( myEntity )
```

See Also

None

Entity.GetParent

Brief

Get the parent entity for the specified entity, if any.

Definition

```
Entity Entity.GetParent(Entity entity)
```

Arguments

entity - entity to obtain parent from.

Return Values

The parent entity, or nil if no parent.

Description

If an entity has been attached to another entity using Entity.AttachToParent, the parent entity can be retrieved with this function.

Examples

```
parent = Entity.Create()
entity = Entity.Create()
entity:AttachToParent(parent)
if (entity:GetParent() == parent) then
    print("Attached!")                      -- prints Attached!
end
```

See Also

- Entity.AttachToParent
- Entity.GetChildren
- Entity.GetSiblings

Entity.GetPointVelocity

Brief

Returns the velocity of a point on a physics collision item.

Definition

```
Vector4 Entity.GetPointVelocity( Entity entity, Vector4 point )
Entity.GetPointVelocity( Entity entity, Vector4 point, Vector4 outVec )
```

Arguments

entity - The entity to query.point - The point of which you wish to know the velocity.outVec - The Vector4 to store the result to.

Return Values

The velocity of the point on the rigid body.

Description

Returns the velocity of a point on a physics collision item in world space. If outVec is specified it stores the result it is used to store the result instead of returning a new Vector4. This function is only available if the entity has a collision item.

Examples

```
print( "Point velocity 1m up = " .. Entity.GetPointVelocity( entity, Vector4.Create( 0, 1, 0 ) ) )
```

See Also

- Entity.EnableDynamics

Entity.GetPosition

Brief

Returns the entity's position.

Definition

```
Vector4 Entity.GetPosition( Entity entity )
Entity.GetPosition( Entity entity, Vector4 outVec )
```

Arguments

entity - The entity to query.outVec - The Vector4 to store the result to.

Return Values

The entity's position.

Description

Returns the entity's position. If the entity has a parent, the returned position is relative to the parent's coordinate system. If outVec is specified it stores the result in there instead of returning a new Vector4.

Examples

```
print( "Entity position = " .. Entity.GetPosition( entity ) )
```

See Also

None

Entity.GetRotationQuat

Brief

Gets the rotation as a quaternion.

Definition

```
Quaternion Entity.GetRotationQuat( Entity entity )
Entity.GetRotationQuat( Entity entity, Quaternion outQuat )
```

Arguments

entity - The entity to query.outQuat - The Quaternion to store the result to.

Return Values

The rotation of the entity in quaternion form, or none using the second syntax.

Description

Gets the rotation of the entity in quaternion form. If outQuat is specified it is used for storing the result instead of returning a new Quaternion.

Examples

```
entity = Entity.Create()
rotation = entity:GetRotationQuat()
```

See Also

- Entity.SetRotationQuat

Entity.GetRotationXyz

Brief

Gets the rotation about the x, y, and z-axis in degrees.

Definition

```
Vector4 Entity.GetRotationXyz( Entity entity )
Entity.GetRotationXyz( Entity entity, Vector4 outVec )
```

Arguments

entity - The entity to query.outVec - The Vector4 to store the result to.

Return Values

The XYZ-ordered axis rotation of the entity.

Description

Gets the rotation about the x, y, and z-axis of the entity's local coordinate system in degrees. The rotation order is x, y, z. The W component of the returned vector is set to 0.0. If outVec is specified it stores the result in there instead of returning a new Vector4.

Examples

```
entity = Entity.Create()
rotation = Entity.GetRotationXyz( entity )
```

See Also

- Entity.SetRotationXyz

Entity.GetScale

Brief

Returns the entity's scale.

Definition

```
Vector4 Entity.GetScale( Entity entity )
Entity.GetScale( Entity entity, Vector4 outVec )
```

Arguments

entity - The entity to query.outVec - The Vector4 to store the result to.

Return Values

The entity's scale.

Description

Returns the entity's scale. If the entity has a parent, the returned scale is relative to the parent's coordinate system. If outVec is specified it stores the result in there instead of returning a new Vector4.

Examples

```
print( "Entity scale = " .. Entity.GetScale( entity ) )
```

See Also

- Entity.AttachToParent
- Entity.SetScale

Entity.GetSiblings

Brief

Get an array of the sibling entities of a specified entity.

Definition

```
table Entity.GetSiblings(Entity entity)
```

Arguments

entity - entity to obtain siblings from.

Return Values

An array of the sibling entities of the specified entity.

Description

If an entity has been attached to a parent entity using Entity.AttachToParent, this function will return an array of all other entities attached to the same parent, excluding this entity. If this entity is the only child, the table will be empty.

Examples

```
parent = Entity.Create()
child1 = Entity.Create()
child2 = Entity.Create()
child1:AttachToParent(parent)
child2:AttachToParent(parent)
child1Siblings = child1:GetSiblings()
child2Siblings = child2:GetSiblings()
if (child1Sublings[1] ~= child2 or child2Siblings[1] ~= child1) then
    error("This will never happen")
end
```

See Also

- Entity.AttachToParent
- Entity.GetParent
- Entity.GetChildren

Entity.GetUniqueId

Brief

Returns a unique integer ID that identifies the entity.

Definition

```
number Entity.GetUniqueId(Entity entity)
```

Arguments

entity - The entity to query.

Return Values

A unique integer ID value.

Description

The integer ID returned is guaranteed to be unique for all currently active entities on the local client, and it is guaranteed that if entity1 == entity2 is true, then entity1:GetUniqueId() == entity2:GetUniqueId() is also true, and vice versa. No assumptions should be made about the nature of the ID value, and it is not guaranteed to be persistent or reproducible across instances or PS Home versions. Note that the ID value isn't replicated in any way, it only applies to local entities.

Examples

```
local hitEntity = raycast:GetHitEntity()
if (myEntityTable[ hitEntity:GetUniqueId() ] == nil) then
    print("New entity hit!")
    myEntityTable[ hitEntity:GetUniqueId() ] = hitEntity
end
```

See Also

None

Entity.GetWorldBounds

Brief

Retrieve the world-space bounds of an entity's model.

Definition

```
Entity.GetWorldBounds(Entity entity, Vector4 min, Vector4 max)
```

Arguments

entity - The entity to query.min (out) - Vector4 to receive the minimum extents.max (out) - Vector4 to receive the maximum extents.

Return Values

None.

Description

This function will retrieve the world-space bounds of the entity's model. If no model is set then an error will be raised. The returned vectors indicate the minimum and maximum extents on each axis in world space, and define a world-space axis-aligned bounding box. The W components will be 1. NOTE: the world bounds are only updated when the model renders. Thus if you move the object in current frame, the world bounds will be out of date until the following frame. If you wish to compute up to date world bounds in this specific case, transform the local bounds and take the AABB of the OBB thus formed.

Examples

```
Vector4 min = Vector4.Create()
Vector4 max = Vector4.Create()
myEntity:GetWorldBounds(min, max)
local center = (min + max) * 0.5
local extents = max - center
```

See Also

- Entity.GetLocalBounds

Entity.GetWorldMatrix

Brief

Returns the current world matrix of the entity.

Definition

```
Matrix44 Entity.GetWorldMatrix( Entity entity )
Entity.GetWorldMatrix( Entity entity, Matrix44 outMat )
```

Arguments

entity - The entity to query.outMat - The Matrix44 to store the result to.

Return Values

The current world matrix of the entity.

Description

Returns the current world matrix of the entity. If outMat is specified it stores the result in there instead of returning a new Matrix44.

Examples

```
entity = Entity.Create()
mat = Entity.GetWorldMatrix( entity )
print( mat )
```

See Also

- Entity.Create

Entity.HasActiveAnims

Brief

Indicates if there are any active animations on the entity.

Definition

```
boolean Entity.HasActiveAnims( Entity entity )
```

Arguments

entity - the entity to query

Return Values

True if there are any active animations being played on the entity, or false if not.

Description

This function is more efficient than checking if Entity.GetNumActiveAnims() is 0.

Examples

```
if ( Entity.HasActiveAnims( myEntity ) ) then
    Entity.StopAnim( myEntity )
end
```

See Also

- Entity.BlendAnimIn
- Entity.BlendAnimOut
- Entity.GetIndexOfAnim
- Entity.GetActiveAnim

Entity.IsActionEnabled

Brief

Query whether entity action is enabled.

Definition

```
boolean Entity.IsActionEnabled( Entity entity, number actionId )
```

Arguments

entity - The entity to which the action belongsactionId - The numerical action ID

Return Values

true if enabled, false otherwise

Description

Query whether an entity action is currently enabled.

Examples

```
entity = Entity.Create()
e:SetCollision("kCollisionResource")
e:EnableDynamics()
actionId = Entity.CreatePointForceAction( e, false )
f = Vector4.Create( 10.0, 0.0, 0.0 )
p = Vector4.Create( 0.0, 1.0, 0.0 )
Entity.EnableAction( e, actionId, true )
print( Entity.IsActionEnabled( e, actionId ) )
Entity.SetActionForce( e, actionId, f )
Entity.SetActionPosition( e, actionId, p )
...
Entity.DestroyAction( e, actionId )
```

See Also

- Entity.EnableAction

Entity.IsDynamicsActive

Brief

Query whether dynamics is active on the specified entity.

Definition

```
boolean Entity.IsDynamicsActive( Entity entity )
```

Arguments

entity - The entity to update.

Return Values

true if dynamics is active, false otherwise.

Description

Query whether dynamics is active on the specified entity. See Entity.SetDynamicsActive documentation for more information. This function is only valid if the entity has a collision item.

Examples

```
-- Activate dynamics (collision must be enabled as well)
Entity.EnableCollision( entity )
Entity.EnableDynamics( entity )
Entity.SetDynamicsActive( entity )
print( Entity.IsDynamicsActive( entity ) )
```

See Also

- Entity.EnableDynamics
- Entity.SetDynamicsActive

Entity.IsDynamicsAutoDeactivationEnabled

Brief

Query whether auto-deactivation of dynamics is enabled on the specified entity.

Definition

```
boolean Entity.IsDynamicsAutoDeactivationEnabled( Entity entity )
```

Arguments

entity - The entity to update.

Return Values

true if enabled, false otherwise.

Description

Query whether auto-deactivation of dynamics is enabled on the specified entity. See Entity.EnableDynamicsAutoDeactivation for more information.

Examples

```
-- Enable auto-deactivation
Entity.EnableCollision( entity )
Entity.EnableDynamics( entity )
Entity.SetDynamicsActive( entity )
Entity.EnableDynamicsAutoDeactivation( entity )
print( ( Entity.IsDynamicsAutoDeactivationEnabled( entity ) ) )
```

See Also

- Entity.EnableDynamics
- Entity.SetDynamicsActive
- Entity.EnableDynamicsAutoDeactivation

Entity.IsInterpolatedModeEnabled

Brief

Query whether or not interpolated mode is enabled.

Definition

```
boolean Entity.IsInterpolatedModeEnabled( Entity entity )
```

Arguments

entity - The entity.

Return Values

true if interpolated mode is enabled, false otherwise.

Description

Query whether or not interpolated mode is enabled. Interpolated mode can help smooth the motion of dynamic entities, particularly

those that are directly controlled by the user; however, there is an extra processing overhead.

Examples

```
print( Entity.IsInterpolatedModeEnabled( entity ) )
```

See Also

- Entity.EnableInterpolatedMode

Entity.IsKeepUprightEnabled

Brief

Query whether or not the entity is being kept upright.

Definition

```
boolean Entity.IsKeepUprightEnabled( Entity entity )
```

Arguments

entity - entity to be queried.

Return Values

true if the entity is being kept upright, false otherwise.

Description

Query whether or not the entity is being kept upright. e = Entity.Create() e:SetCollision("kCollisionResource") e:EnableCollision() e:EnableDynamics() e:EnableKeepUpright(true, 0.1) print(e:IsKeepUprightEnabled())

Examples

```
Entity.EnableKeepUpright
```

See Also

None

Entity.IsLayerCollisionEnabled

Brief

Query whether or not an entity can collide with collision of the specified layer.

Definition

```
boolean Entity.IsLayerCollisionEnabled( Entity entity, enum layer )
```

Arguments

entity - The entity.layer - The collision layer (a CollisionLayer enum).

Return Values

true if collision is enabled, false otherwise.

Description

Query whether or not an entity can collide with collision of the specified layer.

Examples

```
print( Entity.IsLayerCollisionEnabled( entity1, Entity.GetCollisionLayer( entity2 ) ) )
```

See Also

- Entity.EnableLayerCollision

Entity.IsVisible

Brief

Indicates if the entity is currently marked as visible.

Definition

```
Entity.IsVisible(Entity entity)
```

Arguments

entity - the entity to query.

Return Values

true if the object is marked as visible, false otherwise.

Description

An entity can be marked as invisible for several reasons: the script can do so using SetVisible, entities attached to a person that is invisible or too close to the camera, and entities that represent scene objects hidden by the client (for example, active furniture during removal). This function will indicate whether an entity has been marked invisible for any reason. It is not possible to tell from the return value why the entity is invisible, but it may be helpful to a script if it is desired to hide other entities/effects when one entity is invisible. It is not required to check this for all entities, as only entities attached to scene objects as indicated above can be affected outside the script's control. Visibility is also inherited, so children of an invisible entity are automatically made invisible also, and this functionality should be leveraged if it is desired to hide groups of entities when a parent entity is made invisible for any reason.

Examples

```
if (not Active.GetEntity():IsVisible()) then
    -- active has been hidden, pause our special unattached particle effects
    PauseParticles()
end
```

See Also

- Active.GetEntity
- Entity.SetVisible

Entity.LookAt

Brief

Rotate an entity to look at a specified point.

Definition

```
Entity.LookAt( Entity entity, Vector4 targetPoint, Vector up=[0,1,0,0] )
```

Arguments

entity - The entity to update.targetPoint - The point to look at, in the entity's parent space.up - The up vector of the entity, in the entity's parent space.

Return Values

None.

Description

Rotate the entity to look at the specified point. The entity looks down its forward/front vector, the z-axis. Note that this is a one-off reorientation of the entity, it will not track the look-at point if the entity (or its parent) moves.

Examples

```
entity = Entity.Create()
Entity.LookAt( entity, Vector4.Create( 0.5, 0.5, 0 ) )
```

See Also

- Entity.Create
- Vector4.Create
- Entity.LookAtWorld

Entity.LookAtWorld

Brief

Rotate an entity to look at a specified point in world space.

Definition

```
Entity.LookAtWorld( Entity entity, Vector4 targetPoint, Vector up=[0,1,0,0] )
```

Arguments

entity - The entity to update.targetPoint - The point to look at in world space.up - The up vector of the entity in world space.

Return Values

None.

Description

Rotate the entity to look at the specified point in world space. The entity looks down its forward/front vector, the z-axis. Note that this is a one-off reorientation of the entity, it will not track the look-at point if the entity (or its parent) moves.

Examples

```
entity = Entity.Create()
Entity.LookAtWorld( entity, Vector4.Create( 0.5, 0.5, 0 ) )
```

See Also

- Entity.Create

- Vector4.Create
- Entity.LookAt

Entity.ParticleIsAlive

Brief

Returns true if particle effect is still alive.

Definition

```
boolean Entity.ParticleIsAlive( Entity entity )
```

Arguments

entity - The entity to query.

Return Values

true if any particles are still active, false otherwise.

Description

Returns true if particle effect is still alive.

Examples

```
entity = Entity.Create()
Entity.ParticleSetEffect( entity, "myparticle" )
if Entity.ParticleIsAlive( entity ) then
    print( "Particle system still active!" )
end
```

See Also

- Entity.Create
- Entity.ParticleSetEffect

Entity.ParticleReset

Brief

Resets the particle effect.

Definition

```
Entity.ParticleReset( Entity entity )
```

Arguments

entity - The entity to update.

Return Values

None.

Description

Resets the particle effect by clearing the particle buffer and resetting time to 0.0.

Examples

```
entity = Entity.Create()
Entity.ParticleSetEffect( entity, "myparticle" )
Entity.ParticleReset( entity )
```

See Also

- Entity.Create
- Entity.ParticleSetEffect

Entity.ParticleSetAttachType

Brief

Sets how the particle is attached to the entity's world matrix.

Definition

```
Entity.ParticleSetAttachType( Entity entity, string type )
```

Arguments

entity - The entity to update.type - The type of attachment the entity has over the particle effect. Can be one of the following:
"render"
"emitter"

Return Values

None.

Description

Sets how the particle is attached to the entity's world matrix. Can be one of the following: "render" - The entity's world matrix is used as the render matrix of the particle effect. This is the default. "emitter" - The entity's world matrix is used as the emitter matrix of the particle effect.

Examples

```
e = Entity.Create()
Entity.ParticleSetEffect( e, "flaming_torch" )
Entity.ParticleSetAttachType( e, "emitter" )
```

See Also

- Entity.ParticleSetEffect

Entity.ParticleSetEffect

Brief

Sets the entity's particle effect.

Definition

```
Entity.ParticleSetEffect( Entity entity, string resourceName )
Entity.ParticleSetEffect( Entity entity, Resource particleResource )
```

Arguments

entity - The entity to update.resourceName - The resource name of the particle effect or nil to remove it.particleResource - The particle effect resource or nil to remove it.

Return Values

None.

Description

Sets the entity's particle effect to the named resource as specified in the HDK's Object Editor application.

Examples

```
entity = Entity.Create()
Entity.ParticleSetEffect( entity, "myparticle" )
```

See Also

None

Entity.ParticleSetEmitting

Brief

Starts/Stops particle emission.

Definition

```
Entity.ParticleSetEmitting( Entity entity, boolean value )
```

Arguments

entity - The entity to update.value - true to enable emission and false to stop emission.

Return Values

None.

Description

Starts/Stops particle emission.

Examples

```
entity = Entity.Create()
Entity.ParticleSetEffect( entity, "myparticle" )
Entity.ParticleSetEmitting( entity, false )
```

See Also

- Entity.Create

Entity.ParticleSetUpdate

Brief

Starts/stops particle updating.

Definition

```
Entity.ParticleSetUpdate( Entity entity, boolean value )
```

Arguments

entity - The entity to update.value - true to enable particle updates and false to disable them.

Return Values

None.

Description

Starts/stops particle updating. This effectively starts/stops time.

Examples

```
entity = Entity.Create()
Entity.ParticleSetEffect( entity, "myparticle" )
Entity.ParticleSetUpdate( entity, false )
```

See Also

- Entity.Create
- Entity.ParticleSetEffect

Entity.ParticleSetUserParam

Brief

Sets a user parameter in the particle system

Definition

```
boolean Entity.ParticleSetUserParam( Entity entity, string paramName, number paramValue )
boolean Entity.ParticleSetUserParam( Entity entity, string paramName, Vector4 paramValue )
```

Arguments

entity - The entity to update.paramName - The name of the parameter to set in the particle graph.paramValue - The value to set.

Return Values

true if the assignment was made. false if parameter not present in particle system.

Description

The particle editor tool allows for input of user parameters from runtime code. Access this via the "HomeLua..." nodes in the particle editor. There are many supported types. 1->4 float components. (Scalar, Scale, Vector, Colour, etc...) The Scalar types can use the number or Vector4 interface. All other types need to use the Vector4 interface.

Examples

```
entity = Entity.Create()
Entity.ParticleSetEffect( entity, "myparticle" )
Entity.ParticleSetUserParam( entity, "homelua_scalar1", 5.0 )
Entity.ParticleSetUserParam( entity, "homelua_scalar2", Vector4(5.0) )
Entity.ParticleSetUserParam( entity, "homelua_colour1", Vector4(0.3,0.4,0.5,1.0) )
Entity.ParticleSetUserParam( entity, "homelua_vector5", Vector4(1.0,0.0,1.0) )
```

See Also

None

Entity.PauseAnim

Brief

Pauses all current animation(s) if any are playing.

Definition

```
Entity.PauseAnim( Entity entity )
```

Arguments

entity - The entity to update.

Return Values

None.

Description

Pauses all current animation(s) if any are playing. Use ResumeAnim to continue to play the animation(s).

Examples

```
entity = Entity.Create()
Entity.SetModel( entity, "mymodel" )
Entity.SetSkeleton( entity, "myskel" )
Entity.PlayAnim( entity, "myanim" )
Entity.PauseAnim( entity )
```

See Also

- Entity.ResumeAnim

Entity.PlayAnim

Brief

Plays the specified animation resource.

Definition

```
Entity.PlayAnim( Entity entity, string resourceName, boolean looping = true, number time = 0, number
priority = 0 )
Entity.PlayAnim( Entity entity, Resource animResource, boolean looping = true, number time = 0, number
priority = 0 )
```

Arguments

entity - The entity to update.resourceName - The resource name of the animation.animResource - The animation resource.looping - flag indicating if animation should loop, or only play once. Default is true.time - the time at which the animation is played from. If the time is greater than the duration, is taken as 0. Default is 0.priority - The priority affects where in the animation list an animation will be played. The higher the number, the later it will be played.
If a prior animation is being played at the same priority, then the new animation will be placed after the old animation in the list. (default is 0).
*NB Internally, priority is treated as an integer.

Return Values

None.

Description

Plays the specified animation resource as specified in the HDK's Object Editor application. Stops all other animations that are currently playing.

Examples

```
entity = Entity.Create()
Entity.SetModel( entity, "mymodel" )
Entity.SetSkeleton( entity, "myskel" )
Entity.PlayAnim( entity, "myanim" )
```

See Also

- Entity.Create
- Entity.SetModel
- Entity.SetSkeleton

Entity.RegisterCollisionCallback

Brief

Register a function to be called in response to a collision event involving the specified entity.

Definition

```
Entity.RegisterCollisionCallback( Entity entity, EntityCollision eventType, function callback,
userData = nil )
```

Arguments

entity - the entity to register for the collision callbackeventType - an EntityCollision enum value:
EntityCollision.ContactAdded - a new contact point has been added
EntityCollision.ContactProcess - called for all active contact points each frame while they exist
EntityCollision.ContactRemoved - a contact point has been removedcallback - the Lua function to be calleduserData - arbitrary user data of the user's choice

Return Values

None.

Description

Register a function to be called in response to a collision event involving the specified entity. You must enable collision callbacks for the specified entity using EnableCollisionCallbacks() in order to begin receiving callbacks to your registered function(s). You must also set the maximum contacts buffer using Entity.SetMaxContacts(). Finally, the entity must have collision set and enabled for the callbacks to occur.

Examples

```
-- set up a collision data buffer
Entity.SetMaxContacts( 128 )

foo = Entity.Create()
foo:SetModel("model")
foo:SetCollision("collision")
foo:SetPosition(Vector4.Create(0, 5, 0))
foo:SetVisible(true)
foo:EnableCollision(true)
foo:EnableDynamics(true)
foo:EnableCollisionCallbacks(true)
foo:RegisterCollisionCallback(EntityCollision.ContactAdded, ContactAddedCallback, foo)
foo:RegisterCollisionCallback(EntityCollision.ContactRemoved, ContactRemovedCallback, foo)
```

See Also

- Entity.RegisterCollisionCallback
- Entity.SetMaxContacts
- Entity.SetCollision
- Entity.EnableCollision

Entity.Reset

Brief

Reset an entity to a newly-created state.

Definition

```
Entity.Reset(Entity entity)
```

Arguments

entity - Entity to reset.

Return Values

None.

Description

Resets all non-hierarchy entity state, including model, animation, collision, particles, attached cameras, lights and other state. The entity's hierarchy state (parent and children) and transform state (position and rotation) will not be reset, if this is desired it must be done explicitly. Of particular use if using a pool of entities, as entities can be reset upon being released back into the pool (or when allocated from the pool).

Examples

```
myEntity:Reset()
```

See Also

- Entity.Copy

Entity.ResumeAnim

Brief

Resumes all current animation(s) if they are paused.

Definition

```
Entity.ResumeAnim( Entity entity )
```

Arguments

entity - The entity to update.

Return Values

None.

Description

Resumes all current animation(s) if they are paused. Use PauseAnim to pause the animation(s).

Examples

```
entity = Entity.Create()
Entity.SetModel( entity, "mymodel" )
Entity.SetSkeleton( entity, "myskel" )
Entity.PlayAnim( entity, "myanim" )
Entity.PauseAnim( entity )
Entity.ResumeAnim( entity )
```

See Also

- Entity.PauseAnim

Entity.SetActionForce

Brief

Set the force of an entity action.

Definition

```
Entity.SetActionForce( Entity entity, number actionId, Vector4 force )
```

Arguments

entity - The entity to which the action belongsactionId - The numerical action IDforce - The force in world-space

Return Values

None.

Description

Set the force of an entity action. Only applicable to force-based actions. This function is only available if the entity has a collision item.

Examples

```
entity = Entity.Create()
e:SetCollision("kCollisionResource")
e:EnableDynamics()
actionId = Entity.CreatePointForceAction( e, false )
f = Vector4.Create( 10.0, 0.0, 0.0 )
p = Vector4.Create( 0.0, 1.0, 0.0 )
Entity.EnableAction( e, actionId )
Entity.SetActionForce( e, actionId, f )
Entity.SetActionPosition( e, actionId, p )
...
Entity.DestroyAction( e, actionId )
```

See Also

- Entity.SetActionTorque
- Entity.SetActionPosition

Entity.SetActionPosition

Brief

Set the position of an entity action.

Definition

```
Entity.SetActionPosition( Entity entity, number actionId, Vector4 position )
```

Arguments

entity - The entity to which the action belongsactionId - The numerical action IDposition - The position in world-space.

Return Values

None.

Description

Set the position of an entity action. Only applicable to position-based actions. This function is only available if the entity has a collision item.

Examples

```
entity = Entity.Create()
e:SetCollision("kCollisionResource")
e:EnableDynamics()
actionId = Entity.CreatePointForceAction( e, false )
f = Vector4.Create( 10.0, 0.0, 0.0 )
p = Vector4.Create( 0.0, 1.0, 0.0 )
Entity.EnableAction( e, actionId )
Entity.SetActionForce( e, actionId, f )
Entity.SetActionPosition( e, actionId, p )
...
Entity.DestroyAction( e, actionId )
```

See Also

- Entity.SetActionTorque
- Entity.SetActionForce

Entity.SetActionTorque

Brief

Set the torque of an entity action.

Definition

```
Entity.SetActionTorque( Entity entity, number actionId, Vector4 torque )
```

Arguments

entity - The entity to which the action belongsactionId - The numerical action IDtorque - The torque in world-space.

Return Values

None.

Description

Set the torque of an entity action. Only applicable to torque-based actions. This function is only available if the entity has a collision item.

Examples

```
entity = Entity.Create()
e:SetCollision("kCollisionResource")
e:EnableDynamics()
actionId = Entity.CreateTorqueAction( e, false )
t = Vector4.Create( 10.0, 0.0, 0.0 )
p = Vector4.Create( 0.0, 1.0, 0.0 )
Entity.EnableAction( e, actionId )
Entity.SetActionTorque( e, actionId, t )
...
Entity.DestroyAction( e, actionId )
```

See Also

- Entity.SetActionForce
- Entity.SetActionPosition

Entity.SetAngularDamping

Brief

Sets the angular damping of the collision resource attached to the entity.

Definition

```
Entity.SetAngularDamping( Entity entity, number damping )
```

Arguments

entity - The entity to update.damping - The angular damping to apply to the entity.

Return Values

None.

Description

Sets the angular damping of the collision resource attached to the entity. Angular damping reduces the rotational velocity of the entity over time. For example, if you set the angular damping to 0.2, then every second about 20% of the object's angular

velocity will be removed. This function is only valid if the entity has a collision item.

Examples

```
Entity.SetAngularDamping( entity, 0.2 )
```

See Also

- Entity.EnableCollision
- Entity.EnableDynamics
- Entity.SetCollision
- Entity.GetAngularDamping

Entity.SetAngularVelocity

Brief

Sets the angular velocity around the center of mass of the entity.

Definition

```
Entity.SetAngularVelocity( Entity entity, Vector4 velocity )
```

Arguments

entity - The entity to update.velocity - vector specifying angular velocity.

Return Values

None.

Description

Sets the angular velocity around the center of mass of the entity's rigid body in world space. The W component of the specified vector is ignored.

Examples

```
-- Add the entity to the physics simulation
Entity.EnableCollision( entity, true )
Entity.EnableDynamics( entity, true )

Entity.SetAngularVelocity( entity, Vector4.Create( 0, 10, 0 ) )
```

See Also

- Entity.SetLinearVelocity
- Vector4.Create

Entity.SetAttachType

Brief

Set the type of attachment used by the entity.

Definition

```
Entity.SetAttachType( Entity entity, AttachType type )
```

Arguments

entity - The entity to affect (must be attached to another entity or person).type - The type of attachment to use:
AttachType.Translation
AttachType.Rotation
AttachType.Scale
AttachType.Transform (default)

Return Values

None.

Description

The default behaviour for attached entities is to inherit all of their parent's transform, including translation, rotation and scaling. A script can use this function to only inherit some of the parent's transform, to only translate an object with its parent and not alter its rotation for example. Scale is handled so that if translation and scale are both inherited, then the local translation of the child entity will be scaled also, as per normal attachment. If translation is not inherited, the local translation will not be scaled.

Examples

```
child:AttachToParent(parent)
child:SetAttachType(AttachType.Translation)                          -- only inherit parent translation
child:SetAttachType(AttachType.Translation + AttachType.Rotation)    -- ignore parent scale
child:SetAttachType(AttachType.Transform)                            -- same as default
```

See Also

- Entity.AttachToParent
- Entity.AttachToBone

Entity.SetCenterOfMassLocal

Brief

Sets the position of the entity's center of mass in local space.

Definition

```
Entity.SetCenterOfMassLocal( Entity entity, Vector4 com )
```

Arguments

entity - The entity to update.com - The position of the entity's center of mass in local space.

Return Values

None.

Description

Sets the position of the entity's center of mass in local space. This may be offset from the coordinate origin of the entity depending on its physics configuration. This function is only available if the entity has a collision item.

Examples

```
Entity.SetCenterOfMassLocal( entity, Vector4.Create( 0.5, 0.5, 0.5 ) )
```

See Also

- Entity.EnableDynamics

Entity.SetCollision

Brief

Sets the entity's collision.

Definition

```
Entity.SetCollision( Entity entity, string resourceName, Phys3dWorld world = nil )
Entity.SetCollision( Entity entity, Resource collisionResource, Phys3dWorld world = nil )
```

Arguments

entity - The entity to update.resourceName - The collision's resource name or nil to remove it.collisionResource - The collision resource or nil to remove it.world - Phys3dWorld to add the entity to, or nil for the default.

Return Values

None.

Description

Sets the entity's collision to the named resource as specified in the HDK's Object Editor application. Note that an entity with collision will no longer respect attachments specified prior to calling SetCollision. The current world transform of the entity is preserved across this function call. This function will automatically enable collision on the entity.

Examples

```
entity = Entity.Create()
Entity.SetCollision( entity, "mycollision" )

-- Remove the collision from the entity
Entity.SetCollision( entity, nil )
```

See Also

- Entity.Create
- Phys3dWorld.Create

Entity.SetCollisionLayer

Brief

Set the collision layer of an entity.

Definition

```
Entity.SetCollisionLayer( Entity entity, enum layer, boolean preserveCurrentLayerCollisions = true )
```

Arguments

entity - The entity.layer - The collision layer; can be either CollisionLayer.Entity or a "User Layer".preserveCurrentLayerCollisions - true to preserve which layers the entity collides with, false otherwise.

Return Values

None.

Description

Set the collision layer of an entity. If preserveCurrentLayerCollisions is false, the layers with which the entity collides are reset to the default for the specified layer. The defaults can be changed using Scene.SetLayerCollisionDefault.

Examples

```
Entity.SetCollisionLayer( entity, CollisionLayer.User1, false )
```

See Also

- Entity.GetCollisionLayer
- Scene.SetLayerCollisionDefault

Entity.SetCullDistance

Brief

Set a distance at which the entity will be automatically made invisible.

Definition

```
Entity.SetCullDistance(Entity entity, number distance)
```

Arguments

entity - The entity to modify.distance - The distance from camera at which to visibility cull the entity, or 0 to disable.

Return Values

None.

Description

For situations involving a large scene, it may be beneficial for efficiency reasons to set objects to invisible when they are located far away. Setting a cull distance will automatically compute the entity's distance from camera every frame and hide it and any child entities if the distance is greater than the specified cull distance. Note that internally squared distances are compared, so the cull distance should not overflow a floating point number when squared.

Examples

```
myEntity:SetCullDistance(80)        -- set cull distance to 80m
```

See Also

- Entity.GetCullDistance

Entity.SetDynamicsActive

Brief

Activate/deactivate dynamics on the specified entity.

Definition

```
Entity.SetDynamicsActive( Entity entity, boolean activate = true )
```

Arguments

entity - The entity to update.activate - true to activate dynamics, false otherwise (defaults to true if not specified).

Return Values

None.

Description

Dynamics must be enabled with Entity.EnableDynamics() for the activated/deactivated state to take effect. An Entity with activated dynamics is one that is having it's motion updated. If dynamics are deactivated with this function, Home's physics engine will stop updating the Entity's motion, until it is activated again. Entities can be activated by another call to this function or by the application of forces/torques/impulses or by coming into contact with other activated Entities. Entities can be deactivated with this function or by allowing auto-deactivation which will deactivate the Entity if it has not been moving for a while. Auto-deactivation can be enabled/disabled with Entity.EnableDynamicsAutoDeactivation. This function is only valid if the entity has a collision item.

Examples

```
-- Deactivate dynamics
Entity.SetDynamicsActive( entity, false )

-- Activate dynamics (collision must be enabled as well)
Entity.EnableCollision( entity )
Entity.EnableDynamics( entity )
Entity.SetDynamicsActive( entity )
```

See Also

- Entity.EnableDynamics
- Entity.EnableDynamicsAutoDeactivation

Entity.SetFrictionCoeff

Brief

Set the friction coefficient of the collision resource attached to the entity.

Definition

```
Entity.SetFrictionCoeff( Entity entity, number f )
```

Arguments

entity - The entity to update.f - the coefficient of friction.

Return Values

None.

Description

Set the coefficient of friction for the surface of the rigid body that is attached to the entity. This affects the entity's motion relative to surfaces it comes into contact with. Although this is actually a pair-wise coefficient, friction is specified on a per rigid body basis. When affecting the relative motion of two surfaces, the surfaces' friction coefficients are combined to produce a coefficient for the interaction. Usual values for friction are between 0 and 1. Values greater than 1, although physically invalid, are allowed. However, values must be less than or equal to 255.

Examples

```
Entity.SetFrictionCoeff( entity, 0.0 )
```

See Also

- Entity.EnableCollision
- Entity.EnableDynamics
- Entity.SetCollision

Entity.SetGravity

Brief

Set the gravity vector to apply to the entity.

Definition

```
Entity.SetGravity( Entity entity, Vector4 gravityVector )
```

Arguments

entity - entity to be affected.gravityVector - new gravity vector.

Return Values

None.

Description

Only the specified entity is affected by the new gravity force.

Examples

```
-- set gravity to half strength
local g = Scene.GetGravity() * 0.5
entity:SetGravity( g )
```

See Also

- Entity.GetGravity

Entity.SetLight

Brief

Set the entity's light.

Definition

```
Entity.SetLight( Entity entity, Light light = nil )
```

Arguments

entity - The entity to update.light - The Light to attach to the entity or nil to detach the existing light.

Return Values

None.

Description

Set the entity's light.

Examples

```
entity = Entity.Create()
Entity.SetLight( entity, light )
```

See Also

- Entity.Create

Entity.SetLinearDamping

Brief

Sets the linear damping of the collision resource attached to the entity.

Definition

```
Entity.SetLinearDamping( Entity entity, number damping )
```

Arguments

entity - The entity to update.damping - The linear damping to apply to the entity.

Return Values

None.

Description

Sets the linear damping of the collision resource attached to the entity. Linear damping reduces the linear velocity of the entity over time. For example, if you set the linear damping to 0.2, then every second about 20% of the object's linear velocity will be removed. This function is only valid if the entity has a collision item.

Examples

```
Entity.SetLinearDamping( entity, 0.2 )
```

See Also

- Entity.EnableCollision
- Entity.EnableDynamics
- Entity.SetCollision
- Entity.GetLinearDamping

Entity.SetLinearVelocity

Brief

Sets the linear velocity of this object.

Definition

```
Entity.SetLinearVelocity( Entity entity, Vector4 velocity )
```

Arguments

entity - The entity to update.velocity - Vector specifying linear velocity.

Return Values

None.

Description

Sets the linear velocity of the center of mass of the entity's rigid body in world space. The W component of the specified vector is ignored.

Examples

```
-- Add the entity to the physics simulation
Entity.EnableCollision( entity, true )
Entity.EnableDynamics( entity, true )

Entity.SetLinearVelocity( entity, Vector4.Create( 0, 10, 0 ) )
```

See Also

- Entity.SetAngularVelocity
- Vector4.Create

Entity.SetLocalMatrix

Brief

Sets the entity's local transformation matrix.

Definition

```
Entity.SetLocalMatrix( Entity entity, Matrix44 matrix )
```

Arguments

entity - The entity to update.matrix - The matrix to set the entity's transform to.

Return Values

None.

Description

Sets the entity's local transformation matrix. The specified matrix is relative to the coordinate system of the entity's parent, if one exists. If the entity has no parent, the matrix is relative to the world space coordinate system.

Examples

```
entity = Entity.Create()
matrix = Matrix44.Create()

-- Set up a uniform scale matrix
Matrix44.SetScale( matrix, 2, 2, 2, 1, MATRIX_REPLACE )
Entity.SetLocalMatrix( entity, matrix )
print( mat )
```

See Also

- Entity.Create
- Matrix44.Create
- Matrix44.SetScale

Entity.SetMass

Brief

Sets the mass of the collision resource attached to the entity.

Definition

```
Entity.SetMass( Entity entity, number mass )
```

Arguments

entity - The entity to update.mass - The mass of the entity's collision resource in kilograms.

Return Values

None.

Description

Sets the mass of the collision resource that is attached to the specified entity. Very large or very small masses should be avoided, as these can create instability in the simulation. This function can only be called if the entity has a collision resource.

Examples

```
Entity.SetMass( entity, 5 )
```

See Also

- Entity.EnableCollision
- Entity.EnableDynamics
- Entity.SetCollision
- Entity.GetMass

Entity.SetMaxContacts

Brief

Specify the maximum number of contacts to buffer for collision callbacks.

Definition

```
boolean Entity.SetMaxContacts(number maxContacts)
```

Arguments

maxContacts - maximum number of contact events to buffer

Return Values

true if sufficient memory could be allocated or false otherwise.

Description

Allocates an internal buffer to record contact events for provision via collision callbacks. Each contact event generated by PS Home for the types for which the script has registered callbacks will be buffered in this internal buffer, until the contacts are passed to the script via the callbacks. All entities registered for callbacks in the current script environment will share this buffer. If more contacts are generated than the buffer can store, subsequent contacts will not be buffered and will not be

available to the script, and a message indicating this will be displayed in the console. Note that each contact currently requires 64 bytes of memory. If this function fails, there was insufficient memory available to the script to allocate the buffer. In this case, collision callbacks will be disabled until a subsequent call to this function (with a lower maximum) succeeds. This function is not available from within a collision callback and will generate an error.

Examples

```
Entity.SetMaxContacts(128)      -- enough memory for 128 contact events (added/processed/removed) per
frame
```

See Also

- Entity.RegisterCollisionCallback
- Entity.EnableCollisionCallbacks
- Entity.SetCollision
- Entity.EnableCollision

Entity.SetModel

Brief

Set the entity's model.

Definition

```
Entity.SetModel( Entity entity, string resourceName )
Entity.SetModel( Entity entity, Resource modelResource )
```

Arguments

entity - The entity to update.resourceName - The model's resource name or nil to remove the model.modelResource - The model resource or nil to remove the model.

Return Values

None.

Description

Set the entity's model to the named resource as specified in the HDK's Object Editor application.

Examples

```
entity = Entity.Create()
Entity.SetModel( entity, "mymodel" )

-- Remove the model from the entity
Entity.SetModel( entity, nil )
```

See Also

- Entity.Create

Entity.SetPosition

Brief

Sets the entity's position.

Definition

```
Entity.SetPosition( Entity entity, Vector4 position )
```

Arguments

entity - The entity to update.position - The new position for the entity.

Return Values

None.

Description

Sets the entity's position. If the entity has a parent, the specified position is relative to local coordinate system of the parent.

Examples

```
entity = Entity.Create()
Entity.SetPosition( entity, Vector4.Create( 10, 10, 0 ) )
```

See Also

- Entity.Create

Entity.SetRestitutionCoeff

Brief

Sets the restitution coefficient of the collision resource attached to the entity.

Definition

```
Entity.SetRestitutionCoeff( Entity entity, number r )
```

Arguments

entity - The entity to update.r - the coefficient of restitution.

Return Values

None.

Description

Sets the restitution (or elasticity) coefficient for the rigid body attached to the entity. This governs the effect that collisions have on the velocities of colliding entities. Like friction, this is a pair-wise coefficient - when two physical entities collide, their restitution values are combined to produce a coefficient for the interaction. Usual values for restitution are between 0 and 1. Values greater than 1, although physically invalid, are allowed. However, values must be less than 1.99.

Examples

```
-- Prevent entity from bouncing
Entity.SetRestitutionCoeff( entity, 0.0 )
```

See Also

- Entity.EnableCollision

- Entity.EnableDynamics
- Entity.SetCollision

Entity.SetRotationQuat

Brief

Sets the rotation using the supplied quaternion.

Definition

```
Entity.SetRotationQuat( Entity entity, Quaternion quat )
```

Arguments

entity - The entity to update.quat - The quaternion.

Return Values

None.

Description

Sets the rotation of the specified entity using the supplied quaternion.

Examples

```
entity = Entity.Create()
quat = Quaternion.Create()
entity:SetRotationQuat(quat)
```

See Also

- Entity.GetRotationQuat

Entity.SetRotationX

Brief

Sets the rotation about the x-axis in degrees.

Definition

```
Entity.SetRotationX( Entity entity, number rotation )
```

Arguments

entity - The entity to update.rotation - The specified angle of rotation about the x-axis in degrees.

Return Values

None.

Description

Sets the rotation about the x-axis of the entity's local coordinate system in degrees. The rotation order is x, y, z.

Examples

```
entity = Entity.Create()
Entity.SetRotationX( entity, 90.0 )
```

See Also

- Entity.Create

Entity.SetRotationXyz

Brief

Sets the rotation about the x, y, and z-axis in degrees.

Definition

```
Entity.SetRotationXyz( Entity entity, Vector4 rotation )
```

Arguments

entity - The entity to update.rotation - The specified x, y, and z angles of rotation in degrees.

Return Values

None.

Description

Sets the rotation about the x, y, and z-axis of the entity's local coordinate system in degrees. The rotation order is x, y, z.

Examples

```
entity = Entity.Create()
Entity.SetRotationXyz( entity, Vector4.Create( 45, 30, 90 ) )
```

See Also

- Entity.Create

Entity.SetRotationY

Brief

Sets the rotation about the y-axis in degrees.

Definition

```
Entity.SetRotationY( Entity entity, number rotation )
```

Arguments

entity - The entity to update.rotation - The specified angle of rotation about the y-axis in degrees.

Return Values

None.

Description

Sets the rotation about the y-axis of the entity's local coordinate system in degrees. The rotation order is x, y, z.

Examples

```
entity = Entity.Create()
Entity.SetRotationY( entity, 90.0 )
```

See Also

- Entity.Create

Entity.SetRotationZ

Brief

Sets the rotation about the z-axis in degrees.

Definition

```
Entity.SetRotationZ( Entity entity, number rotation )
```

Arguments

entity - The entity to update.rotation - The specified angle of rotation about the z-axis in degrees.

Return Values

None.

Description

Sets the rotation about the z-axis of the entity's local coordinate system in degrees. The rotation order is x, y, z.

Examples

```
entity = Entity.Create()
Entity.SetRotationZ( entity, 90.0 )
```

See Also

- Entity.Create

Entity.SetScale

Brief

Sets the entity's scale.

Definition

```
Entity.SetScale( Entity entity, Vector4 scale )
```

Arguments

entity - The entity to update.scale - The new scale of the entity.

Return Values

None.

Description

Sets the entity's scale. The specified scale is applied to the local transformation matrix of the entity. On creation, an entity has a scale of (1, 1, 1) or no scaling, in other words. Note that the W component of the specified vector is ignored. Note that scale should not be applied to entities with collision enabled.

Examples

```
entity = Entity.Create()
Entity.SetScale( entity, Vector4.Create( 45, 30, 90 ) )
```

See Also

- Entity.Create
- Vector4.Create

Entity.SetScreen

Brief

Set the entity's screen.

Definition

```
Entity.SetScreen( Entity entity, Screen screen )
```

Arguments

entity - The entity to update.screen - The screen to attach to the entity or nil to remove it.

Return Values

None.

Description

Set the entity's screen. The screen's corner coordinates will be transformed by the entity's world transformation matrix.

Examples

```
entity = Entity.Create()
Entity.SetScreen( entity, screen )

-- Remove the screen from the entity
Entity.SetScreen( entity, nil )
```

See Also

- Entity.Create

Entity.SetSkeleton

Brief

Set the entity's skeleton.

Definition

```
Entity.SetSkeleton( Entity entity, string resourceName )
Entity.SetSkeleton( Entity entity, Resource skeletonResource )
```

Arguments

entity - The entity to update. resourceName - The skeleton's resource name or nil to remove it. skeletonResource - Pre-loaded skeleton resource object.

Return Values

None.

Description

Set the entity's skeleton to the named resource as specified in the HDK's Object Editor application.

Examples

```
entity = Entity.Create()
Entity.SetSkeleton( entity, "myskel" )

-- Remove the skeleton from the entity
Entity.SetSkeleton( entity, nil )
```

See Also

- Entity.Create

Entity.SetVisible

Brief

Sets the entity to visible or invisible.

Definition

```
Entity.SetVisible( Entity entity, boolean value )
```

Arguments

entity - The entity to update. value - If true the entity is visible and if false it is invisible.

Return Values

None.

Description

Sets the entity to visible or invisible. Entities default to visible when created.

Examples

```
entity = Entity.Create()
Entity.SetVisible( entity, false )
```

See Also

- Entity.Create

Entity.StopAnim

Brief

Stops the current animation if it is playing.

Definition

```
Entity.StopAnim( Entity entity )
Entity.StopAnim( Entity entity, Animation anim )
```

Arguments

entity - The entity to update.anim - The animation to stop.

Return Values

None.

Description

If the Animation anim passed in is nil, or invalid, nothing is done. If an animation is specified, it will be stopped immediately with no blend out. If no animation is specified, all animations on the entity will be stopped immediately, with no blend out.

Examples

```
entity = Entity.Create()
local animHandle = Animation.Create()
Entity.SetModel( entity, "mymodel" )
Entity.SetSkeleton( entity, "myskel" )
Entity.PlayAnim( entity, "my1stAnim" )
Entity.BlendAnim( entity, "my2ndAnim", true, 0, 0, animHandle )
...
Entity.StopAnim( entity, animHandle ) -- stops specific animation
Entity.StopAnim( entity ) -- stops all other animations
```

See Also

- Entity.Create
- Entity.PlayAnim
- Entity.SetModel
- Entity.SetSkeleton

Environment

Environment.DebugProfileBegin

Brief

Begins profiling using the built in object profiler.

Definition

```
Environment.DebugProfileBegin( number profileIndex )
```

Arguments

profileIndex - Which user profile to use. Ranges from 1 - 4.

Return Values

None

Description

Begins profiling using the built in object profiler. Stop profiling the section with Environment.DebugProfileEnd.

Examples

```
function Update()
    Environment.DebugProfileBegin( 1 )
    RunAllMyCode()
    Environment.DebugProfileEnd( 1 )
end
```

See Also

- Environment.DebugProfileEnd

Environment.DebugProfileEnd

Brief

Ends the current profiling section.

Definition

```
Environment.DebugProfileEnd( number profileIndex )
```

Arguments

profileIndex - Which user profile to use. Ranges from 1 - 4.

Return Values

None

Description

Ends the current profiling section. This requires calling Environment.DebugProfileBegin with the matching profile index beforehand.

Examples

```
function Update()
    Environment.DebugProfileBegin( 3 )
    RunAllMyCode()
    Environment.DebugProfileEnd( 3 )
end
```

See Also

- Environment.DebugProfileBegin

Environment.EnableCallbackGc

Brief

Enable/Disable automatic garbage collection from callbacks

Definition

```
Environment.EnableCallbackGc( boolean value )
```

Arguments

value - true if you want to enable the callback GC and false to disable it

Return Values

None

Description

By default any lua environment created in Home will be subject to a garbage collection cycle. Callback Gc occurs after X non frame based callbacks, for example OnMessageReceived(), and does not include update callbacks such as OnDecideIfLocalPlayerCanJoinGame(), OnUpdate() and OnLocalPlayerUpdateGamePlay(). The garbage collection operation can take several milliseconds to complete, the cost increasing in proportion to the amount of data being handled by the script so it can be beneficial to control the garbage collection.
Note: See the HDK Lua Scripting User Guide for full details on Garbage Collection.

Examples

```
Environment.EnableCallbackGc( true )            -- enables garbage collection
Environment.EnableCallbackGc( false )           -- disables garbage collection
```

See Also

- Environment.EnableFrameGc
- Environment.SetCallbackGcInterval

Environment.EnableFrameGc

Brief

Enable/Disable per-frame automatic garbage collection

Definition

```
Environment.EnableFrameGc( boolean value)
```

Arguments

value - true if you want to enable the frame GC and false to disable it

Return Values

Description

By default any lua environment created in Home will be subject to a garbage collection cycle. This occurs after every frame. The garbage collection operation can take several milliseconds to complete, the cost increasing in proportion to the amount of data being handled by the script so it can be beneficial to control the garbage collection.
Note: See the HDK Lua Scripting User Guide for full details on Garbage Collection.

Examples

```
Environment.EnableFrameGc( true )       -- enables garbage collection
Environment.EnableFrameGc( false )      -- disables garbage collection
```

See Also

- Environment.EnableCallbackGc

Environment.SetCallbackGcInterval

Brief

Sets the intervals that the callback garbage collector is initiated

Definition

```
Environment.SetCallbackGcInterval(n)
```

Arguments

n - the number of times a callback hook is called before a garbage collection is initiated.

Return Values

None

Description

This interval applies to any non-update related callback hooks (such as network bag or message related callbacks). It is possible for several such callbacks to be made inside one frame which can cause lua script to run out of memory, for example 50 callbacks of OnMessageReceived could happen in a single frame. These callbacks are counted and if they reach the specified interval PS Home will force a garbage collect. If any frame garbage collection occurs it will reset the count so the system will not unnecessarily run more than one garbage collection cycle per frame.
By default the interval has a value of 10.
Note: See the HDK Lua Scripting User Guide for full details on Garbage Collection.

Examples

```
Environment.EnableCallbackGc( true )        -- enable gc
Environment.SetCallbackGcStep( 0 )          -- full garbage collection
Environment.SetCallbackGcInterval( 2 )      -- after two callbacks the system will do a garbage
collect
```

See Also

- Environment.EnableCallbackGc

Environment.SetCallbackGcStep

Brief

Sets the heuristic value used for doing a step of garbage collection per callback update

Definition

```
Environment.SetCallbackGcStep( number step )
```

Arguments

step - 0 for full garbage collection or > 0 for smaller steps.

Return Values

None

Description

The step controls how much the callback garbage collector collects per call (when enabled) and has hit the number of required

invervals. The default is for the garbage collector to do a full collect, but by using any number greater than 0 a user can instruct the garbage collector to do smaller steps of work. Larger values equate to more steps during garbage collection. A step value greater than 0 is equivalent to the lua function collectgarbage("step", step). A step value of 0 is equivalent to the lua function collectgarbage("collect").
Note: See the HDK Lua Scripting User Guide for full details on Garbage Collection.

Examples

```
Environment.EnableCallbackGc( true )        -- enable gc
Environment.SetCallbackGcStep( 2 )          -- do a small amount of gc per update
```

See Also

- Environment.EnableCallbackGc
- Environment.SetCallbackGcInterval

Environment.SetFrameGcStep

Brief

Sets the heuristic value used for doing a step of garbage collection per frame

Definition

```
Environment.SetFrameGcStep( number step )
```

Arguments

step - 0 for full garbage collection or > 0 for smaller steps.

Return Values

None

Description

The step controls how much the frame update garbage collector collects per call (when enabled). The default is for the garbage collector to do a full collect, but by using any number greater than 0 a user can instruct the garbage collector to do smaller steps of work. Larger values equate to more steps during garbage collection. A step value greater than 0 is equivalent to the lua function collectgarbage("step", step). A step value of 0 is equivalent to the lua function collectgarbage("collect").
Note: See the HDK Lua Scripting User Guide for full details on Garbage Collection.

Examples

```
Environment.EnableFrameGc( true )           -- enable gc
Environment.SetFrameGcStep( 2 )             -- do a small amount of gc per update
```

See Also

- Environment.EnableFrameGc

Font

Font.Create

Brief

Create a font object given a font resource and parameters.

Definition

```
Font Font.Create( Resource fontResource, float fontSize, float fontSlant = 0 )
Font Font.Create( string resourceName, float fontSize, float fontSlant = 0 )
```

Arguments

fontResource - The font file resource.resourceName - The name of a font file resource in the object.fontSize - The size for the font (value must be positive).fontSlant - The slant for the font in range -1 to 1 (optional parameter, default is 0).

Return Values

The new font instance.

Description

A font resource is a file resource that references a TTF font file. This function will create a font object from the resource, which must be already loaded, that can be passed to Renderer.SetFont to specify a font used for text rendering. Note that both the font object and its associated resource must continue to exist while the font is in use. To allow the font and resource to be garbage collected, any renderers referencing the font should be changed to another custom font or back to a system font.

Examples

```
font = Font.Create(myFontResource, 8)
myRenderer:SetFont(font)
myRenderer:DrawText2d(textPos, "Look at my custom font!")
```

See Also

- Renderer.SetFont

Furniture

Furniture.ClearLayout

Brief

Remove all furniture from the current personal space.

Definition

```
Furniture.ClearLayout()
```

Arguments

None

Return Values

Description

Remove all furniture from the current personal space. This function should only be called if the furniture system is not busy. This function should only be called if the current scene is one in which the local player is allowed to use the furniture placement system.

Examples

```
if not Furniture.IsSystemBusy() then
    Furniture.ClearLayout()
end
```

See Also

- Furniture.IsSystemBusy

Furniture.ExportLayout

Brief

Export the furniture layout of the current scene to a memory container.

Definition

```
MemoryContainer Furniture.ExportLayout()
```

Arguments

None

Return Values

Description

Export the furniture layout of the current scene to a memory container. This function should only be called if the furniture system is not busy. This function should only be called if the current scene is one in which the local player is allowed to use the furniture placement system.

Examples

```
if not Furniture.IsSystemBusy() then
     mc = Furniture.ExportLayout()
end
```

See Also

- Furniture.IsSystemBusy
- Furniture.ImportLayout

Furniture.GetCategories

Brief

Get information about furniture categories.

Definition

```
table Furniture.GetCategories()
```

Arguments

None

Return Values

A table containing information about furniture categories.

Description

Get information about furniture categories. The table will contain an entry for each furniture category, indexed from 1 to the number of categories. Each entry will be a table with the following members: 'id' - the furniture system unique id for the category. 'name' - the localised name for the category. 'thumb' - a handle to the thumbnail for the category.

Examples

```
local function GetCategoryData()
    local   data = {}
    local   categories = Furniture.GetCategories()
    local   numCategories = #categories
    for i = 1, numCategories do
        data[ categories[i].id ] = {}
        data[ categories[i].id ].name = categories[i].name
        data[ categories[i].id ].thumb = categories[i].thumb
        data[ categories[i].id ].objects = {}
    end
    local   numItems = Furniture.GetItemCount()
    for i = 1, numItems do
        local   itemData = Furniture.GetItemData( i )
        data[ itemData.categoryId ].objects[ itemData.objectId ] = true
    end
    return data
end
```

See Also

- Furniture.GetItemCount
- Furniture.GetItemData

Furniture.GetItemCount

Brief

Get number of furniture items in the current scene.

Definition

```
number Furniture.GetItemCount()
```

Arguments

None

Return Values

The number of furniture items in the current scene.

Description

Get number of furniture items in the current scene. This function should only be called if the furniture system is not busy.

Examples

```
if Furniture.IsSystemBusy() == false then
    local   numItems = Furniture.GetItemCount()
    for i = 1, numItems do
        local   itemData = Furniture.GetItemData( i )
        print( itemData )
    end
end
```

See Also

- Furniture.GetItemData
- Furniture.IsSystemBusy

Furniture.GetItemData

Brief

Get information about an item of furniture in the current scene.

Definition

```
table Furniture.GetItemData( number index )
```

Arguments

index - the index of the furniture item (1 <= index <= numItems)

Return Values

A table containing information about the furniture item.

Description

Get information about an item of furniture in the current scene. The table will contain the following information: 'id' - the furniture system unique id for the item. 'categoryId' - a FurnitureCategory enum. 'objectId' - the object id for the item. 'pos' - the position of the item as a Vector4. 'rot' - the rotation of the item as a Quaternion. This function should only be called if the furniture system is not busy.

Examples

```
if Furniture.IsSystemBusy() == false then
    local   numItems = Furniture.GetItemCount()
    for i = 1, numItems do
        local   itemData = Furniture.GetItemData( i )
        print( itemData )
    end
end
```

See Also

- Furniture.GetItemCount
- Furniture.IsSystemBusy

Furniture.ImportLayout

Brief

Import a furniture layout from a memory container into the current scene.

Definition

```
Furniture.ImportLayout( MemoryContainer container )
```

Arguments

None

Return Values

Description

Import a furniture layout from a memory container into the current scene. This function should only be called if the furniture system is not busy. This function should only be called if the current scene is one in which the local player is allowed to use

the furniture placement system.

Examples

```
if not Furniture.IsSystemBusy() then
    Furniture.ImportLayout( mc )
end
```

See Also

- Furniture.IsSystemBusy
- Furniture.ExportLayout

Furniture.IsSystemBusy

Brief

Query whether or not the furniture system is busy.

Definition

```
boolean Furniture.IsSystemBusy()
```

Arguments

None

Return Values

true if the furniture system is busy, false otherwise.

Description

Query whether or not the furniture system is busy.

Examples

```
if Furniture.IsSystemBusy() == false then
    Furniture.ClearLayout()
end
```

See Also

- Furniture.ClearLayout
- Furniture.LoadLayout
- Furniture.SaveLayout

Furniture.LoadLayout

Brief

Load the user's saved furniture layout into the current scene.

Definition

```
Furniture.LoadLayout()
```

Arguments

None

Return Values

Description

Load the user's saved furniture layout into the current scene. If the call was successful, Furniture.IsSystemBusy should now return true. This function should only be called if the furniture system is not busy. This function should not be called from the furniture callback. This function should only be called if the current scene is one in which the local player is allowed to use the furniture placement system.

Examples

```
if not Furniture.IsSystemBusy() then
    Furniture.LoadLayout()
    if not Furniture.IsSystemBusy() then
        print( "Failed" )
    end
end
```

See Also

- Furniture.IsSystemBusy
- Furniture.SaveLayout

Furniture.SaveLayout

Brief

Save the furniture layout of the current scene to the user's save-data.

Definition

```
Furniture.SaveLayout()
```

Arguments

None

Return Values

Description

Save the furniture layout of the current scene to the user's save-data. If the call was successful, Furniture.IsSystemBusy should now return true. This function should only be called if the furniture system is not busy. This function should not be called from the furniture callback. This function should only be called if the current scene is one in which the local player is allowed to use the furniture placement system.

Examples

```
if not Furniture.IsSystemBusy() then
    Furniture.SaveLayout()
    if not Furniture.IsSystemBusy() then
        print( "Failed" )
    end
end
```

See Also

- Furniture.IsSystemBusy

- Furniture.LoadLayout

Furniture.SetCallback

Brief

Set the callback to receive FurnitureEvents for this script.

Definition

```
Furniture.SetCallback( function callback )
```

Arguments

callback - the callback function object

Return Values

Description

Set the callback to receive FurnitureEvents for this script. Only one callback per script.

Examples

```
function FurnitureCallback( eventId, eventData )
    if eventId == FurnitureEvent.SaveSuccess then
        mc = Furniture.SaveLayout()
    end
end
Furniture.SetCallback( FurnitureCallback )
```

See Also

- Furniture.IsSystemBusy
- Furniture.LoadLayout

GameInfo

GameInfo.GetSearchResults

Brief

Obtains the results of the search.

Definition

```
table GameInfo.GetSearchResults()
```

Arguments

None

Return Values

A table containing information on the game searched for, nil if the search was unsuccessful.

Accessible fields within the table:
title - Localized title name.
parentalLevel - Game parental level.

supportsGameLaunching - Indicates whether this game supports Game Launching explicitly.

Description

Obtains the results of the search.

Examples

```
function Update()

    if state == STATE_TITLE_SEARCH then

        if GameInfo.IsSearchFinished() == true then

            local gameInfo = GameInfo.GetSearchResults()

            if gameInfo ~= nil then

                print( string.format("Search Results:\n============\ntitle: %s\nparentalLevel:
%d\nsupportsGameLaunching: %s\n\n",
                        gameInfo.title, gameInfo.parentalLevel,
tostring(gameInfo.supportsGameLaunching) ) )

            else

                print( "GameInfo: title  not found.\n" )

            end

        end

    end

end
```

See Also

GameInfo.IsGamesListBusy

Brief

Checks to see if the games list is busy.

Definition

```
boolean GameInfo.IsGamesListBusy()
```

Arguments

None

Return Values

true if games list is busy and false otherwise.

Description

Checks to see if the games list is busy. Games list can become busy, for example, when a disc is inserted/ejected from the BD drive.

Examples

```
function Update()

    if GameInfo.IsGamesListBusy() == false then

        GameInfo.SearchByTitleId("GLMS00001")
        state = STATE_TITLE_SEARCH

    end

end
```

See Also

None

GameInfo.IsSearchFinished

Brief

Checks to see if the current search query has finished.

Definition

```
boolean GameInfo.IsSearchFinished()
```

Arguments

None

Return Values

true if search has finished, false otherwise.

Description

Checks to see if the current search query has finished.

Examples

```
function Update()

    if state == STATE_TITLE_SEARCH then

        if GameInfo.IsSearchFinished() == true then

            local gameInfo = GameInfo.GetSearchResults()

            if gameInfo ~= nil then

                print( string.format("Search Results:\n============\ntitle: %s\nparentalLevel:
%d\nsupportsGameLaunching: %s\n\n",
                        gameInfo.title, gameInfo.parentalLevel,
tostring(gameInfo.supportsGameLaunching) ) )

            else

                print( "GameInfo: title  not found.\n" )

            end

        end

    end

end
```

See Also

GameInfo.SearchByTitleId

Brief

Search for a game by title ID.

Definition

```
GameInfo.SearchByTitleId()
```

Arguments

None

Return Values

none.

Description

Search for a game by title ID. Both games installed on HDD and also currently in the BD drive will be checked.

Examples

```
function Update()

    if GameInfo.IsGamesListBusy() == false then

        GameInfo.SearchByTitleId("GLMS00001")
        state = STATE_TITLE_SEARCH

    end

end
```

See Also

None

GameLaunch

GameLaunch.CreateSession

Brief

Creates a Game Launching session and advertises it to other players in the scene.

Definition

```
GameLaunch.CreateSession()
```

Arguments

None

Return Values

None

Description

Creates a Game Launching session and advertises it to other players in the scene. Note: This should be called by the session master only.

Examples

```

```

See Also

None

GameLaunch.GetCurrentFocusPersonId

Brief

Get the personId of the person in context.

Definition

```
GameLaunch.GetCurrentFocusPersonId()
```

Arguments

None

Return Values

person ID

Description

Get the personId of the person in context.

Examples

```
function OnAddPlayer()
    local personId = GameLaunch.GetCurrentFocusPersonId();
    print( string.format("New player has joined the session. PersonId = %s", personId) );
end
```

See Also

GameLaunch.GetMaxPlayers

Brief

Retrieves the maximum number of players for this session.

Definition

```
number GameLaunch.GetMaxPlayers()
```

Arguments

None

Return Values

Maximum number of players (including session master), nil if session is invalid/not yet available.

Description

Retrieves the maximum number of players for this session. For Lua scripted GameLaunch objects, this value is set using GameLaunch.SetMaxPlayers(), For objects which use the XML menu format (specified using menu_options_xml of the game_launch object component), this value will be the user selected value from the osd item created using the <MAX_PLAYERS> tag. For the session master this value will become available once the session has been created using GameLaunch.CreateSession() For all other members, this value will become available only after successfully joining a session, indicated by the on_join_game callback.

Examples

```
function MaxPlayersOsdItemChangeCallback()
    local maxPlayers = tonumber(maxPlayersList:GetFocusItem())  -- read value from osd item
    GameLaunch.SetMaxPlayers( maxPlayers )
end
```

See Also

- GameLaunch.SetMaxPlayers

GameLaunch.GetNumberOfPlayers

Brief

Retrieves the number of players currently participating in the session.

Definition

```
number GameLaunch.GetNumberOfPlayers()
```

Arguments

None

Return Values

Number of players (including session master), nil if session is invalid/not yet available.

Description

Retrieves the number of players currently participating in the session. For the session master this value will become available once the session has been created, for all other members, this value will become available only after successfully joining a session.

Examples

```
```

See Also

None

GameLaunch.GetSessionMasterPersonId

Brief

Queries the Session Master person ID for the current session that the local player is part of.

Definition

```
GameLaunch.GetSessionMasterPersonId()
```

Arguments

None

Return Values

Session Master person ID.

Description

Queries the Session Master person ID for the current session that the local player is part of.

Examples

```
local isSessionMaster = LocalPlayer.GetMyId() == GameLaunch.GetSessionMasterPersonId();
```

See Also

None

GameLaunch.GetTitle

Brief

Retrieves information on the title used in the current Game Launch session.

Definition

```
table GameLaunch.GetTitle()
```

Arguments

None

Return Values

A table containing title information, nil if information not available.

Accessible fields within the table:
titleId - Title ID of the title.
title - Localized title name.

Description

Retrieves information on the title associated with the Game Launch session. For the Session Master it is the title chosen when creating a session. For other members it is the title that has been selected as compatible with the joining session.

Examples

```
funcation OnLocalPlayerJoinGame()
    local titleInfo = GameLaunch.GetTitle()

    if titleInfo ~= nil then
        print( string.format("Session details:\n title: %s\ntitleId: %s\n\n", titleInfo.title,
titleInfo.titleId) )

    end
end
```

See Also

None

GameLaunch.GetUserComment

Brief

Retrieves the user comment associated with the current Game Launching session.

Definition

```
string GameLaunch.GetUserComment()
```

Arguments

Return Values

The user comment set for the session (if any). nil if the session is not available.

Description

User comments are only valid once the user is part of the session.

Examples

```

```

See Also

None

GameLaunch.InitiateLaunch

Brief

Initiate the Game Launch sequence.

Definition

```
GameLaunch.InitiateLaunch()
```

Arguments

None

Return Values

None

Description

When players are ready, call this function to begin the launch sequence which includes exporting of the session options and actually launching into the title.

Examples

See Also

None

GameLaunch.LeaveSession

Brief

Submits a request to leave an existing Game Launching session.

Definition

```
GameLaunch.LeaveSession()
```

Arguments

None

Return Values

None

Description

Submits a request to leave an existing Game Launching session. This can be called by any player in a session including the session master. If a session master leaves the session will be cancelled.

Examples

See Also

- GameLaunch.CreateSession

GameLaunch.OnHideUI

Brief

Sends notification to the Game Launching system that the UI has been hidden.

Definition

```
GameLaunch.OnHideUI()
```

Arguments

None

Return Values

None

Description

Sends notification to the Game Launching system that the UI has been hidden. The session will remain active and any pad buttons reserved will be deactivated.

Examples

```
```

See Also

None

GameLaunch.OnShowUI

Brief

Sends notification to the Game Launching system that object intends to activate its UI.

Definition

```
GameLaunch.OnShowUI()
```

Arguments

None

Return Values

None

Description

Sends notification to the Game Launching system that object intends to activate its UI. Any pad buttons reserved will become activate. If the standard Game Launching Osd is used, that will be also become active. Use this function for situations when the script needs to display the object's UI and is not in response to the show_ui callback function invoked by the system. For example, the system invokes the on_launch_request callback which results in a denied launch, it maybe desirable to display a fail message using the object's UI instead of the standard pop-up Osd.

Examples

```
```

See Also

None

GameLaunch.ReturnJoinGameResult

Brief

Returns the result for the custom join game step of the joining process.

Definition

```
GameLaunch.ReturnJoinGameResult( boolean result, string localisedString, boolean showPopup )
```

Arguments

result - true = successful, false = failed. localisedString - If not nil then this string will be displayed in a popup and will replacethe standard join fail/successful message (optional argument)disablePopup - If 'localisedString' == nil then setting this to true will prevent the standard popupbeing shown. It will be the responsibility of the developer to display an appopriatemessage. (Optional argument. Default: false)

Return Values

None

Description

When a callback function in on_join_game is specified, the Game Launch system will wait for this call before proceeding. Passing 'result' as false will result in the object being immediately destroyed and all resources freed, therefore, any messages to be manually displayed should be done prior to this call.

Examples

```
localisedFailMessage = me:GetLocalisedText("fail_noEntitlement")
    GameLaunch.ReturnJoinGameResult(true, localisedFailMessage)
```

See Also

GameLaunch.ReturnLaunchRequestResult

Brief

Returns the result for the custom validation step of the launch process.

Definition

```
GameLaunch.ReturnLaunchRequestResult( boolean result, string localisedString, boolean showPopup )
```

Arguments

result - true = successful, false = failed. localisedString - If not nil then this string will be displayed in a standard Game Launching systempopup (optional argument)disablePopup - If 'localisedString' == nil then setting this to true will prevent the standard popupbeing shown. It will be the responsibility of the developer to display an appopriatemessage. (Optional argument. Default: false)

Return Values

None

Description

When a callback function in on_launch_request is specified, the Game Launch system will wait for this call before proceeding.

Passing 'result' as false will result in the object being immediately destroyed and all resources freed, therefore, any messages to be manually displayed should be done prior to this call.

Examples

```
localisedFailMessage = me:GetLocalisedText("fail_notEnoughPlayers")
    GameLaunch.ReturnInitiateLaunchResult(true, localisedFailMessage)
```

See Also

GameLaunch.SetMaxPlayers

Brief

Sets the maximum number of players for the current game launching session.

Definition

```
GameLaunch.SetMaxPlayers( maxPlayers )
```

Arguments

maxPlayers - maximum number of players that can participate in a session. This value is inclusive of the session master.

Return Values

None

Description

Sets the maximum number of players for the current game launching session. Note: This should be called by the session master only and prior to GameLaunch.CreateSession. After a session is created, subsequent calls using this function will have no effect. Note: This should be called by the session master only.

Examples

```
GameLaunch.SetMaxPlayers(12)
```

See Also

- GameLaunch.SetNumPrivateSlots

GameLaunch.SetNumPrivateSlots

Brief

Marks a set number of slots as private for the current game launching session.

Definition

```
GameLaunch.SetNumPrivateSlots( numPrivateSlots )
```

Arguments

numPrivateSlots - number of slots to be marked private. Does not include the session master.

Return Values

None

Description

Marks a set number of slots as private for the current game launching session. Note: This should be called by the session master only and must be less than the maximum number of players. The call must be made prior to GameLaunch.CreateSession. After a session is created, subsequent calls using this function will have no effect.

Examples

```
GameLaunch.SetNumPrivateSlots(6)
```

See Also

- GameLaunch.SetMaxPlayers

GameLaunch.SetUserComment

Brief

Allows the user to attach a short comment string to the Game Launching session.

Definition

```
GameLaunch.SetUserComment( string userComment )
```

Arguments

userComment - comment string

Return Values

None

Description

The comment string will be visible to other players and is designed to help them decide which session to join. Maximum comment length is 64 bytes.

Examples

```
GameLaunch.SetUserComment("Noobs only")
```

See Also

None

GameLaunchButton

GameLaunchButton.GetNameRef

Brief

Obtains a button's reference name.

Definition

```
string GameLaunchButton.GetNameRef(GameLaunchButton buttonItem)
```

Arguments

Return Values

Returns the name reference string of the button item passed in.

Description

For buttons which contain localized text, the reference string is returned. For buttons non localized text, the non-localized string is returned. GetNameRef() can be useful for debug purposes and also in cases where a developer wishes to manually write out a custom session XML file upon launching. Note: Contents of GameLaunchButton types are not exported to an GAMELAUNCHSESSION.XML

Examples

```
local item = myMenu1:GetFirstItem()

    while item ~= nil do

        if Type(item) == "GameLaunchButton" then
            local buttonName = item:GetNameRef()

            if buttonName == "myButtonNameRef" then

                GameLaunchButton.SetText( item, "My Localized String", false )

            end
        end

        item = myMenu1:GetNextItem(item)
    end
```

See Also

GameLaunchButton.SetCallbackOnPressed

Brief

Specifies a callback function to executed when the button is pressed.

Definition

```
GameLaunchButton.SetCallbackOnPressed(GameLaunchButton button, string callbackFunction)
```

Arguments

Return Values

Description

Specifies a callback function to executed when the button is pressed.

Examples

```
myButton = mainMenu:AddItemButton( "REF_MyUserButton" );
GameLaunchButton.SetCallbackOnPressed( myButton, "MyButtonCB()" )

function MyButtonCB()

    GameLaunchButton( myButton, "REF_buttonPressed", true )

end
```

See Also

GameLaunchButton.SetText

Brief

Sets the text displayed in the button.

Definition

```
GameLaunchButton.SetText(GameLaunchButton button, string buttonText, boolean needsLocalization)
```

Arguments

button - This button object.buttonText - The text which should be displayed in the button.needsLocalization - Specifies whether 'buttonText' should be localized before it is displayed. This is
an optional parameter and defaults to true if unspecified.

Return Values

Description

Sets the text displayed in the button. If the parameter 'needsLocalization' is true, this button's refernce string will become the value of 'buttonText' and will be the string returned by GameLaunchButton.GetNameRef()

Examples

```
myButton = mainMenu:AddItemButton( "REF_MyUserButton" );
GameLaunchButton.SetCallbackOnPressed( myButton, "MyButtonCB()" )

function MyButtonCB()

    GameLaunchButton( myButton, "REF_buttonPressed", true )

end
```

See Also

GameLaunchExporter

GameLaunchExporter.BeginSection

Brief

Begins a new section with the name specified.

Definition

```
GameLaunchExporter.BeginSection()
```

Arguments

Section name

Return Values

None

Description

Begins a new section with the name specified. Must be called after BeginSessionExport and EndSessionExport, and must be accompanied by an EndSection.

Examples

```
GameLaunchExporter.BeginSessionExport()
    GameLaunchExporter.BeginSection("sessionOptions")
        GameLaunchExporter.ExportOption("Track", "TrackA")
    GameLaunchExporter.EndSection()
GameLaunchExporter.EndSessionExport()
```

See Also

- GameLaunchExporter.EndSessionExport
- GameLaunchExporter.EndSection

GameLaunchExporter.BeginSessionExport

Brief

Begins the export of the GAMELAUNCHSESSION.XML file

Definition

```
GameLaunchExporter.BeginSessionExport()
```

Arguments

None

Return Values

None

Description

Begins the export of the GAMELAUNCHSESSION.XML file. Call GameLaunchExporter.EndSessionExport() to finish exporting.

Examples

```
GameLaunchExporter.BeginSessionExport()
    -- your Sections here ...
GameLaunchExporter.EndSessionExport()
```

See Also

- GameLaunchExporter.EndSessionExport

GameLaunchExporter.EndSection

Brief

Ends a section that was began with BeginSection

Definition

```
GameLaunchExporter.EndSection()
```

Arguments

None

Return Values

None

Description

Ends a section that was began with BeginSection

Examples

```
GameLaunchExporter.BeginSessionExport()
    GameLaunchExporter.BeginSection("sessionOptions")
        GameLaunchExporter.ExportOption("Track", "TrackA")
    GameLaunchExporter.EndSection()
GameLaunchExporter.EndSessionExport()
```

See Also

- GameLaunchExporter.BeginSessionExport
- GameLaunchExporter.BeginSection

GameLaunchExporter.EndSessionExport

Brief

Finalizes and writes out the GAMELAUNCHSESSION.XML file

Definition

```
GameLaunchExporter.EndSessionExport()
```

Arguments

None

Return Values

None

Description

Finalizes and writes out the GAMELAUNCHSESSION.XML file

Examples

```
GameLaunchExporter.BeginSessionExport()
    -- your Sections here ...
GameLaunchExporter.EndSessionExport()
```

See Also

- GameLaunchExporter.BeginSessionExport

GameLaunchExporter.ExportOption

Brief

Exports nodes item and value based on two passed string values.

Definition

```
GameLaunchExporter.ExportOption()
```

Arguments

String item, String value

Return Values

None

Description

Exports nodes item and value based on two passed string values.

Examples

```
GameLaunchExporter.BeginSection("sessionOptions")
    GameLaunchExporter.ExportOption("Track", "TrackA")
    GameLaunchExporter.ExportOption("Ambiance", "1")
GameLaunchExporter.EndSection()
```

See Also

- GameLaunchExporter.BeginSessionExport

GameLaunchListItem

GameLaunchListItem.AddItem

Brief

Adds a item to a GameLaunchListItem menu item.

Definition

```
string GameLaunchListItem.AddItem(GameLaunchListItem listItem, string itemNameRef)
```

Arguments

listItem - The list item object to which the item is to be added.
itemNameRef - The localisation reference name of the item to be added.

Return Values

Description

Items added via AddItem() must have an entry in the object's localisation database. If raw values are desired which are not subject to any localisation then AddItemRaw() can be used.

Examples

```
trackList = myMenu1:AddItemList("track")
trackList:AddItem("trackA")
trackList:AddItem("trackB")
trackList:AddItem("trackC")
```

See Also

None

GameLaunchListItem.AddItemRaw

Brief

Adds a item to a GameLaunchListItem menu item.

Definition

```
string GameLaunchListItem.AddItemRaw(GameLaunchListItem listItem, string itemRawText)
```

Arguments

listItem - The list item object to which the item is to be added.
itemRawText - The actual text string for the item to be added.

Return Values

Description

When using AddItemRaw() the string specified by itemRawText will be directly displayed without localisation. If localised strings are required AddItem() should be used.

Examples

```
maxPlayersList = myMenu1:AddItemList("maxPlayers")
maxPlayersList:AddItemRaw("2")
maxPlayersList:AddItemRaw("4")
maxPlayersList:AddItemRaw("6")
```

See Also

None

GameLaunchListItem.GetCallbackFocusItem

Brief

Gets the GameLaunchListItem object for which a focus item has changed.

Definition

```
GameLaunchListItem GameLaunchListItem.GetCallbackFocusItem()
```

Arguments

Return Values

The GameLaunchListItem object for which a focus item has changed.

Description

This function is valid when used within a callback function that was registered with GameLaunchListItem.SetCallbackOnChange()

Examples

```
function OnSessionSetup()
    maxPlayersList = myMenu1:AddItemList("maxPlayers")
    maxPlayersList:AddItemRaw("2")
    maxPlayersList:AddItemRaw("4")
    maxPlayersList:AddItemRaw("6")
    maxPlayersList:SetCallbackOnChange("MaxPlayersChangeCB()")
end

function MaxPlayersChangeCB()
    local listItem = GameLaunchListItem.GetCallbackFocusItem()

    maxPlayers = tonumber( listItem:GetFocusItem() )
end
```

See Also

None

GameLaunchListItem.GetFocusItem

Brief

Queries the current item under focus.

Definition

```
string GameLaunchListItem.GetFocusItem(GameLaunchListItem listItem)
```

Arguments

listItem - The GameLaunchListItem object on which the query is performed.

Return Values

If the item was added using AddItem(), the localisation reference string is returned.
If the item was added using AddItemRaw(), the raw string will be returned.

Description

Examples

```
local listItem = myMenu:GetItemByNameRef("numTracks")

numTracks = tonumber( listItem:GetFocusItem() )
```

See Also

None

GameLaunchListItem.GetNameRef

Brief

Obtains a list item's reference name.

Definition

```
string GameLaunchListItem.GetNameRef(GameLaunchListItem listItem)
```

Arguments

Return Values

Returns the name reference string of the list item item passed in.

Description

GetNameRef() can be useful for debug purposes and also in cases where a developer wishes to manually write out a custom session XML file upon launching.

Examples

```
local item = myMenu1:GetFirstItem()

while item ~= nil do
    -- Reset all sliders

    if Type(item) == "GameLaunchListItem" then
        local listName = item:GetNameRef()

        -- Insert code here.

    end

    item = myMenu1:GetNextItem(item)
end
```

See Also

None

GameLaunchListItem.RemoveAllItems

Brief

Removes all the items from a GameLaunchListItem menu item.

Definition

```
GameLaunchListItem.RemoveAllItems(GameLaunchListItem listItem)
```

Arguments

listItem - The GameLaunchListItem object

Return Values

Description

Examples

```
local myList = myMenu1:FindItemByNameRef("stage3")
myList:RemoveAllItems()
```

See Also

None

GameLaunchListItem.RemoveItem

Brief

Removes an item from a GameLaunchListItem menu item.

Definition

```
GameLaunchListItem.RemoveItem(GameLaunchListItem listItem, string itemNameRef)
```

Arguments

listItem - The GameLaunchListItem object
itemNameRef - The reference string of the item to be removed. If the item was added using AddItem() then the localisation reference string should be passed in, if AddItemRaw() was used then pass in the raw string instead.

Return Values

Description

Examples

```
local myList = myMenu1:FindItemByNameRef("stage3")
myList:RemoveItem("option2")
```

See Also

None

GameLaunchListItem.SetCallbackOnChange

Brief

Sets a callback function to be called when an item selection changes.

Definition

```
GameLaunchListItem.SetCallbackOnChange(string callbackFunction)
```

Arguments

callbackFunction - The name of the callback function. The exact string of a function call should be specified including parenthesis.

Return Values

Description

Examples

```
function OnSessionSetup()
    maxPlayersList = myMenu1:AddItemList("maxPlayers")
    maxPlayersList:AddItemRaw("2")
    maxPlayersList:AddItemRaw("4")
    maxPlayersList:AddItemRaw("6")
    maxPlayersList:SetCallbackOnChange("MaxPlayersChangeCB()")
end

function MaxPlayersChangeCB()
    local listItem = GameLaunchListItem.GetCallbackFocusItem()

    maxPlayers = tonumber( listItem:GetFocusItem() )
end
```

See Also

None

GameLaunchListItem.SetFocusItem

Brief

Sets the current item under focus.

Definition

```
GameLaunchListItem.SetFocusItem(GameLaunchListItem listItem, string itemNameRef)
```

Arguments

listItem - The GameLaunchListItem object
itemNameRef - The reference string of the item that is to become the focus item. If the item was added using AddItem() then the localisation reference string should be passed in, if AddItemRaw() was used then just the raw string instead.

Return Values

Description

Examples

```
maxPlayersList = myMenu1:AddItemList("maxPlayers")
maxPlayersList:AddItemRaw("2")
maxPlayersList:AddItemRaw("4")
maxPlayersList:AddItemRaw("8")
maxPlayersList:SetFocusItem("8")
```

See Also

None

GameLaunchMenu

GameLaunchMenu.AddItemButton

Brief

Adds a button item to the menu specified.

Definition

```
GameLaunchButton GameLaunchMenu.AddItemButton(GameLaunchMenu menu, string itemNameRef, string
insertRef)
```

Arguments

menu - Menu to where the button should be added to. itemNameRef - Localization reference string for the display name of this button. insertRef - Specifies the position of where the item should be inserted. The 'itemNameRef' of an existing item can be passed. This is an optional argument, not specifying
a value will result in the button being appended as the last item.

Return Values

The newly added GameLaunchButton instance.

Description

Adds a button item to the menu specified.

Examples

```
mainMenu = GameLaunchMenuSys.CreateMenu("mainMenu", "mainMenu", true)

if mainMenu ~= nil then

    local userComments = me:GetLocalisedText("ref_userComments")
    myButton = myMenu1:AddItemButton(userComments)

end
```

See Also

GameLaunchMenu.AddItemList

Brief

Adds a list item to the menu specified.

Definition

```
GameLaunchListItem GameLaunchMenu.AddItemList(GameLaunchMenu menu, string listItemNameRef)
```

Arguments

menu - Menu to where the list should be added to. itemNameRef - Localization reference string for the display name of this slider. insertPosition - Specifies the position of where the item should be inserted. The 'itemNameRef' of an existing item can be passed. This is an optional argument, not specifying
a value will result in the slider being appended as the last item.

Return Values

The newly added GameLaunchListItem instance.

Description

Adds a list item to the menu specified.

Examples

```
mainMenu = GameLaunchMenuSys.CreateMenu("mainMenu", "mainMenu", true)

if mainMenu ~= nil then

    spawnDelayList = mainMenu:AddItemList("spawnDelay")
    spawnDelayList:AddItem("none")
    spawnDelayList:AddItem("oneSecond")
    spawnDelayList:AddItem("twoSeconds")
    spawnDelayList:AddItem("threeSeconds")
    spawnDelayList:AddItem("sixSeconds")
    spawnDelayList:SetFocusItem("none")

end
```

See Also

GameLaunchMenu.AddItemSlider

Brief

Adds a slider item to the menu specified.

Definition

```
GameLaunchSlider GameLaunchMenu.AddItemSlider(GameLaunchMenu menu, string itemNameRef, number
minValue, number maxValue, number stepValue, number defaultValue, string insertRef)
```

Arguments

menu - Menu to where the slider should be added to.itemNameRef - Localization reference string for the display name of this slider.minValue - The slider's minimum value.maxValue - The slider's maximum value.stepValue - The step value.defaultValue - Default valueinsertRef - Specifies the position of where the item should be inserted. The 'itemNameRef' of an existing item can be passed. This is an optional argument, not specifying
a value will result in the slider being appended as the last item.

Return Values

The newly added GameLaunchSlider instance.

Description

Adds a slider item to the menu specified.

Examples

```
mainMenu = GameLaunchMenuSys.CreateMenu("mainMenu", "mainMenu", true)

if mainMenu ~= nil then

    mySlider1 = myMenu1:AddItemSlider("difficulty", 1, 8, 1, 1)
    mySlider1:SetPercentageSlider(false)
    mySlider1:SetIntegerSlider(true)

end
```

See Also

GameLaunchMenu.FindItemByNameRef

Brief

Finds a menu item within a given menu.

Definition

```
GameLaunchItem GameLaunchMenu.FindItemByNameRef(GameLaunchMenu menu, string itemNameRef)
```

Arguments

menu - The menu to query. itemNameRef - The localization reference string of the item being searched for.

Return Values

If the search was successful the object of the item is returned. Valid return types are GameLaunchItemList and GameLaunchSlider. If the search failed nil will be returned.

Description

Finds a menu item within a given menu.

Examples

```
local brightnessSlider = menu1:FindItemByNameRef("brightness")

if brightnessSlider ~= nil then
    brightnessSlider:SetValue("20")
end
```

See Also

GameLaunchMenu.GetFirstItem

Brief

Returns the first menu item in a given menu.

Definition

```
[GameLaunchItemList | GameLaunchSlider] GameLaunchMenu.GetFirstItem(GameLaunchMenu menu)
```

Arguments

menu - The menu to query.

Return Values

If successful the object of the item is returned. Valid return types are GameLaunchItemList and GameLaunchSlider. If the search failed nil will be returned.

Description

To be used in conjunction with GameLaunchMenu.GetNextItem()

Examples

```
local item = myMenu1:GetFirstItem()

while item ~= nil do
    -- Reset all sliders

    if Type(item) == "GameLaunchSlider" then
        item:SetValue("0")
    end

    item = myMenu1:GetNextItem(item)
end
```

See Also

GameLaunchMenu.GetId

Brief

Obtains a menu's Id string.

Definition

```
string GameLaunchMenu.GetId(GameLaunchMenu menu)
```

Arguments

menu - The menu to query.

Return Values

Returns the Id of the menu passed in.

Description

Obtains a menu's Id string.

Examples

```
local menu1 = GameLaunchMenuSys.GetFirstMenu()

while menu1 ~= nil do

    if menu1:GetId() == "settings" then

        local brightnessSlider = menu1:FindItemByNameRef("brightness")

        if brightnessSlider ~= nil then
            brightnessSlider:SetValue("20")
        end

        break
    end

    menu1 = GameLaunchMenuSys.GetNextMenu(menu1)
end
```

See Also

GameLaunchMenu.GetNextItem

Brief

Returns the next menu item in a given menu.

Definition

```
[GameLaunchItemList | GameLaunchSlider] GameLaunchMenu.GetNextItem(GameLaunchMenu menu,
[GameLaunchItemSlider|GameLaunchSlider] item)
```

Arguments

menu - The menu to query.item - The item return from the previous call to GetFirstItem() or GetNextItem(), whichever was the most recent.

Return Values

If successful the object of the item is returned. Valid return types are GameLaunchItemList and GameLaunchSlider. If the search failed nil will be returned.

Description

To be used in conjunction with GameLaunchMenu.GetFirstItem()

Examples

```
local item = myMenu1:GetFirstItem()

while item ~= nil do
    -- Reset all sliders

    if Type(item) == "GameLaunchSlider" then
        item:SetValue("0")
    end

    item = myMenu1:GetNextItem(item)
end
```

See Also

GameLaunchMenuSys

GameLaunchMenuSys.AddMenu

Brief

Adds an menu screen to the menu system.

Definition

```
GameLaunchMenuSys.AddMenu( GameLaunchMenu menu, string insertPosition )
```

Arguments

menu - GameLaunchMenu object return from GameLaunchMenuSys.CreateMenu()insertPosition - specifies the position where the menu should be inserted. The menu will be inserted before the
menu with the Id given. This is an optional argument, if it is not specified the menu will be appended to the end.

Return Values

None.

Description

Adds an menu screen to the menu system.

Examples

```
myMenu1 = GameLaunchMenuSys.CreateMenu( "myMenu1", "mainMenu", true )

friendlyFireItem = GameLaunchMenu.AddItemList( myMenu1, "friendlyFire" )
GameLaunchListItem.AddItem( friendlyFireItem, "on" )
GameLaunchListItem.AddItem( friendlyFireItem, "off" )

GameLaunchMenuSys.AddMenu( myMenu1 )
```

See Also

None

GameLaunchMenuSys.CreateMenu

Brief

Creates and returns a GameLaunchMenu object.

Definition

```
GameLaunchMenu GameLaunchMenuSys.CreateMenu( string menuId, string menuNameRef, boolean isSessionMenu
)
```

Arguments

menuId - A unique Id string used to identify this menu screen and is used when exporting the session XML file.menuNameRef - Name given to the menu which will be displayed on screen. Reference string used to look up the translated string in the object's localisation database.isSessionMenu - Session menus are controlled by the Session Master, options contained within such menus become read only once the session has been created.

Return Values

GameLaunchMenu

Description

Creates and returns a GameLaunchMenu object.

Examples

```
menu1 = GameLaunchMenuSys.CreateMenu( "myMenu1", "mainMenu", true )
```

See Also

None

GameLaunchMenuSys.FindMenuById

Brief

Finds a menu from within the menu system.

Definition

```
GameLaunchMenu GameLaunchMenuSys.FindMenuById( string menuId )
```

Arguments

menuId - Menu Id of the menu to search for.

Return Values

GameLaunchMenu object, or nil if not found.

Description

Finds a menu from within the menu system.

Examples

```
local menu = GameLaunchMenuSys.FindMenuById( "myMenu3" )

if menu ~= nil then
    local mySlider1 = GameLaunchMenu.AddItemSlider( menu, "difficulty", 1, 10, 1, 3 )
    GameLaunchSlider.SetPercentageSlider( mySlider1, false )
    GameLaunchSlider.SetIntegerSlider( mySlider1, true )
end
```

See Also

None

GameLaunchMenuSys.GetFirstMenu

Brief

Retrieves the first menu from the menu system.

Definition

```
GameLaunchMenu GameLaunchMenuSys.GetFirstMenu()
```

Arguments

None

Return Values

GameLaunchMenu object, or nil if none found.

Description

Retrieves the first menu from the menu system. Useful when searching for active menus where the GameLaunchMenu object isn't already known, for example, menus created by the Game Launching XML menu format.

Examples

```
local myMenu = GameLaunchMenuSys.GetFirstMenu()

if myMenu ~= nil then

    ...

end
```

See Also

None

GameLaunchMenuSys.GetNextMenu

Brief

Retrieves the next menu from the menu system.

Definition

```
GameLaunchMenu GameLaunchMenuSys.GetNextMenu( GameLaunchMenu myMenu )
```

Arguments

myMenu - GameLaunchMenu object return from the previous GetFirstMenu() or GetNextMenu() call, whichever was the most recent.

Return Values

GameLaunchMenu object, or nil if none found.

Description

Retrieves the next menu from the menu system. To be used in conjunction with GameLaunchMenuSys.GetFirstMenu() Useful when searching for active menus where the GameLaunchMenu object isn't already known, for example, menus created by the Game Launching XML menu format.

Examples

```
local aMenu = GameLaunchMenuSys.GetFirstMenu()

if aMenu ~= nil then

    numMenus = 1

    repeat

        aMenu = GameLaunchMenuSys.GetNextMenu(aMenu)

        if aMenu ~= nil then
            numMenus = numMenus + 1
        end

    until aMenu == nil

end
```

See Also

- GameLaunchMenuSys.GetFirstMenu
- GameLaunchMenuSys.GetNextMenu

GameLaunchMenuSys.RemoveMenu

Brief

Removes an active menu from the menu system.

Definition

```
GameLaunchMenuSys.RemoveMenu( string menuId )
```

Arguments

menuId - Id of the menu to be removed.

Return Values

None.

Description

Removes an active menu from the menu system. This does not destroy the object instance. If the destruction of the object instance is required, this should be done via Lua garbage collection.

Examples

```
local myMenu = GameLaunchMenuSys.FindMenuById( "myMenu1" )

if myMenu ~= nil then
    GameLaunchMenuSys.RemoveMenu( "myMenu" )
end
```

See Also

None

GameLaunchSlider

GameLaunchSlider.GetNameRef

Brief

Obtains a slider's reference name.

Definition

```
string GameLaunchSlider.GetNameRef(GameLaunchSlider sliderItem)
```

Arguments

Return Values

Returns the name reference string of the slider item passed in.

Description

GetNameRef() can be useful for debug purposes and also in cases where a developer wishes to manually write out a custom session XML file upon launching.

Examples

```
local item = myMenu1:GetFirstItem()

while item ~= nil do
    -- Reset all sliders

    if Type(item) == "GameLaunchSlider" then
        local sliderName = item:GetNameRef()

        -- Insert code here.

    end

    item = myMenu1:GetNextItem(item)
end
```

See Also

None

GameLaunchSlider.GetValue

Brief

Gets the current value of the slider.

Definition

```
GameLaunchSlider.GetValue(GameLaunchSlider sliderItem, number value)
```

Arguments

sliderItem - The slider object

Return Values

Description

Examples

```
local levelSlider = myMenu1:FindItemByNameRef("level")

if levelSlider ~= nil then
    currentLevel = levelSlider:GetValue()
end
```

See Also

None

GameLaunchSlider.SetIntegerSlider

Brief

Toggles the slider as a integer type slider.

Definition

```
GameLaunchSlider.SetIntegerSlider(GameLaunchSlider sliderItem, boolean enable)
```

Arguments

sliderItem - The slider object
enable - Flag to specify if slider should be a integer type slider. If this value is false, the slider will use floating point values instead.

Return Values

Description

Examples

```
levelSlider = myMenu1:AddItemSlider("level", 1, 10, 1, 3)
levelSlider:SetPercentageSlider(false)
levelSlider:SetIntegerSlider(true)
```

See Also

None

GameLaunchSlider.SetPercentageSlider

Brief

Toggles the slider as a percentage type slider.

Definition

```
GameLaunchSlider.SetPercentageSlider(GameLaunchSlider sliderItem, boolean enable)
```

Arguments

sliderItem - The slider object
enable - Flag to specify if slider should be a percentage type slider.

Return Values

Description

Note: Percentage sliders using floating point values is not supported.

Examples

```
energySlider = myMenu1:AddItemSlider("energy", 1, 100, 10, 80)
energySlider:SetPercentageSlider(true)
energySlider:SetIntegerSlider(true)
```

See Also

None

GameLaunchSlider.SetValue

Brief

Sets the value of the slider.

Definition

```
GameLaunchSlider.SetValue(GameLaunchSlider sliderItem, number value)
```

Arguments

sliderItem - The slider object
value - The value to which the slider should be set to.

Return Values

Description

Examples

```
local slider = myMenu1:FindItemByNameRef("level")

if slider ~= nil then
    slider:SetValue(3)
end
```

See Also

None

GfxEffect

GfxEffect.Create

Brief

Creates a new GfxEffect object.

Definition

```
GfxEffect GfxEffect.Create( )
```

Arguments

None

Return Values

A new GfxEffect instance.

Description

Creates a new GfxEffect object. By default, an effect is not enabled and will not render. To enable the effect, call GfxEffect.Enable.

Examples

```
gfxEffect = GfxEffect.Create()
-- set up geometry
gfxEffect:Enable(true)
```

See Also

- GfxEffect.Enable

GfxEffect.Enable

Brief

Enable or disable rendering of the GfxEffect.

Definition

```
GfxEffect.Enable(GfxEffect effect, boolean enable)
```

Arguments

effect - Effect to modify.enable - True to enable or false to disable.

Return Values

None.

Description

This function controls whether the GfxEffect instance is rendered or not. This state has no effect on the data within the effect, so it can be enabled and disabled without re-specifying the data. If you wish to delete the effect altogether, allow it to be garbage collected.

Examples

```
gfxEffect:Enable(false)
```

See Also

- GfxEffect.IsEnabled

GfxEffect.IsEnabled

Brief

Indicates whether the GfxEffect instance is enabled.

Definition

```
boolean GfxEffect.IsEnabled(GfxEffect effect)
```

Arguments

effect - Effect to query.

Return Values

True if enabled or false otherwise.

Description

An effect must be enabled in order to render.

Examples

```
if (not gfxEffect:IsEnabled()) then
    gfxEffect:Enable(true)
end
```

See Also

- GfxEffect.Enable

GfxEffect.SetAlphaRef

Brief

Set the reference value used for cutout effects.

Definition

```
GfxEffect.SetAlphaRef( GfxEffect effect, number alphaRef )
```

Arguments

effect - The GfxEffect instance.alphaRef - The reference value to use for alpha testing.

Return Values

None.

Description

An effect rendered with the cutout blend mode will apply an alpha test when rendering, using the value supplied to this function as the reference value. Any geometry with an alpha value greater than or equal to the reference value will be rendered, alpha values lower than the reference will not be rendered.

Examples

```

```

See Also

- GfxEffect.SetBlendMode

GfxEffect.SetBlendMode

Brief

Set the blending mode for the effect.

Definition

```
GfxEffect.SetBlendMode( GfxEffect effect, BlendMode mode )
```

Arguments

effect - The effect instance.mode - The desired blend mode. Valid values are:
BlendMode.Opaque => no blend, render opaque
BlendMode.Cutout => render opaque with alpha test using AlphaRef value
BlendMode.Blend => SrcAlpha*Src + (1-SrcAlpha)*Dst
BlendMode.Add => Src + Dst
BlendMode.Multiply => Src * Dst
BlendMode.AlphaAdd => SrcAlpha*Src + Dst
BlendMode.BlendPremultiplied => Src + (1-SrcAlpha)*Dst

Return Values

None.

Description

The blend mode controls how the GfxEffect is blended onto the final frame. Opaque renders the geometry unblended, and cutout

applies an alpha test with the threshold set with SetAlphaRef when rendering. For effect of other modes please refer to BlendMode value list in this help entry.

Examples

```
```

See Also

- GfxEffect.SetAlphaRef
- GfxEffect.SetRenderStage
- GfxEffect.SetDepthPrepass

GfxEffect.SetBounds

Brief

Sets the local axis aligned bounding box of the effect.

Definition

```
GfxEffect.SetBounds( GfxEffect effect, Vector4 min, Vector4 max )
```

Arguments

effect - The instance to modify.min - The local space minimum extent of the effect.max - The local space maximum extent of the effect.

Return Values

None.

Description

GfxEffect instances require a local bounding box in order to be culled when not in view. The bounding box should encompass all of the geometry created by the effect.

Examples

```
local vertData = MemoryContainer.Create(numVerts * 4)       -- 4 bytes per vertex
local offset = 0
for i = 1, numVerts do
    local v = GenerateVert(i)       -- user function to compute some vertex position
    vertData:SetVector4(offset, v)
    offset = offset + 4
end
-- compute bounds
local min = Vector4.Create(Const.MaxNum, Const.MaxNum, Const.MaxNum)
local max = -min
StreamOps.BoundsVec4(verts, 0, numVerts, min, max)

-- set up gfxeffect
gfxEffect = GfxEffect.Create()
gfxEffect:SetVertexData(GfxVertAttrib.Position, vertData, 4)
-- will also need some indices here...
gfxEffect:SetBounds(min, max)
```

See Also

- StreamOps.BoundsVec4

GfxEffect.SetDepthPrepass

Brief

Set whether the effect renders a depth prepass.

Definition

```
GfxEffect.SetDepthPrepass( GfxEffect effect, boolean flag )
```

Arguments

effect - The GfxEffect instance.flag - True to turn on depth prepass rendering, false otherwise.

Return Values

None.

Description

Setting this flag will render the effect into the depth buffer first, and only render faces passing an equal depth test when rendering to the back buffer. Defaults to off.

Examples

```

```

See Also

- GfxEffect.SetBlendMode

GfxEffect.SetDrawRange

Brief

Sets the range of triangles to draw.

Definition

```
GfxEffect.SetDrawRange( GfxEffect effect, number start, number count )
```

Arguments

effect - The instance to set the draw range on.start - Offset into indices at which to start drawing.count - The number of triangles to draw.

Return Values

None.

Description

It is more optimal to create one GfxEffect and render subsections of its geometry than create separate GfxEffects. This function can be used to draw a subset of the triangles in the GfxEffect.

Examples

```
gfxEffect:SetDrawRange(6, 12)   -- draw 12 triangles starting from index 6 (ie, skip first 2 tris)
```

See Also

- GfxEffect.SetIndexData

GfxEffect.SetIndexData

Brief

Sets the triangle indices for the effect.

Definition

```
GfxEffect.SetIndexData( GfxEffect effect, MemoryContainer indices, number offset = 0 )
```

Arguments

effect - The instance to set the indices on.indices - The memory container with vertex indices as a triangle list, 3 indices per triangle.
The memory container should be allocated in host memory and 128-byte aligned.offset - The offset into the memory container at which to find the indices.

Return Values

None.

Description

The geometry is specified as a triangle list with 3 indices per triangle. As such there will be triangleCount * 3 indices required. Each index is assumed to be 2 bytes in size (65k vertex limit). Note that the index data is required to be 128-byte aligned, and located in host memory. The memory is used by the GfxEffect directly, i.e. no copy is made, so the user should ensure the memory container exists for as long as the GfxEffect exists. Clockwise winding indicates a front-facing triangle. The number of indices to use is indicated by the SetDrawRange function.

Examples

```

```

See Also

- GfxEffect.SetVertexData
- GfxEffect.SetDrawRange

GfxEffect.SetRenderStage

Brief

Set the render stage for semi-transparent effects.

Definition

```
GfxEffect.SetRenderStage( GfxEffect effect, RenderStage mode )
```

Arguments

effect - The effect instance.mode - The desired render stage. Valid values are:
RenderStage.PreWorld
RenderStage.PostWorld
RenderStage.PostParticles

Return Values

None.

Description

For effects being blended into the frame, this function controls the render stage at which the effect is rendered. Geometry rendered in subsequent stages will be blended on top of geometry rendered at at earlier stage. The render stages are listed in this entry in temporal order.

Examples

See Also

GfxEffect.SetShadowEnable

Brief

Set whether the effect casts a dynamic shadow.

Definition

```
GfxEffect.SetShadowEnable( GfxEffect effect, boolean flag )
```

Arguments

effect - The GfxEffect instance.flag - True to cast a dynamic shadow from this effect, or false otherwise.

Return Values

None.

Description

Enabling this flag will cause the effect to cast a dynamic shadow onto the scene, at a small performance cost. Defaults to off.

Examples

See Also

None

GfxEffect.SetTexture

Brief

Sets the given texture on the given shader uniform parameter.

Definition

```
GfxEffect.SetTexture( GfxEffect effect, TextureType type, Texture|string|resource tex )
```

Arguments

effect - The GfxEffect instance.type - The texture type to set:
TextureType.ColorMap
tex - The texture, texture resource name, or texture resource.

Return Values

None.

Description

Set the texture to use for the specified texture slot.

Examples

```
tex = Texture.Find("water")
gfxEffect:SetTexture( TextureType.ColorMap, tex )
```

See Also

- Texture.Find
- Resource.Find

GfxEffect.SetTransform

Brief

Sets the world transform for the effect.

Definition

```
GfxEffect.SetTransform( GfxEffect effect, Matrix44 transform )
```

Arguments

effect - The instance to modify.transform - The world space tranform.

Return Values

None.

Description

Sets the world transform for the effect, used to place it within the scene.

Examples

```
local transform = Matrix44.Create()
transform:SetTranslation(Vector4.Create(1, 2, 3))
gfxEffect:SetTransform(transform)
```

See Also

None

GfxEffect.SetTwoSided

Brief

Set whether the effect is rendered two-sided.

Definition

```
GfxEffect.SetTwoSided( GfxEffect effect, boolean flag )
```

Arguments

effect - The GfxEffect instance.flag - True to render two-sided, false otherwise.

Return Values

None.

Description

If the geometry should be considered two-sided, set this flag to true. Defaults to on.

Examples

See Also

None

GfxEffect.SetVertexData

Brief

Sets a vertex data attribute for the effect.

Definition

```
boolean GfxEffect.SetVertexData( GfxEffect effect, VertexDataType type, MemoryContainer data, number
numElementsPerVertex, number offset = 0, number mode = DataRepeat.None, number frequency = 0)
```

Arguments

effect - The instance to set the vertex data into.type - Vertex data type:
VertexDataType.InPosition
VertexDataType.InTexCoord1
VertexDataType.InColordata - Memory container specifying the vertex data. The memory container should be allocated in host memory.numElementsPerVertex - number of elements, range is [1, 4].mode - The desired vertex data repeat mode. Valid values are:
DataRepeat.None
DataRepeat.Element
DataRepeat.Bufferfrequency - the data repeat frequency, a value of 0 disables element or buffer repeat.

Return Values

True if the call succeeded, false otherwise.

Description

Supply the vertex data (position, colour, texture coords) for a GfxEffect. Note that the memory is used by the GfxEffect directly, no copy is made. Therefore the user should ensure the memory container exists for as long as the GfxEffect exists. All elements are floats, including color. All vertex data streams must be specified for the effect to render.

Examples

See Also

- GfxEffect.SetIndexData

Group

Group.Create

Brief

Create a group.

Definition

```
boolean Group.Create()
```

Arguments

None

Return Values

True if the group creation process started, false otherwise (already a member of a group).

Description

Triggers the group creation process. A script should query for group membership to determine when the group has been created. This function will fail if the user is already a member of a group.

Examples

```
if (Group.Create()) then
    state = CREATING_GROUP
end
```

See Also

- Group.Leave
- Group.GetId
- Group.IsInGroup
- Group.Join

Group.GetId

Brief

Get the ID of the currently joined group.

Definition

```
boolean Group.GetId(BigInt<64> id)
```

Arguments

id - 64-bit BigInt to receive the group ID.

Return Values

True upon success, false if not a member of a group.

Description

Obtain the ID of the group which the current user belongs to. If they are not a member of a group this function will return false and the BigInt is not modified.

Examples

```
local groupId = BigInt.Create(64)
if (Group.GetId(groupId)) then
    local msg = OutboundMessage.Create(JOIN_GROUP, 8)
    msg:WriteBigInt(groupId)
    msg:Send()
end
```

See Also

- Group.Join
- Group.Create

Group.GetMemberCount

Brief

Get the number of members of the current user's group.

Definition

```
number Group.GetMemberCount()
```

Arguments

None

Return Values

The number of group members.

Description

Gets the number of members of the current user's group. If the current user does not have a group, this function will raise an error.

Examples

```
local numMembers = Group.GetMemberCount()
```

See Also

- Group.GetMemberList
- Group.IsInGroup

Group.GetMemberList

Brief

Get the list of members of the current user's group.

Definition

```
table Group.GetMemberList()
```

Arguments

None

Return Values

An array with one table for each member, with the following fields:

{
number id,
User user
}

Description

Get the list of members of the current user's group. If the current user does not have a group, this function will raise an error.

Examples

```
local members = Group.GetMemberList()
print('Group members:')
for i, m in ipairs(members) do
    print(m.user:GetName())
end
```

See Also

- Group.GetMemberCount
- Group.IsInGroup

Group.IsInGroup

Brief

Returns true if the current member is in a group.

Definition

```
boolean Group.IsInGroup()
```

Arguments

None

Return Values

true if the current user is in a group, false otherwise.

Description

Returns true if the current member is in a group.

Examples

```
if (Group.IsInGroup()) then
    print('You have ' .. Group.GetGroupMemberCount() .. ' people in your group!')
end
```

See Also

- Group.GetMemberCount
- Group.GetMemberList

Group.Join

Brief

Join an existing group.

Definition

```
boolean Group.Join(BigInt<64> groupId)
```

Arguments

None

Return Values

True if the group join process started, false otherwise (already a member of a group, or group doesn't exist).

Description

Triggers the group join process. A script should query for group membership to determine when the group has been joined. This function will fail if the user is already a member of a group.

Examples

```
function OnReceivedMessage()
    local msg = ReceivedMessage.GetEventMessage()
    local id = msg:GetType()
    if (id == JOIN_GROUP) then
        groupId = BigInt.Create(64)
        msg:ReadBigInt(groupId)
        joinGroup = true
    end
end

function Update()
    if (joinGroup) then
        joinGroup = false
        if (Group.Join(groupId)) then
            state = JOINING_GROUP
        else
            print("Group join failed!")
            state = JOIN_ERROR      -- could retry, or ask for group ID to be resent
        end
    end
end
```

See Also

- Group.Leave
- Group.GetId
- Group.IsInGroup
- Group.Create

Group.Leave

Brief

Leave a script-created group.

Definition

```
boolean Group.Leave()
```

Arguments

None

Return Values

False if the user is in a non-script created group, or true otherwise.

Description

Removes the local user from a group he or she was previously joined to via script. This function will not remove the user from a group they joined or created themselves.

Examples

```
if (Group.Leave()) then
    state = LEAVING_GROUP
else
    print("User is in their own group!")
end
```

See Also

- Group.IsInGroup
- Group.Create
- Group.Join

GroupDoor

GroupDoor.Create

Brief

Creates a new Group Door.

Definition

```
GroupDoor GroupDoor.Create(string sceneName, boolean isSimpleDoor = true)
```

Arguments

sceneName - the Scene Name of the Group Instance.isSimpleDoor - if set to true (default) this will automate letting the user know if/why they weren't
allowed to enter the door via standard client notifications. Otherwise the user needs to be informed manually,
which will involve using the 'GroupDoor.GetFailedEnterMessage' function. Please see GetFailedEnterMessage
for important details about displaying the message manually.
Note that GroupDoor functionality is currently not available in a personal space. If this function is called
from a personal space, nil is returned.

Return Values

A new Group Door object, or nil on failure.

Description

Creates a new Group Door for accessing a Group Instance, for playing private Group Mini Games.

Examples

```
groupDoor = GroupDoor.Create(sceneName)
groupDoor:Enter()
```

See Also

- GroupDoor.Enter
- GroupDoor.GetFailedEnterMessage

GroupDoor.Enter

Brief

Enters the specified Group Door.

Definition

```
boolean GroupDoor.Enter()
```

Arguments

None

Return Values

True if successful, false otherwise. The requirements for entering a Group Door include being a Member of a Group, and there being two or more Members in that Group.

Description

This will trigger the relocation system (if the Group Door entry requirements are met). Note: The term 'GroupDoor' is developer-facing terminology and should not be presented to users. When using this feature, please describe it in more user-friendly terms, for example: 'Play this game in a Group Space', 'Play this game with your Group', or 'To play this game with your Group, go through this door'.

Examples

```
groupDoor = GroupDoor.Create(sceneName)
groupDoor:Enter()
```

See Also

- GroupDoor.Create
- GroupDoor.GetFailedEnterMessage

GroupDoor.GetFailedEnterMessage

Brief

Provides a localized string, for use by the developer, to inform the user why an attempt to enter the associated Group Door failed.

Definition

```
string GroupDoor.GetFailedEnterMessage()
```

Arguments

None

Return Values

The error message.

Description

If deciding to manually present this information, this string can be used. The string returned is localized correctly for the current user. It's important to note that it should always be assumed that the message string could be of any length. Messages may vary due to additions, updates, and the language used. The way the message is presented should take this into account, with formatting and scrolling as appropriate.

Examples

```
groupDoor = GroupDoor.Create(sceneName, false)
if (not groupDoor:Enter()) then
    popupMessage = groupDoor:GetFailedEnterMessage()
end
```

See Also

- GroupDoor.Create
- GroupDoor.Enter

Hash

Hash.Create

Brief

Creates a new hash object.

Definition

```
Hash.Create( string type )
```

Arguments

type - The hash type

Return Values

A new instance of a hash object.

Description

Creates a new hash object. Supported types are "SHA1", "MD5".

Examples

```
myHash = Hash.Create( "SHA1" )
```

See Also

Hash.Finalize

Brief

Finalize the hash calculation and return the result.

Definition

```
Hash.Finalize()
```

Arguments

None

Return Values

string - The hash result.

Description

Finalize the hash calculation and return the result.

Examples

```
local result = myHash.Finalize()
```

See Also

Hash.Start

Brief

Prepares a hash object ready for calculating a new hash.

Definition

```
Hash.Start()
```

Arguments

None

Return Values

None

Description

Prepares a hash object ready for calculating a new hash.

Examples

```
myHash.Start()
```

See Also

Hash.Update

Brief

Updates a hash calculation.

Definition

```
Hash.Update( variant data )
```

Arguments

data - A string, number or MemoryContainer

Return Values

None

Description

Updates a hash calculation. Numbers will be hashed as floats.

Examples

```
myHash.Update( "My String" )
```

See Also

HelpSystem

HelpSystem.GetObjectHelpText

Brief

Retrieve the help text associated with an object.

Definition

```
HelpSystem.GetObjectHelpText()
```

Arguments

objectId - The ID of the object to retrieve help text for (string)

Return Values

String if help text exists, otherwise nil

Description

Retrieve the help text associated with

Examples

```
helpText = HelpSystem.GetObjectHelpText( "12345678-12345678-12345678-12345678" )
```

See Also

None

HttpPostData

HttpPostData.AddDataHeader

Brief

Adds a data header to the post data.

Definition

```
boolean HttpPostData.AddDataHeader( HttpPostData postData, string key, string mimeType = nil, string
filename )
```

Arguments

postData - The instance of the post data.key - The name of the value to be added to the post data.mimeType - Optional. The mime type of the data to add to the post data (e.g. "text/xml").filename - Optional. A file name to set for the data. May be needed for the webserver to see the data as a file

Return Values

True if the header was added.

Description

Adds a data header to the post data. This assumes the the script will add the actual data directly in some other way such as writing an XML file into the container. The data must be finalized using FinalizeData before any new values can be added to the post data.

Examples

```
success = HttpPostData.AddDataHeader( postData, "file", "text/xml" )
```

See Also

- HttpPostData.FinalizeData

HttpPostData.AddHttpHeader

Brief

Adds an http header key value pair.

Definition

```
HttpPostData.AddHttpHeader( HttpPostData postData, string key, boolean|number|string|nil value )
```

Arguments

postData - The instance of the post data.key - The name of the value to be added to the header.value - A boolean/number/string/nil to use as the value to add to the header.

Return Values

Description

Adds a value to the http header. The function will error if there is not enough space in the container to add the value. Headers must be added before adding of any data, signify the of the header section by calling HttpPostData.FinalizeHttpHeaders.

Examples

```
container = MemoryContainer.Create( 4096 )

postdata = HttpPostData.Create( container, "multipart/form-data" )

postdata:AddHttpHeader( "key1", "value1" )
postdata:AddHttpHeader( "key2", "value2" )
postdata:AddHttpHeader( "key3", "value3" )
postdata:FinalizeHttpHeaders()

postdata:AddValue( "name", "Bob" )
postdata:AddValue( "password", "abc123" )

postdata:Finalize()
```

See Also

HttpPostData.AddValue

Brief

Adds a value to the post data.

Definition

```
boolean HttpPostData.AddValue( HttpPostData postData, string key, variant value )
```

Arguments

postData - The instance of the post data.key - The name of the value to be added to the post data.value - A boolean/number/string to use as the value to add to the post data.

Return Values

True if the value was added.

Description

Adds a value to the post data. The function will return false if there is not enough space in the container to add the value.

Examples

```
success = HttpPostData.AddValue( postData, "name", "Bob" )
```

See Also

None

HttpPostData.Create

Brief

Creates a new HttpPostData object.

Definition

```
HttpPostData.Create( MemoryContainer container, string contentType )
```

Arguments

container - The memory container to write the post data to.contentType - The encoding used for the post data. Specify either "application/x-www-form-urlencoded" or "multipart/form-data".

Return Values

HttpPostData - A new instance of post data.

Description

Creates a new HttpPostData object. The memory container used will be reset so an data held in it will be lost.

Examples

```
container = MemoryContainer.Create( 4096 )
postData = HttpPostData.Create( container )
```

See Also

HttpPostData.Finalize

Brief

Finalizes the post data.

Definition

```
boolean HttpPostData.Finalize( HttpPostData postData )
```

Arguments

postData - The instance of the post data.

Return Values

True if the post data was finalized.

Description

Finalizes the post data so that it can be sent up to a server. No new data can be added after this call.

Examples

```
success = HttpPostData.Finalize( postData )
```

See Also

- HttpPostData.AddDataHeader

HttpPostData.FinalizeData

Brief

Finalizes any data that has been added after calling AddDataHeader.

Definition

```
boolean HttpPostData.FinalizeData( HttpPostData postData )
```

Arguments

postData - The instance of the post data.

Return Values

True if the data value was finalized.

Description

Finalizes any data that has been added after calling AddDataHeader so that new data values can be added to the post data.

Examples

```
success = HttpPostData.FinalizeData( postData )
```

See Also

- HttpPostData.AddDataHeader

HttpPostData.FinalizeHttpHeaders

Brief

Finalizes the header section after calling AddHttpHeader.

Definition

```
HttpPostData.FinalizeHttpHeaders( HttpPostData postData )
```

Arguments

postData - The instance of the post data.

Return Values

Description

Finalizes the header section after calling AddHttpHeader. No further headers can be added after calling this function.

Examples

```
HttpPostData.FinalizeHttpHeaders( postData )
```

See Also

- HttpPostData.AddHttpHeader

HttpPostData.Reset

Brief

Resets the post data.

Definition

```
HttpPostData.Reset( HttpPostData postData )
```

Arguments

postData - The instance of the post data.

Return Values

None.

Description

Resets the post data so that a new request can be created using the same object.

Examples

```
HttpPostData.Reset( postData )
```

See Also

None

Intersect

Intersect.AabbAabb2d

Brief

Intersect two AABBs in 2D.

Definition

```
boolean [, number, number] Intersect.AabbAabb2d(Vector4 box1min, Vector4 box1max, Vector4 box2min,
Vector4 box2max, boolean pd = false)
boolean [, number, number] Intersect.AabbAabb2d(number b1minX, number b1minY, number b1maxX, number
b1maxY,
                                                number b2minX, number b2minY, number b2maxX, number
b2maxY, boolean pd = false)
```

Arguments

box1Min - Minimum extents of AABB 1.box1Max - Maximum extents of AABB 1.box2Min - Minimum extents of AABB 2.box2Max - Maximum extents of AABB 2.b1minX - Minimum extent on X axis of AABB 1.b1maxX - Maximum extent on X axis of AABB 1.b1minY - Minimum extent on Y axis of AABB 1.b1maxY - Maximum extent on Y axis of AABB 1.b2minX - Minimum extent on X axis of AABB 2.b2maxX - Maximum extent on X axis of AABB 2.b2minY - Minimum extent on Y axis of AABB 2.b2maxY - Maximum extent on Y axis of AABB 2.pd - True to calculate and return penetration distances, or false otherwise.

Return Values

True if the boxes overlap and false if not, and if pd is specified, additionally returns the maximum penetration depth on the x and y axes in that order.

Description

Intersect two axis-aligned bounding boxes in 2D.

Examples

```
box1Min = Vector4.Create(1, 0.5)
box1Max = Vector4.Create(3, 1.5)
box2Min = Vector4.Create(2.5, 1)
box2Max = Vector4.Create(5, 3)
intersect, pdx, pdy = Intersect.AabbAabb2d(box1Min, box1Max, box2Min, box2Max, true)
if (intersect) then
    print("Intersection, penetration axis", pdx <= pdy and "X" or "Y")
end
```

See Also

None

Intersect.AabbAabb3d

Brief

Intersect two AABBs in 3D.

Definition

```
boolean [, number, number, number] Intersect.AabbAabb3d(Vector4 box1Min, Vector4 box1Max, Vector4
box2Min, Vector4 box2Max, boolean pd)
```

Arguments

box1Min - Minimum extents of AABB 1.box1Max - Maximum extents of AABB 1.box2Min - Minimum extents of AABB 2.box2Max - Maximum extents of AABB 2.pd - Calculate and return penetration distance on each axis.

Return Values

True if the boxes intersect, or false otherwise. Additionally, if the parameter pd is true, the penetration distance on each axis (X, Y, Z) is returned, in that order.

Description

Intersect two axis-aligned bounding boxes in 3D.

Examples

```
box1Min = Vector4.Create(1, 0.5, 1)
box1Max = Vector4.Create(3, 1.5, 2)
box2Min = Vector4.Create(2.5, 1, 0)
box2Max = Vector4.Create(5, 3, 1)
intersect, pdx, pdy, pdz = Intersect.AabbAabb3d(box1Min, box1Max, box2Min, box2Max, true)
if (intersect) then
    print("Intersection, max penetration", math.max(pdx, pdy, pdz))
end
```

See Also

None

Intersect.AabbPlane3d

Brief

Intersect an AABB with a plane in 3D.

Definition

```
IntersectSide, number Intersect.AabbPlane3d(Vector4 boxMin, Vector4 boxMax, Vector4 plane)
```

Arguments

boxMin - the minimum extents of the AABB.boxMax - the maximum extents of the AABB.plane - plane equation [normal, distance].

Return Values

The classification of the box relative to the plane and the penetration distance of the AABB relative to the plane:
IntersectSide.Positive - In the positive half-space of the plane.
IntersectSide.Intersect - Overlapping (intersecting) the plane.
IntersectSide.Negative - In the negative half-space of the plane.

Description

The AABB is classified as either fully in the positive halfspace of the plane, fully in the negative halfspace, or overlapping the plane. The penetration distance is also given, which will be positive if any part of the box lies in the negative halfspace of the plane, or negative if fully in the positive halfspace. The penetration distance is given along the plane normal.

Examples

```
boxMin = Vector4.Create(1, 0.5, 0)
boxMax = Vector4.Create(3, 1.5, 0)
if (Intersect.AabbPlane3d(boxMin, boxMax, Vector4.Create(0, -1, 0, 1)) == IntersectSide.Intersect)
then
    print("Intersection!")
end
```

See Also

None

Intersect.CircleAabb2d

Brief

Intersect a circle and an AABB in 2D.

Definition

```
boolean Intersect.CircleAabb2d(Vector4 center, number radius, Vector4 boxMin, Vector4 boxMax)
boolean Intersect.CircleAabb2d(number centerX, number centerY, number radius, number boxMinX, number
boxMinY, number boxMaxX, number boxMaxY)
```

Arguments

center - Circle center point. radius - Circle radius. boxMin - AABB minimum extents. boxMax - AABB maximum extents. centerX - Circle center X coordinate. centerY - Circle center Y coordinate. boxMinX - AABB minimum extent on X axis. boxMinY - AABB minimum extent on Y axis. boxMaxX - AABB maximum extent on X axis. boxMaxY - AABB maximum extent on Y axis.

Return Values

True if the circle and AABB overlap, or false if not.

Description

Intersect a circle and an AABB in 2D.

Examples

```
center = Vector4.Create(0, 1)
radius = 2
boxMin = Vector4.Create(1, 0.5)
boxMax = Vector4.Create(3, 1.5)
if (Intersect.CircleAabb2d(center, radius, boxMin, boxMax)) then
    print("Intersection!")
end
```

See Also

None

Intersect.CircleCircle2d

Brief

Intersect two circles in 2D.

Definition

```
boolean [, number] Intersect.CircleCircle2d(Vector4 center1, number radius1, Vector4 center2, number
radius2, boolean pd = false)
```

Arguments

center1 - center point of circle 1.radius1 - radius of circle 1.center2 - center point of circle 2.radius2 - radius of circle 2.pd - return penetration distance.

Return Values

True if circles intersect or false otherwise. If pd is true, additionally returns the penetration distance of the circles.

Description

Intersect two circles in 2D. If pd is true, also returns the penetration distance, which is >= 0 if the circles intersect, or negative otherwise. If the penetration distance is greater than the smaller radius of the two, the small circle lies wholly inside the large one. The penetration distance lies along the vector between the circles' centers.

Examples

```
local intersect, pd = Intersect.CircleCircle2d(center1, radius1, center2, radius2, true)
if (intersect) then
    if (pd >= math.min(radius1, radius2)) then
        print("Big circle wholly contains smaller circle")
    else
        print("Circles intersect with penetration distance", pd)
    end
end
```

See Also

None

Intersect.DistSqLineLine3d

Brief

Calculate the squared distance between two lines in 3D at their point of closest approach.

Definition

```
number [, number, number] Intersect.DistSqLineLine3d(Vector4 line1Point, Vector4 line1Dir, Vector4
line2Point, Vector4 line2Dir, boolean closest = false)
```

Arguments

line1Point - a point on line 1.line1Dir - the normalized direction of line 1.line2Point - a point on line 2.line2Dir - the normalized direction of line 2.closest - If true, return the distance parameters for each line at the point of closest approach.

Return Values

The squared distance between the two lines at their point of closest approach.

Description

Two lines in 3D will rarely intersect exactly, so it will probably be more valuable to know the distance between the lines at their closest point. This function will return the square of that distance, which can be compared to a value suitable to the application. Note that squared distances can quickly exceed the limits of a floating point number, so it may be best to translate the lines to the origin before calling this function.

Examples

```
line1Point = Vector4.Create(0, 1, 0)
line1Dir = Vector4.Create(1, 0, 0)
line2Point = Vector4.Create(0, 2, 0)
line2Dir = Vector4.Create(1, 0, 0)
dist, t1, t2 = Intersect.DistSqLineLine3d(line1Point, line1Dir, line2Point, line2Dir, true)
print("Closest approach distance is", dist, "and closet point of approach is", line1Point +
t1*line1Dir)
```

See Also

None

Intersect.LineAabb2d

Brief

Intersect a line with an AABB in 2D.

Definition

```
boolean, number, number Intersect.LineAabb2d(Vector4 linePoint, Vector4 lineDir, Vector4 boxMin,
Vector4 boxMax)
boolean, number, number Intersect.LineAabb2d(number linePointX, number linePointY, number lineDirX,
number lineDirY,
                              number boxMinX, number boxMinY, number boxMaxX, number boxMaxY)
```

Arguments

linePoint - a point on the line. lineDir - normalized line direction. boxMin - minimum extent of AABB. boxMax - maximum extent of AABB. linePointX - line point X coordinate. linePointY - line point Y coordinate. lineDirX - normalized line direction X coordinate. lineDirY - normalized line direction Y coordinate. boxMinX - minimum X extent of AABB. boxMinY - minimum Y extent of AABB. boxMaxX - maximum X extent of AABB. boxMaxY - maximum Y extent of AABB.

Return Values

True if the line intersects or false otherwise, and if true, the intersection distances on the line.

Description

Intersect a line with an AABB in 2D. If the line intersects the AABB, the intersection parameters are returned also. To compute an intersection point from these parameters, compute linePoint + t*lineDir, where t is the intersection distance returned from the function.

Examples

```
linePoint = Vector4.Create(0, 1)
lineDir = Vector4.Create(1, 0)
boxMin = Vector4.Create(1, 0.5)
boxMax = Vector4.Create(3, 1.5)
intersect, t1, t2 = Intersect.LineAabb2d(linePoint, lineDir, boxMin, boxMax)
if (intersect) then
    print("Intersection 1 at", t1)
    -- t2 can be same as t1 if intersects a corner
    if (t2 ~= t1) then
        print("Intersection 2 at", t2)
    end
end
```

See Also

- Intersect.RayAabb2d
- Intersect.SegmentAabb2d

Intersect.LineAabb3d

Brief

Intersect a line with an AABB in 3D.

Definition

```
boolean, number, number Intersect.LineAabb3d(Vector4 linePoint, Vector4 lineDir, Vector4 boxMin,
Vector4 boxMax)
```

Arguments

linePoint - a point on the line. lineDir - normalized line direction. boxMin - minimum extent of AABB. boxMax - maximum extent of AABB.

Return Values

True if the line intersects or false otherwise, and if true, the intersection distances on the line.

Description

Intersect a line with an AABB in 3D. If the line intersects the AABB, the intersection parameters are returned also. To compute an intersection point from these parameters, compute linePoint + t*lineDir, where t is the intersection distance returned from the function.

Examples

```
linePoint = Vector4.Create(0, 1, 0)
lineDir = Vector4.Create(1, 0, 0)
boxMin = Vector4.Create(1, 0.5, 0)
boxMax = Vector4.Create(3, 1.5, 0)
intersect, t1, t2 = Intersect.LineAabb3d(linePoint, lineDir, boxMin, boxMax)
if (intersect) then
    print("Intersection 1 at", t1)
    -- t2 can be same as t1 if intersects a corner
    if (t2 ~= t1) then
        print("Intersection 2 at", t2)
    end
end
```

See Also

- Intersect.RayAabb3d
- Intersect.SegmentAabb3d

Intersect.LineCircle2d

Brief

Intersect a line with a circle in 2D.

Definition

```
boolean [, number [, number]] Intersect.LineCircle2d(Vector4 linePoint, Vector4 lineDir, Vector4
center, number radius, boolean ip = false)
boolean [, number [, number]] Intersect.LineCircle2d(number linePointX, number linePointY, number
lineDirX, number lineDirY,
                                                     number centerX, number centerY, number radius,
boolean ip = false)
```

Arguments

linePoint - a point on the line (any point).lineDir - normalized direction vector of line (sign is unimportant).center - center point of circle.radius - circle radius.linePointX - X coordinate of a point on the line.linePointY - Y coordinate of a point on the line.lineDirX - X coordinate of normalized direction vector of line.lineDirY - Y coordinate of normalized direction vector of line.centerX - circle center X coordinate.centerY - circle center Y coordinate.ip - true to return intersection distances, false if not required.

Return Values

True if the line intersects the circle, or false otherwise. If ip is specified, additionally returns either 0, 1 or 2 intersection
distances, for the point(s) at which the line intersects the circle, if any.

Description

Intersect a line with a circle in 2D, and if requested return the distances along the line of the intersection point(s). If the intersection distance is not requested it will not be computed, which will be more efficient. Note that a line may intersect a circle at 0 (no intersection), 1 (tangential intersection) or 2 points (all other cases). The number of return values will reflect these cases. Note that a line is considered infinite and thus the "order" of intersection points is not considered. If you wish to interpret the intersection distances in terms of distance then you will need to perform this yourself.

Examples

```
linePoint = Vector4.Create(0, 1)
lineDir = Vector4.Create(1, 0)
center = Vector4.Create(3, 1)
intersect, t1, t2 = Intersect.LineCircle2d(linePoint, lineDir, center, 2, true)
if (intersect) then
    -- t1 must be valid
    print("Intersection 1 at", t1)
    -- t2 may be equal to t1 if line intersects tangentially
    if (t2 ~= t1) then
        print("Intersection 2 at", t2)
    end
end
```

See Also

- Intersect.RayCircle2d
- Intersect.SegmentCircle2d

Intersect.LineLine2d

Brief

Intersect two lines in 2D.

Definition

```
boolean [, number, number] Intersect.LineLine2d(Vector4 line1Point, Vector4 line1Dir, Vector4
line2Point, Vector4 line2Dir, boolean ip = false)
boolean [, number, number] Intersect.LineLine2d(number line1px, number line1py, number line1dx, number
line1dy,
                                                number line2px, number line2py, number line2dx, number
line2dy, boolean ip = false)
```

Arguments

line1Point - a point on line 1 (any point).line1Dir - normalized direction vector of line 1 (sign is unimportant).line2Point - a point on line 2 (any point).line2Dir - normalized direction vector of line 2 (sign is unimportant).line1PointX - X coordinate of a point on line 1 (any point).line1PointY - Y coordinate of a point on line 1 (any point).line1DirX - X coordinate of normalized direction vector of line 1 (sign is unimportant).line1DirY - Y coordinate of normalized direction vector of line 1 (sign is

unimportant).line2PointX - X coordinate of a point on line 2 (any point).line2PointY - Y coordinate of a point on line 2 (any point).line2DirX - X coordinate of normalized direction vector of line 2 (sign is unimportant).line2DirY - Y coordinate of normalized direction vector of line 2 (sign is unimportant).ip - true to return intersection distances, false if not required.

Return Values

True if the lines intersect, false otherwise. If ip is specified, additionally returns the intersection distance for both lines.

Description

Intersect two lines in 2D, and if requested return the intersection distance for both lines if they intersect. Requesting the intersection distances has no additional computation cost for this function. If the intersection point coordinates are required, it can be computed with linePoint + t*lineDir, where t is the intersection distance for that line.

Examples

```
line1Point = Vector4.Create(1, 1)
line1Dir = Vector4.Create(1, 0)
line2Point = Vector4.Create(3, 0)
line2Dir = Vector4.Create(0, 1)
intersect, t1, t2 = Intersect.LineLine2d(line1Point, line1Dir, line2Point, line2Dir, true)
if (intersect) then
    ip = line1Point + t1 * line1Dir     -- or line2Point + t2 * line2Dir
end
```

See Also

- Intersect.RayRay2d
- Intersect.RayLine2d
- Intersect.LineSegment2d
- Intersect.RaySegment2d
- Intersect.SegmentSegment2d

Intersect.LinePlane3d

Brief

Intersect a line with a plane in 3D.

Definition

```
IntersectType [ , number ] Intersect.LinePlane3d(Vector4 linePoint, Vector4 lineDir, Vector4 plane)
```

Arguments

linePoint - a point on the line.lineDir - direction vector of the line.plane - plane equation [normal, distance]

Return Values

If the line intersects the plane, IntersectType.Intersect is returned, if not, IntersectType.Disjoint is returned.
If the line lies in the plane, IntersectType.Overlap is returned. If the result is IntersectType.Intersect, the intersection parameter t is returned (intersection point = linePoint + t*lineDir). Other results have no additional return values.

Description

A line is directionless, so all queries are effectively "two-sided". If Intersect.Disjoint is returned, this indicates the line is parallel to the plane (perpendicular to the plane normal).

Examples

```
if (Intersect.LinePlane3d(linePoint, lineDir, plane) ~= IntersectType.Disjoint) then
    print("Line intersects plane")
end
```

See Also

- Intersect.RayPlane3d
- Intersect.SegmentPlane3d

Intersect.LineSegment2d

Brief

Intersect a line and a segment in 2D.

Definition

```
boolean [, number, number] Intersect.LineSegment2d(Vector4 linePoint, Vector4 lineDir, Vector4
segStart, Vector4 segVec, boolean ip = false)
boolean [, number, number] Intersect.LineSegment2d(number linePointX, number linePointY, number
lineDirX, number lineDirY,
                                                    number segStartX, number segStartY, number segVecX,
number segVecY, boolean ip = false)
```

Arguments

linePoint - a point on the line (any point). lineDir - normalized direction vector of line (sign is unimportant). segStart - first (start) point of segment. segVec - vector from first (start) point of segment to second (end) point of segment. If the segment points
are A and B, the vector is B-A. linePointX - X coordinate of a point on the line. linePointY - Y coordinate of a point on the line. lineDirX - X coordinate of normalized direction vector of line. lineDirY - Y coordinate of normalized direction vector of line. segStartX - X coordinate of segment start point. segStartY - Y coordinate of segment start point. segVecX - X coordinate of segment direction vector (see above). segVecY - Y coordinate of segment direction vector (see above). ip - true to return intersection distances, false if not required.

Return Values

True if the line and segment intersect, false otherwise. If ip is specified, additionally returns the intersection distance for the line and segment, in that order.

Description

Intersect a line and a segment in 2D, and if requested return the intersection distance for both line and segment if they intersect. Requesting the intersection distances has no additional computation cost for this function. If the intersection point coordinates are required, it can be computed with linePoint + t*lineDir, where t is the intersection distance for the line, or segStart + u*segVec, where u is the intersection distance for the segment.

Examples

```
linePoint = Vector4.Create(1, 1)
lineDir = Vector4.Create(1, 0)
segmentStart = Vector4.Create(3, 0)
segmentVec = Vector4.Create(0, 5)
intersect, t1, t2 = Intersect.LineSegment2d(linePoint, lineDir, segmentStart, segmentVec, true)
if (intersect) then
    ip = linePoint + t1 * lineDir       -- or segmentStart + t2 * segmentVec
end
```

See Also

- Intersect.RayRay2d
- Intersect.RayLine2d

- Intersect.RaySegment2d
- Intersect.LineLine2d
- Intersect.SegmentSegment2d

Intersect.LineSphere3d

Brief

Intersect a line with a sphere in 3D.

Definition

```
boolean [, number, number] Intersect.LineSphere3d(Vector4 linePoint, Vector4 lineDir, Vector4 center,
number radius, boolean ip = false)
```

Arguments

linePoint - any point on the line.lineDir - line direction (assumed to be normalized).center - center point of sphere.radius - radius of sphere.ip - if true, compute and return intersection points.

Return Values

True if the line intersects the sphere, false otherwise. If ip is specified, the two intersection distances are returned additionally.

Description

Intersect a line with a sphere in 3D, and if requested return the intersection distances along the line of the intersection points. If the intersection distance is not requested it will not be computed, which will be more efficient. If the intersection points in space are desired, then the points would be computed as linePoint + t*lineDir, with t the intersection distance. Note that a line can either not intersect, intersect at two points, or intersect at one point. In the latter case, both intersection distances will be the same. The ordering of the returned intersection values is not defined as a line is not directed, unlike a ray, and the sphere is considered hollow for the purposes of intersection.

Examples

```
linePoint = Vector4.Create(0, 1, 0)
lineDir = Vector4.Create(1, 0, 0)
center = Vector4.Create(3, 1, 0)
intersect, t1, t2 = Intersect.LineSphere3d(linePoint, lineDir, center, 2, true)
if (intersect) then
    -- t1 must be valid
    print("Intersection 1 at", t1)
    -- t2 be equal to t1 if line intersects tangentially
    if (t2 ~= t1) then
        print("Intersection 2 at", t2)
    end
end
```

See Also

- Intersect.SegmentSphere3d

Intersect.PlanePlane3d

Brief

Intersect two planes in 3D.

Definition

```
IntersectType Intersect.PlanePlane3d(Vector4 plane1, Vector4 plane2, Vector4 linePoint = nil, Vector4
lineDir = nil)
```

Arguments

plane1 - first plane [normal, distance].plane2 - second plane [normal, distance].linePoint - if specified, will be set to a point on the line of intersection of the planes if they intersect.lineDir - if specified, will be set to the normalized direction of the line of intersection of the planes if they intersect.

Return Values

IntersectType.Intersect if the planes are non-parallel. If co-planar, IntersectType.Overlap is returned, or IntersectType.Disjoint otherwise.

Description

Intersects two planes in 3D. Unless parallel, two planes will always intersect at a line. If the line of intersection is required (if it exists), pass Vector4 instances to linePoint and/or lineDir. Not requesting the line of intersection will be more efficient.

Examples

```
linePoint = Vector4.Create()
lineDir = Vector4.Create()
if (Intersect.PlanePlane3d(Vector4.Create(0, 1, 0, 3), Vector4.Create(1, 0, 0, 5), linePoint, lineDir)
== IntersectType.Intersect) then
    print("Lines intersect at line", linePoint, lineDir)
end
```

See Also

None

Intersect.RayAabb2d

Brief

Intersect a ray with an AABB in 2D.

Definition

```
boolean, number, number Intersect.RayAabb2d(Vector4 rayOrig, Vector4 rayDir, Vector4 boxMin, Vector4
boxMax)
boolean, number, number Intersect.RayAabb2d(number rayOrigX, number rayOrigY, number rayDirX, number
rayDirY,
                                            number boxMinX, number boxMinY, number boxMaxX, number
boxMaxY)
```

Arguments

rayOrig - ray origin.rayDir - normalized ray direction.boxMin - minimum extents of AABB.boxMax - maximum extents of AABB.rayOrigX - ray origin X coordinate.rayOrigY - ray origin Y coordinate.rayDirX - normalized ray direction X coordinate.rayDirY - normalized ray direction Y coordinate.boxMinX - minimum X extent of AABB.boxMinY - minimum Y extent of AABB.boxMaxX - maximum X extent of AABB.boxMaxY - maximum Y extent of AABB.

Return Values

True if the ray intersects or false otherwise, and if true, the intersection distance(s) along the ray.

Description

Intersect a ray with an axis-aligned bounding box (AABB) in 2D and return the intersection distance(s) if it intersects. The intersection distances will be returned in order of distance from the ray origin.

Examples

```
rayOrig = Vector4.Create(0, 1)
rayDir = Vector4.Create(1, 0)
boxMin = Vector4.Create(1, 0.5)
boxMax = Vector4.Create(3, 1.5)
intersect, t1, t2 = Intersect.RayAabb2d(rayOrig, rayDir, boxMin, boxMax)
if (intersect) then
    print("First intersection at", t1)
    -- t2 can be nil if ray starts inside box, or same as t1 if intersects a corner
    if (t2 ~= nil and t2 ~= t1) then
        print("Second intersection at", t2)
    end
end
```

See Also

- Intersect.LineAabb2d
- Intersect.SegmentAabb2d

Intersect.RayAabb3d

Brief

Intersect a ray (half-line) with an AABB in 3D.

Definition

```
boolean, number, number Intersect.RayAabb3d(Vector4 rayOrig, Vector4 rayDir, Vector4 boxMin, Vector4
boxMax)
```

Arguments

rayOrig - ray origin.rayDir - normalized ray direction.boxMin - minimum extents of AABB.boxMax - maximum extents of AABB.

Return Values

True if the ray intersects or false otherwise, and if true, the intersection distance(s) along the ray.

Description

Intersect a ray with an AABB in 3D and return the intersection distance(s) if it intersects.

Examples

```
rayOrig = Vector4.Create(0, 1, 0)
rayDir = Vector4.Create(1, 0, 0)
boxMin = Vector4.Create(1, 0.5, 0)
boxMax = Vector4.Create(3, 1.5, 0)
intersect, t1, t2 = Intersect.RayAabb3d(rayOrig, rayDir, boxMin, boxMax)
if (intersect) then
    print("First intersection at", t1)
    -- t2 can be nil if ray starts inside box, or same as t1 if intersects a corner
    if (t2 ~= nil and t2 ~= t1) then
        print("Second intersection at", t2)
    end
end
```

See Also

- Intersect.LineAabb3d

- Intersect.SegmentAabb3d

Intersect.RayCircle2d

Brief

Intersect a ray with a circle in 2D.

Definition

```
boolean [, number [, number]] Intersect.RayCircle2d(Vector4 rayOrig, Vector4 rayDir, Vector4 center,
number radius, boolean ip = false)
boolean [, number [, number]] Intersect.RayCircle2d(number rayOrigX, number rayOrigY, number rayDirX,
number rayDirY,
                                                    number centerX, number centerY, number radius,
boolean ip = false)
```

Arguments

rayOrig - ray origin.rayDir - normalized direction vector of ray.center - center point of circle.radius - circle radius.rayOrigX - ray origin X coordinate.rayOrigY - ray origin Y coordinate.rayDirX - X coordinate of normalized direction vector of ray.rayDirY - Y coordinate of normalized direction vector of ray.centerX - circle center X coordinate.centerY - circle center Y coordinate.ip - true to return intersection distances, false if not required.

Return Values

True if the ray intersects the circle, or false otherwise. If ip is specified, additionally returns either 0, 1 or 2 intersection distances, for the point(s) at which the ray intersects the circle, if any.

Description

Intersect a ray (half-line) with a circle in 2D, and if requested return the distances along the ray of the intersection point(s). If the intersection distance is not requested it will not be computed, which will be more efficient. Note that a ray may intersect a circle at 0 (no intersection), 1 (tangential intersection, or ray starting inside circle) or 2 points (all other cases). The number of return values will reflect these cases.

Examples

```
rayOrigin = Vector4.Create(0, 1)
rayDir = Vector4.Create(1, 0)
center = Vector4.Create(3, 1)
intersect, t1, t2 = Intersect.RayCircle2d(rayOrigin, rayDir, center, 2, true)
if (intersect) then
    -- t1 must be valid
    print("First intersection at", t1)
    -- t2 may be nil (ray starts inside circle), may also be equal to t1 if ray intersects
tangentially
    if (t2 ~= nil and t2 ~= t1) then
        print("Second intersection at", t2)
    end
end
```

See Also

- Intersect.LineCircle2d
- Intersect.SegmentCircle2d

Intersect.RayLine2d

Brief

Intersect a ray (half-line) and a line in 2D.

Definition

```
boolean [, number, number] Intersect.RayLine2d(Vector4 rayOrig, Vector4 rayDir, Vector4 linePoint,
Vector4 lineDir, boolean ip = false)
boolean [, number, number] Intersect.RayLine2d(number rayOrigX, number rayOrigY, number rayDirX,
number rayDirY,
                                                number linePointX, number linePointY, number lineDirX,
number lineDirY, boolean ip = false)
```

Arguments

rayOrig - ray origin.rayDir - normalized direction vector of ray.linePoint - a point on the line (any point).lineDir - normalized direction vector of line (sign is unimportant).rayOrigX - ray origin X coordinate.rayOrigY - ray origin Y coordinate.rayDirX - X coordinate of normalized direction vector of ray.rayDirY - Y coordinate of normalized direction vector of ray.linePointX - X coordinate of a point on the line.linePointY - Y coordinate of a point on the line.lineDirX - X coordinate of normalized direction vector of line.lineDirY - Y coordinate of normalized direction vector of line.ip - true to return intersection distances, false if not required.

Return Values

True if the ray and line intersect, false otherwise. If ip is specified, additionally returns the intersection distance for both the ray and line, in that order.

Description

Intersect a ray (half-line) and a line in 2D, and if requested return the intersection distance for both ray and line if they intersect. Requesting the intersection distances has no additional computation cost for this function. If the intersection point coordinates are required, it can be computed with either rayOrigin + t*rayDir, where t is the intersection distance for the ray, or linePoint + u*lineDir, where u is the intersection distance for the line.

Examples

```
rayOrigin = Vector4.Create(1, 1)
rayDir = Vector4.Create(1, 0)
linePoint = Vector4.Create(3, 0)
lineDir = Vector4.Create(0, 1)
intersect, t1, t2 = Intersect.RayLine2d(rayOrigin, rayDir, linePoint, lineDir, true)
if (intersect) then
    ip = rayOrigin + t1 * rayDir        -- or linePoint + t2 * lineDir
end
```

See Also

- Intersect.RaySegment2d
- Intersect.RayRay2d
- Intersect.LineSegment2d
- Intersect.LineLine2d
- Intersect.SegmentSegment2d

Intersect.RayPlane3d

Brief

Intersect a ray (half-line) with a plane in 3D.

Definition

```
IntersectType [ , number ] Intersect.RayPlane3d(Vector4 rayOrig, Vector4 rayDir, Vector4 plane, bool
twoSided = false)
```

Arguments

rayOrig - ray origin.rayDir - ray direction (assumed to be normalized).plane - plane equation [normal, distance]twoSided - if true, will check for intersection with the "back" side of the plane (ray origin in negative halfspace)

Return Values

If the ray intersects the plane, IntersectType.Intersect is returned, if not, IntersectType.Disjoint is returned.
If the ray lies in the plane, IntersectType.Overlap is returned. If the result is IntersectType.Intersect, the intersection distance along the ray is returned additionally. Other results have no additional return values.

Description

Intersect a ray (half-line) with a plane in 3D. The plane can be considered as either 1- or 2-sided using the twoSided parameter: 1-sided planes will not consider intersections from rays originating in the negative halfspace.

Examples

```
if (Intersect.RayPlane3d(rayOrig, rayDir, plane) ~= IntersectType.Disjoint) then
    print("Ray intersects plane")
end
```

See Also

- Intersect.LinePlane3d
- Intersect.SegmentPlane3d

Intersect.RayRay2d

Brief

Intersect two rays (half-lines) in 2D.

Definition

```
boolean [, number, number] Intersect.RayRay2d(Vector4 ray1Orig, Vector4 ray1Dir, Vector4 ray2Orig,
Vector4 ray2Dir, boolean ip = false)
boolean [, number, number] Intersect.RayRay2d(number ray1OrigX, number ray1OrigY, number ray1DirX,
number ray1DirY,
                                              number ray2OrigX, number ray2OrigY, number ray2DirX,
number ray2DirY, boolean ip = false)
```

Arguments

ray1Orig - ray 1 origin.ray1Dir - normalized direction vector of ray 1.ray2Orig - ray 2 origin.ray2Dir - normalized direction vector of ray 2.ray1OrigX - ray 1 origin X coordinate.ray1OrigY - ray 1 origin Y coordinate.ray1DirX - X coordinate of normalized direction vector of ray 1.ray1DirY - Y coordinate of normalized direction vector of ray 1.ray2OrigX - ray 2 origin X coordinate.ray2OrigY - ray 2 origin Y coordinate.ray2DirX - X coordinate of normalized direction vector of ray 2.ray2DirY - Y coordinate of normalized direction vector of ray 2.ip - true to return intersection distances, false if not required.

Return Values

True if the rays intersect or false otherwise. If ip is true, additionally returns the distance along the two rays to the intersection point.

Description

Intersect two rays (half-lines) in 2D, and if requested return the intersection distance for both rays if they intersect. Requesting the intersection distances has no additional computation cost for this function. If the intersection point coordinates are required, it can be computed with rayOrigin + t*rayDir, where t is the intersection distance for that ray.

Examples

```
ray1Origin = Vector4.Create(1, 1)
ray1Dir = Vector4.Create(1, 0)
ray2Origin = Vector4.Create(3, 0)
ray2Dir = Vector4.Create(0, 1)
intersect, t1, t2 = Intersect.RayRay2d(ray1Origin, ray1Dir, ray2Origin, ray2Dir, true)
if (intersect) then
    ip = ray1Origin + t1 * ray1Dir      -- or ray2Origin + t2 * ray2Dir
end
```

See Also

- Intersect.RaySegment2d
- Intersect.RayLine2d
- Intersect.LineSegment2d
- Intersect.LineLine2d
- Intersect.SegmentSegment2d

Intersect.RaySegment2d

Brief

Intersect a ray (half-line) and a segment in 2D.

Definition

```
boolean [, number, number] Intersect.RaySegment2d(Vector4 rayOrig, Vector4 rayDir, Vector4 segStart,
Vector4 segVec, boolean ip = false)
boolean [, number, number] Intersect.RaySegment2d(number rayOrigX, number rayOrigY, number rayDirX,
number rayDirY,
                                                  number segStartX, number segStartY, number segVecX,
number segVecY, boolean ip = false)
```

Arguments

rayOrig - ray origin.rayDir - normalized direction vector of ray.segStart - first (start) point of segment.segVec - vector from first (start) point of segment to second (end) point of segment. If the segment points
are A and B, the vector is B-A.rayOrigX - ray origin X coordinate.rayOrigY - ray origin Y coordinate.rayDirX - X coordinate of normalized direction vector of ray.rayDirY - Y coordinate of normalized direction vector of ray.segStartX - X coordinate of segment start point.segStartY - Y coordinate of segment start point.segVecX - X coordinate of segment direction vector (see above).segVecY - Y coordinate of segment direction vector (see above).ip - true to return intersection distances, false if not required.

Return Values

True if the ray and segment intersect, false otherwise. If ip is specified, additionally returns the intersection distance for both the ray and segment, in that order.

Description

Intersect a ray (half-line) and a segment in 2D, and if requested return the intersection distance for both ray and segment if they intersect. Requesting the intersection distances has no additional computation cost for this function. If the intersection point coordinates are required, it can be computed with either rayOrigin + t*rayDir, where t is the intersection distance for the ray, or segStart + u*segVec, where u is the intersection distance for the segment.

Examples

```
rayOrigin = Vector4.Create(1, 1)
rayDir = Vector4.Create(1, 0)
segmentStart = Vector4.Create(3, 0)
segmentVec = Vector4.Create(0, 5)
intersect, t1, t2 = Intersect.RaySegment2d(rayOrigin, rayDir, segmentStart, segmentVec, true)
if (intersect) then
    ip = rayOrigin + t1 * rayDir        -- or segmentStart + t2 * segmentVec
end
```

See Also

- Intersect.RayRay2d
- Intersect.RayLine2d
- Intersect.LineSegment2d
- Intersect.LineLine2d
- Intersect.SegmentSegment2d

Intersect.RaySphere3d

Brief

Intersect a ray with a sphere in 3D.

Definition

```
boolean [, number, number] Intersect.RaySphere3d(Vector4 rayOrig, Vector4 rayDir, Vector4 center,
number radius, boolean ip = false)
```

Arguments

rayOrig - ray origin.rayDir - ray direction (assumed to be normalized).center - center point of sphere.radius - radius of sphere.ip - if true, compute and return intersection distances.

Return Values

True if the ray intersects the sphere, false otherwise. If ip is specified, the intersection distances along the ray are returned. The first return value is the first intersection, and the second the final.

Description

Intersect a ray (half-line) with a sphere in 3D, and if requested return the distances along the ray of the intersection points. If "ip" is true the intersection distances will be returned after the boolean return value. If the intersection distance is not requested it will not be computed, which will be more efficient. If the intersection points in space are desired, they can be computed with rayOrig + t*rayDir, where t is the intersection distance. Note that a ray can either not intersect, intersect at two points, or intersect at one point. In the latter case, both intersection distances will be the same. The sphere is considered "solid", so a ray originating inside the sphere will have an initial intersection distance of 0.

Examples

```
rayOrigin = Vector4.Create(0, 1, 0)
rayDir = Vector4.Create(1, 0, 0)
center = Vector4.Create(3, 1, 0)
intersect, t1, t2 = Intersect.RaySphere3d(rayOrigin, rayDir, center, 2, true)
if (intersect) then
    -- t1 must be valid
    print("First intersection at", t1)
    -- t2 may be nil (ray starts inside sphere), may also be equal to t1 if ray intersects
tangentially
    if (t2 ~= nil and t2 ~= t1) then
        print("Second intersection at", t2)
    end
end
```

See Also

- Intersect.LineSphere3d
- Intersect.SegmentSphere3d

Intersect.SegmentAabb2d

Brief

Intersect a segment with an AABB in 2D.

Definition

```
boolean [, number [, number]] Intersect.SegmentAabb2d(Vector4 segStart, Vector4 segVec, Vector4
boxMin, Vector4 boxMax)
boolean [, number [, number]] Intersect.SegmentAabb2d(number segStartX, number segStartY, number
segVecX, number segVecY,
                                                      number boxMinX, number boxMinY, number boxMaxX,
number boxMaxY)
```

Arguments

segStart - segment start point. segVec - vector from segment start point to end point. boxMin - minimum extents of AABB. boxMax - maximum extents of AABB. segStartX - segment start point X coordinate. segStartY - segment start point Y coordinate. segVecX - segment vector X coordinate. segVecY - segment vector Y coordinate. boxMinX - minimum X extent of AABB. boxMinY - minimum Y extent of AABB. boxMaxX - maximum X extent of AABB. boxMaxY - maximum Y extent of AABB.

Return Values

True if the segment intersects or false otherwise, and if true, the intersection distance(s) along the segment.

Description

Intersect a segment with an AABB in 2D and return the intersection distance(s) if it intersects. To compute an intersection point from these parameters, compute segStart + t*segVec, where t is an intersection distance returned from the function.

Examples

```
segStart = Vector4.Create(0, 1)
segVec = Vector4.Create(6, 0)
boxMin = Vector4.Create(1, 0.5)
boxMax = Vector4.Create(3, 1.5)
intersect, t1, t2 = Intersect.SegmentAabb2d(segStart, segVec, boxMin, boxMax)
if (intersect) then
    print("First intersection at", t1)
    -- t2 can be nil (if segment starts inside box or doesn't cross box), or same as t1 if intersects
a corner
    if (t2 ~= nil and t2 ~= t1) then
        print("Second intersection at", t2)
    end
end
```

See Also

- Intersect.LineAabb2d
- Intersect.RayAabb2d

Intersect.SegmentAabb3d

Brief

Intersect a segment with an AABB in 3D.

Definition

```
boolean [, number [, number]] Intersect.SegmentAabb3d(Vector4 segStart, Vector4 segVec, Vector4
boxMin, Vector4 boxMax)
```

Arguments

segStart - segment start point. segVec - vector from segment start point to end point. boxMin - minimum extents of AABB. boxMax - maximum extents of AABB.

Return Values

True if the segment intersects or false otherwise, and if true, the intersection distance(s) along the segment.

Description

Intersect a segment with an AABB in 3D and return the intersection distance(s) if it intersects. To compute an intersection point from these parameters, compute segStart + t*segVec, where t is an intersection distance returned from the function.

Examples

```
segStart = Vector4.Create(0, 1, 0)
segVec = Vector4.Create(6, 0, 0)
boxMin = Vector4.Create(1, 0.5, 0)
boxMax = Vector4.Create(3, 1.5, 0)
intersect, t1, t2 = Intersect.SegmentAabb2d(segStart, segVec, boxMin, boxMax)
if (intersect) then
    print("First intersection at", t1)
    -- t2 can be nil (if segment starts inside box or doesn't cross box), or same as t1 if intersects
a corner
    if (t2 ~= nil and t2 ~= t1) then
        print("Second intersection at", t2)
    end
end
```

See Also

- Intersect.RayAabb3d
- Intersect.LineAabb3d

Intersect.SegmentCircle2d

Brief

Intersect a segment with a circle in 2D.

Definition

```
boolean [, number [, number]] Intersect.SegmentCircle2d(Vector4 segStart, Vector4 segVec, Vector4
center, number radius, boolean ip = false)
boolean [, number [, number]] Intersect.SegmentCircle2d(number segStartX, number segStartY, number
segVecX, number segVecY, number centerX, number centerY, number radius, boolean ip = false)
```

Arguments

segStart - first (start) point of segment.segVec - vector from first (start) point of segment to second (end) point of segment. If the segment points
are A and B, the vector is B-A.center - center point of circle.radius - radius of circle.segStartX - X coordinate of segment start.segStartY - Y coordinate of segment start.segVecX - X coordinate of segment vector (see segVec).segVecY - Y coordinate of segment vector (see segVec).centerX - X coordinate of circle center.centerY - Y coordinate of circle center.ip - true to return intersection distances, false if not required.

Return Values

True if the segment intersects the circle, or false otherwise. If ip is specified, additionally returns either 0, 1 or 2 intersection
distances, for the point(s) at which the segment intersects the circle, if any.

Description

Intersect a segment with a circle in 2D, and if requested return the distances along the segment of the intersection point(s). If the intersection distance is not requested it will not be computed, which will be more efficient. Note that a segment may intersect a circle at 0 (no intersection, including segment wholly inside the circle), 1 (tangential intersection, or segment straddles circle boundary) or 2 points (all other cases). The number of return values will reflect these cases.

Examples

```
segStart = Vector4.Create(0, 1)
segVec = Vector4.Create(6, 0)
center = Vector4.Create(3, 1)
intersect, t1, t2 = Intersect.SegmentCircle2d(segStart, segVec, center, 2, true)
if (intersect) then
    -- t1 must be valid
    print("First intersection at", t1)
    -- t2 may be nil (segment starts inside circle or doesn't cross circle), may also be equal to t1
if segment intersects tangentially
    if (t2 ~= nil and t2 ~= t1) then
        print("Second intersection at", t2)
    end
end
```

See Also

- Intersect.RayCircle2d
- Intersect.LineCircle2d

Intersect.SegmentPlane3d

Brief

Intersect a segment with a plane in 3D.

Definition

```
IntersectType [ , number ] Intersect.SegmentPlane3d(Vector4 segStart, Vector4 segVec, Vector4 plane,
bool twoSided = false)
```

Arguments

segStart - first segment point.segVec - vector from first segment point to second segment point.plane - plane equation [normal, distance]twoSided - if true, will check for intersection with the "back" side of the plane (segment start in negative halfspace)

Return Values

If the segment intersects the plane, IntersectType.Intersect is returned, if not, IntersectType.Disjoint is returned.
If the segment lies in the plane, IntersectType.Overlap is returned. If the result is IntersectType.Intersect, the intersection distance along the segment is returned additionally. Other results have no additional return values.

Description

Intersect a line segment with a plane in 3D. The plane can be considered as either 1- or 2-sided using the twoSided parameter: 1-sided planes will not consider intersections from segments originating in the negative halfspace.

Examples

```
if (Intersect.SegmentPlane3d(segStart, segEnd - segStart, plane) ~= IntersectType.Disjoint) then
    print("Segment intersects plane")
end
```

See Also

- Intersect.RayPlane3d
- Intersect.LinePlane3d

Intersect.SegmentSegment2d

Brief

Intersect two line segments in 2D.

Definition

```
boolean [, number, number] Intersect.SegmentSegment2d(Vector4 seg1Start, Vector4 seg1Vec, Vector4
seg2Start, Vector4 seg2Vec, boolean ip = false)
boolean [, number, number] Intersect.SegmentSegment2d(number seg1StartX, number seg1StartY, number
seg1VecX, number seg1VecY,
                                                      number seg2StartX, number seg2StartX, number
seg2VecX, number seg2VecY, boolean ip = false)
```

Arguments

seg1Start - first (start) point of segment 1.seg1Vec - vector from first (start) point of segment 1 to second (end) point of segment 1. If the segment points
are A and B, the vector is B-A.seg2Start - first (start) point of segment 2.seg2Vec - vector from first (start) point of segment 2 to second (end) point of segment 2. If the segment points
are A and B, the vector is B-A.seg1StartX - X coordinate of segment 1 start point.seg1StartY - Y coordinate of segment 1 start point.seg1VecX - X coordinate of segment 1 direction vector (see above).seg1VecY - Y coordinate of segment 1 direction vector (see above).seg2StartX - X coordinate of segment 2 start point.seg2StartY - Y coordinate of segment 2 start point.seg2VecX - X coordinate of segment 2 direction vector (see above).seg2VecY - Y coordinate of segment 2 direction vector (see above).ip - true to return intersection distances, false if not required.

Return Values

True if the segments intersect, false otherwise. If ip is specified, additionally returns the intersection distance for both segments.

Description

Intersect two segments in 2D, and if requested return the intersection distance for both segments if they intersect. Requesting the intersection distances has no additional computation cost for this function. If the intersection point coordinates are required, it can be computed with segStart + t*segVec, where t is the intersection distance for that segment.

Examples

```
seg1Start = Vector4.Create(1, 1)
seg1Vec = Vector4.Create(1, 0)
seg2Start = Vector4.Create(3, 0)
seg2Vec = Vector4.Create(0, 1)
intersect, t1, t2 = Intersect.SegmentSegment2d(seg1Start, seg1Vec, seg2Start, seg2Vec, true)
if (intersect) then
    ip = seg1Start + t1 * seg1Vec       -- or seg2Start + t2 * seg2Vec
end
```

See Also

- Intersect.RayRay2d
- Intersect.RayLine2d
- Intersect.LineSegment2d
- Intersect.LineLine2d
- Intersect.RaySegment2d

Intersect.SegmentSphere3d

Brief

Intersect a segment with a sphere in 3D.

Definition

```
boolean [, number, number] Intersect.SegmentSphere3d(Vector4 segStart, Vector4 segVec, Vector4 center,
number radius, boolean ip = false)
```

Arguments

segStart - start point of segment.segVec - segment vector.center - center point of sphere.radius - radius of sphere.ip - if true, compute and return intersection points.

Return Values

True if the segment intersects the sphere, false otherwise. If ip is specified, the intersection distance along the segment is returned as the second return value.

Description

Intersect a segment with a sphere in 3D, and if requested return the distance along the segment of the intersection point. If "ip" is true the intersection distance will be returned after the boolean return value. If the intersection distance is not requested it will not be computed, which will be more efficient. Of the two forms of this function, the latter (specifying normalized segment direction and length) is more efficient.

Examples

```
segStart = Vector4.Create(0, 1, 0)
segVec = Vector4.Create(6, 0, 0)
center = Vector4.Create(3, 1, 0)
intersect, t1, t2 = Intersect.SegmentSphere3d(segStart, segVec, center, 2, true)
if (intersect) then
    -- t1 must be valid
    print("First intersection at", t1)
    -- t2 may be nil (segment starts inside sphere or doesn't cross sphere), may also be equal to t1
if segment intersects tangentially
    if (t2 ~= nil and t2 ~= t1) then
        print("Second intersection at", t2)
    end
end
```

See Also

- Intersect.RaySphere3d
- Intersect.LineSphere3d

Intersect.SphereAabb3d

Brief

Intersect a sphere and an AABB in 3D.

Definition

```
boolean Intersect.SphereAabb3d(Vector4 center, number radius, Vector4 boxMin, Vector4 boxMax)
```

Arguments

center - Sphere center point.radius - Sphere radius.boxMin - Minimum extents of AABB.boxMax - Maximum extents of AABB.

Return Values

True if the sphere and AABB overlap, or false if not.

Description

Intersect a sphere and an AABB in 3D.

Examples

```
center = Vector4.Create(0, 1, 0)
radius = 2
boxMin = Vector4.Create(1, 0.5, 0)
boxMax = Vector4.Create(3, 1.5, 0)
if (Intersect.CircleSphere3d(center, radius, boxMin, boxMax)) then
    print("Intersection!")
end
```

See Also

None

Intersect.SpherePlane3d

Brief

Intersect a sphere with a plane in 3D.

Definition

```
number Intersect.SpherePlane3d(Vector4 center, number radius, Vector4 plane)
```

Arguments

center - center point of sphere.radius - radius of sphere.plane - plane equation [normal, distance]

Return Values

Penetration distance of sphere, positive if sphere intersects or lies behind plane, negative otherwise.

Description

Intersect a sphere with a plane in 3D and return the penetration distance, which is positive if the sphere intersects or lies behind the plane, or negative otherwise. If 0 <= penetration distance <= 2*radius, the sphere overlaps the plane.

Examples

```
local pen = Intersect.SpherePlane3d(center, radius, plane)
if (pen < 0) then
    print("Sphere lies wholly in the positive halfspace of the plane (no intersection)")
else if (pen <= 2*radius)
    print("Sphere overlaps the plane (0 <= penetration distance <= diameter)")
else
    print("Sphere lies wholly in the negative halfspace of the plane (no intersection)")
end
```

See Also

None

Intersect.SphereSphere3d

Brief

Intersect two spheres in 3D.

Definition

```
boolean [, number] Intersect.SphereSphere3d(Vector4 center1, number radius1, Vector4 center2, number
radius2, boolean pd = false)
```

Arguments

center1 - center point of sphere 1.radius1 - radius of sphere 1.center2 - center point of sphere 2.radius2 - radius of sphere 2.pd - return penetration distance.

Return Values

True if spheres intersect or false otherwise. If pd is true, additionally returns the penetration distance of the spheres.

Description

Intersect two spheres in 2D. If pd is true, also returns the penetration distance, which is >= 0 if the spheres intersect, or negative otherwise. If the penetration distance is greater than the smaller radius of the two, the small sphere lies wholly inside the large one. The penetration distance is defined along the vector between the spheres' centers.

Examples

```
local intersect, pd = Intersect.SphereSphere3d(center1, radius1, center2, radius2, true)
if (intersect) then
    if (pd >= math.min(radius1, radius2)) then
        print("Big sphere wholly contains smaller sphere")
    else
        print("Spheres intersect with penetration distance", pd)
    end
end
```

See Also

None

Json

Json.Decode

Brief

Decode a JSON structure (held in a memory container) into a table.

Definition

```
Json.Decode( MemoryContainer in, number offset = -1, size = -1 )
```

Arguments

in - The memory container holding the JSON-formatted block.offset - Offset into the container where the JSON structure begins.size - The size of the JSON structure.

Return Values

A table containing the decoded data structure, or a negative number on error.
-1 - Invalid argument.
-2 - Parse error.

Description

This function parses a JSON-formatted data structure held in the provided memory container, and returns a corresponding Lua table. Arrays will be treated as key-value pairs with the array index as the key (the base index will be 1).

Examples

```
container = MemoryContainer.Create( 1024 )
```

See Also

- Json.DecodeFromString
- Json.Encode

Json.DecodeFromString

Brief

Decode a JSON structure (held in a string) into a table.

Definition

```
Json.DecodeFromString( string in )
```

Arguments

in - A string holding a JSON-formatted data structure.

Return Values

A table containing the decoded data structure, or a negative number on error.
-1 - Invalid argument.
-2 - Parse error.

Description

This function parses a JSON-formatted data structure and returns a corresponding Lua table. Arrays will be treated as key-value pairs with the array index as the key (the base index will be 1).

Examples

```
decoded = Json.DecodeFromString(
"{\"session\":{\"@expires\":\"1314268686648\",\"@id\":\"3a50cfb2\",\"person\":{\"@id\":\"e8a41543\",\"dis
)
```

See Also

- Json.Decode
- Json.Encode

Json.Encode

Brief

Encode a table into JSON format, storing the results in the provided memory container.

Definition

```
number Json.Encode( MemoryContainer out, table in )
```

Arguments

out - The memory container to write the JSON-encoded data into.in - The table to be encoded.

Return Values

The number of bytes written to the output memory container, or a negative number on error.
-1 - Invalid argument.
-2 - Not enough space in the memory container.
-3 - Unable to encode table in the JSON format.

Description

Write out a JSON-formatted data structure representing the given table to the provided memory container. The structure of the table must conform to the JSON specification.

Examples

```
testtable =
{"session":{"service":{"@id":"bb88aea9","display_name":"Testing"},"person":{"@id":"e8a41543","display_nam
= MemoryContainer.Create( 1024 )
result = Json.Encode( container, testtable )
```

See Also

- Json.Decode
- Json.DecodeFromString

Light

Light.Create

Brief

Creates a new Light.

Definition

```
Light.Create( string lightType )
```

Arguments

lightType - The type of light. Can be one of the following:
"spot"
"point"

Return Values

A new Light instance.

Description

Creates a new Light. The light can be created as a point light or a spot light. Point lights apply light to the scene in a sphere centered around the light's position. Spot lights apply light to the scene in a frustum (pyramidal) shape. On creation, spot lights point down the positive Z axis. The orientation and position of a light can be modified by setting the position and rotation of the light's entity. The light's color defaults to white (red = 1.0, green = 1.0, blue = 1.0). By default, the light will start to attenuate from the position of the entity to which it is attached. Additionally, its contribution will attenuate to zero 2.5m from the position of the entity to which it is attached. The default attenuation power for the light is 2.0. If the light is a spot light, the default field of view angle on creation is 45 degrees.

Examples

```
-- Create a spot light
spotLight = Light.Create( "spot" )
Entity.SetLight(spotEntity, spotLight)

-- Create a point light
pointLight = Light.Create( "point" )
Entity.SetLight(pointEntity, pointLight)
```

See Also

- Entity.SetLight

Light.SetAttenuationEnd

Brief

Sets the distance to the end of the light's attenuation.

Definition

```
Light.SetAttenuationEnd( Light light, number attenuationEnd )
```

Arguments

light - The light to update.attenuationEnd - The distance to the end of the attenuation (in meters).

Return Values

None.

Description

Sets the distance to the end of the light's attenuation. At this distance, the light's contribution falls to zero.

Examples

```
spotLight = Light.Create( "spot" )
Light.SetAttenuationStart( spotLight, 2 )

-- Attenuate the light to zero brightness 5m from its position
Light.SetAttenuationEnd( spotLight, 5 )
```

See Also

- Light.Create
- Light.SetAttenuationPower
- Light.SetAttenuationStart

Light.SetAttenuationPower

Brief

Sets the light's attenuation power.

Definition

```
Light.SetAttenuationPower( Light light, number attenuationPower )
```

Arguments

light - The light to update.attenuationPower - The attenuation power.

Return Values

None.

Description

Sets the light's attenuation power. By default, a newly created light with have an attenuation power of 2.0. The power of the light affects the attenuation factor of the light as follows: attenFactor = (1.0 - saturate((distanceFromLight - attenStart) / (attenEnd - attenStart))) ^ power

Examples

```
pointLight = Light.Create( "point" )
Light.SetAttenuationStart( pointLight, 2 )
Light.SetAttenuationEnd( pointLight, 5 )
Light.SetAttenuationPower( pointLight, 4 )
```

See Also

- Light.Create
- Light.SetAttenuationEnd
- Light.SetAttenuationStart

Light.SetAttenuationStart

Brief

Sets the distance to the start of the light's attenuation.

Definition

```
Light.SetAttenuationStart( Light light, number attenuationStart )
```

Arguments

light - The light to update.attenuationStart - The distance to the start of the attenuation (in meters).

Return Values

None.

Description

Sets the distance to the start of the light's attenuation. From the position of the light to the attenuation start distance, the light's contribution is at its maximum. Its contribution then falls to zero between the attenuation start and end distances.

Examples

```
spotLight = Light.Create( "spot" )

-- Attenuate the light from full brightness 2m from its position
Light.SetAttenuationStart( spotLight, 2 )
Light.SetAttenuationEnd( spotLight, 5 )
```

See Also

- Light.Create
- Light.SetAttenuationPower
- Light.SetAttenuationEnd

Light.SetColor

Brief

Set the color of the light.

Definition

```
Light.SetColor( Light light, Vector4 rgb )
```

Arguments

light - The light to update.rgb - The color to set the light, stored in red, green, blue order.

Return Values

None.

Description

Set the color of the light. The light color is not clamped and can go above 1.0. The alpha component of the Vector4 is ignored and need not be specified.

Examples

```
spotLight = Light.Create( "spot" )
Light.SetColor( spotLight, Vector4.Create( 4.0, 3.0, 2.0 ) )
```

See Also

- Light.Create

Light.SetMask

Brief

Sets the mask texture for the light.

Definition

```
Light.SetMask( Light light, Texture mask )
```

Arguments

light - The light to update.mask - The mask texture for the light.

Return Values

None.

Description

Sets the mask texture for the light. Spot lights project a quad of light onto the scene. This quad can be masked with the supplied mask texture. The lighter the pixel in the mask, the more light will pass through it. For example, a white circle on a black background will project a circle onto the environment that is the color of the spot light. The mask for a spot light should be a DXT1 DDS texture. Point lights apply light all around them. Therefore, to mask the light of a point light, specify a cube map texture. Once again, this should be in DXT1 DDS format.

Examples

```
-- Create a masked spot light
spotLight = Light.Create( "spot" )
Light.SetMask( spotLight, Texture.Find( "circle_mask" )

-- Create a masked spot light
pointLight = Light.Create( "point" )
Light.SetMask( pointLight, Texture.Find( "cube_map_mask" )
```

See Also

- Light.Create
- Texture.Find

Light.SetShadowQuality

Brief

Set the quality of the dynamic shadows.

Definition

```
Light.SetShadowQuality( Light light, ShadowQuality shadowQuality )
```

Arguments

light - The light to update.shadowQuality - A ShadowQuality enum value:
ShadowQuality.Low
ShadowQuality.Medium
ShadowQuality.High

Return Values

None.

Description

Set the quality of dynamic shadows cast by this light. Higher quality shadows are more expensive to render. The default shadow quality is low.

Examples

```
spotLight = Light.Create( "spot" )
Light.SetSpotShadow( spotLight, true )
Light.SetShadowQuality( spotLight, ShadowQuality.Low )
```

See Also

- Light.Create
- Light.SetMask

Light.SetSpotFov

Brief

Sets the field of view of a spot light.

Definition

```
Light.SetSpotFov( Light light, number fov )
```

Arguments

light - The light to update.fov - The field of view of the spot in degrees.

Return Values

None.

Description

Sets the field of view of a spot light's frustum in degrees. This function is only valid for spot lights. The field of view of a newly created light defaults to 45 degrees.

Examples

```
spotLight = Light.Create( "spot" )
Light.SetSpotFov( spotLight, 90 )
```

See Also

- Light.Create

Light.SetSpotShadow

Brief

Set the spot light to casts shadows or not.

Definition

```
Light.SetSpotShadow( Light light, boolean useShadow )
```

Arguments

light - The light to update.useShadow - If true the spot light casts shadows of objects obstructing it and if false it casts no shadows.

Return Values

None.

Description

If true, the spot light casts shadows of objects obstructing it. If false, the spot light casts no shadows. By default, a newly created spot light casts no shadows.

Examples

```
spotLight = Light.Create( "spot" )
Light.SetSpotShadow( spotLight, true )
```

See Also

- Light.Create
- Light.SetMask

LinearCast

LinearCast.DebugDraw

Brief

Render a debug visual of the linear cast.

Definition

```
LinearCast.DebugDraw(LinearCast cast, Vector4 color)
```

Arguments

cast - cast to draw.color - color to render with.

Return Values

None.

Description

A debug representation of the linear cast will be drawn. Because the swept shape may be quite complex, this function will render the cast collision shape at the begin and end points of the cast and draw a line between the two points. This function requires the debug flag to be set on your object. The render will only persist for one frame, so you will need to call this function every frame while you wish to display the visualization.

Examples

```
cast:DebugDraw(Vector.Create(0, 1, 0, 1))       -- draw in green
```

See Also

- Collision.CreateLinearCast

LinearCast.GetEndPoint

Brief

Get the end point of a linear cast.

Definition

```
LinearCast.GetEndPoint(LinearCast cast, Vector4 point)
```

Arguments

cast - cast to query.point (out) - Vector4 to receive the point.

Return Values

None.

Description

Get the end point of a linear cast.

Examples

```
local v = Vector4.Create()
cast:GetStartPoint(v)
print("Cast start point is:", v)
```

See Also

- LinearCast.GetStartPoint
- LinearCast.GetHitPoint
- LinearCast.IsReady

LinearCast.GetHitCount

Brief

Retrieve the number of objects hit by the linear cast.

Definition

```
number LinearCast.GetHitCount(LinearCast cast)
```

Arguments

cast - the cast to query.

Return Values

The number of hits.

Description

The number of hits will be between 0 and the maxOverlaps value passed to the Collision.CreateLinearCast call.

Examples

```
cast = Collision.CreateLinearCast(ShapeType.Box, Vector4.Create(0.2, 0.8, 0.2), 5)      -- create cast
of box of half-extents [0.2, 0.8, 0.2], 5 max overlaps
rot = Matrix44.Create()
rot:SetRotationXyz(0, 45, 0)
cast:SetRotation(rot)
cast:Start(Vector4.Create(0,0,0), Vector4.Create(5,0,0))
while (not cast:IsReady()) do
    coroutine.yield()
end
local v = Vector4.Create()
for i = 1, cast:GetHitCount() do
    cast:GetHitPoint(i, v)
    print("Hit point:", v)
    cast:GetHitNormal(i, v)
    print("Hit normal:", v)
    local type = cast:GetHitType(i)
    print("Hit type:", type)
    print("Hit distance:", cast:GetHitDistance(i))
    local data = cast:GetHitData(i)
    if (type == HitType.Entity) then
        -- data is an entity
    elseif (type == HitType.Person) then
        -- data is a person
    elseif (type == HitType.Furniture) then
        -- data is a table (see GetHitData)
    elseif (type == HitType.Scene) then
        -- data is user ID of scene feature hit
    else
        -- data is nil - no further info available
    end
end
```

See Also

- LinearCast.GetHitPoint
- LinearCast.GetHitNormal
- LinearCast.GetHitType
- LinearCast.GetHitData
- LinearCast.GetHitDistance

LinearCast.GetHitData

Brief

Retrieve additional data about the object hit by a linear cast.

Definition

```
object LinearCast.GetHitData(LinearCast cast, number index)
```

Arguments

cast - the cast to query. index - the index of the hit to query.

Return Values

If a script entity owned by this object is hit, the Entity will be returned; if the entity is not
owned by the current object, nil will be returned. If the local avatar is
hit, the Person object representing the local person will be hit. If an item of furniture is hit,
a table of information about the furniture item is returned; see Furniture.GetItemData for info. For world collisions, the
surface
user ID will be returned, or 0 if there is none.

Description

Depending on the type of object hit (see LinearCast.GetHitType), this function will access the hit object or further information about the hit object. Note that if an entity owned by another object is hit, this function will return nil.

Examples

```
cast = Collision.CreateLinearCast(ShapeType.Box, Vector4.Create(0.2, 0.8, 0.2), 5)      -- create cast
of box of half-extents [0.2, 0.8, 0.2], 5 max overlaps
rot = Matrix44.Create()
rot:SetRotationXyz(0, 45, 0)
cast:SetRotation(rot)
cast:Start(Vector4.Create(0,0,0), Vector4.Create(5,0,0))
while (not cast:IsReady()) do
    coroutine.yield()
end
local v = Vector4.Create()
for i = 1, cast:GetHitCount() do
    cast:GetHitPoint(i, v)
    print("Hit point:", v)
    cast:GetHitNormal(i, v)
    print("Hit normal:", v)
    local type = cast:GetHitType(i)
    print("Hit type:", type)
    print("Hit distance:", cast:GetHitDistance(i))
    local data = cast:GetHitData(i)
    if (type == HitType.Entity) then
        -- data is an entity
    elseif (type == HitType.Person) then
        -- data is a person
    elseif (type == HitType.Furniture) then
        -- data is a table (see GetHitData)
    elseif (type == HitType.Scene) then
        -- data is user ID of scene feature hit
    else
        -- data is nil - no further info available
    end
end
```

See Also

- LinearCast.GetHitPoint
- LinearCast.GetHitNormal
- LinearCast.GetHitType
- LinearCast.GetHitData
- LinearCast.GetHitCount
- Furniture.GetItemData

LinearCast.GetHitDistance

Brief

Retrieve the distance along the linear cast at which a specified hit occurred.

Definition

```
number LinearCast.GetHitDistance(LinearCast cast, number index)
```

Arguments

cast - the cast to query. index - the index of the hit to query.

Return Values

The distance along the cast, as a proportion of the total distance of the cast [0 to 1].

Description

Returns the TOI (time of impact) along the linear cast path for a specified hit. The distance is in the range [0, 1], and can be converted to a point along the path by computing startPoint + (endPoint-startPoint)*distance. The contact will always appear on the side of the colliding object facing the cast, which may cause confusion if the cast is initially intersecting the object, as the contact distance will be 0 but the contact point may lie "behind" the origin of the cast. Scripts should be sure to check that the hit distance makes sense for their use.

Examples

```
cast = Collision.CreateLinearCast(ShapeType.Box, Vector4.Create(0.2, 0.8, 0.2), 5)      -- create cast
of box of half-extents [ 0.2, 0.8, 0.2 ], 5 max overlaps
rot = Matrix44.Create()
rot:SetRotationXyz(0, 45, 0)
cast:SetRotation(rot)
cast:Start(Vector4.Create(0,0,0), Vector4.Create(5,0,0))
while (not cast:IsReady()) do
    coroutine.yield()
end
local v = Vector4.Create()
for i = 1, cast:GetHitCount() do
    cast:GetHitPoint(i, v)
    print("Hit point:", v)
    cast:GetHitNormal(i, v)
    print("Hit normal:", v)
    local type = cast:GetHitType(i)
    print("Hit type:", type)
    print("Hit distance:", cast:GetHitDistance(i))
    local data = cast:GetHitData(i)
    if (type == HitType.Entity) then
        -- data is an entity
    elseif (type == HitType.Person) then
        -- data is a person
    elseif (type == HitType.Furniture) then
        -- data is a table (see GetHitData)
    elseif (type == HitType.Scene) then
        -- data is user ID of scene feature hit
    else
        -- data is nil - no further info available
    end
end
```

See Also

- LinearCast.GetHitPoint
- LinearCast.GetHitNormal

- LinearCast.GetHitType
- LinearCast.GetHitData
- LinearCast.GetHitCount

LinearCast.GetHitNormal

Brief

Retrieve the normal of the contact point generated for a specified hit.

Definition

```
number LinearCast.GetHitNormal(LinearCast cast, number index, Vector4 normal)
```

Arguments

cast - the cast to query. index - the index of the hit to query. normal (out) - the Vector4 to receive the hit normal.

Return Values

None.

Description

Retrieves the normal of the contact point generated for the specified hit. The contact normal will depend on the shapes of the cast and the colliding object. The contact will always appear on the side of the colliding object facing the cast, which may cause confusion if the cast is initially intersecting the object, as the contact point will be "behind" the start of the cast. Scripts should take care to examine hit normals and ensure they are suitable for the script's purpose.

Examples

```
cast = Collision.CreateLinearCast(ShapeType.Box, Vector4.Create(0.2, 0.8, 0.2), 5)      -- create cast
of box of half-extents [0.2, 0.8, 0.2], 5 max overlaps
rot = Matrix44.Create()
rot:SetRotationXyz(0, 45, 0)
cast:SetRotation(rot)
cast:Start(Vector4.Create(0,0,0), Vector4.Create(5,0,0))
while (not cast:IsReady()) do
    coroutine.yield()
end
local v = Vector4.Create()
for i = 1, cast:GetHitCount() do
    cast:GetHitPoint(i, v)
    print("Hit point:", v)
    cast:GetHitNormal(i, v)
    print("Hit normal:", v)
    local type = cast:GetHitType(i)
    print("Hit type:", type)
    print("Hit distance:", cast:GetHitDistance(i))
    local data = cast:GetHitData(i)
    if (type == HitType.Entity) then
        -- data is an entity
    elseif (type == HitType.Person) then
        -- data is a person
    elseif (type == HitType.Furniture) then
        -- data is a table (see GetHitData)
    elseif (type == HitType.Scene) then
        -- data is user ID of scene feature hit
    else
        -- data is nil - no further info available
    end
end
```

See Also

- LinearCast.GetHitPoint
- LinearCast.GetHitDistance
- LinearCast.GetHitType
- LinearCast.GetHitData
- LinearCast.GetHitCount

LinearCast.GetHitPoint

Brief

Retrieve the contact point generated for a specified hit.

Definition

```
LinearCast.GetHitPoint(LinearCast cast, number index, Vector4 point)
```

Arguments

cast - the cast to query.index - the index of the hit to query.point (out) - the Vector4 to receive the hit point.

Return Values

None.

Description

Retrieves the position of the contact point generated for the specified hit. The contact point will depend on the shapes of the cast and the colliding object. The contact will always appear on the side of the colliding object facing the cast, which may cause confusion if the cast is initially intersecting the object, as the contact point will be "behind" the start of the cast. Scripts should take care to examine hit points and ensure they are suitable for the script's purpose.

Examples

```
cast = Collision.CreateLinearCast(ShapeType.Box, Vector4.Create(0.2, 0.8, 0.2), 5)      -- create cast
of box of half-extents [0.2, 0.8, 0.2], 5 max overlaps
rot = Matrix44.Create()
rot:SetRotationXyz(0, 45, 0)
cast:SetRotation(rot)
cast:Start(Vector4.Create(0,0,0), Vector4.Create(5,0,0))
while (not cast:IsReady()) do
    coroutine.yield()
end
local v = Vector4.Create()
for i = 1, cast:GetHitCount() do
    cast:GetHitPoint(i, v)
    print("Hit point:", v)
    cast:GetHitNormal(i, v)
    print("Hit normal:", v)
    local type = cast:GetHitType(i)
    print("Hit type:", type)
    print("Hit distance:", cast:GetHitDistance(i))
    local data = cast:GetHitData(i)
    if (type == HitType.Entity) then
        -- data is an entity
    elseif (type == HitType.Person) then
        -- data is a person
    elseif (type == HitType.Furniture) then
        -- data is a table (see GetHitData)
    elseif (type == HitType.Scene) then
        -- data is user ID of scene feature hit
    else
        -- data is nil - no further info available
    end
end
```

See Also

- LinearCast.GetHitDistance
- LinearCast.GetHitNormal
- LinearCast.GetHitType
- LinearCast.GetHitData
- LinearCast.GetHitCount

LinearCast.GetHitType

Brief

Retrieve the type of the object hit by the cast.

Definition

```
HitType LinearCast.GetHitType(LinearCast cast, number index)
```

Arguments

cast - the cast to query. index - the index of the hit to query.

Return Values

The type of the colliding object.

Description

There are several types of colliding objects in most scenes, including script entities, the local avatar, furniture and the scene

itself. This function will indicate which type of object the cast has hit. For script entities, the local avatar and furniture, further information is available for the collided object; see LinearCast.GetHitData.

Examples

```
cast = Collision.CreateLinearCast(ShapeType.Box, Vector4.Create(0.2, 0.8, 0.2), 5)      -- create cast of box of half-extents [0.2, 0.8, 0.2], 5 max overlaps
rot = Matrix44.Create()
rot:SetRotationXyz(0, 45, 0)
cast:SetRotation(rot)
cast:Start(Vector4.Create(0,0,0), Vector4.Create(5,0,0))
while (not cast:IsReady()) do
    coroutine.yield()
end
local v = Vector4.Create()
for i = 1, cast:GetHitCount() do
    cast:GetHitPoint(i, v)
    print("Hit point:", v)
    cast:GetHitNormal(i, v)
    print("Hit normal:", v)
    local type = cast:GetHitType(i)
    print("Hit type:", type)
    print("Hit distance:", cast:GetHitDistance(i))
    local data = cast:GetHitData(i)
    if (type == HitType.Entity) then
        -- data is an entity
    elseif (type == HitType.Person) then
        -- data is a person
    elseif (type == HitType.Furniture) then
        -- data is a table (see GetHitData)
    elseif (type == HitType.Scene) then
        -- data is user ID of scene feature hit
    else
        -- data is nil - no further info available
    end
end
```

See Also

- LinearCast.GetHitPoint
- LinearCast.GetHitNormal
- LinearCast.GetHitDistance
- LinearCast.GetHitData
- LinearCast.GetHitCount

LinearCast.GetStartPoint

Brief

Get the start point of a linear cast.

Definition

```
LinearCast.GetStartPoint(LinearCast cast, Vector4 point)
```

Arguments

cast - cast to query.point (out) - Vector4 to receive the point.

Return Values

None.

Description

Get the start point of a linear cast.

Examples

```
local v = Vector4.Create()
cast:GetStartPoint(v)
print("Cast start point is:", v)
```

See Also

- LinearCast.GetEndPoint
- LinearCast.Start

LinearCast.IsReady

Brief

Query whether a linear cast is complete.

Definition

```
boolean LinearCast.IsReady(LinearCast cast)
```

Arguments

cast - cast to query.

Return Values

True if the cast is complete, false otherwise.

Description

If the cast has not been started, this function will return false. No result retrieval functions will work if this function returns false.

Examples

```
cast = Collision.CreateLinearCast(ShapeType.Box, Vector4.Create(0.2, 0.8, 0.2), 5)      -- create cast
of box of half-extents [0.2, 0.8, 0.2], 5 max overlaps
rot = Matrix44.Create()
rot:SetRotationXyz(0, 45, 0)
cast:SetRotation(rot)
cast:Start(Vector4.Create(0,0,0), Vector4.Create(5,0,0))
while (not cast:IsReady()) do
    coroutine.yield()
end
local v = Vector4.Create()
for i = 1, cast:GetHitCount() do
    cast:GetHitPoint(i, v)
    print("Hit point:", v)
    cast:GetHitNormal(i, v)
    print("Hit normal:", v)
    local type = cast:GetHitType(i)
    print("Hit type:", type)
    print("Hit distance:", cast:GetHitDistance(i))
    local data = cast:GetHitData(i)
    if (type == HitType.Entity) then
        -- data is an entity
    elseif (type == HitType.Person) then
        -- data is a person
    elseif (type == HitType.Furniture) then
        -- data is a table (see GetHitData)
    elseif (type == HitType.Scene) then
        -- data is user ID of scene feature hit
    else
        -- data is nil - no further info available
    end
end
```

See Also

- LinearCast.Start

LinearCast.SetCollisionLayersToHit

Brief

Specify which collision layers the cast should hit.

Definition

```
LinearCast.SetCollisionLayersToHit(LinearCast raycast, ...)
```

Arguments

cast - the cast to affect.... - collision layer(s) to hit, specified as separate arguments. Pass no args to clear collision filter.

Return Values

None.

Description

Specify which collision layers the cast should hit. The collision layers should be specified using the CollisionLayer enums. A call to this function will override all previous calls this function. If collision layers are specified, the user should not use the 'collisionLayer' parameter of Raycast.Start. If no collision filter is specified, the cast will revert to standard behaviour and the user may use the 'collisionLayer' parameter of LinearCast.Start.

Examples

```
cast:SetCollisionLayersToHit(CollisionLayer.Avatar, CollisionLayer.Entity)
-- to clear, cast:SetCollisionLayersToHit()
```

See Also

- LinearCast.Start

LinearCast.SetRotation

Brief

Set the rotation on the collision shape of a linear cast.

Definition

```
LinearCast.SetRotation(LinearCast cast, Matrix44 rotation)
```

Arguments

cast - cast to modify. rotation - rotation matrix to use (top-left 3x3 submatrix used only).

Return Values

None.

Description

The collision shape used for the sweep through the world can be rotated if required. The rotation is fixed, so the same orientation is used for the whole sweep (the shape cannot rotate during the sweep). Only the top-left 3x3 portion of the matrix is used. Scale is not supported but will not be validated against, so be sure to not introduce any scale.

Examples

```
cast = Collision.CreateLinearCast(ShapeType.Box, Vector4.Create(0.2, 0.8, 0.2), 5)      -- create cast
of box of half-extents [0.2, 0.8, 0.2], 5 max overlaps
rot = Matrix44.Create()
rot:SetRotationXyz(0, 45, 0)
cast:SetRotation(rot)
cast:Start(Vector4.Create(0,0,0), Vector4.Create(5,0,0))
while (not cast:IsReady()) do
    coroutine.yield()
end
local v = Vector4.Create()
for i = 1, cast:GetHitCount() do
    cast:GetHitPoint(i, v)
    print("Hit point:", v)
    cast:GetHitNormal(i, v)
    print("Hit normal:", v)
    local type = cast:GetHitType(i)
    print("Hit type:", type)
    print("Hit distance:", cast:GetHitDistance(i))
    local data = cast:GetHitData(i)
    if (type == HitType.Entity) then
        -- data is an entity
    elseif (type == HitType.Person) then
        -- data is a person
    elseif (type == HitType.Furniture) then
        -- data is a table (see GetHitData)
    elseif (type == HitType.Scene) then
        -- data is user ID of scene feature hit
    else
        -- data is nil - no further info available
    end
end
```

See Also

- LinearCast.Start

LinearCast.Start

Brief

Perform a linear cast between two points.

Definition

```
LinearCast.Start(LinearCast cast, Vector4 start, Vector4 end, CollisionLayer layer =
CollisionLayer.Cast, Phys3dWorld world = nil)
```

Arguments

start - Start point. end - End point. layer - Collision layer of the cast. Must specify nil here if using the collision filter. world - Phys3dWorld to sweep through, or nil for the default world.

Return Values

None.

Description

Starts the processing of a linear cast. The results will not be available until LinearCast.IsReady returns true. Note that to set the orientation of the cast shape, use SetRotation before calling Start. The orientation will be fixed over the cast.

Examples

```
cast = Collision.CreateLinearCast(ShapeType.Box, Vector4.Create(0.2, 0.8, 0.2), 5)      -- create cast
of box of half-extents [0.2, 0.8, 0.2], 5 max overlaps
rot = Matrix44.Create()
rot:SetRotationXyz(0, 45, 0)
cast:SetRotation(rot)
cast:Start(Vector4.Create(0,0,0), Vector4.Create(5,0,0))
while (not cast:IsReady()) do
    coroutine.yield()
end
local v = Vector4.Create()
for i = 1, cast:GetHitCount() do
    cast:GetHitPoint(i, v)
    print("Hit point:", v)
    cast:GetHitNormal(i, v)
    print("Hit normal:", v)
    local type = cast:GetHitType(i)
    print("Hit type:", type)
    print("Hit distance:", cast:GetHitDistance(i))
    local data = cast:GetHitData(i)
    if (type == HitType.Entity) then
        -- data is an entity
    elseif (type == HitType.Person) then
        -- data is a person
    elseif (type == HitType.Furniture) then
        -- data is a table (see GetHitData)
    elseif (type == HitType.Scene) then
        -- data is user ID of scene feature hit
    else
        -- data is nil - no further info available
    end
end
```

See Also

- Collision.CreateLinearCast
- LinearCast.SetRotation
- LinearCast.IsReady

LocalPlayer

LocalPlayer.AddRepertoire

Brief

Adds the specified repertoire resource from the object provided, to the local player. The action is replicated across the network, so the remote person will add the repertoire

Definition

```
Repertoire LocalPlayer.AddRepertoire( string label, number priority )
```

Arguments

label - the label of the repertoire to add.
priority - the priority over existing repertoires already added. The higher the number, the greater the priority. The default repertoire is added with a priority of zero.

Return Values

Reference to the repertoire resource that was added, or nil if failed to add.

Description

Examples

```
repertoire = LocalPlayer.AddRepertoire( "level1", 0 )
```

See Also

None

LocalPlayer.AddToPlayersMetList

Brief

Add a player to the XMB Players Met List.

Definition

```
LocalPlayer.AddToPlayersMetList( string playerName )
```

Arguments

personId - PSN ID of the player to add.

Return Values

None.

Description

The player won't be added if they're a Friend, or Blocked, or if they're the Local Player. Also, the player won't be added if they're not currently in the scene. It's advised this function is used sparingly and with care. E.g. if used to add other players in a Mini-game, please limit it to 16.

Examples

```
LocalPlayer.AddToPlayersMetList("MyPSNId")
```

See Also

None

LocalPlayer.DebugRemovePortable

Brief

Removes the local player's person's active object.

Definition

```
LocalPlayer.DebugRemoveActiveObject( string objectId )
```

Arguments

None

Return Values

None.

Description

Removes the local player's person's active object. For debug purposes only.

Examples

```
LocalPlayer.DebugRemoveActiveObject()
```

See Also

None

LocalPlayer.DebugSetPortable

Brief

Sets the local player's portable.

Definition

```
LocalPlayer.DebugSetPortable( string objectId )
```

Arguments

objectId - The object ID of the portable.

Return Values

None.

Description

Sets the local player's portable. For debug purposes only.

Examples

```
LocalPlayer.DebugSetPortable( "00000000-00000000-00000000-00000001" )
```

See Also

None

LocalPlayer.DoAction

Brief

Makes the local player's avatar perform the specified action.

Definition

```
LocalPlayer.DoAction( string actionName )
```

Arguments

actionName - Name of the action to be performed. It can be a custom animation loaded through LocalPlayer.AddRepertoire as well as one of the following standard actions:
"agree"
"beckon"
"bow"
"cheer"

"clap"
"disagree"
"frustration"
"laugh"
"point"
"wave"
"stand_up_from_seat"

Return Values

true if the action was recognised, false otherwise.

Description

Makes the local player's avatar perform the specified action.

Examples

```
LocalPlayer.DoAction( "wave" )
```

See Also

- LocalPlayer.IsDoingAction
- LocalPlayer.AddRepertoire

LocalPlayer.EnableAutoHiding

Brief

Enable/Disable the automatic hiding of the LocalPlayer's avatar when the camera gets too close.

Definition

```
LocalPlayer.EnableAutoHiding( boolean enable )
```

Arguments

enable - false to turn off auto-hiding, or true to re-enable it.

Return Values

None.

Description

The client automatically hides the avatar when the camera gets too close, for usability and aesthetic reasons. In certain circumstances it may be advantageous to disable this behavior, such as when creating a first person view.

Examples

```
LocalPlayer.EnableAutoHiding(false)
```

See Also

None

LocalPlayer.GetAccountRegion

Brief

Returns a string representing the PSN account region.

Definition

```
string LocalPlayer.GetAccountRegion()
```

Arguments

None

Return Values

A string representing the PSN account region.

Description

Returns a string representing the PSN account region.

Examples

```
if ( LocalPlayer.GetAccountRegion() == "en-GB" ) then
    print( "I am British!" )
end
```

See Also

- LocalPlayer.GetHomeRegion
- LocalPlayer.GetCurrentRegion

LocalPlayer.GetAccountTerritory

Brief

Returns a string representing the PSN account territory.

Definition

```
string LocalPlayer.GetAccountTerritory()
```

Arguments

None

Return Values

A string representing the PSN account territory.

Description

Returns a string representing the PSN account territory. Can be one of the following:

"SCEE" (Europe)

"SCEA" (North America)

"SCEJ" (Japan)

"SCEAsia" (Rest of Asia)

Examples

```
if ( LocalPlayer.GetAccountTerritory() == "SCEE" ) then
    print( "I am European!" )
end
```

See Also

- LocalPlayer.GetAccountRegion
- System.GetRegionsTerritory

LocalPlayer.GetAge

Brief

Returns the local player's age in years.

Definition

```
number LocalPlayer.GetAge()
```

Arguments

None

Return Values

The local player's age in years.

Description

Returns the local player's age in years. If there are any errors (such as being signed out), this will return -1.

Examples

```
print( 'I am ' .. LocalPlayer.GetAge() .. ' year(s) old.' )
```

See Also

None

LocalPlayer.GetAvatarRig

Brief

Returns the rig type (gender) of the LocalPlayer avatar.

Definition

```
RigType LocalPlayer.GetAvatarRig()
```

Arguments

None

Return Values

The rig type of the LocalPlayer avatar, as a number from the enumerated type RigType (see description).

Description

Returns the rig type (gender) of the LocalPlayer avatar, as one of the following values:
RigType.Undefined
RigType.Male
RigType.Female

Examples

```
if (LocalPlayer.GetAvatarRig() == RigType.Male) then
    LocalPlayer.SetRigComponentObjectId(kMaleHatObjectId)
else
    LocalPlayer.SetRigComponentObjectId(kFemaleHatObjectId)
end
```

See Also

- LocalPlayer.SetAvatarRig

LocalPlayer.GetCurrentLogin

Brief

Returns the date of the current login.

Definition

```
bool LocalPlayer.GetCurrentLogin( Time t )
```

Arguments

t - a time value to fill in.

Return Values

Returns true if the first session date is set and a valid Time object was provided.

Description

This will return the date/time of the current login. Used to determine how long the user has been online. Note that this value is not guaranteed to be tamper-proof as it is obtained from the user's hard disk; the value should be sanity-checked, and if its accuracy is critically important, should be verified with an external source.

Examples

```
t = Time.Create()
if ( LocalPlayer.GetCurrentLogin( t ) )
    print( 'Current login is ' .. Time.GetTime( t ) .. '.' )
```

See Also

- LocalPlayer.GetUserType
- LocalPlayer.GetPreviousUserType
- LocalPlayer.GetFirstLogin
- LocalPlayer.GetLastLogin
- LocalPlayer.GetLoginCount
- Time.Create

LocalPlayer.GetCurrentRegion

Brief

Returns a string representing the region of the space that the local player is currently in.

Definition

```
string LocalPlayer.GetCurrentRegion()
```

Arguments

None

Return Values

A string representing the current region the local player is in.

Description

Returns a string representing the region of the space that the local player is currently in. This is one of the PS Home supported regions.

Examples

```
if ( LocalPlayer.GetCurrentRegion() == "en-US" ) then
    print( "I am in an American space!" )
end
```

See Also

- LocalPlayer.GetHomeRegion
- LocalPlayer.GetAccountRegion

LocalPlayer.GetFirstLogin

Brief

Returns the first login date.

Definition

```
bool LocalPlayer.GetFirstLogin( Time t )
```

Arguments

t - a time value to fill in.

Return Values

Returns true if the first session date is set and a valid Time object was provided.

Description

This will return the date/time of the first login. Login tracking was added for the 1.55 release and therefore this date/time will be dated from this release. Clearing savedata will reset this date. Note that this value is not guaranteed to be tamper-proof as it is obtained from the user's hard disk; the value should be sanity-checked, and if its accuracy is critically important, should be verified with an external source.

Examples

```
t = Time.Create()
if ( LocalPlayer.GetFirstLogin( t ) )
    print( 'First login was ' .. Time.GetTime( t ) .. '.' )
```

See Also

- LocalPlayer.GetUserType
- LocalPlayer.GetPreviousUserType
- LocalPlayer.GetLastLogin
- LocalPlayer.GetCurrentLogin
- LocalPlayer.GetLoginCount
- Time.Create

LocalPlayer.GetFriendCount

Brief

Returns the number of friends.

Definition

```
number LocalPlayer.GetFriendCount()
```

Arguments

None

Return Values

The number of friends.

Description

Returns the number of friends that the local player has.

Examples

```
print( 'I have ' .. LocalPlayer.GetFriendCount() .. ' friends.' )
```

See Also

- LocalPlayer.GetFriends

LocalPlayer.GetFriends

Brief

Returns an array of friends as User objects.

Definition

```
User[] LocalPlayer.GetFriends()
```

Arguments

None

Return Values

An array of User objects

Description

Returns an array of friends as User objects.

Examples

```
friends = LocalPlayer.GetFriends()
for index, friend in ipairs( friends ) do
    print( User.GetName( friend ) )
end
```

See Also

- User.GetName

LocalPlayer.GetHomeAccountId

Brief

Fills in the unique accountId for the player

Definition

```
LocalPlayer.GetLoginCount()
```

Arguments

bigInt - Unique account id to set

Return Values

None.

Description

This 32-bit accountId is allocated by the Home game servers. It is unique and fixed for each account in the current environment. It will not be valid when running offline. The primary use for this function is to acquire a unique 32-bit ID that can be used in the RelocateToUniqueInstance function - allowing you to send the user to a space that is unique to their account.

Examples

```
local myAccountId = BigInt.Create(32)
LocalPlayer.GetHomeAccountId( myAccountId )
```

See Also

None

LocalPlayer.GetHomeRegion

Brief

Returns a string representing the local player's PS Home region.

Definition

```
string LocalPlayer.GetHomeRegion()
```

Arguments

None

Return Values

A string representing the PS Home region.

Description

Returns a string representing the local player's PS Home region. The account region is remapped into a supported PS Home region as PS Home does not support all regions.

Examples

```
if ( LocalPlayer.GetHomeRegion() == "en-GB" ) then
    print( "My PSN account region is mapped to British in PS Home!" )
end
```

See Also

- LocalPlayer.GetHomeRegion
- LocalPlayer.GetCurrentRegion

LocalPlayer.GetLastLogin

Brief

Returns the last login date.

Definition

```
bool LocalPlayer.GetLastLogin( Time t )
```

Arguments

t - a time value to fill in.

Return Values

Returns true if the last session date is set and a valid Time object was provided. Note that this value is not guaranteed to be tamper-proof as it is obtained from the user's hard disk; the value should be sanity-checked, and if its accuracy is critically important, should be verified with an external source.

Description

This will return the date/time of the last login.

Examples

```
t = Time.Create()
if ( LocalPlayer.GetLastLogin( t ) )
    print( 'Lost login was ' .. Time.GetTime( t ) .. '.' )
```

See Also

- LocalPlayer.GetUserType
- LocalPlayer.GetPreviousUserType
- LocalPlayer.GetFirstLogin
- LocalPlayer.GetCurrentLogin
- LocalPlayer.GetLoginCount
- Time.Create

LocalPlayer.GetLoginCount

Brief

Returns the total number of sessions for the user.

Definition

```
number LocalPlayer.GetLoginCount()
```

Arguments

None.

Return Values

number - The number of sessions

Description

The total number of previous sessions the user has logged in. A new user will have a value of 0. Login tracking was added for the 1.55 release and therefore this count will be since this release. Note that this value is not guaranteed to be tamper-proof as it is obtained from the user's hard disk; the value should be sanity-checked, and if its accuracy is critically important, should be verified with an external source. Clearing savedata will reset this count.

Examples

```
print( 'Number of logins: ' .. LocalPlayer.GetLoginCount() )
```

See Also

- LocalPlayer.GetPreviousUserType
- LocalPlayer.GetFirstLogin
- LocalPlayer.GetLastLogin
- LocalPlayer.GetCurrentLogin

LocalPlayer.GetPerson

Brief

Returns the person object of the local player.

Definition

```
Person LocalPlayer.GetPerson()
```

Arguments

None.

Return Values

Returns the person object of the local player.

Description

Returns the person object of the local player.

Examples

```
me = LocalPlayer.GetPerson()
```

See Also

None

LocalPlayer.GetPersonId

Brief

Returns the local player's person ID.

Definition

```
string LocalPlayer.GetPersonId()
```

Arguments

None

Return Values

The local player's person ID.

Description

Returns the local player's person ID. See Person.GetPersonId for more information on person IDs. This will return the same result as LocalPlayer.GetPerson():GetId().

Examples

```
print( 'My person ID is ' ... LocalPlayer.GetPersonId() )
```

See Also

LocalPlayer.GetPreviousUserType

Brief

Returns the user type from the previous login.

Definition

```
UserType LocalPlayer.GetPreviousUserType()
```

Arguments

None.

Return Values

One of the following user types :-
UserType.New - This is a new user
UserType.Returning - An old user who has returned
UserTYpe.Regular - A returning frequent user
UserType.Unknown - An unknown user type

Description

Returns the previous session's user type. This can be used to find out if the user type has just changed.

Examples

```
if ( LocalPlayer.GetUserType() ~= LocalPlayer.GetPrevousUserType() )
    print( 'User type has changed this session' )
```

See Also

- LocalPlayer.GetUserType
- LocalPlayer.GetFirstLogin
- LocalPlayer.GetLastLogin
- LocalPlayer.GetCurrentLogin
- LocalPlayer.GetLoginCount

LocalPlayer.GetUser

Brief

Returns the user object of the local player.

Definition

```
User LocalPlayer.GetUser()
```

Arguments

None

Return Values

Returns the user object of the local player.

Description

Returns the user object of the local player.

Examples

```
me = LocalPlayer.GetUser()
```

See Also

None

LocalPlayer.GetUserType

Brief

Returns the active user type.

Definition

```
UserType LocalPlayer.GetUserType()
```

Arguments

None.

Return Values

One of the following user types :-
UserType.New - This is a new user
UserType.Returning - An old user who has returned
UserTYpe.Regular - A returning frequent user
UserType.Unknown - An unknown user type

Description

Returns the current type of user logged in.

Examples

```
if ( LocalPlayer.GetUserType() == UserType.New )
    print( 'New user' )
```

See Also

- LocalPlayer.GetPreviousUserType
- LocalPlayer.GetFirstLogin
- LocalPlayer.GetLastLogin
- LocalPlayer.GetCurrentLogin
- LocalPlayer.GetLoginCount

LocalPlayer.IsChatAllowed

Brief

Returns true if the local player is not allowed to chat.

Definition

```
boolean LocalPlayer.IsChatAllowed()
```

Arguments

None.

Return Values

True if the local player is not allowed to chat, or false otherwise.

Description

If the local player is not allowed to chat, they're unable to send or receive any Text Chat, or use the Voice Comms in any way.

Examples

```
if (not LocalPlayer.IsChatAllowed())
    DoNormalUpdate()
end
```

See Also

None

LocalPlayer.IsDoingAction

Brief

Returns true if the Local Person is currently performing an action.

Definition

```
boolean LocalPlayer.IsDoingAction( boolean nonInterruptable = true )
```

Arguments

nonInterruptable - If true, only return true if the current action is not interruptable (defaults to true).

Return Values

Returns true if the Local Person is currently performing an action.

Description

Returns true if the Local Person is currently performing an action. An action is a non repeating animation, such as the action "wave".

Examples

```
print( LocalPlayer.IsDoingAction() )
```

See Also

- LocalPlayer.DoAction

LocalPlayer.IsInWardrobe

Brief

Returns true if the local player is currently located in the wardrobe.

Definition

```
boolean LocalPlayer.IsInWardrobe()
```

Arguments

None.

Return Values

True if the local player is in the wardrobe, or false otherwise.

Description

Returns true if the local player is currently located in the wardrobe.

Examples

```
if (not LocalPlayer.IsInWardrobe())
    DoNormalUpdate()
end
```

See Also

None

LocalPlayer.Relocate

Brief

Relocates the local player to the specified scene.

Definition

```
LocalPlayer.Relocate( string sceneName, number instanceParam = 0, number valueParam = 0, string
spawnFrom = nil )
```

Arguments

sceneName - The name of the scene as specified in the scene list.instanceParam - Controls the creation of different instances of the same scene (Optional - defaults to 0)valueParam - Specifies a value parameter that can be read by a script when the player has relocated to the new scene (Optional - defaults to -1)spawnFrom - [Optional] Specifies the scene name that will be used to choose the spawn location in the destination scene

Return Values

true if the destination scene name was recognised and the relocation process triggered, false if not

Description

Relocates the local player to the specified scene. Note that the relocation process is asynchronous, the user will continue to be considered present in the scene and joined to a minigame if applicable until the scene transition occurs. Therefore, expect that your script will continue to execute for some time after this call until the relocation has occurred. The following fixed scene names are also available:
BasicApartment - Harbour Studio
DEFAULT_APARTMENT - The user's default apartment

Examples

```
if PlayerHasUnlockedTheDoor() then
    LocalPlayer.Relocate( 'Secret_Room' )
end
```

See Also

- LocalPlayer.SetRelocateObject

LocalPlayer.RelocateToGroupInstance

Brief

Relocates the local player to a group instance of the specified scene.

Definition

```
LocalPlayer.RelocateToGroupInstance( string sceneName, number valueParam = nil )
```

Arguments

sceneName - The name of the scene as specified in the scene list.[optional] valueParam - A number that can be queried by the destination scene.

Return Values

True if the destination scene name was recognized and the relocation process triggered, false if not.

Description

Relocates the local player to a group instance of the specified scene. Note that the relocation process is asynchronous, the user will continue to be considered present in the scene and joined to a minigame if applicable until the scene transition occurs. Therefore, expect that your script will continue to execute for some time after this call until the relocation has occurred.

Examples

```
```

See Also

- LocalPlayer.Relocate
- LocalPlayer.RelocateToUniqueInstance

LocalPlayer.RelocateToUniqueInstance

Brief

Relocates the local player a unique instance of the specified scene.

Definition

```
LocalPlayer.RelocateToUniqueInstance( number instanceID, string sceneName )
```

Arguments

instanceID - Unique ID number of the scene (1 - 65535)sceneName - The name of the scene as specified in the scene list.[optional] valueParam - A number that can be queried by the destination scene

Return Values

true if the destination scene name was recognised and the relocation process triggered, false if not

Description

Relocates the local player to the specified scene. Note that the relocation process is asynchronous, the user will continue to be considered present in the scene and joined to a minigame if applicable until the scene transition occurs. Therefore, expect that your script will continue to execute for some time after this call until the relocation has occurred. RelocateToUniqueInstance differs from the standard Relocate method with an InstanceParam in 2 ways : 1) There will only ever be one instance of a scene with a particular UID. (If it is full, users will be denied access then returned to a new instance of their current location) 2) It is handled more efficiently on Home's backend (and hence is the preferred method for creating custom scene instances)

Examples

```
if PlayerHasSelectedPrivateSession() then
    SessionUID = 500
    LocalPlayer.RelocateToUniqueInstance( SessionUID, 'Game_Room' )
end
```

See Also

- LocalPlayer.Relocate

LocalPlayer.RemoveRepertoire

Brief

Removes the specified repertoire (by label) from the player

Definition

```
Repertoire LocalPlayer.RemoveRepertoire( string label )
```

Arguments

label - the label of the repertoire to remove.

Return Values

none

Description

Examples

```
LocalPlayer.RemoveRepertoire( "level1" )
```

See Also

None

LocalPlayer.RestoreAvatar

Brief

Returns the avatar (clothing items and rig) to the last user selection made within the wardrobe.

Definition

```
LocalPlayer.RestoreAvatar()
```

Arguments

None

Return Values

None.

Description

This method can only be used if the calling Lua environment has obtained the PersonLockType.Wardrobe lock, otherwise a lua error is thrown.

Examples

```
LocalPlayer.RestoreAvatar()
```

See Also

- LocalPlayer.SetAvatarRig
- LocalPlayer.SetRigComponentObjectId
- Person.Lock
- Person.Unlock

LocalPlayer.SetAvatarRig

Brief

Set the avatar rig type (gender).

Definition

```
boolean LocalPlayer.SetAvatarRig(RigType rigType)
```

Arguments

rigType - The rig type to change to (see description).

Return Values

True if able to change the rig (avatar is fully loaded and ready), otherwise false.

Description

Set the avatar rig type (gender) to one of the following values:
RigType.Male
RigType.Female Note that this function should be used with caution, and only with the user's consent.

Examples

```
LocalPlayer.SetAvatarRig(RigType.Male)
```

See Also

- LocalPlayer.GetAvatarRig
- Person.IsLoaded

LocalPlayer.SetClothingItems

Brief

Sets the clothing items on the player in one call

Definition

```
LocalPlayer.SetClothingItems( args )
```

Arguments

args - a table that contains the command arguments, the fields recognised are:
components [table] - a table of strings, each representing a component for the avatar to equip

Return Values

none

Description

the script must have the lock PersonLockType.Wardrobe, on the LocalPlayer person, before setting clothing items, otherwise a lua error be thrown.

Examples

```
local components = {}
table.insert(components, "00000000-00000000-00000000-01010034")
table.insert(components, "00000000-00000000-00000000-01020001")
table.insert(components, "00000000-00000000-00000000-01030009")
local setting = { components = components }
LocalPlayer.SetClothingItems( setting );
```

See Also

None

LocalPlayer.SetRelocateObject

Brief

Sets the optional objectId that can be used to script the relocation process from one scene to another

Definition

```
LocalPlayer.SetRelocateObject( string objectId, number mode = 0 )
```

Arguments

objectId - The Id of the object to usemode - The scripted relocation mode. Default 0 means the relocation script is activated once the download of the
destination scene has completed.
Mode 1 means the relocate script is started without first confirming the scene download (the scene_transition
object will need to handle the scene download)

Return Values

none

Description

Examples

```
LocalPlayer.SetRelocateObject( "12345678-12345678-12345678-12345678", 0 )
```

See Also

- LocalPlayer.Relocate

LocalPlayer.SetRigComponentObjectId

Brief

Sets the rig's component to the specified object.

Definition

```
LocalPlayer.SetRigComponentObjectId( string objectId, string componentType )
```

Arguments

objectId - The object ID for the component.componentType - The type of the rig component to set. Can be one of the following:
"Hair"
"Tors"
"Hands"
"Legs"
"Feet"
*NB This param is ignored now (1.35 and later) - works out what the componentType is when the object is loaded...

Return Values

true if successful, else false. (Will fail if the player is not ready, ie has not yet been allocated its memory slot)

Description

Sets the rig's component to the specified object. From 1.55, the script must have the lock PersonLockType.Wardrobe, on the LocalPlayer person, before setting clothing items, otherwise a lua error be thrown. The call may fail if the avatar is not in a state to accept new clothing components.

Examples

```
LocalPlayer.SetRigComponentObjectId( jacketObjectId, 'Tors' )
```

See Also

- Person.Lock

Material

Material.GetParam

Brief

Get a predefined parameter from the material

Definition

```
number|nil Material.GetParam(Material material, string name)
```

Arguments

material - The material to query.name - The parameter as it was named in maya.

Return Values

The value of the given param as it was set in maya, or nil if it doesn't exist.

Description

Gets the value of the given param as it was set in maya, or nil if it doesn't exist.

Examples

```
entity = Entity.Create( );
entity:SetModel( "mymodel" );
allMaterials = entity:FindMaterials( );
for i, material in ipairs( allMaterials ) do
    value = material:GetParam( "scrollSpeed" );
    print( "Scrollspeed is " .. tostring( value ) );
end
```

See Also

- Entity.FindMaterials

Material.SetAlphaRef

Brief

Set the alpha reference value of the material

Definition

```
Material.SetAlphaRef(Material material, number value)
```

Arguments

material - The material to set.value - The desired alpha reference value. should be between 0 and 1.

Return Values

None.

Description

Sets the alpha reference value of the material, for use with the "cutout" blend mode.

Examples

```
entity = Entity.Create( );
entity:SetModel( "mymodel" );
allMaterials = entity:FindMaterials( );
material = allMaterials[ 1 ];
material:SetBlendMode( BlendMode.Cutout );
material:SetAlphaRef( 0.1 );
```

See Also

- Entity.FindMaterials
- Material.SetBlendMode

Material.SetBlendMode

Brief

Set the blending mode of the material

Definition

```
Material.SetBlendMode(Material material, number mode)
```

Arguments

material - The material to set.mode - The desired blend mode. Valid values are:
BlendMode.Opaque
BlendMode.Cutout
BlendMode.Blend => SrcAlpha*Src + (1-SrcAlpha)*Dst
BlendMode.Add => Src + Dst
BlendMode.Multiply => Src * Dst
BlendMode.AlphaAdd => SrcAlpha*Src + Dst
BlendMode.BlendPremultiplied => Src + (1-SrcAlpha)*Dst

Return Values

None.

Description

Sets the blend mode of the material.

Examples

```
entity = Entity.Create( );
entity:SetModel( "mymodel" );
allMaterials = entity:FindMaterials( );
material = allMaterials[ 1 ];
material:SetBlendMode( BlendMode.Cutout );
material:SetAlphaRef( 0.1 );
```

See Also

- Entity.FindMaterials
- Material.SetAlphaRef

Material.SetDepthPrepass

Brief

Enable/Disable a depth prepass for this material

Definition

```
Material.SetDepthPrepass(Material material, boolean enable)
```

Arguments

material - The material to set.enable - Whether to enable or disable the depth prepass

Return Values

None.

Description

Enables (or disables) a depth prepass for use with semi-transparent materials (Removes internal pieces of a mesh).

Examples

```
entity = Entity.Create( );
entity:SetPosition( Vector4.Create( 0, 2, 0 ) );
entity:SetModel( "mymodel" );
allMaterials = entity:FindMaterials( );

material = allMaterials[ 1 ];
-- enable alpha blending
material:SetBlendMode( BlendMode.Blend );
-- enable depth prepass so the internals of the model are not visible.
material:SetDepthPrepass( true );
```

See Also

- Entity.FindMaterials
- Material.SetBlendMode

Material.SetMatrix

Brief

Set a matrix constant on the material

Definition

```
Material.SetMatrix(Material material, string name, Matrix44 value)
```

Arguments

material - The material to set. name - The uniform name. value - The value to set.

Return Values

None.

Description

Sets the given matrix shader parameter to the given value

Examples

```
rotation = 0.0;
change = 0.1;
entity = Entity.Create( );
entity:SetModel( "mymodel" );
allMaterials = entity:FindMaterials( );
material = allMaterials[ 3 ];
matrix = Matrix44.Create( );
function Update( )
    matrix:SetAxisRotation( 0, 0, 1, rotation, MATRIX_REPLACE );
    material:SetMatrix( "textureMatrix", matrix );
    rotation = rotation + change;
    if rotation > 360 or rotation < 0.0 then
        change = -change;
    end
end
```

See Also

- Entity.FindMaterials

Material.SetTexture

Brief

Set a texture on the material

Definition

```
Material.SetTexture(Material material, string name, string|Resource|Texture resource)
```

Arguments

material - The material to set. name - The uniform name. resource - The name of the texture resource, or the resource itself.

Return Values

None.

Description

Sets the given texture on the material

Examples

```
entity = Entity.Create( );
entity:SetModel( "mymodel" );
allMaterials = entity:FindMaterials( );
material = allMaterials[ 3 ];
material:SetTexture( "colourMap", "newSkin" );
```

See Also

Material.SetVector

Brief

Set a vector constant on the material

Definition

```
Material.SetVector(Material material, string name, Vector4 value)
```

Arguments

material - The material to set.name - The uniform name.value - The value to set.

Return Values

None.

Description

Sets the given vector shader parameter to the given value. The available vectors are as follows:

Name	Format	Applies to
tintClr	r,g,b,a	All materials with tint enabled
emissiveClr	r,g,b,a	All materials with emissive enabled
blendParams	offset,ramp	Blend materials
fresnelParams	power, bias, scale	Glass materials
waterColour	r,g,b	Water materials

Examples

```
colour = 1.0;
change = 0.01;
entity = Entity.Create( );
entity:SetModel( "gun" );
allMaterials = entity:FindMaterials( "fabric" );
for i, material in ipairs( allMaterials ) do
    material:SetBlendMode( BlendMode.Blend );
end

function Update( )
    for i, material in ipairs( allMaterials ) do
        --change the output colour
        material:SetVector( "tintClr",  Vector4.Create( colour, colour, colour, 1 ) );
    end
    colour = colour + change;
    if colour > 1.5 or colour < 0.5 then
        change = -change;
    end
end
```

See Also

- Entity.FindMaterials

Matrix44

Matrix44.Add

Brief

Performs a matrix addition with source1 and source2 storing the result in destination.

Definition

```
Matrix44.Add( Matrix44 destination, Matrix44 source1, Matrix44 source2 )
```

Arguments

destination - Matrix to receive the result. source1 - The first matrix argument for the addition. source2 - The second matrix argument for the addition.

Return Values

None.

Description

Performs a matrix addition with source1 and source2 storing the result in destination. Equivalent to destination = source1 + source2, but the operator syntax will allocate the destination automatically.

Examples

```
src1 = Matrix44.Create()
src2 = Matrix44.Create()
dst  = Matrix44.Create()
Matrix44.Add( dst, src1, src2 )
```

See Also

- Matrix44.Subtract

Matrix44.Copy

Brief

Copies the values of source and stores them in the destination Matrix44.

Definition

```
Matrix44.Copy( Matrix44 destination, Matrix44 source )
```

Arguments

destination - The matrix to receive the copy.source - The matrix to be copied.

Return Values

None

Description

Copies the values of source and stores them in the destination Matrix44. This is used to copy the values as opposed to pointing to the same matrix as an assignment would do. Use copy as mat2 = mat1 does not copy values but instead makes mat2 point to mat1.

Examples

```
mat1 = Matrix44.Create()
mat2 = Matrix44.Create()
Matrix44.Copy( mat2, mat1 )
```

See Also

None

Matrix44.Create

Brief

Creates a new Matrix44 object with the specified values.

Definition

```
Matrix44 Matrix44.Create( Vector4 x = [1,0,0,0], Vector4 y = [0,1,0,0], Vector4 z = [0,0,1,0], Vector4
w = [0,0,0,1] )
Matrix44 Matrix44.Create( Matrix44 matrix )
Matrix44 Matrix44.Create( Quaternion rotation, Vector4 translation = [0,0,0] )
```

Arguments

x - The first row of the matrix.y - The second row of the matrix.z - The third row of the matrix.w - The fourth row of the matrix.matrix - The matrix to copy.rotation - The quaternion representing the rotation part of the transform.translation - The vector representing the translation part of the transform.

Return Values

A new Matrix44 instance.

Description

Creates a new Matrix44 object with the specified values. With no parameters specified, it is set to the identity matrix.

Examples

```
mat1 = Matrix44.Create()
vec = Vector4.Create( 1, 2, 3, 4 )
mat2 = Matrix44.Create( vec, vec, vec, vec )
mat3 = Matrix44.Create( mat2 )
mat4 = Matrix44.Create( Quaternion.Create() )
```

See Also

None

Matrix44.GetRotationXyz

Brief

Gets the rotation about the x, y, and z-axis in degrees.

Definition

```
Vector4 Matrix44.GetRotationXyz( Matrix44 matrix )
Matrix44.GetRotationXyz( Matrix44 matrix, Vector4 outVec )
```

Arguments

matrix - The matrix to query.outVec - The Vector4 to store the result to.

Return Values

A vector containing the x, y, z rotation angles.

Description

Gets the rotation about the x, y, and z-axis in degrees using a matrix to Euler angle formula. The rotation order is x, y, z. If outVec is specified it stores the result in there instead of returning a new Vector4.

Examples

```
matrix = Matrix44.Create()
rotation = Matrix44.GetRotationXyz( matrix )
```

See Also

- Entity.SetRotationXyz

Matrix44.GetRow

Brief

Returns the specified row of the matrix.

Definition

```
Vector4 Matrix44.GetRow( Matrix44 matrix, number row )
Matrix44.GetRow( Matrix44 matrix, number row, Vector4 outVec )
```

Arguments

matrix - The matrix to be queried.row - The row to return, ranging from 1 to 4.outVec - The Vector4 to store the result to.

Return Values

A vector representing the specified row of the matrix.

Description

Returns the specified row of the matrix. The Vector4 is a copy of the row so changing its values will not affect the matrix. If outVec is specified it stores the result in there instead of returning a new Vector4.

Examples

```
mat = Matrix44.Create()
print( Matrix44.GetRow( mat, 1 ) )
```

See Also

- Matrix44.SetRow

Matrix44.Invert

Brief

Inverts the source matrix storing it in destination.

Definition

```
Matrix44.Invert( Matrix44 destination, Matrix44 source )
```

Arguments

destination - The matrix to receive the inverted matrix. Can be the source matrix.source - The matrix to invert.

Return Values

None

Description

Inverts the source matrix storing it in destination.

Examples

```
matrix = Matrix44.Create()
Matrix44.Invert( matrix, matrix ) -- Self inverse.
```

See Also

None

Matrix44.InvertAffine

Brief

Inverts the source matrix, which is assumed to be an affine transform, storing it in destination.

Definition

```
Matrix44.InvertAffine( Matrix44 destination, Matrix44 source )
```

Arguments

destination - The matrix to receive the inverted matrix. Can be the source matrix.source - The affine transform matrix to invert.

Return Values

None.

Description

For matrices representing an affine transform (that is, containing only scale, rotation and translation), this function will be faster than the general purpose Invert function. If your transform is only composed of rotation and translation (no scale), use the InvertAffineOrthogonal function, which is faster again.

Examples

```
matrix = Matrix44.Create()
Matrix44.InvertAffine( matrix, matrix ) -- Self inverse.
```

See Also

- Matrix44.Invert
- Matrix44.InvertAffineOrthogonal

Matrix44.InvertAffineOrthogonal

Brief

Inverts the source matrix, which is assumed to be a transform containing only rotation and translation, storing it in destination.

Definition

```
Matrix44.InvertAffineOrthogonal( Matrix44 destination, Matrix44 source )
```

Arguments

destination - The matrix to receive the inverted matrix. Can be the source matrix.source - The transform matrix to invert.

Return Values

None.

Description

If your matrix contains only rotation and translation (no scale), this function will calculate the inverse using an optimized method that is faster than both the general purpose Invert function and the InvertAffine function.

Examples

```
matrix = Matrix44.Create()
Matrix44.InvertAffineOrthogonal( matrix, matrix ) -- Self inverse.
```

See Also

- Matrix44.InvertAffine
- Matrix44.Invert

Matrix44.IsEqual

Brief

Tests whether the specified matrices are equal.

Definition

```
boolean Matrix44.IsEqual( Matrix44 m1, Matrix44 m2, number tolerance = 0 )
```

Arguments

m1 - The first matrix to test for equality.m2 - The second matrix to test for equality.tolerance - The amount of error to deviate from the other matrix.

Return Values

true if it the matrices are equal and false otherwise.

Description

Returns true if it the matrices are equal and false otherwise.

Examples

```
mat1 = Matrix44.Create()
mat2 = Matrix44.Create()
print( Matrix44.IsEqual( mat1, mat2 ) )
```

See Also

None

Matrix44.IsIdentity

Brief

Tests whether the specified matrix is an identity matrix.

Definition

```
boolean Matrix44.IsIdentity( Matrix44 matrix, number tolerance = 0 )
```

Arguments

matrix - The matrix to query.tolerance - The amount of error to deviate from the identity matrix.

Return Values

true if it is an identity matrix and false otherwise.

Description

Returns true if it is an identity matrix and false otherwise.

Examples

```
matrix = Matrix44.Create()
print( Matrix44.IsIdentity( matrix ) )
```

See Also

None

Matrix44.Multiply

Brief

Performs a matrix multiply with source1 and source2 storing the result in destination.

Definition

```
Matrix44.Multiply( Matrix44 destination, Matrix44 source1, Matrix44 source2 )
Matrix44.Multiply( Matrix44 destination, number scalar, Matrix44 matrix )
Matrix44.Multiply( Matrix44 destination, Matrix44 matrix, number scalar )
```

Arguments

source1 - The first matrix argument for the multiplication.source2 - The second matrix argument for the multiplication.scalar - Scalar to multiply.matrix - Matrix to multiply.

Return Values

None

Description

Performs a matrix multiply with source1 and source2, or a scalar multiply with scalar and matrix, storing the result in destination. Equivalent to destination = source1 * source2 or destination = scalar * matrix, but the operator syntax will allocate the destination automatically.

Examples

```
src1 = Matrix44.Create()
src2 = Matrix44.Create()
dst  = Matrix44.Create()
Matrix44.Multiply( dst, src1, src2 )
Matrix44.Multiply( dst, src1, 0.5 )
Matrix44.Multiply( dst, 2, src2 )
```

See Also

None

Matrix44.Orthonormalize

Brief

Orthonormalizes the source matrix storing it in destination.

Definition

```
Matrix44.Orthonormalize( Matrix44 destination, Matrix44 source, string axisPriority )
```

Arguments

destination - The matrix to receive the orthonormalized matrix. Can be the source matrix.source - The matrix to orthonormalize.axisPriority - A string representing the axis priority of the operation, eg "yxz".

Return Values

None

Description

Orthonormalizes the source matrix storing it in destination. The first letter in the axis priority string specifies the axis

whose direction will not be changed due to the orthogonalisation. The last specifies the axis whose direction will change the most due to the orthogonalisation. Note that if it is only necessary to remove scale, it is enough to normalise the three basis vectors rather than calling this function.

Examples

```
matrix = Matrix44.Create()
Matrix44.Orthonormalize( matrix, matrix ) -- Self orthonormalize.
```

See Also

None

Matrix44.SetAxisRotation

Brief

Sets up a rotation matrix around a specified axis and angle.

Definition

```
Matrix44.SetAxisRotation( Matrix44 matrix, Vector4 rotationXyz, number angle = 0, number setType =
MATRIX_POSTMULTIPLY )
Matrix44.SetAxisRotation( Matrix44 matrix, number axisX = 1, number axisY = 1, number axisZ = 1,
number angle = 0, number setType = MATRIX_POSTMULTIPLY )
```

Arguments

matrix - The matrix to be rotated. rotationXyz - A vector containing the axis to rotate about. angle - The angle in degrees to rotate around the axis. axisX - The x element of the axis. axisY - The y element of the axis. axisZ - The z element of the axis. setType - How to set the matrix. Must be either MATRIX_POSTMULTIPLY, MATRIX_PREMULTIPLY, or MATRIX_REPLACE.

Return Values

None

Description

Sets up a rotation matrix around a specified axis and angle. Depending on the set type it either post multiplies the current matrix by the rotation matrix (matrix = matrix * rotation), pre multiplies the current matrix by the rotation matrix (matrix = rotation * matrix) or simply sets the current matrix to the rotation matrix (matrix = rotation). Non normalized axes are automatically normalized.

Examples

```
matrix = Matrix44.Create()
Matrix44.SetAxisRotation( matrix, 0.5, 0.5, 0, 90, MATRIX_REPLACE )
```

See Also

None

Matrix44.SetIdentity

Brief

Sets the matrix to the identity matrix.

Definition

```
Matrix44.SetIdentity( Matrix44 matrix )
```

Arguments

matrix - The matrix to set to identity.

Return Values

None

Description

Sets the matrix to the identity matrix.

Examples

```
matrix = Matrix44.Create()
Matrix44.SetIdentity( matrix )
```

See Also

- Matrix44.IsIdentity

Matrix44.SetRotationQuat

Brief

Sets up a rotation matrix using the specified quaternion.

Definition

```
Matrix44.SetRotationQuat( Matrix44 matrix, Quaternion quat, number setType = MATRIX_POSTMULTIPLY )
```

Arguments

matrix - The matrix to be rotated.quat - The quaternion to convert.setType - How to set the matrix. Must be either MATRIX_POSTMULTIPLY, MATRIX_PREMULTIPLY, or MATRIX_REPLACE.

Return Values

None

Description

Sets up a rotation matrix using a specified quaternion. Depending on the set type it either post multiplies the current matrix by the rotation matrix (matrix = matrix * rotation), pre multiplies the current matrix by the rotation matrix (matrix = rotation * matrix) or simply sets the current matrix to the rotation matrix (matrix = rotation).

Examples

```
matrix = Matrix44.Create()
quat = Quaternion.Create()
Matrix44.SetRotationQuat( matrix, quat, MATRIX_REPLACE )
```

See Also

None

Matrix44.SetRotationXyz

Brief

Sets up a rotation matrix from x, y, and z euler angles specified in degrees.

Definition

```
Matrix44.SetRotationXyz( Matrix44 matrix, Vector4 rotationXyz, number setType = MATRIX_POSTMULTIPLY )
Matrix44.SetRotationXyz( Matrix44 matrix, number rotationX = 1, number rotationY = 1, number rotationZ
= 1, number setType = MATRIX_POSTMULTIPLY )
```

Arguments

matrix - The matrix to be rotated.rotationXyz - A vector containing the x, y and z euler angles in degrees to set the rotation matrix to.rotationX - A number containing the x euler angle in degrees to set the rotation matrix to.rotationY - A number containing the x euler angle in degrees to set the rotation matrix to.rotationZ - A number containing the x euler angle in degrees to set the rotation matrix to.setType - How to set the matrix. Must be either MATRIX_POSTMULTIPLY, MATRIX_PREMULTIPLY, or MATRIX_REPLACE.

Return Values

None

Description

Sets up a rotation matrix from x, y, and z euler angles specified in degrees. Depending on the set type it either post multiplies the current matrix by the rotation matrix (matrix = matrix * rotation), pre multiplies the current matrix by the rotation matrix (matrix = rotation * matrix) or simply sets the current matrix to the rotation matrix (matrix = rotation). The rotation order is x, y, z.

Examples

```
matrix = Matrix44.Create()
Matrix44.SetRotationXyz( matrix, 45, 90, 180, MATRIX_REPLACE )
```

See Also

None

Matrix44.SetRow

Brief

Sets the specified row of the matrix.

Definition

```
Matrix44.SetRow( Matrix44 matrix, number row, Vector4 value )
Matrix44.SetRow( Matrix44 matrix, number row, number elem1, number elem2, number elem3, number elem4 )
```

Arguments

matrix - The matrix to be updated.row - The row to set, ranging from 1 to 4.value - A 4 dimensional vector that is copied into the first row of the matrix.elem1 - Value to set column 1 of the specified row of the matrix.elem2 - Value to set column 2 of the specified row of the matrix.elem3 - Value to set column 3 of the specified row of the matrix.elem4 - Value to set column 4 of the specified row of the matrix.

Return Values

None

Description

Sets the specified row of the matrix.

Examples

```
mat = Matrix44.Create()

-- Example 1: Set via vector
vec = Vector4.Create( 1, 2, 3, 4 )
Matrix44.SetRow( mat, 3, vec )

-- Example 2: Set via numbers
Matrix44.SetRow( mat, 3, 1, 2, 3, 4 )
```

See Also

- Matrix44.GetRow

Matrix44.SetScale

Brief

Sets up a scale matrix.

Definition

```
Matrix44.SetScale( Matrix44 matrix, Vector4 scaleXyzw, number setType = MATRIX_POSTMULTIPLY )
Matrix44.SetScale( Matrix44 matrix, number scaleX = 1, number scaleY = 1, number scaleZ = 1, number
scaleW = 1, number setType = MATRIX_POSTMULTIPLY )
```

Arguments

matrix - The matrix to be scaled.scaleXyzw - A vector containing the x, y, z and w scale to set the matrix to.scaleX - A number containing the x scale to set the matrix to.scaleY - A number containing the y scale to set the matrix to.scaleZ - A number containing the z scale to set the matrix to.scaleW - A number containing the w scale to set the matrix to.setType - How to set the matrix. Must be either MATRIX_POSTMULTIPLY, MATRIX_PREMULTIPLY, or MATRIX_REPLACE.

Return Values

None

Description

Sets up a scale matrix and depending on the set type it either post multiplies the current matrix by the scale matrix (matrix = matrix * scale), pre multiplies the current matrix by the scale matrix (matrix = scale * matrix) or simply sets the current matrix to the scale matrix (matrix = scale).

Examples

```
matrix = Matrix44.Create()
Matrix44.SetScale( matrix, 4, 3, 2, 1, MATRIX_REPLACE )
```

See Also

None

Matrix44.SetTranslation

Brief

Sets up a translation matrix

Definition

```
Matrix44.SetTranslation( Matrix44 matrix, Vector4 translationXyz, number setType = MATRIX_POSTMULTIPLY
)
Matrix44.SetTranslation( Matrix44 matrix, number translationX = 1, number translationY = 1, number
translationZ = 1, number setType = MATRIX_POSTMULTIPLY )
```

Arguments

matrix - The matrix to be translated. translationXyz - A vector containing the x, y, and z translation to set the matrix to. translationX - A number containing the x translation to set the matrix to. translationY - A number containing the y translation to set the matrix to. translationZ - A number containing the z translation to set the matrix to. setType - How to set the matrix. Must be either MATRIX_POSTMULTIPLY, MATRIX_PREMULTIPLY, or MATRIX_REPLACE.

Return Values

None

Description

Sets up a translation matrix and depending on the set type it either post multiplies the current matrix by the translation matrix (matrix = matrix * translation), pre multiplies the current matrix by the translation matrix (matrix = translation * matrix) or simply sets the current matrix to the translation matrix (matrix = translation).

Examples

```
matrix = Matrix44.Create()
Matrix44.SetTranslation( matrix, 5, 10, 5, MATRIX_REPLACE )
```

See Also

None

Matrix44.Subtract

Brief

Performs a matrix subtraction with source1 and source2 storing the result in destination.

Definition

```
Matrix44.Subtract( Matrix44 destination, Matrix44 source1, Matrix44 source2 )
```

Arguments

destination - Matrix to receive the result. source1 - The first matrix argument for the subtraction. source2 - The second matrix argument for the subtraction.

Return Values

None.

Description

Performs a matrix subtraction with source1 and source2 storing the result in destination. Equivalent to destination = source1 - source2, but the operator syntax will allocate the destination automatically.

Examples

```
src1 = Matrix44.Create()
src2 = Matrix44.Create()
dst  = Matrix44.Create()
Matrix44.Subtract( dst, src1, src2 )
```

See Also

- Matrix44.Add

Matrix44.Transpose

Brief

Transposes the source matrix storing it in destination.

Definition

```
Matrix44.Transpose( Matrix44 destination, Matrix44 source )
```

Arguments

destination - The matrix to receive the transposed matrix. Can be the source matrix.source - The matrix to transpose.

Return Values

None

Description

Transposes the source matrix storing it in destination.

Examples

```
matrix = Matrix44.Create()
Matrix44.Transpose( matrix, matrix ) -- Self transpose.
```

See Also

None

MediaLibrary

MediaLibrary.ExportImage

Brief

Moves the image file to the Cross Media Bar (XMB).

Definition

```
boolean MediaLibrary.ExportImage( string filename, string caption, function callback = nil )
boolean MediaLibrary.ExportImage( Screenshot shot, string caption, function callback = nil )
```

Arguments

filename - The filename of the image file.shot - A screenshot that has been successfully capturedcaption - The caption of the image file to be displayed in the Cross Media Bar (XMB).callback - Function to call with the export results.

Return Values

True if it started exporting and false if it is busy.

Description

Moves the image file to the Cross Media Bar (XMB). The file is moved from its current location. The operation is aysnchronous. If the callback is supplied, it will be called and passed the result of the export (boolean, true for success) and if unsuccessful, an error code.

Examples

```
if screenshot:Succeeded() == true then
    MediaLibrary.ExportImage( screenshot, "Home Screenshot", ExportCallback )
end
...
function ExportCallback(success, errorCode)
    if (success) then
        print("Picture exported!")
    else
        print("Export error:", errorCode)
    end
end
```

See Also

- MediaLibrary.ExportVideo
- Screenshot.Create

MediaLibrary.ExportVideo

Brief

Moves the video file to the Cross Media Bar (XMB).

Definition

```
boolean MediaLibrary.ExportVideo( FileHandle file, string caption, function callback = nil )
```

Arguments

file - A handle to the video file, obtained from VideoRecorder.GetOutputFile.caption - The caption of the video file to be displayed in the Cross Media Bar (XMB).

Return Values

true if it started exporting and false if it is busy.

Description

Moves the video file to the Cross Media Bar (XMB). The file is moved from its current location. The operation is aysnchronous. Note that the exporter will not handle filenames with spaces, be aware of this when recording the video.

Examples

```
local file = VideoRecorder.GetOutputFile()
MediaLibrary.ExportVideo( file, "Home Video" )
```

See Also

- MediaLibrary.ExportImage
- VideoRecorder.GetOutputFile

MediaLibrary.GetSearchResults

Brief

Gets the current search results once the search has finished.

Definition

```
table[] MediaLibrary.GetSearchResults()
```

Arguments

None

Return Values

Returns an array (as a table) of tables of the search results. If nil is returned there are no results.

Description

Gets the current search results once the search has finished. The format of the results is a table array of tables. Each table has fields included through MediaLibrary.SetResultsToInclude.

Examples

```
local results = MediaLibrary.GetSearchResults()
if results ~= nil then
    for index, result in ipairs( results ) do
        for key, value in pairs( result ) do
            print( string.format( '%4i %s = %s', index, key, value ) )
        end
    end
end

local results = MediaLibrary.GetSearchResults()
print( 'file = ' .. results[ 1 ].file )
```

See Also

- MediaLibrary.IsSearchFinished
- MediaLibrary.SetResultsToInclude

MediaLibrary.IsExporting

Brief

Checks to see if the export is in process.

Definition

```
boolean MediaLibrary.IsExporting()
```

Arguments

None

Return Values

true if it is exporting and false otherwise.

Description

Checks to see if the export is in process.

Examples

```
while MediaLibrary.IsExporting() == true do
    coroutine.yield()
end
```

See Also

None

MediaLibrary.IsSearchFinished

Brief

Checks to see if the search has finished.

Definition

```
boolean MediaLibrary.IsSearchFinished()
```

Arguments

None

Return Values

true if the search has finished and false otherwise.

Description

Checks to see if the search has finished.

Examples

```
while MediaLibrary.IsSearchFinished() == false do
    coroutine.yield()
end
```

See Also

None

MediaLibrary.IsSearching

Brief

Checks to see if the search is in process.

Definition

```
boolean MediaLibrary.IsSearching()
```

Arguments

None

Return Values

true if it is searching and false otherwise.

Description

Checks to see if the search is in process.

Examples

```
while MediaLibrary.IsSearching() == true do
    coroutine.yield()
end
```

See Also

None

MediaLibrary.Search

Brief

Starts the specified search.

Definition

```
MediaLibrary.Search( string positionType, string mediaType, number maxCount = 1 )
```

Arguments

positionType - The position to search from. Valid values are 'first', 'last', 'prev', and 'next'.mediaType - The type of media to search. Valid values are 'video', 'music', and 'photo'.maxCount - The maximum count of results to return at once. Defaults to 1.

Return Values

None

Description

Starts the specified search. You can only do one search at a time and searching is asynchronous. To get the search results use MediaLibrary.IsSearchFinished() and MediaLibrary.GetSearchResults(). The media library remembers the last position of your search for each media type so the typical way to use this would be a search from the beginning and then subsequent searches using 'next' until there are not any more results as long as you call the searches all between the same MediaLibrary.StartSearch() and MediaLibrary.StopSearch()

Examples

```
-- Get the first 10 photos only.
MediaLibrary.StartSearch()
MediaLibrary.SetResultsToInclude( 'file' )
MediaLibrary.Search( 'first', 'photo', 10 )
while MediaLibrary.IsSearchFinished() == false do
    coroutine.yield()
end
results = MediaLibrary.GetSearchResults()
if results ~= nil then
    for index, result in ipairs( results ) do
        count = count + 1
        for key, value in pairs( result ) do
            print( string.format( '%4d %4d %s = %s', count, index, key, value ) )
        end
    end
end
MediaLibrary.StopSearch()
```

See Also

- MediaLibrary.IsSearchFinished
- MediaLibrary.GetSearchResults
- MediaLibrary.SetResultsToInclude

MediaLibrary.SetPrepareSearchFiles

Brief

If true then all media files returned in the search results will be prepared.

Definition

```
MediaLibrary.SetPrepareSearchFiles( boolean value )
```

Arguments

value - If true then media files are prepared and if false they are not prepared.

Return Values

None

Description

If true then all media files returned in the search results will be prepared so they can be used, for instance in loading.

Examples

```
MediaLibrary.SetPrepareSearchFiles( true )
```

See Also

None

MediaLibrary.SetPrepareSearchThumbnails

Brief

If true then all thumbnails of media files returned in the search results will be prepared.

Definition

```
MediaLibrary.SetPrepareSearchThumbnails( boolean value )
```

Arguments

value - If true then thumbnails of the media files are prepared and if false they are not prepared.

Return Values

None

Description

If true then all thumbnails of media files returned in the search results will be prepared so they can be used, for instance in loading.

Examples

```
MediaLibrary.SetPrepareSearchThumbnails( true )
```

See Also

None

MediaLibrary.SetResultsToInclude

Brief

Sets what to include in the search results.

Definition

```
MediaLibrary.SetResultsToInclude( string includeType, boolean value = true )
```

Arguments

includeType - The type to include. Valid values are:
'file' - string, the media's filename.
'thumbnail' - string, the thumbnail filename.
'title' - string, the title of the media.
'width' - number, the width of the image if applicable.
'height' - number, the height of the image if applicable.value - If true then the type is included and if false it is excluded.

Return Values

None

Description

Sets what to include in the search results. By default nothing is included. For each field included a field is created for each media item in the array of search results.

Examples

```
MediaLibrary.StartSearch()
MediaLibrary.SetResultsToInclude( 'file', true )
MediaLibrary.SetResultsToInclude( 'title', true )
```

See Also

None

MediaLibrary.StartSearch

Brief

Prepares the media library for searching.

Definition

```
MediaLibrary.StartSearch()
```

Arguments

None

Return Values

None

Description

Prepares the media library for searching.

Examples

```
MediaLibrary.StartSearch()
```

See Also

- MediaLibrary.StopSearch

MediaLibrary.StopSearch

Brief

Stops the media library from a searching state.

Definition

```
MediaLibrary.StopSearch()
```

Arguments

None

Return Values

None

Description

Stops the media library from a searching state and clears any resources it is using. The only reason to stop searching is when you have definitely finished all your searching so you can free up the resources it has taken.

Examples

```
MediaLibrary.StopSearch()
```

See Also

- MediaLibrary.StartSearch

MemoryContainer

MemoryContainer.Create

Brief

Creates a new memory container.

Definition

```
MemoryContainer MemoryContainer.Create( number size, MemoryType memType = MemoryType.Main, number
alignment = 1 )
MemoryContainer MemoryContainer.Create( Resource resource )
```

Arguments

size - The size in bytes of the memory container.resource - A loaded file resource or based from a file resource.memType - Type of memory to allocate.alignment - Required alignment of the memory container start address, in bytes. Must be a power of 2.

Return Values

A new instance of a memory container.

Description

Creates a new memory container. A memory container is an allocated block of memory where other library functions can output data. For example, the BasicGenx library uses a memory container to output XML which can then be posted to a server via the HttpPostData library. If creating a memory container from a resource the memory container is read only and errors will occur if attempting to write to it. Note that only file resources are accepted; textures, soundstreams and other resource types will generate an error. The container will be allocated from the script's memory budget.

Examples

```
container = MemoryContainer.Create( 4096 )
hostContainer = MemoryContainer.Create( 8192, MemoryType.Host )
myResource = Resource.Find("myFile")
resourceContainer = MemoryContainer.Create( myResource )
```

See Also

- BasicGenx.Create
- HttpPostData.Create

MemoryContainer.DebugLoad

Brief

Loads the contents of a file into a memory container from the host PC.

Definition

```
boolean MemoryContainer.DebugLoad( MemoryContainer container, string filename )
```

Arguments

container - The instance of the memory container.filename - The name of the file to load

Return Values

True if the file loaded successfully, false otherwise.

Description

Loads the contents of a file into a memory container from the host PC.

Examples

```
mc = MemoryContainer.Create( 100 )
MemoryContainer.DebugPrint( mc, "d:/temp/mc.dat" )
```

See Also

- MemoryContainer.Create

MemoryContainer.DebugPrint

Brief

Prints the contents of the container to the tty for debugging.

Definition

```
MemoryContainer.DebugPrint( MemoryContainer container )
```

Arguments

container - The instance of the memory container.

Return Values

None.

Description

Prints the contents of the container to the tty for debugging.

Examples

```
mc = MemoryContainer.Create( 100 )
MemoryContainer.SetFloat( mc, 5, 123.12 )
MemoryContainer.DebugPrint( mc )
```

See Also

- MemoryContainer.Create

MemoryContainer.DebugSave

Brief

Saves the contents of a memory container to a file on the host PC.

Definition

```
MemoryContainer.DebugSave( MemoryContainer container, string filename )
```

Arguments

container - The instance of the memory container.filename - The name of the file to save

Return Values

None.

Description

Saves the contents of a memory container to a file on the host PC.

Examples

```
mc = MemoryContainer.Create( 100 )
MemoryContainer.DebugSave( mc, "d:/temp/mc.dat" )
```

See Also

- MemoryContainer.Create

MemoryContainer.GetBigInt

Brief

Obtains the data at the specified byte index as a BigInt value.

Definition

```
void MemoryContainer.GetBigInt( MemoryContainer container, number byteIndex, BigInt result )
```

Arguments

container - The instance of the memory container.byteIndex - The byte index.result - BigInt to store value

Return Values

None.

Description

If there is insufficient data to read the entire size of the BigInt value an error will result (for example, if there are 32 bits of data remaining but the BigInt is 64 bits in size, the function will fail).

Examples

```
mc = MemoryContainer.Create( 100 )
i = BigInt.Create(64)
MemoryContainer.GetBigInt( mc, 8, i )
```

See Also

- MemoryContainer.Create
- BigInt.Create

MemoryContainer.GetBit

Brief

Returns the value of the specified bit as a boolean.

Definition

```
boolean MemoryContainer.GetBit( MemoryContainer container, number byteIndex, number bitIndex )
```

Arguments

container - The instance of the memory container.byteIndex - The byte index.bitIndex - The bit index, starts at 0 and if it is 8 or more it moves on to the next byte(s).

Return Values

true if the bit is set and false otherwise.

Description

Returns the value of the specified bit as a boolean.

Examples

```
mc = MemoryContainer.Create( 100 )
print( MemoryContainer.GetBit( mc, 5, 0 ) )
```

See Also

- MemoryContainer.Create

MemoryContainer.GetBits

Brief

Returns the value inside the specified bit range as a number.

Definition

```
number MemoryContainer.GetBits( MemoryContainer container, number byteIndex, number bitIndex, number
bitCount )
```

Arguments

container - The instance of the memory container.byteIndex - The byte index to start from.bitIndex - The bit index, starts at 0 and if it is 8 or more it moves on to the next byte(s).bitCount - The number of bits to get. The maxium is 23 due to it mapping to the mantissa of the number.

Return Values

The number contained within the bit range.

Description

Returns the number contained within the bit range. It can cross byte boundaries.

Examples

```
mc = MemoryContainer.Create( 100 )
MemoryContainer.SetInt8( mc, 255 ) )
print( MemoryContainer.GetBits( mc, 5, 1, 3 ) ) -- Get bits 1 - 4 inclusive from byte 5
```

See Also

- MemoryContainer.Create
- MemoryContainer.SetBits
- MemoryContainer.SetBit
- MemoryContainer.GetBit

MemoryContainer.GetFloat16

Brief

Returns the value of the specified 16-bit float as a number.

Definition

```
number MemoryContainer.GetFloat16( MemoryContainer container, number byteIndex )
```

Arguments

container - The instance of the memory container.byteIndex - The byte index.

Return Values

A number converted from a 16-bit float.

Description

Returns the value of the specified float as a number.

Examples

```
mc = MemoryContainer.Create( 100 )
print( MemoryContainer.GetFloat16( mc, 2 ) )
```

See Also

- MemoryContainer.Create

MemoryContainer.GetFloat32

Brief

Returns the value of the specified float as a number.

Definition

```
number MemoryContainer.GetFloat32( MemoryContainer container, number byteIndex )
```

Arguments

container - The instance of the memory container.byteIndex - The byte index.

Return Values

A number which matches the format of a float.

Description

Returns the value of the specified float as a number.

Examples

```
mc = MemoryContainer.Create( 100 )
print( MemoryContainer.GetFloat32( mc, 8 ) )
```

See Also

- MemoryContainer.Create

MemoryContainer.GetInt8

Brief

Returns the value of the specified byte as a number.

Definition

```
number MemoryContainer.GetInt8( MemoryContainer container, number byteIndex )
```

Arguments

container - The instance of the memory container.byteIndex - The byte index.

Return Values

A number ranging from -128 to 127.

Description

Returns the value of the specified byte as a number.

Examples

```
mc = MemoryContainer.Create( 100 )
print( MemoryContainer.GetInt8( mc, 5 ) )
```

See Also

- MemoryContainer.Create

MemoryContainer.GetInt16

Brief

Returns the value of the specified word as a number.

Definition

```
number MemoryContainer.GetInt16( MemoryContainer container, number byteIndex )
```

Arguments

container - The instance of the memory container.byteIndex - The byte index.

Return Values

A number ranging from -32768 to 32767.

Description

Returns the value of the specified word as a number.

Examples

```
mc = MemoryContainer.Create( 100 )
print( MemoryContainer.GetInt16( mc, 5 ) )
```

See Also

- MemoryContainer.Create

MemoryContainer.GetSize

Brief

Returns the size of the memory container in bytes.

Definition

```
number MemoryContainer.GetSize( MemoryContainer container )
```

Arguments

container - The instance of the memory container.

Return Values

The size of the memory container in bytes.

Description

Returns the size of the memory container in bytes.

Examples

```
mc = MemoryContainer.Create( 1024 )
print( MemoryContainer.GetSize( mc ) )
```

See Also

- MemoryContainer.Create

MemoryContainer.GetString

Brief

Returns a string at the specified starting position.

Definition

```
string MemoryContainer.GetString( MemoryContainer container, number byteIndex, number maxSize = nil )
```

Arguments

container - The instance of the memory container.byteIndex - The byte index to the beginning of the string.maxSize - The maximum size in bytes of the string not including the null terminator. If not specified then it stops at the first null terminator or the end of the container.

Return Values

The string created from the memory container.

Description

Returns a string at the specified starting position until it either encounters a null terminator, reaches the specified maximum size or reaches the end of the memory container.

Examples

```
mc = MemoryContainer.Create( 100 )
MemoryContainer.SetString( mc, 0, "hello world." )
print( MemoryContainer.GetString( mc, 0 ) )
```

See Also

- MemoryContainer.SetString

MemoryContainer.GetUInt8

Brief

Returns the unsigned value of the specified byte as a number.

Definition

```
number MemoryContainer.GetUInt8( MemoryContainer container, number byteIndex )
```

Arguments

container - The instance of the memory container.byteIndex - The byte index.

Return Values

A number ranging from 0 to 255.

Description

Returns the value of the specified byte as an unsigned number.

Examples

```
mc = MemoryContainer.Create( 100 )
print( MemoryContainer.GetUInt8( mc, 5 ) )
```

See Also

- MemoryContainer.Create

MemoryContainer.GetUInt16

Brief

Returns the value of the specified word as an unsigned number.

Definition

```
number MemoryContainer.GetUInt16( MemoryContainer container, number byteIndex )
```

Arguments

container - The instance of the memory container.byteIndex - The byte index

Return Values

An unsigned number ranging from 0 to 65535.

Description

Returns the value of the specified word as an unsigned number.

Examples

```
mc = MemoryContainer.Create( 100 )
print( MemoryContainer.GetUInt16( mc, 5 ) )
```

See Also

- MemoryContainer.Create

MemoryContainer.GetUsedSize

Brief

Returns the current number of bytes written to the container.

Definition

```
MemoryContainer.GetUsedSize( MemoryContainer container )
```

Arguments

container - The instance of the memory container.

Return Values

number - The number of bytes written to the container.

Description

Returns the number of bytes written to the container using the ReadX()/WriteX() functions. The SetX() functions have no affect on this value.

Examples

```
mc = MemoryContainer.Create( 100 )
used = MemoryContainer.GetUsedSize( mc )
```

See Also

- MemoryContainer.SetUsedSize
- MemoryContainer.Tell
- MemoryContainer.Seek

MemoryContainer.MemCopy

Brief

Copy memory from one memory container to another.

Definition

```
MemoryContainer.MemCopy(MemoryContainer dest, number destOffset, MemoryContainer src, number
srcOffset, number bytes)
```

Arguments

dest - Destination memory container.destOffset - Offset into destination memory container at which to start writing.src - Source memory container.srcOffset - Offset into source memory container at which to start reading.bytes - Number of bytes to copy.

Return Values

None.

Description

Copy memory from source memory container into destination memory container, starting at respective offsets. Note that large memory copies should be avoided as this could affect the TRC compliance of your script with respect to execution time. If a large copy is required and is not required to complete in the current frame, please see AsyncCommands.StreamCopy.

Examples

```
local myResource = Resource.Find("data")
local resData = MemoryContainer.Create(myResource)
local mem = MemoryContainer.Create(resData:GetSize())
MemoryContainer.MemCopy(mem, 0, resData, 0, resData:GetSize())
```

See Also

- MemoryContainer.MemSet

MemoryContainer.MemSet

Brief

Set a number of bytes of a memory container to a specific value.

Definition

```
MemoryContainer.MemSet(MemoryContainer mem, number offset, number value, number count = nil)
```

Arguments

mem - Memory container to modify.offset - Offset into memory container at which to start writing.value - Unsigned byte value (0 - 255) to write.count - Number of bytes to set, or nil to set from offset to end of container.

Return Values

None.

Description

Sets a number of bytes in a memory container to a specific byte value. Note that large memory sets should be avoided as this could affect the TRC compliance of your script with respect to execution time.

Examples

```
local mem = MemoryContainer.Create(1024)
mem:MemSet(0, 0xff)
```

See Also

- MemoryContainer.MemCopy

MemoryContainer.ReadBigInt

Brief

Sets the value of the specified BigInt from the memory container.

Definition

```
MemoryContainer.ReadBigInt( MemoryContainer container, BigInt bigInt )
```

Arguments

container - The instance of the memory container.bigInt - The BigInt to store the value.

Return Values

None.

Description

Sets the value of the specified BigInt from the memory container. If there is insufficient space in the memory container to read the BigInt, an error will result. For example, if the container is 4 bytes from the end and you try to set a 64-bit BigInt, this function will error.

Examples

```
i = BigInt.Create(64)
MemoryContainer.ReadBigInt( mc, i )
```

See Also

- MemoryContainer.Create
- BigInt.Create
- MemoryContainer.SetBigInt
- MemoryContainer.WriteBigInt

MemoryContainer.ReadFloat16

Brief

Returns the value of the float16 at the read position float as a number.

Definition

```
number MemoryContainer.ReadFloat16( MemoryContainer container )
```

Arguments

container - The instance of the memory container.

Return Values

The value of the next float16 in the container.

Description

Returns the value of the float16 at the read position float as a number. The read position is then advanced two bytes.

Examples

```
mc = MemoryContainer.Create( 100 )
print( MemoryContainer.ReadFloat16( mc ) )
```

See Also

- MemoryContainer.Create

MemoryContainer.ReadFloat32

Brief

Returns the value of the float32 at the read position float as a number.

Definition

```
number MemoryContainer.ReadFloat32( MemoryContainer container )
```

Arguments

container - The instance of the memory container.

Return Values

The value of the next float32 in the container.

Description

Returns the value of the float32 at the read position float as a number. The read position is then advanced four bytes.

Examples

```
mc = MemoryContainer.Create( 100 )
print( MemoryContainer.ReadFloat32( mc ) )
```

See Also

- MemoryContainer.Create

MemoryContainer.ReadInt8

Brief

Returns the value of the byte at the read position float as a number.

Definition

```
number MemoryContainer.ReadInt8( MemoryContainer container )
```

Arguments

container - The instance of the memory container.

Return Values

The value of the next byte in the container.

Description

Returns the value of the byte at the read position float as a number. The read position is then advanced one byte.

Examples

```
mc = MemoryContainer.Create( 100 )
print( MemoryContainer.ReadInt8( mc ) )
```

See Also

- MemoryContainer.Create

MemoryContainer.ReadInt16

Brief

Returns the value of the int16 at the read position float as a number.

Definition

```
number MemoryContainer.ReadInt16( MemoryContainer container )
```

Arguments

container - The instance of the memory container.

Return Values

The value of the next int16 in the container.

Description

Returns the value of the int16 at the read position float as a number. The read position is then advanced two bytes.

Examples

```
mc = MemoryContainer.Create( 100 )
print( MemoryContainer.ReadInt16( mc ) )
```

See Also

- MemoryContainer.Create

MemoryContainer.ReadString

Brief

Returns the value of the string at the read position.

Definition

```
number MemoryContainer.ReadString( MemoryContainer container, readSize = 0 )
```

Arguments

container - The instance of the memory container.

Return Values

The value of the next string in the container.

Description

Reads a string from the memory container. If readSize is not specified or is zero, then the read size size is determined by searching for the terminating 0. If readSize is set, then readSize number of bytes will always be read and the string will never excede that size.

Examples

```
mc = MemoryContainer.Create( 100 )
print( MemoryContainer.ReadString( mc ) )
```

See Also

- MemoryContainer.Create

MemoryContainer.ReadUInt8

Brief

Returns the value of the byte at the read position as an unsigned number.

Definition

```
number MemoryContainer.ReadUInt8( MemoryContainer container )
```

Arguments

container - The instance of the memory container.

Return Values

The value of the 8-bit unsigned integer in the container.

Description

Returns the value of the 8-bit unsingned integer at the read position as a number. The read position is then advanced one byte.

Examples

```
mc = MemoryContainer.Create( 100 )
print( MemoryContainer.ReadUInt8( mc ) )
```

See Also

- MemoryContainer.Create

MemoryContainer.ReadUInt16

Brief

Returns the value of the uint16 at the current read position as a number.

Definition

```
number MemoryContainer.ReadUInt16( MemoryContainer container )
```

Arguments

container - The instance of the memory container.

Return Values

The value of the next uint16 in the container.

Description

Returns the value of the uint16 at the current read position as a number. The read position is then advanced two bytes.

Examples

```
mc = MemoryContainer.Create( 100 )
print( MemoryContainer.ReadUInt16( mc ) )
```

See Also

- MemoryContainer.Create
- MemoryContainer.ReadInt16

MemoryContainer.Reset

Brief

Resets a memory container.

Definition

```
MemoryContainer.Reset( MemoryContainer container )
```

Arguments

container - The instance of the memory container.

Return Values

None.

Description

Resets a memory container so that it is empty again. It will be in the same state as when it was created.

Examples

```
MemoryContainer.Reset( container )
```

See Also

- MemoryContainer.Create

MemoryContainer.Seek

Brief

Move the read/write position within the container.

Definition

```
MemoryContainer.Seek( MemoryContainer container, number position )
```

Arguments

container - The instance of the memory container.position - The new position to seek to.

Return Values

boolean - True on success.

Description

Move the read/write position within the container. This has no affect on the used size of the memory container and you can only seek within the used area of the container.

Examples

```
mc = MemoryContainer.Create( 100 )
MemoryContainer.SetUsedSize( mc, 50 )
MemoryContainer.Seek( mc, 50 )
```

See Also

- MemoryContainer.GetUsedSize
- MemoryContainer.Tell
- MemoryContainer.Seek

MemoryContainer.SetBigInt

Brief

Stores the value of the specified BigInt into the memory container at the given byte index.

Definition

```
number MemoryContainer.SetBigInt( MemoryContainer container, number byteIndex, BigInt value )
```

Arguments

container - The instance of the memory container.byteIndex - The byte index.value - The BigInt value to set into the memory container

Return Values

Returns the next byte index after the value in the BigInt is written.

Description

If there is insufficient space in the memory container to store the BigInt, an error will result. For example, if the byte offset is 4 bytes from the end of the container and you try to set a 64-bit BigInt, this function will error.

Examples

```
mc = MemoryContainer.Create( 100 )
i = BigInt.Create(64, 329823)
MemoryContainer.SetBigInt( mc, 8, i )
```

See Also

- MemoryContainer.Create
- BigInt.Create
- MemoryContainer.WriteBigInt

MemoryContainer.SetBit

Brief

Sets the value of the specified bit.

Definition

```
MemoryContainer.SetBit( MemoryContainer container, number byteIndex, number bitIndex, boolean value )
```

Arguments

container - The instance of the memory container.byteIndex - The byte index.bitIndex - The bit index, starts at 0 and if it is 8 or more it moves on to the next byte(s).value - The value to set the bit to.

Return Values

The next bit index along, can be greater than 7.

Description

Sets the value of the specified bit. If using the return value of this function and want to round it up to the next byte then do something similar to this example:
bitArray =

Unknown macro: { true, false, true, false, false, true, false, true, false, false, true, true }

```
mc = MemoryContainer.Create( 100 )
byteIndex = 23
bitIndex = 3
for index = 1, #bitArray do
bitIndex = MemoryContainer.SetBit( mc, byteIndex, bitIndex, bitArray[ index ] )
end
byteIndex = math.floor( byteIndex + ( bitIndex + 7 ) / 8 )
MemoryContainer.SetInt8( mc, byteIndex, 255 ) )
```

Examples

```
mc = MemoryContainer.Create( 100 )
MemoryContainer.SetBit( mc, 5, 1, true ) )
```

See Also

- MemoryContainer.Create

MemoryContainer.SetBits

Brief

Sets the specified range of bits to the specified value.

Definition

```
MemoryContainer.SetBits( MemoryContainer container, number byteIndex, number bitIndex, number
bitCount, number value )
```

Arguments

container - The instance of the memory container.byteIndex - The byte index.bitIndex - The bit index, starts at 0 and if it is 8 or more it moves on to the next byte(s).bitCount - The number of bits to set. Ranges from 1 to 23.value - The value to pack into the bits.

Return Values

The next bit index along, can be greater than 7.

Description

Sets the values of the specified bits.

Examples

```
mc = MemoryContainer.Create( 100 )
MemoryContainer.SetBits( mc, 5, 1, 4, 5 ) ) -- 3 bits can store values from 0 to 7.
```

See Also

- MemoryContainer.Create
- MemoryContainer.GetBits
- MemoryContainer.SetBit
- MemoryContainer.GetBit

MemoryContainer.SetFloat16

Brief

Stores the value of the specified number starting from the byte index into a Float16.

Definition

```
MemoryContainer.SetFloat16( MemoryContainer container, number byteIndex, number|number[] value )
```

Arguments

container - The instance of the memory container.byteIndex - The byte index.value - The value to set the 16-bit float to or if a table is supplied it sets all the values in the table starting from the byte index.

Return Values

Returns the next byte index after this value is written.

Description

Compresses the value of the specified to a Float16 starting from the byte index. The Float16 does not map directly to a number in lua. It is 2 bytes in size and the bits are distributed as:
bits 0 - 9 : Mantissa
bits 10 - 14 : Exponent
bit 15 : Sign
The range of the 16-bit float is between -131008 and 131008. Values out of this range are clamped to it.

Examples

```
mc = MemoryContainer.Create( 100 )
MemoryContainer.SetFloat16( mc, 5, 123.12 )
```

See Also

- MemoryContainer.Create

MemoryContainer.SetFloat32

Brief

Sets the value of the specified Float32 starting from the byte index.

Definition

```
MemoryContainer.SetFloat32( MemoryContainer container, number byteIndex, number|number[] value )
```

Arguments

container - The instance of the memory container.byteIndex - The byte index.value - The value to set in the memory container or if a table is supplied it sets all the values in the table starting from the byte index.

Return Values

Returns the next byte index after this value is written.

Description

Sets the value of the specified Float32 starting from the byte index. The Float32 maps directly to a number in lua. It is 4 bytes in size and the bits are distributed as:
bits 0 - 22 : Mantissa
bits 23 - 30 : Exponent
bit 31 : Sign

Examples

```
mc = MemoryContainer.Create( 100 )
MemoryContainer.SetFloat32( mc, 5, 123.12 )
```

See Also

- MemoryContainer.Create

MemoryContainer.SetInt8

Brief

Sets the value of the specified byte.

Definition

```
MemoryContainer.SetInt8( MemoryContainer container, number byteIndex, number|number[] value )
```

Arguments

container - The instance of the memory container.byteIndex - The byte index.value - The value to set in the memory container or if a table is supplied it sets all the values in the table starting from the byte index.

Return Values

Returns the next byte index after this value is written.

Description

Sets the value of the specified byte.

Examples

```
mc = MemoryContainer.Create( 100 )
MemoryContainer.SetInt8( mc, 5, 123 )
```

See Also

- MemoryContainer.Create

MemoryContainer.SetInt16

Brief

Sets the value of the specified word starting from the byte index.

Definition

```
MemoryContainer.SetInt16( MemoryContainer container, number byteIndex, number|number[] value )
```

Arguments

container - The instance of the memory container.byteIndex - The byte index.value - The value to set in the memory container or if a table is supplied it sets all the values in the table starting from the byte index.

Return Values

Returns the next byte index after this value is written.

Description

Sets the value of the specified word starting from the byte index.

Examples

```
mc = MemoryContainer.Create( 100 )
MemoryContainer.SetInt16( mc, 5, 12322 )
```

See Also

- MemoryContainer.Create

MemoryContainer.SetString

Brief

Stores a string in the memory container at the specified starting position.

Definition

```
MemoryContainer.SetString( MemoryContainer container, number byteIndex, string value )
```

Arguments

container - The instance of the memory container.byteIndex - The byte index to the beginning of the string.value - The string to store.

Return Values

Returns the next byte index after the string and null terminator are written.

Description

Stores a string in the memory container at the specified starting position. The memory container needs enough avaiable space to store the string and a null terminator. If there is not enough space the memory container generates an error.

Examples

```
mc = MemoryContainer.Create( 100 )
MemoryContainer.SetString( mc, 0, "hello world." )
print( MemoryContainer.GetString( mc, 0 ) )
```

See Also

- MemoryContainer.GetString

MemoryContainer.SetUsedSize

Brief

Sets the used size in bytes in the memory container.

Definition

```
MemoryContainer.GetUsedSize( MemoryContainer container, number usedSize )
```

Arguments

container - The instance of the memory container.usedSize - The number of bytes to set the used size of the container.

Return Values

Description

Sets the used size in bytes in the memory container so you can set the used size and within it and then use the ReadX()/WriteX() functions. The SetX() functions have no affect on this value.

Examples

```
mc = MemoryContainer.Create( 100 )
MemoryContainer.SetInt8( 0, 1 )
MemoryContainer.SetUsedSize( 1 )
MemoryContainer.WriteInt8( 2 )
used = MemoryContainer.SetUsedSize( 50 )
```

See Also

- MemoryContainer.GetUsedSize
- MemoryContainer.Tell
- MemoryContainer.Seek

MemoryContainer.Tell

Brief

Returns the current read/write position within the container.

Definition

```
MemoryContainer.Tell( MemoryContainer container )
```

Arguments

container - The instance of the memory container.

Return Values

number - The current read/write position in the container.

Description

Returns the current read/write position within the container.

Examples

```
mc = MemoryContainer.Create( 100 )
pos = MemoryContainer.Tell( mc )
```

See Also

- MemoryContainer.GetUsedSize
- MemoryContainer.SetUsedSize
- MemoryContainer.Seek

MemoryContainer.WriteBigInt

Brief

Writes the value of the specified BigInt into the memory container.

Definition

```
number MemoryContainer.WriteBigInt( MemoryContainer container, BigInt value )
```

Arguments

container - The instance of the memory container.value - The BigInt value to set into the memory container

Return Values

None.

Description

If there is insufficient space in the memory container to store the BigInt, an error will result. For example, if the container is 4 bytes from the end and you try to set a 64-bit BigInt, this function will error.

Examples

```
mc = MemoryContainer.Create( 100 )
i = BigInt.Create(64, 329823)
MemoryContainer.WriteBigInt( mc, i )
```

See Also

- MemoryContainer.Create
- BigInt.Create
- MemoryContainer.SetBigInt

MemoryContainer.WriteFloat16

Brief

Writes an float16 to a memory container.

Definition

```
MemoryContainer.WriteFloat16( MemoryContainer container, number|number[] value )
```

Arguments

container - The instance of the memory container.value - The value to write to the memory container or if a table then an array of values to write.

Return Values

None.

Description

Writes a float16 to a memory container and advances the write position.

Examples

```
mc = MemoryContainer.Create( 100 )
MemoryContainer.WriteFloat16( mc, 123.4 )
```

See Also

- MemoryContainer.Create

MemoryContainer.WriteFloat32

Brief

Writes a float32 to a memory container.

Definition

```
MemoryContainer.WriteFloat32( MemoryContainer container, number|number[] value )
```

Arguments

container - The instance of the memory container.value - The value to write to the memory container or if a table then an array of values to write.

Return Values

None.

Description

Writes a float32 to a memory container and advances the write position.

Examples

```
mc = MemoryContainer.Create( 100 )
MemoryContainer.WriteFloat32( mc, 123.4 )
```

See Also

- MemoryContainer.Create

MemoryContainer.WriteInt8

Brief

Writes a byte to a memory container.

Definition

```
MemoryContainer.WriteInt8( MemoryContainer container, number|number[] value )
```

Arguments

container - The instance of the memory container.value - The value to write to the memory container or if a table then an array of values to write.

Return Values

None.

Description

Writes a byte to a memory container and advances the write position.

Examples

```
mc = MemoryContainer.Create( 100 )
MemoryContainer.WriteInt8( mc, 123 )
```

See Also

- MemoryContainer.Create

MemoryContainer.WriteInt16

Brief

Writes an int16 to a memory container.

Definition

```
MemoryContainer.WriteInt16( MemoryContainer container, number|number[] value )
```

Arguments

container - The instance of the memory container.value - The value to write to the memory container or if a table then an array of values to write.

Return Values

None.

Description

Writes an int16 to a memory container and advances the write position.

Examples

```
mc = MemoryContainer.Create( 100 )
MemoryContainer.WriteInt16( mc, 123 )
```

See Also

- MemoryContainer.Create

MemoryContainer.WriteString

Brief

Writes a string to a memory container.

Definition

```
MemoryContainer.WriteString( MemoryContainer container, string value, number writeSize = 0 )
```

Arguments

container - The instance of the memory container.value - The value to write.writeSize - The number of bytes to write to the memory container

Return Values

None.

Description

Writes a string to a memory container and advances the write position. If the writeSize is specified, then writeSize number of bytes will always be written to the container.

Examples

```
mc = MemoryContainer.Create( 100 )
MemoryContainer.WriteString( mc, "Hello World!" )
```

See Also

- MemoryContainer.Create

MiniGame

MiniGame.AddCustomExitMenuOption

Brief

Add a custom menu item to the Exit Menu for the mini-game.

Definition

```
MiniGame.AddCustomExitMenuOption(string optionText)
```

Arguments

optionText - The localized string to display. Note: This must contain the "[SQUARE]" tag to denote the SQUARE button selects this option. This will almost certainly be at the start of the text, and likely include a space after it, to separate it from the rest of the text. E.g. "[SQUARE] Settings".

Return Values

None.

Description

This enables the custom menu option in the mini-game's Exit Menu, and sets the text for that option. The text must be localized before input. Only one custom option can be added. Successive calls to this function can be used to alter the text displayed. Use the 'on_custom_exit_menu_option_selected' callback to detect the SQUARE button being pressed.

Examples

```
local customText = Object.GetMe():GetLocalizedText("myCustomOptionText")
MiniGame.AddCustomExitMenuOption(myCustomOptionText)
```

See Also

- MiniGame.RemoveCustomExitMenuOption
- MiniGame.IsAbleToChangeCustomExitMenuOption

MiniGame.CloseSession

Brief

Close the minigame network session to new player entry.

Definition

```
MiniGame.CloseSession()
```

Arguments

None.

Return Values

None.

Description

Close the current minigame network session to prevent new players from joining. While the session is closed, new players will not be able to join the game. Only the session owner may call this function.

Examples

```
if (MiniGame.IsSessionOwner() and gameIsReadyToStart) then      -- some user flag indicating the lobby
period is finished
    MiniGame.CloseSession()
end
```

See Also

- MiniGame.OpenSession
- MiniGame.IsSessionOwner
- MiniGame.IsSessionOpen

MiniGame.ConfirmReceivedWelcomeMessage

Brief

Confirms that a valid welcome message has been received.

Definition

```
MiniGame.ConfirmReceivedWelcomeMessage()
```

Arguments

None

Return Values

None.

Description

Use this function to confirm that a valid welcome message has been received. Until this is called, the local player cannot join the mini-game (if welome messages are not disabled), as it is assumed they do not know the current state of it.

Examples

```
function OnReceivedMessage()
    local message = ReceivedMessage.GetEventMessage()
    local type    = ReceivedMessage.GetType( message )

    if ( type == WELCOME_MESSAGE ) then
        gameState = ReceivedMessage.ReadInt32( message )
        MiniGame.ConfirmReceivedWelcomeMessage()
    else
        -- Handle other message types
    end
end
```

See Also

- MiniGame.DisableWelcomeMessages
- MiniGame.SetSafeToSendWelcomeMessage

MiniGame.DisableWelcomeMessages

Brief

Disables welcome messages.

Definition

```
MiniGame.DisableWelcomeMessages()
```

Arguments

None

Return Values

None.

Description

You must call this function if you do not want to use welcome messages in your mini-game. A welcome message is a full refresh of game state, typically used if an avatar joins a scene and has no prior knowledge of the game. If you don't call this function, the mini-game system will expect welcome messages to be sent and received, and therefore won't allow a player to join the game until one is received from a remote player. You are advised to call this function once when your script initializes.

Examples

```
MiniGame.DisableWelcomeMessages()
```

See Also

- MiniGame.ConfirmReceivedWelcomeMessage

- MiniGame.SetSafeToSendWelcomeMessage

MiniGame.EnableQueue

Brief

When called the mini-game uses the queuing system.

Definition

```
MiniGame.EnableQueue()
```

Arguments

None

Return Values

None.

Description

When called the mini-game uses the queuing system. The default is for it to not use it. Note that this function must only be called at initialization of the script. The queuing system is totally automated and will handle joining players into the mini-game from the queue. Note: Queues act on game-types, not individual game instances. E.g. If you had 4 pool tables, the queue would be for all 4, and would place you on the first one available. The queuing system supplies queues based on the name of the mini-game. Note there's a maximum of 16 game types per scene.

Examples

```
MiniGame.EnableQueue()
```

See Also

None

MiniGame.GetCurrentFocusPersonId

Brief

Gets the person id currently in focus.

Definition

```
string MiniGame.GetCurrentFocusPersonId()
```

Arguments

None

Return Values

The person id relevant to the event callback in which it is called.

Description

Gets the currently focussed person id in the context of certain specific callbacks as specified by the 'mini_game' component in the HDK's Object Editor. This function is only valid for the following callbacks:
on_add_player
on_remove_player

on_local_player_join
on_local_player_leave
on_send_welcome_message

Examples

```
function OnAddPlayer()
    personid = MiniGame.GetCurrentFocusPersonId()
    print( "Player added: " .. personid )
end
```

See Also

- MiniGame.GetCurrentFocusPlayerSlot

MiniGame.GetCurrentFocusPlayerSlot

Brief

Gets the player slot currently in focus.

Definition

```
number MiniGame.GetCurrentFocusPlayerSlot()
```

Arguments

None

Return Values

The player slot relevant to the event callback in which it is called.

Description

Gets the currently focussed player slot in the context of certain specific callbacks as specified by the 'mini_game' component in the HDK's Object Editor. This function is only valid for the following callbacks: on_add_player on_remove_player on_local_player_join on_local_player_leave

Examples

```
function OnRemovePlayer()
    slot = MiniGame.GetCurrentFocusPlayerSlot()
    print( "Player slot of departing player: " .. slot)
end
```

See Also

- MiniGame.GetCurrentFocusPersonId

MiniGame.GetJoinType

Brief

Get the current join type of the minigame.

Definition

```
JoinType MiniGame.GetJoinType()
```

Arguments

None

Return Values

The join type of the minigame.

Description

Get the current join type of the minigame.

Examples

```
if (MiniGame.GetJoinType() == JoinType.Auto) then
    MiniGame.SetJoinType(JoinType.Normal)
end
```

See Also

- MiniGame.SetJoinType

MiniGame.GetMaxPlayerSlots

Brief

Returns the maximum number of player slots in the mini-game.

Definition

```
number MiniGame.GetMaxPlayerSlots()
```

Arguments

None

Return Values

The maximum number of player slots in the mini-game.

Description

Returns the maximum number of player slots in the mini-game. Not all of these necessarily get filled. Imagine a roulette table with 20 seats (the player slots), where only a maximum of 8 people are allowed to play at any one time.

Examples

```
print( "The max player slots are " .. MiniGame.GetMaxPlayerSlots() )
```

See Also

- MiniGame.GetCurrentFocusPlayerSlot
- MiniGame.GetPlayerSlotUsedByPersonId

MiniGame.GetNumberOfUsedPlayerSlots

Brief

Returns the number of people currently joined to the mini-game.

Definition

```
number MiniGame.GetNumberOfUsedPlayerSlots()
```

Arguments

None

Return Values

The number of people currently joined to the mini-game.

Description

Returns the number of people currently joined to the mini-game. Each person takes one player slot.

Examples

```
print( "Player slot count = " .. MiniGame.GetNumberOfUsedPlayerSlots() )
```

See Also

- MiniGame.GetCurrentFocusPlayerSlot
- MiniGame.GetPlayerSlotUsedByPersonId

MiniGame.GetPersonIdFromPlayerSlot

Brief

Returns the person ID of the given player slot. Note: Slots start from 0 (e.g. first player slot is 0, second is 1, etc).

Definition

```
string MiniGame.GetPersonIdFromPlayerSlot( number playerSlot )
```

Arguments

playerSlot - The player slot in question.

Return Values

The person ID of the given player slot.

Description

Returns the person ID of the given player slot.

Examples

```
print( "Person id of slot 0 is " .. MiniGame.GetPersonIdFromPlayerSlot( 0 ) )
```

See Also

- MiniGame.GetCurrentFocusPlayerSlot
- MiniGame.GetPlayerSlotUsedByPersonId

MiniGame.GetPlayerSlotUsedByPersonId

Brief

Returns the player slot of the specified person id.

Definition

```
number MiniGame.GetPlayerSlotUsedByPersonId( string personId )
```

Arguments

personId - Person id of an avatar that has joined the mini-game.

Return Values

The player slot of the specified player person id.

Description

Returns the player slot of the specified person id. Returns -1 if the person does not have a player slot.

Examples

```
localPlayerId = LocalPlayer.GetPersonId()
if MiniGame.IsPersonJoined( localPlayerId ) then
    slot = MiniGame.GetPlayerSlotUsedByPersonId( localPlayerId )
    print( "I am playing the game and my slot is " .. slot )
end
```

See Also

- LocalPlayer.GetPersonId
- MiniGame.IsPersonJoined

MiniGame.GetQueueSize

Brief

Used to determine the length of this mini game's queue.

Definition

```
number MiniGame.GetQueueSize()
```

Arguments

None

Return Values

Size of the queue.

Description

Returns the current size of the queue. Note: the accuracy of this value may fluctuate slightly depending on network traffic and timings etc.

Examples

```
if MiniGame.GetQueueSize() > 10 then
    TellCurrentPlayersToHurryUp()
end
```

See Also

None

MiniGame.GetTriggerRadius

Brief

Gets the mini-game's trigger radius.

Definition

```
number MiniGame.GetTriggerRadius()
```

Arguments

None

Return Values

The mini-game's trigger radius.

Description

Gets the mini-game's trigger radius in meters. This is the radius from the game position that the player must be within in order to have the option to join.

Examples

```
p = Object.GetInitialPosition( Object.GetMe() )
r = MiniGame.GetTriggerRadius()
print( "Game located at "  .. p .. " with trigger radius " .. r  )
```

See Also

- Object.GetInitialPosition
- Object.GetMe

MiniGame.IsAbleToChangeCustomExitMenuOption

Brief

Determines if a custom menu option can currently be added to the Exit Menu.

Definition

```
MiniGame.IsAbleToChangeCustomExitMenuOption()
```

Arguments

None

Return Values

Whether a custom menu option can currently be added to the Exit Menu.

Description

Changing the custom menu option while the Exit Menu is visible is not user friendly and is therefore restricted. This function can be used to determine when 'AddCustomExitMenuOption' and 'RemoveCustomExitMenuOption' will fail.

Examples

```
MiniGame.IsAbleToChangeCustomExitMenuOption()
```

See Also

- MiniGame.AddCustomExitMenuOption
- MiniGame.RemoveCustomExitMenuOption

MiniGame.IsPersonJoined

Brief

Returns true if the specified person has joined and false otherwise.

Definition

```
boolean MiniGame.IsPersonJoined( string personId )
```

Arguments

personId - The ID of the person.

Return Values

true if the specified person has joined and false otherwise.

Description

Returns true if the specified person has joined and false otherwise.

Examples

```
print( MiniGame.IsPersonJoined( LocalPlayer.GetPersonId() ) )
```

See Also

- LocalPlayer.GetPersonId

MiniGame.IsSessionOpen

Brief

Returns true if the minigame network session is currently open.

Definition

```
boolean MiniGame.IsSessionOpen()
```

Arguments

None.

Return Values

True if the network session is open, or false if closed.

Description

Returns true if the minigame network session is currently open. An open session will accept new player entry and a closed session will not.

Examples

```
if (MiniGame.IsSessionOpen() and MiniGame.IsSessionOwner()) then
    MiniGame.CloseSession()
end
```

See Also

- MiniGame.OpenSession
- MiniGame.CloseSession
- MiniGame.IsSessionOwner

MiniGame.IsSessionOwner

Brief

Returns true if the current player is the minigame network session owner.

Definition

```
boolean MiniGame.IsSessionOwner()
```

Arguments

None.

Return Values

True if the current player is the network session owner, or false otherwise.

Description

One player is designated as the network session owner for the minigame at all times. This function can be used to identify this player.

Examples

```
if (MiniGame.IsSessionOwner()) then
    print("I own this game.")
end
```

See Also

- MiniGame.OpenSession
- MiniGame.CloseSession
- MiniGame.IsSessionOpen

MiniGame.LocalPlayerLeaveGame

Brief

Triggers the local player to leave the mini-game.

Definition

```
MiniGame.LocalPlayerLeaveGame()
```

Arguments

None

Return Values

None.

Description

Triggers the local player to leave the mini-game. This will cause the function associated with the 'on_local_player_leave' property of the 'mini_game' object component to be called.

Examples

```
function OnRemovePlayer()
    -- If one player leaves the game, force everyone else to leave too
    me = LocalPlayer.GetPersonId()
    if (MiniGame.IsPersonJoined( me ) then
        MiniGame.LocalPlayerLeaveGame()
    end
end
```

See Also

- LocalPlayer.GetPersonId
- MiniGame.IsPersonJoined

MiniGame.OpenSession

Brief

Open the minigame network session to new player entry.

Definition

```
MiniGame.OpenSession()
```

Arguments

None.

Return Values

None.

Description

Open a closed session to allow new players to join. Calling this on an already open session will have no effect. Only the session owner may call this function, otherwise a Lua error will occur.

Examples

```
if (MiniGame.IsSessionOwner() and not MiniGame.IsSessionOpen()) then
    MiniGame.OpenSession()
end
```

See Also

- MiniGame.CloseSession
- MiniGame.IsSessionOwner
- MiniGame.IsSessionOpen

MiniGame.RemoveCustomExitMenuOption

Brief

Remove the custom menu item from the Exit Menu for the mini-game (if one has been added).

Definition

```
MiniGame.RemoveCustomTargetMenuOption(string optionText)
```

Arguments

None

Return Values

None.

Description

This removes the custom menu option in the mini-game's Exit Menu, if one has previously been added.

Examples

```
MiniGame.RemoveCustomExitMenuOption()
```

See Also

- MiniGame.AddCustomExitMenuOption
- MiniGame.IsAbleToChangeCustomExitMenuOption

MiniGame.SetJoinType

Brief

Set the join type of a minigame.

Definition

```
MiniGame.SetJoinType(JoinType type)
```

Arguments

type - The join type to use (JoinType.Normal, JoinType.Auto, JoinType.Minimal).

Return Values

None.

Description

Set the join type of a minigame. If the user is within range and the join type is changed to JoinType.Auto, the user will automatically join the game.

Examples

```
MiniGame.SetJoinType(JoinType.Normal)
```

See Also

- MiniGame.GetJoinType

MiniGame.SetLocalPlayerExcludeMessage

Brief

Set the message to be displayed if the minigame prevents the local player from joining.

Definition

```
MiniGame.SetLocalPlayerExcludeMessage( string message )
```

Arguments

message - the localized string to display, or an empty string to display the default message

Return Values

None.

Description

If the minigame implements the on_can_local_player_join callback, it is able to return false to indicate that the local player may not join at this time. This may be due to the game being busy or unable to accept new players at this time (for example, if a race is in progress). This function allows the script to specify a message to be displayed to the user in this event to clarify the reason for the game not being joinable, and when it will become available again. The function can be called from anywhere in the script and the message will remain active until set to another message. Note that if queueing is enabled, this function will raise an error.

Examples

```
function OnCanLocalPlayerJoin()
    if (RaceInProgress()) then
        local msg = Object.GetMe():GetLocalizedText("RACE_IN_PROGRESS")
        MiniGame.SetLocalPlayerExcludeMessage(msg)
        return false
    end
    return true
end
```

See Also

None

MiniGame.SetSafeToSendWelcomeMessage

Brief

Use this function to allow the mini-game to send welcome messages.

Definition

```
MiniGame.SetSafeToSendWelcomeMessage( boolean value = true )
```

Arguments

value - true or false. Defaults to true.

Return Values

None.

Description

Use this function to allow the mini-game to send welcome messages (if using welcome messages in the mini-game). As an example, a Pool game will not want to send welcome messages when the balls are moving. So, you should set to true at all times other than when the balls are moving.

Examples

```
function OnUpdate()
    -- Always safe for the mini-game to send a welcome message
    MiniGame.SetSafeToSendWelcomeMessage( true )
end
```

See Also

- MiniGame.ConfirmReceivedWelcomeMessage
- MiniGame.DisableWelcomeMessages

NetPropertyBag

NetPropertyBag.Create

Brief

Creates a new NetPropertyBag.

Definition

```
NetPropertyBag NetPropertyBag.Create( string bagDefinitionName )
```

Arguments

bagDefinitionName - The name of the bag definition XML file resource.

Return Values

A new instance of a network property bag.

Description

Creates a new NetPropertyBag from an XML definition resource file.

Examples

```
bag = NetPropertyBag.Create( "player" )
```

See Also

- NetPropertyBag.GetEventBag

NetPropertyBag.DeleteFromNet

Brief

Deletes a locally owned and replicated bag from the network.

Definition

```
NetPropertyBag.DeleteFromNet( NetPropertyBag bag )
```

Arguments

bag - The instance of the property bag.

Return Values

None.

Description

Deletes a locally owned and replicated bag from the network. The bag isn't deleted from memory at this point and so can be replicated again at a later time.

Examples

```
if NetPropertyBag.IsReplicated( bag ) and NetPropertyBag.IsOwnedLocally( bag ) then
    NetPropertyBag.DeleteFromNet( bag )
end
```

See Also

- NetPropertyBag.IsReplicated
- NetPropertyBag.ReplicateToNet

NetPropertyBag.GetBagIndex

Brief

Returns the index of a property bag.

Definition

```
number NetPropertyBag.GetBagIndex( NetPropertyBag bag )
```

Arguments

bag - The instance of the property bag.

Return Values

The bag index.

Description

Returns the index of a property bag.

Please be aware that this index is only unique to the client that instantiated the bag, so each bag created on a client will have

a unique value, however, it is not a globally unique value across clients.

Examples

```
local bagIndex = NetPropertyBag.GetBagIndex( bag )
```

See Also

None

NetPropertyBag.GetEventBag

Brief

Returns an instance of the bag that is the focus of the current event.

Definition

```
NetPropertyBag NetPropertyBag.GetEventBag()
```

Arguments

None

Return Values

The NetPropertyBag that is the focus of the current event.

Description

Returns an instance of the bag that is the focus of the current event. Only valid during a bag created/deleted event, as named by the 'on_bag_created' and 'on_bag_deleted' properties of the 'network' object component in the HDK's Object Editor application.

Examples

```
gNetBags = { };
function OnBagCreated( )
    local bag = NetPropertyBag.GetEventBag( );
    local bagId = bag:GetTypeId( );
    --We keep a reference to a stats and skills bag which only one person creates
    if bagId == NetPropertyBag.GetTypeIdForBag( "statistics" ) then
        gNetBags[ "gamestats" ] = bag;
    elseif bagId == NetPropertyBag.GetTypeIdForBag( "skills" ) then
        gNetBags[ "skills" ] = bag;
    end
end
```

See Also

- NetPropertyBag.Create

NetPropertyBag.GetField

Brief

Gets a value from a field in the property bag.

Definition

```
variant NetPropertyBag.GetField( NetPropertyBag bag, string fieldName, table dest = nil )
```

Arguments

bag - The instance of the property bag.fieldName - The name of the field to request.dest - An optional table parameter, if provided we fill this table instead of creating one from scratch

Return Values

The boolean/number/string/table value contained in the field.

Description

Gets a value from a field in the property bag. Any client to which the bag has been replicated can call this function. Note that array fields are supported, in this case a table will be returned containing all the values in the array.

Examples

```
local name = NetPropertyBag.GetField( bag, "name" )
print(s)
local array = NetPropertyBag.GetField( bag, "array" )
for i, v in ipairs(array) do
    print(tostring(v))
end
local mytable = {}
local array = NetPropertyBag( bag, "array", mytable )
```

See Also

- NetPropertyBag.SetField

NetPropertyBag.GetOwnerId

Brief

Returns the person id of the owner of the bag.

Definition

```
string NetPropertyBag.GetOwnerId( NetPropertyBag bag )
```

Arguments

bag - The instance of the property bag.

Return Values

The person id of the game owner.

Description

Returns the person id of the owner of the bag.

Examples

```
local bagOwnerId = NetPropertyBag.GetOwnerId( bag )
bagOwner = Person.FindInInstance( bagOwnerId )
```

See Also

- Person.FindInInstance

NetPropertyBag.GetTypeId

Brief

Returns the type id of the bag.

Definition

```
number NetPropertyBag.GetTypeId( NetPropertyBag bag )
```

Arguments

bag - The instance of the property bag.

Return Values

The bag's type id.

Description

Returns the type id of the bag.

Examples

```
bag = NetPropertyBag.Create( "Player" );
if bag:GetTypeId( ) == NetPropertyBag.GetTypeIdForBag( "Player" ) then
    print( "Both bags are of the same type" );
else
    print( "Bags have different types" );
end
```

See Also

- NetPropertyBag.GetTypeIdForBag

NetPropertyBag.GetTypeIdForBag

Brief

Returns the type id for the specified property bag.

Definition

```
number NetPropertyBag.GetTypeIdForBag( string bagDefinitionName )
```

Arguments

bagDefinitionName - The definition name.

Return Values

The bag's type id.

Description

Returns the type id for the specified property bag.

Examples

```
bag = NetPropertyBag.Create( "Player" );
if bag:GetTypeId( ) == NetPropertyBag.GetTypeIdForBag( "Player" ) then
    print( "Both bags are of the same type" );
else
    print( "Bags have different types" );
end
```

See Also

- NetPropertyBag.GetTypeId

NetPropertyBag.IsOwnedLocally

Brief

Returns the local ownership state of a bag.

Definition

```
boolean NetPropertyBag.IsOwnedLocally( NetPropertyBag bag )
```

Arguments

bag - The instance of the property bag.

Return Values

True if the bag is owned locally.

Description

Returns the local ownership state of a bag.

Examples

```
local isOwnedByMe = NetPropertyBag.IsOwnedLocally( bag )
```

See Also

None

NetPropertyBag.IsReplicated

Brief

Returns the replicated state of a bag.

Definition

```
boolean NetPropertyBag.IsReplicated( NetPropertyBag bag )
```

Arguments

bag - The instance of the property bag.

Return Values

True if the bag has been replicated to the network.

Description

Returns the replicated state of a bag.

Examples

```
local isReplicated = NetPropertyBag.IsReplicated( bag )
```

See Also

- NetPropertyBag.DeleteFromNet
- NetPropertyBag.ReplicateToNet

NetPropertyBag.ReplicateToNet

Brief

Replicates a locally owned bag to the network.

Definition

```
NetPropertyBag.ReplicateToNet( NetPropertyBag bag )
```

Arguments

bag - The instance of the property bag.

Return Values

None.

Description

Replicates a locally owned bag to the network.

Examples

```
NetPropertyBag.ReplicateToNet( bag )
```

See Also

- NetPropertyBag.DeleteFromNet
- NetPropertyBag.IsReplicated

NetPropertyBag.SetField

Brief

Sets a field in the property bag.

Definition

```
NetPropertyBag.SetField( NetPropertyBag bag, string fieldName, variant value )
```

Arguments

bag - The instance of the property bag.fieldName - The name of the field to set.value - boolean/number/string/table to store in the field.

Return Values

None.

Description

Sets a field in the property bag. The bag must be owned locally for this function to succeed. Note that array fields are supported, in this case a table should be supplied of an equal or lesser number of items as the array field size. In the case of the table having less elements than the field size, the remaining elements in the field are not updated.

Examples

```
NetPropertyBag.SetField( bag, "booleanValue", true )
NetPropertyBag.SetField( bag, "numberValue", 3.14159 )
NetPropertyBag.SetField( bag, "stringValue", "Hello World" )
NetPropertyBag.SetField( bag, "arrayValue", { 1, 2, 3, 4 } )
```

See Also

- NetPropertyBag.GetField

Object

Object.DebugCreate

Brief

Instantiates an object.

Definition

```
string Object.DebugCreate( string objectId, string type, Vector4 position=[0,0,0,1], Vector4
rotation=[0,0,0,0], Vector4 scale=[1,1,1,1] )
```

Arguments

objectId - The object ID of the object to be created.position - The initial position of the object in world space.rotation - The initial rotation of the object specified as XYZ-ordered local space axis rotations in degrees.scale - The initial scale of the object.type - Type of the object. Supported types are 'portable'.

Return Values

The instance ID of the newly created object.

Description

Instantiates an object. For debug purposes only.

Examples

```
-- Create a new object
objectId = "12345678-12345678-12345678-12345678"
instanceId = Object.DebugCreate( objectId )

-- Destroy the object
Object.DebugDestroy( instanceId )
```

See Also

- Object.DebugDestroy

Object.DebugDestroy

Brief

Destroys an object.

Definition

```
Object.DebugDestroy( string instanceId )
```

Arguments

instanceId - The instance ID of the object to destroy.

Return Values

None.

Description

Destroys an object. For debug purposes only.

Examples

```
-- Create a new object
objectId = "12345678-12345678-12345678-12345678"
instanceId = Object.DebugCreate( objectId )

-- Destroy the object
Object.DebugDestroy( instanceId )
```

See Also

- Object.DebugCreate

Object.DebugEnableDebugRendering

Brief

Enable/disable debug rendering for an object.

Definition

```
Object.DebugEnableDebugRendering( Object object, boolean setEnabled )
```

Arguments

object - The object for which to enable/disable debug rendering.

Return Values

None.

Description

Sets/unsets a flag in the object to indicate that debug rendering is/is not desired. The flag is interpretted by other systems, eg the collision system.

Examples

```
me = Object.GetMe()
Object.DebugEnableDebugRendering( me, true )
```

See Also

None

Object.DebugFind

Brief

Finds the object by the specified instance ID.

Definition

```
Object Object.DebugFind( string instanceId )
```

Arguments

instanceId - The instance ID of the object to find.

Return Values

Returns the object if it is found and nil otherwise.

Description

Finds the object by the specified instance ID.

Examples

```
obj = Object.DebugFind( "0000000000000123" )
```

See Also

None

Object.DebugGetAllInstanceIds

Brief

Returns a table of all the instance IDs in the scene.

Definition

```
string[] Object.DebugGetAllInstanceIds()
```

Arguments

None

Return Values

A table of all the instance IDs in the scene

Description

Returns a table of all the instance IDs in the scene. For debug purposes only.

Examples

```
ids = Object.DebugGetAllInstanceIds()
for i=1,#ids do
    print( ids[ i ] )
end
```

See Also

None

Object.DebugReport

Brief

Reports to the tty the current state of all the objects in the instance.

Definition

```
Object.DebugReport()
```

Arguments

None

Return Values

None.

Description

Reports to the tty the current state of all the objects in the instance. For debug purposes only.

Examples

```
Object.DebugReport()
```

See Also

None

Object.DebugRun

Brief

Runs the specified Lua string in the object's Lua environment.

Definition

```
Object.DebugRun( Object object, string luaString )
```

Arguments

object - The object whose Lua environment will execute the specified command. luaString - The Lua string to run in the object's Lua environment.

Return Values

None.

Description

Runs the specified Lua string in the object's Lua environment if it exists. For debug purposes only.

Examples

```
me = Object.GetMe()
Object.DebugRun( me, "print('hi')" )
```

See Also

None

Object.EnableTargeting

Brief

Enables the targeting system for this object if it has one. Defaults to enabled.

Definition

```
Object.EnableTargeting( Object object, boolean enable )
```

Arguments

object - The object to update.enable - The state to put the targeting in for this object.

Return Values

None.

Description

Enables the targeting system for this object if it has one. Defaults to enabled. This can only be used by the lua environment in the object. The object requires a targetable component.

Examples

```
me = Object.GetMe()
if g_isLocked == true then
    Object.EnableTargeting( me, false )
end
```

See Also

None

Object.GetCurrentListenerData

Brief

Gets the listener data sent to it.

Definition

```
string|table|boolean|number Object.GetCurrentListenerData()
```

Arguments

None

Return Values

The custom result(s) from the listener callback function.

Description

Listener data is only present whilst in the 'on_listener_received_data' callback.

Examples

```
function OnListenerReceivedData()
    data = Object.GetCurrentListenerData()
    if data.type == "letter_message" then
        for i,l in ipairs( data.letters ) do
            print( 'letter ' .. i .. ' = ' .. l )
        end
    end
end
```

See Also

- Object.SendListenerData

Object.GetInitialPosition

Brief

Returns the initial position of the object.

Definition

```
Vector4 Object.GetInitialPosition( Object object )
Object.GetInitialPosition( Object object, Vector4 outPos )
```

Arguments

object - The object to query.outPos - (out) The vector to receive the position.

Return Values

The initial position of the object.

Description

Returns the initial position of the object.

Examples

```
me = Object.GetMe()
print( "My initial position is ", Object.GetInitialPosition( me ) )
```

See Also

None

Object.GetInitialRotation

Brief

Returns the initial rotation of the object.

Definition

```
Vector4 Object.GetInitialRotation( Object object )
Object.GetInitialRotation( Object object, Vector4 outRotation )
```

Arguments

object - The object to query.outRotation - (out) The vector to receive the rotation.

Return Values

Returns the initial rotation of the object.

Description

Returns the initial rotation of the object. The vector returned is the x, y, and z angles in degrees.

Examples

```
me = Object.GetMe()
print( "My initial rotation is ", Object.GetInitialRotation( me ) )
```

See Also

None

Object.GetInitialScale

Brief

Returns the initial scale of the object.

Definition

```
Vector4 Object.GetInitialScale( Object object )
Object.GetInitialScale( Object object, Vector4 outScale )
```

Arguments

object - The object to query.outScale - (out) The vector to receive the scale.

Return Values

The initial scale of the object.

Description

Returns the initial scale of the object.

Examples

```
me = Object.GetMe()
print( "My initial scale is ", Object.GetInitialScale( me ) )
```

See Also

None

Object.GetInstanceId

Brief

Returns the object's instance ID.

Definition

```
string Object.GetInstanceId( Object object )
```

Arguments

object - The object to query.

Return Values

The object's instance ID.

Description

Returns the object's instance ID.

Examples

```
obj = Object.GetMe()
print( "My instance id is " .. Object.GetInstanceId( obj ) )
```

See Also

- Object.GetMe

Object.GetInstanceName

Brief

Returns the instance name that is set in the scene editor.

Definition

```
string Object.GetInstanceName( Object object )
```

Arguments

object - The object to query.

Return Values

Returns the instance name that is set in the scene editor.

Description

Returns the instance name that is set in the scene editor. Currently supported with embedded objects and scene minigames.

Examples

```
print( 'my instance name is ', Object.GetInstanceName( Object.GetMe() ) )
```

See Also

- Object.GetNameRef

- Object.GetObjectId
- Object.GetInstanceId

Object.GetInstanceParameter

Brief

Retrieve the value of a specified instance parameter for an embedded scene object.

Definition

```
string|number|boolean|nil Object.GetInstanceParameter(Object obj, string name)
```

Arguments

obj - the object to queryname - the name of the instance parameter (not case-sensitive)

Return Values

The value of the instance parameter if it exists, or nil if it cannot be found.

Description

Gets the value of a specified instance parameter assigned to an embedded scene object. Only objects with an Embedded component will have access to the instance parameters. The names of the instance parameters are not case sensitive, and the type of the value can be string, boolean or numeric, or nil if not found. "Enum"-typed instance parameters will be returned as strings.

Examples

```
local me = Object.GetMe()
local myGameId = me:GetInstanceParameter("GameID")
if (myGameId == nil or type(myGameId) ~= "number") then
    print("Invalid game ID parameter")
end
```

See Also

Object.GetListenerObjects

Brief

Get the list of objects registered to receive messages from this object.

Definition

```
table Object.GetListenerObjects(number startIndex = 1, number maxCount = -1)
```

Arguments

startIndex - The position to start from. Previous indices are not added to the results.maxCount - The maximum size the results array can be.

Return Values

An array of Objects. The array will be empty if no listening objects are registered.

Description

Objects can register to listen for messages from other objects. This function will retrieve a list of all currently loaded objects that have registered to receive messages from the current object. If no such objects exist an empty table will be

returned.

Examples

```
local listeners = Object.GetListenerObjects()
for i, obj in ipairs(listeners) do
    obj:SendListenerData("Hello!")
end
```

See Also

- Object.SendListenerData
- Scene.GetListenerObjects

Object.GetLocalizedText

Brief

Returns the localized string for the specified reference.

Definition

```
string Object.GetLocalizedText( Object object, string reference )
```

Arguments

object - The object to query.reference - The reference text to find the localization for.

Return Values

The localized string.

Description

Returns the localized string for the specified reference from the object's localization file.

Examples

```
me = Object.GetMe()
localizedGameOver = Object.GetLocalizedText( me, "GAME_OVER_MESSAGE" )
```

See Also

- Object.GetMe
- Object.GetNameRef

Object.GetMe

Brief

Returns the object that owns the script.

Definition

```
Object Object.GetMe()
```

Arguments

None

Return Values

The object that owns the script.

Description

Returns the object that owns the script.

Examples

```
me = Object.GetMe()
```

See Also

None

Object.GetNameRef

Brief

Returns the localization reference of the object's name.

Definition

```
string Object.GetNameRef( Object object )
```

Arguments

object - The object to query.

Return Values

The localization reference of the object's name.

Description

Returns the localization reference of the object's name.

Examples

```
me = Object.GetMe()
print( "My name ref is " .. Object.GetNameRef( me ) )
```

See Also

- Object.GetLocalizedText

Object.GetObjectId

Brief

Returns the object's object ID.

Definition

```
string Object.GetObjectId( Object object )
```

Arguments

object - The object to query.

Return Values

The object ID.

Description

The object's object ID.

Examples

```
me = Object.GetMe()
print( "My object id is " .. Object.GetObjectId( me ) )
```

See Also

- Object.GetMe

Object.GetPersonId

Brief

Returns the person ID that owns this object.

Definition

```
string Object.GetPersonId( Object object )
```

Arguments

object - The object to query.

Return Values

The person ID of the avatar that owns this object, or nil if the object is not owned by a person.

Description

Returns the person ID of the avatar that owns this object. Some objects, such as those placed in a clubhouse, are not considered as owned by a person, and but rather by the scene. Such objects will return nil using this function, scripts should take care to check for this case.

Examples

```
me = Object.GetMe()
ownerId = Object.GetPersonId( me )
if ( ownerId ~= nil ) then
    name = Person.GetName( ownerId )
    print( "My owner is " .. name )
end
```

See Also

- Object.GetMe

Object.HasFocus

Brief

Indicates if the object has the user's focus.

Definition

```
boolean Object.HasFocus( Object object )
```

Arguments

object - The object.

Return Values

True if the game is currently active, or false if it has lost the user input focus for any reason.

Description

Various events may take the user focus away from the currently active object, such as menus and notifications. A script may wish to use this function to poll for this case and pause the update or perform other actions while the user's focus is elsewhere. Note that due to the nature of the various actions that may take away the user focus, it is possible for this state to alternate briefly as certain systems become active or inactive. A script may wish to wait for several frames of the same focus state before reacting to a new state. The list of events that is considered to take focus away from the object is subject to change, but is currently:

- Minigame exit menu
- Menu Screen
- OSK
- Game launching OSD
- Chat log in maximized mode
- Console open (in developer builds)
- Screen adjustment
- Controller disconnection

Examples

```
paused = false
function OnUpdate()
    paused = not Object.GetMe():HasFocus()
    ...
    if (paused) then
        DoPausedLogic()
    else
        DoNormalGameLogic()
    end
end
```

See Also

None

Object.HasLocalizedText

Brief

Returns true if a localized string exists for the specified reference.

Definition

```
bool Object.HasLocalizedText( Object object, string reference )
```

Arguments

object - The object to query.reference - The reference text to find the localization for.

Return Values

True if a localized string exists for the specified reference or false otherwise.

Description

Returns true if there exists a localized string for the specified reference in the object's localization file.

Examples

```
if ( Object.HasLocalizedText( Object.GetMe(), "TEST_MESSAGE" ) ) then
    print( Object.GetLocalizedText( Object.GetMe(), "TEST_MESSAGE" ) )
end
```

See Also

- Object.GetMe
- Object.GetLocalizedText

Object.IsOwnedLocally

Brief

Returns true if the user owns this object and false otherwise.

Definition

```
boolean Object.IsOwnedLocally( Object object )
```

Arguments

object - The object to query.

Return Values

true if the user owns this object and false otherwise.

Description

Returns true if the user owns this object and false otherwise.

Examples

```
obj = Object.GetMe()
if Object.IsOwnedLocally( obj ) then
    print( "I am owned by this user" )
else
    print( "I belong to someone else" )
end
```

See Also

- Object.GetMe

Object.IsPaused

Brief

Reports if the object's Lua environment is paused.

Definition

```
boolean Object.IsPaused( Object object )
```

Arguments

object - The object to query.

Return Values

true if the specified object's Lua environment is paused and false otherwise.

Description

Reports if the Lua environment of the specified object is paused.

Examples

```
me = Object.GetMe()
print( 'pause state = ' .. Object.IsPaused( me ) )
```

See Also

- Object.Pause
- Object.Resume

Object.Pause

Brief

Pauses the lua environment.

Definition

```
Object.Pause( Object object )
```

Arguments

object - The object to be paused.

Return Values

None.

Description

Pauses the lua environment.

Examples

```
me = Object.GetMe()
Object.Pause( me )
```

See Also

- Object.Resume
- Object.IsPaused

Object.Resume

Brief

Resumes the lua environment.

Definition

```
Object.Resume( Object object )
```

Arguments

object - The object to be resumed.

Return Values

None.

Description

Resumes the lua environment if it is paused.

Examples

```
me = Object.GetMe()
Object.Resume( me )
```

See Also

- Object.Pause
- Object.IsPaused

Object.SendListenerData

Brief

Sends data from one object to another object.

Definition

```
string|table|boolean|number Object.SendListenerData( Object object, string|table|boolean|number data )
```

Arguments

object - The object to send the data to.data - The lua string to run in the object's lua environment.

Return Values

Custom return value depending on what the listener callback returns.

Description

Sends data from one object to another object.

Examples

```
sceneObject = Scene.GetVirtualObject()
t = { letters={ "a", "b", "c" }, type="letter_message" }
results = Object.SendListenerData( sceneObject, t )
print( results )
```

See Also

Object.SetFaceActiveTarget

Brief

Set whether the avatar will face the target upon activation.

Definition

```
Object.SetFaceActiveTarget(Object object, boolean value = true)
```

Arguments

object - The object to update.val - Whether the avatar will face the target upon activation.

Return Values

None.

Description

Sets whether the avatar will face the target upon activation. By default, it will face the target. The object requires a targetable component.

Examples

```
me = Object.GetMe()
Object.SetFaceActiveTarget(me, false)
```

See Also

- Object.GetMe

Object.SetTargetableLegend

Brief

Set the text of the targeting trigger (legend) for the object.

Definition

```
Object.SetTargetableLegend( Object object, string text )
```

Arguments

object - the object to update.text - the localized text to use, or nil/empty string to use the default (the object's name)

Return Values

None.

Description

Set the text of the targeting trigger (legend) for the object. This is the initial interaction trigger for the object in the scene, which brings up the targeting menu. The text should be pre-localized. The interaction symbol will be prepended to the text automatically. The object requires a targetable component.

Examples

```
me = Object.GetMe()
Object.SetTargetableLegend( me, "Play a game of pool!" )
```

See Also

- Object.SetTargetableText

Object.SetTargetablePosition

Brief

Sets the object's targetable position if it is targetable.

Definition

```
Object.SetTargetablePosition( Object object, Vector4 position )
```

Arguments

object - The object to update.position - The new targetable position

Return Values

None.

Description

Sets the object's targetable position if it is targetable using the targetable component. The position is in world space. The object requires a targetable component.

Examples

```
me = Object.GetMe()
Object.SetTargetablePosition( me, Vector4.Create( 1, 2, 3 ) )
```

See Also

- Object.GetMe

Object.SetTargetableText

Brief

Sets the text of the interact/play option in the targeting dialog.

Definition

```
Object.SetTargetableText( Object object, string text )
```

Arguments

object - The object to update.text - The localized text to display.

Return Values

None.

Description

Sets the text of the interact/play option in the targeting dialog. The object requires a targetable component.

Examples

```
me = Object.GetMe()
Object.SetTargetableText( me, 'Switch On' )
```

See Also

- Object.SetTargetableLegend

Osd

Osd.GetRoot

Brief

Returns the root OSD object.

Definition

```
OsdObject Osd.GetRoot()
```

Arguments

None

Return Values

The root OSD object.

Description

Returns the root OSD object. NOTE: From HDK 1.3.5 the OSD libraries require the developer to explicitly load each library (including any libraries created using the OSD component in Object Editor). Objects exported with an earlier version of the HDK will continue to automatically load OSD libraries as required. Loading an OSD Lua library also loads any libraries that the OSD object type depends upon. Therefore LoadLibrary("OsdBasicAnimGraphic") also loads OsdBasicSingleGraphic and OsdObject.

Examples

```
osdRoot = Osd.GetRoot()
```

See Also

None

OsdBasicAnimGraphic

OsdBasicAnimGraphic.AddFrame

Brief

Add a frame of animation to a basic anim graphic object.

Definition

```
OsdBasicAnimGraphic.AddFrame( OsdBasicAnimGraphic graphicObject, Texture back, Texture front, boolean
isReverseColor = nil )
```

Arguments

graphicObject - The basic anim graphic object to affect. back - Background texture for animation frame. front - Foreground texture for animation frame. isReverseColor - false if background/foreground textures are to be colored with background/foreground colors respectively, true to swap colors.

Return Values

None.

Description

Add a frame of animation to a basic anim graphic object. The "isReverseColor" parameter is optional and defaults to "false".

Examples

```
OsdBasicAnimGraphic.AddFrame( graphicObject, texture1, texture2, false )
```

See Also

None

OsdBasicAnimGraphic.GetNumFrames

Brief

Get the number of frames in a basic anim graphic object's animation sequence.

Definition

```
OsdBasicAnimGraphic.GetNumFrames( OsdBasicAnimGraphic graphicObject )
```

Arguments

graphicObject - The basic anim graphic object to query.

Return Values

The number of frames of animation.

Description

Get the number of frames in a basic anim graphic object's animation sequence.

Examples

```
local numFrames = OsdBasicAnimGraphic.GetNumFrames( graphicObject )
```

See Also

- OsdBasicAnimGraphic.AddFrame

OsdBasicAnimGraphic.IsAnimating

Brief

Query whether or not the basic anim graphic object is animating.

Definition

```
OsdBasicAnimGraphic.IsAnimating( OsdBasicAnimGraphic graphicObject )
```

Arguments

graphicObject - The basic anim graphic object to query.

Return Values

true if the basic anim graphic object is animating, false otherwise.

Description

Query whether or not the basic anim graphic object is animating.

Examples

```
if OsdBasicAnimGraphic.IsAnimating( graphicObject ) then
    print("AnimGraphic is animating!")
end
```

See Also

None

OsdBasicAnimGraphic.SetAnimating

Brief

Enable/disable the animation.

Definition

```
OsdBasicAnimGraphic.SetAnimating( OsdBasicAnimGraphic graphicObject, boolean setAnimating = true )
```

Arguments

graphicObject - The basic anim graphic object to affect.setAnimating - true to cause the graphic to animate, false to stop the graphic animating. Optional, defaults to true.

Return Values

None.

Description

Enable/disable the animation.

Examples

```
OsdBasicAnimGraphic.SetAnimating( graphicObject, true )
```

See Also

- OsdBasicAnimGraphic.AddFrame

OsdBasicAnimGraphic.SetCurrentFrame

Brief

Set the currently displayed frame.

Definition

```
OsdBasicAnimGraphic.SetCurrentFrame( OsdBasicAnimGraphic graphicObject, number frameIndex )
```

Arguments

graphicObject - The basic anim graphic object to affect. frameIndex - Index of frame to set as current.

Return Values

None.

Description

Set the currently displayed frame. Indices are from 1 to numFrames.

Examples

```
OsdBasicAnimGraphic.SetCurrentFrame( graphicObject, 1 )
```

See Also

- OsdBasicAnimGraphic.AddFrame

OsdBasicAnimGraphic.SetFrameRate

Brief

Set the frame rate at which to animate a basic anim graphic object.

Definition

```
OsdBasicAnimGraphic.SetFrameRate( OsdBasicAnimGraphic graphicObject, number framesPerSecond )
```

Arguments

graphicObject - The basic anim graphic object to affect. framesPerSecond - Frame rate at which to animate.

Return Values

None.

Description

Set the frame rate at which to animate a basic anim graphic object.

Examples

```
OsdBasicAnimGraphic.SetFrameRate( graphicObject, 30.0 )
```

See Also

- OsdBasicAnimGraphic.AddFrame

OsdBasicChip

OsdBasicChip.AddLegendItem

Brief

Creates a new legend item for the chip.

Definition

```
OsdBasicChip.AddLegendItem( OsdBasicChip chipObject, string localizedText )
```

Arguments

chipObject - Object to which the function is applied.text - Localized legend text.

Return Values

None.

Description

Creates a new legend item for the chip. Legend items are added left to right at the bottom of the chip. If more than one legend item is added to the chip, then they will be spaced evenly within the legend area. If more legends are added to the chip than can be displayed simultaneously, then they are cycled over time, with each legend item fading in and out of the legend area of the chip.

Examples

```
me = Object.GetMe()

-- Localized in Object Editor as "[LEFT][RIGHT][LEFTSTICK] Retarget"
legend1 = Object.GetLocalizedText( me, "RETARGET" )
-- Localized in Object Editor as "[CROSS] Select"
legend2 = Object.GetLocalizedText( me, "SELECT" )
-- Localized in Object Editor as "[CIRCLE] Exit"
legend3 = Object.GetLocalizedText( me, "EXIT" )

-- Add the 3 legend strings to the chip
OsdBasicChip.AddLegendItem( chip, legend1 )
OsdBasicChip.AddLegendItem( chip, legend2 )
OsdBasicChip.AddLegendItem( chip, legend3 )
```

See Also

- Object.GetLocalizedText
- Object.GetMe

OsdBasicChip.CreateContentObject

Brief

Creates a new OsdObject-derived object as content of the specified chip object.

Definition

```
variant OsdBasicChip.CreateContentObject( OsdBasicChip chipObject, string contentType, string
contentName )
```

Arguments

chipObject - Object to which the function is applied.contentType - A string representing the type of the content objectcontentName - The name of the content object.

Return Values

A new OsdObject-derived object of the specified type.

Description

Creates a new OsdObject-derived object as content of the specified chip object. Note: Garbage collection will not destroy OSD objects. OSD objects are only destroyed either when the parent OSD object is destroyed, or by explicitly calling OsdObject.DestroyObject (or other similar Destroy functions).

Examples

```
contentObject = OsdBasicChip.CreateContentObject( chipObject, "OsdBasicText", "mycontent" )
```

See Also

None

OsdBasicChip.RemoveAllLegendItems

Brief

Removes all legend items.

Definition

```
OsdBasicChip.RemoveAllLegendItems( OsdBasicChip chipObject )
```

Arguments

chipObject - Object to which the function is applied.

Return Values

None.

Description

Removes all legend items.

Examples

```
OsdBasicChip.RemoveAllLegendItems( chipObject )
```

See Also

None

OsdBasicChip.SetBannerHeadline

Brief

Set the headline text of an OsdBasicChip object's banner.

Definition

```
OsdBasicChip.SetBannerHeadline( OsdBasicChip chipObject, string headline )
```

Arguments

chipObject - Object to which the function is applied.headline - Headline to display on the banner.

Return Values

None.

Description

Set the headline text of an OsdBasicChip object's banner.

Examples

```
me = Object.GetMe()

-- Localized in Object Editor as "BOWLING"
headline = Object.GetLocalizedText( me, "HEADLINE_BOWLING" )

OsdBasicChip.SetBannerHeadline( chipObject, headline )
```

See Also

- Object.GetLocalizedText
- Object.GetMe

OsdBasicChip.SetBannerIconBackTexture

Brief

Set the background icon texture of an OsdBasicChip object's banner.

Definition

```
OsdBasicChip.SetBannerIconBackTexture( OsdBasicChip chipObject, Texture textureObject )
```

Arguments

chipObject - Object to which the function is applied.textureObject - Background texture object.

Return Values

None.

Description

Set the background icon texture of an OsdBasicChip object's banner.

Examples

```
OsdBasicChip.SetBannerIconBackTexture( chipObject, textureObject )
```

See Also

None

OsdBasicChip.SetBannerIconFrontTexture

Brief

Set the foreground icon texture of an OsdBasicChip object's banner.

Definition

```
OsdBasicChip.SetBannerIconFrontTexture( OsdBasicChip chipObject, Texture textureObject )
```

Arguments

chipObject - Object to which the function is applied.textureObject - Foreground texture object.

Return Values

None.

Description

Set the foreground icon texture of an OsdBasicChip object's banner.

Examples

```
OsdBasicChip.SetBannerIconFrontTexture( chipObject, textureObject )
```

See Also

None

OsdBasicChip.SetBannerIconVisible

Brief

Set the visibility of an OsdBasicChip object's banner icon.

Definition

```
OsdBasicChip.SetBannerIconVisible( OsdBasicChip chipObject, boolean setVisible )
```

Arguments

chipObject - Object to which the function is applied.setVisible - true if you want the banner icon to be visible, false otherwise.

Return Values

None.

Description

Set the visibility of an OsdBasicChip object's banner icon. The icon is invisible by default.

Examples

```
OsdBasicChip.SetBannerIconVisible( chipObject, true )
```

See Also

None

OsdBasicChip.SetBannerSubheader

Brief

Set the subheader text of an OsdBasicChip object's banner.

Definition

```
OsdBasicChip.SetBannerSubheader( OsdBasicChip chipObject, string subheader )
```

Arguments

chipObject - Object to which the function is applied.subheader - Subheader to display on the banner.

Return Values

None.

Description

Set the subheader text of an OsdBasicChip object's banner.

Examples

```
me = Object.GetMe()

-- Localized in Object Editor as "Start bowling"
subheader = Object.GetLocalizedText( me, "SUBHEADER_BOWLING" )

OsdBasicChip.SetBannerSubheader( chipObject, subheader )
```

See Also

- Object.GetLocalizedText
- Object.GetMe

OsdBasicChip.SetBannerWantVisible

Brief

Sets the chip's banner to be visible or not.

Definition

```
OsdBasicChip.SetBannerWantVisible( OsdBasicChip chipObject, boolean wantVisible )
```

Arguments

chipObject - Object to which the function is applied.wantVisible - true if you want the banner to be visible, false otherwise.

Return Values

None.

Description

Sets the chip's banner to be visible or not.

Examples

```
OsdBasicChip.SetBannerWantVisible( chipObject, true )
```

See Also

None

OsdBasicChip.SetContentHeight

Brief

Set the height of the chip's content object.

Definition

```
OsdBasicChip.SetContentHeight( OsdBasicChip chipObject, number contentHeight )
```

Arguments

chipObject - Object to which the function is applied.contentHeight - Desired height of chip's content object.

Return Values

None.

Description

Set the height of the chip's content object.

Examples

```
OsdBasicChip.SetContentHeight( chipObject, 20 )
```

See Also

- OsdBasicChip.SetContentHeightSetFromChip
- OsdBasicChip.SetContentHeightSetFromChildren

OsdBasicChip.SetContentHeightSetFromChildren

Brief

Set the height of the chip's content object to be set by its children.

Definition

```
OsdBasicChip.SetContentHeightSetFromChildren( OsdBasicChip chipObject )
```

Arguments

chipObject - Object to which the function is applied.

Return Values

None.

Description

Set the height of the chip's content object to be set by its children (i.e. the actual chip content).

Examples

```
OsdBasicChip.SetContentHeightSetFromChildren( chipObject )
```

See Also

- OsdBasicChip.SetContentHeightSetFromChip
- OsdBasicChip.SetContentHeight

OsdBasicChip.SetContentHeightSetFromChip

Brief

Set the height of the chip's content object to be set by the chip.

Definition

```
OsdBasicChip.SetContentHeightSetFromChip( OsdBasicChip chipObject )
```

Arguments

chipObject - Object to which the function is applied.

Return Values

None.

Description

Set the height of the chip's content object to be set by the chip.

Examples

```
OsdBasicChip.SetContentHeightSetFromChip( chipObject )
```

See Also

- OsdBasicChip.SetContentHeightSetFromChildren
- OsdBasicChip.SetContentHeight

OsdBasicChip.SetContentObjectInFocus

Brief

Focuses on specified content object.

Definition

```
OsdBasicChip.SetContentObjectInFocus( OsdBasicChip chipObject, OsdObject contentObject )
```

Arguments

chipObject - Object to which the function is applied.contentObject - The content object to focus on.

Return Values

None.

Description

Focuses on specified content object. Moves all content so the specified object appears as near to center as possible.

Examples

```
OsdBasicChip.SetContentObjectInFocus( chipObject, contentObject )
```

See Also

None

OsdBasicChip.SetContentPointInFocus

Brief

Focuses on specified point in content space.

Definition

```
OsdBasicChip.SetContentPointInFocus( OsdBasicChip chipObject, enum pointOrigin, number pointOffsetX =
0.0, number pointOffsetY = 0.0 )
```

Arguments

chipObject - Object to which the function is applied.pointOrigin - An enum representing the local-(ie, content-)space origin of the point to focus on. Can be one of the following:
OsdAlign.Center
OsdAlign.Top
OsdAlign.Bottom
OsdAlign.Left
OsdAlign.Right
OsdAlign.TopLeft
OsdAlign.TopRight
OsdAlign.BottomLeft
OsdAlign.BottomRightpointOffsetX - The local (content) space x-offset of the point to focus on.pointOffsetY - The local (content) space y-offset of the point to focus on.

Return Values

None.

Description

Focuses on specified point in content space. Moves all content so the specified point appears as near to the center of the window as possible.

Examples

```
-- Focuses on a point 40 pixels to the left and 20 pixels down from the top right corner of the
content
OsdBasicChip.SetContentPointInFocus( chipObject, OsdAlign.TopRight, -40, 20 )
```

See Also

None

OsdBasicChip.SetContentWidth

Brief

Set the width of the chip's content object.

Definition

```
OsdBasicChip.SetContentWidth( OsdBasicChip chipObject, number contentWidth )
```

Arguments

chipObject - Object to which the function is applied.contentWidth - Desired width of chip's content object.

Return Values

None.

Description

Set the width of the chip's content object.

Examples

```
OsdBasicChip.SetContentWidth( chipObject, 20 )
```

See Also

- OsdBasicChip.SetContentWidthSetFromChip
- OsdBasicChip.SetContentWidthSetFromChildren

OsdBasicChip.SetContentWidthSetFromChildren

Brief

Set the width of the chip's content object to be set by its children.

Definition

```
OsdBasicChip.SetContentWidthSetFromChildren(OsdBasicChip chipObject)
```

Arguments

chipObject - Object to which the function is applied.

Return Values

None.

Description

Set the width of the chip's content object to be set by its children (i.e. the actual chip content).

Examples

```
OsdBasicChip.SetContentWidthSetFromChildren( chipObject )
```

See Also

- OsdBasicChip.SetContentWidthSetFromChip
- OsdBasicChip.SetContentWidth

OsdBasicChip.SetContentWidthSetFromChip

Brief

Set the width of the chip's content object to be set by the chip.

Definition

```
OsdBasicChip.SetContentWidthSetFromChip( OsdBasicChip chipObject )
```

Arguments

chipObject - Object to which the function is applied.

Return Values

None.

Description

Set the width of the chip's content object to be set by the chip.

Examples

```
OsdBasicChip.SetContentWidthSetFromChip( chipObject )
```

See Also

- OsdBasicChip.SetContentWidthSetFromChildren
- OsdBasicChip.SetContentWidth

OsdBasicChip.SetHeightSetFromContent

Brief

Set the height of the chip to be decided by its content.

Definition

```
OsdBasicChip.SetHeightSetFromContent( OsdBasicChip chipObject, boolean isHeightSetFromContent )
```

Arguments

chipObject - Object to which the function is applied.isHeightSetFromContent - true if you want the height of the chip to be decided by its content, false otherwise.

Return Values

None.

Description

Set the height of the chip to be decided by its content.

Examples

```
OsdBasicChip.SetSize( chipObject, true )
```

See Also

None

OsdBasicChip.SetSize

Brief

Set the width of an OsdBasicChip object to a preset size.

Definition

```
OsdBasicChip.SetSize( OsdBasicChip chipObject, string size )
```

Arguments

chipObject - Object to which the function is applied.size - A string representing the size of the chip. Can be one of the following:
"quarter"
"half"
"twothirds"
"full"

Return Values

None.

Description

Set the width of an OsdBasicChip object to a preset size.

Examples

```
OsdBasicChip.SetSize( chipObject, "half" )
```

See Also

None

OsdBasicChip.UpdateContentHeight

Brief

Updates the height of the content object.

Definition

```
OsdBasicChip.UpdateContentHeight( OsdBasicChip chipObject )
```

Arguments

chipObject - Object to which the function is applied.

Return Values

None.

Description

Updates the height of the content object. To be called after adding all content if the content object height depends on its contents.

Examples

```
OsdBasicChip.UpdateContentHeight( chipObject )
```

See Also

None

OsdBasicChip.UpdateContentWidth

Brief

Updates the width of the content object.

Definition

```
OsdBasicChip.UpdateContentWidth( OsdBasicChip chipObject )
```

Arguments

chipObject - Object to which the function is applied.

Return Values

None.

Description

Updates the width of the content object. To be called after adding all content if the content object width depends on its contents.

Examples

```
OsdBasicChip.UpdateContentWidth( chipObject )
```

See Also

None

OsdBasicLegend

OsdBasicLegend.AddItem

Brief

Add a legend item to a basic legend object.

Definition

```
OsdBasicLegend.AddItem( OsdBasicLegend legendObject, string itemText, number itemId )
```

Arguments

legendObject - The basic legend object to which to add the items.itemText - The text of the legend item.itemId - The unique id of the legend item (optional parameter).

Return Values

None.

Description

Add a legend item to a basic legend object.

Examples

```
me = Object.GetMe()

-- Localized in Object Editor as "[CROSS] Continue"
legendText1 = Object.GetLocalizedText( me, "LEGEND_CONTINUE" )
-- Localized in Object Editor as "[CIRCLE] Cancel"
legendText2 = Object.GetLocalizedText( me, "LEGEND_CANCEL" )
-- Localized in Object Editor as "[SQUARE] Help"
legendText3 = Object.GetLocalizedText( me, "LEGEND_HELP" )

OsdBasicLegend.BeginAddingItems( legendObject )
OsdBasicLegend.AddItem( legendObject, legendText1, 1 )
OsdBasicLegend.AddItem( legendObject, legendText2, 2 )
OsdBasicLegend.AddItem( legendObject, legendText3, 3 )
OsdBasicLegend.EndAddingItems( legendObject )
```

See Also

- Object.GetLocalizedText
- Object.GetMe

OsdBasicLegend.BeginAddingItems

Brief

Begin adding legend items to a basic legend object.

Definition

```
OsdBasicLegend.BeginAddingItems( OsdBasicLegend legendObject )
```

Arguments

legendObject - The basic legend object to which to begin adding items.

Return Values

None.

Description

Begin adding legend items to a basic legend object. When adding legend items to a basic legend object, calling this method optimizes the process by delaying some post-processing until EndAddingItems is called.

Examples

```
me = Object.GetMe()

-- Localized in Object Editor as "[CROSS] Continue"
legendText1 = Object.GetLocalizedText( me, "LEGEND_CONTINUE" )
-- Localized in Object Editor as "[CIRCLE] Cancel"
legendText2 = Object.GetLocalizedText( me, "LEGEND_CANCEL" )
-- Localized in Object Editor as "[SQUARE] Help"
legendText3 = Object.GetLocalizedText( me, "LEGEND_HELP" )

OsdBasicLegend.BeginAddingItems( legendObject )
OsdBasicLegend.AddItem( legendObject, legendText1, 1 )
OsdBasicLegend.AddItem( legendObject, legendText2, 2 )
OsdBasicLegend.AddItem( legendObject, legendText3, 3 )
OsdBasicLegend.EndAddingItems( legendObject )
```

See Also

- Object.GetLocalizedText
- Object.GetMe

OsdBasicLegend.EndAddingItems

Brief

End adding legend items to a basic legend object.

Definition

```
OsdBasicLegend.EndAddingItems( OsdBasicLegend legendObject )
```

Arguments

legendObject - The basic legend object to which to end adding items.

Return Values

None.

Description

End adding legend items to a basic legend object. When adding legend items to a basic legend object, calling BeginAddingItems optimizes the process by delaying some post-processing until this method is called.

Examples

```
me = Object.GetMe()

-- Localized in Object Editor as "[CROSS] Continue"
legendText1 = Object.GetLocalizedText( me, "LEGEND_CONTINUE" )
-- Localized in Object Editor as "[CIRCLE] Cancel"
legendText2 = Object.GetLocalizedText( me, "LEGEND_CANCEL" )
-- Localized in Object Editor as "[SQUARE] Help"
legendText3 = Object.GetLocalizedText( me, "LEGEND_HELP" )

OsdBasicLegend.BeginAddingItems( legendObject )
OsdBasicLegend.AddItem( legendObject, legendText1, 1 )
OsdBasicLegend.AddItem( legendObject, legendText2, 2 )
OsdBasicLegend.AddItem( legendObject, legendText3, 3 )
OsdBasicLegend.EndAddingItems( legendObject )
```

See Also

- Object.GetLocalizedText
- Object.GetMe

OsdBasicLegend.RemoveAllItems

Brief

Removes all legend items from a basic legend object.

Definition

```
OsdBasicLegend.RemoveAllItems( OsdBasicLegend legendObject )
```

Arguments

legendObject - The basic legend object from which to remove all legend items.

Return Values

None.

Description

Removes all legend items from a basic legend object.

Examples

```
OsdBasicLegend.RemoveAllItems( legendObject )
```

See Also

None

OsdBasicLegend.SetBorderWidth

Brief

Set the width of the border between the left-/right-most legend items and the left/right edges of a basic legend object.

Definition

```
OsdBasicLegend.SetBorderWidth( OsdBasicLegend legendObject, number borderWidth )
```

Arguments

legendObject - The basic legend object on which to set the border width.borderWidth - The border-width of the legend object.

Return Values

None.

Description

Set the width of the border between the left-/right-most legend items and the left/right edges of a basic legend object.

Examples

```
OsdBasicLegend.SetBorderWidth( legendObject, 50.0 )
```

See Also

None

OsdBasicLegend.SetFixedWidth

Brief

Set the width of a basic legend object.

Definition

```
OsdBasicLegend.SetFixedWidth( OsdBasicLegend legendObject, number fixedWidth )
```

Arguments

legendObject - The basic legend object on which to set the maximum width.fixedWidth - The width of the legend object.

Return Values

None.

Description

Set the width of a basic legend object. Basic legend objects can resize themselves automatically based on the legend items present. This method sets the width of the object to be fixed (i.e. switches off the automatic resizing). If the legend items present demand a larger width to be completely displayed, the displayed legend items will cycle automatically.

Examples

```
OsdBasicLegend.SetFixedWidth( legendObject, 400.0 )
```

See Also

None

OsdBasicLegend.SetItemSpacing

Brief

Set the spacing between the legend items of a basic legend object.

Definition

```
OsdBasicLegend.SetItemSpacing( OsdBasicLegend legendObject, number itemSpacing )
```

Arguments

legendObject - The basic legend object on which to set the item spacing.itemSpacing - The item-spacing of the legend object.

Return Values

None.

Description

Set the spacing between the legend items of a basic legend object.

Examples

```
OsdBasicLegend.SetItemSpacing( legendObject, 50.0 )
```

See Also

None

OsdBasicLegend.SetMaxWidth

Brief

Set the maximum width of a basic legend object.

Definition

```
OsdBasicLegend.SetMaxWidth( OsdBasicLegend legendObject, number maxWidth )
```

Arguments

legendObject - The basic legend object on which to set the maximum width.maxWidth - The maximum width of the legend object.

Return Values

None.

Description

Set the maximum width of a basic legend object. Basic legend objects can resize themselves automatically based on the legend items present. This method sets a maximum width for that resizing. If the legend items present demand a larger width to be completely displayed, the displayed legend items will cycle automatically.

Examples

```
OsdBasicLegend.SetMaxWidth( legendObject, 600.0 )
```

See Also

None

OsdBasicLegend.SetMinWidth

Brief

Set the minimum width of a basic legend object.

Definition

```
OsdBasicLegend.SetMinWidth( OsdBasicLegend legendObject, number minWidth )
```

Arguments

legendObject - The basic legend object on which to set the minimum width.minWidth - The minimum width of the legend object.legendObject - The basic legend object on which to set the minimum width.minWidth - The minimum width of the legend object.

Return Values

None.

Description

Set the minimum width of a basic legend object. Basic legend objects can resize themselves automatically based on the legend items present. This method sets a minimum width for that resizing.

Examples

```
OsdBasicLegend.SetMinWidth( legendObject, 200.0 )
```

See Also

None

OsdBasicLegend.SetText

Brief

Set the text of a basic legend object.

Definition

```
OsdBasicLegend.SetText( OsdBasicLegend legendObject, string text )
```

Arguments

legendObject - The basic legend object on which to set the text.text - The localized text to display on the legend.

Return Values

None.

Description

Set the text of a basic legend object. Basic legend objects support multiple legend items. Calling this method removes all current legend items and adds a new one with the specified text.

Examples

```
me = Object.GetMe()

-- Localized in Object Editor as "[CROSS] Continue"
legendText = Object.GetLocalizedText( me, "LEGEND_CONTINUE" )

OsdBasicLegend.SetText( legendObject, legendText )
```

See Also

- Object.GetLocalizedText
- Object.GetMe

OsdBasicMenu

OsdBasicMenu.AddItem

Brief

Add a menu item to the menu object.

Definition

```
variant OsdBasicMenu.AddItem( OsdBasicMenu self, string type, number id = nil, boolean setEnabled =
true, number beforeId = nil )
```

Arguments

self - The menu object for which to call the method.type - The OSD object type of the menu item.id - Non-negative unique (in this menu) identifier for the menu item, a negative value causes the item to be allocated an id by the system.setEnabled - true to have the menu item enabled (selectable) initially, false otherwise.beforeId - the id of the existing menu item before which to insert the new one.

Return Values

The newly added OsdObject-derived menu item.

Description

Add a menu item to the menu object. The item will be added to the bottom of the menu. AddItem should be called between calls to BeginChanges and EndChanges. If the 'beforeId' argument is negative or nil, the new item will be added to the end of the menu. Note: Garbage collection will not destroy OSD objects. OSD objects are only destroyed either when the parent OSD object is destroyed, or by explicitly calling OsdObject.DestroyObject (or other similar Destroy functions).

Examples

```
menuObject = OsdObject.CreateChildObject( parentObject, "OsdBasicMenu", "menu" )
OsdBasicMenu.BeginChanges( menuObject )
itemObject = OsdBasicMenu.AddItem( menuObject, "OsdBasicSingleGraphic", 0, true )
OsdBasicSingleGraphic.SetBackTexture( itemObject, textureObject )
itemObject = OsdBasicMenu.AddItem( menuObject, "OsdCommon9ComboLayer3", 1, true )
OsdCommon9ComboLayer3.SetWidth( itemObject, 200.0 )
OsdCommon9ComboLayer3.SetHeight( itemObject, 100.0 )
OsdBasicMenu.EndChanges( menuObject )
```

See Also

None

OsdBasicMenu.BeginChanges

Brief

Prepares the menu to receive changes, such as the addition of menu items.

Definition

```
OsdBasicMenu.BeginChanges( OsdBasicMenu self )
```

Arguments

self - The menu object for which to call the method.

Return Values

None.

Description

Prepares the menu to receive changes, such as the addition of menu items. A corresponding call to EndChanges is required on completion of the changes.

Examples

```
menuObject = OsdObject.CreateChildObject( parentObject, "OsdBasicMenu", "menu" )
OsdBasicMenu.BeginChanges( menuObject )
OsdBasicMenu.AddItem( menuObject, "OsdBasicSingleGraphic", 0, true )
OsdBasicMenu.AddItem( menuObject, "OsdCommon9ComboLayer3", 1, true )
OsdBasicMenu.EndChanges( menuObject )
```

See Also

None

OsdBasicMenu.EndChanges

Brief

Tell the menu object that there are no more changes, such as adding items.

Definition

```
OsdBasicMenu.EndChanges( OsdBasicMenu self )
```

Arguments

self - The menu object for which to call the method.

Return Values

None.

Description

Tell the menu object that there are no more changes, such as adding items.

Examples

```
menuObject = OsdObject.CreateChildObject( parentObject, "OsdBasicMenu", "menu" )
OsdBasicMenu.BeginChanges( menuObject )
OsdBasicMenu.AddItem( menuObject, "OsdBasicSingleGraphic", 0, true )
OsdBasicMenu.AddItem( menuObject, "OsdCommon9ComboLayer3", 1, true )
OsdBasicMenu.EndChanges( menuObject )
```

See Also

None

OsdBasicMenu.GetCurrentItemId

Brief

Get the id of the currently selected item.

Definition

```
OsdBasicMenu.GetCurrentItemId( OsdBasicMenu self )
```

Arguments

self - The menu object for which to call the method.

Return Values

The id of the currently selected item.

Description

Get the id of the currently selected item. Assumes there exists a currently selected item, so call OsdBasicMenu.IsCurrentItemSet first.

Examples

```
menuObject = OsdObject.CreateChildObject( parentObject, "OsdBasicMenu", "menu" )
OsdBasicMenu.BeginChanges( menuObject )
OsdBasicMenu.AddItem( menuObject, "OsdBasicSingleGraphic", 0, true )
OsdBasicMenu.AddItem( menuObject, "OsdCommon9ComboLayer3", 1, true )
OsdBasicMenu.EndChanges( menuObject )
if OsdBasicMenu.IsCurrentItemSet( menuObject ) then
    id = OsdBasicMenu.GetCurrentItemId( menuObject )
    OsdBasicMenu.SetItemEnabled( menuObject, id, false )
end
```

See Also

None

OsdBasicMenu.GetItemById

Brief

Get the menu item corresponding to the specified unique id.

Definition

```
variant OsdBasicMenu.GetItemById( OsdBasicMenu self, number id )
```

Arguments

self - The menu object for which to call the method. id - The unique id of the menu item to retrieve.

Return Values

The OsdObject-derived menu item corresponding to the specified unique id.

Description

Get the menu item corresponding to the specified unique id.

Examples

```
menuObject = OsdObject.CreateChildObject( parentObject, "OsdBasicMenu", "menu" )
OsdBasicMenu.BeginChanges( menuObject )
OsdBasicMenu.AddItem( menuObject, "OsdBasicSingleGraphic", 0, true )
OsdBasicMenu.AddItem( menuObject, "OsdCommon9ComboLayer3", 1, true )
OsdBasicMenu.EndChanges( menuObject )
if OsdBasicMenu.IsCurrentItemSet( menuObject ) then
    id = OsdBasicMenu.GetCurrentItemId( menuObject )
    item = OsdBasicMenu.GetItemById( menuObject, id )
    OsdObject.SetAlphaType( item, "half" )
end
```

See Also

None

OsdBasicMenu.IsCurrentItemSet

Brief

Query whether or not the menu has a currently selected item.

Definition

```
OsdBasicMenu.IsCurrentItemSet( OsdBasicMenu self )
```

Arguments

self - The menu object for which to call the method.

Return Values

true if the menu has a currently selected item, false otherwise.

Description

Query whether or not the menu has a currently selected item. If the menu has at least one enabled item, it should have a currently selected item.

Examples

```
menuObject = OsdObject.CreateChildObject( parentObject, "OsdBasicMenu", "menu" )
OsdBasicMenu.BeginChanges( menuObject )
OsdBasicMenu.AddItem( menuObject, "OsdBasicSingleGraphic", 0, true )
OsdBasicMenu.AddItem( menuObject, "OsdCommon9ComboLayer3", 1, true )
OsdBasicMenu.EndChanges( menuObject )
if not OsdBasicMenu.IsCurrentItemSet( menuObject ) then
    OsdBasicMenu.SetCurrentItem( menuObject, 0 )
end
```

See Also

None

OsdBasicMenu.OnInputDown

Brief

Tell the menu to react to the "down" button being pressed.

Definition

```
OsdBasicMenu.OnInputDown( OsdBasicMenu self )
```

Arguments

self - The menu object for which to call the method.

Return Values

None.

Description

Tell the menu to react to the "down" button being pressed.

Examples

```
menuObject = OsdObject.CreateChildObject( parentObject, "OsdBasicMenu", "menu" )
OsdBasicMenu.BeginChanges( menuObject )
OsdBasicMenu.AddItem( menuObject, "OsdBasicSingleGraphic", 0, true )
OsdBasicMenu.AddItem( menuObject, "OsdCommon9ComboLayer3", 1, true )
OsdBasicMenu.EndChanges( menuObject )
OsdBasicMenu.OnInputDown( menuObject )
OsdBasicMenu.OnInputUp( menuObject )
OsdBasicMenu.OnInputSelect( menuObject )
```

See Also

None

OsdBasicMenu.OnInputUp

Brief

Tell the menu to react to the "up" button being pressed.

Definition

```
OsdBasicMenu.OnInputUp( OsdBasicMenu self )
```

Arguments

self - The menu object for which to call the method.

Return Values

None.

Description

Tell the menu to react to the "up" button being pressed.

Examples

```
menuObject = OsdObject.CreateChildObject( parentObject, "OsdBasicMenu", "menu" )
OsdBasicMenu.BeginChanges( menuObject )
OsdBasicMenu.AddItem( menuObject, "OsdBasicSingleGraphic", 0, true )
OsdBasicMenu.AddItem( menuObject, "OsdCommon9ComboLayer3", 1, true )
OsdBasicMenu.EndChanges( menuObject )
OsdBasicMenu.OnInputDown( menuObject )
OsdBasicMenu.OnInputUp( menuObject )
OsdBasicMenu.OnInputSelect( menuObject )
```

See Also

None

OsdBasicMenu.Reset

Brief

Removes and destroys all menu items.

Definition

```
OsdBasicMenu.Reset( OsdBasicMenu self )
```

Arguments

self - The menu object for which to call the method.

Return Values

None.

Description

Removes and destroys all menu items.

Examples

```
menuObject = OsdObject.CreateChildObject( parentObject, "OsdBasicMenu", "menu" )
OsdBasicMenu.BeginChanges( menuObject )
OsdBasicMenu.AddItem( menuObject, "OsdBasicSingleGraphic", 0, true )
OsdBasicMenu.AddItem( menuObject, "OsdCommon9ComboLayer3", 1, true )
OsdBasicMenu.EndChanges( menuObject )
OsdBasicMenu.Reset( menuObject )
```

See Also

None

OsdBasicMenu.SetCurrentItem

Brief

Set the specified menu item to be the currently selected one.

Definition

```
OsdBasicMenu.SetCurrentItem( OsdBasicMenu self, number itemId )
```

Arguments

self - The menu object for which to call the method.itemId - The unique id of the menu item.

Return Values

None.

Description

Set the specified menu item to be the currently selected one. Mainly for initialisation; the OnInput... methods should be used to change the currently selected menu item after that.

Examples

```
menuObject = OsdObject.CreateChildObject( parentObject, "OsdBasicMenu", "menu" )
OsdBasicMenu.BeginChanges( menuObject )
OsdBasicMenu.AddItem( menuObject, "OsdBasicSingleGraphic", 0, true )
OsdBasicMenu.AddItem( menuObject, "OsdCommon9ComboLayer3", 1, true )
OsdBasicMenu.EndChanges( menuObject )
OsdBasicMenu.SetCurrentItem( menuObject, 0 )
```

See Also

None

OsdBasicMenu.SetItemEnabled

Brief

Set the specified menu item to be enabled/disabled.

Definition

```
OsdBasicMenu.SetItemEnabled( OsdBasicMenu self, number itemId, boolean setEnabled )
```

Arguments

self - The menu object for which to call the method.itemId - The unique id of the menu item.setEnabled - true if enabling the item, false otherwise.

Return Values

None.

Description

Set the specified menu item to be enabled/disabled.

Examples

```
menuObject = OsdObject.CreateChildObject( parentObject, "OsdBasicMenu", "menu" )
OsdBasicMenu.BeginChanges( menuObject )
OsdBasicMenu.AddItem( menuObject, "OsdBasicSingleGraphic", 0, true )
OsdBasicMenu.AddItem( menuObject, "OsdCommon9ComboLayer3", 1, true )
OsdBasicMenu.EndChanges( menuObject )
OsdBasicMenu.SetItemEnabled( menuObject, 1, false )
```

See Also

None

OsdBasicMenu.SetItemSpacing

Brief

Set the vertical spacing between menu items.

Definition

```
OsdBasicMenu.SetItemSpacing( OsdBasicMenu self, number itemSpacing )
```

Arguments

self - The menu object for which to call the method.itemSpacing - The spacing in pixels.

Return Values

None.

Description

Set the vertical spacing between menu items. Should be called between calls to BeginChanges and EndChanges. Can be called before or after adding menu items.

Examples

```
menuObject = OsdObject.CreateChildObject( parentObject, "OsdBasicMenu", "menu" )
OsdBasicMenu.BeginChanges( menuObject )
OsdBasicMenu.SetItemSpacing( menuObject, 40.0 )
OsdBasicMenu.AddItem( menuObject, "OsdBasicSingleGraphic", 0, true )
OsdBasicMenu.AddItem( menuObject, "OsdCommon9ComboLayer3", 1, true )
OsdBasicMenu.EndChanges( menuObject )
```

See Also

None

OsdBasicMenu.SetOnSetItemEnabledCallback

Brief

Set the lua callback function to be called when a menu item is enabled/disabled.

Definition

```
OsdBasicMenu.SetOnSetItemEnabledCallback( OsdBasicMenu self, function callback )
```

Arguments

self - The menu object for which to call the method.callback - The callback function.

Return Values

None.

Description

Set the lua callback function to be called when a menu item is enabled/disabled. The function should have two parameters; an OsdObject (the item) and a boolean (true if the item is being enabled, false otherwise).

Examples

```
function GameMenu.SetItemEnabledCallback( item, setEnabled )
    if setEnabled then
        OsdObject.SetColorType( item, "fg" )
    else
        OsdObject.SetColorType( item, "grey" )
    end
end

function GameMenu.Create( parentObject )
    menuObject = OsdObject.CreateChildObject( parentObject, "OsdBasicMenu", "menu" )
    OsdBasicMenu.SetOnSetItemEnabledCallback( menuObject, GameMenu.SetItemEnabledCallback )
    OsdBasicMenu.BeginChanges( menuObject )
    OsdBasicMenu.SetItemSpacing( menuObject, 40.0 )
    OsdBasicMenu.AddItem( menuObject, "OsdBasicSingleGraphic", 0, true )
    OsdBasicMenu.AddItem( menuObject, "OsdCommon9ComboLayer3", 1, true )
    OsdBasicMenu.EndChanges( menuObject )
end
```

See Also

None

OsdBasicMenu.SetOnSetItemSelectedCallback

Brief

Set the lua callback function to be called when a menu item is selected/unselected.

Definition

```
OsdBasicMenu.SetOnSetItemSelectedCallback( OsdBasicMenu self, function callback )
```

Arguments

self - The menu object for which to call the method.callback - The callback function.

Return Values

None.

Description

Set the lua callback function to be called when a menu item is selected/unselected. The function should have two parameters; an OsdObject (the item) and a boolean (true if the item is being selected, false otherwise).

Examples

```
function GameMenu.SetItemSelectedCallback( item, setSelected )
    if setSelected then
        OsdObject.SetAlphaType( item, "normal" )
    else
        OsdObject.SetAlphaType( item, "half" )
    end
end
function GameMenu.Create( parentObject )
    menuObject = OsdObject.CreateChildObject( parentObject, "OsdBasicMenu", "menu" )
    OsdBasicMenu.SetOnSetItemSelectedCallback( menuObject, GameMenu.SetItemSelectedCallback )
    OsdBasicMenu.BeginChanges( menuObject )
    OsdBasicMenu.SetItemSpacing( menuObject, 40.0 )
    OsdBasicMenu.AddItem( menuObject, "OsdBasicSingleGraphic", 0, true )
    OsdBasicMenu.AddItem( menuObject, "OsdCommon9ComboLayer3", 1, true )
    OsdBasicMenu.EndChanges( menuObject )
end
```

See Also

None

OsdBasicPopup

OsdBasicPopup.SetMessage

Brief

Set the message text on a basic popup object.

Definition

```
OsdBasicPopup.SetMessage( OsdBasicPopup popupObject, string messageText )
```

Arguments

popupObject - The basic popup object for which to set the title text.messageText - The localized message text.

Return Values

None.

Description

Set the message text on a basic popup object. Basic popup objects consist of title text and message text.

Examples

```
me = Object.GetMe()

-- Localized in Object Editor as "WARNING!"
title   = Object.GetLocalizedText( me, "WARNING_POPUP_TITLE" )
-- Localized in Object Editor as "Something has gone wrong!"
message = Object.GetLocalizedText( me, "WARNING_POPUP_MESSAGE" )

OsdBasicPopup.SetTitle( popup, title )
OsdBasicPopup.SetMessage( popup, message )
```

See Also

- Object.GetLocalizedText
- Object.GetMe

OsdBasicPopup.SetTitle

Brief

Set the title text on a basic popup object.

Definition

```
OsdBasicPopup.SetTitle( OsdBasicPopup popupObject, string titleText )
```

Arguments

popupObject - The basic popup object for which to set the title text.titleText - The localized title text.

Return Values

None.

Description

Set the title text on a basic popup object. Basic popup objects consist of title text and message text.

Examples

```
me = Object.GetMe()

-- Localized in Object Editor as "WARNING!"
title   = Object.GetLocalizedText( me, "WARNING_POPUP_TITLE" )
-- Localized in Object Editor as "Something has gone wrong!"
message = Object.GetLocalizedText( me, "WARNING_POPUP_MESSAGE" )

OsdBasicPopup.SetTitle( popup, title )
OsdBasicPopup.SetMessage( popup, message )
```

See Also

- Object.GetLocalizedText
- Object.GetMe

OsdBasicSingleGraphic

OsdBasicSingleGraphic.SetBackColorType

Brief

Set the background color type for a basic single graphic object.

Definition

```
OsdBasicSingleGraphic.SetBackColorType( OsdBasicSingleGraphic graphicObject, enum colorType )
```

Arguments

graphicObject - The basic single graphic for which to set the background color.colorType - An enum representing the color type. Can be one of the following:
OsdColor.Black
OsdColor.White
OsdColor.Grey
OsdColor.Green
OsdColor.Yellow
OsdColor.Red
OsdColor.Normal
OsdColor.Background
OsdColor.OppositeBg
OsdColor.Foreground
OsdColor.HighlightFg

Return Values

None.

Description

Set the background color type for a basic single graphic object.

Examples

```
OsdBasicSingleGraphic.SetBackColorType( graphicObject, OsdColor.Background )
```

See Also

- OsdBasicSingleGraphic.SetFrontColorType

OsdBasicSingleGraphic.SetBackTexture

Brief

Set the background texture on an OsdBasicSingleGraphic object.

Definition

```
OsdBasicSingleGraphic.SetBackTexture( OsdBasicSingleGraphic graphic, Texture texture )
```

Arguments

graphic - The OsdBasicSingleGraphic object to affect.texture - The Texture object representing the texture to set as the back texture.

Return Values

None.

Description

Set the background texture on an OsdBasicSingleGraphic object.

Examples

```
texture = Texture.Find( "user_interface_back_image" )
OsdBasicSingleGraphic.SetBackTexture( graphic, texture )
```

See Also

- OsdBasicSingleGraphic.SetFrontTexture
- Texture.Find

OsdBasicSingleGraphic.SetFrontColorType

Brief

Set the foreground color type for a basic single graphic object.

Definition

```
OsdBasicSingleGraphic.SetFrontColorType( OsdBasicSingleGraphic graphicObject, enum colorType )
```

Arguments

graphicObject - The basic single graphic for which to set the foreground color.colorType - An enum representing the color type. Can be one of the following:
OsdColor.Black
OsdColor.White
OsdColor.Grey
OsdColor.Green
OsdColor.Yellow
OsdColor.Red
OsdColor.Normal
OsdColor.Background
OsdColor.OppositeBg
OsdColor.Foreground
OsdColor.HighlightFg

Return Values

None.

Description

Set the foreground color type for a basic single graphic object.

Examples

```
OsdBasicSingleGraphic.SetFrontColorType( graphicObject, OsdColor.Foreground )
```

See Also

- OsdBasicSingleGraphic.SetBackColorType

OsdBasicSingleGraphic.SetFrontTexture

Brief

Set the foreground texture on an OsdBasicSingleGraphic object.

Definition

```
OsdBasicSingleGraphic.SetFrontTexture( OsdBasicSingleGraphic graphic, Texture texture )
```

Arguments

graphic - The OsdBasicSingleGraphic object to affect.texture - The Texture object representing the texture to set as the front texture.

Return Values

None.

Description

Set the foreground texture on an OsdBasicSingleGraphic object. The texture must be a local resource of the script's object.

Examples

```
texture = Texture.Find( "user_interface_front_image" )
OsdBasicSingleGraphic.SetFrontTexture( graphic, texture )
```

See Also

- OsdBasicSingleGraphic.SetBackTexture
- Texture.Find

OsdBasicText

OsdBasicText.GetText

Brief

Get the text string stored in a text object.

Definition

```
string OsdBasicText.GetText( OsdBasicText textObject )
```

Arguments

textObject - The OsdBasicText object to query.

Return Values

The text string.

Description

Get the text string stored in a text object.

Examples

```
print( OsdBasicText.GetText( textObject ) )
```

See Also

None

OsdBasicText.SetClipping

Brief

Set the horizontal clipping are for an OsdBasicText object.

Definition

```
OsdBasicText.SetClipping( OsdBasicText textObject, number minClip, number maxClip )
```

Arguments

textObject - The OsdBasicText object to affect.minClip - An offset from the textObject's local align point representing the minimum clipping value.maxClip - An offset from the textObject's local align point representing the maximum clipping value.

Return Values

None.

Description

Set the horizontal clipping are for an OsdBasicText object.

Examples

```
OsdBasicText.SetClipping( textObject, -100, 100 )
```

See Also

None

OsdBasicText.SetFont

Brief

Set the font on an OsdBasicText object.

Definition

```
OsdBasicText.SetFont( OsdBasicText textObject, string font )
```

Arguments

textObject - The OsdBasicText object to affect.font - String representing the font. Can be one of the following:
"13reg"
"13bold"
"18reg"
"18bold"
"26reg"
"26bold"
"60bold"

Return Values

None.

Description

Set the font on an OsdBasicText object.

Examples

```
OsdBasicText.SetFont( textObject, "13reg" )
```

See Also

None

OsdBasicText.SetIgnoreSpecialCharacters

Brief

Set whether or not to ignore special characters.

Definition

```
OsdBasicText.SetIgnoreSpecialCharacters( OsdBasicText textObject, boolean setIgnore )
```

Arguments

textObject - The OsdBasicText object to affect.setIgnore - true if you want to ignore special characters, false otherwise.

Return Values

None.

Description

Set whether or not to ignore special characters. An example of ignoring special characters would be to not convert the substring "[CROSS]" to the graphic representing the cross-button of the PlayStation(R)3 controller.

Examples

```
OsdBasicText.SetIgnoreSpecialCharacters( textObject, true )
```

See Also

None

OsdBasicText.SetScrolling

Brief

Set the scrolling behaviour are for an OsdBasicText object.

Definition

```
OsdBasicText.SetScrolling( OsdBasicText textObject, string scrollType, number scrollSpeed, number
scrollDelay )
```

Arguments

textObject - The OsdBasicText object to affect.scrollType - String representing the scrolling type.scrollSpeed - Scroll speed in pixels per second.scrollDelay - Scroll delay in seconds.

Return Values

None.

Description

Set the scrolling behaviour are for an OsdBasicText object. Valid scrollType strings are:
"none" - no scrolling
"onecycle" - scroll from start to end and back once
"repeatcycle" - scroll from start to end and back and repeat
"nolimit" - scroll "infinitely" in one direction... text will loop and there may be more than once instance of the text visible, depending on text-size vs clipping-size
If scrollType is "nolimit" (ie potentially multiple instances of the text), scrollDelay is the distance (in pixels) between instances of the text.

Examples

```
OsdBasicText.SetScrolling( textObject, "repeatcycle", 20, 1 )
```

See Also

None

OsdBasicText.SetText

Brief

Set the text on an OsdBasicText object.

Definition

```
OsdBasicText.SetText( OsdBasicText textObject, string text )
```

Arguments

textObject - The OsdBasicText object to affect.text - Localized text string.

Return Values

None.

Description

Set the text on an OsdBasicText object.

Examples

```
me = Object.GetMe()

-- Localized in Object Editor as "Hello World!"
message = Object.GetLocalizedText( me, "GREETING" )

OsdBasicText.SetText( textObject, message )
```

See Also

- Object.GetLocalizedText
- Object.GetMe

OsdObject

OsdObject.Activate

Brief

Activate an OSD object.

Definition

```
OsdObject.Activate( OsdObject object, number fadeTime )
```

Arguments

object - The OSD object to activate.fadeTime - The time (in seconds) taken to fade the object in.

Return Values

None.

Description

The specified OSD object is activated, and all descendents of it are activated as well. When an OSD object is first created, it must be activated for it to be displayed. The specified fade time controls the length of time the OsdObject and all of its children take to fade into view. Note that certain OsdObject types update themselves every frame. However, this update function will only be called if the object is activated. For example, a scrolling text object will only scroll while it is activated. In order to toggle the visibility of individual children of an OsdObject, use the function OsdObject.SetVisible.

Examples

```
OsdObject.Activate( object, 1.0 )
```

See Also

- OsdObject.Deactivate
- OsdObject.SetVisible

OsdObject.CreateChildObject

Brief

Creates a new OSD object as a child of the specified parent.

Definition

```
variant OsdObject.CreateChildObject( OsdObject parentObject, string objectType, string childName )
```

Arguments

parentObject - The parent object.objectType - A string specifying the type of the child. Can either be a registered type or an XML file.childName - Name of the new child (optional - defaults to nil)childUid - Uid of the new child (optional - defaults to invalid)beforeUid - Uid of child to insert the new child before (optional - defaults to invalid)setSelectable - Set this child to be selectable (optional - defaults to false)

Return Values

The new child OsdObject-derived object.

Description

Creates a new OSD object as a child of the specified parent. While the name and uid for the new child object are optional, it is necessary to provide atleast one of them if you ever want to be be able to retrieve the child object after creation (assuming you're not storing the reference returned by this method). Note: Garbage collection will not destroy OSD objects. OSD objects are only destroyed either when the parent OSD object is destroyed, or by explicitly calling OsdObject.DestroyObject (or other similar Destroy functions).

Examples

```
childObject = OsdObject.CreateChildObject( parentObject, "OsdBasicChip", "mychild" )
```

See Also

None

OsdObject.Deactivate

Brief

Deactivate an OSD object.

Definition

```
OsdObject.Deactivate( OsdObject object, number fadeTime )
```

Arguments

object - The OSD object to deactivate.fadeTime - The time (in seconds) taken to fade the object out.

Return Values

None.

Description

The specified OSD object is deactivated, and all descendents of it are deactivated as well. The specified fade time controls the length of time the OsdObject and all of its children take to fade from view. Note that certain OsdObject types update themselves every frame. However, this update function will only be called if the object is activated. For example, a scrolling text object will only scroll while it is activated.

Examples

```
OsdObject.Deactivate( object, 1.0 )
```

See Also

- OsdObject.Activate
- OsdObject.SetVisible

OsdObject.DestroyObject

Brief

Destroys an OSD object.

Definition

```
OsdObject.DestroyObject( OsdObject object )
```

Arguments

object - Object to destroy.

Return Values

None.

Description

Destroys an OSD object. Also recursively destroys all children. Note: Garbage collection will not destroy OSD objects. OSD objects are only destroyed either when the parent OSD object is destroyed, or by explicitly calling OsdObject.DestroyObject (or other similar Destroy functions).

Examples

```
OsdObject.DestroyObject( object )
```

See Also

None

OsdObject.FindObject

Brief

Finds an OSD object by name.

Definition

```
OsdObject OsdObject.FindObject( OsdObject currObject, string longName )
```

Arguments

currObject - The object to which the long name is relative.longName - A string specifying a dos/unix-style path of names to the object.

Return Values

The OSD object found.

Description

Finds an OSD object by name.

Examples

```
-- Example 1: Find the child called "banner" of thisObject
foundObject = OsdObject.FindObject( thisObject, "banner" )

-- Example 2: Find the child called "title" of the child called "banner" of thisObject
foundObject = OsdObject.FindObject( thisObject, "banner/title" )

-- Example 3: Find the parent of thisObject
foundObject = OsdObject.FindObject( thisObject, ".." )

-- Example 4: Find the child called "legend" of parent of thisObject, i.e. the sibling called "legend"
of thisObject
foundObject = OsdObject.FindObject( thisObject, "../legend" )
```

See Also

None

OsdObject.GetColorValue

Brief

Get the color of an OSD object.

Definition

```
Vector4 OsdObject.GetColorValue( OsdObject object, Vector4 color )
```

Arguments

object - The OSD object to affect.color - A Vector4 to receive the color (RGBA).

Return Values

None.

Description

Get the color of an OSD object.

Examples

```
local v = Vector4.Create()
OsdObject.GetColorValue( object, v )
print( v )
```

See Also

- OsdObject.SetColorType
- OsdObject.SetColorValue

OsdObject.GetHeight

Brief

Get the height of an OSD object.

Definition

```
number OsdObject.GetHeight( OsdObject object )
```

Arguments

object - The OSD object to query.

Return Values

The height of the object.

Description

Get the height of an OSD object.

Examples

```
local height = OsdObject.GetHeight( object )
```

See Also

None

OsdObject.GetPositionX

Brief

Get the x-component of an object's screen position.

Definition

```
number OsdObject.GetPositionX( OsdObject object )
```

Arguments

object - Object to query.

Return Values

The x-component of the object's screen position.

Description

Get the x-component of an object's screen position.

Examples

```
local x = OsdObject.GetPositionX( object )
```

See Also

None

OsdObject.GetPositionY

Brief

Get the y-component of an object's screen position.

Definition

```
number OsdObject.GetPositionY( OsdObject object )
```

Arguments

object - Object to query.

Return Values

The y-component of the object's screen position.

Description

Get the y-component of an object's screen position.

Examples

```
local y = OsdObject.GetPositionY( object )
```

See Also

None

OsdObject.GetUniqueId

Brief

Get the unique Id of an OSD object.

Definition

```
number|nil OsdObject.GetUniqueId( OsdObject object )
```

Arguments

object - The OSD object to query.

Return Values

The unique id of the object, or nil if one has not been set.

Description

The unique id must be an integer >= 0. The id must be unique with the group of sibling objects.

Examples

```
local uid = OsdObject.GetUniqueId( object )
```

See Also

- OsdObject.SetUniqueId

OsdObject.GetWidth

Brief

Get the width (in parent-space) of an OSD object.

Definition

```
number OsdObject.GetWidth( OsdObject object )
```

Arguments

object - The OSD object to query.

Return Values

The width (in parent-space) of the object.

Description

Get the width (in parent-space) of an OSD object

Examples

```
local width = OsdObject.GetWidth( object )
```

See Also

OsdObject.HandleInput

Brief

Pass input to the OSD object.

Definition

```
boolean OsdObject.HandleInput( OsdObject parent, number inputId, number analogValue=1.0, number
elapsedTime=0.0 )
```

Arguments

inputId - An OsdInput enumeration representing the input type:

- OsdInput.Up
- OsdInput.Down
- OsdInput.Left
- OsdInput.Right
- OsdInput.Accept
- OsdInput.CancelanalogValue - The analog value of the input. Ranges from 0 to 1.elapsedTime - How long (in seconds) the input is being applied for.

Return Values

boolean - true if the input was "consumed", false otherwise.

Description

Pass input to the OSD object. Valid input ids are OsdInput enumeration values. The input is consumed if the OSD object handling it has decided that it should not be handled by anything else.

Examples

```
OsdObject.HandleInput( OsdInput.Accept )
```

See Also

None

OsdObject.IsActive

Brief

Query whether or not an OSD object is active (activated).

Definition

```
boolean OsdObject.IsActive( OsdObject object )
```

Arguments

object - The OSD object to query.

Return Values

true if the OSD object is activated, false otherwise.

Description

Query whether or not an OSD object is active (activated).

Examples

```
if OsdObject.IsActive( object ) then
    print( "OSD object is active!" )
end
```

See Also

None

OsdObject.SetAlphaType

Brief

Set the alpha type of an OSD object.

Definition

```
OsdObject.SetAlphaType( OsdObject object, enum alphaType )
```

Arguments

object - The OSD object to affectalphaType - An enum representing the alpha type. Can be one of the following:
OsdAlpha.Normal
OsdAlpha.Ninety
OsdAlpha.ThreeQuarter
OsdAlpha.Half
OsdAlpha.Quarter
OsdAlpha.Eighth
OsdAlpha.Sixteenth

Return Values

None.

Description

Set the alpha type of an OSD object.

Examples

```
OsdObject.SetAlphaType( object, OsdAlpha.Normal )
```

See Also

None

OsdObject.SetAlphaValue

Brief

Set the alpha value of an OSD object.

Definition

```
OsdObject.SetAlphaValue( OsdObject object, number alphaValue )
```

Arguments

object - The OSD object to affect.alphaValue - A number between 0 (fully transparent) and 1 (fully opaque).

Return Values

None.

Description

Set the alpha value of an OSD object.

Examples

```
OsdObject.SetAlphaValue( object, 0.5 )
```

See Also

None

OsdObject.SetAngle

Brief

Set the angle of an OSD object.

Definition

```
OsdObject.SetAngle( OsdObject object, number angle )
```

Arguments

object - The OSD object to affect.angle - The angle to set.

Return Values

None.

Description

Set the angle of an OSD object.

Examples

```
OsdObject.SetAngle( object, 10.0 )
```

See Also

None

OsdObject.SetColorType

Brief

Set the color type of an OSD object.

Definition

```
OsdObject.SetColorType( OsdObject object, enum colorType )
```

Arguments

object - The OSD object to affect.colorType - An enum representing the color type. Can be one of the following:

OsdColor.Black
OsdColor.White
OsdColor.Grey
OsdColor.Green
OsdColor.Yellow
OsdColor.Normal
OsdColor.Background
OsdColor.OppositeBg
OsdColor.Foreground
OsdColor.HighlightFg

Return Values

None.

Description

Set the color type of an OSD object.

Examples

```
OsdObject.SetColorType( object, OsdColor.Foreground )
```

See Also

None

OsdObject.SetColorValue

Brief

Set the color of an OSD object.

Definition

```
OsdObject.SetColorValue( OsdObject object, Vector4 color )
```

Arguments

object - The OSD object to affect.color - A Vector4 representing the color (RGBA).

Return Values

None.

Description

Set the color of an OSD object. Setting the color in this way (instead of OsdObject.SetColorType) has an extra memory and processing overhead.

Examples

```
OsdObject.SetColorValue( object, Vector4.Create( 1.0, 0.0, 0.0, 1.0 ) )
```

See Also

- OsdObject.SetColorType

OsdObject.SetHeight

Brief

Set the height of an OSD object.

Definition

```
OsdObject.SetHeight( OsdObject object, number height )
```

Arguments

object - The OSD object to affect.height - The height to set.

Return Values

None.

Description

Set the height of an OSD object.

Examples

```
OsdObject.SetHeight( object, 100.0 )
```

See Also

None

OsdObject.SetHeightFromChildren

Brief

Set the height of an OSD object so as to bound all children.

Definition

```
OsdObject.SetHeightFromChildren( OsdObject object, number border )
```

Arguments

object - The OSD object to affect.border - Extra border to add to height.

Return Values

None.

Description

Set the height of an OSD object so as to bound all children.

Examples

```
OsdObject.SetHeightFromChildren( width, 10.0 )
```

See Also

None

OsdObject.SetLocalSpaceAlignPoint

Brief

Set the local-space align point of an OSD object.

Definition

```
OsdObject.SetLocalSpaceAlignPoint( OsdObject object, enum alignPoint )
```

Arguments

object - The OSD object to affect.alignPoint - An enum representing the align point. Can be one of the following:

- OsdAlign.Center
- OsdAlign.Top
- OsdAlign.Bottom
- OsdAlign.Left
- OsdAlign.Right
- OsdAlign.TopLeft
- OsdAlign.TopRight
- OsdAlign.BottomLeft
- OsdAlign.BottomRight

Return Values

None.

Description

Set the local-space align point of an OSD object.

Examples

```
OsdObject.SetLocalSpaceAlignPoint( object, OsdAlign.Center )
```

See Also

None

OsdObject.SetName

Brief

Set the name of an object.

Definition

```
OsdObject.SetName( OsdObject object, string name )
```

Arguments

object - The object to set the name of.name - The new name for the object.

Return Values

None.

Description

Set the name of an object. The name should be unique in the set of siblings.

Examples

```
OsdObject.SetName( object, "banner" )
```

See Also

None

OsdObject.SetParentSpaceAlignPoint

Brief

Set the parent-space align-point of an OSD object.

Definition

```
OsdObject.SetParentSpaceAlignPoint( OsdObject object, enum alignPointOrigin, number alignPointOffsetX
= 0.0, number alignPointOffsetY = 0.0 )
```

Arguments

object - The OSD object to affect.alignPointOrigin - An enum representing the align point origin. Can be one of the following:

- OsdAlign.Center
- OsdAlign.Top
- OsdAlign.Bottom
- OsdAlign.Left
- OsdAlign.Right
- OsdAlign.TopLeft
- OsdAlign.TopRight
- OsdAlign.BottomLeft
- OsdAlign.BottomRightalignPointOffsetX - A number representing the x-component of the align point offset.alignPointOffsetY - A number representing the y-component of the align point offset.

Return Values

None.

Description

An OSD object is positioned relative to its parent (or, if its parent is "intangible", its next tangible "ancestor"). If a parent has no tangible ancestors, it is positioned in screen-space. An OSD object's position is specified as an offset from a point in its parent; that point is called the "parent-space align-point origin" and the offset is called the "parent-space align-point offset". The resultant position is called the "parent-space align-point". A second align point, the "local-space align-point", is also defined. This point is within the object itself. The object is finally positioned be placing the local-space align-point over the parent-space align-point. Both the parent-space align-point origin and the local-space align-point are defined as one of the following relative points within the parent and child object respectively: center top bottom left right top-left top-right bottom-left bottom-right

Examples

```
OsdObject.SetParentSpaceAlignPoint( object, OsdAlign.TopLeft, 100, 200 )
```

See Also

- OsdObject.SetLocalSpaceAlignPoint

OsdObject.SetPosition

Brief

Sets the position of an object.

Definition

```
OsdObject.SetPosition( OsdObject object, enum parentSpaceAlignPointOrigin, number
parentSpaceAlignPointOffsetX = 0.0, number parentSpaceAlignPointOffsetY = 0.0, enum
localSpaceAlignPoint = nil )
```

Arguments

object - The object to set the position of.parentSpaceAlignPointOrigin - An OsdAlign enum representing the origin of the parent-space align point.parentSpaceAlignPointOffsetX - The x-offset of the parent-space align point.parentSpaceAlignPointOffsetY - The y-offset of the parent-space align point.localSpaceAlignPoint - An OsdAlign enum representing the local-space align point.

Return Values

None.

Description

Sets the position of an object. The position is set relative to the object's (tangible) parent, or relative to the screen if no (tangible) parent exists. The position is defined by lining up two points; one in the parent-space and one in the local-space. The parent-space point is defined by an origin and an offset. If the local-space align point is omitted, it is taken to be the same as the parent-space align point origin.

Examples

```
OsdObject.SetPosition( object, OsdAlign.TopLeft, 100.0, 200.0, OsdAlign.Bottom )
```

See Also

None

OsdObject.SetScale

Brief

Set the scale of an OSD object.

Definition

```
OsdObject.SetScale( OsdObject object, number scale )
```

Arguments

object - The OSD object to affect.scale - The scale to set.

Return Values

None.

Description

Set the scale of an OSD object.

Examples

```
OsdObject.SetScale( object, 2.0 )
```

See Also

None

OsdObject.SetUniqueId

Brief

Set the unique Id of an OSD object.

Definition

```
OsdObject.SetUniqueId( OsdObject object, number uid = nil )
```

Arguments

object - The OSD object.uid - The unique id.

Return Values

None.

Description

The unique id must be an integer >= 0 or nil. The id must be unique with the group of sibling objects. Passing nil in clears the id.

Examples

```
OsdObject.SetUniqueId( object, 1 )
```

See Also

- OsdObject.GetUniqueId

OsdObject.SetVisible

Brief

Set an OSD object to be visible or invisible.

Definition

```
OsdObject.SetVisible( OsdObject object, boolean isVisible = true )
```

Arguments

object - OSD object to affect.isVisible - true or false (defaults to true).

Return Values

None.

Description

The specified OSD object is made visible or invisible, and all descendents of it inherit the same visibility setting. Use this function to show or hide specific components of an OSD interface element. For example, individual menu items of an OsdBasicMenu can be shown or hidden. When a new OsdObject is created, it is initially in a deactivated state. Once it is activated, it will be visible until it is made invisible with this function or deactivated. Invisible objects are still updated. For example, a scrolling text object will continue to scroll even when invisible.

Examples

```
OsdObject.SetVisible( object, false )
```

See Also

- OsdObject.Activate
- OsdObject.Deactivate

OsdObject.SetWidth

Brief

Set the width of an OSD object.

Definition

```
OsdObject.SetWidth( OsdObject object, number width )
```

Arguments

object - The OSD object to affect.width - The width to set.

Return Values

None.

Description

Set the width of an OSD object.

Examples

```
OsdObject.SetWidth( width, 200.0 )
```

See Also

None

OsdObject.SetWidthFromChildren

Brief

Set the width of an OSD object so as to bound all children.

Definition

```
OsdObject.SetWidthFromChildren( OsdObject object, number border )
```

Arguments

object - The OSD object to affect.border - Extra border to add to width.

Return Values

None.

Description

Set the width of an OSD object so as to bound all children.

Examples

```
OsdObject.SetWidthFromChildren( width, 10.0 )
```

See Also

None

OsdSelectionParent

OsdSelectionParent.BeginChanges

Brief

Call before making multiple changes to the selection parent.

Definition

```
OsdSelectionParent.BeginChanges( OsdSelectionParent parent )
```

Arguments

parent - The parent to affect.

Return Values

None.

Description

Call before making multiple changes to the selection parent. For example, adding/disabling/enabling children.

Examples

```
OsdSelectionParent.BeginChanges( parent )
```

See Also

None

OsdSelectionParent.EndChanges

Brief

Call after making changes to the selection parent.

Definition

```
OsdSelectionParent.EndChanges( OsdSelectionParent parent )
```

Arguments

parent - The parent to affect.

Return Values

None.

Description

Call after making changes to the selection parent. For example, adding children.

Examples

```
OsdSelectionParent.EndChanges( parent )
```

See Also

None

OsdSelectionParent.GetCurrentSelectionId

Brief

Returns the id of the currently selected child of the selection parent.

Definition

```
number OsdSelectionParent.GetCurrentSelectionId( OsdSelectionParent parent )
```

Arguments

parent - The OsdSelectionParent object.

Return Values

The id of the currently selected object.

Description

Returns the id of the currently selected child of the selection parent. The ids are defined by id(X) in the init string of the child.

Examples

```
local selected = OsdSelectionParent.GetCurrentSelectionId( parentObject )
```

See Also

None

OsdSelectionParent.SetObjectEnabled

Brief

Enable or disable a child of the selection parent object.

Definition

```
OsdSelectionParent.SetObjectEnabled( OsdSelectionParent parent, OsdObject child, boolean wantEnabled )
```

Arguments

parent - The OsdSelectionParent object. child - The child to enable/disable. wantEnabled - true or false.

Return Values

None.

Description

Enable or disable a child of the selection parent object. Disabled children are removed from the selection process.

Examples

```
OsdSelectionParent.SetObjectEnabled( parent, child, false )
```

See Also

None

OsdSimpleMenu

OsdSimpleMenu.AddItem

Brief

Add a simple menu item to a simple menu object.

Definition

```
number OsdSimpleMenu.AddItem( OsdSimpleMenu self, string title, number id, boolean setEnabled )
```

Arguments

self - The menu object for which to call the method.title - Text to display on the menu item.id - Non-negative unique (in this menu) identifier for the menu item, a negative value causes the item to be allocated an id by the system.setEnabled - true to have the menu item enabled (selectable) initially, false otherwise.

Return Values

The id of the menu item.

Description

Add a simple menu item to a simple menu object. The item will be added to the bottom of the menu. AddItem should be called between calls to BeginChanges and EndChanges.

Examples

```
me = Object.GetMe()

-- Text localized in Object Editor as "Play Game"
title1 = Object.GetLocalizedText( me, "PLAY_MENU_TITLE" )
-- Text localized in Object Editor as "About Game"
title2 = Object.GetLocalizedText( me, "ABOUT_MENU_TITLE" )

menuObject = OsdObject.CreateChildObject( parentObject, "OsdSimpleMenu", "menu" )
OsdSimpleMenu.BeginChanges( menuObject )
OsdSimpleMenu.AddItem( menuObject, title1, 0, true )
OsdSimpleMenu.AddItem( menuObject, title2, 1, true )
OsdSimpleMenu.EndChanges( menuObject )
```

See Also

- Object.GetLocalizedText
- Object.GetMe

OsdSimpleMenu.SetItemTitle

Brief

Change the text displayed on a simple menu item.

Definition

```
OsdSimpleMenu.SetItemTitle( OsdSimpleMenu self, number id, string title )
```

Arguments

self - The menu object for which to call the method.title - Text to display on the menu item.id - Unique identifier for the menu item.

Return Values

None.

Description

Change the text displayed on a simple menu item. The text will have been initially set when the item was added to the menu.

Examples

```
me = Object.GetMe()

-- Text localized in Object Editor as "Play Game"
title1 = Object.GetLocalizedText( me, "PLAY_MENU_TITLE" )
-- Text localized in Object Editor as "About Game"
title2 = Object.GetLocalizedText( me, "ABOUT_MENU_TITLE" )
-- Text localized in Object Editor as "End Game"
title3 = Object.GetLocalizedText( me, "END_MENU_TITLE" )

menuObject = OsdObject.CreateChildObject( parentObject, "OsdSimpleMenu", "menu" )
OsdSimpleMenu.BeginChanges( menuObject )
OsdSimpleMenu.AddItem( menuObject, title1, 0, true )
OsdSimpleMenu.AddItem( menuObject, title2, 1, true )
OsdSimpleMenu.EndChanges( menuObject )
OsdSimpleMenu.SetItemTitle( menuObject, 1, title3 )
```

See Also

- Object.GetLocalizedText
- Object.GetMe

OsdSkinnedBox

OsdSkinnedBox.ResetTextures

Brief

Reset the textures.

Definition

```
OsdSkinnedBox.ResetTextures( OsdSkinnedBox box )
```

Arguments

box - The OsdSkinnedBox object.

Return Values

None.

Description

Reset the textures. This will set all textures to nil, and is equivalent to passing nil to all arguments of SetTextures.

Examples

```
box = OsdObject.CreateChildObject( Osd.GetRoot(), "OsdSkinnedBox", "test" )
OsdSkinnedBox.SetTextures( box, Texture.Find("t1"), Texture.Find("t2"), Texture.Find("t3"),
Texture.Find("t4"), Texture.Find("t5"), Texture.Find("t6"), Texture.Find("t7"), Texture.Find("t8"),
Texture.Find("t9") )
OsdSkinnedBox.ResetTextures( box )
```

See Also

- OsdSkinnedBox.SetTextures

OsdSkinnedBox.SetColorGradientEnabled

Brief

Enable/disable the color gradient.

Definition

```
OsdSkinnedBox.SetColorGradientEnabled( OsdSkinnedBox box, boolean enable = true )
```

Arguments

box - The OsdSkinnedBox object.enable - true to enable the color gradient, false otherwise.

Return Values

None.

Description

Enable/disable the color gradient. The color gradient is applied from top to bottom and the parameters of it cannot be adjusted. The colour is based on the current OSD object colour, transitioning to either a lighter or darker shade, depending on the additive/subtractive setting.

Examples

```
box = OsdObject.CreateChildObject( Osd.GetRoot(), "OsdSkinnedBox", "test" )
OsdSkinnedBox.SetTextures( box, Texture.Find("t1"), Texture.Find("t2"), Texture.Find("t3"),
Texture.Find("t4"), Texture.Find("t5"), Texture.Find("t6"), Texture.Find("t7"), Texture.Find("t8"),
Texture.Find("t9") )
OsdSkinnedBox.SetColorGradientEnabled( box, true )
```

See Also

- OsdSkinnedBox.SetColorGradientSubtractive
- OsdSkinnedBox.SetColorGradientReversed
- OsdSkinnedBox.SetColorGradientMagnitude

OsdSkinnedBox.SetColorGradientMagnitude

Brief

Set the magnitude of the gradient.

Definition

```
OsdSkinnedBox.SetColorGradientMagnitude( OsdSkinnedBox box, number magnitude )
```

Arguments

box - The OsdSkinnedBox object. magnitude - The gradient magnitude.

Return Values

None.

Description

Set the magnitude of the gradient. Valid gradient magnitudes are in the range zero to one.

Examples

```
box = OsdObject.CreateChildObject( Osd.GetRoot(), "OsdSkinnedBox", "test" )
OsdSkinnedBox.SetTextures( box, Texture.Find("t1"), Texture.Find("t2"), Texture.Find("t3"),
Texture.Find("t4"), Texture.Find("t5"), Texture.Find("t6"), Texture.Find("t7"), Texture.Find("t8"),
Texture.Find("t9") )
OsdSkinnedBox.SetColorGradientEnabled( box, true )
OsdSkinnedBox.SetColorGradientMagnitude( box, 0.5 )
```

See Also

- OsdSkinnedBox.SetColorGradientEnabled
- OsdSkinnedBox.SetColorGradientSubtractive
- OsdSkinnedBox.SetColorGradientReversed

OsdSkinnedBox.SetColorGradientReversed

Brief

Set whether or not the direction of the color gradient is reversed.

Definition

```
OsdSkinnedBox.SetColorGradientReversed( OsdSkinnedBox box, boolean reversed = true )
```

Arguments

box - The OsdSkinnedBox object. reversed - true to set the gradient reversed, false otherwise.

Return Values

None.

Description

Set whether or not the direction of the color gradient is reversed.

Examples

```
box = OsdObject.CreateChildObject( Osd.GetRoot(), "OsdSkinnedBox", "test" )
OsdSkinnedBox.SetTextures( box, Texture.Find("t1"), Texture.Find("t2"), Texture.Find("t3"),
Texture.Find("t4"), Texture.Find("t5"), Texture.Find("t6"), Texture.Find("t7"), Texture.Find("t8"),
Texture.Find("t9") )
OsdSkinnedBox.SetColorGradientEnabled( box, true )
OsdSkinnedBox.SetColorGradientReversed( box, true )
```

See Also

- OsdSkinnedBox.SetColorGradientEnabled
- OsdSkinnedBox.SetColorGradientSubtractive
- OsdSkinnedBox.SetColorGradientMagnitude

OsdSkinnedBox.SetColorGradientSubtractive

Brief

Set whether or not the color gradient is subtractive.

Definition

```
OsdSkinnedBox.SetColorGradientSubtractive( OsdSkinnedBox box, boolean subtractive = true )
```

Arguments

box - The OsdSkinnedBox object. subtractive - true to set the gradient subtractive, false otherwise.

Return Values

None.

Description

Set whether or not the color gradient is subtractive. If it is not subtractive, then it is additive. An additive gradient will transition to a lighter shade, subtractive to a darker shade.

Examples

```
box = OsdObject.CreateChildObject( Osd.GetRoot(), "OsdSkinnedBox", "test" )
OsdSkinnedBox.SetTextures( box, Texture.Find("t1"), Texture.Find("t2"), Texture.Find("t3"),
Texture.Find("t4"), Texture.Find("t5"), Texture.Find("t6"), Texture.Find("t7"), Texture.Find("t8"),
Texture.Find("t9") )
OsdSkinnedBox.SetColorGradientEnabled( box, true )
OsdSkinnedBox.SetColorGradientSubtractive( box, true )
```

See Also

- OsdSkinnedBox.SetColorGradientEnabled
- OsdSkinnedBox.SetColorGradientReversed
- OsdSkinnedBox.SetColorGradientMagnitude

OsdSkinnedBox.SetTexture

Brief

Set a texture.

Definition

```
OsdSkinnedBox.SetTexture( OsdSkinnedBox box, enum positionId, Texture t = nil)
```

Arguments

box - The OsdSkinnedBox object.positionId - An enum representing which texture to set. Can be one of:

- OsdAlign.TopLeft
- OsdAlign.Top
- OsdAlign.TopRight
- OsdAlign.Left
- OsdAlign.Center
- OsdAlign.Right
- OsdAlign.BottomLeft
- OsdAlign.Bottom
- OsdAlign.BottomRightt - The texture.

Return Values

None.

Description

Set a texture.

Examples

```
box = OsdObject.CreateChildObject( Osd.GetRoot(), "OsdSkinnedBox", "test" )
OsdSkinnedBox.SetTextures( box, Texture.Find("t1"), Texture.Find("t2"), Texture.Find("t3"),
Texture.Find("t4"), Texture.Find("t5"), Texture.Find("t6"), Texture.Find("t7"), Texture.Find("t8"),
Texture.Find("t9") )
OsdSkinnedBox.SetTexture( box, OsdAlign.Center, Texture.Find("alt") )
```

See Also

- OsdSkinnedBox.SetTextures

OsdSkinnedBox.SetTextures

Brief

Set the textures.

Definition

```
OsdSkinnedBox.SetTextures( OsdSkinnedBox box, Texture t1, Texture t2, Texture t3, Texture t4, Texture
t5, Texture t6, Texture t7, Texture t8, Texture t9 )
```

Arguments

box - The OsdSkinnedBox object.t1 - The top-left texture, or nil to clear.t2 - The top-center texture, or nil to clear.t3 - The top-right texture, or nil to clear.t4 - The left-center texture, or nil to clear.t5 - The center texture, or nil to clear.t6 - The right-center texture, or nil to clear.t7 - The bottom-left texture, or nil to clear.t8 - The bottom-center texture, or nil to clear.t9 - The bottom-right texture, or nil to clear.

Return Values

None.

Description

Set the textures for the skinned box. Nil can be passed to any texture, which will hide that part of the skinned box.

Examples

```
box = OsdObject.CreateChildObject( Osd.GetRoot(), "OsdSkinnedBox", "test" )
OsdSkinnedBox.SetTextures( box, Texture.Find("t1"), Texture.Find("t2"), Texture.Find("t3"),
Texture.Find("t4"), Texture.Find("t5"), Texture.Find("t6"), Texture.Find("t7"), Texture.Find("t8"),
Texture.Find("t9") )
```

See Also

- OsdSkinnedBox.SetTexture
- OsdSkinnedBox.ResetTextures

OsdTextLines

OsdTextLines.GetNumTextLines

Brief

Get the number of lines of text for an OsdTextLines object.

Definition

```
number OsdTextLines.GetNumTextLines( OsdTextLines textLinesObject )
```

Arguments

textLinesObject - OsdTextLines to query.

Return Values

The number of lines of text.

Description

Get the number of lines of text for an OsdTextLines object.

Examples

```
numLines = OsdTextLines.GetNumTextLines( textLinesObject )
```

See Also

None

OsdTextLines.SetFont

Brief

Set the font on an OsdTextLines object.

Definition

```
OsdTextLines.SetFont( OsdTextLines textLinesObject, string font )
```

Arguments

textLinesObject - The OsdTextLines object to affect. font - String representing the font. Can be one of the following:
"13reg"

"13bold"
"18reg"
"18bold"
"26reg"
"26bold"
"60bold"

Return Values

None.

Description

Set the font on an OsdTextLines object.

Examples

```
OsdTextLines.SetFont( textLinesObject, "13reg" )
```

See Also

None

OsdTextLines.SetJustification

Brief

Set the justification on an OsdTextLines object.

Definition

```
OsdTextLines.SetJustification( OsdTextLines textLinesObject, enum justify )
```

Arguments

textLinesObject - The OsdTextLines object to affect. justify - An enum representing the justification. Can be one of the following:
OsdAlign.Left
OsdAlign.Right
OsdAlign.Center
OsdAlign.LeftRight

Return Values

None.

Description

Set the justification on an OsdTextLines object.

Examples

```
OsdTextLines.SetJustification( textLinesObject, OsdAlign.Center )
```

See Also

None

OsdTextLines.SetLineSpacing

Brief

Set the line spacing on an OsdTextLines object.

Definition

```
OsdTextLines.SetLineSpacing( OsdTextLines textLinesObject, number lineSpacing )
```

Arguments

textLinesObject - The OsdTextLines object to affect. lineSpacing - The distance between lines of text.

Return Values

None.

Description

Set the line spacing on an OsdTextLines object.

Examples

```
OsdTextLines.SetLineSpacing( textLinesObject, 10 )
```

See Also

None

OsdTextLines.SetText

Brief

Set the text on an OsdTextLines object.

Definition

```
OsdTextLines.SetText( OsdTextLines textLinesObject, string localizedText )
```

Arguments

textLinesObject - The OsdTextLines object to affect. localizedText - Localized text string.

Return Values

None.

Description

Set the text on an OsdTextLines object.

Examples

```
me = Object.GetMe()

-- Localized in Object Editor as "Hello World!"
message = Object.GetLocalizedText( me, "GREETING" )

OsdTextLines.SetText( textLinesObject, message )
```

See Also

- Object.GetLocalizedText

- Object.GetMe

OsdTextPanel

OsdTextPanel.GetTextBorder

Brief

Get the size of the border around the text for an OsdTextPanel object.

Definition

```
number OsdTextPanel.GetTextBorder( OsdTextPanel textPanelObject )
```

Arguments

textPanelObject - The OsdTextPanel object to query.

Return Values

The size of the border around the text.

Description

Get the size of the border around the text for an OsdTextPanel object.

Examples

```
border = OsdTextPanel.GetTextBorder( textPanelObject )
```

See Also

None

OsdTextPanel.GetTextLines

Brief

Get the internal textlines object from an OsdTextPanel object.

Definition

```
OsdTextLines OsdTextPanel.GetTextLines( OsdTextPanel textPanelObject )
```

Arguments

textPanelObject - The OsdTextPanel object to query.

Return Values

The OsdTextLines object.

Description

Get the internal textlines object from an OsdTextPanel object.

Examples

```
textLinesObject = OsdTextPanel.GetTextLines( textPanelObject )
```

See Also

None

OsdTextPanel.IsFitHeightToText

Brief

Query whether an OsdTextPanel object's height is adjusted to its text.

Definition

```
boolean OsdTextPanel.IsFitHeightToText( OsdTextPanel textPanelObject )
```

Arguments

textPanelObject - The OsdTextPanel object to query.

Return Values

true or false.

Description

Query whether an OsdTextPanel object's height is adjusted to its text.

Examples

```
OsdTextPanel.IsFitHeightToText( textPanelObject )
```

See Also

None

OsdTextPanel.IsPanelVisible

Brief

Query whether an OsdTextPanel object's background panel is visible.

Definition

```
boolean OsdTextPanel.IsPanelVisible( OsdTextPanel textPanelObject )
```

Arguments

textPanelObject - The OsdTextPanel object to query.

Return Values

true or false.

Description

Query whether an OsdTextPanel object's background panel is visible.

Examples

```
if OsdTextPanel.IsPanelVisible( textPanelObject ) then
    print( "Panel is visible!" )
end
```

See Also

None

OsdTextPanel.IsScrollBarVisible

Brief

Query whether an OsdTextPanel object's scroll-bar is visible.

Definition

```
boolean OsdTextPanel.IsScrollBarVisible( OsdTextPanel textPanelObject )
```

Arguments

textPanelObject - The OsdTextPanel object to query.

Return Values

true or false.

Description

Query whether an OsdTextPanel object's scroll-bar is visible.

Examples

```
OsdTextPanel.IsScrollBarVisible( textPanelObject )
```

See Also

None

OsdTextPanel.SetFitHeightToText

Brief

Set whether or not you want the height of the text panel to adjust to the text.

Definition

```
OsdTextPanel.SetFitHeightToText( OsdTextPanel textPanelObject, boolean isFitHeightToText )
```

Arguments

textPanelObject - The OsdTextPanel object to affect. isFitHeightToText - Boolean value (true or false).

Return Values

None.

Description

Set whether or not you want the height of the text panel to adjust to the text.

Examples

```
OsdTextPanel.SetFitHeightToText( textPanelObject, true )
```

See Also

None

OsdTextPanel.SetFont

Brief

Set the font on an OsdTextPanel object.

Definition

```
OsdTextPanel.SetFont( OsdTextPanel textPanelObject, string font )
```

Arguments

textPanelObject - The OsdTextPanel object to affect.font - String representing the font. Can be one of the following:
"13reg"
"13bold"
"18reg"
"18bold"
"26reg"
"26bold"
"60bold"

Return Values

None.

Description

Set the font on an OsdTextPanel object.

Examples

```
OsdTextPanel.SetFont( textPanelObject, "13reg" )
```

See Also

None

OsdTextPanel.SetJustification

Brief

Set the justification on an OsdTextPanel object.

Definition

```
OsdTextPanel.SetJustification( OsdTextPanel textPanelObject, enum justify )
```

Arguments

textPanelObject - The OsdTextPanel object to affect.justify - An enum representing the justification. Can be one of the

following:

- OsdAlign.Left
- OsdAlign.Right
- OsdAlign.Center
- OsdAlign.LeftRight

Return Values

None.

Description

Set the justification on an OsdTextPanel object.

Examples

```
OsdTextPanel.SetJustification( textPanelObject, OsdAlign.Center )
```

See Also

None

OsdTextPanel.SetLineSpacing

Brief

Set the line spacing on an OsdTextPanel object.

Definition

```
OsdTextPanel.SetLineSpacing( OsdTextPanel textPanelObject, number lineSpacing )
```

Arguments

textPanelObject - The OsdTextPanel object to affect. lineSpacing - The distance between lines of text.

Return Values

None.

Description

Set the line spacing on an OsdTextPanel object.

Examples

```
OsdTextPanel.SetLineSpacing( textPanelObject, 10 )
```

See Also

None

OsdTextPanel.SetPanelVisible

Brief

Set whether or not the panel background is visible.

Definition

```
OsdTextPanel.SetPanelVisible( OsdTextPanel textPanelObject, boolean isPanelVisible )
```

Arguments

textPanelObject - The OsdTextPanel object to affect. isPanelVisible - Boolean value (true or false).

Return Values

None.

Description

Set whether or not the panel background is visible.

Examples

```
OsdTextPanel.SetPanelVisible( textPanelObject, true )
```

See Also

None

OsdTextPanel.SetScrollBarVisible

Brief

Set whether or not the scroll bar is visible.

Definition

```
OsdTextPanel.SetScrollBarVisible( OsdTextPanel textPanelObject, boolean isScrollBarVisible )
```

Arguments

textPanelObject - The OsdTextPanel object to affect. isScrollBarVisible - Boolean value (true or false).

Return Values

None.

Description

Set whether or not the scroll bar is visible.

Examples

```
OsdTextPanel.SetScrollBarVisible( textPanelObject, false )
```

See Also

None

OsdTextPanel.SetScrollType

Brief

Set the scroll-type on an OsdTextPanel object.

Definition

```
OsdTextPanel.SetScrollType( OsdTextPanel textPanelObject, enum scrollType )
```

Arguments

textPanelObject - The OsdTextPanel object to affect.scrollType - An OsdMotion enum representing the scroll-type. Can be one of the following:
OsdMotion.None
OsdMotion.ConstSpeed
OsdMotion.Spring

Return Values

None.

Description

Set the scroll-type on an OsdTextPanel object.

Examples

```
OsdTextPanel.SetScrollType( textPanelObject, "smooth" )
```

See Also

None

OsdTextPanel.SetText

Brief

Set the text of a OsdTextPanel object.

Definition

```
OsdTextPanel.SetText(OsdTextPanel textPanelObject, string text, number maxTextLength = -1, boolean
trimLeadingChars = false)
```

Arguments

textPanelObject - The OsdTextPanel object to affect.text - The text.maxTextLength - A positive value to limit the text length, or -1 otherwise.trimLeadingChars - (If the text is too long) true, to trim leading character, false to trim trailing characters.

Return Values

None.

Description

Set the text of a OsdTextPanel object. The maxTextLength parameter can be used to limit the number of characters copied into the object. It actually limits the number of bytes, so if the string contains multibyte characters, there may be fewer actual characters than specified. If the function needs to remove characters to comply with maxTextLength, the characters will be removed from the end unless trimLeadingChars is true, in which case the characters will be removed from the beginning.

Examples

```
me = Object.GetMe()

-- Text localized in Object Editor
panelText = Object.GetLocalizedText( me, "PANEL_MESSAGE" )

OsdTextPanel.SetText( textPanelObject, panelText )
```

See Also

- Object.GetLocalizedText
- Object.GetMe

OsdTextPanel.SetTextBorder

Brief

Set the size of the border around the text on an OsdTextPanel object.

Definition

```
OsdTextPanel.SetTextBorder( OsdTextPanel textPanelObject, number textBorder )
```

Arguments

textPanelObject - The OsdTextPanel object to affect.textBorder - The text border size.

Return Values

None.

Description

Set the size of the border around the text on an OsdTextPanel object.

Examples

```
OsdTextPanel.SetTextBorder( textPanelObject, 10 )
```

See Also

None

OsdTextPanel.SetTextColorType

Brief

Set the color of the text on an OsdTextPanel object.

Definition

```
OsdTextPanel.SetTextColorType( OsdTextPanel textPanelObject, enum colorType )
```

Arguments

textPanelObject - The OsdTextPanel object to affect.colorType - An enum representing the color type. Can be one of the following:
OsdColor.Black
OsdColor.White
OsdColor.Grey

OsdColor.Green
OsdColor.Yellow
OsdColor.Red
OsdColor.Normal
OsdColor.Background
OsdColor.OppositeBg
OsdColor.Foreground
OsdColor.HighlightFg

Return Values

None.

Description

Set the color of the text on an OsdTextPanel object.

Examples

```
OsdTextPanel.SetTextColorType( textPanelObject, OsdColor.Foreground )
```

See Also

None

Osk

Osk.Close

Brief

Closes the OSK opened by this script.

Definition

```
Osk.Close()
```

Arguments

None

Return Values

None

Description

Closes the OSK opened by this script. If the OSK was not opened by this script, then this function has no effect.

Examples

```
Osk.Close()
```

See Also

- Osk.Open

Osk.GetText

Brief

Obtain the text the user entered in the OSK.

Definition

```
string Osk.GetText()
```

Arguments

None

Return Values

The text that was entered by the user or nil if the user cancelled input or not in the callback context.

Description

Returns the text that was entered by the user in the OSK. Only valid during the registered callback for the OSK request that triggered the call.

Examples

```
mytext = Keyboard.GetText()
```

See Also

None

Osk.IsAvailable

Brief

Checks if it is possible to to open the OSK.

Definition

```
boolean Osk.IsAvailable()
```

Arguments

None

Return Values

true if the script is able to open the OSK, false if the OSK is unavailable.

Description

Checks if it is possible to to open the OSK.

Examples

```
if Osk.IsAvailable() == true then
    Osk.Open( "OnText()", "How old are you?", "", 3, false, true, false )
end
```

See Also

None

Osk.Open

Brief

Opens the OSK to request a string from the user.

Definition

```
boolean Osk.Open( function callback, string title, string initialText, number maxCharacters,
CharacterSet characterSet, boolean allowReturn )
```

Arguments

callback - The Lua function to execute when the OSK has a result. Max 127 characters.title - The title to give the OSK. Max 127 characters.initialText - The initial text to add to the OSK. Max 511 characters.maxCharacters - The maximum number of characters (not bytes) to request from the user. Max 511 characters.characterSet - A valid character set enum value (see details)allowReturn - Set to true to allow the user to enter new line characters.

Return Values

false if the OSK could not be started at this time.

Description

Opens up an instance of the OSK to accept text input from the user. Only one instance of the OSK can be open at a time. characterSet can be any of the following:

- CharacterSet.Default
- CharacterSet.LatinOnly
- CharacterSet.NumericalOnly
- CharacterSet.Password

Examples

```
success = Osk.Open( OnNumerical, "What is your pin?", "0000", 4, CharacterSet.NumericalOnly, false )
```

See Also

None

OutboundMessage

OutboundMessage.Create

Brief

Creates a new OutboundMessage for sending.

Definition

```
OutboundMessage OutboundMessage.Create( number messageType, number bufferSize )
```

Arguments

messageType - Message type as a number. This is free for the script to set and is sent as an int32.bufferSize - Size of the buffer to write the message to. This should be equal to the size of the data you want to include in the message.

Return Values

A new instance of an OutboundMessage.

Description

Creates a new OutboundMessage for sending. The size for the buffer will be allocated out of the script's network environment (currently the global network pool).

Examples

```
local message = OutboundMessage.Create( 12345, 16 )
```

See Also

None

OutboundMessage.GetRemainingBufferSize

Brief

Returns the space remaining in the message's buffer.

Definition

```
number OutboundMessage.GetRemainingBufferSize( OutboundMessage msg )
```

Arguments

msg - The instance of the message to query.

Return Values

The remaining buffer space in bytes.

Description

Returns the space remaining in the message's buffer.

Examples

```
local size = OutboundMessage.GetRemainingBufferSize( message )
```

See Also

None

OutboundMessage.Reset

Brief

Resets the buffer for the OutboundMessage.

Definition

```
OutboundMessage.Reset( OutboundMessage msg )
```

Arguments

msg - The instance of the message to reset.

Return Values

None.

Description

Resets the message buffer for the OutboundMessage so that it can be used again with new contents.

Examples

```
local message = OutboundMessage.Create( 0, 4 )
OutboundMessage.WriteInt16( message, value1 )
OutboundMessage.WriteInt16( message, value2 )
OutboundMessage.Send( message, personId )

-- Reuse message
OutboundMessage.Reset( message )
OutboundMessage.WriteInt32( message, value3 )
OutboundMessage.Send( message )
```

See Also

- OutboundMessage.Create
- OutboundMessage.Send
- OutboundMessage.WriteInt32

OutboundMessage.Send

Brief

Sends the OutboundMessage.

Definition

```
OutboundMessage.Send( OutboundMessage msg, string personId = nil )
```

Arguments

msg - The instance of the message to send.personId - The person ID to send the message to. If it is not specified the message is sent to all clients.

Return Values

None.

Description

Sends the specified OutboundMessage. The message can be sent to a specific person or broadcast to all persons located in the current scene.

Examples

```
local message = OutboundMessage.Create( 0, 6 )
OutboundMessage.WriteInt16( message, index )
OutboundMessage.WriteInt32( message, score )

personId = Person.GetId( otherPerson )
OutboundMessage.Send( message, personId )
```

See Also

None

OutboundMessage.SendUnguaranteed

Brief

Sends the OutboundMessage over UDP.

Definition

```
OutboundMessage.SendUnguaranteed( OutboundMessage msg, string personId = nil )
```

Arguments

msg - The instance of the message to send.personId - The person ID to send the message to. If it is not specified the message is sent to all clients.

Return Values

None.

Description

Sends the specified OutboundMessage. The message can be sent to a specific person or broadcast to all persons located in the current scene. Please note that UDP messages are unguaranteed, out-of-order. Care must be taken when using this transport type as it is not suitable for all message. Do not use this if your message must reach the target user. If ordering is essential, this can be achieved by simply adding a time stamp to the message and manually checking and discarding out-of-order messages on the receiving side. The global function GetInstanceTime can be used as one method of manually adding a timestamp.

Examples

```
local message = OutboundMessage.Create( 0, 6 )
OutboundMessage.WriteInt16( message, index )
OutboundMessage.WriteInt32( message, score )

personId = Person.GetId( otherPerson )
OutboundMessage.SendUnguaranteed( message, personId )
```

See Also

- GetInstanceTime
- OutboundMessage.Send

OutboundMessage.WriteBigInt

Brief

Write a BigInt value into an outbound message.

Definition

```
OutboundMessage.WriteBigInt( OutboundMessage msg, BigInt bigInt )
```

Arguments

msg - The message.bigInt - The BigInt value to write.

Return Values

None.

Description

Writes a BigInt value into an outbound message. Any size of BigInt is permissible.

Examples

```
local message = OutboundMessage.Create(0, 8 )
local bigInt = BigInt.Create(64, "0x1234567890ABCDEF")
OutboundMessage.WriteBigInt( message, bigInt )
OutboundMessage.Send( message )
```

See Also

- BigInt.Create

OutboundMessage.WriteFloat

Brief

Writes a 32-bit floating point value (float) to the message.

Definition

```
OutboundMessage.WriteFloat( OutboundMessage msg, number data )
```

Arguments

msg - The instance of the message to write to.data - 32-bit floating point value to write to the message packet.

Return Values

None.

Description

Writes a 32-bit floating point value (float) to the message.

Examples

```
local message = OutboundMessage.Create( 0, 4 )
OutboundMessage.WriteFloat( message, 3.14159 )
OutboundMessage.Send( message )
```

See Also

- OutboundMessage.WriteInt8
- OutboundMessage.WriteInt16
- OutboundMessage.WriteInt32
- OutboundMessage.WriteString

OutboundMessage.WriteInt8

Brief

Writes an 8-bit integer to the message.

Definition

```
OutboundMessage.WriteInt8( OutboundMessage msg, number data )
```

Arguments

msg - The instance of the message to write to.data - 8-bit integer to write to the message packet.

Return Values

None.

Description

Writes an 8-bit integer (or byte) to the message.

Examples

```
local message = OutboundMessage.Create( 0, 1 )
OutboundMessage.WriteInt8( message, value )
OutboundMessage.Send( message )
```

See Also

- OutboundMessage.WriteInt16
- OutboundMessage.WriteInt32
- OutboundMessage.WriteFloat
- OutboundMessage.WriteString

OutboundMessage.WriteInt16

Brief

Writes a 16-bit integer (short) to the message.

Definition

```
OutboundMessage.WriteInt16( OutboundMessage msg, number data )
```

Arguments

msg - The instance of the message to write to.data - 16-bit integer to write to the message packet.

Return Values

None.

Description

Writes a 16-bit integer (short) to the message.

Examples

```
local message = OutboundMessage.Create( 0, 2 )
OutboundMessage.WriteInt16( message, index )
OutboundMessage.Send( message )
```

See Also

- OutboundMessage.WriteInt8
- OutboundMessage.WriteInt32
- OutboundMessage.WriteFloat
- OutboundMessage.WriteString

OutboundMessage.WriteInt32

Brief

Writes a 32-bit integer to the message.

Definition

```
OutboundMessage.WriteInt32( OutboundMessage msg, number data )
```

Arguments

msg - The instance of the message to write to.data - 32-bit integer to write to the message packet.

Return Values

None.

Description

Writes a 32-bit integer to the message.

Examples

```
local message = OutboundMessage.Create( 0, 4 )
OutboundMessage.WriteInt32( message, index )
OutboundMessage.Send( message )
```

See Also

- OutboundMessage.WriteInt8
- OutboundMessage.WriteInt16
- OutboundMessage.WriteFloat
- OutboundMessage.WriteString

OutboundMessage.WriteString

Brief

Writes a string value to the message.

Definition

```
OutboundMessage.WriteString( OutboundMessage msg, string data )
```

Arguments

msg - The instance of the message to write to.data - string value to write to the message packet.

Return Values

None.

Description

Writes a string value to the message.

Examples

```
greeting = "Hello World!"
-- Msg size is length of string plus 1 for terminating character
msgSize  = string.len( greeting ) + 1
local message = OutboundMessage.Create( 0, msgSize )
OutboundMessage.WriteString( message, greeting )
OutboundMessage.Send( message )
```

See Also

- OutboundMessage.WriteInt8
- OutboundMessage.WriteInt16
- OutboundMessage.WriteInt32
- OutboundMessage.WriteFloat

Pad

Pad.CreatePads

Brief

Creates the specified number of pads.

Definition

```
Pad.CreatePads( number padCount = 1 )
```

Arguments

padCount - The number of pads required. Valid range is 1 to 4.

Return Values

None.

Description

Creates the specified number of pads required by the script. This should never be called more than once.

Examples

```
-- Create a single pad
Pad.CreatePads()
pad = Pad.GetPad()
```

See Also

- Pad.GetPad

Pad.GetAnalogExtent

Brief

Returns the analog value of the stick, button or sensor.

Definition

```
number Pad.GetAnalogExtent( Pad pad, number actionId )
```

Arguments

pad - The pad to query.actionId - The user defined action ID corresponding to a button ID.

Return Values

The analogue value of the controller stick, button or sensor.

Description

Returns the analog value of the stick, button or motion sensor in the following ranges: Analogue stick: -1.0 to 1.0. Analogue button: 0.0 (not pressed) to 1.0 (fully pressed). Motion sensor: -1.0 to 1.0. Note: For the motion sensor, if the controller is stationary, gravity will still cause the returned value to be non-zero (depending on the angle at which the controller is held).

Examples

```
-- Initialization code
Pad.CreatePads()
pad = Pad.GetPad()
TURN_LEFT = 5
Pad.Reserve( pad, PAD_DPAD_LEFT, TURN_LEFT )

function OnUpdate()
    print( "Button value = " ..  Pad.GetAnalogExtent( pad, myLeft ) )
end
```

See Also

- Pad.CreatePads
- Pad.GetPad
- Pad.Reserve

Pad.GetPad

Brief

Gets the specified pad object.

Definition

```
Pad Pad.GetPad( number which = 1 )
```

Arguments

which - The pad to get. Ranges from 1 to the number of pads created.

Return Values

The pad corresponding to the specified index or nil on failure.

Description

Gets the specified pad object.

Examples

```
-- Create two pads
Pad.CreatePads( 2 )
pad1 = Pad.GetPad( 1 )
pad2 = Pad.GetPad( 2 )
```

See Also

- Pad.CreatePads

Pad.GetSensorForwards

Brief

Returns the vector for the forwards direction of the pad.

Definition

```
Vector4 Pad.GetSensorForwards( Pad pad )
```

Arguments

pad - The pad to query.

Return Values

The vector for the forwards direction of the pad.

Description

Returns the vector for the forwards direction of the pad.

Examples

```
local direction = Pad.GetSensorForwards( pad )
```

See Also

- Pad.GetSensorRight
- Pad.GetSensorUp

Pad.GetSensorRight

Brief

Returns the vector for the right direction of the pad.

Definition

```
Vector4 Pad.GetSensorRight( Pad pad )
```

Arguments

pad - The pad to query.

Return Values

The vector for the right direction of the pad.

Description

Returns the vector for the right direction of the pad.

Examples

```
local direction = Pad.GetSensorRight( pad )
```

See Also

- Pad.GetSensorUp
- Pad.GetSensorForwards

Pad.GetSensorUp

Brief

Returns the vector for the up direction of the pad.

Definition

```
Vector4 Pad.GetSensorUp( Pad pad )
```

Arguments

pad - The pad to query.

Return Values

The vector for the up direction of the pad.

Description

Returns the vector for the up direction of the pad.

Examples

```
local direction = Pad.GetSensorUp( pad )
```

See Also

- Pad.GetSensorRight
- Pad.GetSensorForwards

Pad.IsCircleAccept

Brief

Returns true if the pad associates PAD_CIRCLE with PAD_ACCEPT and false otherwise.

Definition

```
boolean Pad.IsCircleAccept( Pad pad )
```

Arguments

pad - The pad to query.

Return Values

Returns true if the pad associates PAD_CIRCLE with PAD_ACCEPT and false otherwise.

Description

Returns true if the pad associates PAD_CIRCLE with PAD_ACCEPT and false otherwise.

Examples

```
Pad.CreatePads()
pad = Pad.GetPad()
if( Pad.IsCircleAccept( pad ) ) then
    print( "Pad uses circle for accept" );
else
    print( "Pad uses cross for accept" );
end
```

See Also

- Pad.CreatePads
- Pad.GetPad

Pad.IsConnected

Brief

Returns true if the pad is connected and false otherwise.

Definition

```
boolean Pad.IsConnected( Pad pad )
```

Arguments

pad - The pad to query.

Return Values

true if the pad is connected and false otherwise.

Description

Returns true if the pad is connected and false otherwise.

Examples

```
Pad.CreatePads()
pad = Pad.GetPad()
print( "Pad connected = " .. Pad.IsConnected( pad ) )
```

See Also

- Pad.CreatePads
- Pad.GetPad

Pad.IsHeld

Brief

Detects whether the specified reserved pad button is currently pressed.

Definition

```
boolean Pad.IsHeld( Pad pad, number actionId )
```

Arguments

pad - The pad to query.actionId - The user defined action ID corresponding to a button ID

Return Values

true if the button is held and false otherwise.

Description

Detects whether the reserved pad button associated with the specified action ID is currently pressed.

Examples

```
-- Initialization code
Pad.CreatePads()
pad = Pad.GetPad()
FIRE_LASER = 5
Pad.Reserve( pad, PAD_CROSS, FIRE_LASER )

function OnUpdate()
    if Pad.IsHeld( pad, FIRE_LASER ) then
        print( "Laser firing sustained blast!" )
    end
end
```

See Also

- Pad.CreatePads
- Pad.GetPad
- Pad.Reserve

Pad.Reserve

Brief

Makes a controller function queryable.

Definition

```
Pad.Reserve( Pad pad, number buttonId, number actionId = buttonId, boolean repeat = false, boolean
dropThrough = false )
```

Arguments

pad - The pad to affect.buttonId - The controller function to reserve. Can be one of the following:
PAD_DPAD_UP
PAD_DPAD_DOWN
PAD_DPAD_LEFT
PAD_DPAD_RIGHT
PAD_JOY_LEFT_X
PAD_JOY_LEFT_Y
PAD_JOY_RIGHT_X
PAD_JOY_RIGHT_Y
PAD_ACCEPT
PAD_SQUARE

PAD_TRIANGLE
PAD_DECLINE
PAD_L1_SHOULDER
PAD_L2_SHOULDER
PAD_L3_JOY
PAD_R1_SHOULDER
PAD_R2_SHOULDER
PAD_R3_JOY
PAD_CROSS
PAD_CIRCLE
PAD_SENSOR_AXIS_X
PAD_SENSOR_AXIS_Y
PAD_SENSOR_AXIS_Z
PAD_SENSOR_GYRO_Y
PAD_ACTUATOR_SMALL
PAD_ACTUATOR_BIGactionId - The user defined action ID to map the button ID to.repeat - If set to true, the Pad.WasJustPressed function can return true for a single button press which is held down continuously.dropThrough - If set to true, other input handlers with lower priority can also receive the button event. Normally only the highest priority input handler with the button reserved will receive the button event.

Return Values

None.

Description

Reserves the specified controller function and assigns an abstract action ID to use for that button.

Examples

```
-- Initialization code
Pad.CreatePads()
pad = Pad.GetPad()
FIRE_LASER = 5
Pad.Reserve( pad, PAD_CROSS, FIRE_LASER )
```

See Also

- Pad.UnReserve

Pad.SetActuator

Brief

Sets the specified actuator's vibration strength.

Definition

```
boolean Pad.SetActuator( Pad pad, number actuator, number value )
```

Arguments

pad - The pad to affect.actuator - The actuator to set. Can be one of the following:
PAD_ACTUATOR_SMALL
PAD_ACTUATOR_BIGvalue - The value to set the actuator to. Valid values for PAD_ACTUATOR_SMALL are only 0 or 1, for PAD_ACTUATOR_BIG it is all values from 0.0 to 1.0.

Return Values

true if sucessfully set and false otherwise.

Description

Sets the specified actuator's vibration strength. If called by multiple scripts or the same script multiple times per frame it accumulates the values. The actuator must be reserved before it can be set. The small actuator can only be on or off, the rotation frequency cannot be changed.

Examples

```
-- Initialization code
Pad.CreatePads()
pad = Pad.GetPad()
Pad.Reserve( pad, PAD_ACTUATOR_BIG )
Pad.Reserve( pad, PAD_ACTUATOR_SMALL )

function OnUpdate()
    if ( smallEarthquake == true ) then
        Pad.SetActuator( pad, PAD_ACTUATOR_SMALL, 0.5)
    elseif ( bigEarthquake == true ) then
        Pad.SetActuator( pad, PAD_ACTUATOR_BIG, 1.0)
    end
end
```

See Also

- Pad.CreatePads
- Pad.GetPad
- Pad.Reserve

Pad.UnReserve

Brief

Relinquishes the previously reserved action ID.

Definition

```
Pad.UnReserve( Pad pad, number actionId )
```

Arguments

pad - The pad to affect.actionId - The action ID to unreserve.

Return Values

None.

Description

Relinquishes the previously reserved action ID.

Examples

```
-- Initialization code
Pad.CreatePads()
pad = Pad.GetPad()
FIRE_LASER = 5
Pad.Reserve( pad, PAD_CROSS, FIRE_LASER )

-- Termination code
Pad.UnReserve( pad, FIRE_LASER )
```

See Also

- Pad.CreatePads
- Pad.GetPad
- Pad.Reserve

Pad.UnReserveAll

Brief

Relinquishes any previously reserved action IDs.

Definition

```
Pad.UnReserveAll(Pad pad)
```

Arguments

pad - The pad to affect.

Return Values

None.

Description

Relinquishes any previously reserved action IDs.

Examples

```
Pad.UnReserveAll(pad)
```

See Also

- Pad.Reserve
- Pad.UnReserve

Pad.WasAnyJustPressed

Brief

Detects whether any reserved pad button was pressed.

Definition

```
boolean Pad.WasAnyJustPressed( Pad pad )
```

Arguments

pad - The pad to query.

Return Values

true if any reserved button was just pressed and false otherwise.

Description

Detects whether any reserved pad button was pressed.

Examples

```
-- Initialization code
Pad.CreatePads()
pad = Pad.GetPad()
FIRE_LASER = 5
FIRE_MINIGUN = 6
Pad.Reserve( pad, PAD_SQUARE, FIRE_LASER )
Pad.Reserve( pad, PAD_CIRCLE, FIRE_MINIGUN )

function OnUpdate()
    if Pad.WasAnyJustPressed( pad ) then
        print( "A weapon was fired!" )
    end
end
```

See Also

- Pad.CreatePads
- Pad.GetPad
- Pad.Reserve

Pad.WasJustPressed

Brief

Detects whether the specified reserved pad button was pressed.

Definition

```
boolean Pad.WasJustPressed( Pad pad, number actionId )
```

Arguments

pad - The pad to query.actionId - The user defined action ID corresponding to a button ID.

Return Values

true if the button was pressed and false otherwise.

Description

Detects whether the reserved pad button associated with the specified action ID was pressed. Note that analogue sticks moving outside their dead zone counts as a button press.

Examples

```
-- Initialization code
Pad.CreatePads()
pad = Pad.GetPad()
FIRE_LASER = 5
Pad.Reserve( pad, PAD_CROSS, FIRE_LASER )

function OnUpdate()
    if Pad.WasJustPressed( pad, FIRE_LASER ) then
        print( "Laser fired successfully!" )
    end
end
```

See Also

- Pad.CreatePads

- Pad.GetPad
- Pad.Reserve

Pad.WasJustReleased

Brief

Detects whether the specified reserved pad button was released.

Definition

```
boolean Pad.WasJustReleased( Pad pad, number actionId )
```

Arguments

pad - The pad to query.actionId - The user defined action ID corresponding to a button ID

Return Values

true if the button was just released and false otherwise.

Description

Detects whether the reserved pad button associated with the specified action ID was released. Note that analogue sticks returning to their dead zone counts as a button release.

Examples

```
-- Initialization code
Pad.CreatePads()
pad = Pad.GetPad()
FIRE_LASER = 5
Pad.Reserve( pad, PAD_CROSS, FIRE_LASER )

function OnUpdate()
    if Pad.WasJustReleased( pad, FIRE_LASER ) then
        print( "Laser deactivated!" )
    end
end
```

See Also

- Pad.CreatePads
- Pad.GetPad
- Pad.Reserve

Person

Person.AttachToEntity

Brief

Attaches the person to specified entity so when the entity moves around the person moves with it.

Definition

```
Person.AttachToEntity( Person person, Entity entity = nil, Matrix44 localTransform = nil )
```

Arguments

entity - the entity to attach to or nil to detach player from current entity.

Return Values

None.

Description

When attached to an entity, setting position and rotation will affect the local transform only. The transform must be locked by the calling lua environment otherwise a lua error is generated. If the person is unlocked it automatically detaches from the entity.

Examples

```
person = LocalPlayer.GetPerson()
car = Entity.Create()
Person.AttachToEntity( person, car )
```

See Also

None

Person.DoAction

Brief

Plays an action on a Person

Definition

```
Person.DoAction( Person person, string actionName, boolean autoUnlock = false )
Person.DoAction( Person person, string actionName, boolean autoUnlock, number unlockFeatures )
```

Arguments

person - the person to play an action on.actionName - Name of the action to be performed. It can be a custom animation loaded remotely through
LocalPlayer.AddRepertoire as well as one of the following standard actions:
"agree"
"beckon"
"bow"
"cheer"
"clap"
"disagree"
"frustration"
"laugh"
"point"
"wave"
"stand_up_from_seat"autoUnlock - if true, then the animation lock on the person is automatically released after the action has completed. Defaults to false.unlockFeatures - only used if the autoUnlock parameter is true. If omitted then defaults to releasing the locks that were set when
Person.DoAction was called, otherwise will only release the locks specified.
features specified.

Return Values

Returns true if the action succeeds.

Description

The animation must be locked by the calling lua environment otherwise a lua error is generated. If the action isn't recognized a lua error is generated. If it fails and autoUnlock is true it unlocks immediately.

Examples

```
if Person.HasLock( person ) then
    Person.DoAction( person, "clap" )
end
```

See Also

- Person.Lock
- Person.Unlock
- Person.IsLocked
- Person.HasLock
- Person.SetBehavior
- Person.DoActionBroadcast

Person.DoActionBroadcast

Brief

Plays an action on a Person

Definition

```
Person.DoActionBroadcast( Person person, string actionName, boolean autoUnlock = false )
Person.DoActionBroadcast( Person person, string actionName, boolean autoUnlock, number unlockFeatures
)
```

Arguments

person - the person to play an action on.actionName - Name of the action to be performed. It can be a custom animation loaded remotely through
LocalPlayer.AddRepertoire as well as one of the following standard actions:
"agree"
"beckon"
"bow"
"cheer"
"clap"
"disagree"
"frustration"
"laugh"
"point"
"wave"
"stand_up_from_seat"autoUnlock - if true, then the animation lock on the person is automatically released after the action has completed. Defaults to false.unlockFeatures - only used if the autoUnlock parameter is true. If omitted then defaults to releasing the locks that were set when
Person.DoActionBroadcast was called, otherwise will only release the locks specified.
features specified.

Return Values

Returns true if the action succeeds.

Description

The animation must be locked by the calling lua environment otherwise a lua error is generated. If the action isn't recognized a lua error is generated. If it fails and autoUnlock is true it unlocks immediately. If calling DoActionBroadcast on the LocalPlayer, this function will broadcast the action to other clients in the scene. Using DoActionBroadcast impacts on the amount of network traffic, so avoid calling this function frequently.

Examples

```
if Person.HasLock( person ) then
    Person.DoActionBroadcast( person, "clap" )
end
```

See Also

- Person.Lock
- Person.Unlock
- Person.IsLocked
- Person.HasLock
- Person.SetBehavior
- Person.DoAction

Person.EnableChatBubbles

Brief

Sets chat bubbles on or off.

Definition

```
Person.EnableChatBubbles( Person person, boolean enabled )
```

Arguments

enabled - True to enable chat bubbles or false to disable.

Return Values

None.

Description

With this function, a script can allow chat bubbles to be switched on or off at will. Chat will still appear in the chat log as normal. Please note: In order to use this function, the is_allowed_chat_bubble_change property in the object's mini_game component must be set to true.

Examples

```
local person = LocalPlayer.GetPerson( );
if ( Person.Lock( person, PersonLockType.ChatBubbles ) == true ) then
    Person.EnableChatBubbles( person, false );
end
```

See Also

- Person.Lock
- Person.Unlock

Person.EnableCollision

Brief

Toggles collision detection for the person's avatar.

Definition

```
Person.EnableCollision( Person person, boolean value )
```

Arguments

person - the person to modify.value - Whether collision should be on or off.

Return Values

None.

Description

Toggles collision detection for the person's avatar. The collision must be locked by the calling lua environment otherwise a lua error is generated.

Examples

```
if Person.IsCollision( person ) then
    Person.EnableCollision( person, false )
end
```

See Also

- Person.IsCollision

Person.FindInInstance

Brief

Finds a person in the instance from the person ID.

Definition

```
Person Person.FindInInstance( string personId )
```

Arguments

personId - The person to find.

Return Values

A Person if found and nil otherwise.

Description

Finds a person in the instance from the specified person ID.

Examples

```
me = Person.FindInInstance( LocalPlayer.GetPersonId() )
```

See Also

- LocalPlayer.GetPersonId

Person.GetAnimReg

Brief

Returns the value of the specified animation register.

Definition

```
number|boolean|string Person.GetAnimReg(Person person, string registerName)
```

Arguments

person - person to query.registerName - the name of the animation register to get.

Return Values

The value of the register, with the type based on the register type.

Description

If the register doesn't exist or the animation register doesn't have Lua read access an error will be thrown.

Examples

```
fatness = person:GetAnimReg("Fatness")
```

See Also

- Person.SetAnimReg

Person.GetAvatarStatusType

Brief

Returns the current avatar status type of the person specified

Definition

```
number Person.GetAvatarStatusId( Person person )
```

Arguments

person - The person to query.

Return Values

The enumerated value of the current status type.

Description

The user can change the status of their avatar from the emotes menu. This method returns the current user status type. Changing the status type affects the facial animation (mood) used, and the status text above the avatar that remote users see. The possible status types are:
AvatarStatusType.Neutral
AvatarStatusType.Happy
AvatarStatusType.Sad
AvatarStatusType.Angry
AvatarStatusType.Confused
AvatarStatusType.Unwell
AvatarStatusType.Busy
AvatarStatusType.IsLookingForHelp
AvatarStatusType.WantToChat
AvatarStatusType.WantToPlayGame

Examples

```
me = LocalPlayer.GetPerson()
if Person.GetAvatarStatusType( me ) == AvatarStatusType.Happy then
    DoSomethingHappy()
end
```

See Also

- Person.GetMood

Person.GetBehavior

Brief

Returns the name of the current behavior playing on the person.

Definition

```
string Person.GetBehavior( Person person )
```

Arguments

person - The person to query.

Return Values

The name of the current behavior playing on the person.

Description

Returns the name of the current behavior playing on the person or nil if the behavior is a Home internal behavior. For a list of Home's public behaviors see Person.SetBehavior. For post 1.40 behaviour, if the behaviour belongs to the calling script environment, will only specify the name. If the behaviour belongs to another object, that has allowed its behaviours to be public, then will specify the object id, as well as the name. For pre-1.40 behaviour. For the local player, will only return the name of the behaviour, but for remote persons, will return the fully qualified name (objectId:name) - this is for backwards compatibility. If the behaviour is a system public behaviour, will return the name only.

Examples

```
behaviorName = Person.GetBehavior( person )
```

See Also

- Person.SetBehavior

Person.GetBehaviorCategory

Brief

Returns the specified person's behavior state.

Definition

```
string|nil Person.GetBehaviorCategory( Person person )
```

Arguments

person - The person to update.

Return Values

Returns the specified person's behavior state.

Description

Returns nil if in an internal behavior or be one of the following: "dancing" "posing" "sitting_on_floor" "sitting_on_seat" "standing" "moving" Behaviors should be catagorized into one of the above states. This function can be used to check if a person is dancing etc. without checking the animation behavior name.

Examples

```
if Person.GetBehaviorCategory( LocalPlayer.GetPerson() ) == 'dancing' then
    StartTheParty()
end
```

See Also

- Person.GetBehavior
- Person.SetBehavior

Person.GetBoneCount

Brief

Get the number of bones in the person's skeleton.

Definition

```
number Person.GetBoneCount(Person person)
```

Arguments

person - Person to query.

Return Values

Number of bones in the skeleton, or 0 if the skeleton is not available.

Description

Retrieves the number of bones/joints in the person's skeleton. If no skeleton is present or loaded, 0 is returned.

Examples

```
local boneCount = LocalPlayer.GetPerson():GetBoneCount()
```

See Also

- Person.GetBoneIndex

Person.GetBoneIndex

Brief

Find the index of a bone in the person's skeleton.

Definition

```
number Person.GetBoneIndex(Person person, string boneName)
```

Arguments

person - Person to query.boneName - Name of bone to locate.

Return Values

The index of the bone in the skeleton (in range 1 - GetBoneCount()), or 0 if not found.

Description

Retrieves the index of the specified bone/joint in the person's skeleton. The index is 1-based. If the bone cannot be found, or there is no skeleton attached or loaded, 0 is returned.

Examples

```
local leftHandIndex = LocalPlayer.GetPerson():GetBoneIndex("lefthand")
```

See Also

- Person.GetBoneCount
- Person.GetBoneLocalMatrix
- Person.GetBoneWorldMatrix

Person.GetBoneLocalMatrix

Brief

Get the local transform matrix for a bone in the skeleton.

Definition

```
Person.GetBoneLocalMatrix(Person person, string|number bone, Matrix44 result)
```

Arguments

person - Person to query.bone - Bone name or index.result (out) - Matrix44 value to store result into.

Return Values

None.

Description

Retrieve the local-space transform matrix for a specified bone in the skeleton. If the bone is not found an error will result. Retrieving the matrix by index will be more efficient if done more than once.

Examples

```
local mat = Matrix44.Create()
LocalPlayer.GetPerson():GetBoneLocalMatrix("lefthand", mat)
-- or
local leftHandIndex = LocalPlayer.GetPerson():GetBoneIndex("lefthand")
LocalPlayer.GetPerson():GetBoneLocalMatrix(leftHandIndex, mat)
```

See Also

- Person.GetBoneWorldMatrix
- Person.GetBoneIndex

Person.GetBoneWorldMatrix

Brief

Get the world transform matrix for a bone in the skeleton.

Definition

```
Person.GetBoneWorldMatrix(Person person, string|number bone, Matrix44 result)
```

Arguments

person - Person to query.bone - Bone name or index.result (out) - Matrix44 value to store result into.

Return Values

None.

Description

Retrieve the world-space transform matrix for a specified bone in the skeleton. If the bone is not found an error will result. Retrieving the matrix by index will be more efficient if done more than once.

Examples

```
local mat = Matrix44.Create()
LocalPlayer.GetPerson():GetBoneWorldMatrix("lefthand", mat)
-- or
local leftHandIndex = LocalPlayer.GetPerson():GetBoneIndex("lefthand")
LocalPlayer.GetPerson():GetBoneWorldMatrix(leftHandIndex, mat)
```

See Also

- Person.GetBoneLocalMatrix
- Person.GetBoneIndex

Person.GetCurrentAction

Brief

Returns the name of the current action the person is in, if any.

Definition

```
string Person.GetCurrentAction( Person person )
```

Arguments

person - The person to query.

Return Values

Returns the name of the current action.

Description

If the person is not in an action nil is returned. Home internal actions will also return nil. For a list of Home public actions see Person.DoAction.

Examples

```
actionName = Person.GetCurrentAction( LocalPlayer.GetPerson() )
```

See Also

- LocalPlayer.DoAction
- LocalPlayer.IsDoingAction

Person.GetGroundUserId

Brief

Return the userID of the collision on which the person is standing.

Definition

```
number Person.GetGroundUserId( Person person )
```

Arguments

person - The person to query

Return Values

The userID.

Description

Return the userID of the collision on which the person is standing. Check Person.IsOnGround before calling this function; if the person is not on the ground an error will occur.

Examples

```
if ( person:IsOnGround() ) then
    print( Person.GetGroundUserId( person ) )
end
```

See Also

- Person.IsOnGround

Person.GetHeight

Brief

Retrieve the avatar's height in meters.

Definition

```
number Person.GetHeight(Person person)
```

Arguments

person - the person to query.

Return Values

The height of the person's avatar in meters, or nil if the avatar is not yet loaded.

Description

The value returned is the height of the avatar when in the bind pose, given the current height settings, in meters. This will return nil if the avatar is currently a fallback.

Examples

```
if ( person:GetHeight() < 1.0 ) then print ("Hello shorty") end
```

See Also

None

Person.GetId

Brief

Returns the person's ID.

Definition

```
string Person.GetId( Person person )
```

Arguments

person - The person to query.

Return Values

The person's ID.

Description

Returns the person's ID. A person's ID value is unique per instance, but not across instances and is not persistent. That is, a person ID can be used within an instance to uniquely identify any person in that instance, but it should be assumed to be assigned at random to players entering the instance, and only valid while that player is in the instance. Players in another instance of the same scene will not be accessible by person ID. Players exiting and re-entering the instance will almost certainly be assigned a different person ID, but a person's ID will not change while they remain in the same instance.

Examples

```
me = LocalPlayer.GetPerson()
if Person.GetId( me ) ~= LocalPlayer.GetPersonId() then
    print( "Oh no!" )
end
```

See Also

- LocalPlayer.GetPerson
- LocalPlayer.GetPersonId

Person.GetIdsOfFriendsInInstance

Brief

Returns a table of people IDs in the current instance that are friends.

Definition

```
string[] Person.GetIdsOfFriendsInInstance()
```

Arguments

None

Return Values

A table of person ID strings.

Description

Returns a table of person IDs in the current instance that are friends.

Examples

```
ids = Person.GetIdsOfFriendsInInstance()
numFriends = #ids
print( "There are " .. numFriends .. " friends in the instance." )
```

See Also

None

Person.GetIdsOfRemotePeopleInInstance

Brief

Returns a table of all remote person IDs in the current instance.

Definition

```
string[] Person.GetIdsOfRemotePeopleInInstance()
```

Arguments

None

Return Values

A table of person ID strings.

Description

Returns a table of person IDs for all remote avatars in the current instance.

Examples

```
ids = Person.GetIdsOfRemotePeopleInInstance()
numAvatars = #ids + 1
print( "There are " .. numAvatars .. " avatars in the instance." )
```

See Also

None

Person.GetInstanceSessionMasterId

Brief

Returns the current instance's session master Person ID.

Definition

```
string Person.GetInstanceSessionMasterId()
```

Arguments

None

Return Values

A person ID of the session master or nil if not ready yet.

Description

Returns the current instance's session master Person ID. Each instance has a session master automatically assigned to it. When a new instance is created the first person in that instance will be the session master and if they leave then a new session master is chosen if other users are in the instance.

Examples

```
local sessionMasterId = Person.GetInstanceSessionMasterId()
local sessionMaster = nil
if sessionMasterId ~= nil then
    sessionMaster = Person.FindInInstance( sessionMasterId )
    SendNetworkMessageToRequestInstanceState( sessionMaster )
end
```

See Also

- User.GetSceneOwner

Person.GetInventoryItemObjectId

Brief

Obtain the object id of the currently deployed companion object.

Definition

```
string Person.GetInventoryItemObjectId(Person person)
```

Arguments

person - The person to query.

Return Values

The object ID of the deployed inventory item if present, and nil otherwise.

Description

Obtains the object id of the currently deployed companion object, or returns nil if one is not currently active.

Examples

```
if (Person.GetInventoryItemObjectId(person) == "12345678-12345678-12345678-12345678") then
    isAllowedIntoMiniGame = true
end
```

See Also

None

Person.GetLocalBounds

Brief

Retrieve the local-space bounds of a person.

Definition

```
boolean Person.GetLocalBounds(Person person, Vector4 min, Vector4 max)
```

Arguments

person - The person to query.min (out) - Vector4 value to receive the minimum extent.max (out) - Vector4 value to receive the maximum extent.

Return Values

True if the bounds were retrieved successfully, false if the person is not yet loaded.

Description

Get the local-space bounds of a person's model. This will reflect the user's clothing choices and height selection. If the person's avatar is not loaded, this function will return false.

Examples

```
local min = Vector4.Create()
local max = Vector4.Create()
if (LocalPlayer.GetPerson():GetLocalBounds(min, max)) then
    print("Local player avatar is", max:Y() - min:Y(), "metres tall")
end
```

See Also

- Person.GetWorldBounds

Person.GetMood

Brief

Returns the mood of the specified person.

Definition

```
MoodType Person.GetMood(Person person)
```

Arguments

person - the person to interrogate the mood value for.

Return Values

The enumerated value of the mood.

Description

The mood type is the facial animation that is used on the idling person. This is controlled by the user through the status option in the emotes menu. Different statuses can share the same mood, ie Confused and IsLookingForHelp statuses both use the same mood Confused. The enumerated type of MoodType is:
MoodType.Neutral
MoodType.Happy
MoodType.Sad
MoodType.Angry
MoodType.Confused

Examples

```
if (person:GetMood() == MoodType.Sad) then
    print("Why the long face?")
end
```

See Also

- Person.GetAvatarStatusType

Person.GetName

Brief

Returns the person's Online ID.

Definition

```
string Person.GetName( Person person )
```

Arguments

person - The person to query.

Return Values

The person's Online ID.

Description

Returns the person's Online ID, but only if you launch HomeDeveloper.self in online mode. If launching in offline mode, only the local player's Person will be accessible, and the function will return the string "Me".

Examples

```
me = LocalPlayer.GetPerson()
print( "My name is " .. Person.GetName( me ) )
```

See Also

- LocalPlayer.GetPerson

Person.GetPosition

Brief

Returns the position of the person.

Definition

```
Vector4 Person.GetPosition( Person person )
Person.GetPosition( Person person, Vector4 outVec )
```

Arguments

person - The person to query.outVec - The Vector4 to store the result to.

Return Values

The position of the person.

Description

Returns a vector containing the position of the specified person.

Examples

```
me = LocalPlayer.GetPerson()
print( "I am at: ", Person.GetPosition( me ) )
```

See Also

- LocalPlayer.GetPerson

Person.GetRigComponentObjectId

Brief

Returns the specified component type's object ID.

Definition

```
string Person.GetRigComponentObjectId( Person person, string componentType )
```

Arguments

person - The person to query.componentType - The component type to search for. It is case sensitive.

Return Values

The component's object ID or nil if not found.

Description

Returns the specified component type's object ID. The component must be loaded to identity its type.

Examples

```
me = LocalPlayer.GetPerson()
types = Person.GetRigComponentTypes( person )
print( 'type=' .. types[ 1 ] .. ' object id=' .. Person.GetRigComponentObjectId( person, types[ 1 ] )
)
```

See Also

- Person.GetRigComponentTypes

Person.GetRigComponentTypes

Brief

Returns a table of strings for the current rig's component types.

Definition

```
string[] Person.GetRigComponentTypes( Person person )
```

Arguments

person - The person to query.

Return Values

Table of strings for the rig component types.

Description

Returns a table of strings for the current rig's component types. The rig must be loaded to obtain the information.

Examples

```
me = LocalPlayer.GetPerson()
rigCompTypes = Person.GetRigComponentTypes( me )
if rigCompTypes ~= nil then
    for index, item in ipairs( rigCompTypes ) do
        print( 'type = ' .. item )
    end
end
```

See Also

- Person.GetRigComponentObjectId

Person.GetRigField

Brief

Returns the value of the field and nil if not found

Definition

```
string|number|boolean Person.GetRigField( Person person, string field )
```

Arguments

person - The person to query.field - The field to retrieve. Can be one of the following:
"sex" (returns the string "male" or "female")

Return Values

The value of the field and nil if not found. Supported types are boolean, string and number.

Description

Returns the value of the field and nil if not found. The rig must be loaded to get the correct result.

Examples

```
me = LocalPlayer.GetPerson()
print( "My gender is " .. Person.GetRigField( me, "sex" ) )
```

See Also

- LocalPlayer.GetPerson

Person.GetRotationY

Brief

Returns the y-axis rotation of the person.

Definition

```
number Person.GetRotationY( Person person )
```

Arguments

person - The person to query.

Return Values

The rotation in the y-axis of the person in degrees.

Description

Returns the rotation in the y-axis of the specified person in degrees.

Examples

```
me = LocalPlayer.GetPerson()
print( "My rotation is: " .. Person.GetRotationY( me ) )
```

See Also

- LocalPlayer.GetPerson

Person.GetUser

Brief

Returns the User object for this person.

Definition

```
User Person.GetUser( Person person )
```

Arguments

person - The person to query.

Return Values

The User object for the specified person.

Description

Returns the User object attached to the specified person.

Examples

```
local user = person:GetUser()
print(person:GetName() .. (user:IsFriend() and " is " or " is not ") .. "your friend!")
```

See Also

Person.GetVelocity

Brief

Returns the current velocity of the person.

Definition

```
Vector4 GetVelocity( Person person )
locity( Person person, Vector4 outVec )
```

Arguments

person - the person to queryoutVec - The Vector4 to store the result in

Return Values

Returns the current velocity of the person specified if no outVec is supplied.

Description

Returns a vector containing the velocity of the person specified.

Examples

```
local me = LocalPlayer.GetPerson()
elocity is: ", Person.GetVelocity( me ) )
```

See Also

None

Person.GetWidth

Brief

Retrieves the width factor of an avatar.

Definition

```
number Person.GetWidth(Person person)
```

Arguments

person - the person to query.

Return Values

The width factor of the person in the range [-1, 1], or nil if the avatar isn't yet loaded and available.

Description

Return the width factor of the avatar, which is in the range -1 to 1, with -1 being thin, 0 being average, and 1 being fat.

Examples

```
if ( Person.GetWidth() == 1) then BreakChair() end
```

See Also

None

Person.GetWorldBounds

Brief

Retrieve the world-space bounds of a person.

Definition

```
boolean Person.GetWorldBounds(Person person, Vector4 min, Vector4 max)
```

Arguments

person - The person to query.min (out) - Vector4 value to receive the minimum extent.max (out) - Vector4 value to receive the maximum extent.

Return Values

True if the bounds were retrieved successfully, false if the person is not yet loaded.

Description

Get the world-space bounds of a person's model. This will reflect the user's clothing choices and height selection. If the person's avatar is not loaded, this function will return false.

Examples

```
local min = Vector4.Create()
local max = Vector4.Create()
if (LocalPlayer.GetPerson():GetWorldBounds(min, max)) then
    print("Local player avatar is", max:Y() - min:Y(), "metres tall")
end
```

See Also

- Person.GetLocalBounds

Person.HasLock

Brief

true if the person is locked by the calling Lua environment and false otherwise.

Definition

```
boolean Person.HasLock( Person person )
boolean Person.HasLock( Person person , number lockFeatures, boolean any = false )
```

Arguments

person - the person to query.lockFeatures - specific lock features to test to see if they are locked by this lua environmentany - if lockFeatures is specified, then this parameter is used to indicate that if any of the features specified are locked, then return true
otherwise will only return true if all the features specified are locked.

Return Values

true if the person is locked by the calling Lua environment and false otherwise.

Description

true if the person is locked by the calling Lua environment and false otherwise.

Examples

```
if Person.HasLock( person ) == true then
    Person.DoAction( person, "Clap" )
end
```

See Also

- Person.Lock
- Person.Unlock
- Person.IsLocked
- Person.DoAction
- Person.SetBehavior

Person.InterpolatePosition

Brief

Allows the client to interpolate the position of the person in the frames between successive calls to InterpolatePosition.

Definition

```
Person.InterpolatePosition( Person person, Vector4 position, Vector4 velocity, number facingAngle,
number timeOffset = 0)
```

Arguments

person - the person to interpolate the position onposition - the position to use for the interpolationvelocity - the velocity to use for the interpolationfacingAngle - the angle the person is facingtimeOffset - the difference in time between the session clocks (in seconds) - if omitted then defaults to 0.0

Return Values

None.

Description

The calling lua environment must have atleast a Transform lock on the person for the call to succeed This call will only work on NPC persons (will throw a Lua error for the the local player person).

Examples

```
person:InterpolatePosition( recvdPos, recvdVel, recvdAngle, timeOffset )
```

See Also

- Person.Lock
- Person.GetPosition
- Person.GetVelocity

Person.IsCollision

Brief

Reports whether the person's collision is enabled.

Definition

```
boolean Person.IsCollision( Person person )
```

Arguments

person - the person to query.

Return Values

true if collision for the player is on and false otherwise.

Description

Reports whether the person's collision is enabled.

Examples

```
if Person.IsCollision( person ) then
    Person.EnableCollision( person, false )
end
```

See Also

- Person.EnableCollision

Person.IsLoaded

Brief

Indicates when a person's avatar is fully loaded and visible.

Definition

```
boolean Person.IsLoaded(Person person)
```

Arguments

person - The person to query.

Return Values

True if the person's avatar is fully loaded and ready, or false if still loading or invalid.

Description

When a person initially enters the scene, their avatar and associated components may still be loading in the background. Some functionality such as locking is not available until the avatar is ready, so this function can be used to query the loading state of the person. If the person object is invalid, this function will return false.

Examples

```
if (person:IsValid() and person:IsLoaded()) then
    if (not person:Lock(PersonLockType.All)) then
        print("Unable to get lock!")
    end
end
```

See Also

- Person.IsValid

Person.IsLocked

Brief

true if the person is locked by any lua environment and false otherwise.

Definition

```
boolean Person.IsLocked(Person person)
```

Arguments

person - the person to query.

Return Values

true if the person is locked by any lua environment and false otherwise.

Description

true if the person is locked by any lua environment and false otherwise.

Examples

```
if Person.IsLocked( person ) == true and Person.HasLock( person ) == false then
    print( 'Waiting to get the lock' )
end
```

See Also

- Person.Lock
- Person.Unlock
- Person.HasLock
- Person.DoAction
- Person.SetBehavior

Person.IsOnGround

Brief

Return whether the person is on the ground or not.

Definition

```
boolean Person.IsOnGround( Person person )
```

Arguments

person - The person to query

Return Values

true if on the ground, false otherwise.

Description

Return whether the person is on the ground or not.

Examples

```
print( Person.IsOnGround( person ) );
```

See Also

- Person.GetGroundUserId

Person.IsRigUsingComponent

Brief

Returns true if the specified object is being used and false otherwise.

Definition

```
boolean Person.IsRigUsingComponent( Person person, string objectId )
```

Arguments

person - The person to query.objectId - The object ID to test for its usage.

Return Values

true if the specified object is being used and false otherwise.

Description

Returns true if the specified object is being used and false otherwise. Does not require the specified object have have completed its loading for the function to succeed.

Examples

```
me = LocalPlayer.GetPerson()
smartJacketId = "853B950D-ACF84EE7-920D819B-5A9A6C16"
if Person.IsRigUsingComponent( me, smartJacketId ) then
    accessGranted = true
end
```

See Also

- LocalPlayer.GetPerson

Person.IsValid

Brief

Returns whether the handle to the person is valid or not.

Definition

```
boolean Person.IsValid( Person person )
```

Arguments

person - The person to query.

Return Values

Returns whether the handle to the person is valid or not.

Description

Returns whether the handle to the person is valid or not. Invalid persons can be caused if the person leaves the scene while you still have a handle to that person.

Examples

```
if Person.IsValid( randomPerson ) == false then
    person = GetAnotherPerson()
end
```

See Also

- Person.FindInInstance
- Person.IsLoaded

Person.IsVisible

Brief

Return whether the person is visible or not to the user.

Definition

```
boolean Person.IsVisible( Person person )
```

Arguments

None

Return Values

Return whether the person is visible or not

Description

If a user can see a Person in game this will return true, otherwise false. No lock is required since you are only querying the value.

Examples

```
Person.SetVisibile( person, true );
print( Person.IsVisible( person ) );
```

See Also

- Person.Lock

- Person.Unlock

Person.Lock

Brief

Locks a combination of the person's animation, collision state, transform, label or chatbubbles so it can be controlled by script rather than network messages or pad input.

Definition

```
boolean Person.Lock( Person person, number lockType = nil )
```

Arguments

person - the person to query.lockType - Can be one of the following or nil:
PersonLockType.Animation
PersonLockType.Transform
PersonLockType.Collision
PersonLockType.Visibility
PersonLockType.Label
PersonLockType.ChatBubbles
PersonLockType.EmoteMenu
PersonLockType.Wardrobe
PersonLockType.Default = PersonLockType.Animation + PersonLockType.Transform + PersonLockType.Collision + PersonLockType.Visibility
PersonLockType.All - all the lock types together
If lockType is nil then PersonLockType.Default is used.

Return Values

True if the Lua environment was able to obtain the lock (or already has the lock) on the person, otherwise false.

Description

A Lua error is generated if the lockType passed in aren't one of the possible values above. Lua errors are also generated if PersonLockType.Label or PersonLockType.ChatBubbles is used when not in a minigame which supports it. A lock can only be obtained for people currently joined to the active game, and will be released automatically for all people when the local player quits the game. A script should not rely on this automatic behaviour however, and locks should be explicitly released as soon as they are no longer required. *NB To prevent a delay in character loading causing scripts to fail, always ensure that calls to Person.Lock() return true before using any functions which require a locked character. If the function returns false, wait for a short period and then attempt to lock the character again. This can be performed in a co-routine if necessary. 1.55 Added PersonLockType.Wardrobe. This lock type can only be used on the Local Player person, otherwise a Lua error is thrown.

Examples

```
-- Example 1, locking specific types.
Person.Lock( person, PersonLockType.Transform ) -- Locked the transform
Person.Lock( person, PersonLockType.Collision ) -- Unlocked the transform and now locked the collision
Person.Lock( person, PersonLockType.Collision + PersonLockType.Visibility ) -- Now additionally locked the visibility

-- Example 2, locking transform, collision, animation and visibility.
if Person.Lock( person ) then
    Person.DoAction( person, "clap" )
end
```

See Also

- Person.Unlock
- Person.IsLocked
- Person.HasLock
- Person.DoAction
- Person.SetBehavior

Person.ResetCurrentStatusText

Brief

Reset the status text to show in the over-head person label.

Definition

```
Person.ResetCurrentStatusText( Person person )
```

Arguments

person - The person to update.

Return Values

None.

Description

Reset the status text to show in the over-head person label.

Examples

```
me = LocalPlayer.GetPerson()
Person.ResetCurrentStatusText( me )
```

See Also

- LocalPlayer.GetPerson

Person.SetAnimReg

Brief

Sets the value of the specified animation register for the specified person.

Definition

```
Person.SetAnimReg(Person person, string registerName, number|boolean|string value)
```

Arguments

person - person to set the animation register for.registerName - the name of the animation register to set.value - the value to set the register (the type should agree with the register type).

Return Values

None.

Description

If the register doesn't exist or does not have Lua write access, or the value type and the register type don't match, then an error will be thrown.

Examples

```
person:SetAnimReg("myReg", 0.3)
```

See Also

- Person.GetAnimReg

Person.SetBehavior

Brief

Sets the current behavior of the person

Definition

```
Person.SetBehavior( Person person, string behaviorName, number unlockPeriodSeconds = 0, boolean
snapAnimation = false )
Person.SetBehavior( Person person, string behaviorName, number unlockPeriodSeconds, boolean
snapAnimation, number unlockFeatures )
```

Arguments

person - The Person to set the behavior onbehaviorName - The name of the behavior.unlockPeriodSeconds - if great than 0 then the animation lock on the person is automatically released after
the period specified has elapsed. Defaults to 0.snapAnimation - if true, then removes blend from previous action/behavior (defaults false)unlockFeatures - only used if the unlockPeriodSeconds parameter is greater than 0. If omitted then defaults to releasing the locks that were set when
Person.SetBehavior was called, otherwise will only release the locks specified.

Return Values

Error code. BehaviorError.None if everything was fine, BehaviorError.Invalid if the behavior cannot be set, BehaviorError.NotReady if the person is not ready for setting the behavior

Description

The following behaviors Home behaviors are available for use:
sit_low
sit_mid
sit_high
sit_low_recline
sit_mid_recline
standing
Additionally, any behaviors defined by the developer within an objects repertoire are available for use by the developer. The animation must be locked by the calling lua environment otherwise a lua error is generated. If the behavior specified isn't recognised, then a lua error is thrown.

Examples

```
if Person.HasLock( person ) == true then
    Person.SetBehavior( person, "sit_mid" )
end
```

See Also

- Person.Lock
- Person.Unlock
- Person.IsLocked
- Person.HasLock
- Person.DoAction

Person.SetCurrentStatusText

Brief

Set the status text to show in the over-head person label.

Definition

```
Person.SetCurrentStatusText( Person person, string text, bool persist = false )
```

Arguments

person - The person to update.text - The status text to show in the over-head person label.persist - boolean which determines whether the text is displayed in picture captures.

Return Values

None.

Description

Set the status text to show in the over-head person label. The status text is used to indicate an activity that the user is currently partaking in. This function must not be used without first obtaining permission from your RHQ.

Examples

```
me = LocalPlayer.GetPerson()
Person.SetCurrentStatusText( me, "Racing!" )
```

See Also

None

Person.SetLabelText

Brief

Change the current text label displayed atop the avatar.

Definition

```
Person.SetLabelText( Person person, enumeration setting )
```

Arguments

setting - the current setting to use for player labels.
PersonLabelText.On
PersonLabelText.Off
PersonLabelText.Name

Return Values

None.

Description

With this function, a script can set the labels above a person to show full labels, just the name, or even switch them off.
Please note: In order to use this function, the is_allowed_label_change property in the object's mini_game component must be set to true.

Examples

```
local person = LocalPlayer.GetPerson( );
if ( Person.Lock( person, PersonLockType.Label ) == true ) then
    Person.SetLabelText( person, PersonLabelText.Off );
end
```

See Also

- Person.Lock
- Person.Unlock

Person.SetPosition

Brief

Sets the position of the the person, assuming the calling Lua environment owns the lock (with transformation attribute)

Definition

```
SetPosition( Person person, Vector4 position )
```

Arguments

position - the desired world position of the person

Return Values

None.

Description

The transform must be locked by the calling lua environment otherwise a lua error is generated.

Examples

```
Person.SetPosition( person, pos )
```

See Also

- Person.Lock
- Person.Unlock

Person.SetRotationY

Brief

Sets the rotation in the (world) Y axis of the person.

Definition

```
Person.SetRotationY( Person person, number angle)
```

Arguments

person - The Person to set.angle - The world angle in degrees.

Return Values

None.

Description

The transform must be locked by the calling lua environment otherwise a lua error is generated.

Examples

```
Person.SetRotationY( person, 180 )
```

See Also

- Person.Lock
- Person.Unlock

Person.SetVisible

Brief

Sets the visibility of the the person.

Definition

```
Person.SetVisible( Person person, boolean enable )
```

Arguments

enable - whether to switch visibility on or off for that person

Return Values

None.

Description

Sets the visibility of the the person. The visibility must be locked with visibility attribute PersonLockType.Visibility by the calling lua environment otherwise a lua error is generated. If the person is not valid the call is ignored.

Examples

```
Person.SetVisible( person, false )
```

See Also

- Person.Lock
- Person.Unlock
- Person.IsLocked
- Person.HasLock

Person.Unlock

Brief

Unlocks the features specified on the person. If all features are unlocked, then lock is released, allowing other Lua environments to get a lock.

Definition

Arguments

person - the Person to unlock.unlockFlags - the features to unlock. If 0 is passed in will return false.

Return Values

A true return value only indicates that the features requested to be unlocked were unlocked, which may mean that other features are still locked
and that the person is not yet free to be locked by another calling environment - Use if (Person.HasLock(person, PersonLockType.All)==false)
to verify if lock has been released on the person. *Return value behaviour has changed for 1.45 - see history for previous behaviour.

Description

If all features are unlocked, it then allows other lua environments to lock the person. The features are:
PersonLockType.Animation
PersonLockType.Transform
PersonLockType.Collision
PersonLockType.Visibility
PersonLockType.Label
PersonLockType.ChatBubbles
PersonLockType.EmoteMenu
PersonLockType.Default = PersonLockType.Animation + PersonLockType.Transform + PersonLockType.Collision + PersonLockType.Visibility
PersonLockType.All - all the lock types together

Examples

```
Person.Unlock( person , PersonLockType.Animation + PersonLockType.Transform )
```

See Also

- Person.Lock
- Person.IsLocked
- Person.HasLock
- Person.DoAction
- Person.SetBehavior

Phys3dWorld

Phys3dWorld.Create

Brief

Create a 3d physics world.

Definition

```
Phys3dWorld Phys3dWorld.Create( )
```

Arguments

None

Return Values

The world.

Description

Create a 3D physics world. By default, all rigid bodies in PS Home are loaded into the default physics world, shared by all loaded objects in the current scene. The dynamics engine itself is deterministic, but only if all steps involved in setting up the world and the rigid bodies within it are performed in the same order. As a result, in order for a script to implement a deterministic game involving dynamics it cannot rely solely on the default physics world, as this is being modified by other parties. This function is provided to enable a script to create a "private" physics world for use only by the current script, and thus implement a deterministic game if desired. Scripts should only use this functionality for this purpose, there is no other advantage to creating a physics world manually. Note that you must explicitly add new collision entities to the private world, by default all entities will still be added to the default world.

Examples

```
world = Phys3dWorld.Create()
entity = Entity.Create()
entity:SetCollision("collision", world)
```

See Also

- Entity.SetCollision

Phys3dWorld.Reset

Brief

This resets the world.

Definition

```
Phys3dWorld.Reset( Phys3dWorld world )
```

Arguments

world - The world.

Return Values

None.

Description

This in effect completely clears the physics world. Items would need to be re-added to the world. This is available to developers so they do not have to increase the number of Lua handles.

Examples

```
world = Phys3dWorld.Create( )
Phys3dWorld.Reset( world )
```

See Also

- Phys3dWorld.Create

Phys3dWorld.SetEnabled

Brief

This sets the world to enabled or disabled.

Definition

```
Phys3dWorld.SetEnabled( Phys3dWorld world, boolean enable )
```

Arguments

world - The world.enable - True to enable the world, false otherwise.

Return Values

None.

Description

A world is by default disabled, so items can be added and/or modified at will. When we want the simulation to run, we set the world to be enabled. In effect, items are paused when added to this world until it is enabled.

Examples

```
world = Phys3dWorld.Create( )
Phys3dWorld.SetEnabled( world, true )
```

See Also

- Phys3dWorld.Create

Quaternion

Quaternion.Add

Brief

Adds one quaternion to another, element by element.

Definition

```
Quaternion.Add( Quaternion destination, Quaternion source1, Quaternion source2 )
```

Arguments

destination - Quaternion to store the result of the addition.source1 - Quaternion to be added to source2.source2 - Quaternion to be added to source1.

Return Values

None

Description

Adds source1 and source2 element-wise, storing the result in destination.

Examples

```
result = Quaternion.Create()
q1 = Quaternion.Create( )
q2 = Quaternion.Create( )
Quaternion.Add( result, q1, q2 )        -- result = q1 + q2
```

See Also

- Quaternion.Create

Quaternion.Conjugate

Brief

Conjugate a quaternion.

Definition

```
Quaternion.Conjugate( Quaternion q )
```

Arguments

q - Quaternion to be conjugated.

Return Values

None

Description

Conjugate a quaternion.

Examples

```
q = Quaternion.Create()
Quaternion.Conjugate( q )
```

See Also

- Quaternion.Create

Quaternion.Copy

Brief

Copies the values of one quaternion into another.

Definition

```
Quaternion.Copy( Quaternion destination, Quaternion source )
```

Arguments

destination - The quaternion to copy to.source - The quaternion to be copied.

Return Values

None

Description

Copies the values of the source quaternion and stores them in the destination. Lua has reference semantics for user types, so using q1 = q2 will not copy values but rather make q1 reference q2.

Examples

```
q1 = Quaternion.Create()
q2 = Quaternion.Create()
Quaternion.Copy( q2, q1 )
```

See Also

- Quaternion.Create

Quaternion.Create

Brief

Creates a new Quaternion object using the specified values.

Definition

```
Quaternion Quaternion.Create( )
Quaternion Quaternion.Create( Quaternion source )
Quaternion Quaternion.Create( number x, number y, number z, number w )
Quaternion Quaternion.Create( Matrix44 rotationMatrix )
Quaternion Quaternion.Create( Vector4 axis, number angle )
```

Arguments

source - Quaternion whose values are used to initialize newly created quaternion.x - Number to be written into the first element of the quaternion.y - Number to be written into the second element of the quaternion.z - Number to be written into the third element of the quaternion.w - Number to be written into the fourth element of the quaternion.rotationMatrix - A rotation matrix to be converted to a quaternion.axis - Rotation axis.angle - Rotation angle about the rotation axis in degrees.

Return Values

A new Quaternion instance.

Description

Creates a new Quaternion object using the specified values.

Examples

```
-- Create a default identity (0, 0, 0, 1) quaternion
q1 = Quaternion.Create()

-- Create a quaternion with supplied values
q2 = Quaternion.Create( 0, 0, 0, 1 )

-- Create a quaternion using the values of another
q3 = Quaternion.Create( q2 )

-- Create a quaternion from a rotation matrix
Matrix44 rotation = Matrix44.Create()
q4 = Quaternion.Create( rotation )

-- Create a quaternion from a rotation axis and rotation angle
Vector4 axis = Vector4.Create( 0, 1, 0 )
angle = 45
q5 = Quaternion.Create( axis, angle )
```

See Also

None

Quaternion.Divide

Brief

Performs scalar division on a quaternion.

Definition

```
Quaternion.Divide( Quaternion destination, Quaternion source, number scalar )
```

Arguments

destination - Quaternion to store the result of the division.source1 - Quaternion to divide.scalar - The scalar to divide by.

Return Values

None

Description

Multiplies source by 1/scalar, storing the result in destination.

Examples

```
result = Quaternion.Create()
q1 = Quaternion.Create()
Quaternion.Divide( result, q1, 2 )
```

See Also

- Quaternion.Create

Quaternion.Dot

Brief

Performs a dot product between two quaternions.

Definition

```
number Quaternion.Dot( Quaternion src1, Quaternion src2 )
```

Arguments

src1 - First quaternion to take part in the dot product.src2 - Second quaternion to take part in the dot product.

Return Values

The dot product of the two supplied quaternions.

Description

Performs a dot product between two quaternions.

Examples

```
q1 = Quaternion.Create()
q2 = Quaternion.Create()
result = Quaternion.Dot2( q1, q2 )
```

See Also

None

Quaternion.Length

Brief

Calculates the magnitude of the specified quaternion.

Definition

```
number Quaternion.Length( Quaternion quaternion )
```

Arguments

quaternion - Source quaternion for the calculation.

Return Values

A number holding the magnitude of the specified quaternion.

Description

Calculates the magnitude of the specified quaternion.

Examples

```
q1 = Quaternion.Create()
print( "Quaternion magnitude is: " ..  Quaternion.Length( q1 ) )
```

See Also

None

Quaternion.Multiply

Brief

Multiplies two quaternion by one another or a quaternion by a scalar.

Definition

```
Quaternion.Multiply( Quaternion destination, Quaternion source1, Quaternion source2 )
Quaternion.Multiply( Quaternion destination, Quaternion source, number scalar )
Quaternion.Multiply( Quaternion destination, number scalar, Quaternion source )
```

Arguments

destination - Quaternion to store the result of the multiplication.source1 - The first quaternion argument for the multiplication.source2 - The second quaternion argument for the multiplication.scalar - Number to multiply quaternion components by.

Return Values

None

Description

Multiplies source1 and source2 or source and scalar, storing the result in destination.

Examples

```
result = Quaternion.Create()
v1 = Quaternion.Create()
v2 = Quaternion.Create()
Quaternion.Multiply( result, v1, v2 )
Quaternion.Multiply( result, result, 0.5 )
```

See Also

- Quaternion.Create

Quaternion.Negate

Brief

Negate the components of a quaternion.

Definition

```
Quaternion.Negate( Quaternion q )
```

Arguments

q - Quaternion to be negated.

Return Values

None

Description

Negate the components of a quaternion.

Examples

```
q = Quaternion.Create()
Quaternion.Negate( q )
```

See Also

- Quaternion.Create

Quaternion.Nlerp

Brief

Performs a normalized linear interpolation between two quaternions.

Definition

```
Quaternion.Nlerp( Quaternion destination, Quaternion source1, Quaternion source2, number t )
```

Arguments

destination - Quaternion to store the result of the nlerp. source1 - The first quaternion argument for the nlerp. source2 - The second quaternion argument for the nlerp. t - The interpolation parameter, in range [0, 1].

Return Values

None

Description

Performs a normalized linear interpolation between two quaternions. The shortest path rotation is taken.

Examples

```
result = Quaternion.Create()
q1 = Quaternion.Create( axis, 0 )
q2 = Quaternion.Create( axis, 90 )
Quaternion.Nlerp( result, q1, q2 )
```

See Also

- Quaternion.Create

Quaternion.Normalize

Brief

Normalizes the specified quaternion.

Definition

```
Quaternion.Normalize( Quaternion quaternion )
```

Arguments

quaternion - The quaternion to normalize.

Return Values

None

Description

Normalizes the specified quaternion.

Examples

```
q = Quaternion.Create()
Quaternion.Normalize( q )       -- note that this has no effect to the identity quaternion from Create
```

See Also

- Quaternion.Create

Quaternion.SetAxisAngle

Brief

Sets a quaternion to represent a given axis/angle rotation.

Definition

```
Quaternion.SetAxisAngle( Quaternion destination, Vector4 axis, number angle )
```

Arguments

destination - Quaternion to store the result of the slerp.axis - Rotation axis.angle - Angle of rotation about the rotation axis, in degrees.

Return Values

None

Description

Sets a quaternion to represent a given axis/angle rotation.

Examples

```
q = Quaternion.Create()
axis = Vector4.Create( 0, 1, 0 )
angle = 45
Quaternion.SetAxisAngle( q, axis, angle )
```

See Also

- Quaternion.Create
- Vector4.Create

Quaternion.SetIdentity

Brief

Sets the quaternion to the identity quaternion (0, 0, 0, 1).

Definition

```
Quaternion.SetIdentity( Quaternion quat )
```

Arguments

quat - Quaternion to be set to identity.

Return Values

None

Description

Sets the quaternion to the identity quaternion (0, 0, 0, 1).

Examples

```
q = Quaternion.Create()
Quaternion.SetIdentity( q )     -- this is redundant after using the default constructor however
```

See Also

- Quaternion.Create

Quaternion.SetRotationMatrix

Brief

Sets a quaternion to represent a given rotation matrix.

Definition

```
Quaternion.SetRotationMatrix( Quaternion destination, Matrix44 rotationMatrix )
```

Arguments

destination - Quaternion to store the result of the slerp.rotationMatrix - Rotation matrix.

Return Values

None

Description

Sets a quaternion to represent a given rotation matrix.

Examples

```
q = Quaternion.Create()
m = Matrix44.Create()
Matrix44.SetAxisRotation( m, 0.5, 0.5, 0, 90, MATRIX_REPLACE )
Quaternion.SetRotationMatrix( q, m )
```

See Also

- Matrix44.Create
- Matrix44.SetAxisRotation
- Quaternion.Create

Quaternion.SetW

Brief

Sets the w value of the quaternion.

Definition

```
Quaternion.SetW( Quaternion quaternion, number value )
```

Arguments

quaternion - Quaternion to be set.value - A numerical value that is copied into the w component of the quaternion.

Return Values

None

Description

Sets the w value of the quaternion.

Examples

```
q = Quaternion.Create()
Quaternion.SetW( q, 0 )
```

See Also

- Quaternion.Create

- Quaternion.SetX
- Quaternion.SetY
- Quaternion.SetZ

Quaternion.SetX

Brief

Sets the x value of the quaternion.

Definition

```
Quaternion.SetX( Quaternion quaternion, number value )
```

Arguments

quaternion - Quaternion to be set. value - A numerical value that is copied into the x component of the quaternion.

Return Values

None

Description

Sets the x value of the quaternion.

Examples

```
q = Quaternion.Create()
Quaternion.SetX( q, 0 )
```

See Also

- Quaternion.Create
- Quaternion.SetY
- Quaternion.SetZ
- Quaternion.SetW

Quaternion.SetXyzw

Brief

Sets a Quaternion object with the specified values.

Definition

```
Quaternion.SetXyzw( Quaternion q, number x, number y, number z, number w )
```

Arguments

q - Quaternion to be set. x - Number to be written into the first element of the quaternion. y - Number to be written into the second element of the quaternion. z - Number to be written into the third element of the quaternion. w - Number to be written into the fourth element of the quaternion.

Return Values

None

Description

Sets a Quaternion object with the specified values.

Examples

```
q = Quaternion.Create()
Quaternion.SetXyzw( q, 0, 0, 0, 1 )
```

See Also

- Quaternion.SetX
- Quaternion.SetY
- Quaternion.SetZ
- Quaternion.SetW

Quaternion.SetY

Brief

Sets the y value of the quaternion.

Definition

```
Quaternion.SetY( Quaternion quaternion, number value )
```

Arguments

quaternion - Quaternion to be set. value - A numerical value that is copied into the y component of the quaternion.

Return Values

None

Description

Sets the y value of the quaternion.

Examples

```
q = Quaternion.Create()
Quaternion.SetY( q, 0 )
```

See Also

- Quaternion.Create
- Quaternion.SetX
- Quaternion.SetZ
- Quaternion.SetW

Quaternion.SetZ

Brief

Sets the z value of the quaternion.

Definition

```
Quaternion.SetZ( Quaternion quaternion, number value )
```

Arguments

quaternion - Quaternion to be set. value - A numerical value that is copied into the z component of the quaternion.

Return Values

None

Description

Sets the z value of the quaternion.

Examples

```
q = Quaternion.Create()
Quaternion.SetZ( q, 0 )
```

See Also

- Quaternion.Create
- Quaternion.SetX
- Quaternion.SetY
- Quaternion.SetW

Quaternion.Slerp

Brief

Performs spherical linear interpolation between two quaternions.

Definition

```
Quaternion.Slerp( Quaternion destination, Quaternion source1, Quaternion source2, number t )
```

Arguments

destination - Quaternion to store the result of the slerp. source1 - The first quaternion argument for the slerp. source2 - The second quaternion argument for the slerp. t - The interpolation parameter, in range [0, 1].

Return Values

None

Description

Performs spherical linear interpolation between two quaternions.

Examples

```
result = Quaternion.Create()
q1 = Quaternion.Create( axis, 0 )
q2 = Quaternion.Create( axis, 90 )
Quaternion.Slerp( result, q1, q2, 0.5 )
```

See Also

- Quaternion.Create

Quaternion.Subtract

Brief

Subtracts one quaternion from another, element by element.

Definition

```
Quaternion.Subtract( Quaternion destination, Quaternion source1, Quaternion source2 )
```

Arguments

destination - Quaternion to store the result of the subtraction.source1 - The minuend for the subtraction.source2 - The subtrahend for the subtraction.

Return Values

None

Description

Subtracts source2 from source1 element-wise, storing the result in destination.

Examples

```
result = Quaternion.Create()
q1 = Quaternion.Create()
q2 = Quaternion.Create()
Quaternion.Subtract( result, q1, q2 )       -- result = q1 - q2
```

See Also

- Quaternion.Create

Quaternion.ToAxisAngle

Brief

Get the equivalent rotation in axis/angle representation.

Definition

```
Vector4, number Quaternion.ToAxisAngle( Quaternion q )
```

Arguments

q - Quaternion to be converted to axis angle representation.

Return Values

Vector4 - x,y,z rotation axis, w rotation angle in degrees

Description

Get the equivalent rotation in axis/angle representation.

Examples

```
q = Quaternion.Create()
axisAngle = Quaternion.ToAxisAngle( q )
```

See Also

- Quaternion.Create

Quaternion.W

Brief

Returns the w value of the quaternion.

Definition

```
number Quaternion.W( Quaternion q )
```

Arguments

q - Quaternion to be queried.

Return Values

The fourth element of the quaternion, the w component.

Description

Returns the w value of the quaternion.

Examples

```
q = Quaternion.Create()
print( "Value of W is: " .. Quaternion.W( q ) )
```

See Also

- Quaternion.Create
- Quaternion.X
- Quaternion.Y
- Quaternion.Z

Quaternion.X

Brief

Returns the x value of the quaternion.

Definition

```
number Quaternion.X( Quaternion q )
```

Arguments

q - Quaternion to be queried.

Return Values

The first element of the quaternion, the x component.

Description

Returns the x value of the quaternion.

Examples

```
q = Quaternion.Create()
print( "Value of X is: " .. Quaternion.X( q ) )
```

See Also

- Quaternion.Create
- Quaternion.Y
- Quaternion.Z
- Quaternion.W

Quaternion.Y

Brief

Returns the y value of the quaternion.

Definition

```
number Quaternion.Y( Quaternion q )
```

Arguments

q - Quaternion to be queried.

Return Values

The second element of the quaternion, the y component.

Description

Returns the y value of the quaternion.

Examples

```
q = Quaternion.Create()
print( "Value of Y is: " .. Quaternion.Y( q ) )
```

See Also

- Quaternion.Create
- Quaternion.X
- Quaternion.Z
- Quaternion.W

Quaternion.Z

Brief

Returns the z value of the quaternion.

Definition

```
number Quaternion.Z( Quaternion q )
```

Arguments

q - Quaternion to be queried.

Return Values

The third element of the quaternion, the z component.

Description

Returns the z value of the quaternion.

Examples

```
q = Quaternion.Create()
print( "Value of Z is: " .. Quaternion.Z( q ) )
```

See Also

- Quaternion.Create
- Quaternion.X
- Quaternion.Y
- Quaternion.W

Raycast

Raycast.GetEndPoint

Brief

The value of the end point that was passed in through Raycast.Start.

Definition

```
Vector4 Raycast.GetEndPoint( Raycast raycast )
Raycast.GetEndPoint( Raycast raycast, Vector4 outPoint )
```

Arguments

raycast - The raycast object.outPoint - (out) The vector to receive the end point.

Return Values

The value of the end point that was passed in through Raycast.Start.

Description

Returns the value of the end point that was passed in through Raycast.Start.

Examples

```
thread = coroutine.create( function( ) CastRay( ) end );
function CastRay( )
    ray = Collision.CreateRaycast( );
    rayStart = Vector4.Create( 0, 5, 0 );
    rayEnd = Vector4.Create( 0, -1, 0 );
    Raycast.Start( ray, rayStart, rayEnd, CollisionLayer.Raycast );
    while Raycast.IsReady( ray ) == false do
        coroutine.yield( );
    end

    myPoint = Raycast.GetEndPoint( ray );
    print( tostring( myPoint ) .. " should equal " .. tostring( rayEnd ) );
end

function Update( )
    if coroutine.status( thread ) ~= 'dead' then
        coroutine.resume( thread )
    end
end
```

See Also

- Raycast.Start
- Raycast.GetStartPoint
- Raycast.IsReady

Raycast.GetHitEntity

Brief

Obtain the entity hit by the raycast.

Definition

```
Entity Raycast.GetHitEntity( Raycast raycast )
```

Arguments

raycast - A completed raycast with HasHit() == true

Return Values

The hit entity, or nil if the raycast hit the world or other non-script object.

Description

If a raycast has hit an object, this function will return the entity that was hit if possible, or if the object was not created by script (such as the world, or other non-script-based collision), it will return nil.

Examples

```
thread = coroutine.create( function( ) CastRay( ) end );
function CastRay( )
    ray = Collision.CreateRaycast( );
    rayStart = Vector4.Create( 0, 5, 0 );
    rayEnd = Vector4.Create( 0, -1, 0 );
    Raycast.Start( ray, rayStart, rayEnd, CollisionLayer.Raycast );
    while Raycast.IsReady( ray ) == false do
        coroutine.yield( );
    end

    if Raycast.HasHit( ray ) == true then
        local ent = ray:GetHitEntity();
        if ent ~= nil then
            print( "Entity found at " .. ent:GetPosition( ) )
        end
    end
end

function Update( )
    if coroutine.status( thread ) ~= 'dead' then
        coroutine.resume( thread )
    end
end
```

See Also

- Raycast.Start
- Raycast.HasHit
- Raycast.IsReady

Raycast.GetHitPoint

Brief

Returns the position where the collision occurred if it hit something.

Definition

```
Vector4 Raycast.GetHitPoint( Raycast raycast )
Raycast.GetHitPoint( Raycast raycast, Vector4 outHitPoint )
```

Arguments

raycast - The raycast object.outHitPoint - (out) The vector to receive the hit point.

Return Values

Returns the position where the collision occurred if it hit something.

Description

Returns the position where the collision occurred if it hit something.

Examples

```
function CastRay( )
    ray = Collision.CreateRaycast( );
    rayStart = Vector4.Create( 0, 5, 0 );
    rayEnd = Vector4.Create( 0, -1, 0 );
    Raycast.Start( ray, rayStart, rayEnd, CollisionLayer.Raycast );
    while Raycast.IsReady( ray ) == false do
        coroutine.yield( );
    end
    if Raycast.HasHit( ray ) then
        print( ray:GetHitPoint( ) );
    end
end
```

See Also

- Raycast.Start
- Raycast.HasHit

Raycast.GetReflectedVector

Brief

Returns a normalized vector of the raycast reflected off the surface it hit.

Definition

```
Vector4 Raycast.GetReflectedVector( Raycast raycast )
Raycast.GetEndPoint( Raycast raycast, Vector4 outVector )
```

Arguments

raycast - The raycast object.outVector - (out) The vector to receive the reflected vector.

Return Values

The normalised reflected vector.

Description

Returns a normalised vector of the raycast reflected off the surface it hit. It does not do any further raycasts to see if the reflection collides with anything else. Note: If the hit point is equal to the start point a zero lengh vector is returned.

Examples

```
thread = coroutine.create( function( ) CastRay( ) end );
function CastRay( )
    ray = Collision.CreateRaycast( );
    rayStart = Vector4.Create( 0, 5, 0 );
    rayEnd = Vector4.Create( 0, -1, 0 );
    Raycast.Start( ray, rayStart, rayEnd, CollisionLayer.Raycast );
    while Raycast.IsReady( ray ) == false do
        coroutine.yield( );
    end

    if Raycast.HasHit( ray ) == true then
        mirror = Raycast.GetReflectedVector( ray );
        mirror:Multiply( mirror, rayStart:Length3( ) );
        print( tostring( mirror ) .. " is the mirror image of " .. tostring( rayStart ) .. "
(reflected off a surface normal " .. tostring( ray:GetSurfaceNormal( ) ) .. ")" );
    end
end

function Update( )
    if coroutine.status( thread ) ~= 'dead' then
        coroutine.resume( thread )
    end
end
```

See Also

- Raycast.Start
- Raycast.IsReady
- Raycast.HasHit

Raycast.GetStartPoint

Brief

Returns the value of the start point that was passed in through Raycast.Start.

Definition

```
Vector4 Raycast.GetStartPoint( Raycast raycast )
Raycast.GetStartPoint( Raycast raycast, Vector4 outPoint )
```

Arguments

raycast - The raycast object.outPoint - (out) The vector to receive the start point.

Return Values

The value of the start point that was passed in through Raycast.Start.

Description

Returns the value of the start point that was passed in through Raycast.Start.

Examples

```
thread = coroutine.create( function( ) CastRay( ) end );
function CastRay( )
    ray = Collision.CreateRaycast( );
    rayStart = Vector4.Create( 0, 5, 0 );
    rayEnd = Vector4.Create( 0, -1, 0 );
    Raycast.Start( ray, rayStart, rayEnd, CollisionLayer.Raycast );
    while Raycast.IsReady( ray ) == false do
        coroutine.yield( );
    end

    myPoint = Raycast.GetStartPoint( ray );
    print( tostring( myPoint ) .. " should equal " .. tostring( rayStart ) );
end

function Update( )
    if coroutine.status( thread ) ~= 'dead' then
        coroutine.resume( thread )
    end
end
```

See Also

- Raycast.Start
- Raycast.IsReady

Raycast.GetSurfaceNormal

Brief

Returns the normal of the surface that the raycast hit.

Definition

```
Vector4 Raycast.GetSurfaceNormal( Raycast raycast )
Raycast.GetEndPoint( Raycast raycast, Vector4 outNormal )
```

Arguments

raycast - The raycast object.outNormal - (out) The vector to receive the normal.

Return Values

The normal of the surface that the raycast hit.

Description

Returns the normal of the surface that the raycast hit.

Examples

```
thread = coroutine.create( function( ) CastRay( ) end );
function CastRay( )
    ray = Collision.CreateRaycast( );
    rayStart = Vector4.Create( 0, 5, 0 );
    rayEnd = Vector4.Create( 0, -1, 0 );
    Raycast.Start( ray, rayStart, rayEnd, CollisionLayer.Raycast );
    while Raycast.IsReady( ray ) == false do
        coroutine.yield( );
    end

    direction = rayStart - rayEnd;
    Vector4.Normalize3( direction );
    if Raycast.HasHit( ray ) == true and Vector4.Dot3( Raycast.GetSurfaceNormal( ray ), direction ) >=
0.99 then
        print("We hit the floor (surface normal is almost directly upwards)" );
    end
end

function Update( )
    if coroutine.status( thread ) ~= 'dead' then
        coroutine.resume( thread )
    end
end
```

See Also

- Raycast.Start
- Raycast.IsReady
- Raycast.HasHit

Raycast.GetSurfaceUserId

Brief

Retrieve the user ID of the collision geometry hit by the raycast.

Definition

```
number Raycast.GetSurfaceUserId( Raycast raycast )
```

Arguments

raycast - the raycast to query

Return Values

The user ID, or nil if none is available.

Description

If the raycast has hit, then this function will return the user ID of the collision surface that was hit. If no user ID is available, nil is returned.

Examples

```
thread = coroutine.create( function( ) CastRay( ) end );
function CastRay( )
    ray = Collision.CreateRaycast( );
    rayStart = Vector4.Create( 0, 5, 0 );
    rayEnd = Vector4.Create( 0, -1, 0 );
    Raycast.Start( ray, rayStart, rayEnd, CollisionLayer.Raycast );
    while Raycast.IsReady( ray ) == false do
        coroutine.yield( );
    end

    if Raycast.HasHit( ray ) == true then
        local userID = ray:GetSurfaceUserId( );
        print( "We hit a surface with ID: " .. tostring( userID ) );
    end
end

function Update( )
    if coroutine.status( thread ) ~= 'dead' then
        coroutine.resume( thread )
    end
end
```

See Also

- Raycast.HasHit
- Raycast.HasHit
- Raycast.IsReady

Raycast.HasHit

Brief

Returns true if the raycast hit something and false otherwise.

Definition

```
boolean Raycast.HasHit( Raycast raycast )
```

Arguments

None

Return Values

true if the raycast hit something and false otherwise.

Description

Returns true if the raycast hit something and false otherwise.

Examples

```
thread = coroutine.create( function( ) CastRay( ) end );
function CastRay( )
    ray = Collision.CreateRaycast( );
    rayStart = Vector4.Create( 0, 5, 0 );
    rayEnd = Vector4.Create( 0, -1, 0 );
    Raycast.Start( ray, rayStart, rayEnd, CollisionLayer.Raycast );
    while Raycast.IsReady( ray ) == false do
        coroutine.yield( );
    end
    if Raycast.HasHit( ray ) then
        print( "Hit!" );
    end
end
function Update( )
    if coroutine.status( thread ) ~= 'dead' then
        coroutine.resume( thread )
    end
end
```

See Also

- Raycast.Start
- Raycast.IsReady

Raycast.IsReady

Brief

Returns true if the raycast has finished and false if it hasn't started or is busy calculating.

Definition

```
boolean Raycast.IsReady( Raycast raycast )
```

Arguments

None

Return Values

true if the raycast has finished and false if it hasn't started or is busy calculating.

Description

Returns true if the raycast has finished and false if it hasn't started or is busy calculating.

Examples

```
thread = coroutine.create( function( ) CastRay( ) end );
function CastRay( )
    ray = Collision.CreateRaycast( );
    rayStart = Vector4.Create( 0, 5, 0 );
    rayEnd = Vector4.Create( 0, -1, 0 );
    Raycast.Start( ray, rayStart, rayEnd, CollisionLayer.Raycast );
    while Raycast.IsReady( ray ) == false do
        coroutine.yield( );
    end
end
function Update( )
    if coroutine.status( thread ) ~= 'dead' then
        coroutine.resume( thread )
    end
end
```

See Also

- Collision.CreateRaycast
- Raycast.Start

Raycast.Start

Brief

Starts a basic raycast.

Definition

```
Raycast.Start( Raycast raycast, Vector4 startPoint, Vector4 endPoint, CollisionLayer layer =
CollisionLayer.Raycast)
```

Arguments

startPoint - The start point of the raycast.endPoint - The end point of the raycast.layer - The collision layer of the raycast.
Valid layers are:
CollisionLayer.Entity
CollisionLayer.Avatar
CollisionLayer.Camera
CollisionLayer.Raycast
CollisionLayer.User1
CollisionLayer.User2
CollisionLayer.User3
CollisionLayer.User4

Return Values

None.

Description

Starts a basic raycast.

Examples

```
thread = coroutine.create( function( ) CastRay( ) end );
function CastRay( )
    ray = Collision.CreateRaycast( );
    rayStart = Vector4.Create( 0, 5, 0 );
    rayEnd = Vector4.Create( 0, -1, 0 );
    Raycast.Start( ray, rayStart, rayEnd, CollisionLayer.Raycast );
    while Raycast.IsReady( ray ) == false do
        coroutine.yield( );
    end
end
function Update( )
    if coroutine.status( thread ) ~= 'dead' then
        coroutine.resume( thread )
    end
end
```

See Also

- Collision.CreateRaycast
- Raycast.IsReady

ReceivedMessage

ReceivedMessage.GetEventMessage

Brief

Returns an instance of the message that is the focus of the current event.

Definition

```
ReceivedMessage ReceivedMessage.GetEventMessage()
```

Arguments

None

Return Values

The last message received as a ReceivedMessage.

Description

Returns an instance of the message that is the focus of the current event. Only valid during a message received event, as named by the 'on_received_message' property of the 'network' object component in the HDK's Object Editor application.

Examples

```
function OnReceivedMessage()
    local message = ReceivedMessage.GetEventMessage()
    data          = ReceivedMessage.ReadInt32( message )
end
```

See Also

- ReceivedMessage.ReadInt32

ReceivedMessage.GetRemainingSize

Brief

Returns the size of the unread data in the message.

Definition

```
number ReceivedMessage.GetRemainingSize( ReceivedMessage msg )
```

Arguments

msg - The instance of the received message to query.

Return Values

The number of bytes remaining in the message.

Description

Returns the size of the unread data in the message in bytes.

Examples

```
function OnReceivedMessage()
    local message = ReceivedMessage.GetEventMessage()
    bytesToRead   = ReceivedMessage.GetRemainingSize( message )
    print( bytesToRead ) -- Prints total size
    data          = ReceivedMessage.ReadInt32( message )
    bytesToRead   = ReceivedMessage.GetRemainingSize( message )
    print( bytesToRead ) -- Prints total size - 4
end
```

See Also

- ReceivedMessage.GetEventMessage

ReceivedMessage.GetSenderPersonId

Brief

Returns the person id of the sender of the message.

Definition

```
string ReceivedMessage.GetSenderPersonId( ReceivedMessage msg )
```

Arguments

msg - The instance of the received message to query.

Return Values

The sender's person id as a string.

Description

Returns the person id of the sender of the message.

Examples

```
function OnReceivedMessage()
    local message = ReceivedMessage.GetEventMessage()
    personid      = ReceivedMessage.GetSenderPersonId( message )
    print( personid )
end
```

See Also

- Person.FindInInstance
- Person.GetPosition
- ReceivedMessage.GetEventMessage

ReceivedMessage.GetTotalSize

Brief

Returns the total size of a received message.

Definition

```
number ReceivedMessage.GetTotalSize( ReceivedMessage msg )
```

Arguments

msg - The instance of the received message to query.

Return Values

The size in bytes of the message.

Description

Returns the total size of a received message in bytes.

Examples

```
function OnReceivedMessage()
    local message = ReceivedMessage.GetEventMessage()
    messagesize   = ReceivedMessage.GetTotalSize( message )
    print( messagesize )
end
```

See Also

- ReceivedMessage.GetEventMessage

ReceivedMessage.GetType

Brief

Returns the type of a ReceivedMessage.

Definition

```
number ReceivedMessage.GetType( ReceivedMessage msg )
```

Arguments

msg - The instance of the received message to query.

Return Values

The message type as a number.

Description

Returns the user-defined type of a ReceivedMessage. The type is a value that is set by the script when the message is created by the sender.

Examples

```
function OnReceivedMessage()
    local message = ReceivedMessage.GetEventMessage()
    messagetype   = ReceivedMessage.GetType( message )
    print( "Message type is " .. messagetype )
end
```

See Also

- ReceivedMessage.GetEventMessage

ReceivedMessage.ReadBigInt

Brief

Read a BigInt value from a received message.

Definition

```
ReceivedMessage.ReadBigInt( OutboundMessage msg, BigInt bigInt )
```

Arguments

msg - The message.bigInt - The BigInt to receive the value.

Return Values

None.

Description

Reads a BigInt value from a received message. Any size of BigInt is permissible.

Examples

```
function OnReceivedMessage()
    local message = ReceivedMessage.GetEventMessage()
    local bigInt = BigInt.Create(64)
    ReceivedMessage.ReadBigInt( message, bigInt )
    print( "Received big int:", bigInt )
end
```

See Also

- OutboundMessage.WriteBigInt
- BigInt.Create

ReceivedMessage.ReadFloat

Brief

Reads a 32-bit floating point value from the message.

Definition

```
number ReceivedMessage.ReadFloat( ReceivedMessage msg )
```

Arguments

msg - The instance of the received message to read from.

Return Values

32-bit floating point value read from the message packet.

Description

Reads a 32-bit floating point value from the message. When reading past the end of the message buffer, the function will silently fail and return 0.

Examples

```
function OnReceivedMessage()
    local message = ReceivedMessage.GetEventMessage()
    data          = ReceivedMessage.ReadFloat( message )
    print( "Received floating point value is " .. data )
end
```

See Also

- ReceivedMessage.ReadInt8
- ReceivedMessage.ReadInt16
- ReceivedMessage.ReadInt32
- ReceivedMessage.ReadString

ReceivedMessage.ReadInt8

Brief

Reads an 8-bit integer from the message.

Definition

```
number ReceivedMessage.ReadInt8( ReceivedMessage msg )
```

Arguments

msg - The instance of the received message to read from.

Return Values

8-bit integer read from the message packet.

Description

Reads an 8-bit integer (or byte) from the message. When reading past the end of the message buffer, the function will silently fail and return 0.

Examples

```
function OnReceivedMessage()
    local message = ReceivedMessage.GetEventMessage()
    data          = ReceivedMessage.ReadInt8( message )
    print( "Received byte value is " .. data )
end
```

See Also

- ReceivedMessage.ReadInt16
- ReceivedMessage.ReadInt32
- ReceivedMessage.ReadFloat
- ReceivedMessage.ReadString

ReceivedMessage.ReadInt16

Brief

Reads a 16-bit integer from the message.

Definition

```
number ReceivedMessage.ReadInt16( ReceivedMessage msg )
```

Arguments

msg - The instance of the received message to read from.

Return Values

16-bit integer read from the message packet.

Description

Reads a 16-bit integer (or short) from the message. When reading past the end of the message buffer, the function will silently fail and return 0.

Examples

```
function OnReceivedMessage()
    local message = ReceivedMessage.GetEventMessage()
    data          = ReceivedMessage.ReadInt16( message )
    print( "Received short value is " .. data )
end
```

See Also

- ReceivedMessage.ReadInt8
- ReceivedMessage.ReadInt32
- ReceivedMessage.ReadFloat
- ReceivedMessage.ReadString

ReceivedMessage.ReadInt32

Brief

Reads a 32-bit integer from the message.

Definition

```
number ReceivedMessage.ReadInt32( ReceivedMessage msg )
```

Arguments

msg - The instance of the received message to read from.

Return Values

32-bit integer read from the message packet.

Description

Reads a 32-bit integer from the message. When reading past the end of the message buffer, the function will silently fail and return 0.

Examples

```
function OnReceivedMessage()
    local message = ReceivedMessage.GetEventMessage()
    data          = ReceivedMessage.ReadInt32( message )
    print( "Received integer value is " .. data )
end
```

See Also

- ReceivedMessage.ReadInt8
- ReceivedMessage.ReadInt16
- ReceivedMessage.ReadFloat
- ReceivedMessage.ReadString

ReceivedMessage.ReadString

Brief

Reads a string value from the message.

Definition

```
number ReceivedMessage.ReadString( ReceivedMessage msg )
```

Arguments

msg - The instance of the received message to read from.

Return Values

String value read from the message packet.

Description

Reads a string value from the message. When reading past the end of the message buffer, the function will silently fail and return the empty string.

Examples

```
function OnReceivedMessage()
    local message = ReceivedMessage.GetEventMessage()
    data          = ReceivedMessage.ReadString( message )
    print( "Received string value is " .. data )
end
```

See Also

- ReceivedMessage.ReadInt8
- ReceivedMessage.ReadInt16
- ReceivedMessage.ReadInt32
- ReceivedMessage.ReadFloat

Renderer

Renderer.Camera2dGetPosition

Brief

Returns the current overlay's camera position as a Vector4.

Definition

```
Vector4 Renderer.Camera2dGetPosition( Renderer renderer )
Renderer.Camera2dGetPosition( Renderer renderer, Vector4 outPos )
```

Arguments

renderer - The renderer to query.outPos - (out) The vector to receive the position.

Return Values

The current camera's position.

Description

Returns the current overlay's camera position as a Vector4. Only the X and Y components of the returned vector will be valid.

Examples

```
overlayId = Renderer.GetOverlay( renderer )
pos = Renderer.Camera2dGetPosition( renderer )
print( "Camera for overlay " .. overlayId .. " has position " .. pos )
```

See Also

- Renderer.Camera2dSetPosition

Renderer.Camera2dGetRotation

Brief

Returns the current overlay's camera rotation.

Definition

```
number Renderer.Camera2dGetRotation( Renderer renderer )
```

Arguments

renderer - The renderer to query.

Return Values

The camera's rotation in degrees.

Description

Returns the current overlay's camera rotation in degrees.

Examples

```
overlayId = Renderer.GetOverlay( renderer )
rot = Renderer.Camera2dGetRotation( renderer )
print( "Camera for overlay " .. overlayId .. " has rotation " .. rot )
```

See Also

- Renderer.Camera2dSetRotation

Renderer.Camera2dGetScale

Brief

Returns the current overlay's camera scale.

Definition

```
Vector4 Renderer.Camera2dGetScale( Renderer renderer )
Renderer.Camera2dGetScale( Renderer renderer, Vector4 outScale )
```

Arguments

renderer - The renderer to query.outScale - (out) The vector to receive the scale.

Return Values

The camera's scale in the X and Y components of the returned vector.

Description

Returns the current overlay's camera scale. The Z and W components of the returned Vector4 set to 0.

Examples

```
overlayId = Renderer.GetOverlay( renderer )
s = Renderer.Camera2dGetScale( renderer )
print( "Camera for overlay " .. overlayId .. " has scale X=" .. s:X() .. " Y=" .. s:Y() )
```

See Also

- Renderer.Camera2dSetScale

Renderer.Camera2dSetPosition

Brief

Set the current overlay's camera position.

Definition

```
Renderer.Camera2dSetPosition( Renderer renderer, Vector4 position )
```

Arguments

renderer - The renderer to update.position - The new position for the camera.

Return Values

None.

Description

Set the current overlay's camera position to the specified vector. The Z and W components of the specified position are ignored.

Examples

```
Renderer.Camera2dSetPosition( renderer, Vector4.Create( -512, -360 ) )
```

See Also

- Vector4.Create

Renderer.Camera2dSetRotation

Brief

Set the current overlay's camera rotation.

Definition

```
Renderer.Camera2dSetRotation( Renderer renderer, number rotation )
```

Arguments

renderer - The renderer to update.rotation - The rotation in degrees.

Return Values

None.

Description

Set the current overlay's camera rotation in degrees.

Examples

```
Renderer.Camera2dSetRotation( renderer, 180.0 )
```

See Also

None

Renderer.Camera2dSetScale

Brief

Set the current overlay's camera scale.

Definition

```
Renderer.Camera2dSetScale( Renderer renderer, Vector4 scale )
Renderer.Camera2dSetScale( Renderer renderer, number x, number y )
```

Arguments

renderer - The renderer to update.scale - The 2D scale values as the X and Y components of the vector. The Z and W components are ignored.x - X scale value.y - Y scale value.

Return Values

None.

Description

Set the current overlay's camera scale. The scale values are set into a 2D "view" matrix, and thus the scaling of the result will be the reciprocal of the values. For example, setting scale to (0.5, 0.5) will double the size of the output, (2.0, 2.0) will halve it. Setting a scale of 0 will make the overlay invisible. Note that currently text will not be scaled (nor rotated).

Examples

```
Renderer.Camera2dSetScale( renderer, 0.5, 2.0 )
Renderer.Camera2dSetScale( renderer, Vector4.Create( 0.5, 2.0 ) )
```

See Also

- Renderer.Camera2dGetScale

Renderer.Create

Brief

Creates a new Renderer object.

Definition

```
Renderer Renderer.Create( Vector4 pos=[0, 0, 0, 0], Vector4 size=[1280, 720, 0, 0], Vector4 scale=[1,
1, 1, 1], type = 'normal' )
```

Arguments

pos - The position of the renderer relative to where its attached to.size - The size of the renderer relative to where its attached to.scale - The scale of the renderer relative to where its attached to.type - The type of the renderer. Can be one of the following:
"normal" - Automatically hidden when using the Menu Screen or in the Navigator etc.
"global" - Shows always, only hidden when wardrobe is active.
"scene" - Automatically hidden when using the wardrobe or in the Navigator but not hidden when it renders in the scene.

Return Values

A new Renderer instance.

Description

Creates a new Renderer object. If the renderer does not have a target set, it will default to being full screen with a size of 1280x720. If it is targeted at a screen, then the position is relative to that screen. Each renderer consists of 16 overlays and each overlay has its own camera. A use case for overlays can be having the HUD on one overlay, the game on another and the background sprites on a third.

Examples

```
renderer = Renderer.Create()
```

See Also

- Renderer.SetTarget

Renderer.DrawCircle2d

Brief

Draws a 2-dimensional circle.

Definition

```
Renderer.DrawCircle2d( Renderer renderer, Vector4 center, number radius, number segmentCount = 16,
Vector4 rgba = [1,1,1,1] )
Renderer.DrawCircle2d( Renderer renderer, number x, number y, number radius, number segmentCount = 16,
Vector4 rgba = [1,1,1,1] )
```

Arguments

renderer - Target renderer for the draw call.center - The center of the circle.x - The x position of the circle's center.y - The y position of the circle's center.radius - The radius of the circle.segmentCount - The number of segments to use to make the circle.rgba - RGBA values determine the color of the circle.

Return Values

None.

Description

Draws a 2-dimensional circle. The color defaults to white.

Examples

```
-- Render a blue circle
Renderer.DrawCircle2d( renderer, Vector4.Create( 0, 0 ), 60, 10, Vector4.Create( 0, 0, 1, 1 ) );
```

See Also

None

Renderer.DrawLine2d

Brief

Draws a 2-dimensional line.

Definition

```
Renderer.DrawLine2d( Renderer renderer, Vector4 pos1, Vector4 pos2, Vector4 rgba = [1,1,1,1] )
Renderer.DrawLine2d( Renderer renderer, number x1, number y1, number x2, number y2, Vector4 rgba =
[1,1,1,1] )
```

Arguments

renderer - Target renderer for the draw call.pos1 - The first point in the line. The Z and W components of the vector will be

ignored.pos2 - The second point in the line. The Z and W components of the vector will be ignored.x1 - The first point's X coordinate.y1 - The first point's Y coordinate.x2 - The second point's X coordinate.y2 - The second point's Y coordinate.rgba - RGBA values determine the color of the line.

Return Values

None.

Description

Draws a 2-dimensional line. The color defaults to white.

Examples

```
-- Render a blue line
Renderer.DrawLine2d( renderer, Vector4.Create( 0, 0 ), Vector4.Create( 100, 200 ), Vector4.Create( 0, 0, 1, 1 ) )
```

See Also

- Renderer.DrawText2d

Renderer.DrawLineList2d

Brief

Draws a list of 2D lines.

Definition

```
Renderer.DrawLineList2d( Renderer renderer, Vector4[] lineList, Vector4 rgba = [ 1, 1, 1, 1 ] )
```

Arguments

renderer - Target renderer for the draw call.lineList - Table of vector pairs defining the 2D line segments.rgba - Color of the line list. Defaults to white.

Return Values

None.

Description

Draws a list of 2D lines.

Examples

```
lineList = {}
lineList[1] = Vector4.Create( 10, 10 )
lineList[2] = Vector4.Create( 20, 20 )
Renderer.DrawLineList2d( renderer, lineList )
```

See Also

- Vector4.Create

Renderer.DrawQuad2d

Brief

Draw a single quad with specified vertex details.

Definition

```
Renderer.DrawQuad2d(Renderer renderer, table points, table uvs, table colors, Texture texture = nil)
```

Arguments

renderer - Renderer to use.points - An array of 8 numbers, representing 4 XY pairs, the corner vertices of the rect.uvs - An array of 8 numbers, representing 4 UV pairs, the texture coooordinates of the rect corners. Must be nil if no texture is specified.colors - An array of 4 Vector4 values, representing the color for each vertex. If nil, white (1, 1, 1, 1) is assumed.texture - Texture to use. If nil, no texture is applied and the UV coordinates must be nil.

Return Values

None.

Description

This function is a more flexible alternative to DrawRect2d. All 4 vertices of the quad are specified and the UV (texture) coordinates and colors can be set. Only one texture can be applied to the quad. DrawRect2d will use less memory and be more performant, so prefer that unless you require the additional options this function provides.

Examples

```
points = { -100, 100, 100, 100, 100, -100, -100, -100 }     -- 200x200 square centered on origin
uvs = { 0, 0, 1, 0, 1, 1, 0, 1 }
colors = { Vector4.Create(1, 0, 0, 1), Vector4.Create(1, 0, 0, 1), Vector4.Create(0, 0, 1, 1),
Vector4.Create(0, 0, 1, 1) }
renderer:DrawQuad2d(points, uvs, colors, texture)
```

See Also

- Renderer.DrawRect2d

Renderer.DrawQuads2d

Brief

Draws 2D quads using data contained within MemoryContainers.

Definition

```
Renderer.DrawQuads2d( Renderer renderer, number startVertex, number quadCount, Texture texture=nil,
Matrix44 matrix=nil, MemoryContainer positionData, MemoryContainer uvData=nil, MemoryContainer
colorData|Vector4 colorVec )
```

Arguments

renderer - The renderer to use.quadCount - The number of quads to render.texture - The texture to use or nil if untextured.matrix - The world matrix for the quads or nil if not required.positionData - A MemoryContainer containing positions for the quads. Each quad is made up of 4 vertices and each vertex has an X and Y position which are of type Float32.startVertex - The first vertex index to start from. A startVertex of 0 is the first vertex as MemoryContainer data starts at 0 as well. This applies to all MemoryContainers.uvData - A MemoryContainer containing the texture UVs for the quads. Each quad is made up of 4 vertices and each vertex has a U and V value which are of type Float32.colorData - A MemoryContainer containing the color for the quads. Each quad is made up of 4 vertices and each vertex has 4 color channels (Red, Gree, Blue, Alpha) which are type UInt8 and range from 0 to 255.colorVec - A vector to use for all the quads instead of using colorData.

Return Values

None.

Description

This function lets you draw multiple textured or untextured quads at once using a maximum of one texture with the quad data stored in the supplied memory containers. You can specify the offset of the memory containers and the number of quads you want to

draw so you could use the same set of memory containers and draw sections of the data with different matrices or textures. If a matrix is specified then all of the positions are transformed with the matrix. Example usages are custom fonts, especially where there is a large body of static text that you wish to render so one function can set up your memory containers once and then render it each frame with Renderer.DrawQuads2d. Another usage is to have static background data such as a shoot-em up or platform game where the memory containers are set up once and then you only need to do one call per frame to render it all. You can also update the memory containers dynamically for usages such as a 2d particle system or when you are required to renderer hundreds of sprites.

Note: The memory container set functions can take arrays of values to minimize the API calls when setting up your memory containers.

Examples

```
Renderer.DrawQuads2d( g_renderer, 0, g_quadCount, g_fontTexture, g_worldMatrix, g_positions, g_uvs,
g_cols )
```

See Also

- MemoryContainer.Create

Renderer.DrawRect2d

Brief

Draws a solid 2-dimensional rectangle.

Definition

```
Renderer.DrawRect2d( Renderer renderer, Vector4 bottomLeftPos, Vector4 topRightPos, Vector4 rgba =
[1,1,1,1], Texture texture = nil )
Renderer.DrawRect2d( Renderer renderer, number left, number bottom, number right, number top, Vector4
rgba = [1,1,1,1], Texture texture = nil )
```

Arguments

renderer - Target renderer for the draw call.bottomLeftPos - The bottom left position of the rectangle.topRightPos - The top right position of the rectangle.left - The left coordinate of the rectangle.right - The right coordinate of the rectangle.bottom - The bottom coordinate of the rectangle.top - The top coordinate of the rectangle.rgba - The color of the rectangle.texture - The texture to use.

Return Values

None.

Description

Draws a solid 2-dimensional rectangle. The color defaults to white and it is untextured unless a texture is specified.

Examples

```
-- A red rectangle is produced in this example:
renderer = Renderer.Create()
Renderer.DrawRect2d( renderer, Vector4.Create( 0, 0 ), Vector4.Create( 100, 200 ), Vector4.Create( 1,
0, 0, 1 ) );
```

See Also

- Renderer.Create

Renderer.DrawSprite

Brief

Draws a sprite.

Definition

```
Renderer.DrawSprite( Renderer renderer, Sprite sprite )
```

Arguments

renderer - Target renderer for the draw call.sprite - The sprite to draw.

Return Values

None.

Description

Draws a sprite.

Examples

```
renderer = Renderer.Create()
sprite = Sprite.Create()
Sprite.SetTexture( sprite, Texture.Find( "texture.dds" ) )
Renderer.DrawSprite( renderer, sprite )
```

See Also

- Renderer.Create
- Sprite.Create
- Sprite.SetTexture
- Renderer.DrawSprites

Renderer.DrawSprites

Brief

Draws an array of sprites.

Definition

```
Renderer.DrawSprites( Renderer renderer, Sprite[] sprites, number startIndex = 1, number endIndex =
#sprites )
```

Arguments

renderer - Target renderer for the draw call.sprites - A table of sprites to draw.startIndex - The index in the table to start drawing from. Defaults to 1.endIndex - The index in the table to stop drawing to. Defaults to the length of the table.

Return Values

None.

Description

Draws one or more sprites.

Examples

```
renderer = Renderer.Create()
sprites = { Sprite.Create(), Sprite.Create(), Sprite.Create(), Sprite.Create() }
Sprite.SetTexture( sprites[1], Texture.Find( "texture1.dds" ) )
Sprite.SetTexture( sprites[2], Texture.Find( "texture2.dds" ) )
Sprite.SetTexture( sprites[3], Texture.Find( "texture3.dds" ) )
Sprite.SetTexture( sprites[4], Texture.Find( "texture4.dds" ) )
Renderer.DrawSprites( renderer, sprites ) -- Draw all 4 sprites
Renderer.DrawSprites( renderer, sprites, 2, 4 ) -- Draw 2nd, 3rd and 4th sprite.
```

See Also

- Renderer.Create
- Sprite.Create
- Sprite.SetTexture
- Renderer.DrawSprite

Renderer.DrawText2d

Brief

Draws text using the default font.

Definition

```
Renderer.DrawText2d( Renderer renderer, Vector4 position, string text, Vector4 rgba = [1,1,1,1],
number maxWidth = -1 )
Renderer.DrawText2d( Renderer renderer, number x, number y, string text, Vector4 rgba = [1,1,1,1],
number maxWidth = -1 )
```

Arguments

renderer - Target renderer for the draw call.position - The x,y position of the text. The Z and W components of the vector will be ignored.x - The x position of the text.y - The y position of the text.text - The text to render.rgba - The color of the text.maxWidth - The maximum width in pixels the text can be before it goes onto the next line.

Return Values

None.

Description

Draws text using the default font. The color defaults to white.

Examples

```
Renderer.DrawText2d( renderer, Vector4.Create( 5, 12 ), "hello" )
```

See Also

- Vector4.Create

Renderer.DrawTriList2d

Brief

Renders a list of 2D triangles.

Definition

```
Renderer.DrawTriList2d( Renderer renderer, Vector4[] points, Vector4 color = [1,1,1,1] )
```

Arguments

renderer - Target renderer for the draw call.points - The list of corner coordinates for the triangles. Must be multiple of 3.color - The color for the triangles. Defaults to white.

Return Values

None.

Description

Renders a list of 2D triangles. The color for the triangles defaults to white.

Examples

```
vertList = {}
vertList[1] = Vector4.Create( 10, 10 )
vertList[2] = Vector4.Create( 20, 20 )
vertList[3] = Vector4.Create( 10, 20 )
green = Vector4.Create( 0, 1, 0, 1 )
Renderer.DrawTriList2d( renderer, vertList, green )
```

See Also

- Vector4.Create

Renderer.GetOverlay

Brief

Returns the current overlay.

Definition

```
number Renderer.GetOverlay( Renderer renderer )
```

Arguments

renderer - The renderer to query.

Return Values

The initial overlay is 0 and it can range from 0 to 15.

Description

Returns the current overlay.

Examples

```
-- Save the current overlay
previousOverlay = Renderer.GetOverlay( renderer )
Renderer.SetOverlay( renderer, uiOverlay )

-- Render UI

-- Restore old overlay
Renderer.SetOverlay( renderer, previousOverlay )
```

See Also

- Renderer.SetOverlay

Renderer.GetPosition

Brief

Returns the current position of the renderer.

Definition

```
Vector4 Renderer.GetPosition( Renderer renderer )
Renderer.GetPosition( Renderer renderer, Vector4 outPos )
```

Arguments

renderer - The renderer to query.outPos - (out) The vector to receive the position.

Return Values

The current position of the renderer.

Description

Returns the current position of the renderer.

Examples

```
print( "The current position is " .. Renderer.GetPosition( renderer ) )
```

See Also

None

Renderer.GetScale

Brief

Returns the current scale of the renderer.

Definition

```
Vector4 Renderer.GetScale( Renderer renderer )
Renderer.GetScale( Renderer renderer, Vector4 outScale )
```

Arguments

renderer - The renderer to query.outScale - (out) The vector to receive the scale.

Return Values

The current scale of the renderer.

Description

Returns the current scale of the renderer.

Examples

```
print( "The current scale is " .. Renderer.GetScale( renderer ) )
```

See Also

None

Renderer.GetSize

Brief

Returns the current size of the renderer.

Definition

```
Vector4 Renderer.GetSize( Renderer renderer )
Renderer.GetSize( Renderer renderer, Vector4 outSize )
```

Arguments

renderer - The renderer to query.outSize - (out) The vector to receive the size.

Return Values

The current size of the renderer.

Description

Returns the current size of the renderer.

Examples

```
print( "The current size is " .. Renderer.GetSize( renderer ) )
```

See Also

None

Renderer.GetTextSize

Brief

Calculates the pixel width of the text using the current font.

Definition

```
Vector4 Renderer.GetTextSize( Renderer renderer, string text, number maxWidth = -1 )
```

Arguments

renderer - The renderer to query.text - The text to measure.maxWidth - The text to measure.

Return Values

A vector containing the pixel width and height of the text.

Description

Calculates the pixel width of the text using the current font. The size includes the current font's scale. Only the X and Y components of the returned vector are valid, where the X component corresponds to the specified text's width in pixels, and the Y

component corresponds to its height. The Z and W components are set to zero and can be ignored. If maxWidth is specified the size is calculated with a maximum width that mirrors the DrawText2d maxWidth parameter.

Examples

```
size = Renderer.GetTextSize( renderer, "Hello World!" )
print( "Text width is " .. Vector4.X( size ) )
print( "Text height is " .. Vector4.Y( size ) )
```

See Also

- Renderer.DrawText2d

Renderer.SetBlendMode2d

Brief

Sets the blend mode of the specified renderer's current overlay.

Definition

```
Renderer.SetBlendMode2d( Renderer renderer, number preset )
```

Arguments

renderer - The renderer to set the blend function on.preset - Blend mode preset to access well known definitions. Can be one of the following:
BlendMode2d.Blend
BlendMode2d.Add
BlendMode2d.Multiply
BlendMode2d.Xor

Return Values

Description

This works on the current overlay. All overlays default to BlendMode2d.Blend every frame. You can Change the blend mode state between render calls, for example changing the blend to additive, drawing several sprites, then changing the blend to multiply and drawing quads with Renderer.DrawQuads2d.

Examples

```
Renderer.SetBlendMode2d( renderer, BlendMode2d.Blend ); --Revert to normal blend mode
Renderer.SetBlendMode2d( renderer, BlendMode2d.Add ); --Set to additive
```

See Also

- Material.SetBlendMode

Renderer.SetFont

Brief

Sets the current font for rendering.

Definition

```
Renderer.SetFont( Renderer renderer, string fontName )
Renderer.SetFont( Renderer renderer, Font font )
```

Arguments

renderer - The renderer to use.fontName - The name of the font to use. Values can be one of the following:
"13reg" - the default for Renderer
"13bold"
"18reg"
"18bold"
"26reg"
"26bold"
"60bold"font - A valid custom Font object.

Return Values

None.

Description

Sets the current font for rendering. The default is "13reg".

Examples

```
local text = 'big font'
Renderer.SetFont( renderer, "60bold" )
local pos = Vector4.Create( 5 + Renderer.GetTextSize( text ) * 0.75, 12 )
Renderer.DrawText2d( renderer, pos, text )
...
font = Font.Create( myFontResource, 8 )     -- or Font.Create( "MyFontResourceName", 8 )
Renderer.SetFont( renderer, font )
Renderer.DrawText2d( renderer, textPos, "Check out my custom font!" )
```

See Also

- Font.Create

Renderer.SetFontHorzAlignment

Brief

Sets the horizontal alignment of the font when rendering text.

Definition

```
Renderer.SetFontHorzAlignment( Renderer renderer, string alignment )
```

Arguments

renderer - The renderer to update.alignment - The horizontal alignment of text to be renderered. Can be one of the following:
"left"
"center"
"right"

Return Values

None.

Description

Sets the horizontal alignment of the font when rendering text. When a renderer is created the default value is "left".

Examples

```
Renderer.SetFontHorzAlignment( renderer, "center" )
```

See Also

- Renderer.SetFontVertAlignment

Renderer.SetFontScale

Brief

Sets the font scale.

Definition

```
Renderer.SetFontScale( Renderer renderer, number scaleX, number scaleY = scaleX )
```

Arguments

renderer - The renderer to update.scaleX - The horizontal scale in the x-axis.scaleY - The vertical scale in the y-axis.

Return Values

None.

Description

Sets the font scale.

Examples

```
Renderer.SetFontScale( renderer, 10 )
```

See Also

- Renderer.DrawText2d

Renderer.SetFontVertAlignment

Brief

Sets the vertical alignment of the font when rendering text.

Definition

```
Renderer.SetFontVertAlignment( Renderer renderer, string alignment )
```

Arguments

renderer - The renderer to update.alignment - The vertical alignment of text to be renderered. Can be one of the following:
"top"
"center"
"bottom"

Return Values

None.

Description

Sets the vertical alignment of the font when rendering text. When a renderer is created the default value is "top".

Examples

```
Renderer.SetFontVertAlignment( renderer, "top" )
```

See Also

- Renderer.SetFontHorzAlignment

Renderer.SetOverlay

Brief

Sets the current overlay.

Definition

```
Renderer.SetOverlay( Renderer renderer, number overlay )
```

Arguments

renderer - The renderer to update.overlay - The overlay id which ranges from 0 to 15.

Return Values

None.

Description

Sets the current overlay. Overlays are drawn in the order 0 to 15.

Examples

```
-- Save the current overlay
previousOverlay = Renderer.GetOverlay( renderer )
Renderer.SetOverlay( renderer, uiOverlay )

-- Render UI

-- Restore old overlay
Renderer.SetOverlay( renderer, previousOverlay )
```

See Also

- Renderer.GetOverlay

Renderer.SetPosition

Brief

Sets the renderer to the specified position.

Definition

```
Renderer.SetPosition( Renderer renderer, Vector4 position )
```

Arguments

renderer - The renderer to update.position - The new position of the renderer.

Return Values

None.

Description

Sets the renderer to the specified position. The default is [0, 0, 0, 1].

Examples

```
renderer = Renderer.Create()
Renderer.SetPosition( renderer, Vector4.Create( -100, 42 ) )
```

See Also

- Renderer.Create
- Vector4.Create

Renderer.SetScale

Brief

Sets the renderer to the specified scale.

Definition

```
Renderer.SetScale( Renderer renderer, Vector4 scale )
```

Arguments

renderer - The renderer to update.scale - The new scale of the renderer.

Return Values

None.

Description

Sets the renderer to the specified scale. The default is [1, 1, 1, 1].

Examples

```
renderer = Renderer.Create()
Renderer.SetScale( renderer, Vector4.Create( 2, 2 ) )
```

See Also

- Renderer.Create
- Vector4.Create

Renderer.SetSize

Brief

Sets the renderer to the specified size.

Definition

```
Renderer.SetSize( Renderer renderer, Vector4 size )
```

Arguments

renderer - The renderer to update. size - The new size of the renderer.

Return Values

None.

Description

Sets the renderer to the specified size. The default is [1280, 720, 0, 0]. Anything outside of the size will be clipped.

Examples

```
renderer = Renderer.Create()
Renderer.SetSize( renderer, Vector4.Create( 1024, 720 ) )
```

See Also

- Renderer.Create
- Vector4.Create

Renderer.SetTarget

Brief

Sets the target of the renderer.

Definition

```
Renderer.SetTarget( Renderer renderer, Screen screen )
```

Arguments

renderer - The renderer to update. screen - The screen to render to.

Return Values

None.

Description

The target of the renderer is set to a screen so instead of rendering in full screen mode it is rendered to the specified screen.

Examples

```
renderer = Renderer.Create()
screen = ArcadeGame.GetScreen()
Renderer.SetTarget( renderer, screen )
```

See Also

- ArcadeGame.GetScreen
- Scene.FindScreen
- Screen.Create

Resource

Resource.CancelRequest

Brief

Cancels a pending resource request.

Definition

```
Resource.CancelRequest( Resource resource )
```

Arguments

resource - The resource to cancel the request for.

Return Values

None.

Description

Cancels a pending resource request.

Examples

```
Resource.CancelRequest( res )
```

See Also

- Resource.Request

Resource.DebugDumpToTty

Brief

Debug function to dump a resource content and/or metadata to TTY.

Definition

```
Resource.DebugDumpToTty(Resource resource)
```

Arguments

resource - The resource to dump.

Return Values

None.

Description

Dumps available information about the resource contents to TTY. File resources will be dumped in their entirety, binary resources will have any available human-readable metadata dumped. Note all resource types are currently supported, presently files/xml and textures are. This function is only available if the debug flag is set on the Lua component, and is not available to live content.

Examples

```
url = "http://www.example.com/data.xml"
xmlData1 = Resource.Request( url, "file" )
...
xmlData1:DebugDumpToTty()
```

See Also

- Resource.DebugSave

Resource.DebugSave

Brief

Debug function to save a resource to the home directory.

Definition

```
boolean Resource.DebugSave(Resource resource, string filename)
```

Arguments

resource - The resource to dump.filename - The filename to use (including extension).

Return Values

True on success or false on failure.

Description

Dumps the contents of a resource to the home directory supplied via Target Manager. The operation is synchronous so be aware that large files may involve a long pause. Currently only file/xml is supported. This function is only available if the debug flag is set on the Lua component, and is not available to live content.

Examples

```
url = "http://www.example.com/data.xml"
xmlData1 = Resource.Request( url, "file" )
...
xmlData1:DebugSave("debugXml.xml")
```

See Also

- Resource.DebugDumpToTty

Resource.Exists

Brief

Check if a specified resource exists in the object or a loaded resource pack.

Definition

```
boolean Resource.Exists(string resourceName, ResourcePack resourcePack = nil)
```

Arguments

resourceName - Name of the resource to search for.resourcePack - A loaded ResourcePack object to search in.

Return Values

True if the resource is present in the object or resource pack, or false otherwise.

Description

Checks for a named resource in the current object or a loaded resource pack.

Examples

```
if (Resource.Exists("gun")) then
    gunEntity:SetModel("gun")
end
```

See Also

- Resource.Find

Resource.Find

Brief

Finds the specified resource.

Definition

```
Resource Resource.Find( string resourceName, ResourcePack pack = nil )
```

Arguments

resourceName - The name of the resource to find.pack - ResourcePack from which to load the resource. Must be fully loaded.

Return Values

The Resource object.

Description

Finds the specified resource. The resource name is the name set for a resource in the HDK's Object Editor application. Alternatively, the resource name can be the name set for a scripted asset in the HDK's Scene Editor application. If the resource is not found an error occurs.

Examples

```
res = Resource.Find( "laserGun" )
print( Resource.GetName( res ) )
```

See Also

None

Resource.GetData

Brief

Depending on the type of resource it returns the corresponding object such as a texture.

Definition

```
variant Resource.GetData( Resource resource )
```

Arguments

resource - The resource to query.

Return Values

Depending on the type of resource it returns the corresponding object.

Description

Depending on the type of resource it returns the corresponding object such as a texture. Supported resource types are:
Texture
SoundBank
SoundStream
Texture
Repertoire

Examples

```
texture = Resource.GetData( textureRes )
textureWidth = Texture.GetWidth( texture )
textureHeight = Texture.GetHeight( texture )
```

See Also

- Resource.Request
- MemoryContainer.Create

Resource.GetDownloadProgress

Brief

Retrieve the download progress of a resource request.

Definition

```
number Resource.GetDownloadProgress( Resource resource )
```

Arguments

resource - A requested resource that is currently loading.

Return Values

A number between 0 and 1 representing the download progress.

Description

If the resource is not downloading 0 will be returned, or if it is fully downloaded 1 is returned. The IsLoading and IsLoaded functions should be preferred if simply checking for the load state. Note that the progress is the download progress only, other tasks may need to be performed after downloading and so the download progress may return 1 while IsLoaded is still false. Only when IsLoaded returns true is the load complete. Also note that a resource may be locally cached, and scripts should not assume that all requests will result in a download.

Examples

```
function OnUpdate()
    if (resource:IsLoading()) then
        local progress = resource:GetDownloadProgress()
        if (progress == 0) then
            print("Requesting...")
        elseif (0 < progress and progress > 1) then
            print("Downloading...")
            RenderProgressBar(progress)
        else
            print("Processing...")
        end
    end
end
```

See Also

- Resource.Request
- Resource.IsLoading
- Resource.IsLoaded

Resource.GetName

Brief

Returns the resource name.

Definition

```
string Resource.GetName( Resource resource )
```

Arguments

resource - The resource to query.

Return Values

The resource name.

Description

Returns the resource name. This function should only be used for resources obtained from a call to Resource.Find.

Examples

```
res = Resource.Find( 'laserGun' )
print( Resource.GetName( res ) )
```

See Also

- Resource.Find

Resource.GetStatusCode

Brief

Retrieve the status code for a requested resource download.

Definition

```
number Resource.GetStatusCode(Resource resource)
```

Arguments

resource - A requested resource that is currently loading.

Return Values

A status code (see below).

Description

For HTTP downloads, the status code will reflect the HTTP response status code, such as 404 in the case of file not found. The status code can be used to determine the cause of failure in the case that IsLoaded() == false and IsLoading() == false, which indicates an error has occurred. If no status code is available 0 is returned. Note that calling GetStatusCode in cases other than load failure is not guaranteed to produce an expected result.

Examples

```
resource = Resource.Request("http://www.example.com/myXmlFile.xml", "xml")
resourceDone = false

function OnUpdate()
    if (resource and not resourceDone) then
        if (resource:IsLoaded()) then
            -- use resource
            resourceDone = true
        elseif (not resource:IsLoading()) then
            -- error case: resource is not loaded but also not loading
            local code = resource:GetStatusCode()
            if (code == 404) then
                print("File has gone missing!")
            else
                print("HTTP download fail:", code)
            end
            resourceDone = true
        else
            print("Progress:", resource:GetDownloadProgress())
        end
    end
end
```

See Also

- Resource.Request
- Resource.GetDownloadProgress

Resource.GetSystemResource

Brief

Get a system resource for use by the script.

Definition

```
Resource Resource.GetSystemResource( SysResources resourceId )
```

Arguments

resourceId - ID of system resource to load (see details).

Return Values

The resource object.

Description

Some resources are loaded by the client and are available to all scripts. These resources will be treated as dynamically loaded (similarly to Resource.Request) but will be shared with the client and not require downloading. The available resource IDs are:

- SysResources.TexPsStoreLogo

Examples

```
resource = Resource.GetSystemResource( SysResources.TexPsStoreLogo )
...
if (resource:IsLoaded()) then
    local tex = resource:GetData()
    Renderer.DrawRect2d( renderer, Vector4.Create( 0, 0 ), Vector4.Create( 200, 50 ), Vector4.Create(
1, 1, 1, 1 ), tex )
end
```

See Also

- Resource.Request
- Resource.Find

Resource.IsLoaded

Brief

Returns true if the resource has been loaded successfully.

Definition

```
boolean Resource.IsLoaded( Resource resource )
```

Arguments

resource - The resource to query.

Return Values

true if the resource has been loaded successfully and false otherwise.

Description

Returns true if the resource has been loaded successfully.

Examples

```
local isLoaded = Resource.IsLoaded( res )
```

See Also

- Resource.IsLoading

Resource.IsLoading

Brief

Returns true while the resource is loading.

Definition

```
boolean Resource.IsLoading( Resource resource )
```

Arguments

resource - The resource to query.

Return Values

true if the resource is still loading or being requested and false otherwise.

Description

Returns true while the resource is loading.

Examples

```
local isLoading = Resource.IsLoading( res )
```

See Also

- Resource.IsLoaded

Resource.Release

Brief

Releases the specified resource.

Definition

```
Resource.Release( Resource resource )
```

Arguments

resource - The resource to be released.

Return Values

None.

Description

Releases the specified resource.

Examples

```
res = Resource.Find( 'laserGun' )
Resource.Release( res )
```

See Also

- Resource.Find

Resource.Request

Brief

Requests a new resource.

Definition

```
Resource Resource.Request( string|Screenshot content, string type, HttpPostData postData = nil, string
memory = nil, boolean useHttpCompression = false, boolean readBodyOnErrorStatus = false )
```

Arguments

content - The URL of the resource to request or a Screenshot object.type - The type of resource to request. Can be one of the following:
"file"
"texture"
"soundstream"postData - Optional. Post data to send up to the server when making a request.memory - Optional. Valid values are 'host' or 'main'.useHttpCompression - Boolean, indicates if HTTP compression should be used.readBodyOnErrorStatus - Boolean, indicates if the response body should be read if the status code indicates a problem (e.g. for a '410 gone', the body may be used to communicate a reason).

Return Values

A handle to the requested resource, or nil if the resource request failed (v01.45+).

Description

Requests a new resource. Only HTTP/HTTPS downloads are currently supported. Note that images loaded as textures do not currently have control over the compression or resizing of the image for use by PS Home, and as a result may display at a lower quality than the source image. If you wish to control the conversion options used when creating the texture, see Resource.RequestTexture.

Examples

```
-- Example 1: HTTP download of XML
url = "http://www.example.com/data.xml"
xmlData1 = Resource.Request( url, "file" )

-- Example 2: HTTPS download of XML generated by a server-sidew script
url = "https://www.example.com/script.php"
xmlData2 = Resource.Request( url, "file", postData, 'host' )

-- Example 3: After taking a screenshot load it up as a texture
if screenshot:Succeeded() == true then
  screenShotTexture = Resource.Request( screenshot, "texture" )
end

-- Example 4: HTTP download of a texture
url = "http://www.example.com/texture.dds"
xmlData1 = Resource.Request( url, "texture" )
```

See Also

- Screenshot.Create
- Resource.RequestTexture
- Resource.RequestSecure

Resource.RequestSecure

Brief

Requests a secure resource.

Definition

```
Resource Resource.RequestSecure( string contentId, string type, boolean global = false, MemoryType
memType = MemoryType.Main, boolean useHttpCompression = false, boolean readBodyOnErrorStatus = false )
```

Arguments

contentId - The ID (filename) of the secure resource to request. Note that as this call will
ultimately result in an HTTPS request, the contentId is case-sensitive.type - The type of resource to request. Can be one of the following:
"file"global - Request the non-region-specific resource.memType - Memory type to allocate for the resource.useHttpCompression - Boolean, indicates if HTTP compression should be used.readBodyOnErrorStatus - Boolean, indicates if the response body should be read if the status code indicates a problem (e.g. for a '410 gone', the body may be used to communicate a reason).

Return Values

A handle to the requested resource, or nil if the resource request failed.

Description

Requests a new secure resource. There are two secure storage locations, one for the user's current region and one for all regions. Specifying the global flag will request the non-region-specific file rather than the current region's file. If the secure service is not available this function will return nil, scripts should handle this case gracefully.

Examples

```
-- retrieve the global (non-region-specific) version of myfile.xml from secure storage
xmlData = Resource.RequestSecure( "myfile.xml", "file", true )
```

See Also

- Resource.Request

Resource.RequestTexture

Brief

Request a new resource.

Definition

```
Resource Resource.RequestTexture( string|Screenshot content, TextureOptions options =
TextureOptions.Default )
```

Arguments

content - The URL of the resource to request or a Screenshot object.options - The texture load options to use. Can be combined with the + operator. The options available are
TextureOptions.Default - the standard options, as per Resource.Request.
TextureOptions.NoCompression - do not DXT compress the texture
TextureOptions.NoMipmaps - do not generate mipmaps for the texture
TextureOptions.NoSizeLimit - do not cap the major dimension of the texture to 512 texels
TextureOptions.NoPow2 - do not convert the major dimension of the texture to a power of 2 in size
TextureOptions.Image - all of the above 4 options combined

Return Values

A handle to the requested resource, or nil if the resource request failed.

Description

Requests a new texture resource. Only HTTP/HTTPS downloads are currently supported. The default texture load options can cause an image to lose quality due to compression, size limiting and resizing. A script can use this function to specify the exact texture load options to use if required. In normal cases without strict quality requirements, Resource.Request should be preferred.

Examples

```
fullscreenImage = Resource.RequestTexture("http://www.mywebsite.com/HDBackground.jpg",
TextureOptions.Image)
hiResTexture = Resource.RequestTexture("http://www.mywebsite.com/HiResTexture.png",
TextureOptions.NoSizeLimit)
myScreenshot = Resource.RequestTexture(screenshotObject, TextureOptions.NoSizeLimit +
TextureOptions.NoCompression + TextureOptions.NoPow2 + TextureOptions.NoMipmaps)
```

See Also

- Resource.Request

Resource.Run

Brief

Executes the specified resource as a Lua script.

Definition

```
Resource.Run( string resourceName, ResourcePack pack = nil )
Resource.Run( Resource fileResource )
```

Arguments

resourceName - The name of the resource. The type of the resource must be 'lua'.resourcePack - The ResourcePack to search for the specified resourceName, if desired.fileResource - The resource to run. The type of the resource must be 'lua'.

Return Values

None.

Description

Executes the specified resource as a Lua script.

Examples

```
Resource.Run( "utils.lua" )
```

See Also

- LoadLibrary

Resource.StartLoading

Brief

Starts loading the resource if it is not already loaded.

Definition

```
boolean Resource.StartLoading( Resource resource, number memorySlotId )
```

Arguments

resource - The resource to be loaded.memorySlotId - The id of the memory slot to use or 0 for the default budget

Return Values

true if successful and false otherwise.

Description

Starts loading the resource if it is not already loaded. If a callback is specified, it will be passed a boolean value indicating whether the load succeeded.

Examples

```
res = Resource.Find( 'laserGun' )
Resource.StartLoading( res )
res2 = Resource.Find( 'mapSection2' )
Resource.StartLoading( res2, 2 )        -- load into memory slot 2
```

See Also

None

ResourcePack

ResourcePack.DebugGetLoadError

Brief

Get the error code for a load error.

Definition

```
RpLoadError ResourcePack.DebugGetLoadError( ResourcePack pack )
```

Arguments

pack - Resource Pack object.

Return Values

An RpLoadError value indicating the reason for the load failure:
RpLoadError.None - no error
RpLoadError.MountFail - object package could not be mounted
RpLoadError.OfflineFail - offline object could not be loaded
RpLoadError.DownloadFail - download of object package failed
RpLoadError.ResourcesFail - resources XML did not load or parse correctly
RpLoadError.ResourceAddFail - one of the object resources did not load
RpLoadError.LocalisationFail - localization XML did not load or parse correctly
RpLoadError.CreateDefFail - object definition could not be created
RpLoadError.ObjectDefFail - object XML did not load or parse correctly
RpLoadError.CreateFail - object create failed
RpLoadError.MakeFail - object type could not be found
RpLoadError.Component - one of the components failed to load (should not occur for RPs)
RpLoadError.NotAResourcePack - load references an object that is not an RP
RpLoadError.PermissionDenied - either requesting object or RP doesn't list object in permissions

Description

If GetLoadState indicates an error has occurred, this function can be used to retrieve the error state. This function is not available in live content and is not intended for display to a user, but can assist when developing the script.

Examples

```
myResPack = ResourcePack.Load( "DEADBEEF-00000000-00000000-00000000" )
if (not myResPack) then
    local e = myResPack:GetLoadError()

end
...
while (myResPack:GetLoadState() ~= RpLoadState.Loaded and myResPack:GetLoadState() ~=
RpLoadState.Error) do
    coroutine.yield()
end
if (myResPack:GetLoadState() == RpLoadState.Error) then
    local e = myResPack:GetLoadError()
end
```

See Also

- ResourcePack.GetLoadState
- ResourcePack.Load

ResourcePack.GetLoadProgress

Brief

Get the progress percentage of the current load state.

Definition

```
number ResourcePack.GetLoadProgress( ResourcePack pack )
```

Arguments

pack - Resource Pack object.

Return Values

A number in the range [0, 1] representing the proportion of the current load state that is complete.

Description

The Downloading and Loading states will each take a period of time to complete. A script should indicate to the user the progress of these steps individually so the load process is clearly understood. If this function is called for other states the value is likely to be fixed at 0 or 1 and should not be displayed to the user.

Examples

```
if ( loadState == RpLoadState.Downloading ) then
    myDownloadProgressBarGui:SetProgress( myResPack:GetLoadProgress() * 100.0 )     -- %
elseif ( loadState == RpLoadState.Loading ) then
    myLoadProgressBarGui:SetProgress( myResPack:GetLoadProgress() * 100.0 )     -- %
end
```

See Also

- ResourcePack.GetLoadState
- ResourcePack.Load

ResourcePack.GetLoadState

Brief

Get the current load state of a ResourcePack.

Definition

```
RpLoadState ResourcePack.GetLoadState( ResourcePack pack )
```

Arguments

pack - Resource Pack object.

Return Values

A RpLoadState enum indicating the current load state:
RpLoadState.Created
RpLoadState.Queued
RpLoadState.Downloading
RpLoadState.Loading
RpLoadState.Loaded
RpLoadState.Error

Description

The load of a ResourcePack is an asynchronous process which may take some time to complete. This function will indicate the current state of the ResourcePack load. Once the Loaded status is returned the object is ready to use. If this function indicates Error, the script can use GetLoadError to find out why the load failed. The Downloading and Loading states can be further queried for their progress using the GetLoadProgress function.

Examples

```
while (myResPack:GetLoadState() ~= RpLoadState.Loaded and myResPack:GetLoadState() ~=
RpLoadState.Error) do
    coroutine.yield()
end
if (myResPack:GetLoadState() == RpLoadState.Error) then
    print("ResourcePack failed to load!")
end
```

See Also

- ResourcePack.Load
- ResourcePack.GetLoadProgress

ResourcePack.IsLoading

Brief

Returns true if resource pack is currently downloading or loading.

Definition

```
ResourcePack.IsLoading(ResourcePack pack)
```

Arguments

pack - The ResourcePack object.

Return Values

True if the resource pack is download or loading, false otherwise.

Description

This function will check the load state and indicate whether the resource pack is downloading/loading. Resource packs that have completed loading or failed to load will return false.

Examples

```
if (myResPack:IsLoading()) then
    RenderProgressBar(myResPack:GetLoadState(), myResPack:GetLoadProgress())        -- user function
end
```

See Also

- ResourcePack.GetLoadState

ResourcePack.Load

Brief

Creates a ResourcePack and starts downloading the object.

Definition

```
ResourcePack ResourcePack.Load( string objectId, number memorySlotId = 0 )
```

Arguments

objectId - ID of resource pack object to load.memorySlotId - ID number of memory slot to load the resource pack into.

Return Values

A new ResourcePack object, or nil if object not found or could not be loaded.

Description

Initiates a background load of the specified ResourcePack object. The object ID must relate to a ResourcePack object for which this object has load permission, other object IDs will be refused. The load is asynchronous, so the script should query the progress using GetLoadState and GetLoadProgress, and display appropriate feedback to the user while the load process is taking place. If nil is returned the object ID is likely to be invalid, and the load request should not be retried. The script should call Unload when the ResourcePack is no longer required, ensuring that no resources remain in use.

Examples

```
myResPack = ResourcePack.Load( "11111111-11111111-11111111-11111111" )
if (not myResPack) then
    print("Error loading resource pack!")
    return
end
```

See Also

- ResourcePack.GetLoadState
- ResourcePack.GetLoadProgress
- ResourcePack.Unload

ResourcePack.Unload

Brief

Unloads the ResourcePack and all associated resources.

Definition

```
ResourcePack.Unload( ResourcePack pack )
```

Arguments

pack - Resource Pack object.

Return Values

None.

Description

Unloads the ResourcePack object and all resources referenced by it. Note that the script should ensure that no resources owned by the ResourcePack remain in use by PS Home as the client will not check for this. The ResourcePack Lua object remains valid and can be used to load a new ResourcePack. Garbage collected ResourcePack Lua objects will call this function automatically.

Examples

```
if (loadingNextLevel) then
    myResPack:Unload()
end
```

See Also

- ResourcePack.Load

Rewards

Rewards.AddTicket

Brief

Adds a reward ticket corresponding to the specified object ID to the list of tickets awaiting collection by the user.

Definition

```
boolean Rewards.AddTicket( string objectId, string callBackString = nil )
```

Arguments

objectId string - The ObjectID for which the ticket is allocated.callBackString - The optional call back string to run once the ticket has been processed.

Return Values

False if immediately failed to add the ticket, such as it being in the inventory already and true if successfully added. It may fail later though.

Description

This function assigns a reward ticket to a player which can award an object to a player instantly or can be used to collect a reward at a later point. The type of ticket given is determined by the entitlement_id fields in the object's metadata (edited using the HDK's Object Editor application). These are set per region and can be either AUTOMATIC_REWARD (instantly awarded), or LUA_REWARD (awarded using Rewards.CollectTicket()). If a callback is specified you can use Rewards.GetCurrentFocusObjectId and Rewards.GetCurrentFocusState inside the callback to check the result.

The PS Home client must be run in online mode in order for this function to work.

Examples

```
Rewards.AddTicket( "12345678-12345678-12345678-12345678" )
```

See Also

- Rewards.DoesHaveTicket
- Rewards.CollectTicket

Rewards.CollectTicket

Brief

Attempts to collect a reward ticket that must have been previously added to the user's reward ticket list.

Definition

```
boolean Rewards.CollectTicket( string objectId, string param, string callBackString = nil )
```

Arguments

objectId - The object ID of the object to collect as a reward.param - Reserved for future use.callBackString - Optional lua string thats run once the ticket has been collected or an error occurs in collection.

Return Values

true if call succeeds, or false if there is no ticket to collect.

Description

This function asks the PS Home Reward System to attempt to collect the reward ticket corresponding to the specified object ID. This will add the object to the user's permanent inventory. Note that PS Home must first retrieve data regarding the object from its content servers to verify that the item can be added, and so it is likely that the object will not appear in the user's inventory immediately. There are many conditions (such as the object not having been set to be rewarded through Lua script) that can mean the object is not actually added to the inventory. A return value of true from this function only indicates that the initial call was successful, it does not indicate that the user now owns the item. This can be verified by querying LocalPlayer.IsInInventory on successive frames. The PS Home client must be run in online mode in order for this function to work.

Examples

```
Rewards.CollectTicket( "12345678-12345678-12345678-12345678" )
```

See Also

- Rewards.DoesHaveTicket
- Rewards.AddTicket

Rewards.DoesHaveTicket

Brief

Checks whether a Reward Ticket for the specified ObjectID is currently awaiting collection.

Definition

```
boolean Rewards.DoesHaveTicket( string objectId )
```

Arguments

objectId - The object ID to check for.

Return Values

true if the reward ticket for the specified object ID is currently in the list awaiting collection.

Description

This function is used to determine when the user has been awarded a reward ticket but has not yet collected it. For example, an external title may award the ticket using a REWARDS.XML written to the PLAYSTATION3's hard drive, based on a certain condition within the external title. The title's developer could then develop a LUA script for PS Home, that checks if the reward ticket has been awarded, then shows appropriate feedback to the user before collecting the reward ticket. The PS Home client must be run in online mode in order for this function to work.

Examples

```
rewardObjId = "12345678-12345678-12345678-12345678"
if ( Rewards.DoesHaveTicket( rewardObjId ) ) then
    print("I have that ticket!")
end
```

See Also

- Rewards.AddTicket
- Rewards.CollectTicket

Rewards.GetCurrentFocusObjectId

Brief

Returns the object ID for the callback being called.

Definition

```
string Rewards.GetCurrentFocusObjectId()
```

Arguments

None

Return Values

The object ID in focus.

Description

Returns the object ID for the callback being called. Only valid when inside a callback. The PS Home client must be run in online mode in order for this function to work.

Examples

```
function RewardCallBack()
    local id = Rewards.GetCurrentFocusObjectId()
    if id == g_laserGun then
        local state = Rewards.GetCurrentFocusState()
        if state == 'collected' or state == 'added' then
            ShowLaserMessageBox()
        else
            ShowNoLaserForYouError()
        end
    end
end
```

See Also

- Rewards.CollectTicket
- Rewards.GetCurrentFocusState

Rewards.GetCurrentFocusState

Brief

Returns the state for the callback being called.

Definition

```
string Rewards.GetCurrentFocusState()
```

Arguments

None

Return Values

The state of callback in focus.

Description

Returns the state for the callback being called. Only valid when inside a callback. States returned are:
"collected" - The ticket has been collected.
"added" - The ticket has been added and ready for collection.
"noinfo" - There is no reward information for the object.
"failed" - A general error occurred.
"failed_automatic" - The collection failed as the reward is automaticly collected.
The PS Home client must be run in online mode in order for this function to work.

Examples

```
function RewardCallBack()
    local id = Rewards.GetCurrentFocusObjectId()
    if id == g_laserGun then
        local state = Rewards.GetCurrentFocusState()
        if state == 'collected' or state == 'added' then
            ShowLaserMessageBox()
        else
            ShowNoLaserForYouError()
        end
    end
end
```

See Also

- Rewards.GetCurrentFocusObjectId

RtGame

RtGame.AddCustomExitMenuOption

Brief

Add a custom menu item to the Exit Menu for the mini-game.

Definition

```
RtGame.AddCustomExitMenuOption(string optionText)
```

Arguments

optionText - The localized string to display. Note: This must contain the "[SQUARE]" tag to denote the SQUARE button selects this option. This will almost certainly be at the start of the text, and likely include a space after it, to separate it from the rest of the text. E.g. "[SQUARE] Settings".

Return Values

None.

Description

This enables the custom menu option in the mini-game's Exit Menu, and sets the text for that option. The text must be localized before input. Only one custom option can be added. Successive calls to this function can be used to alter the text displayed.

Examples

```
local customText = Object.GetMe():GetLocalizedText("myCustomOptionText")
RtGame.AddCustomExitMenuOption(myCustomOptionText)
```

See Also

- RtGame.RemoveCustomExitMenuOption
- RtGame.IsAbleToChangeCustomExitMenuOption

RtGame.CreateGame

Brief

Creates an RtGame session manually.

Definition

```
RtGame.CreateGame()
```

Arguments

None

Return Values

None.

Description

Creates an RtGame session. When the session is set up, a unique "room ID" is generated. This is available via the function RtGame.GetRoomId.

Examples

```
roomId = BigInt.Create(64)
if (not RtGame.IsJoined()) then
    RtGame.CreateGame()
    creating = true
end

function OnUpdate()
    if (creating and RtGame.IsJoined()) then
        RtGame.GetRoomId(roomId)
        creating = false
    end
end
```

See Also

- RtGame.JoinGame
- RtGame.IsJoined
- RtGame.GetRoomId

RtGame.CreateGameObject

Brief

Creates a RtGameObject instance.

Definition

```
RtGameObject RtGame.CreateGameObject( string objectType )
```

Arguments

objectType - The name of the object type defined within the network_definitions_xml file

Return Values

An RtGameObject instance, or nil if failed.

Description

Creates a RtGameObject instance. Once an object is created, initial values should be set before registering it for network replication. The RtGameObject fields can be accessed in normal Lua table format.

Examples

```
if IsSessionMaster() then
    session = RtGame.CreateGameObject( "gamestate" )
    session.gameMode = 0
    session.timeLeft = 10.0
    session.teamAScore = 0
    session.teamBScore = 0
    RtGame.ReplicateToNet( session )
end
```

See Also

- RtGame.ReplicateToNet

RtGame.DestroyGameObject

Brief

Destroys a RtGameObject instance.

Definition

```
RtGame.DestroyGameObject( RtGameObject rtGameObject )
```

Arguments

rtGameObject - the RtGameObject instance to be destroyed.

Return Values

none.

Description

Destroys a RtGameObject instance. Only RtGame objects owned by the local player can be destroyed.

Examples

```
RtGame.DestroyGameObject(myGameObject)
```

See Also

- RtGame.CreateGameObject

RtGame.GetConnectionInfo

Brief

Retrieves connection quality information for a member.

Definition

```
number RtGame.GetConnectionInfo( number memberId, number connectionInfoType )
```

Arguments

memberId - A player's member ID.connectionInfoType - Connection information value to obtain (see details).

Return Values

Connection info value requested, or nil on failure.

Description

Retrieves connection quality information for a member. connectionInfoType should be specified using the following enums:

- RtConnInfo.Rtt - Round-trip time (microseconds)
- RtConnInfo.Bandwidth - Estimate bandwidth (bits/second)
  Only one piece of information may be queried for each call to GetConnectionInfo.

Examples

```
for i = 1, RtGame.GetPlayerCount() do
    local memberId = RtGame.GetPlayer(i)
    if (memberId ~= nil) then
        local user = RtGame.GetUserFromMemberId(memberId)
        local rtt  = RtGame.GetConnectionInfo( memberId, RtConnInfo.Rtt )
        if (rtt ~= nil and user ~= nil) then
            print( string.format("Player %d: Name=%d, rtt=%d", i, user:GetName(), rtt) )
        end
    end
end
```

See Also

None

RtGame.GetMaxPlayers

Brief

Retrieves the maximum number of players for this session.

Definition

```
number RtGame.GetMaxPlayers()
```

Arguments

None

Return Values

Maximum number of players.

Description

Retrieves the maximum number of players for this session.

Examples

```
print( string.format("%d/%d players joined", RtGame.GetPlayerCount(), RtGame.GetMaxPlayers()) )
```

See Also

None

RtGame.GetMemberIdFromUser

Brief

Retrieves a member ID given a User object.

Definition

```
number RtGame.GetMemberIdFromUser( User user )
```

Arguments

user - A user to look up.

Return Values

MemberId associated with user, or nil if the user could not be located in the RT Game session.

Description

Retrieves a member ID given a User object if the user is part of the RT Game session.

Examples

```
friends = LocalPlayer.GetFriends()
for i, friend in ipairs( friends ) do
    local memberId = RtGame.GetMemberIdFromUser(friend)
    if (memberId ~= nil) then
        print( string.format("Friend %s is in game, with member ID %d", friend:GetName(), memberId) )
    end
end
```

See Also

None

RtGame.GetNumTokens

Brief

Retrieves the number of tokens available for this game.

Definition

```
number RtGame.GetNumTokens()
```

Arguments

None

Return Values

The maximum number of tokens.

Description

Retrieves the number of tokens available for this game. This will be the value set in the realtime_game component within the Object Editor.

Examples

```
print("Num tokens =", RtGame.GetNumTokens())
```

See Also

None

RtGame.GetOwnerMemberId

Brief

Retrieves the session owner's member ID.

Definition

```
number RtGame.GetOwnerMemberId()
```

Arguments

None

Return Values

Session owner's member ID, or nil if not in session or there is no owner.

Description

Retrieves the session owner's member ID.

Examples

```
function IsSessionOwner()
    return localPlayerMemberId == RtGame.GetOwnerMemberId()
end
```

See Also

None

RtGame.GetPlayer

Brief

Retrieves a member ID for a player in the session.

Definition

```
number RtGame.GetPlayer( number playerNumber )
```

Arguments

playerNumber - The index of the player slot to query, in the range [1, RtGame.GetPlayerCount()]

Return Values

memberId for player at given slot

Description

Retrieves a member ID for a player in the session.

Examples

```
for i = 1, RtGame.GetPlayerCount() do
    local memberId = RtGame.GetPlayer(i)
    print( string.format("Player %d: ID=%d", i, memberId) )
end
```

See Also

None

RtGame.GetPlayerCount

Brief

Retrieves the number of players currently participating in this session.

Definition

```
number RtGame.GetPlayerCount()
```

Arguments

None

Return Values

Number of players.

Description

Retrieves the number of players currently participating in this session.

Examples

```
print( string.format("%d/%d players joined", RtGame.GetPlayerCount(), RtGame.GetMaxPlayers()) )
```

See Also

None

RtGame.GetRoomId

Brief

Get the unique room ID of the currently joined RtGame session.

Definition

```
RtGame.GetRoomId(BigInt<64> roomId)
BigInt<64> RtGame.GetRoomId()
```

Arguments

roomId - a 64-bit BigInt in which to store the room ID.

Return Values

A 64-bit BigInt of the room ID if no BigInt was passed in.

Description

If the user is currently joined to an RtGame session, this function will return the room ID of the session.

Examples

```
roomId = BigInt.Create(64)
if (RtGame.IsJoined()) then
    RtGame.GetRoomId(roomId)
end
```

See Also

- RtGame.IsJoined
- RtGame.JoinGame

RtGame.GetSentInstanceTime

Brief

Retrieves the instance time of when the data was sent.

Definition

```
number, number RtGame.GetSentInstanceTime()
```

Arguments

None

Return Values

Instance time, split into 2 number values.

Description

Retrieves the instance time of when the data was sent for data receipt callbacks. See GetInstanceTime() for more details. This function is only valid within the following functions: on_object_replicated on_object_updated on_message

Examples

```
b,t = RtGame.GetSentInstanceTime()
print( string.format( "The instance time in hex is %#.2x%.6x", t, b ) )
```

See Also

- GetInstanceTime

RtGame.GetTokenOwnerMemberId

Brief

Queries the ownership of a token.

Definition

```
number RtGame.GetTokenOwnerMemberId( number tokenId )
```

Arguments

tokenId - Token number to query.

Return Values

Member ID of the token's owner, nil if the token is unowned.

Description

Queries the ownership of a token. Please note that the return value only reflects the current token state on the local client, an unowned token does not guarantee that a request of that token will succeed.

Examples

```
if (localMemberId == RtGame.GetTokenOwnerMemberId( weaponTokenId )) then
    print("You have the weapon!")
end
```

See Also

- RtGame.RequestToken
- RtGame.ReleaseToken

RtGame.GetUserFromMemberId

Brief

Retrieves a User object associated with a member ID.

Definition

```
User RtGame.GetUserFromMemberId( number memberId )
```

Arguments

memberId - A player's member ID.

Return Values

User object associated with a memberId, or nil on failure.

Description

Retrieves a User object associated with a memberId.

Examples

```
for i = 1, RtGame.GetPlayerCount() do
    local memberId = RtGame.GetPlayer(i)
    if (memberId ~= nil) then
        local user = RtGame.GetUserFromMemberId(memberId)
        if (user ~= nil) then
            print( string.format("Player %d: Name=%d", i, user:GetName()) )
        end
    end
end
```

See Also

None

RtGame.IsAbleToChangeCustomExitMenuOption

Brief

Determines if a custom menu option can currently be added to the Exit Menu.

Definition

```
RtGame.IsAbleToChangeCustomExitMenuOption()
```

Arguments

None

Return Values

Whether a custom menu option can currently be added to the Exit Menu.

Description

Changing the custom menu option while the Exit Menu is visible is not user fiendly and is therefore restricted. This function can be used to determine when 'AddCustomExitMenuOption' and 'RemoveCustomExitMenuOption' will fail.

Examples

```
RtGame.IsAbleToChangeCustomExitMenuOption()
```

See Also

- RtGame.AddCustomExitMenuOption
- RtGame.RemoveCustomExitMenuOption

RtGame.IsJoined

Brief

Indicates whether the local user is joined to an RtGame session.

Definition

```
boolean RtGame.IsJoined()
```

Arguments

None

Return Values

True if the local user is joined to an RtGame session, or false otherwise.

Description

Indicates whether the local user is joined to an RtGame session.

Examples

```
roomId = BigInt.Create(64)
if (not RtGame.IsJoined()) then
    RtGame.CreateGame()
    creating = true
end

function OnUpdate()
    if (creating and RtGame.IsJoined()) then
        RtGame.GetRoomId(roomId)
        creating = false
    end
end
```

See Also

- RtGame.CreateGame

- RtGame.JoinGame
- RtGame.GetRoomId

RtGame.IsSessionOpen

Brief

Queries whether the session is open.

Definition

```
boolean RtGame.IsSessionOpen()
```

Arguments

None.

Return Values

true if session is open, false otherwise.

Description

Queries whether the session is open. This function is valid only for the session owner. After a session ownership change event, the new session owner may use this function to query the session state.

Examples

```
function OnSessionOwnerChange( newOwnerId )
   print( "The session is open = ", tostring( RtGame.IsSessionOpen() ) )
end
```

See Also

- RtGame.SetSessionClosed

RtGame.JoinGame

Brief

Joins an existing RtGame session.

Definition

```
RtGame.JoinGame(BigInt<64> roomId)
```

Arguments

roomId - a 64-bit BigInt containing the room to join.

Return Values

None.

Description

Joins the local user to an existing RtGame session with the specified room ID. The user should not be connected to an RtGame session already.

Examples

```
-- roomId has been obtained from session owner...
if (not RtGame.IsJoined()) then
    RtGame.JoinGame(roomId)
    joining = true
end

function OnUpdate()
    if (joining and RtGame.IsJoined()) then
        joining = false
    end
end
```

See Also

- RtGame.CreateGame
- RtGame.GetRoomId

RtGame.Leave

Brief

Leave the session that the local user is part of.

Definition

```
RtGame.Leave()
```

Arguments

None

Return Values

None

Description

Leave the session that the local user is part of.

Examples

```
if (userWantsToQuit) then
    RtGame.Leave()
end
```

See Also

None

RtGame.ReleaseToken

Brief

Requests that ownership of a token is released.

Definition

```
boolean RtGame.ReleaseToken( number tokenId )
```

Arguments

tokenId - Token number to release.

Return Values

True if request submitted successfully, false if local user is not the owner of the token.

Description

Requests that ownership of a token is released. Token release requests are sent to the session owner. Use the on_token_ownership_change callback to detect if the request was successful. Only the token's owner or the session owner can request a token's release.

Examples

```
if (dropWeapon) then
    RtGame.ReleaseToken( weaponTokenId )
end
```

See Also

- RtGame.RequestToken

RtGame.RemoveCustomExitMenuOption

Brief

Remove the custom menu item from the Exit Menu for the mini-game (if one has been added).

Definition

```
RtGame.RemoveCustomTargetMenuOption(string optionText)
```

Arguments

None

Return Values

None.

Description

This removes the custom menu option in the mini-game's Exit Menu, if one has previously been added.

Examples

```
RtGame.RemoveCustomExitMenuOption()
```

See Also

- RtGame.AddCustomExitMenuOption
- RtGame.IsAbleToChangeCustomExitMenuOption

RtGame.ReplicateToNet

Brief

Marks an object for auto replication.

Definition

```
RtGame.ReplicateToNet( RtGameObject rtGameObject )
```

Arguments

rtGameObject - the RtGameObject instance to be replicated.

Return Values

None.

Description

Marks an object for auto replication. Once an object is created, it remains locally until this function is called. Once replicated to the network, the object will continue to be updated until it is destroyed or the player leaves the session.

Examples

```
if IsSessionMaster() then
    session = RtGame.CreateGameObject( "gamestate" )
    session.gameMode = 0
    session.timeLeft = 10.0
    session.teamAScore = 0
    session.teamBScore = 0
    RtGame.ReplicateToNet( session )
end
```

See Also

RtGame.RequestToken

Brief

Requests ownership of a token.

Definition

```
RtGame.RequestToken( number tokenId )
```

Arguments

tokenId - Token number to request.

Return Values

None.

Description

Requests ownership of a token. Tokens are requested from the session owner. Use the on_token_ownership_change callback to detect if the request was successful. As the session owner will have the advantage with token requests, it may be worth adding mechanisms within your script to make your game more fair. For example, delaying requests for the session owner.

Examples

```
RtGame.RequestToken( weaponTokenId )
```

See Also

- RtGame.ReleaseToken

RtGame.ReturnJoinGameResult

Brief

Returns the result for the custom join game step of the joining process.

Definition

```
RtGame.ReturnJoinGameResult( boolean result )
```

Arguments

result - true to allow the user to join, or false otherwise.

Return Values

None.

Description

When a callback function in on_join_game_request is specified, the system will wait for this call before proceeding with the game join process. Note that this call need not be made immediately within the callback, the script may perform asynchronous processing if necessary and call this function at a later point. In this case the script should provide an indication to the user that the join process is still in progress. See the on_local_player_join_request field within the realtime_game component of the object.

Examples

```
function OnLocalPlayerJoinRequest()
    RtGame.ReturnJoinGameResult(IsGameJoinable())
end
```

See Also

None

RtGame.SendMessage

Brief

Marks a message for sending

Definition

```
RtGame.SendMessage( string messageType, table message, number memberId = -1 )
```

Arguments

messageType - The name of the message as specified in the network_definition_file xml.message - The lua table containing all the message fields.memberId - MemberId of user to receive the message. This is an optional argument, omitting it or specifying -1 will send the message to all members.

Return Values

None.

Description

Marks a message for sending. Note: The message table does not need to persist after the call to this function.

Examples

```
local msg = { id = missleType, damage = 23, vX = getFacingX(), vY = getFacingY(), vZ = getFacingZ() }
RtGame.SendMessage( "fireMissile", msg )
```

See Also

None

RtGame.SetSessionClosed

Brief

Sets the session open/close state.

Definition

```
RtGame.SetSessionClosed( boolean close )
```

Arguments

close - True to close session, false to open.

Return Values

None.

Description

Sets the session open/close state. Once the session is closed, no other users may join. Note that this is a server request and therefore does not take immmediate effect, use the on_session_access_change callback to determine when the action is complete. This function is valid only for the session owner.

Examples

```
if (raceIsReadyToStart) then
    RtGame.SetSessionClosed(true)
end
```

See Also

None

RtGameObject

RtGameObject.GetInstanceId

Brief

Queries the object's owner.

Definition

```
number RtGameObject.GetInstanceId( RtGameObject obj )
```

Arguments

obj - The RT Game Object to query.

Return Values

Object's instance ID.

Description

Queries the object's instance ID. The ID returned is globally unique to the creator, across all objects of any type. This ID in combination with the owner will uniquely identify a given RT Game Object.

Examples

```
function OnObjectReplicated( rtGameObject )

    local owner = rtGameObject:GetOwner()
    local instanceId = rtGameObject:GetInstanceId()
    print( string.format("OnObjectReplicated = %s, owner = %s, instnaceId = %d",
rtGameObject:GetName(), owner:GetName(), instanceId) )

end
```

See Also

None

RtGameObject.GetName

Brief

Queries the object's name.

Definition

```
string RtGameObject.GetName( RtGameObject obj )
```

Arguments

obj - The RT Game Object to query.

Return Values

Object's name.

Description

Queries the object's name. The name is the string specified in the 'name' attribute when defining this object in the network_definition_file file given in the Object Editor.

Examples

```
function OnObjectReplicated( rtGameObject )
    print( string.format("OnObjectReplicated = %s", rtGameObject:GetName()) )
end
```

See Also

None

RtGameObject.GetOwnerMemberId

Brief

Queries the object's owner.

Definition

```
number RtGameObject.GetOwnerMemberId( RtGameObject obj )
```

Arguments

obj - The RT Game Object to query.

Return Values

Retrieves the memberId of the object's owner.

Description

Queries the object's owner.

Examples

```
function OnObjectReplicated( rtGameObject )

    ownerId = rtGameObject:GetOwnerMemberId()
    print( string.format("OnObjectReplicated = %s, ownerId = %d", rtGameObject:GetName(), ownerId) )

end
```

See Also

None

RtGameObject.GetOwnershipType

Brief

Queries the object's ownership type.

Definition

```
number RtGameObject.GetOwnershipType( RtGameObject obj )
```

Arguments

obj - The RT Game Object to query.

Return Values

Object's ownership type (see enums below)

Description

Queries the object's ownership type, as defined by the 'type' attribute of the object's defninition in the network defnition xml.

- RtObjOwnership.Personal
- RtObjOwnership.Session

Examples

```
function OnObjectReplicated( rtGameObject )

    local owner = rtGameObject:GetOwner()

    if rtGameObject:GetOwnershipType() == RtObjOwnership.Personal then

        print( string.format("OnObjectReplicated = %s, owner = %s, Personal obj",
rtGameObject:GetName(), owner:GetName()) )

    else

        print( string.format("OnObjectReplicated = %s, owner = %s, Session obj",
rtGameObject:GetName(), owner:GetName()) )

    end

end
```

See Also

None

SaveData

SaveData.Create

Brief

Creates a new save data request object.

Definition

```
SaveData.Create()
```

Arguments

None.

Return Values

A new instance of a save data request object.

Description

Creates a new save data request object.

Examples

```
saveDataRequest = SaveData.Create()
```

See Also

SaveData.GetState

Brief

Returns the current status of the request.

Definition

```
SaveData.GetState( SaveData request )
```

Arguments

request - The request to check the state for

Return Values

One of the following states:
0 - Idle. The request has not started or has completed without any problems
1 - The request is in progress
2 - The request is in progress (but is in a new internal state, scripts should treat this state no differently from 1)
-1 - The request has failed
-2 - The was no data to load (not technically an error as this will happen when the script is run for the first time)

Description

Returns the current status of the request. Scripts can monitor the status to see when a request completes and if it was successful or not.

Examples

```
local state = SaveData.GetState( saveDataRequest )
```

See Also

SaveData.StartLoading

Brief

Submits a save data load request.

Definition

```
number SaveData.StartLoading( SaveData request, string dataId, number type, MemoryContainer container
)
```

Arguments

request - The request to submitsaveId - The data identifier. Must match the id used for saving.type - Reserved, must be 0container - The container to load the data into

Return Values

0 on success or one of the following errors:
-1 - Invalid params.
-2 - This request object is already busy.
-3 - This is a duplicate of another request in progress (only affects save request).
-4 - The container is too large to save. Save sizes are limited to 4kb.
-5 - Loading and saving has been disable due to an earlier problem.

Description

Submits a save data load request. Multiple load requests for the same data can be submitted providing that there are no in progress save requests. Please contact your Playstation(R)Home Regional Account Manager to request a save data key. Note: save data keys are currently limited and use of the save data API will initially be on a restricted basis.

Examples

```
local result = SaveData.StartLoading( saveDataRequest, "XXXXXX", 0, container )
```

See Also

SaveData.StartSaving

Brief

Submits a save data save request.

Definition

```
number SaveData.StartSaving( SaveData request, string dataId, number type, MemoryContainer container,
number size = 0 )
```

Arguments

request - The request to submitsaveId - The data identifier. Must match the id used for loadingtype - Reserved, must be 0container - The container to save the data fromsize - The number of bytes to be saved. If not specified, the UsedSize of the memory container will be used if set. Otherwise, the total memory container size will be used.

Return Values

0 on success or one of the following errors:
-1 - Invalid params.
-2 - This request object is already busy.
-3 - This is a duplicate of another request in progress (only affects save request).
-4 - The container is too large to save. Save sizes are limited to 4kb.
-5 - Loading and saving has been disable due to an earlier problem.

Description

Submits a save data save request. Only a single save request for a single dataId can be in progress at a time. If there are any other loads or saves in progress, the script will need to wait for those to complete first. Please contact your Playstation(R)Home Regional Account Manager to request a save data key. Note: save data keys are currently limited and use of the save data API will initially be on a restricted basis.

Examples

```
local result = SaveData.StartSaving( saveDataRequest, "XXXXXX", 0, container )
```

See Also

Scene

Scene.FindEntity

Brief

Finds an entity that was created from the scene

Definition

```
Entity Scene.FindEntity( string name )
```

Arguments

name - The name of the entity as set in the HDK's Scene Editor application.

Return Values

The entity if it is found and nil otherwise.

Description

Finds an entity that was created from the scene, (i.e. one added to the world as part of the .scene file)

Examples

```
anotherDancingMonkey = Scene.FindEntity( "DancingMonkey342" )
Entity.SetAnim( anotherDancingMonkey, "monkey_excited" )
```

See Also

Scene.FindScreen

Brief

Finds a screen in the active scene.

Definition

```
Screen Scene.FindScreen( string name )
```

Arguments

name - The name of the screen as set in the HDK's Scene Editor application.

Return Values

The screen matching the specified name or nil if the screen is not found.

Description

Finds a screen in the active scene.

Examples

```
screen = Scene.FindScreen( "MyVideoScreen" )
if screen ~= nil then
    Screen.SetContent( screen, "http://www.example.com/myvideo.mp4" )
end
```

See Also

- Screen.SetContent

Scene.FindTriggerVolume

Brief

Finds a trigger volume that was created from the scene

Definition

```
TriggerVolume Scene.FindTriggerVolume( string name )
```

Arguments

name - The name of the trigger volume as set in the HDK's Scene Editor application.

Return Values

The trigger volume if it is found and nil otherwise.

Description

Finds a trigger volume that was created from the scene, (i.e. one added to the world as part of the .scene file)

Examples

```
trigVol = Scene.FindTriggerVolume( "sceneTriggerVolume1" )
TriggerVolume.AddCallback( trigVol, MySceneTriggerVolumeCallback )
```

See Also

- TriggerVolume.AddCallback

Scene.GetGraphicsEngineParam

Brief

Get the value of a parameter from the graphics engine.

Definition

```
Scene.GetGraphicsEngineParam(paramName)
```

Arguments

paramName - a valid graphics engine param name. Valid values are:
GraphicsEngine.ShadowEnable
GraphicsEngine.ShadowQuality
GraphicsEngine.SunlightColor
GraphicsEngine.SunlightDir
GraphicsEngine.ColorBalance
GraphicsEngine.Saturation
GraphicsEngine.Contrast
GraphicsEngine.ShadowPlaneDistance
GraphicsEngine.ShadowPlaneMultiplier
GraphicsEngine.ShadowPlaneCount
GraphicsEngine.ShadowDarkness
GraphicsEngine.AlphaSortEnable
GraphicsEngine.LightProbeTint
GraphicsEngine.GlowType
GraphicsEngine.GlowColor
GraphicsEngine.GlowBlendMode

Return Values

The current value of the graphics engine parameter in question.

Description

For details of the types returned by this function, see Scene.SetGraphicsEngineParam.

Examples

```
print(Scene.GetGraphicsEngineParam(GraphicsEngine.SunlightColor)) -- output is "[1.64, 1.52, 1.08, 0]"
print(Scene.GetGraphicsEngineParam(GraphicsEngine.ShadowEnable)) -- output is "true"
```

See Also

- Scene.SetGraphicsEngineParam

Scene.GetInstanceId

Brief

Returns a unique id for the scene instance the user is currently in

Definition

```
string Scene.GetInstanceId()
```

Arguments

None.

Return Values

string - The unique scene instance id

Description

Returns a unique id for the scene instance the user is currently in

Examples

```
sceneInstaceId = Scene.GetInstanceId()
```

See Also

None

Scene.GetLaunchParam

Brief

Returns one of the text parameters that may have been passed to PS Home when launched.

Definition

```
string Scene.GetLaunchParam( number index = 1 )
```

Arguments

index - Index of the parameter to fetch (1 - 16).

Return Values

The indexed string that PS Home was launched with, or "none" if none.

Description

This function allows your script to query optional values passed to Home through a configuration file either set up by SCE regional teams or generated from a game launched by Home. This allows you to gamelaunch from Home into your external title, then when the user selects to return to PS Home they are directed to different spaces depending on whether they won or lost (for instance). Note that these values are only valid in the first scene the user enters after returning to Home. See external documentation for more details on setting up this system.

Examples

```
param = Scene.GetLaunchParam( 1 )
if ( param ~= "none" ) then
    if ( param == "winner" ) then       -- custom string passed to PS Home by title
        print("You won!")
    else
        print("You lost.")
    end
end
```

See Also

None

Scene.GetLayerCollisionDefault

Brief

Query the default enabledness of collision detection between collision of the specified layers.

Definition

```
boolean Scene.GetLayerCollisionDefault( enum layerA, enum layerB )
```

Arguments

layerA - The first layer. layerB - The second layer.

Return Values

true if collision detection is enabled by default, false otherwise.

Description

Query the default enabledness of collision detection between collision of the specified layers.

Examples

```
print( Scene.GetLayerCollisionDefault( CollisionLayer.User1, CollisionLayer.FurniturePlacement ) )
```

See Also

- Scene.SetLayerCollisionDefault

Scene.GetListenerObjects

Brief

Returns an array of scene objects that are listening for data.

Definition

```
Object[] Scene.GetListenerObjects( number startIndex = 1, number maxCount = -1 )
```

Arguments

startIndex - The position to start from. Previous indices are not added to the results. maxCount - The maximum size the results array can be.

Return Values

A table of Object instances.

Description

Returns an array of scene objects that are listening for data.

Examples

```
sceneListenerObjects = Scene.GetListenerObjects()
t = { letters={ "a", "b", "c" }, type="letter_message" }
for i, sceneListenerObject in ipairs( sceneListenerObjects ) do
    result1, result2, result3 = Object.SendListenerData( sceneListenerObject, t )
end
```

See Also

None

Scene.GetMaxSaveDataSize

Brief

Indicates the maximum size in bytes of the available save data storage area.

Definition

```
number Scene.GetMaxSaveDataSize()
```

Arguments

None

Return Values

The maximum size in bytes of the save data storage area.

Description

This function will return the maximum size in bytes of the available save data storage area. This number will never decrease. The number is valid only if the scene has save data access.

Examples

```
local access = Scene.GetSaveDataPermission()
if access == SaveDataPermission.ReadWrite and not Scene.IsSaveDataBusy() then
    mc = MemoryContainer.Create( Scene.GetMaxSaveDataSize() )
    -- modify container here
    Scene.SaveData( mc )
end
```

See Also

None

Scene.GetSaveDataPermission

Brief

Obtain the level of save data access available to the current user in this scene.

Definition

```
SaveDataPermission Scene.GetSaveDataPermission()
```

Arguments

None

Return Values

A permission level (see details).

Description

Some scenes (currently personal spaces and clubhouses) provide the ability for a script to save data. Public scenes will not provide this ability. Only the owner or club leader is currently able to read or write the save data, so the ReadOnly permission level below is not yet available. This function will return the save data access level for the current user in the current scene, from the following values:
SaveDataPermission.None
SaveDataPermission.ReadOnly
SaveDataPermission.ReadWrite
For convenience these values can also be compared in the expected order (see example). Note that the save data system may also temporarily be busy, even if the user has permission. This can be checked with the IsSaveDataBusy function.

Examples

```
local access = Scene.GetSaveDataPermission()
if (access > SaveDataPermission.None and not Scene.IsSaveDataBusy()) then
    Scene.LoadData(memContainer)
end
```

See Also

- Scene.IsSaveDataBusy

Scene.GetStartupInstanceParam

Brief

Returns the Instance parameter that was passed to the scene when it was initialised

Definition

```
number Scene.GetStartupInstanceParam()
```

Arguments

None.

Return Values

number - The Instance parameter that the scene was started with

Description

When the user is relocated to a new scene (either by using a relocate region placed in another scene, or by the LocalPlayer.Relocate function), an optional 'instance' parameter can be passed. The instance parameter is used to create different instances of the scene, and can then be queried by the destination scene's scene script to change the behaviour in this new instance of the scene. For example, you might have a 'lobby' scene and a 'game-room' scene connected to it. When the user chooses to move from the lobby to the game-room, the lobby's script might decide (for instance), to send all European users to game-room instance 1, and all US users to game-room instance 2. European users would all appear in one set of instances of the game-room, US users would all appear in another set of instances of the game-room. The game-room script could then use this function to determine if the current instance is a European or US one and change behaviour appropriately.

Examples

```
param = Scene.GetStartupInstanceParam()
```

See Also

Scene.GetStartupValueParam

Brief

Returns the value parameter that was passed to the scene when it was initialised

Definition

```
number Scene.GetStartupValueParam()
```

Arguments

None.

Return Values

number - The value parameter that the scene was started with

Description

The value parameter is usually derived from relocation regions in the scene editor. e.g. You might have one scene that has 2 doorways leading to copies of the same scene When placing the relocation region for the doorways, you add the name of the scene the doorways lead to and (optionally) a value and text parameter. Hence, the first doorway could lead to space B with value parameter 1, and the second doorway could lead to space B with value parameter 2. The scene script in space B could then use this function to, for instance, set different screen content depending on which doorway the user came through. Note that users entering the same scene will see other users in that scene, even though they might have entered with a different value parameter To create separate instances of the scene controlled by parameter, see Scene.GetStartupInstanceParam()

Examples

```
param = Scene.GetStartupValueParam()
```

See Also

Scene.GetVirtualObject

Brief

Returns the current scene's virtual Object and nil if not present.

Definition

```
Object Scene.GetVirtualObject()
```

Arguments

None

Return Values

the current scene's virtual Object and nil if not present.

Description

Returns the current scene's virtual Object and nil if not present.

Examples

```
sceneObject = Scene.GetVirtualObject()
t = { letters={ "a", "b", "c" }, type="letter_message" }
results = Object.SendListenerData( sceneObject, t )
```

See Also

None

Scene.IsLoading

Brief

Indicates if current scene is currently loading.

Definition

```
boolean Scene.IsLoading()
```

Arguments

None

Return Values

True if a scene is currently loading or false if fully loaded.

Description

A scene is considered loaded once the scene itself and all objects marked as load before spawn are loaded, and the local player is spawned. Object and scene scripts will start to run before a scene is fully loaded, this is by design as it allows background loading to continue where necessary. If a script wishes to disable certain functionality while the scene is still loading, this function can be used to identify this state. Note that scene objects not marked as load before spawn will continue to load after this function returns true; to detect this state use Scene.IsLoadingObjects.

Examples

```
if (Scene.IsLoading()) then
    g_canStartGame = false          -- disallow playing a game until scene is loaded
end
```

See Also

- Scene.IsLoadingObjects

Scene.IsLoadingObjects

Brief

Indicates if current scene is currently loading scene objects.

Definition

```
boolean Scene.IsLoadingObjects()
```

Arguments

None

Return Values

True if a scene is currently loading scene objects or false if all scene objects are loaded.

Description

A scene is considered loaded once the scene itself and all objects marked as load before spawn are loaded, and the local player is spawned. Scene objects not marked as load before spawn will however continue to be downloaded after this point. This function will return true while there are still scene objects downloading and loading in the scene, and will only return false when all scene objects are loaded and active. If you only wish to detect when the scene and local player and all load before spawn objects are loaded, use Scene.IsLoading.

Examples

```
if (g_isLoading and not Scene.IsLoadingObjects()) then
    g_isLoading = false
    SendListenerDataToAllObjects()      -- user function
end
```

See Also

- Scene.IsLoading

Scene.IsSaveDataBusy

Brief

Returns true if the save data system is currently busy indicating a save data request would fail.

Definition

```
boolean Scene.IsSaveDataBusy()
```

Arguments

None

Return Values

true if the save data system is busy and false otherwise.

Description

This function is only useful if the user has save data access (see GetSaveDataPermission). If this function returns true, any attempt to use the save data system will cause a Lua error.

Examples

```
local access = Scene.GetSaveDataPermission()
if access == SaveDataPermission.ReadWrite and not Scene.IsSaveDataBusy() then
    mc = MemoryContainer.Create( Scene.GetMaxSaveDataSize() )
    -- modify container here
    Scene.SaveData( mc )
end
```

See Also

- Scene.GetSaveDataPermission

Scene.LoadData

Brief

Copies the scene's save data (if available) into the supplied memory container.

Definition

```
void Scene.LoadData( MemoryContainer memContainer, int size = Scene.GetMaxSaveDataSize( ), int offset
= 0 )
```

Arguments

memContainer - The memory container to load into.size - Number of bytes to obtain.offset - Offset in bytes into the memory container at which to store the data.

Return Values

None.

Description

All scenes which allow saving (clubhouses and personal space), have an allocated save data area which allow embedded objects and scenescripts to save data about the scene. Note that nothing should rely on specific data being present in the save data (as the user may delete their Home save data at any time) and thus actives should be able to recognise and handle the case where no save data is present. When a scene is accessed for the first time, all of its save data is set to zero.

Examples

```
local access = Scene.GetSaveDataPermission()
if access > SaveDataPermission.None and not Scene.IsSaveDataBusy() then
    local memContainer = MemoryContainer.Create( Scene.GetMaxSaveDataSize( ) )
    Scene.LoadData(memContainer)
    if (memContainer:GetUint16(0) == 48879) then        -- 0xBEEF
        print('Save data is valid: found the beef')
        g_mySize = memContainer:GetFloat32(4)
    end
end
```

See Also

- Scene.SaveData
- Scene.GetSaveDataPermission
- Scene.IsSaveDataBusy

Scene.LockPadControls

Brief

Locks the pad controls.

Definition

```
Scene.LockPadControls(boolean val)
```

Arguments

val - true or false to lock or unlock the pad controls respectively.

Return Values

Description

Locks the input pad controls, bar the SELECT button (for the Safe Screen). The developer is advised to use this with care, and to interact with the user appropriately during the time the controls are locked. If it's not obvious why the controls are locked, the user should be informed.

Examples

```
Scene.LockPadControls(true)
```

See Also

None

Scene.OverrideAvatarCloseUpCamAspect

Brief

Set the aspect for the standard close-up camera that follows the Avatar.

Definition

```
Scene.OverrideAvatarCloseUpCamAspect(boolean isOverride, number value)
```

Arguments

isOverride - whether or not to override the default setting.value - the value to set.

Return Values

None.

Description

Examples

```
Scene.OverrideAvatarCloseUpCamAspect(true, 16.0 / 9.0)
```

```
Scene.OverrideAvatarCloseUpCamAspect(false)
```

See Also

- Scene.OverrideAvatarCloseUpCamFov
- Scene.OverrideAvatarCloseUpCamZNear
- Scene.OverrideAvatarStandardCamAspect
- Scene.OverrideAvatarStandardCamZNear
- Scene.OverrideAvatarStandardCamZFar

Scene.OverrideAvatarCloseUpCamFov

Brief

Set the Field Of View for the standard close-up camera that follows the Avatar.

Definition

```
Scene.OverrideAvatarCloseUpCamFov(boolean isOverride, number value)
```

Arguments

isOverride - whether or not to override the default setting.value - the value to set.

Return Values

None.

Description

Examples

```
Scene.OverrideAvatarCloseUpCamFov(true, 60)
```

```
Scene.OverrideAvatarCloseUpCamFov(false)
```

See Also

- Scene.OverrideAvatarCloseUpCamAspect
- Scene.OverrideAvatarCloseUpCamZNear
- Scene.OverrideAvatarStandardCamAspect
- Scene.OverrideAvatarStandardCamZNear
- Scene.OverrideAvatarStandardCamZFar

Scene.OverrideAvatarCloseUpCamZFar

Brief

Set the far clipping for the standard close-up camera that follows the Avatar.

Definition

```
Scene.OverrideAvatarCloseUpCamZFar(boolean isOverride, number value)
```

Arguments

isOverride - whether or not to override the default setting.value - the value to set.

Return Values

None.

Description

Examples

```
Scene.OverrideAvatarCloseUpCamZFar(true, 1000)
```

```
Scene.OverrideAvatarCloseUpCamZFar(false)
```

See Also

- Scene.OverrideAvatarCloseUpCamFov
- Scene.OverrideAvatarCloseUpCamAspect

- Scene.OverrideAvatarStandardCamAspect
- Scene.OverrideAvatarStandardCamZNear
- Scene.OverrideAvatarStandardCamZFar

Scene.OverrideAvatarCloseUpCamZNear

Brief

Set the near clipping for the standard close-up camera that follows the Avatar.

Definition

```
Scene.OverrideAvatarCloseUpCamZNear(boolean isOverride, number value)
```

Arguments

isOverride - whether or not to override the default setting.value - the value to set.

Return Values

None.

Description

Examples

```
Scene.OverrideAvatarCloseUpCamZNear(true, 0.1)
```

```
Scene.OverrideAvatarCloseUpCamZNear(false)
```

See Also

- Scene.OverrideAvatarCloseUpCamFov
- Scene.OverrideAvatarCloseUpCamAspect
- Scene.OverrideAvatarStandardCamAspect
- Scene.OverrideAvatarStandardCamZNear
- Scene.OverrideAvatarStandardCamZFar

Scene.OverrideAvatarStandardCamAspect

Brief

Set the aspect for the standard close-up camera that follows the Avatar.

Definition

```
Scene.OverrideAvatarStandardCamAspect(boolean isOverride, number value)
```

Arguments

isOverride - whether or not to override the default setting.value - the value to set.

Return Values

None.

Description

Examples

```
Scene.OverrideAvatarStandardCamAspect(true, 16.0 / 9.0)
```

```
Scene.OverrideAvatarStandardCamAspect(false)
```

See Also

- Scene.OverrideAvatarStandardCamFov
- Scene.OverrideAvatarStandardCamZNear
- Scene.OverrideAvatarCloseUpCamAspect
- Scene.OverrideAvatarCloseUpCamZNear
- Scene.OverrideAvatarCloseUpCamZFar

Scene.OverrideAvatarStandardCamFov

Brief

Set the Field Of View for the standard close-up camera that follows the Avatar.

Definition

```
Scene.OverrideAvatarStandardCamFov(boolean isOverride, number value)
```

Arguments

isOverride - whether or not to override the default setting.value - the value to set.

Return Values

None.

Description

Examples

```
Scene.OverrideAvatarStandardCamFov(true, 60)
```

```
Scene.OverrideAvatarStandardCamFov(false)
```

See Also

- Scene.OverrideAvatarStandardCamAspect
- Scene.OverrideAvatarStandardCamZNear
- Scene.OverrideAvatarCloseUpCamAspect
- Scene.OverrideAvatarCloseUpCamZNear
- Scene.OverrideAvatarCloseUpCamZFar

Scene.OverrideAvatarStandardCamZFar

Brief

Set the far clipping for the standard close-up camera that follows the Avatar.

Definition

```
Scene.OverrideAvatarStandardCamZFar(boolean isOverride, number value)
```

Arguments

isOverride - whether or not to override the default setting.value - the value to set.

Return Values

None.

Description

Examples

```
Scene.OverrideAvatarStandardCamZFar(true, 1000)
```

```
Scene.OverrideAvatarStandardCamZFar(false)
```

See Also

- Scene.OverrideAvatarStandardCamFov
- Scene.OverrideAvatarStandardCamAspect
- Scene.OverrideAvatarCloseUpCamAspect
- Scene.OverrideAvatarCloseUpCamZNear
- Scene.OverrideAvatarCloseUpCamZFar

Scene.OverrideAvatarStandardCamZNear

Brief

Set the Far clipping for the standard close-up camera that follows the Avatar.

Definition

```
Scene.OverrideAvatarStandardCamZNear(boolean isOverride, number value)
```

Arguments

isOverride - whether or not to override the default setting.value - the value to set.

Return Values

None.

Description

Examples

```
Scene.OverrideAvatarStandardCamZNear(true, 0.1)
```

```
Scene.OverrideAvatarStandardCamZNear(false)
```

See Also

- Scene.OverrideAvatarStandardCamFov
- Scene.OverrideAvatarStandardCamAspect
- Scene.OverrideAvatarCloseUpCamAspect
- Scene.OverrideAvatarCloseUpCamZNear
- Scene.OverrideAvatarCloseUpCamZFar

Scene.ResetAvatarCamOverrides

Brief

Resets all previously set Avatar Camera overrides.

Definition

```
Scene.ResetAvatarCamOverrides()
```

Arguments

None

Return Values

None.

Description

Examples

```
Scene.ResetAvatarCamOverrides()
```

See Also

- Scene.OverrideAvatarStandardCamFov
- Scene.OverrideAvatarStandardCamAspect
- Scene.OverrideAvatarStandardCamZNear
- Scene.OverrideAvatarCloseUpCamAspect
- Scene.OverrideAvatarCloseUpCamZNear
- Scene.OverrideAvatarCloseUpCamZFar

Scene.SaveData

Brief

Copies the data from the supplied memory container into the scene's save data area and requests it to save.

Definition

```
Scene.SaveData( MemoryContainer memContainer, number size = Scene.GetMaxSaveDataSize( ), int offset =
0 )
```

Arguments

memContainer - The memory container to read data from.size - Number of bytes to obtain.offset - Offset in bytes into the memory container at which to start reading the data to save.

Return Values

None.

Description

All scenes which allow saving (clubhouses and personal spaces), have an allocated save data area which allow embedded objects and scene scripts to save data about the scene. Note that nothing should rely on specific data being present in the save data (as the user may delete their Home save data at any time) and thus a script should be able to recognise and handle the case where no or invalid save data is present. When a scene is accessed for the first time, all of its save data is set to zero. Note: If size is not specified and your memory container is bigger than the maximum save data size an error will occur to catch accidental data truncation.

Examples

```
local access = Scene.GetSaveDataPermission()
if (access == SaveDataPermission.ReadWrite and not Scene.IsSaveDataBusy()) then
    local memContainer = MemoryContainer.Create( 8 )
    memContainer:SetUint16(0, 48879)        -- 0xBEEF, tags our data as valid
    memContainer:SetFloat32(4, mySize)      -- save our current size so we can keep growing next time
    Scene.SaveData(memContainer, 8)
end
```

See Also

- Scene.LoadData
- Scene.GetSaveDataPermission
- Scene.IsSaveDataBusy

Scene.SetGraphicsEngineParam

Brief

Set the value of a parameter in the graphics engine.

Definition

```
Scene.SetGraphicsEngineParam(paramName, value)
```

Arguments

paramName - a valid graphics engine param name. Valid values are:
GraphicsEngine.ShadowEnable boolean - enable or disable shadows, defaults to true)
GraphiceEngine.ShadowQuality set the sun shadow quality. Should be one of:
ShadowQuality.Low
ShadowQuality.Medium
ShadowQuality.High
GraphicsEngine.SunlightColor Vector4 - set the color/brightness of the sun, w component is ignored. Each channel range is 0=>3
GraphicsEngine.SunlightDir Vector4 - set the direction of the sun. The value does not need to be normalized
GraphicsEngine.ColorBalance Vector4 - set the post-process color balance, components should be in the range -1=>1, defaults to (0,0,0)
GraphicsEngine.Saturation scalar - set the post-process saturation, should be in the range -2=>2, defaults to 1
GraphicsEngine.Contrast scalar - set the post-process contrast, should be in the range -1=>1, defaults to 0
GraphicsEngine.ShadowPlaneDistance scalar - distance to the first shadow plane in metres, defaults to 2.0

GraphicsEngine.ShadowPlaneMultiplier scalar - value to muliply each shadow plane distance by to get the distance to the next plane, defaults to 3.0 and must be greater than 1.0
GraphicsEngine.ShadowPlaneCount scalar - number of shadow planes to render, should be in the range 0=>4, defaults to 4
GraphicsEngine.ShadowDarkness scalar - set the darkness of indoor shadows (has no effect in sunlit scenes), should be in the range 0=>1, defaults to 0.2
GraphicsEngine.AlphaSortEnable boolean - enable or disable sorting of semitransparent meshes. for scenes exported with HDK1.40 onwards, this defaults to true.
GraphicsEngine.LightProbeTint Vector4 - set a global tint colour for the light probes. defaults to (1,1,1)
GraphicsEngine.GlowType set the type of glow. Should be one of:
GlowType.Disabled
GlowType.Small
GlowType.Default
GlowType.Large
GlowType.ExtraLarge
GlowType.Horizontal
GlowType.Vertical
GraphicsEngine.GlowColor Vector4 - set the tint colour for the glow
GraphicsEngine.GlowBlendMode set the blend type for the glow, should be one of:
GlowBlend.Default
GlowBlend.Add - add the glow buffer to the framebuffer
GlowBlend.Squared - square the glow buffer and add it to the framebuffer
GraphicsEngine.EnvironmentMap Set the current environment map for dynamic objects.
The argument should be the name of a texture resource, or the resource itselfvalue - the value to set.

Return Values

None.

Description

Examples

```
Scene.SetGraphicsEngineParam(GraphicsEngine.SunlightColor, Vector4.Create(2,2,0)) -- set the sun color to bright yellow
Scene.SetGraphicsEngineParam(GraphicsEngine.ShadowEnable, false) -- disable shadows in this scene
```

See Also

None

Scene.SetLayerCollisionDefault

Brief

Set the default enabledness of collision detection between collision of the specified layers.

Definition

```
Scene.SetLayerCollisionDefault( enum layerA, enum layerB, boolean enabled )
```

Arguments

layerA - The first layer; must be a "user layer".layerB - The second layer.enabled - true to enable collision detection by default, false otherwise.

Return Values

None.

Description

Set the default enabledness of collision detection between collision of the specified layers.

Examples

```
Scene.SetLayerCollisionDefault( CollisionLayer.User1, CollisionLayer.FurniturePlacement, false )
```

See Also

- Scene.GetLayerCollisionDefault

Scene.SetLightProbeData

Brief

Set the light probe data for the scene

Definition

```
Scene.SetLightProbeData(string resourcename, number slotIndex, bool updateSun=false)
Scene.SetLightProbeData(Resource resource, number slotIndex, bool updateSun=false)
```

Arguments

resource - The probe data file resource, or resource name, or nil to clear the slot
This resource can not be released whilst it is in use on a slot. Set the slot to another resource or nil before attempting to release it.slotIndex - The light probe volume index to set. There are 9 slots (numbered 0-8) available.updateSun - If this is true, update the sun parameters based on the values from the given probe data

Return Values

none

Description

Sets the light probe volume at the given index to the given data. If this volume overlaps an existing volume, then the one with the lowest slot number will have precedence. Note, there is no way to query the light probe slots once assigned.

Examples

```
-- load in new scene geometry --
entity = Entity.Create()
entity:SetModel( "my_new_scene_geometry.mdl" )
-- load corresponding lighting data --
Scene.SetLightProbeData( "mylighting.probe", 1 )
-- clear old lighting --
Scene.SetLightProbeData( nil, 0 )
```

See Also

- Scene.SetLightProbeTransform

Scene.SetLightProbeTransform

Brief

Set the transform for the light probe volume in the given slot

Definition

```
Scene.SetLightProbeTransform(Matrix44 transform, number slotIndex)
```

Arguments

transform - The transform to apply to the light probe volume in the given slotslotIndex - The light probe volume index to set. There are 9 slots (numbered 0-8) available.

Return Values

none

Description

Sets the transform for the light probe volume at the given index

Examples

```
-- load in new scene geometry --
entity = Entity.Create()
entity:SetModel( "my_new_scene_geometry.mdl" )
-- load corresponding lighting data --
Scene.SetLightProbeData( "mylighting.probe", 1 )
-- reposition model and probes --
pos = Vector4.Create(10,10,0)
entity:SetPosition( pos )
Scene.SetLightProbeTransform( Matrix44.Create( Quaternion.Create(), pos ), 1 )
```

See Also

- Scene.SetLightProbeData

SceneDownload

SceneDownload.CancelDownload

Brief

Cancels the specified scripted scene download.

Definition

```
boolean SceneDownload.CancelDownload( string sceneName )
```

Arguments

None

Return Values

True if download was cancelled, false otherwise.

Description

Only scene downloads triggered by script can be cancelled with this function.

Examples

```
if (SceneDownload.CancelDownload("myscene")) then
    print("Download cancelled")
end
```

See Also

- SceneDownload.StartDownload

SceneDownload.GetCurrentDownload

Brief

Returns the scripted scene download currently in progress (or nil if none is active).

Definition

```
string SceneDownload.GetCurrentDownload()
```

Arguments

None

Return Values

The current scene download, or nil if none is activate.

Description

Returns the scene name of the current scene download if one is active.

Examples

```
local scene = SceneDownload.GetCurrentDownload()
if (scene) then
    print("Downloading " .. scene .. "...")
end
```

See Also

- SceneDownload.StartDownload

SceneDownload.GetDownloadProgress

Brief

Gets the progress value for the download of the requested scene.

Definition

```
number SceneDownload.GetDownloadProgress( string sceneName )
```

Arguments

None

Return Values

A value between 0 (no data received) and 1 (all data received) indicating the progress of the download.

Description

Retrieves the progress proportion of the download of a specified scene.

Examples

```
local progress = SceneDownload.GetDownloadProgress("myscene")
RenderProgressBar(progress)     -- user function
```

See Also

- SceneDownload.StartDownload

SceneDownload.GetDownloadSize

Brief

Gets the size of the download for the specified scene name, in bytes.

Definition

```
number SceneDownload.GetDownloadSize( string sceneName )
```

Arguments

None

Return Values

The size in bytes of the specified scene download.

Description

Gets the size of the download for a specified scene. The download itself is not triggered.

Examples

```
local size = SceneDownload.GetDownloadSize("myscene")
```

See Also

None

SceneDownload.QuerySceneCacheStatus

Brief

Queries whether the specified scene is currently cached and up-to-date.

Definition

```
SceneDownload.QuerySceneCacheStatus( string sceneName )
```

Arguments

None

Return Values

One of the following result values:
CacheCheckResult.Downloading - scene is downloading
CacheCheckResult.InCache - scene is cached
CacheCheckResult.NeedsDownload - scene is out of date and needs re-downloading
CacheCheckResult.AccessRestricted - scene has failed age restriction or failed to download

Description

Queries whether the specified scene is currently cached and up-to-date.

Examples

```
if (SceneDownload.QuerySceneCacheStatus("myscene") == CacheCheckResult.InCache) then
    print("Scene is cached!")
end
```

See Also

- SceneDownload.StartDownload

SceneDownload.StartDownload

Brief

Starts a background download of the specified scene (if none is active).

Definition

```
boolean SceneDownload.StartDownload( string sceneName )
```

Arguments

None

Return Values

True if the scene download was started successfully, false otherwise.

Description

Triggers a background download of the named scene. Any download that is in progress will be cancelled.

Examples

```
if (SceneDownload.StartDownload("myscene")) then
    print("Scene download triggered!")
end
```

See Also

- SceneDownload.GetCurrentDownload

SceneInfo

SceneInfo.GetSceneId

Brief

Returns the sceneId of the current scene.

Definition

```
string SceneInfo.GetSceneId()
```

Arguments

None

Return Values

The sceneId.

Description

Returns the sceneId of the current scene.

Examples

```
print("Current scene ID:", SceneInfo.GetSceneId())
```

See Also

- SceneInfo.GetSceneName
- SceneInfo.GetUniqueInstanceId

SceneInfo.GetSceneName

Brief

Returns the localized name of the current scene.

Definition

```
string SceneInfo.GetSceneName()
```

Arguments

None

Return Values

The localized name of the scene.

Description

Returns the localized name of the current scene.

Examples

```
print("Entering scene:", SceneInfo.GetSceneName())
```

See Also

- SceneInfo.GetSceneId
- SceneInfo.GetUniqueInstanceId

SceneInfo.GetUniqueInstanceId

Brief

Return a unique instance ID for the current scene instance.

Definition

```
string SceneInfo.GetUniqueInstanceId()
```

Arguments

None

Return Values

The instance ID.

Description

This function will construct and return a unique ID string for the current scene instance. This value will be unique for different instances of the same scene. No assumptions should be made about the format of the instance ID.

Examples

```
print("Scene instance ID:", SceneInfo.GetUniqueInstanceId())
```

See Also

- SceneInfo.GetSceneId
- SceneInfo.GetSceneName

SceneTransition

SceneTransition.ExitScript

Brief

Terminates the SceneTransition object.

Definition

```
SceneTransition.ExitScript()
```

Arguments

None

Return Values

None.

Description

This function must be called when the scene transition object has finished its execution to release the object and hand control over to the user.

Examples

```
if (SceneTransition.IsSceneChangeComplete()) then
    SceneTransition.ExitScript()
end
```

See Also

None

SceneTransition.GetDestinationScene

Brief

Returns the name of the destination scene for a scene transition.

Definition

```
string SceneTransition.GetDestinationScene()
```

Arguments

None

Return Values

The scene name.

Description

Returns the name of the destination scene for a scene transition.

Examples

```
local scene = SceneTransition.GetDestinationScene()
```

See Also

- SceneTransition.GetOriginScene

SceneTransition.GetObject

Brief

Returns the SceneTransition object (if it is running).

Definition

```
Object SceneTransition.GetObject()
```

Arguments

None

Return Values

SceneTransition object if it is running, or nil otherwise.

Description

Returns the SceneTransition object if a relocation process is occurring.

Examples

```
local object = SceneTransition.GetObject()
```

See Also

None

SceneTransition.GetOriginScene

Brief

Returns the name of the scene that started the scene transition.

Definition

```
string SceneTransition.GetOriginScene()
```

Arguments

None

Return Values

The scene name.

Description

Returns the name of the scene that started the scene transition.

Examples

```
local scene = SceneTransition.GetOriginScene()
```

See Also

- SceneTransition.GetDestinationScene

SceneTransition.GetSceneChangeProgress

Brief

Returns a value indicating the progress of the current relocation from 0.0 to 1.0.

Definition

```
number SceneTransition.GetSceneChangeProgress()
```

Arguments

None

Return Values

A value between 0 and 1 indicating the transition progress.

Description

Returns a value indicating the progress of the current relocation from 0.0 to 1.0.

Examples

```
local progress = SceneTransition.GetSceneChangeProgress()
RenderProgressBar(progress)         -- user function
```

See Also

None

SceneTransition.IsSceneChangeComplete

Brief

Indicates when the scene change is fully complete and the new scene is ready to interact with.

Definition

```
boolean SceneTransition.IsSceneChangeComplete()
```

Arguments

None

Return Values

True if the scene change is complete, or false if still in progress.

Description

If the scene change has completed and the new scene is ready to interact with, this function will return true.

Examples

```
if (SceneTransition.IsSceneChangeComplete()) then
    print("Scene ready!")
end
```

See Also

None

SceneTransition.StartSceneChange

Brief

Begins the scene relocation process.

Definition

```
boolean SceneTransition.StartSceneChange()
```

Arguments

None

Return Values

True if scene change started successfully, false otherwise.

Description

Activates the scene relocation, removing the user from the current scene and starting the process of loading and moving to the new one.

Examples

```
if (SceneTransition.StartSceneChange()) then
    print("Scene change started!")
end
```

See Also

None

Screen

Screen.AreSceneSoundsMuted

Brief

Return if scene sound is muted while in fullscreen

Definition

```
boolean Screen.AreSceneSoundsMuted(Screen screen)
```

Arguments

screen

Return Values

true if scene sound is muted while in fullscreen

Description

Return if scene sound is muted while in fullscreen

Examples

```
local screen = Scene.FindScreen( "video1" )
screen.AreSceneSoundsMuted()
```

See Also

- Scene.FindScreen

Screen.Create

Brief

Creates a new screen.

Definition

```
Screen Screen.Create( Vector4 topLeft, Vector4 bottomRight, Vector4 bottomLeft, string type, string
source = nil )
```

Arguments

topLeft - World space coordinate of top left hand corner of the screen.bottomRight - World space coordinate of bottom right hand corner of the screen.bottomLeft - World space coordinate of bottom left hand corner of the screen.type - String denoting the type of screen to be created. Can be one of the following:
"HSML"
"IMAGE"
"LUA"
"MEDIARSS"
"VIDEO"
source - Source for the data to be displayed on the screen (a URL or file path).

Return Values

A new Screen instance.

Description

Creates a new screen. The screen type can be one of the following: "HSML" - for displaying 'Home Screen Markup Language' pages. "IMAGE" - for displaying a DDS, JPG or PNG file. "LUA" - for a screen that can have a Renderer object attached. These will be protected by default. "MEDIARSS" - for displaying media syndicated via an RSS feed. "VIDEO" - for displaying video content.

Examples

```
-- Create a screen for submitting graphical primitives from Lua
spriteScreen = Screen.Create( topLeft, bottomRight, bottomLeft, "LUA" )
renderer = Renderer.Create()
Renderer.SetTarget(renderer, spriteScreen)
Renderer.DrawSprite(renderer, sprite)

-- Create a video screen
url = "http://www.example.com/video.mp4"
vidScreen = Screen.Create( topLeft, bottomRight, bottomLeft, "VIDEO", url )
```

See Also

- Renderer.SetTarget
- Screen.SetProtected

Screen.EnableBackground

Brief

Enables the default background for IMAGE screen types.

Definition

```
Screen.EnableBackground( Screen screen, boolean disable )
```

Arguments

screen - The screen to toggle the background for.disable - true to enable the default background, false to disable it (default is enabled).

Return Values

None.

Description

Enables or disables the the default black background for IMAGE screen types.

Examples

```
local screen = Scene.FindScreen( "Poster" )
screen.EnableBackground( false )
```

See Also

- Scene.FindScreen

Screen.GetCenter

Brief

Returns a vector holding the central position of the screen.

Definition

```
Vector4 Screen.GetCenter( Screen screen )
Screen.GetCenter( Screen screen, Vector4 outPos )
```

Arguments

screen - The screen to query.outPos - (out) The vector to receive the position.

Return Values

A vector holding the central position of the screen.

Description

Returns a vector holding the central position of the screen. This vector is in world space.

Examples

```
screen = Scene.FindScreen( "MyVideoScreen" )
center = Screen.GetCenter( screen )
print( "The screen center point is " .. center )
```

See Also

- ArcadeGame.GetScreen
- Scene.FindScreen
- Screen.Create

Screen.GetContentCount

Brief

Returns the number of different content attached to the screen.

Definition

```
number Screen.GetContentCount( Screen screen )
```

Arguments

screen - The screen to query.

Return Values

The number of different content attached to the screen.

Description

Returns the number of different content attached to the screen.

Examples

```
-- Step to the next content source if it has been downloaded
screen = Scene.FindScreen( "MyVideoScreen" )

contentCount = Screen.GetContentCount( screen )
currentIndex = Screen.GetCurrentContentIndex( screen )

nextIndex = ( currentIndex + 1 ) % contentCount
if Screen.IsContentReady( screen, nextIndex ) then
    Screen.SetCurrentContentIndex( screen, nextIndex )
end
```

See Also

- Scene.FindScreen
- Screen.GetCurrentContentIndex
- Screen.IsContentReady
- Screen.SetCurrentContentIndex

Screen.GetCurrentContentIndex

Brief

Returns the current source index of the screen.

Definition

```
number Screen.GetCurrentContentIndex( Screen screen )
```

Arguments

screen - The screen to query.

Return Values

The current content index of the screen.

Description

Returns the current source index of the screen.

Examples

```
-- Step to the next content source if it has been downloaded
screen = Scene.FindScreen( "MyVideoScreen" )

contentCount = Screen.GetContentCount( screen )
currentIndex = Screen.GetCurrentContentIndex( screen )

nextIndex = ( currentIndex + 1 ) % contentCount
if Screen.IsContentReady( screen, nextIndex ) then
    Screen.SetCurrentContentIndex( screen, nextIndex )
end
```

See Also

- Scene.FindScreen
- Screen.GetContentCount
- Screen.IsContentReady

- Screen.SetCurrentContentIndex

Screen.GetCurrentTime

Brief

Returns the current time of the screen in seconds.

Definition

```
number Screen.GetCurrentTime( Screen screen )
```

Arguments

screen - The screen to query.

Return Values

The current time of the screen in seconds.

Description

This is currently only supported on video screens and media-rss screens currently playing video content. Note: If the content is not ready it will return 0. Use Screen.IsContentReady to check the status.

Examples

```
screen = Scene.FindScreen( "MyVideoScreen" )
if ( Screen.GetCurrentTime( screen ) >= 20 ) then
    state = STATE_CHEER
end
```

See Also

- Scene.FindScreen
- Screen.GetTotalTime

Screen.GetMatrix

Brief

Returns the world matrix for the screen.

Definition

```
Screen.GetMatrix( Screen screen, Matrix44 matrix )
```

Arguments

screen - The screen to query.matrix - A matrix to set.

Return Values

None.

Description

Returns the world matrix for the screen. An identity matrix is returned if the screen is invalid. Note that the world matrix will scale the resolution of the screen to the world-space size of the screen, and thus will typically include a large scale component, so do not assume that any row of the matrix will be unit length. If only the screen normal is required, use

Screen.GetNormal.

Examples

```
local matrix = Matrix44.Create()
screen = Scene.FindScreen( "MyVideoScreen" )
Screen.GetMatrix( screen, matrix )
print( 'Screen translation is ', matrix.GetRow( 4 ) )
```

See Also

- ArcadeGame.GetScreen
- Scene.FindScreen
- Screen.Create
- Screen.GetNormal

Screen.GetNormal

Brief

Returns the normal for the screen.

Definition

```
Screen.GetNormal( Screen screen, Vector4 vector )
```

Arguments

screen - The screen to query.vector - A vector to set to the normal.

Return Values

None.

Description

Returns the normal for the screen, that is, the vector pointing in the facing direction of the screen. The normal will be unit length.

Examples

```
local vector = Vector4.Create()
screen = Scene.FindScreen( "MyVideoScreen" )
Screen.GetNormal( screen, vector )
print( "The screen normal is ", vector )
```

See Also

- ArcadeGame.GetScreen
- Scene.FindScreen
- Screen.Create

Screen.GetSize

Brief

Returns a vector holding the screen size.

Definition

```
Vector4 Screen.GetSize( Screen screen )
Screen.GetSize( Screen scree, Vector4 outSize )
```

Arguments

screen - The screen to query.outSize - (out) The vector to receive the size.

Return Values

A vector holding the size of the screen's content in pixels. Only X and Y vector components are valid.

Description

Returns a vector holding the size of the screen's content in pixels. The X component of the returned vector holds the screen's content width and the Y component holds the screen's content height.

Examples

```
screen = Scene.FindScreen( "MyVideoScreen" )
contentSize = Screen.GetSize( screen )
print( "The screen content size is " .. contentSize )
```

See Also

- ArcadeGame.GetScreen
- Scene.FindScreen
- Screen.Create

Screen.GetTotalTime

Brief

Returns the current time of the screen in seconds.

Definition

```
number Screen.GetTotalTime( Screen screen )
```

Arguments

screen - The screen to query.

Return Values

The total time of the screen in seconds.

Description

This returns the total duration of the currently playing video. This is currently only supported on video screens and media-rss screens playing video content. Note: If the content is not ready it will return 0. Use Screen.IsContentReady to check the status.

Examples

```
screen = Scene.FindScreen( "MyVideoScreen" )
if ( Screen.GetCurrentTime( screen ) / Screen.GetTotalTime( screen ) >= 0.5 ) then
    state = STATE_HALF_WAY
end
```

See Also

- Scene.FindScreen

Screen.GetTriggerPosition

Brief

Retrieves the trigger position for the screen.

Definition

```
Screen.GetTriggerPosition( Screen screen, Vector4 position )
```

Arguments

screen - Screen object to query.position - The current trigger position.

Return Values

None.

Description

Retrieves the current trigger position, which determines where the user can target this screen. If a custom trigger position has not been set this will return the same as Screen.GetCenter(). This will set a position of 0 if the screen is not valid.

Examples

```
local position = Vector4.Create()
screen = Scene.FindScreen( "MyVideoScreen" )
Screen.GetTriggerPosition( screen, position )
```

See Also

- Screen.SetTriggerPosition
- Screen.GetTriggerRadius
- Screen.Create
- Scene.FindScreen

Screen.GetTriggerRadius

Brief

Retrieves the trigger radius for the screen.

Definition

```
number Screen.GetTriggerRadius( Screen object )
```

Arguments

object - Screen object to query.

Return Values

range - The current trigger radius in meters.

Description

Retrieves the current trigger radius, which determines from how far away the user can target this screen. Note that 0 is considered as an infinite target radius.

Examples

```
range = Screen.GetTriggerRadius( object )
```

See Also

- Screen.SetTriggerRadius
- Screen.Create
- Scene.FindScreen

Screen.GetVolume

Brief

Returns the volume of the screen.

Definition

```
Number Screen.GetVolume( Screen screen )
```

Arguments

screen - The screen to query.

Return Values

A number from 0.0 to 1.0 indicating the current volume of the screen.

Description

Returns the current volume of the screen. A value of 0.0 will be returned if the screen is invalid.

Examples

```
screen = Scene.FindScreen( "MyVideoScreen" )
volume = Screen.Getvolume( screen )
print( "The screen volume size is " .. volume )
```

See Also

- Scene.FindScreen
- Screen.Create
- Screen.SetVolume

Screen.IsContentPlaying

Brief

Returns true if the content is playing and false otherwise.

Definition

```
boolean Screen.IsContentPlaying( Screen screen )
```

Arguments

screen - The screen to query.

Return Values

Returns true if the content is playing and false otherwise.

Description

This is currently only supported on video screens and media-rss screens currently playing video content. Note: If the content is not ready it will return false. Use Screen.IsContentReady to check the status.

Examples

```
if Screen.IsContentPlaying( screen ) == true then
    Screen.Pause( screen )
end
```

See Also

- Screen.Pause
- Screen.Resume

Screen.IsContentReady

Brief

Reports whether the content is ready to be displayed.

Definition

```
boolean Screen.IsContentReady( Screen screen, number contentIndex )
```

Arguments

screen - The screen to query.contentIndex - The source index to query. Ranges from 1 to the content count.

Return Values

True if the content is ready and false otherwise.

Description

Reports whether the content is ready to be displayed. This is only supported for video screens and media-rss screens while playing video content.

Examples

```
-- Step to the next content source if it has been downloaded
screen = Scene.FindScreen( "MyVideoScreen" )

contentCount = Screen.GetContentCount( screen )
currentIndex = Screen.GetCurrentContentIndex( screen )

nextIndex = ( currentIndex + 1 ) % contentCount
if Screen.IsContentReady( screen, nextIndex ) then
    Screen.SetCurrentContentIndex( screen, nextIndex )
end
```

See Also

- Scene.FindScreen
- Screen.GetContentCount
- Screen.GetCurrentContentIndex
- Screen.SetCurrentContentIndex

Screen.IsFogEnabled

Brief

Retrieve the fog enable state for a screen.

Definition

```
boolean Screen.IsFogEnabled(Screen screen)
```

Arguments

screen - the screen to access

Return Values

True if the screen is affected by the scene fog or false otherwise.

Description

Retrieve the fog enable state for a screen. If this value is true the screen will be affected by the fog settings in the scene.

Examples

```
fogEnabled = myScreen:IsFogEnabled();
```

See Also

- Screen.SetFogEnabled

Screen.IsHDFullScreenModeSet

Brief

Return if a video is in real full screen mode

Definition

```
boolean Screen.IsHDFullScreenModeSet(Screen screen)
```

Arguments

screen

Return Values

true if real full screen, false otherwise

Description

Return if a video is in real full screen mode

Examples

```
local screen = Scene.FindScreen( "video1" )
screen.IsHDFullScreenModeSet()
```

See Also

- Scene.FindScreen

Screen.IsInFullScreenMode

Brief

Returns true if the screen is in full screen mode and false otherwise.

Definition

```
boolean Screen.IsInFullScreenMode(Screen screen)
```

Arguments

Screen - The screen to query.

Return Values

True if the screen is in full screen mode and false otherwise.

Description

Examples

```
screen = Scene.FindScreen("MyVideoScreen")
isFullScreenMode = Screen.IsInFullScreenMode(screen)
```

See Also

None

Screen.IsProtected

Brief

Retrieve the protection state for a screen.

Definition

```
boolean Screen.IsProtected(Screen screen)
```

Arguments

screen - the screen to access

Return Values

True if the screen is protected or false otherwise.

Description

Retrieve the protection state for a screen. If this value is true the screen will not be rendered into any still or video captures performed by PlayStation(R) Home.

Examples

```
if (not myVidScreen:IsProtected()) then
    myVidScreen:SetProtected(true)
end
```

See Also

- Screen.SetProtected

Screen.IsVisible

Brief

Returns whether a screen is visible or not.

Definition

```
Boolean Screen.IsVisible( Screen object )
```

Arguments

object - Screen object to affect.isVisible - true or false (defaults to true).

Return Values

Whether the screen is visible or not.

Description

This function returns whether a screen has been tagged as visible or invisible.

Examples

```
isVisible = Screen.IsVisible( object )
```

See Also

- Screen.SetVisible

Screen.Pause

Brief

Pauses the currently playing screen.

Definition

```
Screen.Pause( Screen screen )
```

Arguments

screen - The screen to be paused.

Return Values

None.

Description

Pauses the currently playing screen.
Note: This is currently only supported for video, hsml and media-rss screens.

Examples

```
s = Scene.FindScreen( "MyVideoScreen" )
Screen.Pause( s )
```

See Also

- Screen.Resume

Screen.RestartStream

Brief

Restart the stream.

Definition

```
boolean Screen.RestartStream( Screen object )
```

Arguments

object - Screen object to restart.

Return Values

true if succeeded

Description

Restart the current stream.

Examples

```
boolReturn = Screen.RestartStream( object )
```

See Also

None

Screen.Resume

Brief

Resumes the currently paused screen.

Definition

```
Screen.Resume( Screen screen )
```

Arguments

screen - The screen to be resumed.

Return Values

None.

Description

Resumes the currently paused screen.
Note: This is currently only supported for video, hsml and media-rss screens.

Examples

```
s = Scene.FindScreen( "MyVideoScreen" )
Screen.Resume( s )
```

See Also

- Screen.Pause

Screen.SetChannel

Brief

Sets the content channel for the screen.

Definition

```
Screen.SetChannel( Screen screen, string channelName )
```

Arguments

screen - The screen to be updated.channelName - the name of the content channel to display on the screen.

Return Values

None.

Description

Home stores a list of media content 'channels' - e.g. one channel might be "SonyAdChannel1" - with each channel providing an indirection to the screen type and content. The content for each channel can be updated independently of the main package and scene data.

Examples

```
Screen.SetChannel( myScreen, "SonyAdChannel1" )
```

See Also

- Screen.SetContent
- Screen.SetType

Screen.SetContent

Brief

Sets a content source or sources on the screen.

Definition

```
Screen.SetContent( Screen screen, string[] contentList )
Screen.SetContent( Screen screen, string content, number contentIndex = 1 )
```

Arguments

screen - The screen to update.contentList - An array of strings specifying multiple content sources.content - A string for the content source.contentIndex - The index to set the content to. Ranges from 1 to the number of content sources. Defaults to 1.

Return Values

None.

Description

Sets a content source or sources on the screen. The source string will either be a URL or a path relative to the HDK's build root directory. Note that paths relative to the HDK's build root directory can only be used for testing/development purposes. Scripts uploaded to a content server must reference content served over HTTP. Also note that HomeDeveloper.self must be launched in online mode in order for online sources to be specified.

Examples

```
-- Example1: Testing a video served from the host PC's hard drive for debug
localSource = "localvideo/testvideo.mp4"
Debug.SetContent( myScreen1, localSource )

-- Example 2: Play back a video served from an HTTP server
onlineSource = "http://mycontentserver.com/onlinevideo.mp4"
Screen.SetContent( myScreen2, onlineSource )
```

See Also

- Scene.FindScreen

Screen.SetCurrentContentIndex

Brief

Sets the screen to the specified content index.

Definition

```
Screen.SetCurrentContentIndex( Screen screen, number contentIndex )
```

Arguments

screen - The screen to update.contentIndex - The index of the content to set the screen to. Ranges from 1 to the content count.

Return Values

None.

Description

Sets the screen to the specified content index.

Examples

```
-- Step to the next content source if it has been downloaded
screen = Scene.FindScreen( "MyVideoScreen" )

contentCount = Screen.GetContentCount( screen )
currentIndex = Screen.GetCurrentContentIndex( screen )

nextIndex = ( currentIndex + 1 ) % contentCount
if Screen.IsContentReady( screen, nextIndex ) then
    Screen.SetCurrentContentIndex( screen, nextIndex )
end
```

See Also

- Scene.FindScreen
- Screen.GetContentCount
- Screen.GetCurrentContentIndex
- Screen.IsContentReady

Screen.SetCurrentTime

Brief

Set the current time for the currently selected content.

Definition

```
Screen.SetCurrentTime(Screen screen, number time)
```

Arguments

screen - the screen to modifytime - the time (in seconds) to set

Return Values

Description

Set the current playback time for the currently selected content. For a video this will set the playback time to "time" seconds from the beginning of the video for example. This is currently only supported on video screens and media-rss screens playing video content. Note that setting the time will necessarily cause the video to rebuffer, so there will be a delay before the movie changes to the new time. This is worth bearing in mind if you are attempting to synchronize video playback between clients.

Examples

```
s = Screen.Create( 'video', 'http://www.example.com/video.mp4' )
s:Pause()
s:SetCurrentContentIndex( 3 )
s:SetCurrentTime( timeOffset )      -- timeOffset calculated to synchronise this user with the others
s:Resume()
```

See Also

- Screen.Pause
- Screen.Resume

- Screen.GetCurrentTime

Screen.SetData

Brief

Sets the data for an HSML screen.

Definition

```
Screen.SetData( Screen screen, <string|MemoryContainer|Resource> data, string contentRoot )
```

Arguments

screen - The screen to update. data - Different types depending on the type of the screen. contentRoot - Root path to which all content referenced by the HSML is relative to.

Return Values

Description

Currently only supported for HSML screens. For HSML screens the data can be either a string, a MemoryContainer containing BasicGenx output, or an XML Resource.

Examples

```
-- Example 1: Setting HSML content from a string
hsmlScreen = Scene.FindScreen( "HsmlScreen" )
data = '<XML><PAGE><RECT X="0" Y="0" W="1280" H="720" col="#FF00C0" /><TEXT X="430" Y="20"
col="#FFFFFF" size="4">'
data = data .. Person.GetName( LocalPlayer.GetPerson() )
data = data .. '</TEXT></PAGE></XML>'
Screen.SetData( hsmlScreen, data )

-- Example 2: Setting HSML content from a MemoryContainer
mem = MemoryContainer.Create( 4096 )
writer = BasicGenx.Create( mem )
BasicGenx.StartDocument( writer )
-- Use BasicGenx to write HSML XML data here...
BasicGenx.EndDocument( writer )
Screen.SetData( hsmlScreen, mem )

-- Example 3: Setting HSML content from a downloaded XML resource
url = "http://www.example.com/hsmlscreen.xml"
hsmlResource = Resource.Request( url, "xml" )
while Resource.IsLoading( hsmlResource ) do
    coroutine.yield()
end
if Resource.IsLoaded( hsmlResource ) then
    Screen.SetData( hsmlScreen, hsmlResource )
end
```

See Also

- BasicGenx.Create
- MemoryContainer.Create
- Resource.Request
- Scene.FindScreen

Screen.SetFogEnabled

Brief

Set whether this screen should be affected by fog (defaults to false)

Definition

```
Screen.SetFogEnabled(Screen screen, boolean enable)
```

Arguments

screen - the screen to modifyenable - true to enable fogging or false otherwise

Return Values

None.

Description

Set whether this screen should be affected by fog. If this flag is set to true, the screen will be affected by the fog in the scene, and will therefore need to be positioned in front of a solid backing in order to provide the z-value (screens themselves do not write to the depth buffer).

Examples

```
myScreen:SetFogEnabled( true )
```

See Also

- Screen.IsFogEnabled

Screen.SetHDFullScreenMode

Brief

Set a video full screen mode to real full screen.

Definition

```
Screen.SetHDFullScreenMode( Screen screen, boolean value )
```

Arguments

screen - The screen to toggle the background for.value - true to enable real full screen, false to disable it (default is disabled).

Return Values

None.

Description

Set a video full screen mode to real full screen.

Examples

```
local screen = Scene.FindScreen( "video1" )
screen.SetHDFullScreenMode( true )
```

See Also

- Scene.FindScreen

Screen.SetHighPriority

Brief

Set a video screen to high priority.

Definition

```
Screen.SetHighPriority( Screen screen, boolean value )
```

Arguments

screen - The screen to toggle the background for.value - true to enable high priority, false to disable it (default is disabled).

Return Values

None.

Description

Set a video screen to high priority.

Examples

```
local screen = Scene.FindScreen( "video1" )
screen.SetHighPriority( false )
```

See Also

- Scene.FindScreen

Screen.SetMuteSceneSounds

Brief

Set the sounds in the scene to mute if video in full screen.

Definition

```
Screen.SetMuteSceneSounds( Screen screen, boolean value )
```

Arguments

screenvalue - true to disable scene sound while in full screen, false to disable it (default is disabled).

Return Values

None.

Description

Set the sounds in the scene to mute if video in full screen.

Examples

```
local screen = Scene.FindScreen( "video1" )
screen.SetMuteSceneSounds( true )
```

See Also

- Scene.FindScreen

Screen.SetProtected

Brief

Enable or disable capture protection on a screen.

Definition

```
Screen.SetProtected(Screen screen, boolean protected)
```

Arguments

screen - the screen to modifyprotected - true to protect the screen or false otherwise

Return Values

None.

Description

Enable or disable capture of a screen's contents. If this flag is set to true, any capture process (screenshot or video) will cause the screen's content to not be rendered, thus protecting any copyrighted or sensitive material. Setting to false will unprotect the screen's content. "LUA" screens will default to protected, other screen types will default to unprotected.

Examples

```
-- Create a screen for submitting graphical primitives from Lua
vidScreen = Screen.Create( topLeft, bottomRight, bottomLeft, "VIDEO",
"http://www.example.com/CopyrightedVideo.mp4" )
vidScreen:SetProtected( true )
```

See Also

- Screen.IsProtected

Screen.SetTargetable

Brief

Sets the screen to be targetable or not.

Definition

```
Screen.SetTargetable( Screen screen, boolean value )
```

Arguments

screen - The screen to update.value - If true then the screen is targetable and if false it is not targetable.

Return Values

None.

Description

Sets the screen to be targetable or not. Screens created in Lua default to have targeting disabled. Screens that appear in the HDK's Scene Editor application default to have targeting enabled.

Examples

```
-- Example 1: This type of screen defaults to targetable
screen1 = Scene.FindScreen( "MyVideoScreen" )
Screen.SetTargetable( screen1, false )

-- Example 2: This type of screen defaults to not targetable
screen2 = Screen.Create( topLeft, bottomRight, bottomLeft, "LUA" )
Screen.SetTargetable( screen2, true )
```

See Also

- Scene.FindScreen
- Screen.Create

Screen.SetTriggerPosition

Brief

Changes the trigger position for the screen.

Definition

```
Screen.SetTriggerPosition( Screen object, Vector4 position = nil )
```

Arguments

object - Screen object to modify.position - The new trigger position. If nil is specified the trigger position will reset to default.

Return Values

None.

Description

Modifies the default trigger position to affect where the user can target this screen. Specifying nil will reset the trigger position to be based on Screen.GetCenter().

Examples

```
local position = Vector4.Create( 10, 0, 10 )
local screen = Scene.FindScreen( "MyVideoScreen" )
Screen.SetTriggerPosition( screen, position ) -- set a custom trigger position
Screen.SetTriggerPosition( screen, nil ) -- reset the position to default
```

See Also

- Screen.GetTriggerPosition
- Screen.SetTriggerRadius
- Screen.Create
- Scene.FindScreen

Screen.SetTriggerRadius

Brief

Changes the trigger radius for the screen.

Definition

```
Screen.SetTriggerRadius( Screen object, number range )
```

Arguments

object - Screen object to modify.range - The new trigger radius in meters. Must be positive or 0, 0 is considered infinite.

Return Values

None.

Description

Modifies the default trigger radius to affect from how far away the user can target this screen. Specifying 0 will make the screen targetable from any distance.

Examples

```
Screen.SetTriggerRadius( screen, 10 )
```

See Also

- Screen.GetTriggerRadius
- Screen.Create
- Scene.FindScreen

Screen.SetType

Brief

Sets the content type for the screen.

Definition

```
Screen.SetType( Screen screen, string type )
```

Arguments

screen - The screen to update.type - The new screen type. Can be one of the following:
"HSML"
"IMAGES"
"LUA"
"MEDIARSS"
"VIDEO"

Return Values

None.

Description

Sets the content type for the screen.

Examples

```
Screen.SetType( myScreen, "VIDEO" )
Screen.SetContent( myScreen, "http://www.example.com/myvideo.mp4" )
```

See Also

- Screen.SetContent

Screen.SetVisible

Brief

Set screen to be visible or invisible.

Definition

```
Screen.SetVisible( Screen object, boolean isVisible )
```

Arguments

object - Screen object to affect. isVisible - true to make the screen visible and false otherwise.

Return Values

None.

Description

Set screen to be visible or invisible. This will also mute videos and remove the ability for the screen to be targetable. Will error when applied to a geometry screen.

Examples

```
Screen.SetVisible( object, false )
```

See Also

- Screen.IsVisible

Screen.SetVolume

Brief

Sets the volume of the screen.

Definition

```
Screen.SetVolume( Screen screen, number volume )
```

Arguments

screen - The screen to query. volume - The volume to set, 0.0 to 1.0.

Return Values

None.

Description

Sets the volume of the screen. A value outside the 0.0 to 1.0 range will be capped.

Examples

```
screen = Scene.FindScreen( "MyVideoScreen" )
Screen.SetVolume( screen, 0.5 )
```

See Also

- Scene.FindScreen
- Screen.Create
- Screen.GetVolume

Screenshot

Screenshot.Capture

Brief

Start a screen capture.

Definition

```
boolean Screenshot.Capture(Screenshot shot,
                           boolean showScreens = false,
                           boolean showOsd = true,
                           boolean showLabels = false,
                           boolean showChat = false,
                           boolean showWatermark = true)
```

Arguments

shot - the screenshot object to useshowScreens - enable screen rendering for screenshotshowOsd - enable OSD rendering for screenshotshowLabels - show player labelsshowChat - show player chat bubblesshowWatermark - show PlayStation Home watermark on screenshot

Return Values

true if capture is successfully started, or false if an existing screenshot is still capturing

Description

Start a screen capture with the specified options. Note that a true return value does not indicate that the capture has finished, the script must wait until Screen.IsInProgress returns false. The following arguments must not be changed from their default values without permission from a PlayStation(R)Home Regional Manager: * showScreens * showLabels * showChat Any screens marked as protected will not be shown regardless of the value of showScreens. Any changes to screen, OSD or label rendering is temporary and only active for the duration of the screenshot. The developer should consult their regional SCE manager about the legal ramifications of enabling screen rendering in screenshots. Note that Lua screens are excepted from this: even if showScreens is false, Lua-created screens will be still be shown, as they are often required to maintain the look of scene.

Examples

```
shot = Screenshot.Create()
capturing = shot:Capture(true, true, false, false, true)    -- everything on except labels and chat.
if (not capturing) then
    print('Capture failed')
end
...
function OnUpdate()
    if (capturing and not shot:IsInProgress() and shot:Succeeded()) then
        MediaLibrary.ExportImage( shot, "Home Screenshot" )
    end
end
```

See Also

- Screenshot.Create
- Screenshot.IsInProgress
- Screenshot.Succeeded
- Screen.IsProtected

Screenshot.Create

Brief

Create a new screenshot object.

Definition

```
Screenshot Screenshot.Create()
```

Arguments

None

Return Values

A new screenshot object, or nil on failure

Description

Creates a new screenshot object, used to save an image of the current frame buffer contents. Only a limited number of screenshot objects may exist at once, if this becomes exhausted, nil is returned so you can either fail or keep retrying every frame until another script releases a Screenshot handle.

Examples

```
shot = Screenshot.Create()
```

See Also

- Screenshot.Capture
- Screenshot.IsInProgress
- Screenshot.Succeeded
- MediaLibrary.ExportImage

Screenshot.IsInProgress

Brief

Query whether a screen capture is still completing.

Definition

```
boolean Screenshot.IsInProgress(Screenshot shot)
```

Arguments

shot - the screenshot object to check

Return Values

true if still capturing, false if finished (either successfully or with an error)

Description

Query whether a screen capture initiated with Screenshot.Capture() has completed. If this returns false an error may also have occurred; check the value of Screenshot.Succeeded() to determine this. Once this returns false it is then safe to set the screenshot object to nil if it is no longer required.

Examples

```
if (shot:IsInProgress()) then
    print('Capturing...')
elseif (shot:Succeeded()) then
    -- do something with screenshot
end
```

See Also

Screenshot.Succeeded

Brief

Query whether a screen capture completed successfully.

Definition

```
boolean Screenshot.Succeeded(Screenshot shot)
```

Arguments

shot - the screenshot object to check

Return Values

true if capture completed successfully or false if still in progress or encountered an error

Description

Query whether a screen capture initiated with Screenshot.Capture() completed successfully. If the capture is still in progress false will be returned.

Examples

```
if (shot:Succeeded()) then
    MediaLibrary.ExportImage(shot, "Home Screenshot")
end
```

See Also

None

Sound

Sound.Create

Brief

Creates an empty sound handle.

Definition

```
Sound Sound.Create()
```

Arguments

None

Return Values

A fresh sound handle.

Description

The returned sound handle can be used repeatedly, removing the allocation cost associated with playing sounds from the SoundBank library.

Examples

```
mySound = Sound.Create()
myBank = SoundBank.Find("mySounds")
mySound:Play3d(myBank, "bang", LocalPlayer.GetPosition(), 1, 0, 5, 8, 0, 0, 0, 0)
```

See Also

- SoundBank.Play3d
- SoundBank.Play2d

Sound.GetVolume

Brief

Retrieves the volume of a sound.

Definition

```
number Sound.GetVolume( Sound sound )
```

Arguments

sound - The sound object.

Return Values

A floating point value between 0.0 (silent) and 1.0 (full volume).

Description

Retrieves the volume of a sound.

Examples

```
volume = Sound.GetVolume( mySound )
```

See Also

- Sound.SetVolume

Sound.IsPaused

Brief

Reports if the sound is paused or not.

Definition

```
boolean Sound.IsPaused( Sound sound )
```

Arguments

sound - The sound object.

Return Values

Returns true if the sound is paused and false otherwise.

Description

Returns true if the sound is paused and false otherwise.

Examples

```
if ( Sound.IsPaused( mySound ) ) then
    Sound.Resume( mySound )
end
```

See Also

- Sound.Pause
- Sound.Resume

Sound.IsPlaying

Brief

Reports if the sound is playing or not.

Definition

```
boolean Sound.IsPlaying( Sound sound )
```

Arguments

sound - The sound object.

Return Values

Returns true if the sound is playing and false otherwise.

Description

Returns true if the sound is playing and false otherwise.

Examples

```
if ( Sound.IsPlaying( mySound ) ) then
    Sound.Stop( mySound )
end
```

See Also

- Sound.Stop

Sound.IsValid

Brief

Can be used to check if a sound is valid.

Definition

```
boolean Sound.IsValid()
```

Arguments

None

Return Values

true if the sound is valid, false otherwise.

Description

Can be used to check if a sound is valid. A sound can become invalid if its SoundBank has been released.

Examples

```
if Sound.IsValid( mySound ) == false then
    mySound = nil
end
```

See Also

- Resource.Release

Sound.Pause

Brief

Pauses the sound.

Definition

```
Sound.Pause( Sound sound )
```

Arguments

sound - The sound object.

Return Values

None

Description

Pauses the sound.

Examples

```
if ( Sound.IsPlaying( mySound ) ) then
    Sound.Pause( mySound )
end
```

See Also

- Sound.IsPlaying

Sound.Play2d

Brief

Plays a named sound from a sound bank.

Definition

```
Sound.Play2d( Sound sound, SoundBank soundbank, string name, number volume = 1.0, number pan = 0.0,
number pitch = 0.0, number r0 = 0.0, number r1 = 0.0, number r2 = 0.0, number r3 = 0.0  )
```

Arguments

soundbank - The soundbank to play the sound fromname - The name of the sample within the SoundBank as set within SCREAM.volume - Value between 0.0 and 1.0 for controlling the volume of the sound.pan - Value between -1.0 and 1.0 for controlling the stereo output of the sound.pitch - This parameter will be implemented in the next release of the HDK.r0 - Passed to your scream sound's register 0.r1 - Passed to your scream sound's register 1.r2 - Passed to your scream sound's register 2.r3 - Passed to your scream sound's register 3.

Return Values

None

Description

Plays a named sound in the sound bank.

Examples

```
soundbank = SoundBank.Find( "game_bank" )
dingSound = Sound.Create()
dingSound:Play2d( soundbank, "ding" )
```

See Also

- Sound.Play3d
- Sound.Create
- Sound.Play2d

Sound.Play3d

Brief

Plays a named 3D sound from a sound bank.

Definition

```
Sound.Play3d( Sound sound, SoundBank soundbank, string name, Vector4 position, number volume = 1.0,
number pitch = 0.0, number falloffStart = 0.0, number falloffEnd = 1.0, number r1 = 0.0, number r2 =
0.0, number r3 = 0.0, number r4 = 0.0)
```

Arguments

soundbank - The soundbank to play the sound fromname - The name of the sample within the SoundBank as set within SCREAM.position - 3D world space coordinate of the sound (w-component is ignored).volume - Value between 0.0 and 1.0 for controlling the volume of the sound.pitch - This parameter will be implemented in the next release of the HDK.falloffStart - Value greater than 0.0 for controlling the distance from the position parameter at which the sound's volume begins to linearly attenuate.falloffEnd - Value greater than or equal to falloffStart for controlling the distance from the position parameter at which the sound's volume falls to zero.r0 - Passed to your scream sound's register 0.r1 - Passed to your scream sound's register 1.r2 - Passed to your scream sound's register 2.r3 - Passed to your scream sound's register 3.

Return Values

None

Description

Plays a named 3D sound from the sound bank.

Examples

```
soundbank = SoundBank.Find( "game_bank" )
ballRollingSound = Sound.Create()
ballRollingSound:Play3d( soundbank, "roll", LocalPlayer.GetPosition(), 1.0, 0.0, 10.0, 20.0)
```

See Also

- Sound.Play2d
- Sound.Create
- Sound.Play3d

Sound.Resume

Brief

Resumes a paused sound.

Definition

```
Sound.Resume( Sound sound )
```

Arguments

sound - The sound object.

Return Values

None

Description

Resumes a paused sound.

Examples

```
if ( Sound.IsPaused( mySound ) ) then
    Sound.Resume( mySound )
end
```

See Also

- Sound.IsPaused

Sound.SetPosition

Brief

Set the 3D world position of the sound

Definition

```
Sound.SetPosition( Sound sound, Vector4 position )
```

Arguments

sound - The sound object.position - 3D world space coordinate of the sound (w-component is ignored)

Return Values

None

Description

Set the 3D world position of the sound

Examples

```
Sound.SetPosition( sound, Vector4.Create(0, 10, 0, 0) )
```

See Also

- SoundBank.Play3d

Sound.SetRegister

Brief

Sets the specified register's value to the specified value.

Definition

```
Sound.SetRegister( Sound sound, number register, number value )
```

Arguments

sound - The sound object.register - Which register (0 - 3) to setvalue - the value to set the specified register to

Return Values

Description

Sets the specified register's value to the specified value.

Examples

```
Sound.SetRegister( sound, 2, 10 )
```

See Also

- SoundBank.Play2d

- SoundBank.Play3d

Sound.SetVolume

Brief

Sets the volume of a sound.

Definition

```
Sound.SetVolume( Sound sound, number volume )
```

Arguments

sound - The sound object.volume - floating point value between 0.0 (silent) and 1.0 (full volume)

Return Values

None

Description

Sets the volume of a sound retrieved from a sound bank.

Examples

```
Sound.SetVolume( mySound, 0.5 )
```

See Also

- Sound.GetVolume

Sound.Stop

Brief

Stops the sound playing.

Definition

```
Sound.Stop( Sound sound )
```

Arguments

sound - The sound object.

Return Values

None

Description

Stops the sound playing.

Examples

```
if ( Sound.IsPlaying( mySound ) ) then
    Sound.Stop( mySound )
end
```

See Also

- Sound.IsPlaying

SoundBank

SoundBank.Find

Brief

Looks up and returns the sound bank with the specified name.

Definition

```
SoundBank SoundBank.Find( string name, ResourcePack rp = nil )
```

Arguments

name - The string name associated with the sound bank as specified in Object Editor.rp - The ResourcePack to search.

Return Values

The sound bank instance associated with the specified string name.

Description

Looks up and returns the sound bank with the specified name. Returns nil if the sound bank is not found.

Examples

```
soundbank = SoundBank.Find( "game_bank" )
```

See Also

- SoundBank.Play2d
- SoundBank.Play3d

SoundBank.Play2d

Brief

Plays a named sound in the sound bank.

Definition

```
Sound SoundBank.Play2d( SoundBank soundbank, string name, number volume = 1.0, number pan = 0.0,
number pitch = 0.0, number r0 = 0.0, number r1 = 0.0, number r2 = 0.0, number r3 = 0.0  )
```

Arguments

name - The name of the sample within the SoundBank as set within SCREAM.volume - Value between 0.0 and 1.0 for controlling the volume of the sound.pan - Value between -1.0 and 1.0 for controlling the stereo output of the sound.pitch - Values different to 0 represent an octave multiplier, 1 is 1 octave higher, -1 is 1 octave lower, 2 is 2 octaves higher, and so on.r0 - Passed to your scream sound's register 0.r1 - Passed to your scream sound's register 1.r2 - Passed to your scream sound's register 2.r3 - Passed to your scream sound's register 3.

Return Values

Returns a Sound to handle the instance of the new sound.

Description

Plays a named sound in the sound bank.

Examples

```
soundbank = SoundBank.Find( "game_bank" )
dingSound = SoundBank.Play2d( soundbank, "ding" )
```

See Also

- SoundBank.Find
- SoundBank.Play3d

SoundBank.Play3d

Brief

Plays a named 3D sound in the sound bank.

Definition

```
Sound SoundBank.Play3d( SoundBank soundbank, string name, Vector4 position, number volume = 1.0,
number pitch = 0.0, number falloffStart = 0.0, number falloffEnd = 1.0, number r1 = 0.0, number r2 =
0.0, number r3 = 0.0, number r4 = 0.0  )
```

Arguments

name - The name of the sample within the SoundBank as set within SCREAM.position - 3D world space coordinate of the sound (w-component is ignored).volume - Value between 0.0 and 1.0 for controlling the volume of the sound.pitch - Values different to 0 represent an octave multiplier, 1 is 1 octave higher, -1 is 1 octave lower, 2 is 2 octaves higher, and so on.falloffStart - Value greater than 0.0 for controlling the distance from the position parameter at which the sound's volume begins to linearly attenuate.falloffEnd - Value greater than or equal to falloffStart for controlling the distance from the position parameter at which the sound's volume falls to zero.r0 - Passed to your scream sound's register 0.r1 - Passed to your scream sound's register 1.r2 - Passed to your scream sound's register 2.r3 - Passed to your scream sound's register 3.

Return Values

Returns a Sound to handle the instance of the new sound.

Description

Plays a named 3D sound in the sound bank.

Examples

```
soundbank = SoundBank.Find( "game_bank" )
bombSound = SoundBank.Play3d( soundbank, "explosion", Vector4.Create(0, 10, 0, 0), 1.0, 0.0, 10.0,
20.0)
```

See Also

- SoundBank.Find
- SoundBank.Play2d

SoundStream

SoundStream.Find

Brief

Finds a named SoundStream in the script's list of resources.

Definition

```
SoundStream SoundStream.Find( string name )
```

Arguments

name - the string name associated with the SoundStream as specified in an object's resources.xml file.

Return Values

The sound stream instance associated with the specified string name.

Description

Finds a named SoundStream in the script's list of resources.

Examples

```
bgmusic = SoundStream.Find( "music" )
```

See Also

SoundStream.GetCurrentTime

Brief

Get the current playback time for the stream.

Definition

```
number SoundStream.GetCurrentTime( SoundStream stream )
```

Arguments

stream - The sound stream object.

Return Values

time - The current playback time of the stream in seconds.

Description

Get the current playback time for the stream.

Examples

```
bgmusic = SoundStream.Find( "music" )
time = SoundStream.GetCurrentTime()
```

See Also

- SoundStream.SetCurrentTime

SoundStream.GetTotalTime

Brief

Get the total playback time for the stream.

Definition

```
number SoundStream.GetTotalTime( SoundStream stream )
```

Arguments

stream - The sound stream object.

Return Values

time - The total time of the stream in seconds.

Description

Get the total playback time for the stream.

Examples

```
bgmusic = SoundStream.Find( "music" )
time = SoundStream.GetTotalTime()
```

See Also

- SoundStream.GetCurrentTime

SoundStream.GetVolume

Brief

Get the sound stream's volume.

Definition

```
number SoundStream.GetVolume( SoundStream stream )
```

Arguments

stream - The sound stream object.

Return Values

volume - value between 0.0 and 1.0

Description

Get the sound stream's volume.

Examples

```
bgmusic = SoundStream.Find( "music" )
volume = SoundStream.GetVolume( bgmusic )
```

See Also

- SoundStream.SetVolume

SoundStream.IsPaused

Brief

Indicates whether the stream is paused

Definition

```
SoundStream.IsPaused( SoundStream stream )
```

Arguments

stream - The sound stream object.

Return Values

true if the stream is paused

Description

Indicates whether the stream is paused

Examples

```
idLoaded = SoundStream.IsPaused()
```

See Also

SoundStream.IsPlaying

Brief

Indicates whether the stream is playing

Definition

```
SoundStream.IsPlaying( SoundStream stream )
```

Arguments

stream - The sound stream object.

Return Values

true if the stream is playing

Description

Indicates whether the stream is playing

Examples

```
idLoaded = SoundStream.IsPlaying()
```

See Also

SoundStream.Pause

Brief

Pause the sound stream.

Definition

```
SoundStream.Pause( SoundStream stream )
```

Arguments

stream - The sound stream object.

Return Values

None

Description

Pause the sound stream.

Examples

```
bgmusic = SoundStream.Find( "music" )
SoundStream.Pause( bgmusic )
```

See Also

- SoundStream.Play2d
- SoundStream.Play3d

SoundStream.Play2d

Brief

Starts playing the sound stream.

Definition

```
SoundStream.Play2d( SoundStream stream, boolean looping = false, number volume = 1.0 )
```

Arguments

stream - The sound stream object. looping - true to loop the stream, false to play it once. volume - value between 0.0 and 1.0 for controlling the volume of the stream.

Return Values

None

Description

Starts playing the sound stream. Volume of 1.0 is 100% volume.

Examples

```
bgmusic = SoundStream.Find( "music" )
SoundStream.Play2d( bgmusic, true, 1.0 )
```

See Also

- SoundStream.Stop

SoundStream.Play3d

Brief

Starts playing the sound stream at a 3D position in the world

Definition

```
SoundStream.Play3d( SoundStream stream, boolean looping = false, number volume = 1.0, Vector4 position
= (0,0,0,0), number falloffStart = 0, number falloffEnd = 1 )
```

Arguments

stream - The sound stream object. looping - true to loop the stream, false to play it once. volume - value between 0.0 and 1.0 for controlling the volume of the stream. position - 3D world space coordinate of the sound (w-component is ignored) falloffStart - Value greater than or equal to 0.0 for controlling the distance from the position parameter at which the sound's volume begins to linearly attenuate. falloffEnd - Value greater than or equal to falloffStart for controlling the distance from the position parameter at which the sound's volume falls to zero.

Return Values

None

Description

Starts playing the sound stream at a 3D position in the world. Volume of 1.0 is 100% volume.

Examples

```
bgmusic = SoundStream.Find( "music" )
SoundStream.Play3d( bgmusic, true, 1.0, Vector4.Create(0, 10, 0, 0), 10, 20 )
```

See Also

- SoundStream.Stop

SoundStream.Resume

Brief

Resume the sound stream.

Definition

```
SoundStream.Resume( SoundStream stream )
```

Arguments

stream - The sound stream object.

Return Values

None

Description

Resume the sound stream.

Examples

```
bgmusic = SoundStream.Find( "music" )
SoundStream.Resume( bgmusic )
```

See Also

- SoundStream.Play2d
- SoundStream.Play3d

SoundStream.SetCurrentTime

Brief

Set the current playback time for the stream.

Definition

```
boolean SoundStream.SetCurrentTime( SoundStream stream, number time )
```

Arguments

stream - The sound stream object.time - the time (in seconds) to set

Return Values

result - true if setted false if not

Description

Set the current playback time for the stream.

Examples

```
bgmusic = SoundStream.Find( "music" )
result = SoundStream.SetCurrentTime( 10 )
```

See Also

- SoundStream.GetCurrentTime

SoundStream.SetPosition

Brief

Set the 3D world position of the stream

Definition

```
SoundStream.SetPosition( SoundStream stream, Vector4 position )
```

Arguments

stream - The sound stream object.position - 3D world space coordinate of the sound (w-component is ignored)

Return Values

None

Description

Set the 3D world position of the stream

Examples

```
bgmusic = SoundStream.Find( "music" )
SoundStream.SetPosition( bgmusic, Vector4.Create(0, 10, 0, 0) )
```

See Also

- SoundStream.Play3d

SoundStream.SetVolume

Brief

Set the sound stream's volume.

Definition

```
SoundStream.SetVolume( SoundStream stream, number volume )
```

Arguments

stream - The sound stream object.volume - value between 0.0 and 1.0 for controlling the volume of the stream.

Return Values

None

Description

Set the sound stream's volume.

Examples

```
bgmusic = SoundStream.Find( "music" )
SoundStream.SetVolume( bgmusic, 0.5 )
```

See Also

- SoundStream.GetVolume

SoundStream.Stop

Brief

Stops the sound stream.

Definition

```
SoundStream.Stop( SoundStream stream )
```

Arguments

stream - The sound stream object.

Return Values

None

Description

Stops the sound stream.

Examples

```
bgmusic = SoundStream.Find( "music" )
SoundStream.Stop( bgmusic )
```

See Also

- SoundStream.Play2d
- SoundStream.Play3d

Sprite

Sprite.Create

Brief

Creates a new Sprite object.

Definition

```
Sprite Sprite.Create()
```

Arguments

None

Return Values

A new Sprite instance.

Description

Creates a new Sprite object.

Examples

```
sprite = Sprite.Create()
```

See Also

- Renderer.DrawSprite

Sprite.GetColor

Brief

Returns the color of the sprite as a Vector4.

Definition

```
Vector4 Sprite.GetColor( Sprite sprite )
Sprite.GetPosition( Sprite sprite, Vector4 outColor )
```

Arguments

sprite - The sprite to query.outColor - (out) The vector to receive the color.

Return Values

A Vector4 containing the RGBA values (in that order) of the sprite.

Description

Returns the color of the sprite as a Vector4 where x is red, y, is green, z is blue, w is alpha. They range from 0 to 1 where 1 is the maximum color.

Examples

```
col = Sprite.GetColor( sprite )
print( col )
```

See Also

- Sprite.SetColor

Sprite.GetElapsedAnimTime

Brief

Returns the elapsed animation time in seconds.

Definition

```
number Sprite.GetElapsedAnimTime( Sprite sprite )
```

Arguments

sprite - The sprite to query.

Return Values

The elapsed time in seconds of the sprite's animation.

Description

Returns the elapsed animation time in seconds.

Examples

```
print( Sprite.GetElapsedAnimTime( sprite ) )
```

See Also

- Sprite.GetTotalAnimTime
- Sprite.IsAnimPaused
- Sprite.PlayAnim
- Sprite.ResumeAnim

- Sprite.StopAnim

Sprite.GetPosition

Brief

Returns the screen space coordinate of a sprite.

Definition

```
Vector4 Sprite.GetPosition( Sprite sprite )
Sprite.GetPosition( Sprite sprite, Vector4 outPos )
```

Arguments

sprite - The sprite to query.outPos - (out) The vector to receive the position.

Return Values

A Vector4 object containing the screen space position of the sprite. Only the X and Y members will be valid.

Description

Returns the position of the sprite. Note that this vector is a copy created by the function and so changing its value will not change the sprite.

Examples

```
pos = Sprite.GetPosition( sprite )
print( pos )
```

See Also

- Sprite.SetPosition

Sprite.GetRotation

Brief

Returns the angle of rotation of the sprite.

Definition

```
number Sprite.GetRotation( Sprite sprite )
```

Arguments

sprite - The sprite to query.

Return Values

A floating point value containing the rotation in degrees of the sprite.

Description

Returns the angle of rotation in degrees of the sprite.

Examples

```
rot = Sprite.GetRotation( sprite )
print( rot .. " degrees" )
```

See Also

- Sprite.SetRotation

Sprite.GetScale

Brief

Returns as a Vector4 the scale of the sprite.

Definition

```
Vector4 Sprite.GetScale( Sprite sprite )
Sprite.GetScale( Sprite sprite, Vector4 outScale )
```

Arguments

sprite - The sprite to query.outScale - (out) The vector to receive the scale.

Return Values

A Vector4 instance containing the sprite's scale. Only the X and Y members will be valid.

Description

Returns the scale of the sprite. Note that this vector is a copy of the vector and so changing its value will not change the sprite.

Examples

```
scale = Sprite.GetScale( sprite )
print( scale )
```

See Also

- Sprite.GetScale

Sprite.GetTotalAnimTime

Brief

Returns the total animation time in seconds.

Definition

```
number Sprite.GetTotalAnimTime( Sprite sprite )
```

Arguments

sprite - The sprite to query.

Return Values

The length in seconds of the sprite's animation.

Description

Returns the total animation time in seconds. If the animation is looping it returns the total time for one loop of the animation.

Examples

```
print( Sprite.GetTotalAnimTime( sprite ) )
```

See Also

- Sprite.GetElapsedAnimTime
- Sprite.IsAnimPaused
- Sprite.PlayAnim
- Sprite.ResumeAnim
- Sprite.StopAnim

Sprite.IsAnimPaused

Brief

Reports if the animation is paused.

Definition

```
boolean Sprite.IsAnimPaused( Sprite sprite )
```

Arguments

sprite - The sprite to query.

Return Values

true if the sprite's animation is paused, false otherwise.

Description

Returns true if the sprite's animation is paused (using Sprite.PauseAnim) and false otherwise.

Examples

```
if Sprite.IsAnimPaused( sprite ) then
    print( "Animation is paused" )
end
```

See Also

- Sprite.GetElapsedAnimTime
- Sprite.GetTotalAnimTime
- Sprite.PlayAnim
- Sprite.ResumeAnim
- Sprite.StopAnim

Sprite.IsAnimPlaying

Brief

Reports if an animation is playing on the sprite.

Definition

```
boolean Sprite.IsAnimPlaying( Sprite sprite )
```

Arguments

sprite - The sprite to query.

Return Values

true if an animation is playing and false otherwise.

Description

Returns true if an animation is playing and false otherwise. Note that it is still considered as playing even if the animation is paused.

Examples

```
if Sprite.IsAnimPlaying( sprite ) then
    print( "Playing an animation" )
end
```

See Also

- Sprite.GetElapsedAnimTime
- Sprite.GetTotalAnimTime
- Sprite.PlayAnim
- Sprite.ResumeAnim
- Sprite.StopAnim

Sprite.PauseAnim

Brief

Pauses an animation.

Definition

```
Sprite.PauseAnim( Sprite sprite )
```

Arguments

sprite - The sprite to update.

Return Values

None.

Description

Pauses an animation. It can then be resumed with ResumeAnim.

Examples

```
Sprite.PauseAnim( sprite )
```

See Also

- Sprite.GetElapsedAnimTime
- Sprite.GetTotalAnimTime
- Sprite.PlayAnim
- Sprite.ResumeAnim
- Sprite.StopAnim

Sprite.PlayAnim

Brief

Plays a texture animation on the sprite.

Definition

```
Sprite.PlayAnim( Sprite sprite, SpriteAnim anim, number framesPerSecond = 1.0 )
```

Arguments

sprite - The sprite to update.anim - The animation to apply on the sprite.framesPerSecond - The number of animation frames to be displayed every second.

Return Values

None.

Description

Plays a texture animation with the specified anim at the given animation speed. The Sprites UVs are overwritten with the UVs from the animation.

Examples

```
Sprite.PlayAnim( sprite, anim, 30 )
```

See Also

- Sprite.GetElapsedAnimTime
- Sprite.GetTotalAnimTime
- Sprite.PauseAnim
- Sprite.ResumeAnim
- Sprite.StopAnim

Sprite.ResumeAnim

Brief

Resumes a paused animation.

Definition

```
Sprite.ResumeAnim( Sprite sprite )
```

Arguments

sprite - The sprite to update.

Return Values

None.

Description

Resumes a paused animation.

Examples

```
Sprite.ResumeAnim( sprite )
```

See Also

- Sprite.GetElapsedAnimTime
- Sprite.GetTotalAnimTime
- Sprite.PauseAnim
- Sprite.PlayAnim
- Sprite.StopAnim

Sprite.SetColor

Brief

Sets the color of the sprite.

Definition

```
Sprite.SetColor( Sprite sprite, Vector4 rgba )
Sprite.SetColor( Sprite sprite, number red = 1, number green = 1, number blue = 1, number alpha = 1 )
```

Arguments

sprite - The sprite to update.rgba - A Vector4 containing the RGBA values where each component must be between 0.0 and 1.0.red - The red component, ranges from 0 to 1. Defaults to 1.green - The green component, ranges from 0 to 1. Defaults to 1.blue - The blue component, ranges from 0 to 1. Defaults to 1.alpha - The alpha component, ranges from 0 to 1. Defaults to 1.

Return Values

None.

Description

Sets the color of the sprite.

Examples

```
redHalfAlpha = Vector4.Create( 1, 0, 0, 0.5 )
sprite = Sprite.Create()
Sprite.SetColor( sprite, redHalfAlpha )
```

See Also

- Sprite.GetColor

Sprite.SetPosition

Brief

Sets the position of the sprite.

Definition

```
Sprite.SetPosition( Sprite sprite, Vector4 position )
Sprite.SetPosition( Sprite sprite, number x = 0, number y = 0 )
```

Arguments

sprite - The sprite to update.position - The new sprite position, where only the X and Y members are used.x - The X component of the sprite position.y - The Y component of the sprite position.

Return Values

None.

Description

Sets the screen coordinate of the sprite. The position specifies the coordinate of the center of the sprite.

Examples

```
pos = Vector4.Create( 10, 10 )
sprite = Sprite.Create()
Sprite.SetPosition( sprite, pos )
```

See Also

- Sprite.GetPosition

Sprite.SetRotation

Brief

Sets the rotation of the sprite.

Definition

```
Sprite.SetRotation( Sprite sprite, number rotation )
```

Arguments

sprite - The sprite to update.rotation - The new sprite rotation (in degrees).

Return Values

None.

Description

Sets the rotation of the sprite in degrees.

Examples

```
sprite = Sprite.Create()
Sprite.SetRotation( sprite, 90.0 )
```

See Also

- Sprite.GetRotation

Sprite.SetScale

Brief

Sets the scale of the sprite.

Definition

```
Sprite.SetScale( Sprite sprite, Vector4 vector )
Sprite.SetScale( Sprite sprite, number x = 1, number y = 1 )
```

Arguments

sprite - The sprite to update.vector - The new sprite scale, where only the X and Y members are used.x - The horizontal sprite scale.y - The vertical sprite scale.

Return Values

None.

Description

Sets the x and y scale of the sprite.

Examples

```
scale = Vector4.Create( 10, 10 )
sprite = Sprite.Create()
Sprite.SetScale( sprite, scale )
```

See Also

- Sprite.GetScale

Sprite.SetTexture

Brief

Sets the current texture of the sprite.

Definition

```
Sprite.SetTexture( Sprite sprite, Texture texture )
```

Arguments

sprite - The sprite to update.texture - An instance of the Texture class that will represent the graphic of the sprite.

Return Values

None.

Description

Sets the current texture of the sprite. Note that this texture is replaced if an animation is played on the sprite. The UVs are set to default, i.e. the whole of the texture.

Examples

```
sprite = Sprite.Create()
Sprite.SetTexture( sprite, Texture.Find( "SpriteTexture" ) )
```

See Also

- Sprite.SetUvs

Sprite.SetUvs

Brief

Sets the UVs of the sprite.

Definition

```
Sprite.SetUvs( Sprite sprite, number tlu, number tlv, number bru, number brv )
Sprite.SetUvs( Sprite sprite, Vector4 tlbr )
Sprite.SetUvs( Sprite sprite, number tlu, number tlv, number blu, number blv, number bru, number brv,
number tru, number trv )
Sprite.SetUvs( Sprite sprite, Vector4 tlbl, Vector4 brtr )
```

Arguments

sprite - The sprite to update.tlu - The u-component of the top left texture coordinate for the sprite.tlv - The v-component of the top left texture coordinate for the sprite.blu - The u-component of the bottom left texture coordinate for the sprite.blv - The v-component of the bottom left texture coordinate for the sprite.bru - The u-component of the bottom right texture coordinate for the sprite.brv - The v-component of the bottom right texture coordinate for the sprite.tru - The u-component of the top right texture coordinate for the sprite.trv - The v-component of the top right texture coordinate for the sprite.tlbr - A vector containing the top left and bottom right UV coordinates for the sprite.tlbl - A vector containing the top left and bottom left UV coordinates for the sprite.brtr - A vector containing the bottom right and top right UV coordinates for the sprite.

Return Values

None.

Description

Sets the UVs of the sprite. If passing in 4 numbers either as numbers or packed inside a Vector4 it assumes a rectangle and generates the two other uvs from the two uvs specified. If 4 uvs are supplied it uses them as is. The default uvs for a Sprite are [0,0],[0,1],[1,1],[1,0]. Also, UVs are set to the default when Sprite.SetTexture is called or Sprite.PlayAnim.

Examples

```
sprite = Sprite.Create()
Sprite.SetTexture( sprite, texture )

-- Example 1: Set to default UVs using each prototype
Sprite.SetUvs( sprite, 0, 0, 1, 1 )
Sprite.SetUvs( sprite, Vector4.Create( 0, 0, 1, 1 ) )
Sprite.SetUvs( sprite, 0, 0, 0, 1, 1, 1, 1, 0 )
Sprite.SetUvs( sprite, Vector4.Create( 0, 0, 0, 1 ), Vector4.Create( 1, 1, 1, 0 ) )

-- Example 2: Set the sprite to use the top left quadrant of the texture
Sprite.SetUvs( sprite, 0, 0, 0.5, 0.5 )
```

See Also

- Sprite.SetTexture
- Sprite.PlayAnim

Sprite.StopAnim

Brief

Stops the current animation if it is playing.

Definition

```
Sprite.StopAnim( Sprite sprite )
```

Arguments

sprite - The sprite to update.

Return Values

None.

Description

Stops the current animation if it is playing. The sprite's texture is left at whatever was last displayed.

Examples

```
Sprite.StopAnim( sprite )
```

See Also

- Sprite.GetElapsedAnimTime
- Sprite.GetTotalAnimTime
- Sprite.PauseAnim
- Sprite.PlayAnim
- Sprite.ResumeAnim

SpriteAnim

SpriteAnim.AddFrames

Brief

Adds one or more new frames to an animation.

Definition

```
SpriteAnim.AddFrames( SpriteAnim anim, Texture texture, number frameWidth = textureWidth, number
frameHeight = textureHeight )
```

Arguments

texture - the new frame of animation.frameWidth - the width of the frame (defaults to the width of the specified texture).frameHeight - the width of the frame (defaults to the height of the specified texture).

Return Values

None.

Description

Adds one or more new frames to an animation. If frameWidth and/or frameHeight is specified it subdivides the texture up into several frames, where frames are in left to right and then top to bottom order.

Examples

```
-- Create a 2 frame sprite animation
anim = SpriteAnim.Create()
SpriteAnim.AddFrames( anim, Texture.Find( "image1" ) )
SpriteAnim.AddFrames( anim, Texture.Find( "image2" ) )

-- Loop the animation every second
sprite = Sprite.Create()
Sprite.PlayAnim( sprite, anim, 2 )
```

See Also

- Sprite.Create
- Sprite.PlayAnim
- SpriteAnim.Create

SpriteAnim.Create

Brief

Creates a new SpriteAnim object.

Definition

```
SpriteAnim SpriteAnim.Create()
```

Arguments

None

Return Values

A new SpriteAnim instance.

Description

Creates a new SpriteAnim object.

Examples

```
anim = SpriteAnim.Create()
```

See Also

- SpriteAnim.AddFrames

SpriteAnim.IsLooping

Brief

Reports if the animation is looping or not.

Definition

```
boolean SpriteAnim.IsLooping( SpriteAnim anim )
```

Arguments

anim - The srite anim object.

Return Values

Returns true if the animation has been set to looping and false otherwise.

Description

Reports if the animation is looping or not.

Examples

```
print( SpriteAnim.IsLooping( myAnim ) )
```

See Also

- SpriteAnim.SetLooping

SpriteAnim.SetLooping

Brief

Sets the animation looping status.

Definition

```
SpriteAnim.SetLooping( SpriteAnim anim, boolean value )
```

Arguments

value - true to make the animation to loop, false to make the animation play once.

Return Values

None.

Description

Sets the animation to loop or not.

Examples

```
anim = SpriteAnim.Create()
SpriteAnim.SetLooping( anim, true )
```

See Also

- SpriteAnim.IsLooping

StreamOps

StreamOps.AddFloat32

Brief

Add two float buffers or add a scalar onto a float buffer.

Definition

```
StreamOps.AddFloat32(MemoryContainer dest, number destOffset, MemoryContainer src1, number src1Offset,
                     MemoryContainer src2, number src2Offset, number count, number stride = 1)
StreamOps.AddFloat32(MemoryContainer dest, number destOffset, MemoryContainer src1, number src1Offset,
                     number scalar, number count, number stride = 1)
```

Arguments

dest - Destination memory container.destOffset - Offset into destination memory container at which to start writing results.src1 - First source memory container.src1Offset - Offset into first source memory container at which to start reading source data.src2 - Second source memory container.src2Offset - Offset into second source memory container at which to start reading source data.count - Number of floats to process.stride - Number of floats to increment offset by. Used for both sources and destination.scalar - A scalar to add to all source elements.

Return Values

None.

Description

Note that all offsets and strides are in "elements", float32s in this case. This function is equivalent to:
local offset = 0
for i = 1, count do
dest:SetFloat32(destOffset * 4 + offset, src1:GetFloat32(src1Offset * 4 + offset) + src2:GetFloat32(src2Offset * 4 + offset))
- OR dest:SetFloat32(destOffset * 4 + offset, src1:GetFloat32(src1Offset * 4 + offset) + scalar)
offset = offset + stride * 4
end

Examples

```

```

See Also

- StreamOps.SubFloat32
- StreamOps.MulFloat32
- StreamOps.DivFloat32

StreamOps.BoundsVec4

Brief

Compute the bounds of a list of Vector4 values.

Definition

```
StreamOps.BoundsVec4(MemoryContainer src, number srcOffset, number count, Vector4 min, Vector4 max,
number stride = 1)
```

Arguments

src - Source memory container. Operation will be faster if 16-byte aligned.srcOffset - Offset into source memory container at which to start processing.count - Number of Vector4 values to process.min - Vector4 to return the minimum bound into. Assumed to be pre-set to the desired initial value.max - Vector4 to return the maximum bound into. Assumed to be pre-set to the desired initial value.stride - Number of Vector4s to increment offset by with each vector processed.

Return Values

None.

Description

Note that all offsets and strides are in "elements", Vector4s in this case. This function is equivalent to:
local offset = srcOffset * 4
local min = Vector4.Create()
local max = Vector4.Create()
local v = Vector4.Create()
for i = 1, count do
src:GetVector4(offset, v)
Vector4.Min(min, min, v)
Vector4.Max(max, max, v)
offset = offset + stride * 16
end
Note that the min and max parameters are not initialized to any value in this function, they should be initialized by the script to appropriate starting values for the bounds calculation.

Examples

```
local min = Vector4.Create(Const.MaxNum, Const.MaxNum, Const.MaxNum)
local max = Vector4.Create(-Const.MaxNum, -Const.MaxNum, -Const.MaxNum)
StreamOps.BoundsVec4(verts, 0, numVerts, min, max)
```

See Also

None

StreamOps.DivFloat32

Brief

Divide two float buffers or divide a float buffer by a scalar.

Definition

```
StreamOps.DivFloat32(MemoryContainer dest, number destOffset, MemoryContainer src1, number src1Offset,
                     MemoryContainer src2, number src2Offset, number count, number stride = 1)
StreamOps.DivFloat32(MemoryContainer dest, number destOffset, MemoryContainer src1, number src1Offset,
                     number scalar, number count, number stride = 1)
StreamOps.DivFloat32(MemoryContainer dest, number destOffset, number scalar, MemoryContainer src1,
number src1Offset,
                     number count, number stride = 1)
```

Arguments

dest - Destination memory container.destOffset - Offset into destination memory container at which to start writing results.src1 - First source memory container.src1Offset - Offset into first source memory container at which to start reading source data.src2 - Second source memory container.src2Offset - Offset into second source memory container at which to start reading source data.count - Number of floats to process.stride - Number of floats to increment offset by. Used for both sources and destination.scalar - A scalar to apply to each source element.

Return Values

None.

Description

Note that all offsets and strides are in "elements", float32s in this case. This function is equivalent to:
local offset = 0
for i = 1, count do
dest:SetFloat32(destOffset * 4 + offset, src1:GetFloat32(src1Offset * 4 + offset) / src2:GetFloat32(src2Offset * 4 + offset))
- OR dest:SetFloat32(destOffset * 4 + offset, src1:GetFloat32(src1Offset * 4 + offset) + scalar)
offset = offset + stride * 4
end

Examples

See Also

- StreamOps.SubFloat32
- StreamOps.AddFloat32
- StreamOps.MulFloat32

StreamOps.Float32ToUInt8

Brief

Convert 32-bit floating point values to unsigned 8-bit integer values.

Definition

```
StreamOps.Float32ToUInt8(MemoryContainer dest, number destOffset, MemoryContainer src, number
srcOffset, number count, number stride = 1, number multiplier = 1)
```

Arguments

dest - Destination memory container.destOffset - Offset into destination memory container at which to start storing results.src - Source memory container.srcOffset - Offset into source memory container at which to start reading values.count - Number of floats to process.stride - Number of floats to increment offset by. Used for source only.multiplier - Number to multiply float by prior to conversion.

Return Values

None.

Description

A common use for this function might be to convert float colors in the range [0, 1] to byte colors in the range [0, 255].
Note that all offsets and strides are in "elements", float32s in this case. This function is equivalent to:
local offset = srcOffset * 4
for i = 1, count do
dest:SetUInt8(destOffset, src:GetFloat32(offset) * multiplier)
offset = offset + stride * 4
destOffset = destOffset + 1 - int8
end

Examples

See Also

None

StreamOps.HorizAddFloat32

Brief

Sum values within a memory container.

Definition

```
number StreamOps.HorizAddFloat32(MemoryContainer src, number srcOffset, number count, number stride =
1)
```

Arguments

src - Source memory container.srcOffset - Offset into source memory container at which to start adding values.count - Number of floats to process.stride - Number of floats to increment offset by. Used for both sources and destination.

Return Values

The sum of the values.

Description

Note that all offsets and strides are in "elements", float32s in this case. This function is equivalent to
```
local offset = srcOffset * 4
local sum = 0
for i = 1, count do
sum = sum + src:GetFloat32(offset)
offset = offset + stride * 4
end
```

Examples

See Also

- StreamOps.AddFloat32

StreamOps.MulFloat32

Brief

Multiply two float buffers or multiply a float buffer by a scalar.

Definition

```
StreamOps.MulFloat32(MemoryContainer dest, number destOffset, MemoryContainer src1, number src1Offset,
                     MemoryContainer src2, number src2Offset, number count, number stride = 1)
StreamOps.MulFloat32(MemoryContainer dest, number destOffset, MemoryContainer src1, number src1Offset,
                     number scalar, number count, number stride = 1)
```

Arguments

dest - Destination memory container.destOffset - Offset into destination memory container at which to start writing results.src1 - First source memory container.src1Offset - Offset into first source memory container at which to start reading source data.src2 - Second source memory container.src2Offset - Offset into second source memory container at which to start reading source data.count - Number of floats to process.stride - Number of floats to increment offset by. Used for both sources and destination.scalar - A scalar to multiply each source element by.

Return Values

None.

Description

Note that all offsets and strides are in "elements", float32s in this case. This function is equivalent to:
```
local offset = 0
for i = 1, count do
dest:SetFloat32( destOffset * 4 + offset, src1:GetFloat32(src1Offset * 4 + offset) * src2:GetFloat32(src2Offset * 4 + offset) )
- OR dest:SetFloat32( destOffset * 4 + offset, src1:GetFloat32(src1Offset * 4 + offset) + scalar )
offset = offset + stride * 4
end
```

Examples

See Also

- StreamOps.SubFloat32

- StreamOps.AddFloat32
- StreamOps.DivFloat32

StreamOps.ShufFloat32

Brief

Shuffle a float buffer based on a supplied index table.

Definition

```
StreamOps.ShufFloat32(MemoryContainer dest, number destOffset, MemoryContainer src1,
                      MemoryContainer indices, number count, number stride = 1)
```

Arguments

dest - Destination memory container.destOffset - Offset into destination memory container at which to start writing results.src1 - Source memory container.indices - Index memory container. Indices are float offsets into src1.count - Number of floats to process.stride - Number of floats to increment offset by. Used for both sources and destination.

Return Values

None.

Description

If the object has the debug flag set, all indices will be range-checked. Otherwise, it is the responsibility of the developer to ensure that all indices are valid and within the size of the source memory container. Note that all offsets and strides are in "elements", float32s in this case. This function is equivalent to:

```
local offset = 0
local indexOffset = 0
for i = 1, count do
local index = indices:GetInt16(indexOffset)
dest:SetFloat32(destOffset * 4 + offset, src1:GetFloat32(index * 4))
offset = offset + stride * 4
indexOffset = indexOffset + 2 - int16
end
```

Examples

See Also

None

StreamOps.ShufInt8

Brief

Shuffle an int8 buffer based on a supplied index table.

Definition

```
StreamOps.ShufInt8(MemoryContainer dest, number destOffset, MemoryContainer src1,
                      MemoryContainer indices, number count, number stride = 1)
```

Arguments

dest - Destination memory container.destOffset - Offset into destination memory container at which to start writing results.src1 - Source memory container.indices - Index memory container. Indices are 16-bit (int16) byte offsets into src1.count - Number of bytes to process.stride - Number of bytes to increment offset by. Used for both sources and destination.

Return Values

None.

Description

If the object has the debug flag set, all indices will be range-checked. Otherwise, it is the responsibility of the developer to ensure that all indices are valid and within the size of the source memory container. Note that all offsets and strides are in bytes (int8s). This function is equivalent to:

```
local offset = 0
local indexOffset = 0
for i = 1, count do
local index = indices:GetInt16(indexOffset)
dest:SetInt8(destOffset + offset, src1:GetInt8(index))
offset = offset + stride
indexOffset = indexOffset + 2 - int16
end
```

Examples

See Also

None

StreamOps.SubFloat32

Brief

Calculate difference of two float buffers or subtract a scalar from a float buffer.

Definition

```
StreamOps.SubFloat32(MemoryContainer dest, number destOffset, MemoryContainer src1, number src1Offset,
                     MemoryContainer src2, number src2Offset, number count, number stride = 1)
StreamOps.SubFloat32(MemoryContainer dest, number destOffset, MemoryContainer src1, number src1Offset,
                     number scalar, number count, number stride = 1)
StreamOps.SubFloat32(MemoryContainer dest, number destOffset, number scalar, MemoryContainer src1,
number src1Offset,
                     number count, number stride = 1)
```

Arguments

dest - Destination memory container.destOffset - Offset into destination memory container at which to start writing results.src1 - First source memory container.src1Offset - Offset into first source memory container at which to start reading source data.src2 - Second source memory container.src2Offset - Offset into second source memory container at which to start reading source data.count - Number of floats to process.stride - Number of floats to increment offset by. Used for both sources and destination.scalar - A scalar to apply to each source element.

Return Values

None.

Description

Note that all offsets and strides are in "elements", float32s in this case. This function is equivalent to:

```
local offset = 0
for i = 1, count do
dest:SetFloat32( destOffset * 4 + offset, src1:GetFloat32(src1Offset * 4 + offset) - src2:GetFloat32(src2Offset * 4 + offset) )
- OR dest:SetFloat32( destOffset * 4 + offset, src1:GetFloat32(src1Offset * 4 + offset) + scalar )
offset = offset + stride * 4
end
```

Examples

See Also

- StreamOps.AddFloat32
- StreamOps.MulFloat32
- StreamOps.DivFloat32

System

System.AddChatLogMessage

Brief

Adds a lua message in the chat log.

Definition

```
System.AddChatLogMessage( string message, string nameOfSpeaker )
```

Arguments

message - The message to add to the chat log.nameOfSpeaker - The name of the speaker (optional).

Return Values

None.

Description

Adds a lua message in the chat log. If the chat log is not visible when this function is called, it will activate and display the message. The message text will appear in a different color to the other messages (r = 0.4, g = 0.7, b = 1.0).

Examples

```
me = LocalPlayer.GetPerson()
System.AddChatLogMessage( 'Good morning ' .. Person.GetName( me ) )
```

See Also

None

System.CommercePointIsClosed

Brief

Returns true if the commerce point is closed and false if it is open (active).

Definition

```
boolean System.CommercePointIsClosed()
```

Arguments

None

Return Values

Returns true if the commerce point is closed and false if it is open (active).

Description

Returns true if the commerce point is closed and false if it is open (active).

Examples

```
if System.CommercePointIsClosed() == true then
    System.CommercePointOpen( myUrl )
end
```

See Also

- System.CommercePointOpen

System.CommercePointOpen

Brief

Opens a commerce point with the specified url.

Definition

```
number System.CommercePointOpen( string url, string rootNodeId = nil, string initNodeId = nil )
```

Arguments

url - The url of the commerce point.rootNodeId - The "node id" of the commerce point xml element to treat as the "root node" of the commerce point.initNodeId - The "node id" of the commerce point xml element to treat as the "init node" of the commerce point.

Return Values

The result of the open request. Return values are:
1 - commerce point was opened
-1 - commerce point is already running

Description

Opens a commerce point with the specified url. See the return result to see if it failed or succeeded. The URL for the commerce point requires the correct path - these are set up using a standard format as follows:
SCENE_NAME/REGION_CODE/CP<REGION_CODE><CP_ShortNAME><2_CHAR_NUM>.xml

where:

SCENE_NAME is the name of the scene including the code on the end
e.g.
E_COMPA_DW1_5C2G_J5V
XFORM_Lounge_2_9KEL_1N8L
SCEE_PureFire_Lobby_0R2J_766

REGION_CODE is the PSN/PMT region code (SCEE=E, SCEA=A, SCEJ=J, SCEAsia=H)
CP_ShortNAME is a 'shortcode' that should be meaningful to you.
2_CHAR_NUM is simply a two character number which can mean anything, and again should be meaningful you (e.g. a version number)

When opening a commerce point on demand using the call, simply build the commerce point file path using the known information.

For example:

SCEA user opening a specific commerce point:
XFORM_Lounge_2_9KEL_1N8L/A/CPAXFL00.xml

where the CP_ShortNAME="XFL" and 2_CHAR_NUM="00"

SCEE user opening the same commerce point for their region:
XFORM_Lounge_2_9KEL_1N8L/E/CPEXFL00.xml If the rootNodeId parameter is specified, the "root node" will be set. If the root node is a <node> element, then navigation of the commerce point will be restricted to the browser page corresponding to that element, all child pages, the item info page and the basket. If the root node is a <object> or <product> element, then navigation of the commerce point will be restricted to the item info page and basket. If the initNodeId parameter is specified, the "init node" will be set. If the init node is a <node> element, then navigation of the commerce point will begin at the browser page corresponding to that element. If the init node is a <object> or <product> element, then navigation of the commerce point will begin at the item info page.

Examples

```
System.CommercePointOpen( 'XFORM_Lounge_2_9KEL_1N8L/E/CPEXFL00.xml' )
```

```
System.CommercePointOpen( 'XFORM_Lounge_2_9KEL_1N8L/E/CPEXFL00.xml', 'male headgear' )
```

```
System.CommercePointOpen( 'XFORM_Lounge_2_9KEL_1N8L/E/CPEXFL00.xml', 'male headgear', 'big hat' )
```

```
System.CommercePointOpen( 'XFORM_Lounge_2_9KEL_1N8L/E/CPEXFL00.xml', nil, 'big hat' )
```

See Also

- System.CommercePointIsClosed

System.DrawText2dInScene

Brief

Allow text to be displayed in 3D space.

Definition

```
System.DrawText2dInScene(string text, Vector4 position, Vector4 color,
                  Text3dAlign alignment, number size,
                  boolean isShadowed, Vector4 shadowColor,
                  boolean isBold, boolean isAlwaysOnTop, boolean isFixedSize)
```

Arguments

text - the text to be displayed.position - the 3D position at which the text will appear.color - the color of the text to be displayed.alignment - the alignment of the text, in relation to the position given.
This will be one of the following:

- Text3dAlignment.TopLeft
- Text3dAlignment.TopMiddle
- Text3dAlignment.TopRight
- Text3dAlignment.Left
- Text3dAlignment.Center
- Text3dAlignment.Right
- Text3dAlignment.BottomLeft
- Text3dAlignment.BottomMiddle
- Text3dAlignment.BottomRight
  Default is 'Text3dAlignCenter'.size - the size of the text. Default is '1.0',isShadowed - whether the text should have a drop shadow. Default is 'false'.shadowColor - the colour of the shadow, if 'isShadowed' is set to true. Default is black, if no color is provided.isBold - whether the text is in bold font. Default is 'false'.isAlwaysOnTop - whether the text should adhere to 3D geometry obscuring it. Default is 'false'.isFixedSize - whether the text should resize depending on distance away from camera. Default is 'false'.

Return Values

Description

Displays 2D text in 3D space, much like the avatar label user name.

Examples

```
System.DrawText2dInScene(text, position, color)
```

See Also

None

System.EnableIdleDisconnect

Brief

Sets whether the idle disconnect is allowed.

Definition

```
System.EnableIdleDisconnect(boolean enable)
```

Arguments

enable - True to enable or false to disable.

Return Values

None.

Description

If set to false, the user will not be disconnected if idle for too long. This can be useful for disabling the auto-disconnect while the user is watching a full-length video for example. The idle disconnect defaults to enabled.

Examples

```
System.EnableIdleDisconnect(false)
```

See Also

None

System.ExecTitle

Brief

Launches an installed title.

Definition

```
number System.ExecTitle( string titleId, number param, string errorCallback )
```

Arguments

titleId - The title ID of the title to launch.param - Integer value that will be passed through to the launched title.errorCallback - Callback function which will be invoked if the title fails to execute.

Return Values

0 if title not available, 1 otherwise.

Description

Launches an installed title. The integer value that is passed through to the game can be queried using the execData parameter of cellGameGetBootGameInfo. In the case where user confirmation is required before proceeding with ExecTitle, for example confirmation to leave a Game Launching session, 1 will be returned to indicate no errors so far and ExecTitle will only proceed if user confirms. If user cancels then the error callback will be invoked and this particular case should be handled. If the title fails to execute the callback will be invoked, within this callback use System.GetExecTitleErrorCode() to retreive the error code. Scripts that make use of this function must also include a game_launch_properties component in the object to allow PS Home to manage any potential system conflicts. Failure to comply will result in a Lua error being generated.

Examples

```
gameOptions = 10
if GameInfo.IsGamesListBusy() == false then
    System.ExecTitle( "XXXX12345", gameOptions, "MyLaunchErrorCB()" )
end
```

See Also

- GameInfo.IsGamesListBusy
- GameInfo.SearchByTitleId
- GameInfo.IsSearchFinished
- GameInfo.GetSearchResults

System.GetChatLogMode

Brief

Gets the current display mode of the Chat Log.

Definition

```
ChatLogMode System.GetChatLogMode()
```

Arguments

None

Return Values

The current mode of the Chat Log. This will be one of the following:

- ChatLogMode.Minimized
- ChatLogMode.Maximized
- ChatLogMode.Closed

Description

Returns whether the mode of the Chat Log is minimized (visible but not interactive), maximized (visible and interactive), or closed.

Examples

```
if (System.GetChatLogMode() ~= ChatLogMode.Closed) then
    HideHud()
end
```

See Also

- System.SetChatLogMode

System.GetConsoleTestValue

Brief

Gets the value of the specified test key

Definition

```
string System.GetConsoleTestValue()
```

Arguments

None

Return Values

Description

Returns the value for a specified test key, which can be set (in the HDK and QA/test builds) by using the command 'SetConsoleTest' either manually through the debug console or through a hubstartup.txt file

Examples

```
[With 'SetConsoleTest MyKey,MyValue' included in hubstartup.txt]
value = System.GetConsoleTestValue( 'MyKey' )
```

See Also

None

System.GetExecTitleErrorCode

Brief

Retrieves the error code set upon ExecTitle failure.

Definition

```
number System.GetExecTitleErrorCode()
```

Arguments

Return Values

error code

Description

Retrieves the error code set upon ExecTitle failure. This function is only valid within the error callback specified as part of the System.ExecTitle() call. Error codes: 1 - unknown error 2 - parental level restricted 3 - exec title cancelled The script should use this opportunity to gracefully recover from the failure.

Examples

```
function LaunchTitle()
            System.ExecTitle( "XXXX12345", gameOptions, "MyLaunchErrorCB()" )
        end

        function MyLaunchErrorCB()
            local errorCode = System.GetExecTitleErrorCode()

            -- Insert script here to handle the error
        end
```

See Also

System.GetNatType

Brief

Gets the NAT type of the user.

Definition

```
int System.GetNatType()
```

Arguments

None

Return Values

The NAT type for the user, which will be either 1, 2 or 3.

Description

The user's NAT type will depend on their network configuration. Users with NAT type 1 and 2 should not have any issues with PS Home functionality. Users on NAT type 3 may not be able to fully join RtGame sessions.

Examples

```
allowUserToHostRtGame = true
if (System.GetNatType() == 3) then
    allowUserToHostRtGame = false
    -- display OSD informing user of the issue
end
```

See Also

None

System.GetRegionsTerritory

Brief

Returns the territory of the specified region.

Definition

```
string System.GetRegionsTerritory( string region )
```

Arguments

region - The region to get the territory for, e.g. "en-GB".

Return Values

The territory of the specified region.

Description

Returns the territory of the specified region. Can be one of the following:
"SCEE" (Europe)
"SCEA" (North America)
"SCEJ" (Japan)
"SCEAsia" (Rest of Asia)
Note: the default territory is set to SCEE - this means that if the region submitted is invalid then SCEE is returned.

Examples

```
currentRegion = LocalPlayer.GetCurrentRegion()
print( 'My territory is ' .. System.GetRegionsTerritory( currentRegion ) )
```

See Also

- LocalPlayer.GetCurrentRegion
- LocalPlayer.GetHomeRegion
- LocalPlayer.GetAccountRegion

System.GetScreenBottomRight

Brief

Returns the x and y pixel values of the bottom right of the screen.

Definition

```
Vector4 System.GetScreenBottomRight()
```

Arguments

None

Return Values

A Vector4 containing the x and y pixel values of the bottom right of the screen.

Description

Returns a Vector4 containing the x and y pixel values of the bottom right of the screen. The value differs between widescreen and 4/3 mode.

Examples

```
Renderer.DrawText2d( renderer, System.GetScreenBottomRight(), "Bottom right" )
```

See Also

- System.IsWideScreen
- System.GetScreenTopLeft

System.GetScreenTopLeft

Brief

Returns the x and y pixel values of the top left of the screen.

Definition

```
Vector4 System.GetScreenTopLeft()
```

Arguments

None

Return Values

A Vector4 containing the x and y pixel values of the top left of the screen.

Description

Returns a Vector4 containing the x and y pixel values of the top left of the screen. The value differs between widescreen and 4/3 mode.

Examples

```
Renderer.DrawText2d( renderer, System.GetScreenTopLeft(), "Top left" )
```

See Also

- System.IsWideScreen
- System.GetScreenBottomRight

System.GetUserValue

Brief

Gets a value for a specified system key

Definition

```
string System.GetUserValue(string key)
```

Arguments

key - Key to get

Return Values

Returns the key's value or NIL if none is set

Description

Gets a system option value. System keys are controlled by HPG, and relate to future systems.

Examples

```

```

See Also

None

System.GetVideoUploadResult

Brief

Retrieve the result of the last video upload request.

Definition

```
Result System.GetVideoUploadResult()
```

Arguments

None

Return Values

One of the following result codes:
VidUploadResult.Success - success
VidUploadResult.ErrBusy - the upload utility was busy
VidUploadResult.ErrNetwork - a network error occurred
VidUploadResult.ErrUnavailable - the upload service is currently unavailable
VidUploadResult.ErrQuote - service limit on upload size was exceeded
VidUploadResult.ErrMemory - out of memory
VidUploadResult.ErrFile - unable to open specified file
VidUploadResult.ErrUnknown - other unspecified error

Description

Reports the current status of the video upload utility.

Examples

```
if ( System.GetVideoUploadResult() ~= VidUploadResult.Success ) then
    print('Upload failed')
end
```

See Also

- System.UploadVideo
- System.GetVideoUploadStatus
- System.GetVideoUploadUrl

System.GetVideoUploadStatus

Brief

Retrieve the status of the current video upload request, if any.

Definition

```
Status System.GetVideoUploadStatus()
```

Arguments

None

Return Values

One of the following status codes:
VidUploadStatus.Idle - no current upload
VidUploadStatus.Busy - upload utility is currently busy with another task and is unavailable
VidUploadStatus.Uploading - currently uploading
VidUploadStatus.Error - an error occurred during upload

Description

Reports the current status of the video upload utility.

Examples

```
while ( System.GetVideoUploadStatus() == VidUploadStatus.Uploading ) do
    coroutine.yield()
end
```

See Also

- System.UploadVideo
- System.GetVideoUploadResult
- System.GetVideoUploadUrl

System.GetVideoUploadUrl

Brief

Retrieve the resulting URL of the last successfully uploaded video.

Definition

```
string System.GetVideoUploadUrl()
```

Arguments

None

Return Values

The URL as a string.

Description

Returns the URL of the uploaded video if upload succeeded, or an empty string if the upload failed.

Examples

```
print( 'Your video has been uploaded to ' .. System.GetVideoUploadUrl() )
```

See Also

- System.UploadVideo
- System.GetVideoUploadStatus
- System.GetVideoUploadResult

System.IsGameLaunchSessionActive

Brief

Query whether the local user is part of a game launch session.

Definition

```
boolean System.IsGameLaunchSessionActive()
```

Arguments

None

Return Values

True if the local user is part of a game launch session, or false otherwise.

Description

If the local user is part of a game launch session, a script may wish to prevent access to certain functionality. This function will indicate whether the user is joined to a game launch session.

Examples

```
if (not System.IsGameLaunchSessionActive()) then
    StartVideoRecording()
end
```

See Also

None

System.IsLookXInverted

Brief

Returns the user setting for X axis look invert option.

Definition

```
boolean System.IsLookXInverted()
```

Arguments

None

Return Values

true if the user's settings in PS Home has the X axis look inverted and false otherwise.

Description

Returns true if the user's settings in PS Home has the X axis look inverted and false otherwise.

Examples

```
if ( System.IsLookXInverted() ) then
    angle = -angle
end
```

See Also

None

System.IsLookYInverted

Brief

Returns the user setting for Y axis look invert option.

Definition

```
boolean System.IsLookYInverted()
```

Arguments

None

Return Values

true if the user's settings in PS Home has the Y axis look inverted and false otherwise.

Description

Returns true if the user's settings in PS Home has the Y axis look inverted and false otherwise.

Examples

```
if ( System.IsLookYInverted() ) then
    angle = -angle
end
```

See Also

None

System.IsVideoSystemLocked

Brief

Returns the status of the video system lock.

Definition

```
boolean System.IsVideoSystemLocked()
```

Arguments

None

Return Values

True if the lock is being used.

Description

This will check to see if any object currently has the video system locked. This function will not return true if the scene is using the video system - this is not considered a lock, only objects will lock the video system.

Examples

```
if ( not System.IsVideoSystemLocked() ) then
    if ( System.VideoSystemLock() ) then
        print("Video system locked!")
    else
        print("Oops, the scene is using the video system. Lock will never succeed.")
    end
else
    print("Oops, another object holds the lock! Can try again later.")
end
```

See Also

- System.VideoSystemLock
- System.VideoSystemUnlock
- Screen.Create

System.IsWideScreen

Brief

Queries whether PS Home is running in widescreen mode or not.

Definition

```
boolean System.IsWideScreen()
```

Arguments

None

Return Values

true if PS Home is running in widescreen mode and false otherwise.

Description

Returns true if PS Home is running in widescreen mode and false otherwise.

Examples

```
if System.IsWideScreen() then
    print( "You are in widescreen" )
end
```

See Also

- System.GetScreenBottomRight
- System.GetScreenTopLeft

System.PostScore

Brief

Posts a score to the score service

Definition

```
System.PostScore( string guid, number score, table metadata = nil )
```

Arguments

guid - A unique developer guid for external trackingscore - The users scoremetadata - An optional lua table containing metadata for this score

Return Values

Description

This function will upload a users score value to the score service with optional metadata. ill be encoded in Json and if any errors occur parsing the data the score will be uploaded without it.

Examples

```
LoadLibrary( "System" )
local score = 100
local data = { track ="Track9", car="Car4", time="1:23:57" }
System.PostScore( "8dd2eea4-fd37-48c1-9f90-d0cb16ca307f", score, data )
```

See Also

- Json.Decode

System.ProfanityFilterText

Brief

Filters a string through the profanity filter.

Definition

```
string System.ProfanityFilterText( string unfilteredText )
```

Arguments

unfilteredText - The string to be filtered for profanity.

Return Values

The string as filtered by the profanity filter.

Description

Filters a string through the profanity filter. To test if filtering happened, perform string equality test on original and returned text. Note that profanity filtering should be applied by the receiver of text (for display purposes) and not the sender.

Examples

```
displayText = System.ProfanityFilterText( receivedText )
```

See Also

None

System.SendNpMessage

Brief

Sends an NP message to the specified users.

Definition

```
System.SendNpMessage( User[] users, string subject, string body )
```

Arguments

users - An array of User objects.subject - The subject of the message.body - The body/main text of the message.

Return Values

None.

Description

Sends an NP message to the specified users. A popup appears requiring user interaction to send the message. Users are grouped in batches of 12. The maximum subject length is 18 bytes including 1 byte for null termination. The maximum text body is 512 bytes including 1 byte for null termination.

Examples

```
friends = LocalPlayer.GetFriends()
subject = "Game Launching"
body = "Hi! We're about to game launch. Come join us!"
System.SendNpMessage( friends, subject, body )
```

See Also

- LocalPlayer.GetFriends

System.SetChatLogMode

Brief

Sets the display mode of the Chat Log.

Definition

```
System.SetChatLogMode(ChatLogMode mode)
```

Arguments

mode - the desired mode of the Chat Log. This will be one of the following:

- ChatLogMode.Minimized
- ChatLogMode.Maximized
- ChatLogMode.Closed

Return Values

Description

Sets whether the mode of the Chat Log is minimized (visible but not interactive), maximized (visible and interactive), or closed.

Examples

```
System.SetChatLogMode(ChatLogMode.Maximized)
```

See Also

- System.GetChatLogMode

System.UploadVideo

Brief

Upload a specified video to a supported video sharing site.

Definition

```
boolean System.UploadVideo( FileHandle file, siteCode, ... )
```

Arguments

file - a handle to a video file obtained from VideoRecorder.GetOutputFilesiteCode - video sharing site ID code (see details)... - further parameters, depending on the site code (see details)

Return Values

true if upload was successfully initiated, false otherwise.

Description

Initiates an upload of the specified video to a supported video sharing site. A variety of extra metadata may be supplied depending on the site, see the list below:

YouTube (R) - [string clientId] [string developerKey] [string title], [string description], [string keyword1], <boolean isPrivate = true> <number rating = 0> <string keyword2 = ""), <string keyword3 = "">
clientId - Client ID chosen by developer, max 63 bytes ASCII alphanumeric characters only.
developerKey - Developer key on YouTube. A developer should obtain this from YouTube directly (see below).
title - Video title, max 60 bytes in UTF-8.
description - Video description, max 1023 bytes in UTF-8.
keyword1, keyword2, keyword3 - Video keywords, each max 24 bytes in UTF-8.
isPrivate - Private video (only visible to uploading user).
rating - Viewer age restriction, 0 indicates no restrictions.

The client ID can be chosen by the developer and represent the object using the upload API. The developer key must be obtained from YouTube (http://code.google.com/apis/youtube/dashboard/) and is separate from the user's username and password. The user will be prompted for their own username and password automatically as part of the upload. Also note that it is forbidden to charge a fee either directly or indirectly to upload videos to YouTube, this is against the YouTube Terms of Use.

[] = compulsory
<> = optional
The siteCodes available are:
VideoSite.YouTube
Note that a successful result from this function only indicates that the upload process has started successfully, you must check the progress using the GetVideoUploadStatus() function. The upload is not complete until the utility status is VidUploadStatus.Idle or VidUploadStatus.Error. If this function returns false, this will indicate the uploader is busy, the script may try again a fixed number of times but it should not assume that the upload will always succeed.

Examples

```
local video = VideoRecorder.GetOutputFile()
if ( System.UploadVideo( video, VideoSite.YouTube, myClientId, myDevKey, 'My Video', 'My PS Home
video', 'Home' ) ) then
    print('Video uploading...')
end
```

See Also

- System.GetVideoUploadStatus
- System.GetVideoUploadResult
- System.GetVideoUploadUrl
- VideoRecorder.GetOutputFile

System.VideoSystemLock

Brief

Allows a non scene object to attempt to gain access to the video system.

Definition

```
boolean System.VideoSystemLock()
```

Arguments

None

Return Values

True if the object successfully locked the video system or false otherwise.

Description

The VideoSystem is primarily used for scenes and scene objects. Other types of object may attempt to lock the video system for use if it is not already in use by the scene. If the lock returns true, this instance of the object may freely create and play videos. Once playback has finished, the object must unlock the video system for use by other objects. If the lock fails, no videos can be created by non scene objects. A script can distinguish between the two lock failure cases (another object having the lock, and the scene using the video system) by checking the IsVideoSystemLocked function; this will only return true in the former case.

Examples

```
local screen = Screen.Create()
if ( not System.IsVideoSystemLocked() ) then
    if ( System.VideoSystemLock() ) then
        print("Video system locked!")
        Screen.SetContent( screen, "http://www.mycontentserver.com/video.mp4" )
    else
        print("Oops, the scene is using the video system. Lock will never succeed.")
    end
else
    print("Oops, another object holds the lock! Can try again later.")
end
```

See Also

- System.VideoSystemUnlock
- System.IsVideoSystemLocked
- Screen.Create

System.VideoSystemUnlock

Brief

Unlocks the video system assuming the object currently has the lock.

Definition

```
System.VideoSystemUnlock()
```

Arguments

None

Return Values

None.

Description

Assuming a non scene object successfully acquired the lock for the video system, this function will release the lock once the object has finished using it. This must be called before termination of the object. The unlock call will only succeed once the video has been deleted so it may not be possible to immediately reacquire the lock after this call.

Examples

```
if ( not Screen.IsContentPlaying( myScreen ) ) then
    System.VideoSystemUnlock()
end
```

See Also

- System.VideoSystemLock
- System.IsVideoSystemLocked

- Screen.Create

System.WebBrowserIsClosed

Brief

Checks if the user has closed the browser.

Definition

```
boolean System.WebBrowserIsClosed()
```

Arguments

None

Return Values

true if the user has closed the browser, false otherwise.

Description

Checks if the user has closed the browser.

Examples

```
local hasFinished = System.WebBrowserIsClosed()
```

See Also

- System.WebBrowserOpen

System.WebBrowserOpen

Brief

Requests opening of the web browser.

Definition

```
number System.WebBrowserOpen( string url )
```

Arguments

url - The HTTP-based URL to open in the web browser.

Return Values

The result of the open request. Return values are:
1 - Browser was opened
0 - Browser is not available yet
-1 - Browser is already running
-2 - Browser is not permitted to run

Description

Checks for the availability of the web browser and opens it at the specified URL if available. The web browser that is opened supports Adobe Flash Player 7.

Examples

```
local browserWasOpened = System.WebBrowserOpen( "http://www.example.com/" )
```

See Also

- System.WebBrowserIsClosed

System.WriteScriptedGameLaunchXML

Brief

Writes XML data from a buffer to the file SCRIPTEDGAMELAUNCHSESSION.XML

Definition

```
System.WriteScriptedGameLaunchXML( memcontainer )
```

Arguments

memcontainer - Memory container containing XML data.

Return Values

None.

Description

Writes XML data from a buffer to the file SCRIPTEDGAMELAUNCHSESSION.XML in the hard drive directory given by the following Cell SDK function: int cellGameGetHomeLaunchOptionPath(char *commonPath, char *personalPath)

Examples

```
local container = MemoryContainer.Create( 2048 )
local writer = BasicGenx.Create( container )

BasicGenx.StartDocument( writer )
-- Write XML data into container here.
BasicGenx.EndDocument( writer )

System.WriteScriptedGameLaunchXML( container )
```

See Also

- BasicGenx.Create
- BasicGenx.EndDocument

Texture

Texture.Find

Brief

Looks up and returns the texture with the specified name.

Definition

```
Texture Texture.Find( string name, ResourcePack pack = nil )
```

Arguments

name - the string name associated with the texture as specified in an object's resources.xml file.pack - ResourcePack to load texture from, if required.

Return Values

The texture instance associated with the specified string name.

Description

Looks up and returns the texture with the specified name. Returns nil if the texture is not found. If a resource pack is specified, the resource pack is searched rather than the local object.

Examples

```
tex = Texture.Find( "myimage" )
tex2 = Texture.Find( "myimage", resourcePack )
```

See Also

- Texture.GetHeight
- Texture.GetWidth

Texture.GetHeight

Brief

Returns the height of the texture.

Definition

```
number Texture.GetHeight( Texture texture )
```

Arguments

texture - The texture to query.

Return Values

The height in pixels of the texture.

Description

Returns the height of the texture.

Examples

```
tex = Texture.Find( "myimage" )
height = Texture.GetHeight( tex )
```

See Also

- Texture.GetWidth

Texture.GetWidth

Brief

Returns the width of the texture.

Definition

```
number Texture.GetWidth( Texture texture )
```

Arguments

texture - The texture to query.

Return Values

The width in pixels of the texture.

Description

Returns the width of the texture.

Examples

```
tex = Texture.Find( "myimage" )
width = Texture.GetWidth( tex )
```

See Also

- Texture.GetHeight

Time

Time.AddDays

Brief

Add a specified number of days to the stored time.

Definition

```
Time.AddDays(Time t, number days)
```

Arguments

t - Time to modify.days - Number of days to add. Negative values will cause subtraction.

Return Values

None.

Description

Adds the specified number of days onto the time stored in the time object. Will correctly account for month lengths and leap years. To subtract, pass a negative value for days. Only integer values are permitted, fractional values will be truncated.

Examples

```
time = Time.Create()
time:SetFromServer()
time:AddDays(90)            -- add 90 days onto the current server time
```

See Also

- Time.AddMinutes

- Time.AddHours
- Time.AddSeconds
- Time.AddWeeks
- Time.AddMonths
- Time.AddYears

Time.AddHours

Brief

Add a specified number of hours to the stored time.

Definition

```
Time.AddHours(Time t, number hours)
```

Arguments

t - Time to modify.hours - Number of hours to add. Negative values will cause subtraction.

Return Values

None.

Description

Adds the specified number of hours onto the time stored in the time object. Will correctly account for month lengths and leap years. To subtract, pass a negative value for hours. Only integer values are permitted, fractional values will be truncated.

Examples

```
time = Time.Create()
time:SetFromServer()
time:AddHours(60)           -- add 60 hours onto the current server time
```

See Also

- Time.AddMinutes
- Time.AddSeconds
- Time.AddDays
- Time.AddWeeks
- Time.AddMonths
- Time.AddYears

Time.AddMinutes

Brief

Add a specified number of minutes to the stored time.

Definition

```
Time.AddMinutes(Time t, number minutes)
```

Arguments

t - Time to modify.minutes - Number of minutes to add. Negative values will cause subtraction.

Return Values

None.

Description

Adds the specified number of minutes onto the time stored in the time object. Will correctly account for month lengths and leap years. To subtract, pass a negative value for minutes.Only integer values are permitted, fractional values will be truncated.

Examples

```
time = Time.Create()
time:SetFromServer()
time:AddMinutes(45)         -- add 45 minutes onto the current server time
```

See Also

- Time.AddSeconds
- Time.AddHours
- Time.AddDays
- Time.AddWeeks
- Time.AddMonths
- Time.AddYears

Time.AddMonths

Brief

Add a specified number of months to the stored time.

Definition

```
Time.AddMonths(Time t, number months)
```

Arguments

t - Time to modify.months - Number of months to add. Negative values will cause subtraction.

Return Values

None.

Description

Adds the specified number of months onto the time stored in the time object. Will correctly account for month lengths and leap years. To subtract, pass a negative value for months. Only integer values are permitted, fractional values will be truncated.

Examples

```
time = Time.Create()
time:SetFromServer()
time:AddMonths(6)           -- add 6 months onto the current server time
```

See Also

- Time.AddMinutes
- Time.AddHours
- Time.AddDays
- Time.AddWeeks
- Time.AddSeconds
- Time.AddYears

Time.AddSeconds

Brief

Add a specified number of seconds to the stored time.

Definition

```
Time.AddSeconds(Time t, number seconds)
```

Arguments

t - Time to modify. seconds - Number of seconds to add. Negative values will cause subtraction.

Return Values

None.

Description

Adds the specified number of seconds onto the time stored in the time object. Will correctly account for month lengths and leap years. To subtract, pass a negative value for seconds. Only integer values are permitted, fractional values will be truncated.

Examples

```
time = Time.Create()
time:SetFromServer()
time:AddSeconds(30)         -- add 30 seconds onto the current server time
```

See Also

- Time.AddMinutes
- Time.AddHours
- Time.AddDays
- Time.AddWeeks
- Time.AddMonths
- Time.AddYears

Time.AddWeeks

Brief

Add a specified number of weeks to the stored time.

Definition

```
Time.AddWeeks(Time t, number weeks)
```

Arguments

t - Time to modify.weeks - Number of weeks to add. Negative values will cause subtraction.

Return Values

None.

Description

Adds the specified number of weeks onto the time stored in the time object. Will correctly account for month lengths and leap years. To subtract, pass a negative value for weeks. Only integer values are permitted, fractional values will be truncated.

Examples

```
time = Time.Create()
time:SetFromServer()
time:AddWeeks(8)            -- add 8 weeks onto the current server time
```

See Also

- Time.AddMinutes
- Time.AddHours
- Time.AddDays
- Time.AddSeconds
- Time.AddMonths
- Time.AddYears

Time.AddYears

Brief

Add a specified number of years to the stored time.

Definition

```
Time.AddYears(Time t, number years)
```

Arguments

t - Time to modify.years - Number of years to add. Negative values will cause subtraction.

Return Values

None.

Description

Adds the specified number of years onto the time stored in the time object. Will correctly account for month lengths and leap years. To subtract, pass a negative value for years. Only integer values are permitted, fractional values will be truncated.

Examples

```
time = Time.Create()
time:SetFromServer()
time:AddYears(1)            -- add 1 year onto the current server time
```

See Also

- Time.AddMinutes
- Time.AddHours
- Time.AddDays
- Time.AddWeeks
- Time.AddMonths
- Time.AddSeconds

Time.ConvertToLocal

Brief

Convert the stored time from UTC time to local time.

Definition

```
Time.ConvertToLocal(Time t)
```

Arguments

t - Time to modify.

Return Values

None.

Description

Converts a time specified in UTC time to local time, based on the current system settings. Note that this relies on the user having set the system timezone settings correctly, and as such should not be relied upon for absolute accuracy.

Examples

```
local t = Time.Create()
t:SetFromServer()
t:ConvertToLocal()
print("Current server time in local time is:", t)
```

See Also

- Time.ConvertToUtc

Time.ConvertToUtc

Brief

Convert the stored time from local time to UTC time.

Definition

```
Time.ConvertToUtc(Time t)
```

Arguments

t - Time to modify.

Return Values

None.

Description

Converts a time specified in local time to UTC time, based on the current system settings. Note that this relies on the user having set the system timezone settings correctly, and as such should not be relied upon for absolute accuracy.

Examples

```
local t = Time.Create()
t:SetFromLocal()
t:ConvertToUtc()
print("Current time in UTC is:", t)
```

See Also

- Time.ConvertToLocal

Time.Copy

Brief

Copy a Time value.

Definition

```
Time.Copy(Time destination, Time source)
```

Arguments

destination - Time value to copy into.source - Time value to copy.

Return Values

None.

Description

Copies a Time value from one instance to another.

Examples

```
local time1 = Time.Create(2011, 10, 31)
local time2 = Time.Create()
Time.Copy(time2, time1)
```

See Also

- Time.Create

Time.Create

Brief

Creates a new Time object.

Definition

```
Time Time.Create()
Time Time.Create(number year, number month, number day, number hours = 0, number minutes = 0, number
seconds = 0)
Time Time.Create(Time t)
```

Arguments

year - Year to set.month - Month to set.day - Day to set.hours - Hours to set (24 hour time).minutes - Minutes to set.seconds - Seconds to set.t - Time to copy.

Return Values

A new Time instance.

Description

Creates a new Time object. By default it is set to 00:00 1/1/1. When defining a specific date/time, invalid date/times will throw a Lua error.

Examples

```
t = Time.Create()
```

See Also

None

Time.GetAsPosixTimestamp

Brief

Retrieve the stored time value as a POSIX-compatible 64-bit timestamp.

Definition

```
Time.GetAsPosixTimestamp(Time t, BigInt<64> timestamp)
```

Arguments

t - Time object to convert.timestamp - A 64-bit BigInt value to receive the timestamp.

Return Values

Description

Retrieve the time value converted to a POSIX-compatible 64-bit timestamp (time_t).

Examples

```
local timestamp = BigInt.Create(64)
local t = Time.Create()
t:SetFromServer(t)
t:GetAsPosixTimestamp(timestamp)
```

See Also

- Time.SetFromPosixTimestamp

Time.GetDate

Brief

Returns a string representing the date.

Definition

```
string Time.GetDate( Time time )
```

Arguments

time - The time object to query.

Return Values

A string representing the date.

Description

Returns a string representing the date following the DDMMYYYY, YYYYMMDD, or MMDDYYYY format specified in the XMB.

Examples

```
t = Time.Create()
Time.SetFromLocal( t )
print( Time.GetTime() )
```

See Also

None

Time.GetDayOfMonth

Brief

Returns the day of the month value.

Definition

```
number Time.GetDayOfMonth( Time time )
```

Arguments

time - The time object to query.

Return Values

The day of the month value, ranging from 1 to 31.

Description

Returns the day of the month value.

Examples

```
t = Time.Create()
Time.SetFromServer( t )
print( 'The server day of the month is ' .. Time.GetDayOfMonth( t ) .. '.' )
```

See Also

None

Time.GetDayOfWeek

Brief

Returns the day of the week value.

Definition

```
number Time.GetDayOfWeek( Time time )
```

Arguments

time - The time object to query.

Return Values

The day of the week value, ranging from 1 (Sunday) to 7 (Saturday).

Description

Returns the day of the week value.

Examples

```
t = Time.Create()
Time.SetFromServer( t )
print( 'The server day of the week is ' .. Time.GetDayOfWeek( t ) .. '.' )
```

See Also

None

Time.GetDifference

Brief

Compute the difference between two Time values in years, months, days, hours, minutes and seconds.

Definition

```
number, number, number, number, number, number Time.GetDifference(Time t1, Time t2)
```

Arguments

t1 - The first time value.t2 - The second time value.

Return Values

The difference from t1 to t2 in terms of years, months, days, hours, minutes and seconds, in that order.

Description

Computes the offset of t2 relative to t1 (i.e. the difference will be negative if t2 is earlier than t1, otherwise will be positive). This function is more efficient than using the other GetDifferenceIn* functions when requiring more than one unit. However, the individual functions are recommended when only a single unit is needed (e.g. days, weeks, etc).

Examples

```
local now = Time.Create()
now:SetFromServer()
local y, m, d, hh, mm, ss = Time.GetDifference(lastSpaceVisit, now)
print(string.format("You were last here %d years, %d months, %d days, %d hours, %d minutes and %d
seconds ago",
    y, m, d, hh, mm, ss))
```

See Also

- Time.GetDifferenceInSeconds
- Time.GetDifferenceInMinutes
- Time.GetDifferenceInHours
- Time.GetDifferenceInMonths
- Time.GetDifferenceInYears

Time.GetDifferenceInDays

Brief

Get the difference between two Time values in days.

Definition

```
number Time.GetDifferenceInDays(Time t1, Time t2)
```

Arguments

t1 - The first time value.t2 - The second time value.

Return Values

The number of whole days between t1 and t2.

Description

Computes the offset of t2 from t1 in days (i.e. the difference will be negative if t2 is earlier than t1, or positive otherwise). If the difference overflows the size of a Lua number, the maximum representable Lua integer will be returned.

Examples

```
local t = Time.Create()
t:SetFromServer()
-- if last visit to the space was over 5 days ago
if (Time.GetDifferenceInDays(lastVisitTime, t) >= 5) then
    print("What kept you?")
end
```

See Also

- Time.GetDifferenceInSeconds
- Time.GetDifferenceInMinutes
- Time.GetDifferenceInHours
- Time.GetDifferenceInYears
- Time.GetDifference

Time.GetDifferenceInHours

Brief

Get the difference between two Time values in hours.

Definition

```
number Time.GetDifferenceInHours(Time t1, Time t2)
```

Arguments

t1 - The first time value.t2 - The second time value.

Return Values

The number of whole hours between t1 and t2.

Description

Computes the offset of t2 from t1 in hours (i.e. the difference will be negative if t2 is earlier than t1, or positive otherwise). If the difference overflows the size of a Lua number, the maximum representable Lua integer will be returned.

Examples

```
local t = Time.Create()
t:SetFromServer()
-- if last server update was over 2 hours ago, upload again
if (Time.GetDifferenceInHours(lastServerUpdate, t) >= 2) then
    UploadToServer()
end
```

See Also

- Time.GetDifferenceInSeconds
- Time.GetDifferenceInMinutes
- Time.GetDifferenceInDays
- Time.GetDifferenceInYears
- Time.GetDifference

Time.GetDifferenceInMinutes

Brief

Get the difference between two Time values in minutes.

Definition

```
number Time.GetDifferenceInMinutes(Time t1, Time t2)
```

Arguments

t1 - The first time value.t2 - The second time value.

Return Values

The number of whole minutes between t1 and t2.

Description

Computes the offset of t2 from t1 in minutes (i.e. the difference will be negative if t2 is earlier than t1, or positive otherwise). If the difference overflows the size of a Lua number, the maximum representable Lua integer will be returned.

Examples

```
local t = Time.Create()
t:SetFromServer()
-- if last server update was over 30 minutes ago, upload again
if (Time.GetDifferenceInMinutes(lastServerUpdate, t) >= 30) then
    UploadToServer()
end
```

See Also

- Time.GetDifferenceInSeconds
- Time.GetDifferenceInHours
- Time.GetDifferenceInDays
- Time.GetDifferenceInYears
- Time.GetDifference

Time.GetDifferenceInMonths

Brief

Get the difference between two Time values in months.

Definition

```
number Time.GetDifferenceInMonths(Time t1, Time t2)
```

Arguments

t1 - The first time value.t2 - The second time value.

Return Values

The number of whole months between t1 and t2.

Description

Computes the offset of t2 from t1 in months (i.e. the difference will be negative if t2 is earlier than t1, or positive otherwise). If the difference overflows the size of a Lua number, the maximum representable Lua integer will be returned.

Examples

```
local t = Time.Create()
t:SetFromServer()
-- if last visit to the space was over 3 months ago
if (Time.GetDifferenceInMonths(lastVisitTime, t) >= 3) then
    print("What kept you?")
end
```

See Also

- Time.GetDifferenceInSeconds
- Time.GetDifferenceInMinutes

- Time.GetDifferenceInHours
- Time.GetDifferenceInYears
- Time.GetDifference

Time.GetDifferenceInSeconds

Brief

Get the difference between two Time values in seconds.

Definition

```
number Time.GetDifferenceInSeconds(Time t1, Time t2)
```

Arguments

t1 - The first time value.t2 - The second time value.

Return Values

The number of whole seconds between t1 and t2.

Description

Computes the offset of t2 from t1 in seconds (i.e. the difference will be negative if t2 is earlier than t1, or positive otherwise). If the difference overflows the size of a Lua number, the maximum representable Lua integer will be returned.

Examples

```
local t = Time.Create()
t:SetFromServer()
-- if last server update was over 2.5 minutes ago, upload again
if (Time.GetDifferenceInSeconds(lastServerUpdate, t) >= 150) then
    UploadToServer()
end
```

See Also

- Time.GetDifferenceInMinutes
- Time.GetDifferenceInHours
- Time.GetDifferenceInDays
- Time.GetDifferenceInYears
- Time.GetDifference

Time.GetDifferenceInWeeks

Brief

Get the difference between two Time values in weeks.

Definition

```
number Time.GetDifferenceInWeeks(Time t1, Time t2)
```

Arguments

t1 - The first time value.t2 - The second time value.

Return Values

The number of whole weeks between t1 and t2.

Description

Computes the offset of t2 from t1 in weeks (i.e. the difference will be negative if t2 is earlier than t1, or positive otherwise). If the difference overflows the size of a Lua number, the maximum representable Lua integer will be returned.

Examples

```
local t = Time.Create()
t:SetFromServer()
-- if last visit to the space was over 12 weeks ago
if (Time.GetDifferenceInWeeks(lastVisitTime, t) >= 12) then
    print("What kept you?")
end
```

See Also

- Time.GetDifferenceInSeconds
- Time.GetDifferenceInMinutes
- Time.GetDifferenceInHours
- Time.GetDifferenceInYears
- Time.GetDifference

Time.GetDifferenceInYears

Brief

Get the difference between two Time values in years.

Definition

```
number Time.GetDifferenceInYears(Time t1, Time t2)
```

Arguments

t1 - The first time value.t2 - The second time value.

Return Values

The number of whole years between t1 and t2.

Description

Computes the offset of t2 from t1 in years (i.e. the difference will be negative if t2 is earlier than t1, or positive otherwise). If the difference overflows the size of a Lua number, the maximum representable Lua integer will be returned.

Examples

```
local t = Time.Create()
t:SetFromServer()
-- if last visit to the space was over 2 years ago
if (Time.GetDifferenceInYears(lastVisitTime, t) >= 2) then
    print("What kept you?")
end
```

See Also

- Time.GetDifferenceInSeconds
- Time.GetDifferenceInMinutes
- Time.GetDifferenceInHours
- Time.GetDifferenceInMonths
- Time.GetDifference

Time.GetHour

Brief

Returns the hour value.

Definition

```
number Time.GetHour( Time time )
```

Arguments

time - The time object to query.

Return Values

The hour value, ranging from 0 to 23.

Description

Returns the hour value.

Examples

```
t = Time.Create()
Time.SetFromServer( t )
print( 'The server hour is ' .. Time.GetHour( t ) .. '.' )
```

See Also

None

Time.GetMinute

Brief

Returns the minute value.

Definition

```
number Time.GetMinute( Time time )
```

Arguments

time - The time object to query.

Return Values

The minute value, ranging from 0 to 59.

Description

Returns the minute value.

Examples

```
t = Time.Create()
Time.SetFromServer( t )
print( 'The server minute is ' .. Time.GetMinute( t ) .. '.' )
```

See Also

None

Time.GetMonth

Brief

Returns the month value.

Definition

```
number Time.GetMonth( Time time )
```

Arguments

time - The time object to query.

Return Values

The month value, ranging from 1 to 12.

Description

Returns the month value.

Examples

```
t = Time.Create()
Time.SetFromServer( t )
print( 'The server month is ' .. Time.GetMonth( t ) .. '.' )
```

See Also

None

Time.GetSecond

Brief

Returns the second value.

Definition

```
number Time.GetSecond( Time time )
```

Arguments

time - The time object to query.

Return Values

The second value, ranging from 0 to 59.

Description

Returns the second value.

Examples

```
t = Time.Create()
Time.SetFromServer( t )
print( 'The server second is ' .. Time.GetSecond( t ) .. '.' )
```

See Also

None

Time.GetTime

Brief

Returns a string representing the time.

Definition

```
string Time.GetTime( Time time )
```

Arguments

time - The time object to query.

Return Values

A string representing the time.

Description

Returns a string representing the time following the 12h or 24h format specified in the XMB.

Examples

```
t = Time.Create()
Time.SetFromLocal( t )
print( Time.GetTime() )
```

See Also

None

Time.GetYear

Brief

Returns the year value.

Definition

```
number Time.GetYear( Time time )
```

Arguments

time - The time object to query.

Return Values

The year value, ranging from 0 to 9999.

Description

Returns the year value.

Examples

```
t = Time.Create()
Time.SetFromServer( t )
print( 'The server year is ' .. Time.GetYear( t ) .. '.' )
```

See Also

None

Time.IsValid

Brief

Check if a stored date/time value represents a valid date and time.

Definition

```
boolean Time.IsValid(Time time)
```

Arguments

time - The Time object to check.

Return Values

True if the date and time contain valid values, or false if not.

Description

Since 1.70 the Time object will no longer internally enforce that the stored value is always valid. The validity will only be checked if attempting to perform arithmetic (Add* or GetDifference* functions) or convert the time between time zones or formats. If you wish to check a time for validity without generating a Lua error this function can be used, and will check that the date/time value represents a valid date and time, taking account of leap years where appropriate.

Examples

```
if (not myTime:IsValid()) then
    myTime:SetFromServer()  -- revert to a valid time
end
```

See Also

None

Time.SetDayOfMonth

Brief

Sets the day of the month value.

Definition

```
Time.SetDayOfMonth( Time time, number value )
```

Arguments

time - The time object to update.value - The day of the month value, ranging from 1 to 31.

Return Values

None

Description

Sets the day of the month value.

Examples

```
t = Time.Create()
Time.SetDayOfMonth( t, 29 )
print( 'The server day of the month is ' .. Time.GetDayOfMonth( t ) .. '.' )
```

See Also

None

Time.SetFromLocal

Brief

Sets the time object to the current local time of the PlayStation(R)3.

Definition

```
boolean Time.SetFromLocal( Time time )
```

Arguments

time - The time object to update.

Return Values

true if successful and false otherwise.

Description

Sets the time object to the current local time of the PlayStation(R)3. This is the time the user has set their PlayStation(R)3 to.

Examples

```
t = Time.Create()
Time.SetFromLocal( t )
print( 'The time is ' .. Time.GetTime( t ) .. '.' )
```

See Also

None

Time.SetFromPosixTimestamp

Brief

Convert a POSIX-compatible 64-bit timestamp into a Time value.

Definition

```
Time.SetFromPosixTimestamp(Time t, BigInt<64> timestamp)
```

Arguments

t - Time object to modify.timestamp - A 64-bit BigInt value containing the timestamp.

Return Values

Description

Converts a POSIX-compatible 64-bit timestamp (time_t) into a Time value.

Examples

```
local timestamp = BigInt.Create(64)
ReadTimestampFromExternalSource(timestamp)
local t = Time.Create()
t:SetFromPosixTimestamp(timestamp)
```

See Also

- Time.GetAsPosixTimestamp

Time.SetFromServer

Brief

Sets the time object to the current server time.

Definition

```
boolean Time.SetFromServer( Time time )
```

Arguments

time - The time object to update.

Return Values

true if successful and false otherwise.

Description

Sets the time object to the current server time. The time object is not automatically updated with the server time so you must call it every time you want to refresh it. The server time is independent of the PlayStation(R)3 region or local time settings and always returns UTC (Coordinated Universal Time). Note that the Home client does not need to be run in online mode for this function to work. However, the target machine must be signed in to the PSN.

Examples

```
t = Time.Create()
Time.SetFromServer( t )
print( 'The time is ' .. Time.GetTime( t ) .. '.' )
```

See Also

None

Time.SetHour

Brief

Sets the hour value.

Definition

```
Time.SetHour( Time time, number value )
```

Arguments

time - The time object to update.value - The hour value, ranging from 0 to 23.

Return Values

None

Description

Sets the hour value.

Examples

```
t = Time.Create()
Time.SetHour( t, 12 )
print( 'The server hour is ' .. Time.GetHour( t ) .. '.' )
```

See Also

None

Time.SetMinute

Brief

Sets the minute value.

Definition

```
Time.SetMinute( Time time, number value )
```

Arguments

time - The time object to update.value - The minute value, ranging from 0 to 59.

Return Values

None

Description

Sets the minute value.

Examples

```
t = Time.Create()
Time.SetMinute( t, 55 )
print( 'The server minute is ' .. Time.GetMinute( t ) .. '.' )
```

See Also

None

Time.SetMonth

Brief

Sets the month value.

Definition

```
Time.SetMonth( Time time, number value )
```

Arguments

time - The time object to update.value - The month value, ranging from 1 to 12.

Return Values

None

Description

Sets the month value.

Examples

```
t = Time.Create()
Time.SetMonth( t, 12 )
print( 'The server month is ' .. Time.GetMonth( t ) .. '.' )
```

See Also

None

Time.SetSecond

Brief

Sets the second value.

Definition

```
Time.SetSecond( Time time, number value )
```

Arguments

time - The time object to update.value - The second value, ranging from 0 to 59.

Return Values

None

Description

Sets the second value.

Examples

```
t = Time.Create()
Time.SetSecond( t, 12 )
print( 'The server second is ' .. Time.GetSecond( t ) .. '.' )
```

See Also

None

Time.SetYear

Brief

Sets the year value.

Definition

```
Time.SetYear( Time time, number value )
```

Arguments

time - The time object to update.value - The year value, ranging from 0 to 9999.

Return Values

None

Description

Sets the year value.

Examples

```
t = Time.Create()
Time.SetYear( t, 1066 )
print( 'The server year is ' .. Time.GetYear( t ) .. '.' )
```

See Also

None

TriggerVolume

TriggerVolume.AddCallback

Brief

Add a callback function to the trigger volume.

Definition

```
TriggerVolume.AddCallback( TriggerVolume triggerVolume, function callback )
```

Arguments

triggerVolume - The trigger volume to update.callback - The callback function to add.

Return Values

None.

Description

Add a callback function to the trigger volume. The callback will be called when an object enters or exits the trigger volume. The callback should be of the form:
Callback(TriggerVolume triggerVolume, number overlapState, number numOverlaps) Valid values for overlapState are:
TriggerVolumeOverlapState.Enter
TriggerVolumeOverlapState.Exit The callback will be called once per-frame, per-overlap-state if there are any overlaps of that state during that frame.

Examples

```
function MyTriggerVolumeCallback1( triggerVolume, state, count )
    if state == TriggerVolumeOverlapState.Enter then
        print( count .. " new overlaps for trigger volume " .. tostring( triggerVolume ) .. " this
frame" )
    elseif state == TriggerVolumeOverlapState.Exit then
        print( count .. " old overlaps for trigger volume " .. tostring( triggerVolume ) .. " this
frame" )
    end
end

function MyTriggerVolumeCallback2( triggerVolume, state, count )
    for i = 1, count do
        local   overlapType = TriggerVolume.GetOverlapType( triggerVolume, i )
        local   overlapData = TriggerVolume.GetOverlapData( triggerVolume, i )
        print( "trigger volume " .. tostring( triggerVolume ) .. " overlap " .. tostring( i ) .. ":
type=" .. tostring( overlapType ) .. ", object=" .. tostring( overlapData.object ) )
    end
end

triggerVolumes = {}
triggerVolumes[1] = Collision.CreateTriggerVolume( "simpleTriggerVolume1", 16,
TriggerVolumeShape.Sphere, 1.0 )
triggerVolumes[2] = Collision.CreateTriggerVolume( "simpleTriggerVolume2", 16, TriggerVolumeShape.Box,
3.0, 2.0, 4.0 )
triggerVolumes[3] = Collision.CreateTriggerVolume( "resourceTriggerVolume1", 16,
"kTriggerVolumeShapeResource1" )
triggerVolumes[4] = Collision.CreateTriggerVolume( "resourceTriggerVolume2", 16, Resource.Find(
"kTriggerVolumeShapeResource2" ) )

TriggerVolume.SetLocalMatrix( triggerVolumes[1], triggerVolumeTransformMatrix[1] )
TriggerVolume.SetLocalMatrix( triggerVolumes[2], TriggerVolume.GetLocalMatrix( triggerVolumes[1] ) )
TriggerVolume.SetPosition( triggerVolumes[3], triggerVolumePosition[1] )
TriggerVolume.SetPosition( triggerVolumes[4], TriggerVolume.GetPosition( triggerVolumes[3] ) )

for i = 1, #triggerVolumes do
    TriggerVolume.AddCallback( triggerVolumes[i], MyTriggerVolumeCallback1 )
    TriggerVolume.AddCallback( triggerVolumes[i], MyTriggerVolumeCallback2 )
    if TriggerVolume.IsLayerOverlapsEnabled( triggerVolumes[i], CollisionLayer.Scene ) then
        TriggerVolume.EnableLayerOverlaps( triggerVolumes[i], CollisionLayer.Scene, false )
    end
    if not TriggerVolume.IsEnabled( triggerVolumes[i] ) then
        TriggerVolume.Enable( triggerVolumes[i], true )
    end
end

TriggerVolume.RemoveCallback( triggerVolumes[3], MyTriggerVolumeCallback2 )
TriggerVolume.RemoveAllCallbacks( triggerVolumes[4] )

local   numOverlaps = TriggerVolume.GetOverlapCount( triggerVolumes[1] )

for i = 1, numOverlaps do
    local   overlapType = TriggerVolume.GetOverlapType( triggerVolumes[1], i )
    local   overlapData = TriggerVolume.GetOverlapData( triggerVolumes[1], i )
    print( "trigger volume " .. tostring( triggerVolumes[1] ) .. " overlap " .. tostring( i ) .. ":
type=" .. tostring( overlapType ) .. ", object=" .. tostring( overlapData.object ) )
end
```

See Also

None

TriggerVolume.Enable

Brief

Enable/disable the trigger volume.

Definition

```
TriggerVolume.Enable( TriggerVolume triggerVolume, boolean enabled = true, Phys3dWorld world = nil )
```

Arguments

triggerVolume - The trigger volume to update.enabled - true to enable the trigger volume, false otherwise.world - The physics world to add the trigger volume to.

Return Values

None.

Description

Enable/disable the trigger volume. A trigger volume must be enabled to detect overlaps. If the world parameter is not specified (or nil), the default physics world will be used.

Examples

```
function MyTriggerVolumeCallback1( triggerVolume, state, count )
    if state == TriggerVolumeOverlapState.Enter then
        print( count .. " new overlaps for trigger volume " .. tostring( triggerVolume ) .. " this
frame" )
    elseif state == TriggerVolumeOverlapState.Exit then
        print( count .. " old overlaps for trigger volume " .. tostring( triggerVolume ) .. " this
frame" )
    end
end

function MyTriggerVolumeCallback2( triggerVolume, state, count )
    for i = 1, count do
        local   overlapType = TriggerVolume.GetOverlapType( triggerVolume, i )
        local   overlapData = TriggerVolume.GetOverlapData( triggerVolume, i )
        print( "trigger volume " .. tostring( triggerVolume ) .. " overlap " .. tostring( i ) .. ":
type=" .. tostring( overlapType ) .. ", object=" .. tostring( overlapData.object ) )
    end
end

triggerVolumes = {}
triggerVolumes[1] = Collision.CreateTriggerVolume( "simpleTriggerVolume1", 16,
TriggerVolumeShape.Sphere, 1.0 )
triggerVolumes[2] = Collision.CreateTriggerVolume( "simpleTriggerVolume2", 16, TriggerVolumeShape.Box,
3.0, 2.0, 4.0 )
triggerVolumes[3] = Collision.CreateTriggerVolume( "resourceTriggerVolume1", 16,
"kTriggerVolumeShapeResource1" )
triggerVolumes[4] = Collision.CreateTriggerVolume( "resourceTriggerVolume2", 16, Resource.Find(
"kTriggerVolumeShapeResource2" ) )

TriggerVolume.SetLocalMatrix( triggerVolumes[1], triggerVolumeTransformMatrix[1] )
TriggerVolume.SetLocalMatrix( triggerVolumes[2], TriggerVolume.GetLocalMatrix( triggerVolumes[1] ) )
TriggerVolume.SetPosition( triggerVolumes[3], triggerVolumePosition[1] )
TriggerVolume.SetPosition( triggerVolumes[4], TriggerVolume.GetPosition( triggerVolumes[3] ) )

for i = 1, #triggerVolumes do
    TriggerVolume.AddCallback( triggerVolumes[i], MyTriggerVolumeCallback1 )
    TriggerVolume.AddCallback( triggerVolumes[i], MyTriggerVolumeCallback2 )
    if TriggerVolume.IsLayerOverlapsEnabled( triggerVolumes[i], CollisionLayer.Scene ) then
        TriggerVolume.EnableLayerOverlaps( triggerVolumes[i], CollisionLayer.Scene, false )
    end
    if not TriggerVolume.IsEnabled( triggerVolumes[i] ) then
        TriggerVolume.Enable( triggerVolumes[i], true )
    end
end

TriggerVolume.RemoveCallback( triggerVolumes[3], MyTriggerVolumeCallback2 )
TriggerVolume.RemoveAllCallbacks( triggerVolumes[4] )

local   numOverlaps = TriggerVolume.GetOverlapCount( triggerVolumes[1] )

for i = 1, numOverlaps do
    local   overlapType = TriggerVolume.GetOverlapType( triggerVolumes[1], i )
    local   overlapData = TriggerVolume.GetOverlapData( triggerVolumes[1], i )
    print( "trigger volume " .. tostring( triggerVolumes[1] ) .. " overlap " .. tostring( i ) .. ":
type=" .. tostring( overlapType ) .. ", object=" .. tostring( overlapData.object ) )
end
```

See Also

None

TriggerVolume.EnableDebugRender

Brief

Enable debug rendering of the trigger volume.

Definition

```
TriggerVolume.EnableDebugRender( TriggerVolume triggerVolume, boolean enable )
```

Arguments

triggerVolume - The trigger volume to update.enable - true to enable, false otherwise.

Return Values

None.

Description

Enable debug rendering of the trigger volume. The trigger volume shape will be rendered:

1. red - enabled, with overlaps
2. green - enabled, no overlaps
3. blue - disabled

Examples

See Also

None

TriggerVolume.EnableLayerOverlaps

Brief

Enable/disable overlap detection between the trigger volume and objects of a given collision layer.

Definition

```
TriggerVolume.EnableLayerOverlaps( TriggerVolume triggerVolume, enum collisionLayer, boolean enabled =
true )
```

Arguments

triggerVolume - The trigger volume to update.collisionLayer - The CollisionLayer enum.enabled - true to enable overlaps, false otherwise.

Return Values

None.

Description

Enable/disable overlap detection between the trigger volume and objects of a given collision layer. Changes will only take effect for new overlaps. If the trigger volume and an object are already overlapping (or very close), disabling their overlap detection via this function will not take effect until they separate. Also, if the trigger volume and an object are overlapping but overlap detection is disabled, then enabling their overlap detection via this function will not take effect until they separate. Therefore, it is advised to set up layer overlap detection before enabling the trigger volume. In other words, call TriggerVolume.EnableLayerOverlaps before TriggerVolume.Enable. The following CollisionLayer enums are supported by trigger volumes: * CollisionLayer.Avatar - just detects the local player, not remote players * CollisionLayer.Entity - detects entities and placed furniture * CollisionLayer.FurniturePlacement - detects furniture during the furniture placement process

Examples

```
function MyTriggerVolumeCallback1( triggerVolume, state, count )
    if state == TriggerVolumeOverlapState.Enter then
        print( count .. " new overlaps for trigger volume " .. tostring( triggerVolume ) .. " this
frame" )
    elseif state == TriggerVolumeOverlapState.Exit then
        print( count .. " old overlaps for trigger volume " .. tostring( triggerVolume ) .. " this
frame" )
    end
end

function MyTriggerVolumeCallback2( triggerVolume, state, count )
    for i = 1, count do
        local   overlapType = TriggerVolume.GetOverlapType( triggerVolume, i )
        local   overlapData = TriggerVolume.GetOverlapData( triggerVolume, i )
        print( "trigger volume " .. tostring( triggerVolume ) .. " overlap " .. tostring( i ) .. ":
type=" .. tostring( overlapType ) .. ", object=" .. tostring( overlapData.object ) )
    end
end

triggerVolumes = {}
triggerVolumes[1] = Collision.CreateTriggerVolume( "simpleTriggerVolume1", 16,
TriggerVolumeShape.Sphere, 1.0 )
triggerVolumes[2] = Collision.CreateTriggerVolume( "simpleTriggerVolume2", 16, TriggerVolumeShape.Box,
3.0, 2.0, 4.0 )
triggerVolumes[3] = Collision.CreateTriggerVolume( "resourceTriggerVolume1", 16,
"kTriggerVolumeShapeResource1" )
triggerVolumes[4] = Collision.CreateTriggerVolume( "resourceTriggerVolume2", 16, Resource.Find(
"kTriggerVolumeShapeResource2" ) )

TriggerVolume.SetLocalMatrix( triggerVolumes[1], triggerVolumeTransformMatrix[1] )
TriggerVolume.SetLocalMatrix( triggerVolumes[2], TriggerVolume.GetLocalMatrix( triggerVolumes[1] ) )
TriggerVolume.SetPosition( triggerVolumes[3], triggerVolumePosition[1] )
TriggerVolume.SetPosition( triggerVolumes[4], TriggerVolume.GetPosition( triggerVolumes[3] ) )

for i = 1, #triggerVolumes do
    TriggerVolume.AddCallback( triggerVolumes[i], MyTriggerVolumeCallback1 )
    TriggerVolume.AddCallback( triggerVolumes[i], MyTriggerVolumeCallback2 )
    if TriggerVolume.IsLayerOverlapsEnabled( triggerVolumes[i], CollisionLayer.Scene ) then
        TriggerVolume.EnableLayerOverlaps( triggerVolumes[i], CollisionLayer.Scene, false )
    end
    if not TriggerVolume.IsEnabled( triggerVolumes[i] ) then
        TriggerVolume.Enable( triggerVolumes[i], true )
    end
end

TriggerVolume.RemoveCallback( triggerVolumes[3], MyTriggerVolumeCallback2 )
TriggerVolume.RemoveAllCallbacks( triggerVolumes[4] )

local   numOverlaps = TriggerVolume.GetOverlapCount( triggerVolumes[1] )

for i = 1, numOverlaps do
    local   overlapType = TriggerVolume.GetOverlapType( triggerVolumes[1], i )
    local   overlapData = TriggerVolume.GetOverlapData( triggerVolumes[1], i )
    print( "trigger volume " .. tostring( triggerVolumes[1] ) .. " overlap " .. tostring( i ) .. ":
type=" .. tostring( overlapType ) .. ", object=" .. tostring( overlapData.object ) )
end
```

See Also

None

TriggerVolume.GetLocalMatrix

Brief

Gets the trigger volume's transform matrix.

Definition

```
TriggerVolume.GetLocalMatrix( TriggerVolume triggerVolume, Matrix44 matrix )
Matrix44 TriggerVolume.GetLocalMatrix( TriggerVolume triggerVolume )
```

Arguments

triggerVolume - The trigger volume to update.matrix - The variable to store the trigger volume's transform matrix in.

Return Values

The transform matrix (or nil).

Description

Gets the trigger volume's transform matrix.

Examples

```
function MyTriggerVolumeCallback1( triggerVolume, state, count )
    if state == TriggerVolumeOverlapState.Enter then
        print( count .. " new overlaps for trigger volume " .. tostring( triggerVolume ) .. " this
frame" )
    elseif state == TriggerVolumeOverlapState.Exit then
        print( count .. " old overlaps for trigger volume " .. tostring( triggerVolume ) .. " this
frame" )
    end
end

function MyTriggerVolumeCallback2( triggerVolume, state, count )
    for i = 1, count do
        local   overlapType = TriggerVolume.GetOverlapType( triggerVolume, i )
        local   overlapData = TriggerVolume.GetOverlapData( triggerVolume, i )
        print( "trigger volume " .. tostring( triggerVolume ) .. " overlap " .. tostring( i ) .. ":
type=" .. tostring( overlapType ) .. ", object=" .. tostring( overlapData.object ) )
    end
end

triggerVolumes = {}
triggerVolumes[1] = Collision.CreateTriggerVolume( "simpleTriggerVolume1", 16,
TriggerVolumeShape.Sphere, 1.0 )
triggerVolumes[2] = Collision.CreateTriggerVolume( "simpleTriggerVolume2", 16, TriggerVolumeShape.Box,
3.0, 2.0, 4.0 )
triggerVolumes[3] = Collision.CreateTriggerVolume( "resourceTriggerVolume1", 16,
"kTriggerVolumeShapeResource1" )
triggerVolumes[4] = Collision.CreateTriggerVolume( "resourceTriggerVolume2", 16, Resource.Find(
"kTriggerVolumeShapeResource2" ) )

TriggerVolume.SetLocalMatrix( triggerVolumes[1], triggerVolumeTransformMatrix[1] )
TriggerVolume.SetLocalMatrix( triggerVolumes[2], TriggerVolume.GetLocalMatrix( triggerVolumes[1] ) )
TriggerVolume.SetPosition( triggerVolumes[3], triggerVolumePosition[1] )
TriggerVolume.SetPosition( triggerVolumes[4], TriggerVolume.GetPosition( triggerVolumes[3] ) )

for i = 1, #triggerVolumes do
    TriggerVolume.AddCallback( triggerVolumes[i], MyTriggerVolumeCallback1 )
    TriggerVolume.AddCallback( triggerVolumes[i], MyTriggerVolumeCallback2 )
    if TriggerVolume.IsLayerOverlapsEnabled( triggerVolumes[i], CollisionLayer.Scene ) then
        TriggerVolume.EnableLayerOverlaps( triggerVolumes[i], CollisionLayer.Scene, false )
    end
    if not TriggerVolume.IsEnabled( triggerVolumes[i] ) then
        TriggerVolume.Enable( triggerVolumes[i], true )
    end
end

TriggerVolume.RemoveCallback( triggerVolumes[3], MyTriggerVolumeCallback2 )
TriggerVolume.RemoveAllCallbacks( triggerVolumes[4] )

local   numOverlaps = TriggerVolume.GetOverlapCount( triggerVolumes[1] )

for i = 1, numOverlaps do
    local   overlapType = TriggerVolume.GetOverlapType( triggerVolumes[1], i )
    local   overlapData = TriggerVolume.GetOverlapData( triggerVolumes[1], i )
    print( "trigger volume " .. tostring( triggerVolumes[1] ) .. " overlap " .. tostring( i ) .. ":
type=" .. tostring( overlapType ) .. ", object=" .. tostring( overlapData.object ) )
end
```

See Also

None

TriggerVolume.GetOverlapCount

Brief

Get the current number of overlaps.

Definition

```
number TriggerVolume.GetOverlapCount( TriggerVolume triggerVolume )
```

Arguments

triggerVolume - The trigger volume to query.

Return Values

The number of overlaps.

Description

Get the current number of overlaps.

Examples

```
function MyTriggerVolumeCallback1( triggerVolume, state, count )
    if state == TriggerVolumeOverlapState.Enter then
        print( count .. " new overlaps for trigger volume " .. tostring( triggerVolume ) .. " this
frame" )
    elseif state == TriggerVolumeOverlapState.Exit then
        print( count .. " old overlaps for trigger volume " .. tostring( triggerVolume ) .. " this
frame" )
    end
end

function MyTriggerVolumeCallback2( triggerVolume, state, count )
    for i = 1, count do
        local   overlapType = TriggerVolume.GetOverlapType( triggerVolume, i )
        local   overlapData = TriggerVolume.GetOverlapData( triggerVolume, i )
        print( "trigger volume " .. tostring( triggerVolume ) .. " overlap " .. tostring( i ) .. ":
type=" .. tostring( overlapType ) .. ", object=" .. tostring( overlapData.object ) )
    end
end

triggerVolumes = {}
triggerVolumes[1] = Collision.CreateTriggerVolume( "simpleTriggerVolume1", 16,
TriggerVolumeShape.Sphere, 1.0 )
triggerVolumes[2] = Collision.CreateTriggerVolume( "simpleTriggerVolume2", 16, TriggerVolumeShape.Box,
3.0, 2.0, 4.0 )
triggerVolumes[3] = Collision.CreateTriggerVolume( "resourceTriggerVolume1", 16,
"kTriggerVolumeShapeResource1" )
triggerVolumes[4] = Collision.CreateTriggerVolume( "resourceTriggerVolume2", 16, Resource.Find(
"kTriggerVolumeShapeResource2" ) )

TriggerVolume.SetLocalMatrix( triggerVolumes[1], triggerVolumeTransformMatrix[1] )
TriggerVolume.SetLocalMatrix( triggerVolumes[2], TriggerVolume.GetLocalMatrix( triggerVolumes[1] ) )
TriggerVolume.SetPosition( triggerVolumes[3], triggerVolumePosition[1] )
TriggerVolume.SetPosition( triggerVolumes[4], TriggerVolume.GetPosition( triggerVolumes[3] ) )

for i = 1, #triggerVolumes do
    TriggerVolume.AddCallback( triggerVolumes[i], MyTriggerVolumeCallback1 )
    TriggerVolume.AddCallback( triggerVolumes[i], MyTriggerVolumeCallback2 )
    if TriggerVolume.IsLayerOverlapsEnabled( triggerVolumes[i], CollisionLayer.Scene ) then
        TriggerVolume.EnableLayerOverlaps( triggerVolumes[i], CollisionLayer.Scene, false )
    end
    if not TriggerVolume.IsEnabled( triggerVolumes[i] ) then
        TriggerVolume.Enable( triggerVolumes[i], true )
    end
end

TriggerVolume.RemoveCallback( triggerVolumes[3], MyTriggerVolumeCallback2 )
TriggerVolume.RemoveAllCallbacks( triggerVolumes[4] )

local   numOverlaps = TriggerVolume.GetOverlapCount( triggerVolumes[1] )

for i = 1, numOverlaps do
    local   overlapType = TriggerVolume.GetOverlapType( triggerVolumes[1], i )
    local   overlapData = TriggerVolume.GetOverlapData( triggerVolumes[1], i )
    print( "trigger volume " .. tostring( triggerVolumes[1] ) .. " overlap " .. tostring( i ) .. ":
type=" .. tostring( overlapType ) .. ", object=" .. tostring( overlapData.object ) )
end
```

See Also

None

TriggerVolume.GetOverlapData

Brief

Get the data relating to a trigger volume overlap.

Definition

```
table TriggerVolume.GetOverlapData( TriggerVolume triggerVolume, number index )
```

Arguments

triggerVolume - The trigger volume to query.index - The index of the overlap.

Return Values

A table of data or nil

Description

Get the data relating to a trigger volume overlap. Only to be called from a trigger volume callback. The table will contain members:
type - a TriggerVolumeOverlapType value
object - the type of this data depends on the type of the overlap Valid TriggerVolumeOverlapType values are:
TriggerVolumeOverlapType.Entity
TriggerVolumeOverlapType.Person
TriggerVolumeOverlapType.Furniture

Examples

```
function MyTriggerVolumeCallback1( triggerVolume, state, count )
    if state == TriggerVolumeOverlapState.Enter then
        print( count .. " new overlaps for trigger volume " .. tostring( triggerVolume ) .. " this
frame" )
    elseif state == TriggerVolumeOverlapState.Exit then
        print( count .. " old overlaps for trigger volume " .. tostring( triggerVolume ) .. " this
frame" )
    end
end

function MyTriggerVolumeCallback2( triggerVolume, state, count )
    for i = 1, count do
        local   overlapType = TriggerVolume.GetOverlapType( triggerVolume, i )
        local   overlapData = TriggerVolume.GetOverlapData( triggerVolume, i )
        print( "trigger volume " .. tostring( triggerVolume ) .. " overlap " .. tostring( i ) .. ":
type=" .. tostring( overlapType ) .. ", object=" .. tostring( overlapData.object ) )
    end
end

triggerVolumes = {}
triggerVolumes[1] = Collision.CreateTriggerVolume( "simpleTriggerVolume1", 16,
TriggerVolumeShape.Sphere, 1.0 )
triggerVolumes[2] = Collision.CreateTriggerVolume( "simpleTriggerVolume2", 16, TriggerVolumeShape.Box,
3.0, 2.0, 4.0 )
triggerVolumes[3] = Collision.CreateTriggerVolume( "resourceTriggerVolume1", 16,
"kTriggerVolumeShapeResource1" )
triggerVolumes[4] = Collision.CreateTriggerVolume( "resourceTriggerVolume2", 16, Resource.Find(
"kTriggerVolumeShapeResource2" ) )

TriggerVolume.SetLocalMatrix( triggerVolumes[1], triggerVolumeTransformMatrix[1] )
TriggerVolume.SetLocalMatrix( triggerVolumes[2], TriggerVolume.GetLocalMatrix( triggerVolumes[1] ) )
TriggerVolume.SetPosition( triggerVolumes[3], triggerVolumePosition[1] )
TriggerVolume.SetPosition( triggerVolumes[4], TriggerVolume.GetPosition( triggerVolumes[3] ) )

for i = 1, #triggerVolumes do
    TriggerVolume.AddCallback( triggerVolumes[i], MyTriggerVolumeCallback1 )
    TriggerVolume.AddCallback( triggerVolumes[i], MyTriggerVolumeCallback2 )
    if TriggerVolume.IsLayerOverlapsEnabled( triggerVolumes[i], CollisionLayer.Scene ) then
        TriggerVolume.EnableLayerOverlaps( triggerVolumes[i], CollisionLayer.Scene, false )
    end
    if not TriggerVolume.IsEnabled( triggerVolumes[i] ) then
        TriggerVolume.Enable( triggerVolumes[i], true )
    end
end

TriggerVolume.RemoveCallback( triggerVolumes[3], MyTriggerVolumeCallback2 )
TriggerVolume.RemoveAllCallbacks( triggerVolumes[4] )

local   numOverlaps = TriggerVolume.GetOverlapCount( triggerVolumes[1] )

for i = 1, numOverlaps do
    local   overlapType = TriggerVolume.GetOverlapType( triggerVolumes[1], i )
    local   overlapData = TriggerVolume.GetOverlapData( triggerVolumes[1], i )
    print( "trigger volume " .. tostring( triggerVolumes[1] ) .. " overlap " .. tostring( i ) .. ":
type=" .. tostring( overlapType ) .. ", object=" .. tostring( overlapData.object ) )
end
```

See Also

None

TriggerVolume.GetOverlapType

Brief

Get the type of a trigger volume overlap.

Definition

```
enum TriggerVolume.GetOverlapType( TriggerVolume triggerVolume, number index )
```

Arguments

triggerVolume - The trigger volume to query.index - The index of the overlap.

Return Values

A TriggerVolumeOverlapType enum or nil.

Description

Get the type of a trigger volume overlap. Only to be called from a trigger volume callback. Valid TriggerVolumeOverlapType values are:
TriggerVolumeOverlapType.Entity
TriggerVolumeOverlapType.Person
TriggerVolumeOverlapType.Furniture

Examples

```
function MyTriggerVolumeCallback1( triggerVolume, state, count )
    if state == TriggerVolumeOverlapState.Enter then
        print( count .. " new overlaps for trigger volume " .. tostring( triggerVolume ) .. " this
frame" )
    elseif state == TriggerVolumeOverlapState.Exit then
        print( count .. " old overlaps for trigger volume " .. tostring( triggerVolume ) .. " this
frame" )
    end
end

function MyTriggerVolumeCallback2( triggerVolume, state, count )
    for i = 1, count do
        local   overlapType = TriggerVolume.GetOverlapType( triggerVolume, i )
        local   overlapData = TriggerVolume.GetOverlapData( triggerVolume, i )
        print( "trigger volume " .. tostring( triggerVolume ) .. " overlap " .. tostring( i ) .. ":
type=" .. tostring( overlapType ) .. ", object=" .. tostring( overlapData.object ) )
    end
end

triggerVolumes = {}
triggerVolumes[1] = Collision.CreateTriggerVolume( "simpleTriggerVolume1", 16,
TriggerVolumeShape.Sphere, 1.0 )
triggerVolumes[2] = Collision.CreateTriggerVolume( "simpleTriggerVolume2", 16, TriggerVolumeShape.Box,
3.0, 2.0, 4.0 )
triggerVolumes[3] = Collision.CreateTriggerVolume( "resourceTriggerVolume1", 16,
"kTriggerVolumeShapeResource1" )
triggerVolumes[4] = Collision.CreateTriggerVolume( "resourceTriggerVolume2", 16, Resource.Find(
"kTriggerVolumeShapeResource2" ) )

TriggerVolume.SetLocalMatrix( triggerVolumes[1], triggerVolumeTransformMatrix[1] )
TriggerVolume.SetLocalMatrix( triggerVolumes[2], TriggerVolume.GetLocalMatrix( triggerVolumes[1] ) )
TriggerVolume.SetPosition( triggerVolumes[3], triggerVolumePosition[1] )
TriggerVolume.SetPosition( triggerVolumes[4], TriggerVolume.GetPosition( triggerVolumes[3] ) )

for i = 1, #triggerVolumes do
    TriggerVolume.AddCallback( triggerVolumes[i], MyTriggerVolumeCallback1 )
    TriggerVolume.AddCallback( triggerVolumes[i], MyTriggerVolumeCallback2 )
    if TriggerVolume.IsLayerOverlapsEnabled( triggerVolumes[i], CollisionLayer.Scene ) then
        TriggerVolume.EnableLayerOverlaps( triggerVolumes[i], CollisionLayer.Scene, false )
    end
    if not TriggerVolume.IsEnabled( triggerVolumes[i] ) then
        TriggerVolume.Enable( triggerVolumes[i], true )
    end
end

TriggerVolume.RemoveCallback( triggerVolumes[3], MyTriggerVolumeCallback2 )
TriggerVolume.RemoveAllCallbacks( triggerVolumes[4] )

local   numOverlaps = TriggerVolume.GetOverlapCount( triggerVolumes[1] )

for i = 1, numOverlaps do
    local   overlapType = TriggerVolume.GetOverlapType( triggerVolumes[1], i )
    local   overlapData = TriggerVolume.GetOverlapData( triggerVolumes[1], i )
    print( "trigger volume " .. tostring( triggerVolumes[1] ) .. " overlap " .. tostring( i ) .. ":
type=" .. tostring( overlapType ) .. ", object=" .. tostring( overlapData.object ) )
end
```

See Also

None

TriggerVolume.GetPosition

Brief

Gets the trigger volume's position.

Definition

```
TriggerVolume.GetPosition( TriggerVolume triggerVolume, Vector4 position )
Vector4 TriggerVolume.GetPosition( TriggerVolume triggerVolume )
```

Arguments

triggerVolume - The trigger volume to update.position - The variable to store the trigger volume's position in.

Return Values

The position vector (or nil).

Description

Gets the trigger volume's position.

Examples

```
function MyTriggerVolumeCallback1( triggerVolume, state, count )
    if state == TriggerVolumeOverlapState.Enter then
        print( count .. " new overlaps for trigger volume " .. tostring( triggerVolume ) .. " this
frame" )
    elseif state == TriggerVolumeOverlapState.Exit then
        print( count .. " old overlaps for trigger volume " .. tostring( triggerVolume ) .. " this
frame" )
    end
end

function MyTriggerVolumeCallback2( triggerVolume, state, count )
    for i = 1, count do
        local   overlapType = TriggerVolume.GetOverlapType( triggerVolume, i )
        local   overlapData = TriggerVolume.GetOverlapData( triggerVolume, i )
        print( "trigger volume " .. tostring( triggerVolume ) .. " overlap " .. tostring( i ) .. ":
type=" .. tostring( overlapType ) .. ", object=" .. tostring( overlapData.object ) )
    end
end

triggerVolumes = {}
triggerVolumes[1] = Collision.CreateTriggerVolume( "simpleTriggerVolume1", 16,
TriggerVolumeShape.Sphere, 1.0 )
triggerVolumes[2] = Collision.CreateTriggerVolume( "simpleTriggerVolume2", 16, TriggerVolumeShape.Box,
3.0, 2.0, 4.0 )
triggerVolumes[3] = Collision.CreateTriggerVolume( "resourceTriggerVolume1", 16,
"kTriggerVolumeShapeResource1" )
triggerVolumes[4] = Collision.CreateTriggerVolume( "resourceTriggerVolume2", 16, Resource.Find(
"kTriggerVolumeShapeResource2" ) )

TriggerVolume.SetLocalMatrix( triggerVolumes[1], triggerVolumeTransformMatrix[1] )
TriggerVolume.SetLocalMatrix( triggerVolumes[2], TriggerVolume.GetLocalMatrix( triggerVolumes[1] ) )
TriggerVolume.SetPosition( triggerVolumes[3], triggerVolumePosition[1] )
TriggerVolume.SetPosition( triggerVolumes[4], TriggerVolume.GetPosition( triggerVolumes[3] ) )

for i = 1, #triggerVolumes do
    TriggerVolume.AddCallback( triggerVolumes[i], MyTriggerVolumeCallback1 )
    TriggerVolume.AddCallback( triggerVolumes[i], MyTriggerVolumeCallback2 )
    if TriggerVolume.IsLayerOverlapsEnabled( triggerVolumes[i], CollisionLayer.Scene ) then
        TriggerVolume.EnableLayerOverlaps( triggerVolumes[i], CollisionLayer.Scene, false )
    end
    if not TriggerVolume.IsEnabled( triggerVolumes[i] ) then
        TriggerVolume.Enable( triggerVolumes[i], true )
    end
end

TriggerVolume.RemoveCallback( triggerVolumes[3], MyTriggerVolumeCallback2 )
TriggerVolume.RemoveAllCallbacks( triggerVolumes[4] )

local   numOverlaps = TriggerVolume.GetOverlapCount( triggerVolumes[1] )

for i = 1, numOverlaps do
    local   overlapType = TriggerVolume.GetOverlapType( triggerVolumes[1], i )
    local   overlapData = TriggerVolume.GetOverlapData( triggerVolumes[1], i )
    print( "trigger volume " .. tostring( triggerVolumes[1] ) .. " overlap " .. tostring( i ) .. ":
type=" .. tostring( overlapType ) .. ", object=" .. tostring( overlapData.object ) )
end
```

See Also

None

TriggerVolume.IsEnabled

Brief

Query whether or not the trigger volume is enabled.

Definition

```
boolean TriggerVolume.IsEnabled( TriggerVolume triggerVolume, Phys3dWorld world = nil )
```

Arguments

triggerVolume - The trigger volume to query.world - The physics world.

Return Values

true if enabled, false otherwise.

Description

Query whether or not the trigger volume is enabled. A trigger volume must be enabled to detect overlaps. If the world parameter is specified (and not or nil), the function will only return true if the trigger volume is enabled and added to the specified physics world.

Examples

```
function MyTriggerVolumeCallback1( triggerVolume, state, count )
    if state == TriggerVolumeOverlapState.Enter then
        print( count .. " new overlaps for trigger volume " .. tostring( triggerVolume ) .. " this
frame" )
    elseif state == TriggerVolumeOverlapState.Exit then
        print( count .. " old overlaps for trigger volume " .. tostring( triggerVolume ) .. " this
frame" )
    end
end

function MyTriggerVolumeCallback2( triggerVolume, state, count )
    for i = 1, count do
        local   overlapType = TriggerVolume.GetOverlapType( triggerVolume, i )
        local   overlapData = TriggerVolume.GetOverlapData( triggerVolume, i )
        print( "trigger volume " .. tostring( triggerVolume ) .. " overlap " .. tostring( i ) .. ":
type=" .. tostring( overlapType ) .. ", object=" .. tostring( overlapData.object ) )
    end
end

triggerVolumes = {}
triggerVolumes[1] = Collision.CreateTriggerVolume( "simpleTriggerVolume1", 16,
TriggerVolumeShape.Sphere, 1.0 )
triggerVolumes[2] = Collision.CreateTriggerVolume( "simpleTriggerVolume2", 16, TriggerVolumeShape.Box,
3.0, 2.0, 4.0 )
triggerVolumes[3] = Collision.CreateTriggerVolume( "resourceTriggerVolume1", 16,
"kTriggerVolumeShapeResource1" )
triggerVolumes[4] = Collision.CreateTriggerVolume( "resourceTriggerVolume2", 16, Resource.Find(
"kTriggerVolumeShapeResource2" ) )

TriggerVolume.SetLocalMatrix( triggerVolumes[1], triggerVolumeTransformMatrix[1] )
TriggerVolume.SetLocalMatrix( triggerVolumes[2], TriggerVolume.GetLocalMatrix( triggerVolumes[1] ) )
TriggerVolume.SetPosition( triggerVolumes[3], triggerVolumePosition[1] )
TriggerVolume.SetPosition( triggerVolumes[4], TriggerVolume.GetPosition( triggerVolumes[3] ) )

for i = 1, #triggerVolumes do
    TriggerVolume.AddCallback( triggerVolumes[i], MyTriggerVolumeCallback1 )
    TriggerVolume.AddCallback( triggerVolumes[i], MyTriggerVolumeCallback2 )
    if TriggerVolume.IsLayerOverlapsEnabled( triggerVolumes[i], CollisionLayer.Scene ) then
        TriggerVolume.EnableLayerOverlaps( triggerVolumes[i], CollisionLayer.Scene, false )
    end
    if not TriggerVolume.IsEnabled( triggerVolumes[i] ) then
        TriggerVolume.Enable( triggerVolumes[i], true )
    end
end

TriggerVolume.RemoveCallback( triggerVolumes[3], MyTriggerVolumeCallback2 )
TriggerVolume.RemoveAllCallbacks( triggerVolumes[4] )

local   numOverlaps = TriggerVolume.GetOverlapCount( triggerVolumes[1] )

for i = 1, numOverlaps do
    local   overlapType = TriggerVolume.GetOverlapType( triggerVolumes[1], i )
    local   overlapData = TriggerVolume.GetOverlapData( triggerVolumes[1], i )
    print( "trigger volume " .. tostring( triggerVolumes[1] ) .. " overlap " .. tostring( i ) .. ":
type=" .. tostring( overlapType ) .. ", object=" .. tostring( overlapData.object ) )
end
```

See Also

None

TriggerVolume.IsLayerOverlapsEnabled

Brief

Query whether overlaps between the trigger volume and objects of a given collision layer are enabled.

Definition

```
boolean TriggerVolume.IsLayerOverlapsEnabled( TriggerVolume triggerVolume, enum collisionLayer )
```

Arguments

triggerVolume - The trigger volume to query.collisionLayer - The CollisionLayer enum.

Return Values

true if enabled, false otherwise.

Description

Query whether overlaps between the trigger volume and objects of a given collision layer are enabled.

Examples

```
function MyTriggerVolumeCallback1( triggerVolume, state, count )
    if state == TriggerVolumeOverlapState.Enter then
        print( count .. " new overlaps for trigger volume " .. tostring( triggerVolume ) .. " this
frame" )
    elseif state == TriggerVolumeOverlapState.Exit then
        print( count .. " old overlaps for trigger volume " .. tostring( triggerVolume ) .. " this
frame" )
    end
end

function MyTriggerVolumeCallback2( triggerVolume, state, count )
    for i = 1, count do
        local   overlapType = TriggerVolume.GetOverlapType( triggerVolume, i )
        local   overlapData = TriggerVolume.GetOverlapData( triggerVolume, i )
        print( "trigger volume " .. tostring( triggerVolume ) .. " overlap " .. tostring( i ) .. ":
type=" .. tostring( overlapType ) .. ", object=" .. tostring( overlapData.object ) )
    end
end

triggerVolumes = {}
triggerVolumes[1] = Collision.CreateTriggerVolume( "simpleTriggerVolume1", 16,
TriggerVolumeShape.Sphere, 1.0 )
triggerVolumes[2] = Collision.CreateTriggerVolume( "simpleTriggerVolume2", 16, TriggerVolumeShape.Box,
3.0, 2.0, 4.0 )
triggerVolumes[3] = Collision.CreateTriggerVolume( "resourceTriggerVolume1", 16,
"kTriggerVolumeShapeResource1" )
triggerVolumes[4] = Collision.CreateTriggerVolume( "resourceTriggerVolume2", 16, Resource.Find(
"kTriggerVolumeShapeResource2" ) )

TriggerVolume.SetLocalMatrix( triggerVolumes[1], triggerVolumeTransformMatrix[1] )
TriggerVolume.SetLocalMatrix( triggerVolumes[2], TriggerVolume.GetLocalMatrix( triggerVolumes[1] ) )
TriggerVolume.SetPosition( triggerVolumes[3], triggerVolumePosition[1] )
TriggerVolume.SetPosition( triggerVolumes[4], TriggerVolume.GetPosition( triggerVolumes[3] ) )

for i = 1, #triggerVolumes do
    TriggerVolume.AddCallback( triggerVolumes[i], MyTriggerVolumeCallback1 )
    TriggerVolume.AddCallback( triggerVolumes[i], MyTriggerVolumeCallback2 )
    if TriggerVolume.IsLayerOverlapsEnabled( triggerVolumes[i], CollisionLayer.Scene ) then
        TriggerVolume.EnableLayerOverlaps( triggerVolumes[i], CollisionLayer.Scene, false )
    end
    if not TriggerVolume.IsEnabled( triggerVolumes[i] ) then
        TriggerVolume.Enable( triggerVolumes[i], true )
    end
end

TriggerVolume.RemoveCallback( triggerVolumes[3], MyTriggerVolumeCallback2 )
TriggerVolume.RemoveAllCallbacks( triggerVolumes[4] )

local   numOverlaps = TriggerVolume.GetOverlapCount( triggerVolumes[1] )

for i = 1, numOverlaps do
    local   overlapType = TriggerVolume.GetOverlapType( triggerVolumes[1], i )
    local   overlapData = TriggerVolume.GetOverlapData( triggerVolumes[1], i )
    print( "trigger volume " .. tostring( triggerVolumes[1] ) .. " overlap " .. tostring( i ) .. ":
type=" .. tostring( overlapType ) .. ", object=" .. tostring( overlapData.object ) )
end
```

See Also

None

TriggerVolume.RemoveAllCallbacks

Brief

Remove all callback functions from the trigger volume.

Definition

```
TriggerVolume.RemoveAllCallbacks( TriggerVolume triggerVolume )
```

Arguments

triggerVolume - The trigger volume to update.

Return Values

None.

Description

Remove all callback functions from the trigger volume. Will only remove callbacks added by this script.

Examples

```
function MyTriggerVolumeCallback1( triggerVolume, state, count )
    if state == TriggerVolumeOverlapState.Enter then
        print( count .. " new overlaps for trigger volume " .. tostring( triggerVolume ) .. " this
frame" )
    elseif state == TriggerVolumeOverlapState.Exit then
        print( count .. " old overlaps for trigger volume " .. tostring( triggerVolume ) .. " this
frame" )
    end
end

function MyTriggerVolumeCallback2( triggerVolume, state, count )
    for i = 1, count do
        local   overlapType = TriggerVolume.GetOverlapType( triggerVolume, i )
        local   overlapData = TriggerVolume.GetOverlapData( triggerVolume, i )
        print( "trigger volume " .. tostring( triggerVolume ) .. " overlap " .. tostring( i ) .. ":
type=" .. tostring( overlapType ) .. ", object=" .. tostring( overlapData.object ) )
    end
end

triggerVolumes = {}
triggerVolumes[1] = Collision.CreateTriggerVolume( "simpleTriggerVolume1", 16,
TriggerVolumeShape.Sphere, 1.0 )
triggerVolumes[2] = Collision.CreateTriggerVolume( "simpleTriggerVolume2", 16, TriggerVolumeShape.Box,
3.0, 2.0, 4.0 )
triggerVolumes[3] = Collision.CreateTriggerVolume( "resourceTriggerVolume1", 16,
"kTriggerVolumeShapeResource1" )
triggerVolumes[4] = Collision.CreateTriggerVolume( "resourceTriggerVolume2", 16, Resource.Find(
"kTriggerVolumeShapeResource2" ) )

TriggerVolume.SetLocalMatrix( triggerVolumes[1], triggerVolumeTransformMatrix[1] )
TriggerVolume.SetLocalMatrix( triggerVolumes[2], TriggerVolume.GetLocalMatrix( triggerVolumes[1] ) )
TriggerVolume.SetPosition( triggerVolumes[3], triggerVolumePosition[1] )
TriggerVolume.SetPosition( triggerVolumes[4], TriggerVolume.GetPosition( triggerVolumes[3] ) )

for i = 1, #triggerVolumes do
    TriggerVolume.AddCallback( triggerVolumes[i], MyTriggerVolumeCallback1 )
    TriggerVolume.AddCallback( triggerVolumes[i], MyTriggerVolumeCallback2 )
    if TriggerVolume.IsLayerOverlapsEnabled( triggerVolumes[i], CollisionLayer.Scene ) then
        TriggerVolume.EnableLayerOverlaps( triggerVolumes[i], CollisionLayer.Scene, false )
    end
    if not TriggerVolume.IsEnabled( triggerVolumes[i] ) then
        TriggerVolume.Enable( triggerVolumes[i], true )
    end
end

TriggerVolume.RemoveCallback( triggerVolumes[3], MyTriggerVolumeCallback2 )
TriggerVolume.RemoveAllCallbacks( triggerVolumes[4] )

local   numOverlaps = TriggerVolume.GetOverlapCount( triggerVolumes[1] )

for i = 1, numOverlaps do
    local   overlapType = TriggerVolume.GetOverlapType( triggerVolumes[1], i )
    local   overlapData = TriggerVolume.GetOverlapData( triggerVolumes[1], i )
    print( "trigger volume " .. tostring( triggerVolumes[1] ) .. " overlap " .. tostring( i ) .. ":
type=" .. tostring( overlapType ) .. ", object=" .. tostring( overlapData.object ) )
end
```

See Also

None

TriggerVolume.RemoveCallback

Brief

Remove a callback function from the trigger volume.

Definition

```
TriggerVolume.RemoveCallback( TriggerVolume triggerVolume, function callback )
```

Arguments

triggerVolume - The trigger volume to update.callback - The callback function to remove.

Return Values

None.

Description

Remove a callback function from the trigger volume.

Examples

```
function MyTriggerVolumeCallback1( triggerVolume, state, count )
    if state == TriggerVolumeOverlapState.Enter then
        print( count .. " new overlaps for trigger volume " .. tostring( triggerVolume ) .. " this
frame" )
    elseif state == TriggerVolumeOverlapState.Exit then
        print( count .. " old overlaps for trigger volume " .. tostring( triggerVolume ) .. " this
frame" )
    end
end

function MyTriggerVolumeCallback2( triggerVolume, state, count )
    for i = 1, count do
        local   overlapType = TriggerVolume.GetOverlapType( triggerVolume, i )
        local   overlapData = TriggerVolume.GetOverlapData( triggerVolume, i )
        print( "trigger volume " .. tostring( triggerVolume ) .. " overlap " .. tostring( i ) .. ":
type=" .. tostring( overlapType ) .. ", object=" .. tostring( overlapData.object ) )
    end
end

triggerVolumes = {}
triggerVolumes[1] = Collision.CreateTriggerVolume( "simpleTriggerVolume1", 16,
TriggerVolumeShape.Sphere, 1.0 )
triggerVolumes[2] = Collision.CreateTriggerVolume( "simpleTriggerVolume2", 16, TriggerVolumeShape.Box,
3.0, 2.0, 4.0 )
triggerVolumes[3] = Collision.CreateTriggerVolume( "resourceTriggerVolume1", 16,
"kTriggerVolumeShapeResource1" )
triggerVolumes[4] = Collision.CreateTriggerVolume( "resourceTriggerVolume2", 16, Resource.Find(
"kTriggerVolumeShapeResource2" ) )

TriggerVolume.SetLocalMatrix( triggerVolumes[1], triggerVolumeTransformMatrix[1] )
TriggerVolume.SetLocalMatrix( triggerVolumes[2], TriggerVolume.GetLocalMatrix( triggerVolumes[1] ) )
TriggerVolume.SetPosition( triggerVolumes[3], triggerVolumePosition[1] )
TriggerVolume.SetPosition( triggerVolumes[4], TriggerVolume.GetPosition( triggerVolumes[3] ) )

for i = 1, #triggerVolumes do
    TriggerVolume.AddCallback( triggerVolumes[i], MyTriggerVolumeCallback1 )
    TriggerVolume.AddCallback( triggerVolumes[i], MyTriggerVolumeCallback2 )
    if TriggerVolume.IsLayerOverlapsEnabled( triggerVolumes[i], CollisionLayer.Scene ) then
        TriggerVolume.EnableLayerOverlaps( triggerVolumes[i], CollisionLayer.Scene, false )
    end
    if not TriggerVolume.IsEnabled( triggerVolumes[i] ) then
        TriggerVolume.Enable( triggerVolumes[i], true )
    end
end

TriggerVolume.RemoveCallback( triggerVolumes[3], MyTriggerVolumeCallback2 )
TriggerVolume.RemoveAllCallbacks( triggerVolumes[4] )

local   numOverlaps = TriggerVolume.GetOverlapCount( triggerVolumes[1] )

for i = 1, numOverlaps do
    local   overlapType = TriggerVolume.GetOverlapType( triggerVolumes[1], i )
    local   overlapData = TriggerVolume.GetOverlapData( triggerVolumes[1], i )
    print( "trigger volume " .. tostring( triggerVolumes[1] ) .. " overlap " .. tostring( i ) .. ":
type=" .. tostring( overlapType ) .. ", object=" .. tostring( overlapData.object ) )
end
```

See Also

None

TriggerVolume.SetLocalMatrix

Brief

Sets the trigger volume's local transformation matrix.

Definition

```
TriggerVolume.SetLocalMatrix( TriggerVolume triggerVolume, Matrix44 matrix )
```

Arguments

triggerVolume - The trigger volume to update.matrix - The matrix to set the trigger volume's transform to.

Return Values

None.

Description

Sets the trigger volume's local transformation matrix. The specified matrix is relative to the coordinate system of the trigger volume's parent, if one exists. If the trigger volume has no parent, the matrix is relative to the world space coordinate system. A trigger volume's transform matrix can not support scale.

Examples

```
function MyTriggerVolumeCallback1( triggerVolume, state, count )
    if state == TriggerVolumeOverlapState.Enter then
        print( count .. " new overlaps for trigger volume " .. tostring( triggerVolume ) .. " this
frame" )
    elseif state == TriggerVolumeOverlapState.Exit then
        print( count .. " old overlaps for trigger volume " .. tostring( triggerVolume ) .. " this
frame" )
    end
end

function MyTriggerVolumeCallback2( triggerVolume, state, count )
    for i = 1, count do
        local   overlapType = TriggerVolume.GetOverlapType( triggerVolume, i )
        local   overlapData = TriggerVolume.GetOverlapData( triggerVolume, i )
        print( "trigger volume " .. tostring( triggerVolume ) .. " overlap " .. tostring( i ) .. ":
type=" .. tostring( overlapType ) .. ", object=" .. tostring( overlapData.object ) )
    end
end

triggerVolumes = {}
triggerVolumes[1] = Collision.CreateTriggerVolume( "simpleTriggerVolume1", 16,
TriggerVolumeShape.Sphere, 1.0 )
triggerVolumes[2] = Collision.CreateTriggerVolume( "simpleTriggerVolume2", 16, TriggerVolumeShape.Box,
3.0, 2.0, 4.0 )
triggerVolumes[3] = Collision.CreateTriggerVolume( "resourceTriggerVolume1", 16,
"kTriggerVolumeShapeResource1" )
triggerVolumes[4] = Collision.CreateTriggerVolume( "resourceTriggerVolume2", 16, Resource.Find(
"kTriggerVolumeShapeResource2" ) )

TriggerVolume.SetLocalMatrix( triggerVolumes[1], triggerVolumeTransformMatrix[1] )
TriggerVolume.SetLocalMatrix( triggerVolumes[2], TriggerVolume.GetLocalMatrix( triggerVolumes[1] ) )
TriggerVolume.SetPosition( triggerVolumes[3], triggerVolumePosition[1] )
TriggerVolume.SetPosition( triggerVolumes[4], TriggerVolume.GetPosition( triggerVolumes[3] ) )

for i = 1, #triggerVolumes do
    TriggerVolume.AddCallback( triggerVolumes[i], MyTriggerVolumeCallback1 )
    TriggerVolume.AddCallback( triggerVolumes[i], MyTriggerVolumeCallback2 )
    if TriggerVolume.IsLayerOverlapsEnabled( triggerVolumes[i], CollisionLayer.Scene ) then
        TriggerVolume.EnableLayerOverlaps( triggerVolumes[i], CollisionLayer.Scene, false )
    end
    if not TriggerVolume.IsEnabled( triggerVolumes[i] ) then
        TriggerVolume.Enable( triggerVolumes[i], true )
    end
end

TriggerVolume.RemoveCallback( triggerVolumes[3], MyTriggerVolumeCallback2 )
TriggerVolume.RemoveAllCallbacks( triggerVolumes[4] )

local   numOverlaps = TriggerVolume.GetOverlapCount( triggerVolumes[1] )

for i = 1, numOverlaps do
    local   overlapType = TriggerVolume.GetOverlapType( triggerVolumes[1], i )
    local   overlapData = TriggerVolume.GetOverlapData( triggerVolumes[1], i )
    print( "trigger volume " .. tostring( triggerVolumes[1] ) .. " overlap " .. tostring( i ) .. ":
type=" .. tostring( overlapType ) .. ", object=" .. tostring( overlapData.object ) )
end
```

See Also

None

TriggerVolume.SetPosition

Brief

Sets the trigger volume's position.

Definition

```
TriggerVolume.SetPosition( TriggerVolume triggerVolume, Vector4 position )
```

Arguments

triggerVolume - The trigger volume to update.position - The position to set the trigger volume to.

Return Values

None.

Description

Sets the trigger volume's position. The specified position is relative to the coordinate system of the trigger volume's parent, if one exists. If the trigger volume has no parent, the position is relative to the world space coordinate system.

Examples

```
function MyTriggerVolumeCallback1( triggerVolume, state, count )
    if state == TriggerVolumeOverlapState.Enter then
        print( count .. " new overlaps for trigger volume " .. tostring( triggerVolume ) .. " this
frame" )
    elseif state == TriggerVolumeOverlapState.Exit then
        print( count .. " old overlaps for trigger volume " .. tostring( triggerVolume ) .. " this
frame" )
    end
end

function MyTriggerVolumeCallback2( triggerVolume, state, count )
    for i = 1, count do
        local   overlapType = TriggerVolume.GetOverlapType( triggerVolume, i )
        local   overlapData = TriggerVolume.GetOverlapData( triggerVolume, i )
        print( "trigger volume " .. tostring( triggerVolume ) .. " overlap " .. tostring( i ) .. ":
type=" .. tostring( overlapType ) .. ", object=" .. tostring( overlapData.object ) )
    end
end

triggerVolumes = {}
triggerVolumes[1] = Collision.CreateTriggerVolume( "simpleTriggerVolume1", 16,
TriggerVolumeShape.Sphere, 1.0 )
triggerVolumes[2] = Collision.CreateTriggerVolume( "simpleTriggerVolume2", 16, TriggerVolumeShape.Box,
3.0, 2.0, 4.0 )
triggerVolumes[3] = Collision.CreateTriggerVolume( "resourceTriggerVolume1", 16,
"kTriggerVolumeShapeResource1" )
triggerVolumes[4] = Collision.CreateTriggerVolume( "resourceTriggerVolume2", 16, Resource.Find(
"kTriggerVolumeShapeResource2" ) )

TriggerVolume.SetLocalMatrix( triggerVolumes[1], triggerVolumeTransformMatrix[1] )
TriggerVolume.SetLocalMatrix( triggerVolumes[2], TriggerVolume.GetLocalMatrix( triggerVolumes[1] ) )
TriggerVolume.SetPosition( triggerVolumes[3], triggerVolumePosition[1] )
TriggerVolume.SetPosition( triggerVolumes[4], TriggerVolume.GetPosition( triggerVolumes[3] ) )

for i = 1, #triggerVolumes do
    TriggerVolume.AddCallback( triggerVolumes[i], MyTriggerVolumeCallback1 )
    TriggerVolume.AddCallback( triggerVolumes[i], MyTriggerVolumeCallback2 )
    if TriggerVolume.IsLayerOverlapsEnabled( triggerVolumes[i], CollisionLayer.Scene ) then
        TriggerVolume.EnableLayerOverlaps( triggerVolumes[i], CollisionLayer.Scene, false )
    end
    if not TriggerVolume.IsEnabled( triggerVolumes[i] ) then
        TriggerVolume.Enable( triggerVolumes[i], true )
    end
end

TriggerVolume.RemoveCallback( triggerVolumes[3], MyTriggerVolumeCallback2 )
TriggerVolume.RemoveAllCallbacks( triggerVolumes[4] )

local   numOverlaps = TriggerVolume.GetOverlapCount( triggerVolumes[1] )

for i = 1, numOverlaps do
    local   overlapType = TriggerVolume.GetOverlapType( triggerVolumes[1], i )
    local   overlapData = TriggerVolume.GetOverlapData( triggerVolumes[1], i )
    print( "trigger volume " .. tostring( triggerVolumes[1] ) .. " overlap " .. tostring( i ) .. ":
type=" .. tostring( overlapType ) .. ", object=" .. tostring( overlapData.object ) )
end
```

See Also

None

User

User.GetName

Brief

Returns the user's Online ID.

Definition

```
string User.GetName( User User )
```

Arguments

user - The user object.

Return Values

The user's Online ID.

Description

Returns the user's Online ID, but only if you launch HomeDeveloper.self in online mode. If launching in offline mode, this function returns the string "Me".

Examples

```
myUser = LocalPlayer.GetUser()
print( "My name is " .. User.GetName( myUser ) )
```

See Also

- LocalPlayer.GetUser

User.GetPerson

Brief

Returns a Person object if the user is found in the current instance and nil otherwise.

Definition

```
Person User.GetPerson( User user )
```

Arguments

user - The User object

Return Values

Returns a Person object if the user is found and nil otherwise.

Description

Returns a Person object if the user is found and nil otherwise. Person objects only exist in the current instance of the scene.

Examples

```
local person = User.GetPerson( user )
if person ~= nil then
    friendLockStatus = Person.Lock( person ) -- locking a friend
end
```

See Also

None

User.GetSceneOwner

Brief

Returns the owner of a scene as a User object.

Definition

```
User|nil User.GetSceneOwner()
```

Arguments

None

Return Values

The owner as a User or nil if there is no owner or it hasn't calculated the owner yet.

Description

Returns the owner of a scene as a User object. If called from an apartment it will return the owner of the apartment and if called from a club house it will return the leader of the club house. All other types of scene will return nil. It may initially return nil if you are not the owner of the scene so you should keep calling it each frame until it returns a User. The cases in which this could happen are club houses and when visiting a friend's apartment.

Examples

```
local owner = User.GetSceneOwner()
if owner ~= nil and User.GetName( owner ) == Person.GetName( LocalPlayer.GetPerson() ) then
    print( 'I am the owner' )
end
```

See Also

- User.GetPerson
- Person.GetInstanceSessionMasterId

User.GoToLocation

Brief

Go to the user's location.

Definition

```
User.GoToLocation( User user )
```

Arguments

user - The user object.

Return Values

Description

Go to the user's location.

Examples

```
friends = LocalPlayer.GetFriends()
User.GoToLocation( friends[ 1 ] )
```

See Also

- LocalPlayer.GetFriends

User.InviteToLocation

Brief

Invites the user to the Local player's location.

Definition

```
User.InviteToLocation( User user )
```

Arguments

user - The user object.

Return Values

Description

Invites the user to the Local player's location.

Examples

```
friends = LocalPlayer.GetFriends()
for index, friend in ipairs( friends ) do
    if User.IsInHome( friend ) == true then
        User.InviteToLocation( friend )
    end
end
```

See Also

- LocalPlayer.GetFriends

User.IsFriend

Brief

Returns true if the user is a friend.

Definition

```
boolean User.IsFriend( User user )
```

Arguments

user - The user object.

Return Values

Returns true if the user is a friend and false otherwise.

Description

Returns true if the user is a friend.

Examples

```
friends = LocalPlayer.GetFriends()
for index, friend in ipairs( friends ) do
    if User.IsFriend( friend ) == true then
        print( User.GetName( friend ) .. ' is a friend.' )
    else
        print( 'Something bad has happened.' )
    end
end
```

See Also

- LocalPlayer.GetFriends

User.IsInHome

Brief

Returns true if the user is in PS Home.

Definition

```
boolean User.IsInHome( User user )
```

Arguments

user - The user object.

Return Values

Returns true if the user is in PS Home and false otherwise.

Description

Returns true if the user is in PS Home.

Examples

```
friends = LocalPlayer.GetFriends()
for index, friend in ipairs( friends ) do
    if User.IsInHome( friend ) == true then
        print( User.GetName( friend ) .. ' is in PS Home.' )
    end
end
```

See Also

- LocalPlayer.GetFriends
- User.IsOnline

User.IsOnline

Brief

Returns true if the user is online.

Definition

```
boolean User.IsOnline( User user )
```

Arguments

user - The user object.

Return Values

Returns true if the user is online and false otherwise.

Description

Returns true if the user is online. They are not necessarily in PS Home when online.

Examples

```
friends = LocalPlayer.GetFriends()
for index, friend in ipairs( friends ) do
    if User.IsOnline( friend ) == true then
        print( User.GetName( friend ) .. ' is online.' )
    end
end
```

See Also

- LocalPlayer.GetFriends
- User.IsInHome

Vector4

Vector4.Abs

Brief

Compute the component-wise absolute value of a vector.

Definition

```
Vector4.Abs(Vector4 dest, Vector4 v)
```

Arguments

dest - destination vector to store result.v - vector to negate.

Return Values

None.

Description

Sets dest to [math.abs(v:X()), math.abs(v:Y()), math.abs(v:Z()), math.abs(v:W())].

Examples

```
local v = Vector4.Create(-1, 2, -3)
v:Abs(v)
print(v)        -- [1, 2, 3]
```

See Also

None

Vector4.Add

Brief

Adds one vector or number to another and stores the result in the destination.

Definition

```
Vector4.Add( Vector4 destination, Vector4|number source1, Vector4|number source2 )
```

Arguments

destination - Vector to store the result of the addition.source1 - Vector or number to be added to source2.source2 - Vector or number to be added to source1.

Return Values

None.

Description

Adds one vector or number to another and stores the result in the destination.

Descriptive notation:

destination.x = source1.x|source1 + source2.x|source2

destination.y = source1.y|source1 + source2.y|source2

destination.z = source1.z|source1 + source2.z|source2

destination.w = source1.w|source1 + source2.w|source2

Examples

```
result = Vector4.Create()
v1 = Vector4.Create( 1, 2, 3, 4 )
v2 = Vector4.Create( 4, 3, 2, 1 )
Vector4.Add( result, v1, v2 )
Vector4.Add( result, 3, v2 )
```

See Also

- Vector4.Create

Vector4.Calculate2dLineIntersection

Brief

Calculates a 2-dimensional line intersection.

Definition

```
boolean Vector4.Calculate2dLineIntersection( Vector4 intersectionPoint, Vector4 p0, Vector4 p1,
Vector4 q0, Vector4 q1, boolean infiniteIntersection = false )
```

Arguments

intersectionPoint - The result of the line intersection.p0 - First endpoint of first line.p1 - Second endpoint of first line.q0 - First endpoint of second line.q1 - Second endpoint of second line.infiniteIntersection - true if the supplied lines are to be treated as infinite rays, false otherwise

Return Values

Returns true if lines intersect, false otherwise.

Description

Calculates a 2-dimensional line intersection with the line p, defined by points p0 and p1, with the line q, defined by points q0 and q1. If the infiniteIntersection is true it returns false if p and q are parallel. If infiniteIntersection is false it returns false if the lines are parallel or the intersection does not lie on both p and q.

Examples

```
p0 = Vector4.Create( 1, 0 )
p1 = Vector4.Create( 0, 0 )
q0 = Vector4.Create( 0, 0.5 )
q1 = Vector4.Create( 1, 0.5 )
ip = Vector4.Create()

if Vector4.Calculate2dLineIntersection( ip, p0, p1, q0, q1 ) then
    print( "The lines intersect!" )
end
```

See Also

- Vector4.Create

Vector4.Copy

Brief

Copies the values of one vector into another.

Definition

```
Vector4.Copy( Vector4 destination, Vector4 source )
```

Arguments

destination - The vector to copy to.source - The vector to be copied.

Return Values

None.

Description

Copies the values of the source vector and stores them in the destination. This is used to copy the values as opposed to pointing to the same vector as an assignment would do. Use copy as vec2 = vec1 does not copy values but instead makes vec2 point to vec1.

Examples

```
vec1 = Vector4.Create( 4, 3, 2, 1 )
vec2 = Vector4.Create()
Vector4.Copy( vec2, vec1 )
```

See Also

- Vector4.Create

Vector4.Create

Brief

Creates a new Vector4 object using the specified values.

Definition

```
Vector4 Vector4.Create( Vector4 source )
Vector4 Vector4.Create( number x = 0.0, number y = 0.0, number z = 0.0, number w = 0.0 )
```

Arguments

source - Vector whose values are used to initialize newly created vector.x - Number to be written into the first element of the vector (defaults to 0).y - Number to be written into the second element of the vector (defaults to 0).z - Number to be written into the third element of the vector (defaults to 0).w - Number to be written into the fourth element of the vector (defaults to 0).

Return Values

A new Vector4 instance.

Description

Creates a new Vector4 object using the specified values.

Examples

```
-- Create a default zero (0, 0, 0, 0) vector
vec1 = Vector4.Create()

-- Create a vector with supplied values
vec2 = Vector4.Create( 1, 2, 3, 4 )

-- Create a vector using the values of another
vec3 = Vector4.Create( vec2 )
```

See Also

- Vector4.CreateBroadcast

Vector4.CreateBroadcast

Brief

Creates a new Vector4 and sets all its elements to the specified value.

Definition

```
Vector4 Vector4.CreateBroadcast( number value )
```

Arguments

value - The value to broadcast to all the elements of the new vector.

Return Values

A new Vector4 instance.

Description

Creates a new Vector4 and sets all its elements to the specified value.

Examples

```
vecHalf = Vector4.CreateBroadcast( 0.5 )
vec = Vector4.Create( 1, 2, 3 )
Vector4.Multiply( vec, vec, vecHalf )
```

See Also

- Vector4.Create

Vector4.Cross

Brief

Performs a cross product between two 3-dimensional vectors.

Definition

```
Vector4.Cross( Vector4 destination, Vector4 src1, Vector4 src2 )
```

Arguments

destination - Vector to store the result of the cross product.src1 - First vector to take part in the cross product.src2 - Second vector to take part in the cross product.

Return Values

None.

Description

Performs a 3-dimensional cross product between src1 and src2 storing the result in destination. The vector generated will be perpendicular to both src1 and src2.

Vector algebra notation:

src1 X src2

Descriptive notation:

destination.x = (src1.y * src2.z) - (src1.z * src2.y)

destination.y = (src1.z * src2.x) - (src1.x * src2.z)

destination.z = (src1.x * src2.y) - (src1.y * src2.x)

Examples

```
xAxis = Vector4.Create( 1, 0, 0 )
yAxis = Vector4.Create( 0, 1, 0 )
result = Vector4.Create()
Vector4.Cross( result, xAxis, yAxis )
```

See Also

- Vector4.Create

Vector4.Dist2Sqr

Brief

Computes the squared Euclidean distance between two points in 2D space.

Definition

```
number Vector4.Dist2Sqr( Vector4 p1, Vector4 p2 )
```

Arguments

p1 - A point in 2D space.p2 - A second point in 2D space.

Return Values

The squared Euclidean distance between the two points.

Description

Computes the squared Euclidean distance between two points in 2D space.

Vector algebra notation:

((p2-p1) dot (p2-p1))

Examples

```
v1 = Vector4.Create(1, 2)
v2 = Vector4.Create(2, 3)
print("Distance between", v1, "and", v2, "is:", math.sqrt(Vector4.Dist2Sqr(v1, v2)))
```

See Also

- Vector4.Dist3Sqr
- Vector4.Dist4Sqr

Vector4.Dist3Sqr

Brief

Computes the squared Euclidean distance between two points in 3D space.

Definition

```
number Vector4.Dist3Sqr( Vector4 p1, Vector4 p2 )
```

Arguments

p1 - A point in 3D space.p2 - A second point in 3D space.

Return Values

The squared Euclidean distance between the two points.

Description

Computes the squared Euclidean distance between two points in 3D space.

Vector algebra notation:

((p2-p1) dot (p2-p1))

Examples

```
v1 = Vector4.Create(1, 2, 3)
v2 = Vector4.Create(2, 3, 4)
print("Distance between", v1, "and", v2, "is:", math.sqrt(Vector4.Dist3Sqr(v1, v2)))
```

See Also

- Vector4.Dist2Sqr
- Vector4.Dist4Sqr

Vector4.Dist4Sqr

Brief

Computes the squared Euclidean distance between two points in 4D space.

Definition

```
number Vector4.Dist4Sqr( Vector4 p1, Vector4 p2 )
```

Arguments

p1 - A point in 4D space.p2 - A second point in 4D space.

Return Values

The squared Euclidean distance between the two points.

Description

Computes the squared Euclidean distance between two points in 4D space.

Vector algebra notation:

((p2-p1) dot (p2-p1))

Examples

```
v1 = Vector4.Create( 1, 2, 3, 4 )
v2 = Vector4.Create( 2, 3, 4, 5 )
print("Distance between", v1, "and", v2, "is:", math.sqrt(Vector4.Dist4Sqr(v1, v2)))
```

See Also

- Vector4.Dist2Sqr
- Vector4.Dist3Sqr

Vector4.Divide

Brief

Divides one vector or scalar by the other vector or scalar.

Definition

```
Vector4.Divide( Vector4 destination, Vector4|number source1, Vector4|number source2 )
```

Arguments

destination - Vector to store the result of the division.source1 - The dividend of the vector division.source2 - The divisor of the vector division.

Return Values

None.

Description

Divides source1 by source2 storing the result in destination.

Descriptive notation:

destination.x = source1.x|source1 / source2.x|source2

destination.y = source1.y|source1 / source2.y|source2

destination.z = source1.z|source1 / source2.z|source2

destination.w = source1.w|source1 / source2.w|source2

Examples

```
result = Vector4.Create()
v1 = Vector4.Create( 1, 2, 3, 4 )
v2 = Vector4.Create( 4, 3, 2, 1 )
Vector4.Divide( result, v1, v2 )
Vector4.Divide( result, 1.0, v2 )
```

See Also

- Vector4.Create

Vector4.Dot2

Brief

Performs a dot product between two 2-dimensional vectors.

Definition

```
number Vector4.Dot2( Vector4 src1, Vector4 src2 )
```

Arguments

src1 - First vector to take part in the dot product.src2 - Second vector to take part in the dot product.

Return Values

The number representing the dot product of the two supplied 2-dimensional vectors.

Description

Performs a 2-dimensional dot product with src1 and src2 returning the result.

Vector algebra notation:

src1 . src2

Descriptive notation:

returnvalue = (src1.x * src2.x) + (src1.y * src2.y)

Examples

```
v1 = Vector4.Create( 1, 2 )
v2 = Vector4.Create( 4, 3 )
result = Vector4.Dot2( v1, v2 )
```

See Also

- Vector4.Create
- Vector4.Dot3
- Vector4.Dot4

Vector4.Dot3

Brief

Performs a dot product between two 3-dimensional vectors.

Definition

```
number Vector4.Dot3( Vector4 src1, Vector4 src2 )
```

Arguments

src1 - First vector to take part in the dot product.src2 - Second vector to take part in the dot product.

Return Values

The number representing the dot product of the two supplied 3-dimensional vectors.

Description

Performs a 3-dimensional dot product with src1 and src2 returning the result.

Vector algebra notation:

src1 . src2

Descriptive notation:

returnvalue = (src1.x * src2.x) + (src1.y * src2.y) + (src1.z * src2.z)

Examples

```
v1 = Vector4.Create( 1, 2, 3 )
v2 = Vector4.Create( 4, 3, 2 )
result = Vector4.Dot3( v1, v2 )
```

See Also

- Vector4.Create
- Vector4.Dot2
- Vector4.Dot4

Vector4.Dot4

Brief

Performs a dot product between two 4-dimensional vectors.

Definition

```
number Vector4.Dot4( Vector4 src1, Vector4 src2 )
```

Arguments

src1 - First vector to take part in the dot product.src2 - Second vector to take part in the dot product.

Return Values

The number representing the dot product of the two supplied 4-dimensional vectors.

Description

Performs a 4-dimensional dot product with src1 and src2 returning the result.

Vector algebra notation:

src1 . src2

Descriptive notation:

returnvalue = (src1.x * src2.x) + (src1.y * src2.y) + (src1.z * src2.z) + (src1.w * src2.w)

Examples

```
v1 = Vector4.Create( 1, 2, 3, 4 )
v2 = Vector4.Create( 4, 3, 2, 1 )
result = Vector4.Dot4( v1, v2 )
```

See Also

- Vector4.Create
- Vector4.Dot2
- Vector4.Dot3

Vector4.HorizAdd3

Brief

Compute the horizontal sum of a 3D vector.

Definition

```
number Vector4.HorizAdd3(Vector4 v)
```

Arguments

v - input vector.

Return Values

The sum of the X, Y and Z components of the input vector.

Description

Returns v:X() + v:Y() + v:Z(). W is ignored.

Examples

```
local v = Vector4.Create(1, 2, 3)
print(v:HorizAdd3())        -- 6
```

See Also

- Vector4.HorizAdd4

Vector4.HorizAdd4

Brief

Compute the horizontal sum of a 4D vector.

Definition

```
number Vector4.HorizAdd4(Vector4 v)
```

Arguments

v - input vector.

Return Values

The sum of the X, Y, Z and W components of the input vector.

Description

Returns v:X() + v:Y() + v:Z() + v:W().

Examples

```
local v = Vector4.Create(1, 2, 3, 4)
print(v:HorizAdd4())        -- 10
```

See Also

- Vector4.HorizAdd3

Vector4.Length2

Brief

Calculates the length of a 2-dimensional vector.

Definition

```
number Vector4.Length2( Vector4 vector )
```

Arguments

vector - Source vector for the calculation.

Return Values

A number holding the magnitude of the specified 2-dimensional vector.

Description

Calculates the length of the specified 2-dimensional vector.

Vector algebra notation:

vector	or sqrt(vector dot vector)

Descriptive notation:

returnvalue = sqrt((vector.x * vector.x) + (vector.y * vector.y))

Examples

```
v1 = Vector4.Create( 1, 2 )
print( "Vector magnitude is: " .. Vector4.Length2( v1 ) )
```

See Also

- Vector4.Create
- Vector4.Length3
- Vector4.Length4

Vector4.Length2Sqr

Brief

Calculates the squared length of a 2-dimensional vector.

Definition

```
number Vector4.Length2Sqr( Vector4 vector )
```

Arguments

vector - Source vector for the calculation.

Return Values

A number holding the squared magnitude of the specified 2-dimensional vector.

Description

Calculates the squared length of the specified 2-dimensional vector.

Vector algebra notation:

(vector dot vector)

Descriptive notation:

returnvalue = (vector.x * vector.x) + (vector.y * vector.y)

Examples

```
v1 = Vector4.Create( 1, 2 )
print( "Vector squared magnitude is: " .. Vector4.Length2Sqr( v1 ) )
```

See Also

- Vector4.Create

- Vector4.Length3Sqr
- Vector4.Length4Sqr
- Vector4.Length2

Vector4.Length3

Brief

Calculates the length of a 3-dimensional vector.

Definition

```
number Vector4.Length3( Vector4 vector )
```

Arguments

vector - Source vector for the calculation.

Return Values

A number holding the magnitude of the specified 3-dimensional vector.

Description

Calculates the length of the specified 3-dimensional vector.

Vector algebra notation:

vector	or sqrt(vector dot vector)

Descriptive notation:

returnvalue = sqrt((vector.x * vector.x) + (vector.y * vector.y) + (vector.z * vector.z))

Examples

```
v1 = Vector4.Create( 1, 2, 3 )
print( "Vector magnitude is: " .. Vector4.Length3( v1 ) )
```

See Also

- Vector4.Create
- Vector4.Length2
- Vector4.Length4

Vector4.Length3Sqr

Brief

Calculates the squared length of a 3-dimensional vector.

Definition

```
number Vector4.Length3Sqr( Vector4 vector )
```

Arguments

vector - Source vector for the calculation.

Return Values

A number holding the squared magnitude of the specified 3-dimensional vector.

Description

Calculates the squared length of the specified 3-dimensional vector.

Vector algebra notation:

(vector dot vector)

Descriptive notation:

returnvalue = (vector.x * vector.x) + (vector.y * vector.y) + (vector.z * vector.z)

Examples

```
v1 = Vector4.Create( 1, 2, 3 )
print( "Vector squared magnitude is: " .. Vector4.Length34Sqr( v1 ) )
```

See Also

- Vector4.Create
- Vector4.Length2Sqr
- Vector4.Length4Sqr
- Vector4.Length3

Vector4.Length4

Brief

Calculates the length of a 4-dimensional vector.

Definition

```
number Vector4.Length4( Vector4 vector )
```

Arguments

vector - Source vector for the calculation.

Return Values

A number holding the magnitude of the specified 4-dimensional vector.

Description

Calculates the length of the specified 4-dimensional vector.

Vector algebra notation:

vector	or sqrt(vector dot vector)

Descriptive notation:

returnvalue = sqrt((vector.x * vector.x) + (vector.y * vector.y) + (vector.z * vector.z) + (vector.w * vector.w))

Examples

```
v1 = Vector4.Create( 1, 2, 3, 4 )
print( "Vector magnitude is: " .. Vector4.Length4( v1 ) )
```

See Also

- Vector4.Create
- Vector4.Length2
- Vector4.Length3

Vector4.Length4Sqr

Brief

Calculates the squared length of a 4-dimensional vector.

Definition

```
number Vector4.Length4Sqr( Vector4 vector )
```

Arguments

vector - Source vector for the calculation.

Return Values

A number holding the squared magnitude of the specified 4-dimensional vector.

Description

Calculates the squared length of the specified 4-dimensional vector.

Vector algebra notation:

(vector dot vector)

Descriptive notation:

returnvalue = (vector.x * vector.x) + (vector.y * vector.y) + (vector.z * vector.z) + (vector.w * vector.w)

Examples

```
v1 = Vector4.Create( 1, 2, 3, 4 )
print( "Vector squared magnitude is: " .. Vector4.Length4Sqr( v1 ) )
```

See Also

- Vector4.Create
- Vector4.Length2Sqr
- Vector4.Length3Sqr
- Vector4.Length4

Vector4.Max

Brief

Compute the maximum element values of two vectors.

Definition

```
Vector4.Max(Vector4 dest, Vector4 source1, Vector4 source2)
```

Arguments

dest - Vector to hold result. May be the same vector as either of source vectors if desired.source1 - First source vector.source2 - Second source vector.

Return Values

None.

Description

Compares each element in source1 with equivalent element in source2 and stores the greater value in the equivalent element of dest. That is:
dest =

Unknown macro: { max(source1.x, source2.x), max(source1.y, source2.y), max(source1.z, source2.z), max(source1.w, source2.w) }

Examples

```
-- assuming verts is an array of Vector4 values
local min = Vector4.Create(Const.MaxNum, Const.MaxNum, Const.MaxNum)
local max = Vector4.Create(-Const.MaxNum, -Const.MaxNum, -Const.MaxNum)
for i = 1, numVerts do
    Vector4.Min(min, min, verts[i])
    Vector4.Max(max, max, verts[i])
end
-- min and max now store bounds of vert list
```

See Also

- Vector4.Min
- StreamOps.BoundsVec4

Vector4.Min

Brief

Compute the minimum element values of two vectors.

Definition

```
Vector4.Min(Vector4 dest, Vector4 source1, Vector4 source2)
```

Arguments

dest - Vector to hold result. May be the same vector as either of source vectors if desired.source1 - First source vector.source2 - Second source vector.

Return Values

None.

Description

Compares each element in source1 with equivalent element in source2 and stores the lower value in the equivalent element of dest. That is:
dest =

Unknown macro: { min(source1.x, source2.x), min(source1.y, source2.y), min(source1.z, source2.z), min(source1.w, source2.w) }

Examples

```
-- assuming verts is an array of Vector4 values
local min = Vector4.Create(Const.MaxNum, Const.MaxNum, Const.MaxNum)
local max = Vector4.Create(-Const.MaxNum, -Const.MaxNum, -Const.MaxNum)
for i = 1, numVerts do
    Vector4.Min(min, min, verts[i])
    Vector4.Max(max, max, verts[i])
end
-- min and max now store bounds of vert list
```

See Also

- Vector4.Max
- StreamOps.BoundsVec4

Vector4.Multiply

Brief

Multiplies two vectors or scalars together storing the result as a vector.

Definition

```
Vector4.Multiply( Vector4 destination, Vector4|number source1, Vector4|number source2 )
```

Arguments

destination - Vector to store the result of the multiplication.source1 - The first argument for the multiplication, can be a Vector4 or a scalar.source2 - The second argument for the multiplication, can be a Vector4 or a scalar.

Return Values

None.

Description

Multiplies source1 and source2 together storing the result in the destination vector.

Descriptive notation:

destination.x = source1.x|source1 * source2.x|source2

destination.y = source1.y|source1 * source2.y|source2

destination.z = source1.z|source1 * source2.z|source2

destination.w = source1.w|source1 * source2.w|source2

Examples

```
result = Vector4.Create()
v1 = Vector4.Create( 1, 2, 3, 4 )
v2 = Vector4.Create( 4, 3, 2, 1 )
Vector4.Multiply( result, v1, v2 )
Vector4.Multiply( result, v1, 4.0 )
Vector4.Multiply( result, -3.0, v2 )
Vector4.Multiply( result, -3.0, 4 )
```

See Also

- Vector4.Create

Vector4.MultiplyAdd

Brief

Multiplies two vectors together and then adds the third vector.

Definition

```
Vector4.MultiplyAdd( Vector4 destination, Vector4|number source1, Vector4|number source2,
Vector4|number source3 )
```

Arguments

destination - Vector to store the result of the multiplication.source1 - The first vector argument for the multiplication.source2 - The second vector argument for the multiplication.source3 - The third vector argument for the addition.

Return Values

None.

Description

Multiplies source1 and source2 together and then adds the third vector, storing the result in destination.

Descriptive notation:

destination.x = (source1.x|source1 * source2.x|source2) + source3.x|source3

destination.y = (source1.y|source1 * source2.y|source2) + source3.y|source3

destination.z = (source1.z|source1 * source2.z|source2) + source3.z|source3

destination.w = (source1.w|source1 * source2.w|source2) + source3.w|source3

Examples

```
result = Vector4.Create()
v1 = Vector4.Create( 1, 2, 3, 4 )
v2 = Vector4.Create( 4, 3, 2, 1 )
v3 = Vector4.Create( 1, 0, 0, 1 )
Vector4.MultiplyAdd( result, v1, v2, v3 )
Vector4.MultiplyAdd( result, 1, 2, 6 )
```

See Also

- Vector4.Create

Vector4.Negate

Brief

Negate a vector in-place.

Definition

```
Vector4.Negate(Vector4 v)
```

Arguments

v - vector to negate.

Return Values

None.

Description

Sets v to -v.

Examples

```
local v = Vector4.Create(1, 2, 3)
v:Negate()
print(v)        -- [-1, -2, -3]
```

See Also

None

Vector4.Normal2

Brief

Returns the normalization of a 2-dimensional vector.

Definition

```
Vector4 Vector4.Normal2( Vector4 vector )
```

Arguments

vector - The vector to normalize.

Return Values

A new Vector4 instance containing a normalized copy of the source vector.

Description

Returns the normalization of the specified 2-dimensional vector (i.e. a new vector of unit length). The source vector is not modified.

Descriptive notation:

length = sqrt(src.x * src.x + src.y * src.y)

returnvec.x = src.x / length

returnvec.y = src.y / length

returnvec.z = 0

returnvec.w = 0

Examples

```
vec = Vector4.Create( 1, 2 )
result = Vector4.Normal2( vec )
print( result )
```

See Also

- Vector4.Create
- Vector4.Normal3
- Vector4.Normal4

Vector4.Normal3

Brief

Returns the normalization of a 3-dimensional vector.

Definition

```
Vector4 Vector4.Normal3( Vector4 src )
```

Arguments

src - The vector to use as a source to normalize.

Return Values

A new Vector4 instance containing a normalized copy of the source vector.

Description

Returns the normalization of the specified 3-dimensional vector (i.e. a new vector of unit length). The source vector is not modified.

Descriptive notation:

length = sqrt(src.x*src.x + src.y*src.y + src.z*src.z)

returnvec.x = src.x / length

returnvec.y = src.y / length

returnvec.z = src.z / length

returnvec.w = 0

Examples

```
vec = Vector4.Create( 1, 2, 3 )
result = Vector4.Normal3( vec )
print( result )
```

See Also

- Vector4.Create
- Vector4.Normal2
- Vector4.Normal4

Vector4.Normal4

Brief

Returns the normalization of a 4-dimensional vector.

Definition

```
Vector4 Vector4.Normal4( Vector4 src )
```

Arguments

src - The vector to use as a source to normalize.

Return Values

A new Vector4 instance containing a normalized copy of the source vector.

Description

Returns the normalization of the specified 4-dimensional vector (i.e. a new vector of unit length). The source vector is not modified.

Descriptive notation:

length = sqrt(src.x*src.x + src.y*src.y + src.z*src.z + src.w*src.w)

returnvec.x = src.x / length

returnvec.y = src.y / length

returnvec.z = src.z / length

returnvec.w = src.z / length

Examples

```
vec = Vector4.Create( 1, 2, 3, 4 )
result = Vector4.Normal4( vec )
print( result )
```

See Also

- Vector4.Create
- Vector4.Normal2
- Vector4.Normal3

Vector4.Normalize2

Brief

Normalizes a vector using Length2 of that vector.

Definition

```
Vector4.Normalize2( Vector4 vector )
```

Arguments

vector - The vector to normalize.

Return Values

None.

Description

Normalizes the specified vector using Length2 of the vector.

Descriptive notation:

length = sqrt(vector.x*vector.x + vector.y*vector.y)

vector.x = vector.x / length

vector.y = vector.y / length

vector.z = 0

vector.w = 0

Examples

```
vector = Vector4.Create( 1, 3, 2, 4 )
Vector4.Normalize2( vector )
print( vector )
```

See Also

- Vector4.Create
- Vector4.Normalize3
- Vector4.Normalize4

Vector4.Normalize3

Brief

Normalizes a vector using Length3 of that vector.

Definition

```
Vector4.Normalize3( Vector4 vector )
```

Arguments

vector - The vector to normalize.

Return Values

None.

Description

Normalizes the specified vector using Length3 of the vector.

Descriptive notation:

length = sqrt(vector.x*vector.x + vector.y*vector.y + vector.z*vector.z)

vector.x = vector.x / length

vector.y = vector.y / length

vector.z = vector.z / length

vector.w = 0

Examples

```
vector = Vector4.Create( 1, 3, 2, 4 )
Vector4.Normalize3( vector )
print( vector )
```

See Also

- Vector4.Create
- Vector4.Normalize2
- Vector4.Normalize4

Vector4.Normalize4

Brief

Normalizes a vector using Length4 of that vector.

Definition

```
Vector4.Normalize4( Vector4 vector )
```

Arguments

vector - The vector to normalize.

Return Values

None.

Description

Normalizes the specified vector using Length4 of the vector.

Descriptive notation:

length = sqrt(vector.x*vector.x + vector.y*vector.y + vector.z*vector.z + vector.w*vector.w)

vector.x = vector.x / length

vector.y = vector.y / length

vector.z = vector.z / length

vector.w = vector.w / length

Examples

```
vector = Vector4.Create( 1, 3, 2, 4 )
Vector4.Normalize4( vector )
print( vector )
```

See Also

- Vector4.Create
- Vector4.Normalize2
- Vector4.Normalize3

Vector4.Recip

Brief

Compute the component-wise reciprocal of a vector.

Definition

```
Vector4.Recip(Vector4 dest, Vector4 v)
```

Arguments

dest - destination vector to store result.v - input vector.

Return Values

None.

Description

Sets dest to [1 / v:X(), 1 / v:Y(), 1 / v:Z(), 1 / v:W()]. If any elements of v are 0 the result is undefined. Note that due to floating point arithmetic precision, Recip(Recip) is not necessarily x.

Examples

```
local v = Vector4.Create(-1, 2, -3)
v:Recip(v)
print(v)        -- [-1, 0.5, -0.333333]
```

See Also

- Vector4.RecipSqrt

Vector4.RecipSqrt

Brief

Compute the component-wise reciprocal square root of a vector.

Definition

```
Vector4.RecipSqrt(Vector4 dest, Vector4 v)
```

Arguments

dest - destination vector to store result.v - input vector.

Return Values

None.

Description

Sets dest to [1 / math.sqrt(v:X()), 1 / math.sqrt(v:Y()), 1 / math.sqrt(v:Z()), 1 / math.sqrt(v:W()]. If any elements of v are <= 0 the result is undefined. This operation is faster than the equivalent combination of Sqrt and Recip.

Examples

```
local v = Vector4.Create(1, 4, 9)
v:RecipSqrt(v)
print(v)        -- [1, 0.5, 0.333333]
```

See Also

- Vector4.Recip
- Vector4.Sqrt

Vector4.Rotate

Brief

Rotate a vector about and axis.

Definition

```
Vector4.Rotate( Vector4 destination, Vector4 source, Vector4 axis, number angle )
```

Arguments

destination - Vector to store the result of the rotation.source - Vector to be rotated.axis - Axis to rotate about.angle - Angle (in degrees) to rotate by.

Return Values

Description

Rotate a vector about and axis.

Examples

```
src = Vector4.Create( 12, 2, -3 )
axis = Vector4.Create( 0, 0.707, -0.707 )
dst = Vector4.Create()
Vector4.Rotate( dst, src, axis, 45 )
```

See Also

- Vector4.Create

Vector4.SetBroadcast

Brief

Sets the all of the vector's elements to the specified value.

Definition

```
Vector4 Vector4.SetBroadcast( Vector4 vector, number value )
```

Arguments

vector - Vector to be set.value - The value to broadcast to all the elements of the vector.

Return Values

None.

Description

Sets the all of the vector's elements to the specified value.

Examples

```
vecHalf = Vector4.Create()
Vector4.SetBroadcast( vecHalf, 0.5 )

vec = Vector4.Create( 1, 2, 3 )
Vector4.Multiply( vec, vec, vecHalf )
```

See Also

- Vector4.Create
- Vector4.CreateBroadcast

Vector4.SetW

Brief

Sets the w value of the vector.

Definition

```
Vector4.SetW( Vector4 vector, number value )
```

Arguments

vector - Vector to be set. value - A numerical value that is copied into the w component of the vector.

Return Values

None.

Description

Sets the w value of the vector.

Examples

```
vec = Vector4.Create( 1, 2, 3, 4 )
Vector4.SetW( vec, 6 )
```

See Also

- Vector4.Create
- Vector4.SetX
- Vector4.SetY
- Vector4.SetZ

Vector4.SetX

Brief

Sets the x value of the vector.

Definition

```
Vector4.SetX( Vector4 vector, number value )
```

Arguments

vector - Vector to be set. value - A numerical value that is copied into the x component of the vector.

Return Values

None.

Description

Sets the x value of the vector.

Examples

```
vec = Vector4.Create( 1, 2, 3, 4 )
Vector4.SetX( vec, 6 )
```

See Also

- Vector4.Create
- Vector4.SetY
- Vector4.SetZ
- Vector4.SetW

Vector4.SetXyzw

Brief

Sets a Vector4 object with the specified values.

Definition

```
Vector4.SetXyzw( Vector4 vector, number x = 0, number y = 0, number z = 0, number w = 0 )
```

Arguments

vector - Vector to be set.x - Number to be written into the first element of the vector (defaults to 0).y - Number to be written into the second element of the vector (defaults to 0).z - Number to be written into the third element of the vector (defaults to 0).w - Number to be written into the fourth element of the vector (defaults to 0).

Return Values

None.

Description

Sets a Vector4 object with the specified values.

Examples

```
vec = Vector4.Create()
Vector4.SetXyzw( vec, 1, 2, 3, 4 )
```

See Also

- Vector4.Create

Vector4.SetY

Brief

Sets the y value of the vector.

Definition

```
Vector4.SetY( Vector4 vector, number value )
```

Arguments

vector - Vector to be set.value - A numerical value that is copied into the y component of the vector.

Return Values

None.

Description

Sets the y value of the vector.

Examples

```
vec = Vector4.Create( 1, 2, 3, 4 )
Vector4.SetY( vec, 6 )
```

See Also

- Vector4.Create
- Vector4.SetX
- Vector4.SetZ
- Vector4.SetW

Vector4.SetZ

Brief

Sets the z value of the vector.

Definition

```
Vector4.SetZ( Vector4 vector, number value )
```

Arguments

vector - Vector to be set. value - A numerical value that is copied into the z component of the vector.

Return Values

None.

Description

Sets the z value of the vector.

Examples

```
vec = Vector4.Create( 1, 2, 3, 4 )
Vector4.SetZ( vec, 6 )
```

See Also

- Vector4.Create
- Vector4.SetX
- Vector4.SetY
- Vector4.SetW

Vector4.Sign

Brief

Compute the sign (-1 or 1) of all elements in a vector.

Definition

```
Vector4.Sign(Vector4 dest, Vector4 v)
```

Arguments

dest - destination vector to store result.v - input vector.

Return Values

None.

Description

Each element of dest is set to the sign (-1 or 1) of the equivalent element of v. 0 is considered positive and thus becomes 1, if a separate 0 value is required use the Signum function.

Examples

```
local v = Vector4.Create(-3.1, 0, -0.4, 1.2)
v:Sign(v)
print(v)        -- [-1, 1, -1, 1]
```

See Also

- Vector4.Signum

Vector4.Signum

Brief

Compute the sign (-1, 0 or 1) of all elements in a vector.

Definition

```
Vector4.Signum(Vector4 dest, Vector4 v)
```

Arguments

dest - destination vector to store result.v - input vector.

Return Values

None.

Description

Each element of dest is set to the sign (-1, 0 or 1) of the equivalent element of v. If a separate 0 value is not required, use the Sign function.

Examples

```
local v = Vector4.Create(-3.1, 0, -0.4, 1.2)
v:Signum(v)
print(v)        -- [-1, 0, -1, 1]
```

See Also

- Vector4.Sign

Vector4.Sqrt

Brief

Compute the component-wise square root of a vector.

Definition

```
Vector4.Sqrt(Vector4 dest, Vector4 v)
```

Arguments

dest - destination vector to store result.v - input vector.

Return Values

None.

Description

Sets dest to [math.sqrt(v:X()), math.sqrt(v:Y()), math.sqrt(v:Z()), math.sqrt(v:W())]. If any elements of v are < 0 the result is undefined.

Examples

```
local v = Vector4.Create(1, 4, 9)
v:Sqrt(v)
print(v)        -- [1, 2, 3]
```

See Also

- Vector4.RecipSqrt

Vector4.Subtract

Brief

Subtracts one vector from another.

Definition

```
Vector4.Subtract( Vector4 destination, Vector4|number source1, Vector4|number source2 )
```

Arguments

destination - Vector to store the result of the subtraction.source1 - The minuend for the subtraction.source2 - The subtrahend for the subtraction.

Return Values

None.

Description

Subtracts source2 from source1 storing the result in destination.

Descriptive notation:

destination.x = source1.x|source1 - source2.x|source2

destination.y = source1.y|source1 - source2.y|source2

destination.z = source1.z|source1 - source2.z|source2

destination.w = source1.w|source1 - source2.w|source2

Examples

```
result = Vector4.Create()
v1 = Vector4.Create( 1, 2, 3, 4 )
v2 = Vector4.Create( 4, 3, 2, 1 )
Vector4.Subtract( result, v1, v2 )
Vector4.Subtract( result, v1, 3 )
```

See Also

- Vector4.Create

Vector4.Transform

Brief

Transforms a vector by a matrix.

Definition

```
Vector4.Transform( Vector4 destination, Matrix44 matrix, Vector4 source )
Vector4.Transform( table destination, Matrix44 matrix, table source )
```

Arguments

destination - Vector or table of vectors to store the result of the multiplication.matrix - Matrix to transform the source vector or table of vectors.source - Vector or table of vectors to be transformed.

Return Values

Description

Transforms the specified source vector by a matrix and stores the resultant vector in the destination. It is equivalent to:

destination = matrix * source
It is also possible to transform a table of vectors by the specified matrix. The result is stored in a destination table with the same number of vectors as the source table. This can be considerably faster than transforming vectors individually.

Examples

```
-- Create the matrix for the example code below
matrix = Matrix44.Create()
Matrix44.SetRotationXyz( matrix, 45, 12, 7 )

-- Example 1: Transform a single vector by a matrix
src = Vector4.Create( 12, 2, -3, 1 )
dst = Vector4.Create()
Vector4.Transform( dst, matrix, src )

-- Example 2: Transform a table of vectors by a matrix
source = {}
destination = {}
for v = 1, 10 do
    source[v] = Vector4.Create(v, v, v, v)
    destination[v] = Vector4.Create()
end
Vector4.Transform( destination, matrix, source )
```

See Also

- Matrix44.Create
- Vector4.Create

Vector4.TransformPoint

Brief

Transforms a point by a matrix.

Definition

```
Vector4.TransformPoint( Vector4 destination, Matrix44 matrix, Vector4 source )
Vector4.TransformPoint( table destination, Matrix44 matrix, table source )
```

Arguments

destination - Point or table of points to store the result of the multiplication.matrix - Matrix to transform the source vector or table of vectors.source - Point or table of points to be transformed.

Return Values

Description

Transforms the specified source point by a matrix and stores the resultant point in the destination. It is equivalent to:

destination = matrix * source (where source.w is substituted with 1.0)

It is also possible to transform a table of points by the specified matrix. The result is stored in a destination table with the same number of points as the source table. This can be considerably faster than transforming points individually.

Examples

```
-- Create the matrix for the example code below
matrix = Matrix44.Create()
Matrix44.SetRotationXyz( matrix, 45, 12, 7 )

-- Example 1: Transform a single point by a matrix
src = Vector4.Create( 12, 2, -3 )
dst = Vector4.Create()
Vector4.TransformPoint( dst, matrix, src )

-- Example 2: Transform a table of points by a matrix
source = {}
destination = {}
for v = 1, 10 do
    source[v] = Vector4.Create(v, v, v)
    destination[v] = Vector4.Create()
end
Vector4.TransformPoint( destination, matrix, source )
```

See Also

- Matrix44.Create
- Vector4.Create

Vector4.TransformVector

Brief

Transforms a vector by a matrix.

Definition

```
Vector4.TransformVector( Vector4 destination, Matrix44 matrix, Vector4 source )
Vector4.TransformVector( table destination, Matrix44 matrix, table source )
```

Arguments

destination - Vector or table of vectors to store the result of the multiplication.matrix - Matrix to transform the source vector or table of vectors.source - Vector or table of vectors to be transformed.

Return Values

Description

Transforms the specified source vector by a matrix and stores the resultant vector in the destination. It is equivalent to:

destination = matrix * source (where source.w is substituted with 0.0)

It is also possible to transform a table of vectors by the specified matrix. The result is stored in a destination table with the same number of vectors as the source table. This can be considerably faster than transforming vectors individually.

Examples

```
-- Create the matrix for the example code below
matrix = Matrix44.Create()
Matrix44.SetRotationXyz( matrix, 45, 12, 7 )

-- Example 1: Transform a single vector by a matrix
src = Vector4.Create( 12, 2, -3 )
dst = Vector4.Create()
Vector4.TransformVector( dst, matrix, src )

-- Example 2: Transform a table of vectors by a matrix
source = {}
destination = {}
for v = 1, 10 do
    source[v] = Vector4.Create(v, v, v)
    destination[v] = Vector4.Create()
end
Vector4.TransformVector( destination, matrix, source )
```

See Also

- Matrix44.Create
- Vector4.Create

Vector4.W

Brief

Returns the w value of the vector.

Definition

```
number Vector4.W( Vector4 vector )
```

Arguments

vector - Vector to be queried.

Return Values

Returns the fourth element of the vector, the w component.

Description

Returns the w value of the vector.

Examples

```
vec = Vector4.Create( 1, 2, 3, 4 )
print( "Value of W is: " .. Vector4.W( vec ) )
```

See Also

- Vector4.Create
- Vector4.X
- Vector4.Y
- Vector4.Z

Vector4.X

Brief

Returns the x value of the vector.

Definition

```
number Vector4.X( Vector4 vector )
```

Arguments

vector - Vector to be queried.

Return Values

The first element of the vector, the x component.

Description

Returns the x value of the vector.

Examples

```
vec = Vector4.Create( 10 )
print( "Value of X is: " .. Vector4.X( vec ) )
```

See Also

- Vector4.Create
- Vector4.Y
- Vector4.Z
- Vector4.W

Vector4.Y

Brief

Returns the y value of the vector.

Definition

```
number Vector4.Y( Vector4 vector )
```

Arguments

vector - Vector to be queried.

Return Values

Returns the second element of the vector, the y component.

Description

Returns the y value of the vector.

Examples

```
vec = Vector4.Create( 1, 2 )
print( "Value of Y is: " .. Vector4.Y( vec ) )
```

See Also

- Vector4.Create
- Vector4.X
- Vector4.Z
- Vector4.W

Vector4.Z

Brief

Returns the z value of the vector.

Definition

```
number Vector4.Z( Vector4 vector )
```

Arguments

vector - Vector to be queried.

Return Values

Returns the third element of the vector, the z component.

Description

Returns the z value of the vector.

Examples

```
vec = Vector4.Create( 1, 2, 3 )
print( "Value of Z is: " .. Vector4.Z( vec ) )
```

See Also

- Vector4.Create
- Vector4.X
- Vector4.Y
- Vector4.W

View

View.CalculatePickRay

Brief

Calculate a pick ray through a specified screen point into the scene.

Definition

```
boolean View.CalculatePickRay(Vector4 screenPoint, Vector4 rayOrigin, Vector4 rayDirection)
boolean View.CalculatePickRay(number x, number y, Vector4 rayOrigin, Vector4 rayDirection)
```

Arguments

screenPoint - Screen point to calculate pick ray for.rayOrigin - Vector4 to return ray origin into.rayDirection - Vector4 to return ray direction vector into.x - Screen point X coordinate, in the range [0 (left) .. 1280 (right)].y - Screen point Y coordinate, in the range [0 (top) .. 720 (bottom)].

Return Values

True on success, false on failure (only in case of no valid camera view active).

Description

This function will calculate the origin and direction of a ray through the specified screen-space point into the scene, based on the camera properties. It is effectively the inverse operation to View.WorldToScreen, except that the ray indicates all the (infinite) points that map onto the specified screen-space point. The ray origin will be coplanar with the near clip plane, the ray direction is not normalized but is rather scaled to the distance between the near and far clip planes.

Examples

```
RETICLE_OFFSET_X = 0
RETICLE_OFFSET_Y = 10
local pickOrigin = Vector4.Create()
local pickDir = Vector4.Create()
if (View.CalculatePickRay(RETICLE_OFFSET_X, RETICLE_OFFSET_Y, pickOrigin, pickDir)) then
    g_raycast = Collision.CreateRaycast()
    g_raycast:Start(pickOrigin, pickOrigin + pickDir)
    while (not g_raycast:IsReady()) do
        coroutine.yield()
    end
    if (g_raycast:HasHit()) then
        local entity = g_raycast:GetHitEntity()
        if (entity and entity:GetUniqueId() == targetEntityId) then
            print("You hit the target!")
            score = score + 10
        end
    end
end
```

See Also

- View.WorldToScreen
- Collision.CreateRaycast

- Raycast.Start

View.WorldToScreen

Brief

Converts a 3d world position to a screen position.

Definition

```
boolean View.WorldToScreen( Vector4 screenPosition, Vector4 worldPosition )
```

Arguments

screenPosition - The 2D screen position to write the result to. The screen space coordinates are in the range [0,0] (top left) -> [1280,720] (bottom right). Z is set to 0, and W is set to 1.worldPosition - The 3d position in the world.

Return Values

Returns true if it could calculate it, even if it is off screen.

Description

Converts a 3d world position to a 2d screen position if possible. It can return screen coordinates that are actually off screen as you may want to render text for instance that is partially visible.

Examples

```
if View.WorldToScreen( screenPos, worldPos ) == true then
    RenderMyText( screenPos, 'hello' )
end
```

See Also

None

Wardrobe

Wardrobe.ResetCommercePoints

Brief

Reset all commerce point URLs back to default values.

Definition

```
Wardrobe.ResetCommercePoints()
```

Arguments

None

Return Values

None.

Description

Reset all commerce point URLs back to default values.

Examples

```
Wardrobe.ResetCommercePoints()
```

See Also

- Wardrobe.SetCommercePoint

Wardrobe.SetCommercePoint

Brief

Set the commerce point URL for a specific gender/clothing-category combination.

Definition

```
Wardrobe.SetCommercePoint( enum genderId, enum categoryId, string url )
```

Arguments

genderId - A WardrobeGender enum.categoryId - A WardrobeCategory enum.url - The URL of the commerce point definition file.

Return Values

None.

Description

Set the commerce point URL for a specific gender/clothing-category combination. Available gender ids are:
WardrobeGender.Male
WardrobeGender.Female Available category ids are:
WardrobeCategory.All
WardrobeCategory.Hats
WardrobeCategory.Torso
WardrobeCategory.Legs
WardrobeCategory.Hands
WardrobeCategory.Feet
WardrobeCategory.Jewellery
WardrobeCategory.Glasses
WardrobeCategory.Headphones
WardrobeCategory.Outfits If the URL starts with "http:", "https:", etc, it will be considered absolute; otherwise, it will be considered relative to the content server CommercePoints directory.

Examples

```
Wardrobe.SetCommercePoint( WardrobeGender.Male, WardrobeCategory.Torso,
"http://absolute.path.to/commercepoint.xml" )
Wardrobe.SetCommercePoint( WardrobeGender.Female, WardrobeCategory.Hands,
"relative/path/to/commercepoint.xml" )
```

See Also

- Wardrobe.ResetCommercePoints

Xml

Xml.Create

Brief

Creates a new instance of an XML parser.

Definition

```
Xml Xml.Create()
```

Arguments

None

Return Values

A new instance of an XML parser.

Description

Creates a new instance of an XML parser.

Examples

```
local parser = Xml.Create()
```

See Also

- Xml.SetData

Xml.FindAttribute

Brief

Finds a named attribute, or iterates through the available attributes.

Definition

```
boolean Xml.FindAttribute( Xml parser, string attributeName )
```

Arguments

parser - The instance of the XML parser.attributeName - The attribute to search for, or nil to iterate all attributes.

Return Values

True if the attribute was found. If iterating, returns true if there are further attributes or
false if the iteration is complete.

Description

If a valid name is specified, finds the named attribute of the current element. If nil is passed as the name, the function will iterate the attributes in an implementation-defined order with each call. The iteration is complete when the function returns false.

Examples

```
local found = Xml.FindAttribute( parser, "type" )
while ( parser2:FindAttribute( nil ) ) do
    print( parser2:GetAttributeValueText() )
end
```

See Also

None

Xml.FindElement

Brief

Finds the first element that matches the named element.

Definition

```
boolean Xml.FindElement( Xml parser, string elementName )
```

Arguments

parser - The instance of the XML parser.elementName - The name of the XML element to search for. Can be nil.

Return Values

true if the element was found.

Description

Finds the first element at the current level that matches the named element. If nil is passed in as the element name, then the first element is found.

Examples

```
local found = Xml.FindElement( parser, "xml" )
```

See Also

- Xml.Create
- Xml.FindNextElement

Xml.FindNextElement

Brief

Finds the next element that matches the named element.

Definition

```
boolean Xml.FindNextElement( Xml parser, string elementName )
```

Arguments

parser - The instance of the XML parser.elementName - The name of the XML element to search for. Can be nil.

Return Values

true if the element was found.

Description

Finds the next element at the current level that matches the named element. If nil is passed in, then the next element regardless of its name is found.

Examples

```
local found = Xml.FindNextElement( parser, nil )
```

See Also

- Xml.Create
- Xml.FindElement

Xml.GetAttributeName

Brief

Returns the current attribute name, or nil if there is no current attribute.

Definition

```
string Xml.GetAttributeName( Xml parser )
```

Arguments

parser - The instance of the XML parser.

Return Values

The attribute name as a string, or nil if there is no current attribute.

Description

Returns the current attribute name. This will match the last name specified to FindAttribute, or if FindAttribute is used to iterate the attributes, the current attribute name.

Examples

```
while ( parser:FindAttribute( nil ) ) do
    print( parser:GetAttributeName(), "=", parser:GetAttributeValueText() )
end
```

See Also

- Xml.FindAttribute

Xml.GetAttributeValueBool

Brief

Returns the value of the current attribute.

Definition

```
boolean Xml.GetAttributeValueBool( Xml parser )
```

Arguments

parser - The instance of the XML parser.

Return Values

The attribute value as a boolean.

Description

Returns the value of the current attribute.

Examples

```
local value = Xml.GetAttributeValueBool( parser )
```

See Also

None

Xml.GetAttributeValueFloat

Brief

Returns the value of the current attribute.

Definition

```
number Xml.GetAttributeValueFloat( Xml parser )
```

Arguments

parser - The instance of the XML parser.

Return Values

The attribute value as a float.

Description

Returns the value of the current attribute.

Examples

```
local value = Xml.GetAttributeValueFloat( parser )
```

See Also

None

Xml.GetAttributeValueInt

Brief

Returns the value of the current attribute.

Definition

```
number Xml.GetAttributeValueInt( Xml parser )
```

Arguments

parser - The instance of the XML parser.

Return Values

The attribute value as an integer number.

Description

Returns the value of the current attribute.

Examples

```
local value = Xml.GetAttributeValueInt( parser )
```

See Also

None

Xml.GetAttributeValueText

Brief

Returns the value of the current attribute.

Definition

```
string Xml.GetAttributeValueText( Xml parser )
```

Arguments

parser - The instance of the XML parser.

Return Values

The attribute value as a string.

Description

Returns the value of the current attribute.

Examples

```
local value = Xml.GetAttributeValueText( parser )
```

See Also

None

Xml.GetElementName

Brief

Returns the name of the current XML element.

Definition

```
string Xml.GetElementName( Xml parser )
```

Arguments

parser - The instance of the XML parser.

Return Values

The name of the current XML element.

Description

Returns the name of the current XML elemet.

Examples

```
local name = Xml.GetElementName( parser )
```

See Also

None

Xml.GetElementValueBool

Brief

Returns the value of the current XML element.

Definition

```
boolean Xml.GetElementValueBool( Xml parser )
```

Arguments

parser - The instance of the XML parser.

Return Values

The element value as a boolean.

Description

Returns the value of the current XML element.

Examples

```
local value = Xml.GetElementValueBool( parser )
```

See Also

None

Xml.GetElementValueFloat

Brief

Returns the value of the current XML element.

Definition

```
number Xml.GetElementValueFloat( Xml parser )
```

Arguments

parser - The instance of the XML parser.

Return Values

The element value as a float.

Description

Returns the value of the current XML element.

Examples

```
local value = Xml.GetElementValueFloat( parser )
```

See Also

None

Xml.GetElementValueInt

Brief

Returns the value of the current XML element.

Definition

```
number Xml.GetElementValueInt( Xml parser )
```

Arguments

parser - The instance of the XML parser.

Return Values

The element value as an int.

Description

Returns the value of the current XML element.

Examples

```
local value = Xml.GetElementValueInt( parser )
```

See Also

None

Xml.GetElementValueText

Brief

Returns the value of the current XML element.

Definition

```
string Xml.GetElementValueText( Xml parser )
```

Arguments

parser - The instance of the XML parser.

Return Values

The element value as a string.

Description

Returns the value of the current XML element.

Examples

```
local value = Xml.GetElementValueText( parser )
```

See Also

None

Xml.IntoElement

Brief

Navigate into an XML element.

Definition

```
boolean Xml.IntoElement( Xml parser )
```

Arguments

parser - The instance of the XML parser.

Return Values

true if it was possible to navigate into the element.

Description

Navigate into an XML element so that its childen can be examined.

Examples

```
local success = Xml.IntoElement( parser )
```

See Also

- Xml.OutOfElement

Xml.OutOfElement

Brief

Navigate out of the current XML element.

Definition

```
boolean Xml.OutOfElement( Xml parser )
```

Arguments

parser - The instance of the XML parser.

Return Values

true it was possible to navigate out of the current element.

Description

Navigate out of the current XML element.

Examples

```
local success = Xml.OutOfElement( parser )
```

See Also

- Xml.IntoElement

Xml.SetData

Brief

Sets the data for the XML parser.

Definition

```
Xml.SetData( Xml parser, string resourceName )
Xml.SetData( Xml parser, Resource resource)
```

Arguments

parser - The instance of the XML parser.resourceName - The name of the resource in the object containing the XML data.resource - The resource containing the XML data.

Return Values

None.

Description

Sets the data for the XML parser. The specified resource may be the resource of an object or it may be loaded via the Resource library from a URL.

Examples

```
-- Example 1: Set the parser to an XML resource in the object
parser = Xml.Create()
Xml.SetData( parser, "settings" )

-- Example 2: Set the parser to a downloaded XML resource
res = Resource.Request( "http://www.example.com/data.xml", "xml")

-- Do some work while XML resource is downloaded

if ( Resource.IsLoaded( res ) ) then
    Xml.SetData( parser, res )
end
```

See Also

- Resource.IsLoaded
- Resource.Request
- Xml.Create

Xml.SetDataFromContainer

Brief

Sets the data for the XML parser from a memory container.

Definition

```
Xml.SetDataFromContainer( Xml parser, MemoryContainer container )
```

Arguments

parser - The instance of the XML parser.container - The container that holds the XML data.

Return Values

None.

Description

Sets the data for the XML parser from a memory container. This is for testing XML output from the BasicGenx library and has no real practical use.

Examples

```
-- Write some XML to a memory container
container = MemoryContainer.Create( 1024 )
writer = BasicGenx.Create( container )
BasicGenx.StartDocument( writer )
BasicGenx.StartElement( writer, "XML" )
BasicGenx.EndElement( writer )
BasicGenx.EndDocument( writer )

-- Set the XML reader to the output from BasicGenx
Xml.SetDataFromContainer( parser, container )
```

See Also

- BasicGenx.Create
- MemoryContainer.Create

HDK API Reference

HDK API is the full HDK scripting cookbook which allows you to add extra functionality to your tools using IronPython.

Collapse all
Expand all Collapse all

HdkApi.Core.Collections

Summary

Type
Description
Assembly
HdkDictionary
HdkDictionary is a light wrapper around a .NET Dictionary, it provides all the standard functionality of the .NET Dictionary.

Home.Core.Scripting

HdkApi.Core.Collections.HdkDictionary

HdkDictionary

HdkDictionary is a light wrapper around a .NET Dictionary, it provides all the standard functionality of the .NET Dictionary.

Namespace: HdkApi.Core.Collections
Assembly: Home.Core.Scripting.dll

Constructors

HdkDictionary( )

HdkDictionary(resizable, dictionary)

HdkDictionary(equalityComparer)

HdkDictionary(size)

HdkDictionary(resizable, dictionary, equalityComparer)

HdkDictionary(size, equalityComparer)

Methods

Add(key, value)

System.Boolean ContainsKey(key)

System.Boolean Remove(key)

System.Boolean TryGetValue(key, value)

Add(item)

Clear( )

System.Boolean Contains(item)

CopyTo(array, arrayIndex)

System.Boolean Remove(item)

System.Collections.Generic.IEnumerator[System.Collections.Generic.KeyValuePair[T1,T2]] GetEnumerator( )

Properties

Keys

Values

[key]

Count

IsReadOnly

Example

```
#<copyright>
#----------------------------------------------------------------------------------------------------
#   SCE CONFIDENTIAL
#   Home Development Kit (HDK) 1.60.0
#   Copyright (C) 2011 Sony Computer Entertainment Inc.
#   All Rights Reserved.
#----------------------------------------------------------------------------------------------------
#</copyright>

# The HdkDictionary class must be imported from Collections before it is used
from HdkApi.Core.Collections import HdkDictionary

class HdkDictionaryExample():
    def __init__(self):

        # Create a new HdkDictionary mapping int to string
        dictionary = HdkDictionary[int, str]()

        # Add some new entries into the dictionary
        dictionary.Add(1, "red")
        dictionary.Add(2, "two")

        # Edit the value of an entry
        dictionary[1] = "one"

        # Entries can also be added by trying to edit a key which does not yet exist
        dictionary[3] = "three"
        dictionary[4] = "four"

        # Check that the dictionary contains a specific key
        if dictionary.ContainsKey(1):
            # Get the value associated with that key
            print dictionary[1]

            # Remove the entry with the key '1'
            dictionary.Remove(1)

        # Iterate through all the keys in the dictionary
        for key in dictionary.Keys:
            # Get each value
            print dictionary[key]
```

HdkApi.Core.Communication

Summary

Type
Description
Assembly
Target
Provides information about a PS3 target obtained from the TargetManager class. Once connected allows communications such as sending of messages.
Home.Core.Scripting
TargetException

Exception which provides a Result Code for a failed communications operation
Home.Core.Scripting
TargetManager
TargetManager is used to manage and connect to PS3 targets
Home.Core.Scripting
TargetResult
Result codes for operations with the Target Manager, can be accessed through the TargetException's ResultCode
Home.Core.Scripting

HdkApi.Core.Communication.Target

Target

Provides information about a PS3 target obtained from the TargetManager class. Once connected allows communications such as sending of messages.

Namespace: HdkApi.Core.Communication
Assembly: Home.Core.Scripting.dll

Methods

Connect( )

ConnectAsync(connectCompleted)

Disconnect( )

Send(command)

RegisterChannelOutput(channel)

UnregisterChannelOutput(channel)

Properties

Name

Ip

Port

Status

IsConnected

Events

ReceivedResponse

OnConnectionStatusChanged

Delegates

Response

ConnectCompleted

ConnectionStatus

Example

```
#<copyright>
#-------------------------------------------------------------------------------------------------
#   SCE CONFIDENTIAL
#   Home Development Kit (HDK) 1.55.0
#   Copyright (C) 2011 Sony Computer Entertainment Inc.
#   All Rights Reserved.
#-------------------------------------------------------------------------------------------------
#</copyright>

# First import the Communication classes
from HdkApi.Core.Communication import TargetManager, Target, TargetException, TargetResult

# Example of connecting to a target
class TargetManagerExample():
    def __init__(self):
        # Before using the TargetManager API you must call Initialize
        # Initialize automatically connects to the first Target it finds which already has a
'connected' status
        # If there are no targets with a 'connected' status the TargetManager API will remain
disconnected
        TargetManager.Initialize()

        # Check whether the TargetManager is connected to a target
        if TargetManager.ConnectedTarget == None:
            try:
                # Connect to the first available target
                # This function blocks until the connect is completed
                TargetManager.ConnectToDefault()
            except TargetException, e:
                # Catch any exceptions thrown by the connect
                print "TargetManagerExample failed to connect to default target:
{0}".format(e.ResultCode)

                return

        # Send a command enabling rendering of sound volumes in the client
        exampleCmd = "enableAudioRender3d 1"
        TargetManager.ConnectedTarget.Send(exampleCmd)

        # Send a lua command activating the editor camera in the client
        exampleLuaCmd = "lc Editor.ActivateEditorCamera()"
        TargetManager.ConnectedTarget.Send(exampleLuaCmd)

# Example of asynchronously connecting to a target
class TargetManagerAsyncExample():
    def __init__(self):
        # Before using the TargetManager API you must call Initialize
        TargetManager.Initialize()

        # Check whether the TargetManager is connected to a target
        if TargetManager.ConnectedTarget == None:
            try:
                # Connect to the first available target asynchronously
                # The ConnectCallback function will be called when the connect has completed
                TargetManager.ConnectToDefaultAsync(ConnectCallback)
            except TargetException, e:
                # Catch any exceptions thrown before the asynchronous connect begins
                # Any exceptions thrown will cause the connect to be aborted so the ConnectCallback
will never be called
                print "TargetManagerAsyncExample failed to connect to default target:
{0}".format(e.ResultCode)
```

```
            return
    else:
        # If the target is already connected proceed with sending the commands
        self.SendCommands()

def SendCommands(self):
    # Send a command enabling rendering of sound volumes in the client
    exampleCmd = "enableAudioRender3d 1"
    TargetManager.ConnectedTarget.Send(exampleCmd)

    # Send a lua command activating the editor camera in the client
    exampleLuaCmd = "lc Editor.ActivateEditorCamera()"
    TargetManager.ConnectedTarget.Send(exampleLuaCmd)

def ConnectCallback(self, target, result):
```

```
        # Example of a callback function triggered by a completed asynchronous connect
        if result == TargetResult.SUCCEEDED:
            self.SendCommands()
```

HdkApi.Core.Communication.TargetException

TargetException

Exception which provides a Result Code for a failed communications operation

Namespace: HdkApi.Core.Communication
Assembly: Home.Core.Scripting.dll

Properties

ResultCode

Message

HdkApi.Core.Communication.TargetManager

TargetManager

TargetManager is used to manage and connect to PS3 targets

Namespace: HdkApi.Core.Communication
Assembly: Home.Core.Scripting.dll

Remarks

Before attempting to connect to a target TargetManager must be initialised by calling TargetManager.Initialize()
If multiple targets are connected to through the SN Target Manager, only the first connected target will be connected to in the HdkApi TargetManager
TargetManager only supports connecting and communicating with one target at a time

Methods

TargetManager.Initialize( )

System.Collections.Generic.Dictionary[System.String,HdkApi.Core.Communication.Target] TargetManager.ListTargets( )

TargetManager.DisplayTargetsForm(connectCompleted)

TargetManager.ConnectToDefault( )

TargetManager.ConnectToDefaultAsync(connectCompleted)

Properties

ConnectedTarget

Example

```
#<copyright>
#----------------------------------------------------------------------------------------------------
#   SCE CONFIDENTIAL
#   Home Development Kit (HDK) 1.55.0
#   Copyright (C) 2011 Sony Computer Entertainment Inc.
#   All Rights Reserved.
```

```
#-------------------------------------------------------------------------------------------------
#</copyright>

# First import the Communication classes
from HdkApi.Core.Communication import TargetManager, Target, TargetException, TargetResult

# Example of connecting to a target
class TargetManagerExample():
    def __init__(self):
        # Before using the TargetManager API you must call Initialize
        # Initialize automatically connects to the first Target it finds which already has a
'connected' status
        # If there are no targets with a 'connected' status the TargetManager API will remain
disconnected
        TargetManager.Initialize()

        # Check whether the TargetManager is connected to a target
        if TargetManager.ConnectedTarget == None:
            try:
                # Connect to the first available target
                # This function blocks until the connect is completed
                TargetManager.ConnectToDefault()
            except TargetException, e:
                # Catch any exceptions thrown by the connect
                print "TargetManagerExample failed to connect to default target:
{0}".format(e.ResultCode)

                return

        # Send a command enabling rendering of sound volumes in the client
        exampleCmd = "enableAudioRender3d 1"
        TargetManager.ConnectedTarget.Send(exampleCmd)

        # Send a lua command activating the editor camera in the client
        exampleLuaCmd = "lc Editor.ActivateEditorCamera()"
        TargetManager.ConnectedTarget.Send(exampleLuaCmd)

# Example of asynchronously connecting to a target
class TargetManagerAsyncExample():
    def __init__(self):
        # Before using the TargetManager API you must call Initialize
        TargetManager.Initialize()

        # Check whether the TargetManager is connected to a target
        if TargetManager.ConnectedTarget == None:
            try:
                # Connect to the first available target asynchronously
                # The ConnectCallback function will be called when the connect has completed
                TargetManager.ConnectToDefaultAsync(ConnectCallback)
            except TargetException, e:
                # Catch any exceptions thrown before the asynchronous connect begins
                # Any exceptions thrown will cause the connect to be aborted so the ConnectCallback
will never be called
                print "TargetManagerAsyncExample failed to connect to default target:
{0}".format(e.ResultCode)

                return
        else:
            # If the target is already connected proceed with sending the commands
            self.SendCommands()

    def SendCommands(self):
        # Send a command enabling rendering of sound volumes in the client
        exampleCmd = "enableAudioRender3d 1"
        TargetManager.ConnectedTarget.Send(exampleCmd)

        # Send a lua command activating the editor camera in the client
        exampleLuaCmd = "lc Editor.ActivateEditorCamera()"
        TargetManager.ConnectedTarget.Send(exampleLuaCmd)
```

```
    def ConnectCallback(self, target, result):
```

```
        # Example of a callback function triggered by a completed asynchronous connect
        if result == TargetResult.SUCCEEDED:
            self.SendCommands()
```

HdkApi.Core.Communication.TargetResult

TargetResult

Result codes for operations with the Target Manager, can be accessed through the TargetException's ResultCode

Namespace: HdkApi.Core.Communication
Assembly: Home.Core.Scripting.dll

Fields

SUCCEEDED

TM_FAILED_TO_START

TARGET_COMMS_ERROR

TARGET_IN_USE

TM_NO_TARGETS

TARGET_NOT_FOUND

TARGET_NOT_CONNECTED

TM_NOT_INITIALIZED

FAILED

Example

```
#<copyright>
#-----------------------------------------------------------------------------------------------
#   SCE CONFIDENTIAL
#   Home Development Kit (HDK) 1.55.0
#   Copyright (C) 2011 Sony Computer Entertainment Inc.
#   All Rights Reserved.
#-----------------------------------------------------------------------------------------------
#</copyright>

# First import the Communication classes
from HdkApi.Core.Communication import TargetManager, Target, TargetException, TargetResult

# Example of using Target Result
class TargetResultExample():
    def __init__(self):
        try:
            TargetManager.Initialise()
        except TargetException, e:
            # Print the Target Result for the exception
            print e.ResultCode

            # Check for a specific Target Result
            if e.ResultCode == TargetResult.TM_FAILED_TO_START:
                print "Target Manager failed to start"
                # Run some fallback code
```

HdkApi.Core.Debug

Summary

Type
Description
Assembly
Logger
Provides simple file logging functionality, allowing output from a program to be written to a log file.
Home.Core.Scripting

HdkApi.Core.Debug.Logger

Logger

Provides simple file logging functionality, allowing output from a program to be written to a log file.

Namespace: HdkApi.Core.Debug
Assembly: Home.Core.Scripting.dll

Constructors

Logger(logFilePath)

Methods

ClearLog( )

LogInfo(info)

LogWarning(warning)

LogError(error)

LogFatal(fatal)

Example

```
#<copyright>
#--------------------------------------------------------------------------------------------------
#   SCE CONFIDENTIAL
#   Home Development Kit (HDK) 1.60.0
#   Copyright (C) 2011 Sony Computer Entertainment Inc.
#   All Rights Reserved.
#--------------------------------------------------------------------------------------------------
#</copyright>

# The Logger class must be imported from Debug before it is used
from HdkApi.Core.Debug import Logger

class LoggerExample():
    def __init__(self):

        # Create a new logger which writes to the file C:/MyLog.txt
        logger = Logger("C:/MyLog.txt")

        # Write an information message to the log
        logger.LogInfo("Example started")

        # Write a warning message to the log
        logger.LogWarning("Example warning")

        # Write an error message to the log
        logger.LogError(("Example error"))

        try:
            raise Exception("Example fatal error")
        except Exception as e:
            # Write a fatal error message to the log
            logger.LogFatal(str(e))
```

HdkApi.Core.Maths

Summary

Type	Description	Assembly
Converters	Provides helper functions for performing various mathematical conversions such as converting radian values to degrees.	Home.Core.Scripting
Vector3	A simple vector structure containing 3 float components.	Home.Core.Scripting

HdkApi.Core.Maths.Converters

Converters

Provides helper functions for performing various mathematical conversions such as converting radian values to degrees.

Namespace: HdkApi.Core.Maths
Assembly: Home.Core.Scripting.dll

Methods

System.Double Converters.RadToDeg(radians)

System.Double Converters.DegToRad(degrees)

Example

```
#<copyright>
#-------------------------------------------------------------------------------------------------
#   SCE CONFIDENTIAL
#   Home Development Kit (HDK) 1.60.0
#   Copyright (C) 2011 Sony Computer Entertainment Inc.
#   All Rights Reserved.
#-------------------------------------------------------------------------------------------------
#</copyright>

# The Converters class must be imported from Maths before it is used
from HdkApi.Core.Maths import Converters

class ConvertersExample():
    def __init__(self):

        angleInDegrees = 180

        # Convert our angle in degrees to radians
        angleInRadians = Converters.DegToRad(angleInDegrees)
        print angleInRadians

        # Convert our angle in radians to degrees
        angleInDegrees = Converters.RadToDeg(angleInRadians)
        print angleInDegrees
```

HdkApi.Core.Maths.Vector3

Vector3

A simple vector structure containing 3 float components.

Namespace: HdkApi.Core.Maths
Assembly: Home.Core.Scripting.dll

Constructors

Vector3(x, y, z)

Methods

System.String ToString(format, formatProvider)

Properties

X

Y

Z

Example

```
#<copyright>
#--------------------------------------------------------------------------------------------------
#   SCE CONFIDENTIAL
#   Home Development Kit (HDK) 1.55.0
#   Copyright (C) 2011 Sony Computer Entertainment Inc.
#   All Rights Reserved.
#--------------------------------------------------------------------------------------------------
#</copyright>

# First import the Vector3 class from Maths
from HdkApi.Core.Maths import Vector3

class Vector3Example():
    def __init__(self):
        # Example of creating Vector3 objects
        vecA = Vector3(1.0, 1.0, 1.0)
        vecB = Vector3(1.0, 1.0, 1.0)

        # Various arithmetic operations can be performed on the Vector3 objects
        vecC = vecA + vecB
        vecD = vecA - vecB
        vecE = vecA * vecB
        vecF = vecA * 5
        vecG = vecA / vecB
        vecH = vecA / 2

        # Comparison operations can also be performed
        if vecA == vecB:
            print "Vectors are equal"

        if vecA != vecB:
            print "Vectors are not equal"

        # The X, Y and Z components of a Vector3 can be accessed through properties
        print vecC.X
        print vecC.Y
        print vecC.Z
```

HdkApi.Core.Utilities

Summary

Type
Description
Assembly
PathUtilities
PathUtilities can be used to get the current Project path and the HDK install root. These paths are useful for referring to scripts or other files relative from your project or HDK install.
Home.Core.Scripting

HdkApi.Core.Utilities.PathUtilities

PathUtilities

PathUtilities can be used to get the current Project path and the HDK install root. These paths are useful for referring to scripts or other files relative from your project or HDK install.

Namespace: HdkApi.Core.Utilities
Assembly: Home.Core.Scripting.dll

Properties

ProjectPath

HdkInstallRoot

Example

```
#<copyright>
#-------------------------------------------------------------------------------------------------
#   SCE CONFIDENTIAL
#   Home Development Kit (HDK) 1.55.0
#   Copyright (C) 2011 Sony Computer Entertainment Inc.
#   All Rights Reserved.
#-------------------------------------------------------------------------------------------------
#</copyright>

# The PathUtilities class must be imported from Utilities before it is used
from HdkApi.Core.Utilities import PathUtilities

class PathUtilitiesExample():
    def __init__(self):

        # Get the current project path
        projectPath = PathUtilities.ProjectPath

        textFilePath = System.IO.Path.Combine(projectPath, "myFile.txt")
        print(textFilePath)

        # Get the current hdk install root
        hdkInstallRoot = PathUtilities.HdkInstallRoot

        iconPath = System.IO.Path.Combine(hdkInstallRoot, "dev/scripts/SceneEditor/icons/myFile.png")
        print(iconPath)
```

HdkApi.Data.AgeRating

Summary

Type
Description
Assembly
AgeRatingItem
Provides information about an age rating item
Home.Data.Scripting

HdkApi.Data.AgeRating.AgeRatingItem

AgeRatingItem

Provides information about an age rating item

Namespace: HdkApi.Data.AgeRating
Assembly: Home.Data.Scripting.dll

Additional Information

Country	Code	Region	Rating System
United States	US	SCEA	ESRB
Canada	CA	SCEA	ESRB
United Kingdom	GB	SCEE	PEGI BBFC
Ireland	IE	SCEE	PEGI
Belgium	BE	SCEE	PEGI
Luxembourg	LU	SCEE	PEGI
Netherlands	NL	SCEE	PEGI
France	FR	SCEE	PEGI
Germany	DE	SCEE	USK
Austria	AT	SCEE	PEGI
Switzerland	CH	SCEE	PEGI
Italy	IT	SCEE	PEGI
Portugal	PT	SCEE	PEGI
Denmark	DK	SCEE	PEGI
Finland	FI	SCEE	PEGI
Norway	NO	SCEE	PEGI
Sweden	SE	SCEE	PEGI
Australia	AU	SCEE	ACB
New Zealand	NZ	SCEE	OFLCNZ ACB
Spain	ES	SCEE	PEGI
Russia	RU	SCEE	PEGI
UAE	AE	SCEE	PEGI
South Africa	ZA	SCEE	PEGI
Czech Republic	CZ	SCEE	PEGI
Saudi Arabia	SA	SCEE	PEGI
Poland	PL	SCEE	PEGI
Greece	GR	SCEE	PEGI
Hong Kong	HK	SCEAsia	
Taiwan	TW	SCEAsia	CGSRR
Singapore	SG	SCEAsia	
Malaysia	MY	SCEAsia	
Thailand	TH	SCEAsia	

Indonesia	ID	SCEAsia	
Korea	KR	SCEK	GRB
Japan	JP	SCEJ	SCEJ

Rating System	Rating Code	Description	Descriptor
ESRB	EC E E10+ T M AO RP	Early Childhood Everyone Everyone 10+ Teen Mature Adults Only Rating Pending	Alcohol Reference Animated Blood Blood Blood and Gore Cartoon Violence Comic Mischief Crude Humour Drug Reference Edutainment Fantasy Violence Informational Intense Violence Language Lyrics Mature Humour Mild Blood Mild Cartoon Violence Mild Fantasy Violence Mild Language Mild Lyrics Mild Violence Mild Suggestive Themes Nudity Partial Nudity Real Gambling Sexual Themes Sexual Violence Simulated Gambling Some Assistance May Be Needed Strong Language Strong Lyrics Strong Sexual Content Suggestive Themes Tobacco References Use of Alcohol Use of Drugs Use of Tobacco Violence Online Interactions Not Rated by the ESRB Visit www.esrb.org for rating information. May contain content inappropriate for children.
PEGI	OK 3+ 4+ 6+ 7+ 12+ 16+ 18+	PEGI OK PEGI 3+ PEGI 4+ PEGI 6+ PEGI 7+ PEGI 12+ PEGI 16+ PEGI 18+	Bad Language Discrimination Drugs Fear Sexual Content Violence Gambling Online
BBFC	Uc U PG 12A 15 18 R18	Universal Children Universal Parental Guidance 12 Advisory 15 18 Restricted 18	
USK	0 6 12 16 18	0 6 12 16 18	

ACB	CTC G PG M MA15+	Check The Classification General ParentalGuidance Mature Mature Restricted	Blood and Gore Content May Change When Played Online Crude Humor Dangerous Stunts Mild Battle Violence Mild Coarse Language Mild Drug References Mild Fantasy Violence Mild Gambling References Mild Horror Theme Mild Nudity Mild Science Fiction Violence Mild Sex Scene Mild Sex Scenes Mild Sexual Reference Mild Sexual References Mild Themes Mild Violence Moderate Battle Violence Moderate Coarse Language Moderate Drug References Moderate Drug Use Moderate Fantasy Violence Moderate Horror Themes Moderate Horror Violence Moderate Nudity Moderate Science Fiction Violence Moderate Sex Scene Moderate Sex Scenes Moderate Sexual Reference Moderate Sexual References Moderate Themes Moderate Violence Scary Scenes Strong Battle Violence Strong Coarse Language Strong Drug References Strong Drug Use Strong Fantasy Violence Strong Graphic Violence Strong Horror Themes Strong Horror Violence Strong Nudity Strong Science Fiction Violence Strong Sex Scene Strong Sex Scenes Strong Sexual Reference Strong Sexual References Strong Sexual Violence Strong Themes Strong Violence

OFLCNZ	G PG M R13 R15 R16 R18 R	General ParentalGuidance Mature Restricted 13 Restricted 15 Restricted 16 Restricted 18 Restricted	Adult Themes Anti-Social Behaviour Battle Violence Brutal Violence Content That May Disturb Content That May Offend Could Have a Negative Effect on Children Cruelty Depicts Graphic And Realistic War Scenes Disturbing Content Drug Use Explicit Language Explicit Sex Scenes Extra Material May Offend or Disturb Graphic Animal Cruelty Graphic Violence Horror Scenes Irresponsible Behaviour Associated With Alcohol Nudity Offensive Language Other Content That May Offend Realistic Horror Sadistic Cruelty Sadistic Violence Sex Scenes Sexual Abuse Sexual Abuse Themes Sexual Coercion Sexual Content That May Offend Sexual Material Sexual References Sexual Themes Sexual Violence Strong Offensive Language Suicide Torture Use of Menstrual Blood Violence
GRB	ALL 12 15 18	ALL 12 15 18	Violence Sexual Horror Language Drug Crime Gamble
SCEJ	B C D Z	12+ 15+ 17+ 18+	Love Sexual Content Violence Horror Gambling Crime Use of alcohol or tobacco Use of drugs Language
CGSRR	All Ages 6 and above 12 and above 18 and above	All Ages 6 and above 12 and above 18 and above	

Methods

HdkApi.Data.AgeRating.AgeRatingItem AgeRatingItem.Create(rating)

Properties

Rating

MinimumAge

ParentalControlLevel

ContentDescriptors

AdditionalData

Example

```
#<copyright>
#-------------------------------------------------------------------------------------------------
#   SCE CONFIDENTIAL
#   Home Development Kit (HDK) 1.65.0
#   Copyright (C) 2011 Sony Computer Entertainment Inc.
#   All Rights Reserved.
#-------------------------------------------------------------------------------------------------
#</copyright>

# Import the Object classes
from HdkApi.Data.Object import ObjectManager, Uuid, ComponentTypes

# Import the AgeRatingItem class
from HdkApi.Data.AgeRating import AgeRatingItem

class AgeRatingsExample():
    def __init__(self):

        uuid = Uuid.Create("00000000-00000000-00000000-00000018")
        loadedObject = None

        try:
            # First load the object whose age rating data we wish to edit
            loadedObject = ObjectManager.LoadByUuid(uuid)

        except Exception, e:
            print "Failed to load object '{0}': {1}".format(uuid, e)
            return

        if not loadedObject:
            print "Object '{0}' was not found".format(uuid)
            return

        # An object's age ratings are accessed through its header component
        header = loadedObject.Components[ComponentTypes.Header]

        # The age ratings are returned as a HdkDictionary keyed by the country which each rating
belongs to
        for ageRating in header.AgeRatings:
            print "{0} - {1}".format(ageRating.Key, ageRating.Value.Rating)

        # To create a new age rating use the AgeRatingItem.Create method
        # This method takes in a string containing the rating system and rating
        pegiItem = AgeRatingItem.Create("PEGI_12+")

        # To set the Parental Control Level use the ParentalControlLevel property
        pegiItem.ParentalControlLevel = 8

        # To set the Minimum Age use the MinimumAge property
        pegiItem.MinimumAge = 12

        # To set a particular content descriptor use the ContentDescriptors property
        # This property returns a HdkDictionary of descriptors to booleans
        # By default all descriptors are set to False
        pegiItem.ContentDescriptors["Violence"] = True

        # Once the age rating has been setup it can be assigned to a country
        # If the country does not support the age rating's rating system an exception is thrown
        header.AgeRatings["GB"] = pegiItem

        # The same age rating can be assigned to multiple countries
        header.AgeRatings["FR"] = pegiItem
```

```
# An age rating can also be assigned to all countries which use a particular rating system
header.AgeRatings["ALL-PEGI"] = pegiItem

# To remove an age rating use the Remove method of the HdkDictionary
header.AgeRatings.Remove("ALL-PEGI")
```

```
# To write the changed age ratings to disc use the HomeObject Save method
loadedObject.Save()
```

HdkApi.Data.Collect

Summary

Type
Description
Assembly
Collect
A utility class used to collect dependencies of objects or scenes.
Home.Data.Scripting

HdkApi.Data.Collect.Collect

Collect

A utility class used to collect dependencies of objects or scenes.

Namespace: HdkApi.Data.Collect
Assembly: Home.Data.Scripting.dll

Methods

System.String[] Collect.DependenciesForObject(obj)

System.String[] Collect.DependenciesForObjects(objs)

System.String[] Collect.DependenciesForScene(scene)

System.String[] Collect.DependenciesForScenes(scenes)

Example

```
#<copyright>
#--------------------------------------------------------------------------------------------------
#   SCE CONFIDENTIAL
#   Home Development Kit (HDK) 1.70.0
#   Copyright (C) 2011 Sony Computer Entertainment Inc.
#   All Rights Reserved.
#--------------------------------------------------------------------------------------------------
#</copyright>
from HdkApi.Data.Object import ObjectManager
from HdkApi.Data.Collect import Collect
import zipfile


class CollectExample():
 def __init__(self):
  components = ["network"]
  foundObjects = ObjectManager.SearchByComponents(components)
  dependentFiles = Collect.DependenciesForObjects(foundObjects)

  f = zipfile.ZipFile('d:\\collect_test.zip', 'w')
  for dependentFile in dependentFiles:
   f.write(dependentFile)
  f.close()

collectExample = CollectExample()
```

Associated Types

- HdkApi.Data.Object.Uuid

HdkApi.Data.Object

Summary

Type
Description
Assembly
ObjectComponent
Provides information about an object component
Home.Data.Scripting
ComponentTypes
Lists all of the object component types
Home.Data.Scripting
HomeObject
Provides information about an object loaded through the ObjectManager class. Allows an object's data to be viewed and edited.

Home.Data.Scripting
LocalisationItem
Home.Data.Scripting
ObjectManager
ObjectManager is used to load objects for viewing and editing
Home.Data.Scripting
ObjectResource
Provides information about a resource on disk which may be associated with an object and used by a component
Home.Data.Scripting
TerritoryTypes
Lists all of the territory types
Home.Data.Scripting
Uuid
Uuid object used to identify an object. To create a new Uuid use Uuid.Create()
Home.Data.Scripting

HdkApi.Data.Object.ComponentTypes

ComponentTypes

Lists all of the object component types

Namespace: HdkApi.Data.Object
Assembly: Home.Data.Scripting.dll

Remarks

Each type is used to look up a specific component in the Components dictionary accessed through an Object's Components property

Constructors

ComponentTypes( )

Fields

Active

ArcadeGame

Clothing

EmbeddedObject

Furniture

GameLaunch

Header

MiniGame

Portable

Targetable

LuaEnvironment

Entity

Particles

Network

EventTimer

Osd

Pad

Renderer

Screen

Camera

System

GameLaunchProperties

Light

Repertoire

GameSpawner

RealTimeGame

GroupDoor

ResourcePack

MemorySlots

SceneTransition

Associated Types

- HdkApi.Data.Object.HomeObject
- HdkApi.Data.Object.ObjectComponent

HdkApi.Data.Object.HomeObject

HomeObject

Provides information about an object loaded through the ObjectManager class. Allows an object's data to be viewed and edited.

Namespace: HdkApi.Data.Object
Assembly: Home.Data.Scripting.dll

Methods

System.Boolean Validate(validatorOutput)

System.Boolean Package(validatorOutput, packagerOutput)

Save( )

System.Collections.Generic.List[HdkApi.Data.Object.ObjectResource] GetResources( )

System.Collections.Generic.List[HdkApi.Data.Object.ObjectResource] GetResources(type)

HdkApi.Data.Object.ObjectResource AddResource(name, type, filepath)

RemoveResource(resource)

System.Collections.Generic.List[HdkApi.Data.Object.LocalisationItem] GetLocalisationItems( )

HdkApi.Data.Object.LocalisationItem GetLocalisationItem(key)

HdkApi.Data.Object.LocalisationItem AddLocalisationItem(reference)

RemoveLocalisationItem(item)

Properties

Uuid

Name

Components

EntitlementId

CategoryId

ProductId

Dependencies

Example

```
#<copyright>
#----------------------------------------------------------------------------------------------------
#   SCE CONFIDENTIAL
#   Home Development Kit (HDK) 1.55.0
#   Copyright (C) 2011 Sony Computer Entertainment Inc.
#   All Rights Reserved.
#----------------------------------------------------------------------------------------------------
#</copyright>

# First import the Object classes
from HdkApi.Data.Object import ObjectManager, HomeObject, ObjectComponent, Uuid, ComponentTypes,
TerritoryTypes

# Import the MessageType class as we need this for the object validation and packaging output
from HdkApi.Data.Reporting import MessageType

# Also import PathUtilities as we use it to get the project path
from HdkApi.Core.Utilities import PathUtilities

class ObjectManagerExample():
    def __init__(self):
        try:
            # Load an object by its uuid
            uuid = Uuid.Create("00000000-00000000-00000000-00000018")
            loadedObject = ObjectManager.LoadByUuid(uuid)

            # Load an object by its object.xml path
            objectPath = PathUtilities.ProjectPath +
"/build/objects/00000000-00000000-00000000-00000018/object.xml"
            loadedObject = ObjectManager.LoadByFilename(objectPath)
```

```
    except Exception, e:
        print "Failed to load object {0}: {1}".format(uuid, e)

    # Get the Uuids of all objects which contain either a 'furniture' or an 'active' component
    components = ["furniture", "active"]
    foundObjects = ObjectManager.SearchByComponents(components)

    # Get the Uuids of all objects which contain a resource with the path 'resourcePath'
    resourcePath = "objects/00000000-00000000-00000000-00000018/main.lua"
    foundObjects = ObjectManager.SearchByResource(resourcePath)

    # Print the entitlement_id for the territory 'SCEA'
    print loadedObject.EntitlementId[TerritoryTypes.SCEA]

    # Set the entitlement_id for the territory 'SCEE'
    loadedObject.EntitlementId[TerritoryTypes.SCEE] = "LUA_REWARD"

    # Print the name of each component which loadedObject contains
    for component in loadedObject.Components.Keys:
        print component

    # Try to access the furniture component and set the furniture metadata item
    if loadedObject.Components.ContainsKey(ComponentTypes.Furniture):
        loadedObject.Components[ComponentTypes.Furniture].Furniture = "Chair"

    # Validate an object
    if loadedObject.Validate(self.Output):
        print "{0} - {1} passed validations\n".format(loadedObject.Name, loadedObject.Uuid)

    # Validate and package an object
    # The first passed in callback is triggered for validation output
    # The second callback is triggered for packager output
    if loadedObject.Package(self.Output, self.Output):
        print "{0} - {1} succesfully packaged\n".format(loadedObject.Name, loadedObject.Uuid)

# Example function for handling object validation and packager output
def Output(self, msg, msgType):
    if (msgType == MessageType.INFO):
        print "INFO: {0}\n".format(msg)

    elif (msgType == MessageType.WARNING):
        print "WARNING: {0}\n".format(msg)

    elif (msgType == MessageType.ERROR):
        print "ERROR: {0}\n".format(msg)

    elif (msgType == MessageType.FATAL):
```

```
            print "FATAL: {0}\n".format(msg)

objectManagerExample = ObjectManagerExample()
```

Associated Types

- HdkApi.Data.Object.ObjectManager
- HdkApi.Data.Object.ObjectComponent

HdkApi.Data.Object.LocalisationItem

LocalisationItem

Namespace: HdkApi.Data.Object
Assembly: Home.Data.Scripting.dll

Methods

System.Boolean HasLanguage(language)

System.String GetPhrase(language)

SetPhrase(language, phrase)

HdkApi.Core.Collections.HdkDictionary[System.String,System.String] GetPhrases( )

Properties

Key

SupportedLanguages

Associated Types

- HdkApi.Data.Object.ObjectManager
- HdkApi.Data.Object.HomeObject

HdkApi.Data.Object.ObjectComponent

ObjectComponent

Provides information about an object component

Namespace: HdkApi.Data.Object
Assembly: Home.Data.Scripting.dll

Properties

Type

Example

```
#<copyright>
#----------------------------------------------------------------------------------------------
#   SCE CONFIDENTIAL
```

```
#   Home Development Kit (HDK) 1.55.0
#   Copyright (C) 2011 Sony Computer Entertainment Inc.
#   All Rights Reserved.
#-----------------------------------------------------------------------------------------------
#</copyright>

# First import the Object classes
from HdkApi.Data.Object import ObjectManager, HomeObject, ObjectComponent, Uuid, ComponentTypes,
TerritoryTypes

# Import the MessageType class as we need this for the object validation and packaging output
from HdkApi.Data.Reporting import MessageType

# Also import PathUtilities as we use it to get the project path
from HdkApi.Core.Utilities import PathUtilities

class ObjectManagerExample():
    def __init__(self):
        try:
            # Load an object by its uuid
            uuid = Uuid.Create("00000000-00000000-00000000-00000018")
            loadedObject = ObjectManager.LoadByUuid(uuid)

            # Load an object by its object.xml path
            objectPath = PathUtilities.ProjectPath +
"/build/objects/00000000-00000000-00000000-00000018/object.xml"
            loadedObject = ObjectManager.LoadByFilename(objectPath)

        except Exception, e:
            print "Failed to load object {0}: {1}".format(uuid, e)

        # Get the Uuids of all objects which contain either a 'furniture' or an 'active' component
        components = ["furniture", "active"]
        foundObjects = ObjectManager.SearchByComponents(components)

        # Get the Uuids of all objects which contain a resource with the path 'resourcePath'
        resourcePath = "objects/00000000-00000000-00000000-00000018/main.lua"
        foundObjects = ObjectManager.SearchByResource(resourcePath)

        # Print the entitlement_id for the territory 'SCEA'
        print loadedObject.EntitlementId[TerritoryTypes.SCEA]

        # Set the entitlement_id for the territory 'SCEE'
        loadedObject.EntitlementId[TerritoryTypes.SCEE] = "LUA_REWARD"

        # Print the name of each component which loadedObject contains
        for component in loadedObject.Components.Keys:
            print component

        # Try to access the furniture component and set the furniture metadata item
        if loadedObject.Components.ContainsKey(ComponentTypes.Furniture):
            loadedObject.Components[ComponentTypes.Furniture].Furniture = "Chair"

        # Validate an object
        if loadedObject.Validate(self.Output):
            print "{0} - {1} passed validations\n".format(loadedObject.Name, loadedObject.Uuid)

        # Validate and package an object
        # The first passed in callback is triggered for validation output
        # The second callback is triggered for packager output
        if loadedObject.Package(self.Output, self.Output):
            print "{0} - {1} succesfully packaged\n".format(loadedObject.Name, loadedObject.Uuid)

    # Example function for handling object validation and packager output
    def Output(self, msg, msgType):
        if (msgType == MessageType.INFO):
            print "INFO: {0}\n".format(msg)

        elif (msgType == MessageType.WARNING):
            print "WARNING: {0}\n".format(msg)
```

```
        elif (msgType == MessageType.ERROR):
            print "ERROR: {0}\n".format(msg)

        elif (msgType == MessageType.FATAL):
```

```
            print "FATAL: {0}\n".format(msg)

objectManagerExample = ObjectManagerExample()
```

Associated Types

- HdkApi.Data.Object.HomeObject

HdkApi.Data.Object.ObjectManager

ObjectManager

ObjectManager is used to load objects for viewing and editing

Namespace: HdkApi.Data.Object
Assembly: Home.Data.Scripting.dll

Methods

HdkApi.Data.Object.HomeObject ObjectManager.LoadByFilename(objectFilename)

HdkApi.Data.Object.HomeObject ObjectManager.LoadByUuid(uuidToLoad)

System.Collections.Generic.IEnumerable[HdkApi.Data.Object.Uuid] ObjectManager.SearchByComponents(components)

System.Collections.Generic.IEnumerable[HdkApi.Data.Object.Uuid] ObjectManager.SearchByResource(resourcePath)

Example

```
#<copyright>
#----------------------------------------------------------------------------------------------------
#   SCE CONFIDENTIAL
#   Home Development Kit (HDK) 1.55.0
#   Copyright (C) 2011 Sony Computer Entertainment Inc.
#   All Rights Reserved.
#----------------------------------------------------------------------------------------------------
#</copyright>

# First import the Object classes
from HdkApi.Data.Object import ObjectManager, HomeObject, ObjectComponent, Uuid, ComponentTypes,
TerritoryTypes

# Import the MessageType class as we need this for the object validation and packaging output
from HdkApi.Data.Reporting import MessageType

# Also import PathUtilities as we use it to get the project path
from HdkApi.Core.Utilities import PathUtilities

class ObjectManagerExample():
    def __init__(self):
        try:
            # Load an object by its uuid
            uuid = Uuid.Create("00000000-00000000-00000000-00000018")
            loadedObject = ObjectManager.LoadByUuid(uuid)

            # Load an object by its object.xml path
            objectPath = PathUtilities.ProjectPath +
"/build/objects/00000000-00000000-00000000-00000018/object.xml"
            loadedObject = ObjectManager.LoadByFilename(objectPath)

        except Exception, e:
            print "Failed to load object {0}: {1}".format(uuid, e)
```

```
    # Get the Uuids of all objects which contain either a 'furniture' or an 'active' component
    components = ["furniture", "active"]
    foundObjects = ObjectManager.SearchByComponents(components)

    # Get the Uuids of all objects which contain a resource with the path 'resourcePath'
    resourcePath = "objects/00000000-00000000-00000000-00000018/main.lua"
    foundObjects = ObjectManager.SearchByResource(resourcePath)

    # Print the entitlement_id for the territory 'SCEA'
    print loadedObject.EntitlementId[TerritoryTypes.SCEA]

    # Set the entitlement_id for the territory 'SCEE'
    loadedObject.EntitlementId[TerritoryTypes.SCEE] = "LUA_REWARD"

    # Print the name of each component which loadedObject contains
    for component in loadedObject.Components.Keys:
        print component

    # Try to access the furniture component and set the furniture metadata item
    if loadedObject.Components.ContainsKey(ComponentTypes.Furniture):
        loadedObject.Components[ComponentTypes.Furniture].Furniture = "Chair"

    # Validate an object
    if loadedObject.Validate(self.Output):
        print "{0} - {1} passed validations\n".format(loadedObject.Name, loadedObject.Uuid)

    # Validate and package an object
    # The first passed in callback is triggered for validation output
    # The second callback is triggered for packager output
    if loadedObject.Package(self.Output, self.Output):
        print "{0} - {1} succesfully packaged\n".format(loadedObject.Name, loadedObject.Uuid)

# Example function for handling object validation and packager output
def Output(self, msg, msgType):
    if (msgType == MessageType.INFO):
        print "INFO: {0}\n".format(msg)

    elif (msgType == MessageType.WARNING):
        print "WARNING: {0}\n".format(msg)

    elif (msgType == MessageType.ERROR):
        print "ERROR: {0}\n".format(msg)

    elif (msgType == MessageType.FATAL):
```

```
            print "FATAL: {0}\n".format(msg)

objectManagerExample = ObjectManagerExample()
```

Associated Types

- HdkApi.Data.Object.HomeObject

HdkApi.Data.Object.ObjectResource

ObjectResource

Provides information about a resource on disk which may be associated with an object and used by a component

Namespace: HdkApi.Data.Object
Assembly: Home.Data.Scripting.dll

Additional Information

Resource Types:

- "model"
- "texture"
- "particle"
- "file"
- "xml"
- "collision"
- "anim"
- "skeleton"
- "soundbank"
- "soundstream"
- "repertoire"
- "rig_settings"
- "control_rig"
- "emotes"
- "lua"
- "image"

Properties

Path

Type

Name

Example

```
#<copyright>
#--------------------------------------------------------------------------------------------------
#   SCE CONFIDENTIAL
#   Home Development Kit (HDK) 1.65.0
#   Copyright (C) 2011 Sony Computer Entertainment Inc.
#   All Rights Reserved.
#--------------------------------------------------------------------------------------------------
#</copyright>

# First import the Object classes
from HdkApi.Data.Object import ObjectManager, HomeObject, Uuid, ObjectResource

# Also import PathUtilities as we use it to get the project path
from HdkApi.Core.Utilities import PathUtilities

class HomeObjectResourceExample():
    def __init__(self):
        try:
            # Load an object by its uuid
            uuid = Uuid.Create("00000000-00000000-00000000-00000018")
            loadedObject = ObjectManager.LoadByUuid(uuid)

            # Add model resource to the object
            resourcePath = PathUtilities.ProjectPath +
'/Build/Furniture/Home_Rug_Sample/Home_Rug_Sample.mdl'
            resource = loadedObject.AddResource("test_resource", 'model', resourcePath)

            # Print the name of each resource the loadedObject contains
            for objectResource in loadedObject.GetResources():
                print objectResource.Name

            # Print the name of each model resource the loadedObject contains
            for objectResource in loadedObject.GetResources('model'):
                print objectResource.Name

            # Remove the resource from the object
            loadedObject.RemoveResource(resource)

        except Exception, e:
            print "{0}".format(e)

resourceExample = HomeObjectResourceExample()
```

Associated Types

- HdkApi.Data.Object.HomeObject

HdkApi.Data.Object.TerritoryTypes

TerritoryTypes

Lists all of the territory types

Namespace: HdkApi.Data.Object
Assembly: Home.Data.Scripting.dll

Remarks

Each type is used to look up a specific value in the entitlement_id, category_id and product_id dictionaries accessed through an Object's EntitlementId, CategoryId and ProductId properties

Fields

SCEE

SCEA

SCEJ

SCEAsia

Example

```
#<copyright>
#-------------------------------------------------------------------------------------------------
#   SCE CONFIDENTIAL
#   Home Development Kit (HDK) 1.55.0
#   Copyright (C) 2011 Sony Computer Entertainment Inc.
#   All Rights Reserved.
#-------------------------------------------------------------------------------------------------
#</copyright>

# First import the TerritoryTypes class
from HdkApi.Data.Object import TerritoryTypes

class TerritoryTypes_Example():
    def __init__(self):

  territories = [TerritoryTypes.SCEA,TerritoryTypes.SCEE,TerritoryTypes.SCEJ,TerritoryTypes.SCEAsia]
  for territory in territories:
   print (territory)

tt_example = TerritoryTypes_Example()
```

Associated Types

- HdkApi.Data.Object.HomeObject

HdkApi.Data.Object.Uuid

Uuid

Uuid object used to identify an object. To create a new Uuid use Uuid.Create()

Namespace: HdkApi.Data.Object
Assembly: Home.Data.Scripting.dll

Methods

HdkApi.Data.Object.Uuid Uuid.Create(uuid)

System.Boolean Uuid.IsValid(uuid)

Example

```
#<copyright>
#--------------------------------------------------------------------------------------------------
#   SCE CONFIDENTIAL
#   Home Development Kit (HDK) 1.55.0
#   Copyright (C) 2011 Sony Computer Entertainment Inc.
#   All Rights Reserved.
#--------------------------------------------------------------------------------------------------
#</copyright>

# First import the Uuid class
from HdkApi.Data.Object import Uuid

class UuidExample():
    def __init__(self):

  strHelicopterUuid = "45864836-89475616-84123186-13213216"
  isValid = Uuid.IsValid(strHelicopterUuid)
  if isValid:
      HelicopterUuid = Uuid.Create(strHelicopterUuid)
      print "Get to the chopper! %s" % strHelicopterUuid

  strDinosaurUuid = "DINOSAUR-SARENOTA-UUIDVELO-CIRAPTOR"
  isValid = Uuid.IsValid(strDinosaurUuid)
  if isValid == False:
   print "Dinosaurs are extinct!"

ux = UuidExample()
```

Associated Types

- HdkApi.Data.Object.ObjectManager
- HdkApi.Data.Object.HomeObject

HdkApi.Data.Object.Components

Summary

Type
Description
Assembly
ClothingComponent
Provides information about a clothing component
Home.Data.Scripting
HeaderComponent
Header information for an object, used for indexing and general information.
Home.Data.Scripting
FurnitureComponent

Provides information about a furniture component
Home.Data.Scripting
GameLaunchComponent
Provides information about a game_launch component
Home.Data.Scripting

HdkApi.Data.Object.Components.ClothingComponent

ClothingComponent

Provides information about a clothing component

Namespace: HdkApi.Data.Object.Components
Assembly: Home.Data.Scripting.dll

Properties

Name

ClothingType

Clothing

Rigs

Type

Example

```
#<copyright>
#--------------------------------------------------------------------------------------------------
#   SCE CONFIDENTIAL
#   Home Development Kit (HDK) 1.60.0
#   Copyright (C) 2011 Sony Computer Entertainment Inc.
#   All Rights Reserved.
#--------------------------------------------------------------------------------------------------
#</copyright>

# Import the Object classes
from HdkApi.Data.Object import ObjectManager, Uuid, ComponentTypes

# Import the ClothingComponent class
from HdkApi.Data.Object.Components import ClothingComponent

class ClothingExample():
    def __init__(self):

        # Load an object by its uuid
        uuid = Uuid.Create("00000000-00000000-00000000-00000018")

        try:
            loadedObject = ObjectManager.LoadByUuid(uuid)

            if loadedObject.Components.ContainsKey(ComponentTypes.Clothing):
                # Change the category of the clothing item
                loadedObject.Components[ComponentTypes.Clothing].Clothing = "Hat"

                # Change the rig which the clothing item is designed for
                loadedObject.Components[ComponentTypes.Clothing].Rigs =
"00000000-00000000-00000010-00000000"

        except Exception, e:
            print "Failed to load object {0}: {1}".format(uuid, e)
```

Associated Types

- HdkApi.Data.Object.HomeObject

HdkApi.Data.Object.Components.FurnitureComponent

FurnitureComponent

Provides information about a furniture component

Namespace: HdkApi.Data.Object.Components
Assembly: Home.Data.Scripting.dll

Properties

Furniture

Type

Example

```
#<copyright>
#--------------------------------------------------------------------------------------------------
#   SCE CONFIDENTIAL
#   Home Development Kit (HDK) 1.60.0
#   Copyright (C) 2011 Sony Computer Entertainment Inc.
#   All Rights Reserved.
#--------------------------------------------------------------------------------------------------
#</copyright>

# Import the Object classes
from HdkApi.Data.Object import ObjectManager, Uuid, ComponentTypes

# Import the FurnitureComponent class
from HdkApi.Data.Object.Components import FurnitureComponent

class FurnitureExample():
    def __init__(self):

        # Load an object by its uuid
        uuid = Uuid.Create("00000000-00000000-00000000-00000018")

        try:
            loadedObject = ObjectManager.LoadByUuid(uuid)

            if loadedObject.Components.ContainsKey(ComponentTypes.Furniture):
                # Change the category of the furniture item
                loadedObject.Components[ComponentTypes.Furniture].Furniture = "Desk"

        except Exception, e:
            print "Failed to load object {0}: {1}".format(uuid, e)
```

Associated Types

- HdkApi.Data.Object.HomeObject

HdkApi.Data.Object.Components.GameLaunchComponent

GameLaunchComponent

Provides information about a game_launch component

Namespace: HdkApi.Data.Object.Components
Assembly: Home.Data.Scripting.dll

Properties

CommunicationId

TitleId

Type

Example

```
#<copyright>
#--------------------------------------------------------------------------------------------------
#   SCE CONFIDENTIAL
#   Home Development Kit (HDK) 1.60.0
#   Copyright (C) 2011 Sony Computer Entertainment Inc.
#   All Rights Reserved.
#--------------------------------------------------------------------------------------------------
#</copyright>

# Import the Object classes
from HdkApi.Data.Object import ObjectManager, Uuid, ComponentTypes

# Import the GameLaunchComponent class
from HdkApi.Data.Object.Components import GameLaunchComponent

class GameLaunchExample():
    def __init__(self):

        # Load an object by its uuid
        uuid = Uuid.Create("00000000-00000000-00000000-00000018")

        try:
            loadedObject = ObjectManager.LoadByUuid(uuid)

            if loadedObject.Components.ContainsKey(ComponentTypes.GameLaunch):
                # Change the communication id
                loadedObject.Components[ComponentTypes.GameLaunch].CommunicationId = "COMM_ID"

                # Change the title id
                loadedObject.Components[ComponentTypes.GameLaunch].TitleId = "GAME_TITLE"

        except Exception, e:
            print "Failed to load object {0}: {1}".format(uuid, e)
```

Associated Types

- HdkApi.Data.Object.HomeObject

HdkApi.Data.Object.Components.HeaderComponent

HeaderComponent

Header information for an object, used for indexing and general information.

Namespace: HdkApi.Data.Object.Components
Assembly: Home.Data.Scripting.dll

Properties

Name

Description

Author

ObjectVersion

Publisher

Template

TemplateVersion

AgeRatings

HelpText

Lpid

Type

Example

```
#<copyright>
#-------------------------------------------------------------------------------------------------
#   SCE CONFIDENTIAL
#   Home Development Kit (HDK) 1.55.0
#   Copyright (C) 2011 Sony Computer Entertainment Inc.
#   All Rights Reserved.
#-------------------------------------------------------------------------------------------------
#</copyright>

# First import the Object classes
from HdkApi.Data.Object import ObjectManager, HomeObject, ObjectComponent, Uuid, ComponentTypes,
TerritoryTypes

# Import the MessageType class as we need this for the object validation and packaging output
from HdkApi.Data.Reporting import MessageType

# Also import PathUtilities as we use it to get the project path
from HdkApi.Core.Utilities import PathUtilities

class ObjectManagerExample():
    def __init__(self):
        try:
            # Load an object by its uuid
            uuid = Uuid.Create("00000000-00000000-00000000-00000018")
            loadedObject = ObjectManager.LoadByUuid(uuid)

            # Load an object by its object.xml path
            objectPath = PathUtilities.ProjectPath +
"/build/objects/00000000-00000000-00000000-00000018/object.xml"
            loadedObject = ObjectManager.LoadByFilename(objectPath)

        except Exception, e:
            print "Failed to load object {0}: {1}".format(uuid, e)

        # Get the Uuids of all objects which contain either a 'furniture' or an 'active' component
        components = ["furniture", "active"]
        foundObjects = ObjectManager.SearchByComponents(components)

        # Get the Uuids of all objects which contain a resource with the path 'resourcePath'
        resourcePath = "objects/00000000-00000000-00000000-00000018/main.lua"
        foundObjects = ObjectManager.SearchByResource(resourcePath)

        # Print the entitlement_id for the territory 'SCEA'
        print loadedObject.EntitlementId[TerritoryTypes.SCEA]

        # Set the entitlement_id for the territory 'SCEE'
        loadedObject.EntitlementId[TerritoryTypes.SCEE] = "LUA_REWARD"

        # Print the name of each component which loadedObject contains
        for component in loadedObject.Components.Keys:
            print component

        # Try to access the furniture component and set the furniture metadata item
        if loadedObject.Components.ContainsKey(ComponentTypes.Furniture):
            loadedObject.Components[ComponentTypes.Furniture].Furniture = "Chair"

        # Validate an object
        if loadedObject.Validate(self.Output):
            print "{0} - {1} passed validations\n".format(loadedObject.Name, loadedObject.Uuid)

        # Validate and package an object
```

```
    # The first passed in callback is triggered for validation output
    # The second callback is triggered for packager output
    if loadedObject.Package(self.Output, self.Output):
        print "{0} - {1} succesfully packaged\n".format(loadedObject.Name, loadedObject.Uuid)

# Example function for handling object validation and packager output
def Output(self, msg, msgType):
    if (msgType == MessageType.INFO):
        print "INFO: {0}\n".format(msg)

    elif (msgType == MessageType.WARNING):
        print "WARNING: {0}\n".format(msg)

    elif (msgType == MessageType.ERROR):
        print "ERROR: {0}\n".format(msg)

    elif (msgType == MessageType.FATAL):
```

```
            print "FATAL: {0}\n".format(msg)

objectManagerExample = ObjectManagerExample()
```

Associated Types

- HdkApi.Data.Object.HomeObject

HdkApi.Data.Packager

Summary

Type
Description
Assembly
Packager
The Packager class is used to package objects and scenes
Home.Data.Scripting

HdkApi.Data.Packager.Packager

Packager

The Packager class is used to package objects and scenes

Namespace: HdkApi.Data.Packager
Assembly: Home.Data.Scripting.dll

Methods

System.Boolean Packager.PackageObject(uuid, validatorOutput, packagerOutput)

System.Boolean Packager.PackageScene(scenePath, sceneType, validatorOutput, packagerOutput)

HdkApi.Data.Reporting

Summary

Type
Description
Assembly
MessageType
The types of messages which can be returned through the MessageOutput delegate

Home.Data.Scripting

HdkApi.Data.Reporting.MessageType

MessageType

The types of messages which can be returned through the MessageOutput delegate

Namespace: HdkApi.Data.Reporting
Assembly: Home.Data.Scripting.dll

Fields

INFO

WARNING

ERROR

FATAL

HdkApi.Data.Scene

Summary

Type
Description
Assembly
GameObjectFolder
Provides information about a GameObjectFolder within a Scene
Home.Data.Scripting
GameObject
Provides information about a Game Object within a Scene
Home.Data.Scripting
GameObjectTypes
Lists all of the game object types
Home.Data.Scripting
HomeScene
Provides information about a scene

Home.Data.Scripting
SceneConverter
Provides functions for converting between scene (.scene) and scene project (.scproj) files.
Home.Data.Scripting
SceneTypes
Lists all of the scene types
Home.Data.Scripting

HdkApi.Data.Scene.GameObject

GameObject

Provides information about a Game Object within a Scene

Namespace: HdkApi.Data.Scene
Assembly: Home.Data.Scripting.dll

Properties

Name

Type

Translation

Rotation

Scale

Example

```
#<copyright>
#--------------------------------------------------------------------------------------------------
#   SCE CONFIDENTIAL
#   Home Development Kit (HDK) 1.60.0
#   Copyright (C) 2011 Sony Computer Entertainment Inc.
#   All Rights Reserved.
#--------------------------------------------------------------------------------------------------
#</copyright>

# First import the SceneProject classes
from HdkApi.Data.Scene import HomeScene, GameObject

# Import the SceneEditor class so we can get the current scene project open in the Scene Editor
from HdkApi.SceneEditor import SceneEditor

# Import the Maths Converters class so we can use the degrees to radians function
from HdkApi.Core.Maths import Converters

class SceneProjectExample():
    def __init__(self):
        # Get the scene project which is currently open in the Scene Editor
        openSceneProj = SceneEditor.CurrentScene

        # Print the scene project's full file path
        print openSceneProj.FilePath

        # Print the name and type of each game object within the scene project
        for gameObject in openSceneProj.AllGameObjects:
            print "{0} : {1}".format(gameObject.Name, gameObject.Type)

        # Find a game object by its name
        spawnPoint = openSceneProj.FindGameObjectByName("Spawn Point")

        if spawnPoint != None:
            # Change the translation, rotation and scale of the spawn point
            spawnPoint.Translation = Vector3(20, 0, 0)
            spawnPoint.Rotation = Vector3(Converters.DegToRad(90), 0, 0)
            spawnPoint.Scale = Vector3(2, 1, 1)

        # Find all game objects of the type 'entityType'
        entities = openSceneProj.FindGameObjectsByType(GameObjectTypes.Entity)

        # Print the name of all entities in the scene project
        for entity in entities:
            print entity.Name
```

Associated Types

- HdkApi.Data.Scene.HomeScene
- HdkApi.Data.Scene.GameObjectTypes

HdkApi.Data.Scene.GameObjectFolder

GameObjectFolder

Provides information about a GameObjectFolder within a Scene

Namespace: HdkApi.Data.Scene
Assembly: Home.Data.Scripting.dll

Properties

Name

GameObjects

Folders

Associated Types

- HdkApi.Data.Scene.HomeScene

HdkApi.Data.Scene.GameObjectTypes

GameObjectTypes

Lists all of the game object types

Namespace: HdkApi.Data.Scene
Assembly: Home.Data.Scripting.dll

Fields

CollisionFile

Entity

EnvironmentMap

EyeEnvironmentMap

ProbeFile

AtmosphereFile

Model

SkinEnvironmentMap

DefaultFurniturePoint

Seat

SeatArea

PictureHook

SpawnPoint

Script

EmbeddedObject

Screen

ClubBulletinScreen

ScreenLinksFile

ArcadeGame

MiniGame

AmbientSound

SphereEmitter

CubeEmitter

ConeEmitter

CylinderEmitter

ConePointSound

CubePointSound

CylinderPointSound

SpherePointSound

SoundBank

SoundPortal

SoundZone

ChangeEventBoxTrigger

ChangeEventTrigger

ParticleEffect

GameObjectGroup

Associated Types

- HdkApi.Data.Scene.HomeScene
- HdkApi.Data.Scene.GameObject

HdkApi.Data.Scene.HomeScene

HomeScene

Provides information about a scene

Namespace: HdkApi.Data.Scene
Assembly: Home.Data.Scripting.dll

Methods

HdkApi.Data.Scene.GameObjectFolder FindGameObjectFolderByName(name)

HdkApi.Data.Scene.GameObject FindGameObjectByName(name)

System.Collections.Generic.IEnumerable[HdkApi.Data.Scene.GameObject] FindGameObjectsByType(type)

System.Boolean Validate(validatorOutput, sceneType)

System.Boolean Package(validatorOutput, packagerOutput, sceneType)

Save( )

Properties

FilePath

AllGameObjects

GameObjectFolder

Dependencies

Example

```
#<copyright>
#--------------------------------------------------------------------------------------------------
#   SCE CONFIDENTIAL
#   Home Development Kit (HDK) 1.60.0
#   Copyright (C) 2011 Sony Computer Entertainment Inc.
#   All Rights Reserved.
#--------------------------------------------------------------------------------------------------
#</copyright>

# First import the SceneProject classes
from HdkApi.Data.Scene import HomeScene, GameObject

# Import the SceneEditor class so we can get the current scene project open in the Scene Editor
from HdkApi.SceneEditor import SceneEditor

# Import the Maths Converters class so we can use the degrees to radians function
from HdkApi.Core.Maths import Converters

class SceneProjectExample():
    def __init__(self):
        # Get the scene project which is currently open in the Scene Editor
        openSceneProj = SceneEditor.CurrentScene

        # Print the scene project's full file path
        print openSceneProj.FilePath

        # Print the name and type of each game object within the scene project
        for gameObject in openSceneProj.AllGameObjects:
            print "{0} : {1}".format(gameObject.Name, gameObject.Type)

        # Find a game object by its name
        spawnPoint = openSceneProj.FindGameObjectByName("Spawn Point")

        if spawnPoint != None:
            # Change the translation, rotation and scale of the spawn point
            spawnPoint.Translation = Vector3(20, 0, 0)
            spawnPoint.Rotation = Vector3(Converters.DegToRad(90), 0, 0)
            spawnPoint.Scale = Vector3(2, 1, 1)

        # Find all game objects of the type 'entityType'
        entities = openSceneProj.FindGameObjectsByType(GameObjectTypes.Entity)

        # Print the name of all entities in the scene project
        for entity in entities:
            print entity.Name
```

Associated Types

- HdkApi.Data.Scene.GameObject
- HdkApi.Data.Scene.GameObjectTypes

HdkApi.Data.Scene.SceneConverter

SceneConverter

Provides functions for converting between scene (.scene) and scene project (.scproj) files.

Namespace: HdkApi.Data.Scene
Assembly: Home.Data.Scripting.dll

Methods

System.String SceneConverter.SceneToProject(sceneToConvert, forceOverwrite)

System.String SceneConverter.ProjectToScene(projectToConvert, forceOverwrite)

System.String SceneConverter.GetSceneVersion(sceneToCheck)

Example

```
#<copyright>
#-------------------------------------------------------------------------------------------------
#   SCE CONFIDENTIAL
#   Home Development Kit (HDK) 1.60.0
#   Copyright (C) 2011 Sony Computer Entertainment Inc.
#   All Rights Reserved.
#-------------------------------------------------------------------------------------------------
#</copyright>

# First import the SceneConverter class
from HdkApi.Data.Scene import SceneConverter

class SceneConverterExample():
    def __init__(self):

  # Convert a scene to a project
  scprojFile = SceneConverter.SceneToProject("D:\foo\bar\demo.scene", True)

  # Convert a project to a scene
  sceneFile = SceneConverter.ProjectToScene("D:\foo\bar\demo.scproj", True)

  # Get a scenes version number
  version = SceneConverter.GetSceneVersion("D:\foo\bar\demo.scene")
```

HdkApi.Data.Scene.SceneTypes

SceneTypes

Lists all of the scene types

Namespace: HdkApi.Data.Scene
Assembly: Home.Data.Scripting.dll

Fields

Public

Personal

ClubHouse

Game16

Game32

Game60

PublicNoVideo

Video

PersonalVideo

PersonalNoVideo

Associated Types

- HdkApi.Data.Scene.HomeScene

HdkApi.Data.Scene.GameObjects

Summary

Type
Description
Assembly
GameObjectGroup
Provides information about a Game Object Group
Home.Data.Scripting

HdkApi.Data.Scene.GameObjects.GameObjectGroup

GameObjectGroup

Provides information about a Game Object Group

Namespace: HdkApi.Data.Scene.GameObjects
Assembly: Home.Data.Scripting.dll

Properties

GameObjects

Name

Type

Translation

Rotation

Scale

Example

```
#<copyright>
#--------------------------------------------------------------------------------------------------
#   SCE CONFIDENTIAL
#   Home Development Kit (HDK) 1.60.0
#   Copyright (C) 2011 Sony Computer Entertainment Inc.
#   All Rights Reserved.
#--------------------------------------------------------------------------------------------------
#</copyright>

# First import the SceneProject classes
from HdkApi.Data.Scene import HomeScene, GameObject

# Import the SceneEditor class so we can get the current scene project open in the Scene Editor
from HdkApi.SceneEditor import SceneEditor

# Import the Maths Converters class so we can use the degrees to radians function
from HdkApi.Core.Maths import Converters

class SceneProjectExample():
    def __init__(self):
        # Get the scene project which is currently open in the Scene Editor
        openSceneProj = SceneEditor.CurrentScene

        # Print the scene project's full file path
        print openSceneProj.FilePath

        # Print the name and type of each game object within the scene project
        for gameObject in openSceneProj.AllGameObjects:
            print "{0} : {1}".format(gameObject.Name, gameObject.Type)

        # Find a game object by its name
        spawnPoint = openSceneProj.FindGameObjectByName("Spawn Point")

        if spawnPoint != None:
            # Change the translation, rotation and scale of the spawn point
            spawnPoint.Translation = Vector3(20, 0, 0)
            spawnPoint.Rotation = Vector3(Converters.DegToRad(90), 0, 0)
            spawnPoint.Scale = Vector3(2, 1, 1)

        # Find all game objects of the type 'entityType'
        entities = openSceneProj.FindGameObjectsByType(GameObjectTypes.Entity)

        # Print the name of all entities in the scene project
        for entity in entities:
            print entity.Name
```

Associated Types

- HdkApi.Data.Scene.HomeScene

HdkApi.Data.Validator

Summary

Type
Description
Assembly
Validator

The Validator class is used to validate objects and scenes
Home.Data.Scripting

HdkApi.Data.Validator.Validator

Validator

The Validator class is used to validate objects and scenes

Namespace: HdkApi.Data.Validator
Assembly: Home.Data.Scripting.dll

Methods

System.Boolean Validator.ValidateObject(uuid, validatorOutput)

System.Boolean Validator.ValidateScene(scenePath, sceneType, validatorOutput)

HdkApi.ObjectEditor

Summary

Type
Description
Assembly
ObjectEditor
Provides access to core functionality of the Object Editor, such as events for when objects are opened.
Home.Editors.ObjectEditor.Scripting

HdkApi.ObjectEditor.ObjectEditor

ObjectEditor

Provides access to core functionality of the Object Editor, such as events for when objects are opened.

Namespace: HdkApi.ObjectEditor
Assembly: Home.Editors.ObjectEditor.Scripting.dll

Methods

ObjectEditor.OpenObjectByUuid(uuid)

ObjectEditor.CloseEditor( )

System.Collections.Generic.IEnumerable[HdkApi.Data.Object.HomeObject] ObjectEditor.GetOpenObjects( )

Events

OnSaving

OnSaved

OnOpened

OnPackaging

OnPackaged

Delegates

ObjectAction

Example

```
#<copyright>
#----------------------------------------------------------------------------------------------------
#   SCE CONFIDENTIAL
#   Home Development Kit (HDK) 1.65.0
#   Copyright (C) 2011 Sony Computer Entertainment Inc.
#   All Rights Reserved.
#----------------------------------------------------------------------------------------------------
#</copyright>

# First import the ObjectEditor class
from HdkApi.ObjectEditor import ObjectEditor

# Import the HomeObject class
from HdkApi.Data.Object import HomeObject

class ObjectEditorExample():
    def __init__(self):
        # Hook up to the OnOpened event called when an object is opened in the Object Editor
        ObjectEditor.OnOpened += self.onOpened

        # Hook up to the OnSaving event called when an object is about to be saved
        ObjectEditor.OnSaving += self.onSaving

        # Hook up to the OnSaved event called when an object has been successfully saved
        ObjectEditor.OnSaved += self.onSaved

        # Hook up to the OnPackaging event called when an object is about to be packaged
        ObjectEditor.OnPackaging += self.onPackaging

        # Hook up to the OnPackaged event called when an object has been successfully packaged
        ObjectEditor.OnPackaged += self.onPackaged

    # Example of a callback function triggered when an object is opened in the Object Editor
    def onOpened(self, object):
        print "Opened: {0}".format(object.FilePath)

    # Example of a callback function triggered when an object is about to be saved in the Object
Editor
    def onSaving(self, object):
        print "Saving: {0}".format(object.FilePath)

    # Example of a callback function triggered when an object has been successfully saved in the
Object Editor
    def onSaved(self, object):
        print "Saved: {0}".format(object.FilePath)

    # Example of a callback function triggered when an object is about to be packaged in the Object
Editor
    def onPackaging(self, object):
        print "Packaging: {0}".format(object.FilePath)

    # Example of a callback function triggered when an object has been successfully packaged in the
Object Editor
    def onPackaged(self, object):
        print "Packaged: {0}".format(object.FilePath)
```

Associated Types

- HdkApi.Data.Object.HomeObject

HdkApi.ObjectEditor.UserInterface

Summary

Type	Description	Assembly
Menu	Menu represents a menu in the Object Editor UI, new items can be added to the menu by calling the AddMenuItem function, a new Menu can be created through MenuManager.CreateMenu	Home.Editors.ObjectEditor.Scripting
MenuItem	MenuItem represents an item within a Menu in the Object Editor UI, it provides access to properties about the menu item such as what icon it uses and whether it has a button associated with it, for a menu item to be visible it must be added to a Menu through AddMenuItem, to create a new MenuItem call MenuManager.CreateMenuItem	Home.Editors.ObjectEditor.Scripting
MenuManager	MenuManager is used to add new menus and menu items to the Object Editor UI, to add a new menu call CreateMenu, to add a new menu item call CreateMenuItem and then pass the returned MenuItem to a menu's AddMenuItem function	Home.Editors.ObjectEditor.Scripting

HdkApi.ObjectEditor.UserInterface.Menu

Menu

Menu represents a menu in the Object Editor UI, new items can be added to the menu by calling the AddMenuItem function, a new Menu can be created through MenuManager.CreateMenu

Namespace: HdkApi.ObjectEditor.UserInterface
Assembly: Home.Editors.ObjectEditor.Scripting.dll

Methods

AddMenuItem(menuItem)

Example

```
#<copyright>
#--------------------------------------------------------------------------------------------------
#   SCE CONFIDENTIAL
#   Home Development Kit (HDK) 1.55.0
#   Copyright (C) 2011 Sony Computer Entertainment Inc.
#   All Rights Reserved.
#--------------------------------------------------------------------------------------------------
#</copyright>

# First import the UserInterface classes
from HdkApi.SceneEditor.UserInterface import MenuManager, Menu, MenuItem

# Also add the System.Windows.Forms dll and import Keys so we can set a keyboard shortcut
clr.AddReference("System.Windows.Forms")
from System.Windows.Forms import Keys

class UiExample():
    def __init__(self):
        # Create a new menu
        menu = MenuManager.CreateMenu("Example Menu")

        # Create a new MenuItem
        itemDisplayText = "Example Item"
        menuItem = MenuManager.CreateMenuItem(itemDisplayText)
        menuItem.Description = "Example menu item used in the UI Example"
        menuItem.Toggleable = True
        menuItem.Shortcut = Keys.Control | Keys.Shift | Keys.M
        menuItem.HasToolbarButton = True

        # Hook up the execute event to set what happens when the menu item is pressed
        menuItem.Execute += self.exampleExecute

        # Hook up the can execute event to set whether the menu item is enabled
        menuItem.CanExecute += self.exampleCanExecute

        # Add the MenuItem to the example menu
        menu.AddMenuItem(menuItem)

    def exampleCanExecute(self):
        return True

    def exampleExecute(self, sender, args):
        print("Example executed")

example = UiExample()
```

Associated Types

- HdkApi.ObjectEditor.UserInterface.MenuManager
- HdkApi.ObjectEditor.UserInterface.MenuItem

HdkApi.ObjectEditor.UserInterface.MenuItem

MenuItem

MenuItem represents an item within a Menu in the Object Editor UI, it provides access to properties about the menu item such as what icon it uses and whether it has a button associated with it, for a menu item to be visible it must be added to a Menu through AddMenuItem, to create a new MenuItem call MenuManager.CreateMenuItem

Namespace: HdkApi.ObjectEditor.UserInterface
Assembly: Home.Editors.ObjectEditor.Scripting.dll

Properties

DisplayText

Description

IconPath

Shortcut

Toggleable

HasToolbarButton

Events

Execute

CanExecute

Delegates

CanExecuteHandler

Example

```
#<copyright>
#--------------------------------------------------------------------------------------------------
#   SCE CONFIDENTIAL
#   Home Development Kit (HDK) 1.55.0
#   Copyright (C) 2011 Sony Computer Entertainment Inc.
#   All Rights Reserved.
#--------------------------------------------------------------------------------------------------
#</copyright>

# First import the UserInterface classes
from HdkApi.SceneEditor.UserInterface import MenuManager, Menu, MenuItem

# Also add the System.Windows.Forms dll and import Keys so we can set a keyboard shortcut
clr.AddReference("System.Windows.Forms")
from System.Windows.Forms import Keys

class UiExample():
    def __init__(self):
        # Create a new menu
        menu = MenuManager.CreateMenu("Example Menu")

        # Create a new MenuItem
        itemDisplayText = "Example Item"
        menuItem = MenuManager.CreateMenuItem(itemDisplayText)
        menuItem.Description = "Example menu item used in the UI Example"
        menuItem.Toggleable = True
        menuItem.Shortcut = Keys.Control | Keys.Shift | Keys.M
        menuItem.HasToolbarButton = True

        # Hook up the execute event to set what happens when the menu item is pressed
        menuItem.Execute += self.exampleExecute

        # Hook up the can execute event to set whether the menu item is enabled
        menuItem.CanExecute += self.exampleCanExecute

        # Add the MenuItem to the example menu
        menu.AddMenuItem(menuItem)

    def exampleCanExecute(self):
        return True

    def exampleExecute(self, sender, args):
        print("Example executed")

example = UiExample()
```

Associated Types

- HdkApi.ObjectEditor.UserInterface.MenuManager
- HdkApi.ObjectEditor.UserInterface.Menu

HdkApi.ObjectEditor.UserInterface.MenuManager

MenuManager

MenuManager is used to add new menus and menu items to the Object Editor UI, to add a new menu call CreateMenu, to add a new menu item call CreateMenuItem and then pass the returned MenuItem to a menu's AddMenuItem function

Namespace: HdkApi.ObjectEditor.UserInterface
Assembly: Home.Editors.ObjectEditor.Scripting.dll

Methods

HdkApi.ObjectEditor.UserInterface.Menu MenuManager.CreateMenu(name)

HdkApi.ObjectEditor.UserInterface.Menu MenuManager.GetMenu(name)

HdkApi.ObjectEditor.UserInterface.MenuItem MenuManager.CreateMenuItem(displayText)

Example

```
#<copyright>
#-------------------------------------------------------------------------------------------------
#   SCE CONFIDENTIAL
#   Home Development Kit (HDK) 1.55.0
#   Copyright (C) 2011 Sony Computer Entertainment Inc.
#   All Rights Reserved.
#-------------------------------------------------------------------------------------------------
#</copyright>

# First import the UserInterface classes
from HdkApi.SceneEditor.UserInterface import MenuManager, Menu, MenuItem

# Also add the System.Windows.Forms dll and import Keys so we can set a keyboard shortcut
clr.AddReference("System.Windows.Forms")
from System.Windows.Forms import Keys

class UiExample():
    def __init__(self):
        # Create a new menu
        menu = MenuManager.CreateMenu("Example Menu")

        # Create a new MenuItem
        itemDisplayText = "Example Item"
        menuItem = MenuManager.CreateMenuItem(itemDisplayText)
        menuItem.Description = "Example menu item used in the UI Example"
        menuItem.Toggleable = True
        menuItem.Shortcut = Keys.Control | Keys.Shift | Keys.M
        menuItem.HasToolbarButton = True

        # Hook up the execute event to set what happens when the menu item is pressed
        menuItem.Execute += self.exampleExecute

        # Hook up the can execute event to set whether the menu item is enabled
        menuItem.CanExecute += self.exampleCanExecute

        # Add the MenuItem to the example menu
        menu.AddMenuItem(menuItem)

    def exampleCanExecute(self):
        return True

    def exampleExecute(self, sender, args):
        print("Example executed")

example = UiExample()
```

Associated Types

- HdkApi.ObjectEditor.UserInterface.Menu
- HdkApi.ObjectEditor.UserInterface.MenuItem

HdkApi.RepertoireEditor

Summary

Type

Description

Assembly

Action

An action defines one off animations within repertoire

Home.Editors.RepertoireEditor.Scripting

Behaviour

A behaviour defines repetitive behaviour in repertoire

Home.Editors.RepertoireEditor.Scripting

EaseCurveType

Determines how a curve blends over time

Home.Editors.RepertoireEditor.Scripting

EaseCurve

Describes how a blend in or out of a state happens over time

Home.Editors.RepertoireEditor.Scripting

Locomotion

Locomotion defines movement within repertoires

Home.Editors.RepertoireEditor.Scripting

LocomotionState

Locomotion state module used to define locomotions in repertoire

Home.Editors.RepertoireEditor.Scripting

RepertoireRegister

Home.Editors.RepertoireEditor.Scripting

BooleanRegister

A boolean value register

Home.Editors.RepertoireEditor.Scripting

IntegerRegister
An integer value register
Home.Editors.RepertoireEditor.Scripting
NumberRegister
A number value register
Home.Editors.RepertoireEditor.Scripting
StringRegister
A string value register
Home.Editors.RepertoireEditor.Scripting
Repertoire
Repertoire class used to create and access repertoire modules
Home.Editors.RepertoireEditor.Scripting
RepertoireEditor
The main class for repertoire editor, use it to access currently open repertoires
Home.Editors.RepertoireEditor.Scripting
RepertoireEvent
Home.Editors.RepertoireEditor.Scripting
RepertoireTargetRig
The rig, emotes and repertoire resources referenced in the repertoire component.
Home.Editors.RepertoireEditor.Scripting
State
A state can be part of a behaviour or an action
Home.Editors.RepertoireEditor.Scripting
Transition

Create a transition between 2 animations by specifying a transition animation

Home.Editors.RepertoireEditor.Scripting

HdkApi.RepertoireEditor.Action

Action

An action defines one off animations within repertoire

Namespace: HdkApi.RepertoireEditor
Assembly: Home.Editors.RepertoireEditor.Scripting.dll

Methods

HdkApi.RepertoireEditor.Conditions.Condition GetCondition( )

HdkApi.RepertoireEditor.Conditions.Condition CreateCondition(name)

SetDuration(duration)

System.Object GetDuration( )

HdkApi.RepertoireEditor.State GetInitialState( )

SetInitialState(state)

System.Collections.Generic.List[HdkApi.RepertoireEditor.Outputs.Output] GetOutputs( )

HdkApi.RepertoireEditor.Outputs.Output CreateOutput(name)

System.Collections.Generic.List[HdkApi.RepertoireEditor.State] GetStates( )

HdkApi.RepertoireEditor.State CreateState(name)

Delete( )

Properties

Layer

Blocking

Priority

States

Attributes

Position

Name

Example

```
#<copyright>
#-----------------------------------------------------------------------------------------------------
#   SCE CONFIDENTIAL
#   Home Development Kit (HDK) 1.65.0
#   Copyright (C) 2011 Sony Computer Entertainment Inc.
#   All Rights Reserved.
#-----------------------------------------------------------------------------------------------------
#</copyright>

def CreateAction():
    rep = RepertoireEditor.CurrentRepertoire
    if(rep):
        action = rep.CreateAction("Action")
        if(action):
            #set some properties
            action.Name = "Action"
            action.Layer = "my_layer"
            action.Blocking = True
            action.Priority = 1
            action.Position = Point(500,100)

            #set the duration of the action
            action.SetDuration(5)

            #set some attributes
            action.Attributes["something"] = 10

            #create a condition for this action
            condition = action.CreateCondition("condition")
            if(condition):
                print("created condition, you may wish to use this to add some logic")
                condition.Position = Point(350, 100)

            #create a state for this action
            state = action.CreateState("State")
            if(state):
                print("created state, you may wish to add animations to this state")
                state.Position = Point(500, 100)

            #set this state as the initial state for the action
            action.SetInitialState(state)

            #create an output for this action
            output = action.CreateOutput("Output")
            if(output):
                print("created output, you may wish to use this to define behaviour on entry and exit
to this action")
                output.Position = Point(500, 200)

            #get the conditon back from the action
            theCondition = action.GetCondition()
            if(theCondition):
                print("got the condition of this action")

            #get the list of states from this action
            theStates = action.GetStates()
            if(theStates):
                print("got the states of this action")

            #get the list of outputs from this action
            theOutputs = action.GetOutputs()
            if(theOutputs):
                print("got the list of outputs of this action")

CreateAction()
```

HdkApi.RepertoireEditor.Behaviour

Behaviour

A behaviour defines repetitive behaviour in repertoire

Namespace: HdkApi.RepertoireEditor
Assembly: Home.Editors.RepertoireEditor.Scripting.dll

Methods

HdkApi.RepertoireEditor.Conditions.Condition GetCondition( )

HdkApi.RepertoireEditor.Conditions.Condition CreateCondition(name)

System.Collections.Generic.List[HdkApi.RepertoireEditor.Outputs.Output] GetOutputs( )

HdkApi.RepertoireEditor.Outputs.Output CreateOutput(name)

HdkApi.RepertoireEditor.State GetInitialState( )

SetInitialState(state)

System.Collections.Generic.List[HdkApi.RepertoireEditor.State] GetStates( )

HdkApi.RepertoireEditor.State CreateState(name)

System.Collections.Generic.List[HdkApi.RepertoireEditor.LocomotionState] GetLocomotionStates( )

HdkApi.RepertoireEditor.LocomotionState CreateLocomotionState(name)

Delete( )

Properties

States

Attributes

Position

Name

Example

```
#<copyright>
#-----------------------------------------------------------------------------------------------
#   SCE CONFIDENTIAL
#   Home Development Kit (HDK) 1.65.0
#   Copyright (C) 2011 Sony Computer Entertainment Inc.
#   All Rights Reserved.
#-----------------------------------------------------------------------------------------------
#</copyright>

def CreateBehaviour():
    rep = RepertoireEditor.CurrentRepertoire
    if(rep):
        behaviour = rep.CreateBehaviour("Behaviour")
        if(behaviour):
            #set some properties
            behaviour.Name = "Behaviour"
            behaviour.Position = Point(500,100)

            #set some attributes
            behaviour.Attributes["something"] = 10

            #create a condition for this behaviour
            condition = behaviour.CreateCondition("condition")
```

```
            if(condition):
                print("created condition, you may wish to use this to add some logic")
                condition.Position = Point(350,100)

            #create a state for this behaviour
            state = behaviour.CreateState("State")
            if(state):
                print("created state, you may wish to add animations to this state")
                state.Position = Point(500,100)

            #set the initial state
            behaviour.SetInitialState(state)

            #create a locomotion state for this behaviour
            locomotionState = behaviour.CreateLocomotionState("Locomotion_State")
            if(locomotionState):
                print("created locomotion state, you may wish to add animations to this state")
                locomotionState.Position = Point(500,200)

            #create an output for this behaviour
            output = behaviour.CreateOutput("Output")
            if(output):
                print("created output, you may wish to use this to define behaviour on entry and exit
to this behaviour")
                output.Position = Point(500,300)

            #get the conditon back from the behaviour
            theCondition = behaviour.GetCondition()
            if(theCondition):
                print("got the condition of this behaviour")

            #get the list of states from this behaviour
            theStates = behaviour.GetStates()
            if(theStates):
                print("got the states of this behaviour")

            #get the list of states from this behaviour
            theLocomotionStates = behaviour.GetLocomotionStates()
            if(theLocomotionStates):
                print("got the locomotion states of this behaviour")

            #get the list of outputs from this behaviour
            theOutputs = behaviour.GetOutputs()
            if(theOutputs):
```

```
                print("got the list of outputs of this behaviour")

CreateBehaviour()
```

HdkApi.RepertoireEditor.BooleanRegister

BooleanRegister

A boolean value register

Namespace: HdkApi.RepertoireEditor
Assembly: Home.Editors.RepertoireEditor.Scripting.dll

Methods

Delete( )

Properties

Name

RepertoireWrite

RepertoireRead

LuaRead

LuaWrite

IsCore

HdkApi.RepertoireEditor.EaseCurve

EaseCurve

Describes how a blend in or out of a state happens over time

Namespace: HdkApi.RepertoireEditor
Assembly: Home.Editors.RepertoireEditor.Scripting.dll

Constructors

EaseCurve(curveType, duration)

Methods

SetDuration(duration)

System.Object GetDuration( )

HdkApi.RepertoireEditor.EaseCurveType GetCurveType( )

SetCurveType(curveType)

Example

```
#<copyright>
#--------------------------------------------------------------------------------------------------
#   SCE CONFIDENTIAL
#   Home Development Kit (HDK) 1.65.0
#   Copyright (C) 2011 Sony Computer Entertainment Inc.
#   All Rights Reserved.
#--------------------------------------------------------------------------------------------------
#</copyright>

def CreateEaseCurve():
    #create an ease curve
    easeCurve = EaseCurve(EaseCurveType.linear, 5.0)

    if(easeCurve):
        print("created ease curve")

        #get the type
        type = easeCurve.GetCurveType()
        print("type = " + str(type))
        #get the duration
        duration = easeCurve.GetDuration()
        print("duration = " + str(duration))

        #set the duration
        easeCurve.SetDuration(2.5)

        #set the type
        easeCurve.SetCurveType(EaseCurveType.s_curve)

CreateEaseCurve()
```

HdkApi.RepertoireEditor.EaseCurveType

EaseCurveType

Determines how a curve blends over time

Namespace: HdkApi.RepertoireEditor
Assembly: Home.Editors.RepertoireEditor.Scripting.dll

Fields

linear

s_curve

Example

```
#<copyright>
#--------------------------------------------------------------------------------------------------
#   SCE CONFIDENTIAL
#   Home Development Kit (HDK) 1.65.0
#   Copyright (C) 2011 Sony Computer Entertainment Inc.
#   All Rights Reserved.
#--------------------------------------------------------------------------------------------------
#</copyright>

def CreateEaseCurve():
    #create an ease curve
    easeCurve = EaseCurve(EaseCurveType.linear, 5.0)

    if(easeCurve):
        print("created ease curve")

        #get the type
        type = easeCurve.GetCurveType()
        print("type = " + str(type))
        #get the duration
        duration = easeCurve.GetDuration()
        print("duration = " + str(duration))

        #set the duration
        easeCurve.SetDuration(2.5)

        #set the type
        easeCurve.SetCurveType(EaseCurveType.s_curve)

CreateEaseCurve()
```

HdkApi.RepertoireEditor.IntegerRegister

IntegerRegister

An integer value register

Namespace: HdkApi.RepertoireEditor
Assembly: Home.Editors.RepertoireEditor.Scripting.dll

Methods

Delete( )

Properties

Min

Max

Name

RepertoireWrite

RepertoireRead

LuaRead

LuaWrite

IsCore

HdkApi.RepertoireEditor.Locomotion

Locomotion

Locomotion defines movement within repertoires

Namespace: HdkApi.RepertoireEditor
Assembly: Home.Editors.RepertoireEditor.Scripting.dll

Methods

HdkApi.RepertoireEditor.Animation.SkeletalAnimation GetSkeletalAnimation( )

HdkApi.RepertoireEditor.Animation.SkeletalAnimation CreateSkeletalAnimation(name)

HdkApi.RepertoireEditor.Animation.BlendTreeAnimation GetBlendTreeAnimation( )

HdkApi.RepertoireEditor.Animation.BlendTreeAnimation CreateBlendTreeAnimation(name)

Delete( )

Properties

Speed

Threshold

Position

Name

HdkApi.RepertoireEditor.LocomotionState

LocomotionState

Locomotion state module used to define locomotions in repertoire

Namespace: HdkApi.RepertoireEditor
Assembly: Home.Editors.RepertoireEditor.Scripting.dll

Methods

HdkApi.RepertoireEditor.Conditions.Condition GetCondition( )

HdkApi.RepertoireEditor.Conditions.Condition CreateCondition(name)

System.Collections.Generic.List[HdkApi.RepertoireEditor.Locomotion] GetLocomotions( )

HdkApi.RepertoireEditor.Locomotion CreateLocomotion(name)

System.Collections.Generic.List[HdkApi.RepertoireEditor.Animation.SkeletalAnimation] GetSkeletalAnimations( )

System.Collections.Generic.List[HdkApi.RepertoireEditor.Animation.BlendTreeAnimation] GetBlendTreeAnimations( )

SetDuration(duration)

System.Object GetDuration( )

HdkApi.RepertoireEditor.EaseCurve GetEaseIn( )

SetEaseIn(easeCurve)

HdkApi.RepertoireEditor.EaseCurve GetEaseOut( )

SetEaseOut(easeCurve)

System.Collections.Generic.List[HdkApi.RepertoireEditor.Outputs.Output] GetOutputs( )

HdkApi.RepertoireEditor.Outputs.Output CreateOutput(name)

Delete( )

Properties

SkeletalAnimations

BlendTreeAnimations

Position

Name

HdkApi.RepertoireEditor.NumberRegister

NumberRegister

A number value register

Namespace: HdkApi.RepertoireEditor
Assembly: Home.Editors.RepertoireEditor.Scripting.dll

Methods

Delete( )

Properties

Min

Max

Name

RepertoireWrite

RepertoireRead

LuaRead

LuaWrite

IsCore

HdkApi.RepertoireEditor.Repertoire

Repertoire

Repertoire class used to create and access repertoire modules

Namespace: HdkApi.RepertoireEditor
Assembly: Home.Editors.RepertoireEditor.Scripting.dll

Methods

System.Boolean Save( )

HdkApi.Core.Collections.HdkDictionary[System.String,System.Collections.Generic.List[System.String]] Validate( )

System.Boolean Export( )

System.Collections.Generic.List[HdkApi.RepertoireEditor.Action] GetActions( )

HdkApi.RepertoireEditor.Action CreateAction(name)

HdkApi.RepertoireEditor.Behaviour CreateBehaviour(name)

System.Collections.Generic.List[HdkApi.RepertoireEditor.Behaviour] GetBehaviours( )

System.Collections.Generic.List[HdkApi.RepertoireEditor.Transition] GetTransitions( )

HdkApi.RepertoireEditor.Emotes.RegionalEmotes CreateRegionalEmotes(name)

System.Collections.Generic.List[HdkApi.RepertoireEditor.Emotes.RegionalEmotes] GetRegionalEmotes( )

HdkApi.RepertoireEditor.Emotes.GlobalEmotes CreateGlobalEmotes(name)

HdkApi.RepertoireEditor.Emotes.GlobalEmotes GetGlobalEmotes( )

HdkApi.RepertoireEditor.Animation.BlendTree CreateBlendTree(name)

System.Collections.Generic.List[HdkApi.RepertoireEditor.Animation.BlendTree] GetBlendTrees( )

HdkApi.Data.Object.HomeObject GetObject( )

System.Collections.Generic.List[HdkApi.Data.Object.ObjectResource] GetAnimationResources( )

HdkApi.RepertoireEditor.RepertoireRegister CreateStringRegister(name)

HdkApi.RepertoireEditor.RepertoireRegister CreateBooleanRegister(name)

HdkApi.RepertoireEditor.RepertoireRegister CreateNumberRegister(name)

HdkApi.RepertoireEditor.RepertoireRegister CreateIntegerRegister(name)

Properties

RepertoireCircuitFilepath

RepertoireFilepath

EmotesFilepath

Actions

Behaviours

Transitions

BlendTrees

Events

Registers

Example

```
#<copyright>
#-------------------------------------------------------------------------------------------------------
#   SCE CONFIDENTIAL
#   Home Development Kit (HDK) 1.65.0
#   Copyright (C) 2011 Sony Computer Entertainment Inc.
#   All Rights Reserved.
#-------------------------------------------------------------------------------------------------------
#</copyright>
import System


def UseRepertoire():
    repertoire = RepertoireEditor.CurrentRepertoire
    if(repertoire):
        action = repertoire.CreateAction("action")
        if(action):
            action.Position = Point(500,100)
            print("created action")

        behaviour = repertoire.CreateBehaviour("behaviour")
        if(behaviour):
            behaviour.Position = Point(500,100)
            print("created behaviour")

        emotes = repertoire.CreateGlobalEmotes("global_emotes")
        if(emotes):
            emotes.Position = Point(500,100)
            print("created emotes")

        object = repertoire.GetObject()
        if(object):
            print "Object Name : " + object.Name

        animationResources = repertoire.GetAnimationResources()
        for resource in animationResources:
            print "Object Resource : " + resource.Name

        print("Save")
        repertoire.Save()


        #you could validate your repertoire like this:
        print("Validate")
        results = repertoire.Validate()

        for resultLevel in results.Keys:
            for resultValue in results[resultLevel]:
                print(resultLevel + " : " + resultValue)

        #or handle the exceptions thrown from the validation that happens when you export.
        try:
            repertoire.Export()
        except System.Exception, e:
            print(e.Message)
            print(e.InnerException)

UseRepertoire()
```

HdkApi.RepertoireEditor.RepertoireEditor

RepertoireEditor

The main class for repertoire editor, use it to access currently open repertoires

Namespace: HdkApi.RepertoireEditor

Assembly: Home.Editors.RepertoireEditor.Scripting.dll

Methods

HdkApi.RepertoireEditor.Repertoire RepertoireEditor.Open(homeObject, targetRig)

RepertoireEditor.Close(repertoire)

RepertoireEditor.CloseAll( )

System.Collections.Generic.List[HdkApi.RepertoireEditor.Repertoire] RepertoireEditor.GetOpenRepertoires( )

System.Collections.Generic.List[HdkApi.RepertoireEditor.RepertoireTargetRig] RepertoireEditor.GetRepertoireTargetRigs(homeObject)

Properties

SynchronizationObject

CurrentRepertoire

Events

OnSaving

OnSaved

OnOpened

OnExporting

OnExported

Delegates

RepertoireAction

Example

```
#<copyright>
#--------------------------------------------------------------------------------------------------
#   SCE CONFIDENTIAL
#   Home Development Kit (HDK) 1.65.0
#   Copyright (C) 2011 Sony Computer Entertainment Inc.
#   All Rights Reserved.
#--------------------------------------------------------------------------------------------------
#</copyright>
class TestRepertoireEditor():
    def __init__(self):
        RepertoireEditor.OnExporting += self.exporting
        RepertoireEditor.OnExported += self.exported
        RepertoireEditor.OnSaving += self.saving
        RepertoireEditor.OnSaved += self.saved
        RepertoireEditor.OnOpened += self.opened

    def exporting(self, args):
        print("exporting")

    def exported(self, args):
        print("exported")

    def saving(self, args):
        print("saving")

    def saved(self, args):
        print("saved")

    def opened(self, args):
        print("opened")

    def openARepertoireFromAnObject(self):
        components = ["repertoire"]
        foundObjects = ObjectManager.SearchByComponents(components)
        if(len(foundObjects) > 0):
            #we have a loaded object that contains a repertoire component
            obj = ObjectManager.LoadByUuid(foundObjects[0])

            #we need to get the supported target rigs of the object, a mini game will have male and
female, a full body suit may only have one.
            supportedTargetRigs = RepertoireEditor.GetRepertoireTargetRigs(obj)
            if(len(supportedTargetRigs) > 0):
                RepertoireEditor.Open(obj, supportedTargetRigs[0])


example = TestRepertoireEditor()
example.openARepertoireFromAnObject()
```

HdkApi.RepertoireEditor.RepertoireEvent

RepertoireEvent

Namespace: HdkApi.RepertoireEditor
Assembly: Home.Editors.RepertoireEditor.Scripting.dll

Properties

Name

Values

HdkApi.RepertoireEditor.RepertoireRegister

RepertoireRegister

Namespace: HdkApi.RepertoireEditor
Assembly: Home.Editors.RepertoireEditor.Scripting.dll

Methods

Delete( )

Properties

Name

RepertoireWrite

RepertoireRead

LuaRead

LuaWrite

IsCore

HdkApi.RepertoireEditor.RepertoireTargetRig

RepertoireTargetRig

The rig, emotes and repertoire resources referenced in the repertoire component.

Namespace: HdkApi.RepertoireEditor
Assembly: Home.Editors.RepertoireEditor.Scripting.dll

Properties

Name

RepertoireResource

EmotesResource

HdkApi.RepertoireEditor.State

State

A state can be part of a behaviour or an action

Namespace: HdkApi.RepertoireEditor
Assembly: Home.Editors.RepertoireEditor.Scripting.dll

Methods

HdkApi.RepertoireEditor.Conditions.Condition GetCondition( )

HdkApi.RepertoireEditor.Conditions.Condition CreateCondition(name)

System.Collections.Generic.List[HdkApi.RepertoireEditor.Animation.SkeletalAnimation] GetSkeletalAnimations( )

HdkApi.RepertoireEditor.Animation.SkeletalAnimation CreateSkeletalAnimation(name)

System.Collections.Generic.List[HdkApi.RepertoireEditor.Animation.BlendTreeAnimation] GetBlendTreeAnimations( )

HdkApi.RepertoireEditor.Animation.BlendTreeAnimation CreateBlendTreeAnimation(name)

System.Collections.Generic.List[HdkApi.RepertoireEditor.Outputs.Output] GetOutputs( )

HdkApi.RepertoireEditor.Outputs.Output CreateOutput(name)

SetDuration(duration)

System.Object GetDuration( )

HdkApi.RepertoireEditor.EaseCurve GetEaseIn( )

SetEaseIn(easeCurve)

HdkApi.RepertoireEditor.EaseCurve GetEaseOut( )

SetEaseOut(easeCurve)

Delete( )

Properties

SkeletalAnimations

BlendTreeAnimations

Position

Name

Example

```
#<copyright>
#--------------------------------------------------------------------------------------------
#   SCE CONFIDENTIAL
#   Home Development Kit (HDK) 1.65.0
#   Copyright (C) 2011 Sony Computer Entertainment Inc.
#   All Rights Reserved.
#--------------------------------------------------------------------------------------------
#</copyright>

def CreateState():
    rep = RepertoireEditor.CurrentRepertoire
    if(rep):
        action = rep.CreateAction("Action")
        if(action):
            #set some properties
            action.Name = "Action"
            action.Position = Point(500,100)

            #create a state for this action
            state = action.CreateState("State")
            if(state):
                state.Position = Point(500,100)

                condition = state.CreateCondition("Condition")
                if(condition):
                    condition.Position = Point(350,100)

                animation = state.CreateSkeletalAnimation("Animation")
                if(animation):
                    animation.Position = Point(500,100)
CreateState()
```

HdkApi.RepertoireEditor.StringRegister

StringRegister

A string value register

Namespace: HdkApi.RepertoireEditor
Assembly: Home.Editors.RepertoireEditor.Scripting.dll

Methods

Delete( )

Properties

Values

Name

RepertoireWrite

RepertoireRead

LuaRead

LuaWrite

IsCore

HdkApi.RepertoireEditor.Transition

Transition

Create a transition between 2 animations by specifying a transition animation

Namespace: HdkApi.RepertoireEditor
Assembly: Home.Editors.RepertoireEditor.Scripting.dll

Remarks

 Transitions can be achieved by other means

Methods

HdkApi.Data.Object.ObjectResource GetToAnimation( )

HdkApi.Data.Object.ObjectResource GetFromAnimation( )

HdkApi.RepertoireEditor.Animation.SkeletalAnimation GetSkeletalAnimation( )

Delete( )

Properties

Position

Name

Example

```
#<copyright>
#--------------------------------------------------------------------------------------------------
#   SCE CONFIDENTIAL
#   Home Development Kit (HDK) 1.65.0
#   Copyright (C) 2011 Sony Computer Entertainment Inc.
#   All Rights Reserved.
#--------------------------------------------------------------------------------------------------
#</copyright>

def GetTransitions():
    rep = RepertoireEditor.CurrentRepertoire
    if(rep):
        transitions = rep.GetTransitions()
        for transition in transitions:
            print(transition.Name)
            fromAnimationResource = transition.GetFromAnimation()
            if(fromAnimationResource):
                print("From: " + fromAnimationResource.Name)

            transitionAnimationModule = transition.GetSkeletalAnimation()

            if(transitionAnimationModule):
                transitionAnimationResource = transitionAnimationModule.GetResource()
                if(transitionAnimationResource):
                    print("Via: " + transitionAnimationResource.Name)

            toAnimationResource = transition.GetToAnimation()
            if(toAnimationResource):
                print("To: " + toAnimationResource.Name)

GetTransitions()
```

HdkApi.RepertoireEditor.Animation

Summary

Type
Description
Assembly
BlendTreeAnimation
Blend tree animation used for referencing blend trees
Home.Editors.RepertoireEditor.Scripting
BlendNode
Blends up to 8 animations or blends together to produce one output
Home.Editors.RepertoireEditor.Scripting
BlendTree

A circuit of blend node animations
Home.Editors.RepertoireEditor.Scripting
SkeletalAnimation
Skeletal animation used for referencing animation object resources
Home.Editors.RepertoireEditor.Scripting

HdkApi.RepertoireEditor.Animation.BlendNode

BlendNode

Blends up to 8 animations or blends together to produce one output

Namespace: HdkApi.RepertoireEditor.Animation
Assembly: Home.Editors.RepertoireEditor.Scripting.dll

Methods

HdkApi.Core.Collections.HdkDictionary[System.Int32,HdkApi.RepertoireEditor.Animation.BlendNode] GetBlendNodes( )

HdkApi.RepertoireEditor.Animation.BlendNode GetBlendNode(index)

HdkApi.RepertoireEditor.Animation.BlendNode CreateBlendNode(name, index)

HdkApi.Core.Collections.HdkDictionary[System.Int32,HdkApi.RepertoireEditor.Animation.SkeletalAnimation] GetSkeletalAnimations( )

HdkApi.RepertoireEditor.Animation.SkeletalAnimation GetSkeletalAnimation(index)

HdkApi.RepertoireEditor.Animation.SkeletalAnimation CreateSkeletalAnimation(name, index)

SetRegister(register)

HdkApi.RepertoireEditor.RepertoireRegister GetRegister( )

Delete( )

Properties

Position

Name

Example

```
#<copyright>
#--------------------------------------------------------------------------------------------------
#   SCE CONFIDENTIAL
#   Home Development Kit (HDK) 1.65.0
#   Copyright (C) 2011 Sony Computer Entertainment Inc.
#   All Rights Reserved.
#--------------------------------------------------------------------------------------------------
#</copyright>
def CreateBlendTree():
    blendTree = repertoire.CreateBlendTree(name)
    if(blendTree):
        root = blendTree.GetRootBlendNode()
        if(root):
            child1 = root.CreateBlendNode(0)
            anim0 = child1.CreateSkeletalAnimation("0", 0)
            anim1 = child1.CreateSkeletalAnimation("1", 1)

            child2 = root.CreateBlendNode(1)
            anim2 = child2.CreateSkeletalAnimation("0", 0)
            anim3 = child2.CreateSkeletalAnimation("1", 1)

CreateBlendTree()
```

HdkApi.RepertoireEditor.Animation.BlendTree

BlendTree

A circuit of blend node animations

Namespace: HdkApi.RepertoireEditor.Animation
Assembly: Home.Editors.RepertoireEditor.Scripting.dll

Methods

HdkApi.RepertoireEditor.Animation.BlendNode GetBlendNode( )

HdkApi.RepertoireEditor.Animation.BlendNode CreateBlendNode(name)

Delete( )

Properties

Position

Name

Example

```
#<copyright>
#--------------------------------------------------------------------------------------------------
#   SCE CONFIDENTIAL
#   Home Development Kit (HDK) 1.65.0
#   Copyright (C) 2011 Sony Computer Entertainment Inc.
#   All Rights Reserved.
#--------------------------------------------------------------------------------------------------
#</copyright>
def CreateBlendTree():
    blendTree = repertoire.CreateBlendTree(name)
    if(blendTree):
        root = blendTree.GetRootBlendNode()
        if(root):
            child1 = root.CreateBlendNode(0)
            anim0 = child1.CreateSkeletalAnimation("0", 0)
            anim1 = child1.CreateSkeletalAnimation("1", 1)

            child2 = root.CreateBlendNode(1)
            anim2 = child2.CreateSkeletalAnimation("0", 0)
            anim3 = child2.CreateSkeletalAnimation("1", 1)

CreateBlendTree()
```

HdkApi.RepertoireEditor.Animation.BlendTreeAnimation

BlendTreeAnimation

Blend tree animation used for referencing blend trees

Namespace: HdkApi.RepertoireEditor.Animation
Assembly: Home.Editors.RepertoireEditor.Scripting.dll

Methods

HdkApi.RepertoireEditor.Animation.BlendTree GetBlendTree( )

SetBlendTree(blendTree)

Delete( )

Properties

Position

Name

HdkApi.RepertoireEditor.Animation.SkeletalAnimation

SkeletalAnimation

Skeletal animation used for referencing animation object resources

Namespace: HdkApi.RepertoireEditor.Animation
Assembly: Home.Editors.RepertoireEditor.Scripting.dll

Methods

HdkApi.Data.Object.ObjectResource GetResource( )

SetResource(resource)

Delete( )

Properties

OverrideDuration

Position

Name

HdkApi.RepertoireEditor.Conditions

Summary

Type
Description
Assembly
And
Performs an and operation on inputs
Home.Editors.RepertoireEditor.Scripting
RandomValue
A random value from a selection or range
Home.Editors.RepertoireEditor.Scripting
CompareCurrentBehaviour
Compare a behaviour to the behaviour which is currently playing
Home.Editors.RepertoireEditor.Scripting
CompareCurrentNetworkAge
Compare the current network age to a number
Home.Editors.RepertoireEditor.Scripting
CompareNetworkAge
Compare the network age to a number
Home.Editors.RepertoireEditor.Scripting
ComparePreviousBehaviour
Compare the previous behaviour to behaviour

Home.Editors.RepertoireEditor.Scripting
ComparePreviousState
Compare the previous state to a state
Home.Editors.RepertoireEditor.Scripting
CompareRegister
Compare a register to a value
Home.Editors.RepertoireEditor.Scripting
CompareStateAge
Compare the current state age to a value
Home.Editors.RepertoireEditor.Scripting
ComparisonType
An enumeration of comparison types
Home.Editors.RepertoireEditor.Scripting
Condition
Define condition logic for use with behaviours, actions and states
Home.Editors.RepertoireEditor.Scripting
Or
Performs an or operation on inputs
Home.Editors.RepertoireEditor.Scripting

HdkApi.RepertoireEditor.Conditions.And

And

Performs an and operation on inputs

Namespace: HdkApi.RepertoireEditor.Conditions
Assembly: Home.Editors.RepertoireEditor.Scripting.dll

Methods

System.Collections.Generic.List[System.Object] GetInputs( )

HdkApi.RepertoireEditor.Conditions.CompareCurrentBehaviour CreateCompareCurrentBehaviour( )

HdkApi.RepertoireEditor.Conditions.CompareCurrentNetworkAge CreateCompareCurrentNetworkAge( )

HdkApi.RepertoireEditor.Conditions.CompareNetworkAge CreateCompareNetworkAge( )

HdkApi.RepertoireEditor.Conditions.ComparePreviousBehaviour CreateComparePreviousBehaviour( )

HdkApi.RepertoireEditor.Conditions.ComparePreviousState CreateComparePreviousState( )

HdkApi.RepertoireEditor.Conditions.CompareRegister CreateCompareRegister( )

HdkApi.RepertoireEditor.Conditions.CompareStateAge CreateCompareStateAge( )

HdkApi.RepertoireEditor.Conditions.And CreateAnd( )

HdkApi.RepertoireEditor.Conditions.Or CreateOr( )

HdkApi.RepertoireEditor.Conditions.RandomValue CreateRandomValue( )

Delete( )

Properties

Position

Name

Example

```
#<copyright>
#--------------------------------------------------------------------------------------------
#   SCE CONFIDENTIAL
#   Home Development Kit (HDK) 1.65.0
#   Copyright (C) 2011 Sony Computer Entertainment Inc.
#   All Rights Reserved.
#--------------------------------------------------------------------------------------------
#</copyright>
```

HdkApi.RepertoireEditor.Conditions.CompareCurrentBehaviour

CompareCurrentBehaviour

Compare a behaviour to the behaviour which is currently playing

Namespace: HdkApi.RepertoireEditor.Conditions
Assembly: Home.Editors.RepertoireEditor.Scripting.dll

Methods

SetComparison(comparisonType, behaviour)

IronPython.Runtime.PythonTuple GetComparison( )

Delete( )

Properties

Position

Name

Example

```
#<copyright>
#-----------------------------------------------------------------------------------------------
#   SCE CONFIDENTIAL
#   Home Development Kit (HDK) 1.65.0
#   Copyright (C) 2011 Sony Computer Entertainment Inc.
#   All Rights Reserved.
#-----------------------------------------------------------------------------------------------
#</copyright>
```

HdkApi.RepertoireEditor.Conditions.CompareCurrentNetworkAge

CompareCurrentNetworkAge

Compare the current network age to a number

Namespace: HdkApi.RepertoireEditor.Conditions
Assembly: Home.Editors.RepertoireEditor.Scripting.dll

Methods

SetComparison(comparisonType, value)

IronPython.Runtime.PythonTuple GetComparison( )

Delete( )

Properties

Position

Name

Example

```
#<copyright>
#-----------------------------------------------------------------------------------------------
#   SCE CONFIDENTIAL
#   Home Development Kit (HDK) 1.65.0
#   Copyright (C) 2011 Sony Computer Entertainment Inc.
#   All Rights Reserved.
#-----------------------------------------------------------------------------------------------
#</copyright>
```

HdkApi.RepertoireEditor.Conditions.CompareNetworkAge

CompareNetworkAge

Compare the network age to a number

Namespace: HdkApi.RepertoireEditor.Conditions
Assembly: Home.Editors.RepertoireEditor.Scripting.dll

Methods

SetComparison(comparisonType, value)

IronPython.Runtime.PythonTuple GetComparison( )

Delete( )

Properties

Position

Name

HdkApi.RepertoireEditor.Conditions.CompareBehaviour

CompareBehaviour

Compare the behaviour to behaviour

Namespace: HdkApi.RepertoireEditor.Conditions
Assembly: Home.Editors.RepertoireEditor.Scripting.dll

Methods

SetComparison(comparisonType, behaviour)

IronPython.Runtime.PythonTuple GetComparison( )

Delete( )

Properties

Position

Name

Example

```
#<copyright>
#--------------------------------------------------------------------------------------------------
#   SCE CONFIDENTIAL
#   Home Development Kit (HDK) 1.65.0
#   Copyright (C) 2011 Sony Computer Entertainment Inc.
#   All Rights Reserved.
#--------------------------------------------------------------------------------------------------
#</copyright>
```

HdkApi.RepertoireEditor.Conditions.ComparePreviousBehaviour

ComparePreviousBehaviour

Compare the previous behaviour to behaviour

Namespace: HdkApi.RepertoireEditor.Conditions
Assembly: Home.Editors.RepertoireEditor.Scripting.dll

Methods

SetComparison(comparisonType, behaviour)

IronPython.Runtime.PythonTuple GetComparison( )

Delete( )

Properties

Position

Name

Example

HdkApi.RepertoireEditor.Conditions.ComparePreviousState

ComparePreviousState

Compare the previous state to a state

Namespace: HdkApi.RepertoireEditor.Conditions
Assembly: Home.Editors.RepertoireEditor.Scripting.dll

Methods

SetComparison(comparisonType, state)

IronPython.Runtime.PythonTuple GetComparison( )

Delete( )

Properties

Position

Name

Example

```
#<copyright>
#--------------------------------------------------------------------------------------------------
#   SCE CONFIDENTIAL
#   Home Development Kit (HDK) 1.65.0
#   Copyright (C) 2011 Sony Computer Entertainment Inc.
#   All Rights Reserved.
#--------------------------------------------------------------------------------------------------
#</copyright>
```

HdkApi.RepertoireEditor.Conditions.CompareRegister

CompareRegister

Compare a register to a value

Namespace: HdkApi.RepertoireEditor.Conditions
Assembly: Home.Editors.RepertoireEditor.Scripting.dll

Methods

SetComparison(type, register, value)

IronPython.Runtime.PythonTuple GetComparison( )

Delete( )

Properties

Position

Name

Example

```
#<copyright>
#--------------------------------------------------------------------------------------------------
#   SCE CONFIDENTIAL
#   Home Development Kit (HDK) 1.65.0
#   Copyright (C) 2011 Sony Computer Entertainment Inc.
#   All Rights Reserved.
#--------------------------------------------------------------------------------------------------
#</copyright>
```

HdkApi.RepertoireEditor.Conditions.CompareStateAge

CompareStateAge

Compare the current state age to a value

Namespace: HdkApi.RepertoireEditor.Conditions
Assembly: Home.Editors.RepertoireEditor.Scripting.dll

Methods

SetComparison(comparisonType, value)

IronPython.Runtime.PythonTuple GetComparison( )

Delete( )

Properties

Position

Name

Example

```
#<copyright>
#-------------------------------------------------------------------------------------------------
#   SCE CONFIDENTIAL
#   Home Development Kit (HDK) 1.65.0
#   Copyright (C) 2011 Sony Computer Entertainment Inc.
#   All Rights Reserved.
#-------------------------------------------------------------------------------------------------
#</copyright>
```

HdkApi.RepertoireEditor.Conditions.ComparisonType

ComparisonType

An enumeration of comparison types

Namespace: HdkApi.RepertoireEditor.Conditions
Assembly: Home.Editors.RepertoireEditor.Scripting.dll

Fields

Equal

Not_Equal

Less_Than

Less_Than_Or_Equal_To

Greater_Than

Greater_Than_Or_Equal_To

HdkApi.RepertoireEditor.Conditions.Condition

Condition

Define condition logic for use with behaviours, actions and states

Namespace: HdkApi.RepertoireEditor.Conditions
Assembly: Home.Editors.RepertoireEditor.Scripting.dll

Methods

Delete( )

HdkApi.RepertoireEditor.Conditions.CompareCurrentBehaviour CreateCompareCurrentBehaviour( )

HdkApi.RepertoireEditor.Conditions.CompareCurrentNetworkAge CreateCompareCurrentNetworkAge( )

HdkApi.RepertoireEditor.Conditions.CompareNetworkAge CreateCompareNetworkAge( )

HdkApi.RepertoireEditor.Conditions.ComparePreviousBehaviour CreateComparePreviousBehaviour( )

HdkApi.RepertoireEditor.Conditions.ComparePreviousState CreateComparePreviousState( )

HdkApi.RepertoireEditor.Conditions.CompareRegister CreateCompareRegister( )

HdkApi.RepertoireEditor.Conditions.CompareStateAge CreateCompareStateAge( )

HdkApi.RepertoireEditor.Conditions.And CreateAnd( )

HdkApi.RepertoireEditor.Conditions.Or CreateOr( )

HdkApi.RepertoireEditor.Conditions.RandomValue CreateRandomValue( )

System.Object GetInput( )

Properties

Position

Name

HdkApi.RepertoireEditor.Conditions.Or

Or

Performs an or operation on inputs

Namespace: HdkApi.RepertoireEditor.Conditions
Assembly: Home.Editors.RepertoireEditor.Scripting.dll

Methods

System.Collections.Generic.List[System.Object] GetInputs( )

HdkApi.RepertoireEditor.Conditions.CompareCurrentBehaviour CreateCompareCurrentBehaviour( )

HdkApi.RepertoireEditor.Conditions.CompareCurrentNetworkAge CreateCompareCurrentNetworkAge( )

HdkApi.RepertoireEditor.Conditions.CompareNetworkAge CreateCompareNetworkAge( )

HdkApi.RepertoireEditor.Conditions.ComparePreviousBehaviour CreateComparePreviousBehaviour( )

HdkApi.RepertoireEditor.Conditions.ComparePreviousState CreateComparePreviousState( )

HdkApi.RepertoireEditor.Conditions.CompareRegister CreateCompareRegister( )

HdkApi.RepertoireEditor.Conditions.CompareStateAge CreateCompareStateAge( )

HdkApi.RepertoireEditor.Conditions.And CreateAnd( )

HdkApi.RepertoireEditor.Conditions.Or CreateOr( )

HdkApi.RepertoireEditor.Conditions.RandomValue CreateRandomValue( )

Delete( )

Properties

Position

Name

Example

```
#<copyright>
#-------------------------------------------------------------------------------------------
#   SCE CONFIDENTIAL
#   Home Development Kit (HDK) 1.65.0
#   Copyright (C) 2011 Sony Computer Entertainment Inc.
#   All Rights Reserved.
#-------------------------------------------------------------------------------------------
#</copyright>
```

HdkApi.RepertoireEditor.Conditions.RandomValue

RandomValue

A random value from a selection or range

Namespace: HdkApi.RepertoireEditor.Conditions
Assembly: Home.Editors.RepertoireEditor.Scripting.dll

Methods

SetComparison(type, value, range)

SetComparison(type, value, intSelection)

IronPython.Runtime.PythonTuple GetComparison( )

Delete( )

Properties

Position

Name

Example

```
#<copyright>
#-------------------------------------------------------------------------------------------------
#   SCE CONFIDENTIAL
#   Home Development Kit (HDK) 1.65.0
#   Copyright (C) 2011 Sony Computer Entertainment Inc.
#   All Rights Reserved.
#-------------------------------------------------------------------------------------------------
#</copyright>
```

HdkApi.RepertoireEditor.Emotes

Summary

Type
Description
Assembly
ActionEmote
An emote to trigger an action
Home.Editors.RepertoireEditor.Scripting
BehaviourEmote
An emote to trigger a behaviour
Home.Editors.RepertoireEditor.Scripting

EmoteContext
Group emotes into sub-menus
Home.Editors.RepertoireEditor.Scripting
EmoteGroup
Group emotes into sub-menus
Home.Editors.RepertoireEditor.Scripting
GlobalEmotes
Global emotes are emotes that are available in every region.
Home.Editors.RepertoireEditor.Scripting
RegionalEmotes
Determines which region these emotes will be used
Home.Editors.RepertoireEditor.Scripting

HdkApi.RepertoireEditor.Emotes.ActionEmote

ActionEmote

An emote to trigger an action

Namespace: HdkApi.RepertoireEditor.Emotes
Assembly: Home.Editors.RepertoireEditor.Scripting.dll

Methods

SetAction(action)

HdkApi.RepertoireEditor.Action GetAction( )

Delete( )

Properties

Position

Name

Example

```
#<copyright>
#--------------------------------------------------------------------------------------------------
#   SCE CONFIDENTIAL
#   Home Development Kit (HDK) 1.65.0
#   Copyright (C) 2011 Sony Computer Entertainment Inc.
#   All Rights Reserved.
#--------------------------------------------------------------------------------------------------
#</copyright>
def CreateEmotes():
    rep = RepertoireEditor.CurrentRepertoire
    if(rep):
        print("create a test behaviour and a test action")
        behaviour = rep.CreateBehaviour("Behaviour")
        action = rep.CreateAction("Action")

        if(behaviour and action):
            behaviour.Position = Point(500,100)
            action.Position = Point(500,100)

            globalEmotes = rep.CreateGlobalEmotes("global")
            print("Created global emotes")
            if(globalEmotes):
                globalEmotes.Position = Point(500,100)

                regionalSitIdleContext = globalEmotes.SitIdle
                regionalSitOnFloorContext = globalEmotes.SitOnFloor
                regionalStandIdleContext = globalEmotes.StandIdle

                if(regionalSitIdleContext):
                    someGroup = regionalSitOnFloorContext.CreateEmoteGroup("group")
                    print("Created an emote group within the sit on floor context")
                    if(someGroup):
                        someGroup.Position = Point(500,100)

                        actionEmote = someGroup.CreateActionEmote("action_emote")
                        print("Created an action emote")
                        if(actionEmote):
                            actionEmote.Position = Point(500,100)
                            actionEmote.SetAction(action)

                        behaviourEmote = someGroup.CreateBehaviourEmote("behaviour_emote")
                        print("Created a behaviour emote")
                        if(behaviourEmote):
                            behaviourEmote.Position = Point(500,200)
                            behaviourEmote.SetBehaviour(behaviour)

CreateEmotes()
```

HdkApi.RepertoireEditor.Emotes.BehaviourEmote

BehaviourEmote

An emote to trigger a behaviour

Namespace: HdkApi.RepertoireEditor.Emotes
Assembly: Home.Editors.RepertoireEditor.Scripting.dll

Methods

SetBehaviour(behaviour)

HdkApi.RepertoireEditor.Behaviour GetBehaviour( )

Delete( )

Properties

Position

Name

Example

```
#<copyright>
#-------------------------------------------------------------------------------------------------
#   SCE CONFIDENTIAL
#   Home Development Kit (HDK) 1.65.0
#   Copyright (C) 2011 Sony Computer Entertainment Inc.
#   All Rights Reserved.
#-------------------------------------------------------------------------------------------------
#</copyright>
def CreateEmotes():
    rep = RepertoireEditor.CurrentRepertoire
    if(rep):
        print("create a test behaviour and a test action")
        behaviour = rep.CreateBehaviour("Behaviour")
        action = rep.CreateAction("Action")

        if(behaviour and action):
            behaviour.Position = Point(500,100)
            action.Position = Point(500,100)

            globalEmotes = rep.CreateGlobalEmotes("global")
            print("Created global emotes")
            if(globalEmotes):
                globalEmotes.Position = Point(500,100)

                regionalSitIdleContext = globalEmotes.SitIdle
                regionalSitOnFloorContext = globalEmotes.SitOnFloor
                regionalStandIdleContext = globalEmotes.StandIdle

                if(regionalSitIdleContext):
                    someGroup = regionalSitOnFloorContext.CreateEmoteGroup("group")
                    print("Created an emote group within the sit on floor context")
                    if(someGroup):
                        someGroup.Position = Point(500,100)

                        actionEmote = someGroup.CreateActionEmote("action_emote")
                        print("Created an action emote")
                        if(actionEmote):
                            actionEmote.Position = Point(500,100)
                            actionEmote.SetAction(action)

                        behaviourEmote = someGroup.CreateBehaviourEmote("behaviour_emote")
                        print("Created a behaviour emote")
                        if(behaviourEmote):
                            behaviourEmote.Position = Point(500,200)
                            behaviourEmote.SetBehaviour(behaviour)

CreateEmotes()
```

HdkApi.RepertoireEditor.Emotes.EmoteContext

EmoteContext

Group emotes into sub-menus

Namespace: HdkApi.RepertoireEditor.Emotes
Assembly: Home.Editors.RepertoireEditor.Scripting.dll

Remarks

The available emote contexts are specified by the client and may not be created, deleted or renamed.

Methods

HdkApi.RepertoireEditor.Emotes.EmoteGroup CreateEmoteGroup(groupName)

System.Collections.Generic.List[HdkApi.RepertoireEditor.Emotes.EmoteGroup] GetEmoteGroups( )

Delete( )

Properties

Position

Name

Example

```
#<copyright>
#-------------------------------------------------------------------------------------------------------
#   SCE CONFIDENTIAL
#   Home Development Kit (HDK) 1.65.0
#   Copyright (C) 2011 Sony Computer Entertainment Inc.
#   All Rights Reserved.
#-------------------------------------------------------------------------------------------------------
#</copyright>
def CreateEmotes():
    rep = RepertoireEditor.CurrentRepertoire
    if(rep):
        print("create a test behaviour and a test action")
        behaviour = rep.CreateBehaviour("Behaviour")
        action = rep.CreateAction("Action")

        if(behaviour and action):
            behaviour.Position = Point(500,100)
            action.Position = Point(500,100)

            globalEmotes = rep.CreateGlobalEmotes("global")
            print("Created global emotes")
            if(globalEmotes):
                globalEmotes.Position = Point(500,100)

                regionalSitIdleContext = globalEmotes.SitIdle
                regionalSitOnFloorContext = globalEmotes.SitOnFloor
                regionalStandIdleContext = globalEmotes.StandIdle

                if(regionalSitIdleContext):
                    someGroup = regionalSitOnFloorContext.CreateEmoteGroup("group")
                    print("Created an emote group within the sit on floor context")
                    if(someGroup):
                        someGroup.Position = Point(500,100)

                        actionEmote = someGroup.CreateActionEmote("action_emote")
                        print("Created an action emote")
                        if(actionEmote):
                            actionEmote.Position = Point(500,100)
                            actionEmote.SetAction(action)

                        behaviourEmote = someGroup.CreateBehaviourEmote("behaviour_emote")
                        print("Created a behaviour emote")
                        if(behaviourEmote):
                            behaviourEmote.Position = Point(500,200)
                            behaviourEmote.SetBehaviour(behaviour)

CreateEmotes()
```

HdkApi.RepertoireEditor.Emotes.EmoteGroup

EmoteGroup

Group emotes into sub-menus

Namespace: HdkApi.RepertoireEditor.Emotes
Assembly: Home.Editors.RepertoireEditor.Scripting.dll

Methods

System.Collections.Generic.List[HdkApi.RepertoireEditor.Emotes.ActionEmote] GetActionEmotes( )

HdkApi.RepertoireEditor.Emotes.ActionEmote CreateActionEmote(emoteName)

System.Collections.Generic.List[HdkApi.RepertoireEditor.Emotes.BehaviourEmote] GetBehaviourEmotes( )

HdkApi.RepertoireEditor.Emotes.BehaviourEmote CreateBehaviourEmote(emoteName)

Delete( )

Properties

Position

Name

Example

```
#<copyright>
#--------------------------------------------------------------------------------------------------
#   SCE CONFIDENTIAL
#   Home Development Kit (HDK) 1.65.0
#   Copyright (C) 2011 Sony Computer Entertainment Inc.
#   All Rights Reserved.
#--------------------------------------------------------------------------------------------------
#</copyright>
def CreateEmotes():
    rep = RepertoireEditor.CurrentRepertoire
    if(rep):
        print("create a test behaviour and a test action")
        behaviour = rep.CreateBehaviour("Behaviour")
        action = rep.CreateAction("Action")

        if(behaviour and action):
            behaviour.Position = Point(500,100)
            action.Position = Point(500,100)

            globalEmotes = rep.CreateGlobalEmotes("global")
            print("Created global emotes")
            if(globalEmotes):
                globalEmotes.Position = Point(500,100)

                regionalSitIdleContext = globalEmotes.SitIdle
                regionalSitOnFloorContext = globalEmotes.SitOnFloor
                regionalStandIdleContext = globalEmotes.StandIdle

                if(regionalSitIdleContext):
                    someGroup = regionalSitOnFloorContext.CreateEmoteGroup("group")
                    print("Created an emote group within the sit on floor context")
                    if(someGroup):
                        someGroup.Position = Point(500,100)

                        actionEmote = someGroup.CreateActionEmote("action_emote")
                        print("Created an action emote")
                        if(actionEmote):
                            actionEmote.Position = Point(500,100)
                            actionEmote.SetAction(action)

                        behaviourEmote = someGroup.CreateBehaviourEmote("behaviour_emote")
                        print("Created a behaviour emote")
                        if(behaviourEmote):
                            behaviourEmote.Position = Point(500,200)
                            behaviourEmote.SetBehaviour(behaviour)

CreateEmotes()
```

HdkApi.RepertoireEditor.Emotes.GlobalEmotes

GlobalEmotes

Global emotes are emotes that are available in every region.

Namespace: HdkApi.RepertoireEditor.Emotes
Assembly: Home.Editors.RepertoireEditor.Scripting.dll

Methods

Delete( )

Properties

StandIdle

SitIdle

SitOnFloor

Position

Name

Example

```
#<copyright>
#-------------------------------------------------------------------------------------------------
#   SCE CONFIDENTIAL
#   Home Development Kit (HDK) 1.65.0
#   Copyright (C) 2011 Sony Computer Entertainment Inc.
#   All Rights Reserved.
#-------------------------------------------------------------------------------------------------
#</copyright>
def CreateEmotes():
    rep = RepertoireEditor.CurrentRepertoire
    if(rep):
        print("create a test behaviour and a test action")
        behaviour = rep.CreateBehaviour("Behaviour")
        action = rep.CreateAction("Action")

        if(behaviour and action):
            behaviour.Position = Point(500,100)
            action.Position = Point(500,100)

            globalEmotes = rep.CreateGlobalEmotes("global")
            print("Created global emotes")
            if(globalEmotes):
                globalEmotes.Position = Point(500,100)

                regionalSitIdleContext = globalEmotes.SitIdle
                regionalSitOnFloorContext = globalEmotes.SitOnFloor
                regionalStandIdleContext = globalEmotes.StandIdle

                if(regionalSitIdleContext):
                    someGroup = regionalSitOnFloorContext.CreateEmoteGroup("group")
                    print("Created an emote group within the sit on floor context")
                    if(someGroup):
                        someGroup.Position = Point(500,100)

                        actionEmote = someGroup.CreateActionEmote("action_emote")
                        print("Created an action emote")
                        if(actionEmote):
                            actionEmote.Position = Point(500,100)
                            actionEmote.SetAction(action)

                        behaviourEmote = someGroup.CreateBehaviourEmote("behaviour_emote")
                        print("Created a behaviour emote")
                        if(behaviourEmote):
                            behaviourEmote.Position = Point(500,200)
                            behaviourEmote.SetBehaviour(behaviour)

CreateEmotes()
```

HdkApi.RepertoireEditor.Emotes.RegionalEmotes

RegionalEmotes

Determines which region these emotes will be used

Namespace: HdkApi.RepertoireEditor.Emotes
Assembly: Home.Editors.RepertoireEditor.Scripting.dll

Methods

Delete( )

Properties

Region

StandIdle

SitIdle

SitOnFloor

Position

Name

Example

```
#<copyright>
#--------------------------------------------------------------------------------------------------
#   SCE CONFIDENTIAL
#   Home Development Kit (HDK) 1.65.0
#   Copyright (C) 2011 Sony Computer Entertainment Inc.
#   All Rights Reserved.
#--------------------------------------------------------------------------------------------------
#</copyright>
def CreateEmotes():
    rep = RepertoireEditor.CurrentRepertoire
    if(rep):
        print("create a test behaviour and a test action")
        behaviour = rep.CreateBehaviour("Behaviour")
        action = rep.CreateAction("Action")

        if(behaviour and action):
            behaviour.Position = Point(500,100)
            action.Position = Point(500,100)

            regionalEmotes = rep.CreateRegionalEmotes("japan")
            print("Created regional emotes")
            if(regionalEmotes):
                regionalEmotes.Position = Point(500,100)
                regionalEmotes.Region = "ja-JP"

                regionalSitIdleContext = regionalEmotes.SitIdle
                regionalSitOnFloorContext = regionalEmotes.SitOnFloor
                regionalStandIdleContext = regionalEmotes.StandIdle

                if(regionalSitIdleContext):
                    someGroup = regionalSitIdleContext.CreateEmoteGroup("group")
                    print("Created an emote group within the sit idle context")
                    if(someGroup):
                        someGroup.Position = Point(500,100)

                        actionEmote = someGroup.CreateActionEmote("action_emote")
                        print("Created an action emote")
                        if(actionEmote):
                            actionEmote.Position = Point(500,100)
                            actionEmote.SetAction(action)

                        behaviourEmote = someGroup.CreateBehaviourEmote("behaviour_emote")
                        print("Created a behaviour emote")
                        if(behaviourEmote):
                            behaviourEmote.Position = Point(500,200)
                            behaviourEmote.SetBehaviour(behaviour)

CreateEmotes()
```

HdkApi.RepertoireEditor.Outputs

Summary

Type
Description
Assembly
ClearRegister
Clear a register value as part of an output

Home.Editors.RepertoireEditor.Scripting
OutputType
Whether onEntry or onExit
Home.Editors.RepertoireEditor.Scripting
Output
Outputs are a way of defining behaviour before or after an action/behaviour/state is triggered
Home.Editors.RepertoireEditor.Scripting
SetNextBehaviour
Set the next behaviour as part of an output
Home.Editors.RepertoireEditor.Scripting
SetRegister
Set a register value as part of an output
Home.Editors.RepertoireEditor.Scripting
SpawnAction
Spawn an action as part of an output
Home.Editors.RepertoireEditor.Scripting
TriggerEvent
Trigger an event as part of an output
Home.Editors.RepertoireEditor.Scripting

HdkApi.RepertoireEditor.Outputs.ClearRegister

ClearRegister

Clear a register value as part of an output

Namespace: HdkApi.RepertoireEditor.Outputs
Assembly: Home.Editors.RepertoireEditor.Scripting.dll

Methods

SetRegister(register)

HdkApi.RepertoireEditor.RepertoireRegister GetRegister( )

Delete( )

Properties

Position

Name

Example

```
#<copyright>
#-------------------------------------------------------------------------------------------------
#   SCE CONFIDENTIAL
#   Home Development Kit (HDK) 1.65.0
#   Copyright (C) 2011 Sony Computer Entertainment Inc.
#   All Rights Reserved.
#-------------------------------------------------------------------------------------------------
#</copyright>

def CreateOutput():
    repertoire = RepertoireEditor.CurrentRepertoire

    #create action
    action = repertoire.CreateAction("action")

    #add an output
    output = action.CreateOutput("output")

    #add a few output types
    clearRegister = output.CreateClearRegister("clearRegister", OutputType.OnEntry)
 clearRegister.SetRegister(register)

    setNextBehaviour = output.CreateSetNextBehaviour("setNextBehaviour", OutputType.OnEntry)
 setNextBehaviour.SetBehaviour(behaviour)

    setRegister = output.CreateSetRegister("setRegister", OutputType.OnEntry)
 setRegister.SetValue(register, value)

    spawnAction = output.CreateSpawnAction("spawnAction", OutputType.OnExit)
 spawnAction.SetAction(action)

    triggerEvent = output.CreateTriggerEvent("triggerEvent", OutputType.OnExit)
 triggerEvent.SetEvent(event)

CreateOutput()
```

HdkApi.RepertoireEditor.Outputs.Output

Output

Outputs are a way of defining behaviour before or after an action/behaviour/state is triggered

Namespace: HdkApi.RepertoireEditor.Outputs
Assembly: Home.Editors.RepertoireEditor.Scripting.dll

Methods

System.Collections.Generic.List[HdkApi.RepertoireEditor.Outputs.BaseOutput] GetOutputs(type)

HdkApi.RepertoireEditor.Outputs.ClearRegister CreateClearRegister(type)

HdkApi.RepertoireEditor.Outputs.SetNextBehaviour CreateSetNextBehaviour(type)

HdkApi.RepertoireEditor.Outputs.SetRegister CreateSetRegister(type)

HdkApi.RepertoireEditor.Outputs.SpawnAction CreateSpawnAction(type)

HdkApi.RepertoireEditor.Outputs.TriggerEvent CreateTriggerEvent(type)

Delete( )

Properties

Position

Name

Example

```
#<copyright>
#--------------------------------------------------------------------------------------------
#   SCE CONFIDENTIAL
#   Home Development Kit (HDK) 1.65.0
#   Copyright (C) 2011 Sony Computer Entertainment Inc.
#   All Rights Reserved.
#--------------------------------------------------------------------------------------------
#</copyright>

def CreateOutput():
    repertoire = RepertoireEditor.CurrentRepertoire

    #create action
    action = repertoire.CreateAction("action")

    #add an output
    output = action.CreateOutput("output")

    #add a few output types
    clearRegister = output.CreateClearRegister("clearRegister", OutputType.OnEntry)
 clearRegister.SetRegister(register)

    setNextBehaviour = output.CreateSetNextBehaviour("setNextBehaviour", OutputType.OnEntry)
 setNextBehaviour.SetBehaviour(behaviour)

    setRegister = output.CreateSetRegister("setRegister", OutputType.OnEntry)
 setRegister.SetValue(register, value)

    spawnAction = output.CreateSpawnAction("spawnAction", OutputType.OnExit)
 spawnAction.SetAction(action)

    triggerEvent = output.CreateTriggerEvent("triggerEvent", OutputType.OnExit)
 triggerEvent.SetEvent(event)

CreateOutput()
```

HdkApi.RepertoireEditor.Outputs.OutputType

OutputType

Whether onEntry or onExit

Namespace: HdkApi.RepertoireEditor.Outputs
Assembly: Home.Editors.RepertoireEditor.Scripting.dll

Fields

OnEntry

OnExit

Example

```
#<copyright>
#-------------------------------------------------------------------------------------------------
#   SCE CONFIDENTIAL
#   Home Development Kit (HDK) 1.65.0
#   Copyright (C) 2011 Sony Computer Entertainment Inc.
#   All Rights Reserved.
#-------------------------------------------------------------------------------------------------
#</copyright>

def CreateOutput():
    repertoire = RepertoireEditor.CurrentRepertoire

    #create action
    action = repertoire.CreateAction("action")

    #add an output
    output = action.CreateOutput("output")

    #add a few output types
    clearRegister = output.CreateClearRegister("clearRegister", OutputType.OnEntry)
 clearRegister.SetRegister(register)

    setNextBehaviour = output.CreateSetNextBehaviour("setNextBehaviour", OutputType.OnEntry)
 setNextBehaviour.SetBehaviour(behaviour)

    setRegister = output.CreateSetRegister("setRegister", OutputType.OnEntry)
 setRegister.SetValue(register, value)

    spawnAction = output.CreateSpawnAction("spawnAction", OutputType.OnExit)
 spawnAction.SetAction(action)

    triggerEvent = output.CreateTriggerEvent("triggerEvent", OutputType.OnExit)
 triggerEvent.SetEvent(event)

CreateOutput()
```

HdkApi.RepertoireEditor.Outputs.SetNextBehaviour

SetNextBehaviour

Set the next behaviour as part of an output

Namespace: HdkApi.RepertoireEditor.Outputs
Assembly: Home.Editors.RepertoireEditor.Scripting.dll

Methods

HdkApi.RepertoireEditor.Behaviour GetBehaviour( )

SetBehaviour(behaviour)

Delete( )

Properties

Position

Name

Example

```
#<copyright>
#--------------------------------------------------------------------------------------------------
#   SCE CONFIDENTIAL
#   Home Development Kit (HDK) 1.65.0
#   Copyright (C) 2011 Sony Computer Entertainment Inc.
#   All Rights Reserved.
#--------------------------------------------------------------------------------------------------
#</copyright>

def CreateOutput():
    repertoire = RepertoireEditor.CurrentRepertoire

    #create action
    action = repertoire.CreateAction("action")

    #add an output
    output = action.CreateOutput("output")

    #add a few output types
    clearRegister = output.CreateClearRegister("clearRegister", OutputType.OnEntry)
 clearRegister.SetRegister(register)

    setNextBehaviour = output.CreateSetNextBehaviour("setNextBehaviour", OutputType.OnEntry)
 setNextBehaviour.SetBehaviour(behaviour)

    setRegister = output.CreateSetRegister("setRegister", OutputType.OnEntry)
 setRegister.SetValue(register, value)

    spawnAction = output.CreateSpawnAction("spawnAction", OutputType.OnExit)
 spawnAction.SetAction(action)

    triggerEvent = output.CreateTriggerEvent("triggerEvent", OutputType.OnExit)
 triggerEvent.SetEvent(event)

CreateOutput()
```

HdkApi.RepertoireEditor.Outputs.SetRegister

SetRegister

Set a register value as part of an output

Namespace: HdkApi.RepertoireEditor.Outputs
Assembly: Home.Editors.RepertoireEditor.Scripting.dll

Methods

SetValue(register, value)

IronPython.Runtime.PythonTuple GetValue( )

Delete( )

Properties

Position

Name

Example

```
#<copyright>
#--------------------------------------------------------------------------------------------------
#   SCE CONFIDENTIAL
#   Home Development Kit (HDK) 1.65.0
#   Copyright (C) 2011 Sony Computer Entertainment Inc.
#   All Rights Reserved.
#--------------------------------------------------------------------------------------------------
#</copyright>

def CreateOutput():
    repertoire = RepertoireEditor.CurrentRepertoire

    #create action
    action = repertoire.CreateAction("action")

    #add an output
    output = action.CreateOutput("output")

    #add a few output types
    clearRegister = output.CreateClearRegister("clearRegister", OutputType.OnEntry)
 clearRegister.SetRegister(register)

    setNextBehaviour = output.CreateSetNextBehaviour("setNextBehaviour", OutputType.OnEntry)
 setNextBehaviour.SetBehaviour(behaviour)

    setRegister = output.CreateSetRegister("setRegister", OutputType.OnEntry)
 setRegister.SetValue(register, value)

    spawnAction = output.CreateSpawnAction("spawnAction", OutputType.OnExit)
 spawnAction.SetAction(action)

    triggerEvent = output.CreateTriggerEvent("triggerEvent", OutputType.OnExit)
 triggerEvent.SetEvent(event)

CreateOutput()
```

HdkApi.RepertoireEditor.Outputs.SpawnAction

SpawnAction

Spawn an action as part of an output

Namespace: HdkApi.RepertoireEditor.Outputs
Assembly: Home.Editors.RepertoireEditor.Scripting.dll

Methods

HdkApi.RepertoireEditor.Action GetAction( )

SetAction(action)

Delete( )

Properties

Position

Name

Example

```
#<copyright>
#--------------------------------------------------------------------------------------------------
#   SCE CONFIDENTIAL
#   Home Development Kit (HDK) 1.65.0
#   Copyright (C) 2011 Sony Computer Entertainment Inc.
#   All Rights Reserved.
#--------------------------------------------------------------------------------------------------
#</copyright>

def CreateOutput():
    repertoire = RepertoireEditor.CurrentRepertoire

    #create action
    action = repertoire.CreateAction("action")

    #add an output
    output = action.CreateOutput("output")

    #add a few output types
    clearRegister = output.CreateClearRegister("clearRegister", OutputType.OnEntry)
 clearRegister.SetRegister(register)

    setNextBehaviour = output.CreateSetNextBehaviour("setNextBehaviour", OutputType.OnEntry)
 setNextBehaviour.SetBehaviour(behaviour)

    setRegister = output.CreateSetRegister("setRegister", OutputType.OnEntry)
 setRegister.SetValue(register, value)

    spawnAction = output.CreateSpawnAction("spawnAction", OutputType.OnExit)
 spawnAction.SetAction(action)

    triggerEvent = output.CreateTriggerEvent("triggerEvent", OutputType.OnExit)
 triggerEvent.SetEvent(event)

CreateOutput()
```

HdkApi.RepertoireEditor.Outputs.TriggerEvent

TriggerEvent

Trigger an event as part of an output

Namespace: HdkApi.RepertoireEditor.Outputs
Assembly: Home.Editors.RepertoireEditor.Scripting.dll

Methods

SetEvent(repertoireEvent, value)

IronPython.Runtime.PythonTuple GetEvent( )

Delete( )

Properties

Position

Name

Example

```
#<copyright>
#--------------------------------------------------------------------------------------------------
#   SCE CONFIDENTIAL
#   Home Development Kit (HDK) 1.65.0
#   Copyright (C) 2011 Sony Computer Entertainment Inc.
#   All Rights Reserved.
#--------------------------------------------------------------------------------------------------
#</copyright>

def CreateOutput():
    repertoire = RepertoireEditor.CurrentRepertoire

    #create action
    action = repertoire.CreateAction("action")

    #add an output
    output = action.CreateOutput("output")

    #add a few output types
    clearRegister = output.CreateClearRegister("clearRegister", OutputType.OnEntry)
 clearRegister.SetRegister(register)

    setNextBehaviour = output.CreateSetNextBehaviour("setNextBehaviour", OutputType.OnEntry)
 setNextBehaviour.SetBehaviour(behaviour)

    setRegister = output.CreateSetRegister("setRegister", OutputType.OnEntry)
 setRegister.SetValue(register, value)

    spawnAction = output.CreateSpawnAction("spawnAction", OutputType.OnExit)
 spawnAction.SetAction(action)

    triggerEvent = output.CreateTriggerEvent("triggerEvent", OutputType.OnExit)
 triggerEvent.SetEvent(event)

CreateOutput()
```

HdkApi.RepertoireEditor.UserInterface

Summary

Type
Description
Assembly
Menu
Menu represents a menu in the Scene Editor UI, new items can be added to the menu by calling the AddMenuItem function, a new Menu can be created through MenuManager.CreateMenu
Home.Editors.RepertoireEditor.Scripting
MenuItem
MenuItem represents an item within a Menu in the Scene Editor UI, it provides access to properties about the menu item such as what icon it uses and whether it has a button associated with it, for a menu item to be visible it must be added to a Menu through AddMenuItem, to create a new MenuItem call MenuManager.CreateMenuItem

Home.Editors.RepertoireEditor.Scripting

MenuManager

MenuManager is used to add new menus and menu items to the Scene Editor UI, to add a new menu call CreateMenu, to add a new menu item call CreateMenuItem and then pass the returned MenuItem to a menu's AddMenuItem function

Home.Editors.RepertoireEditor.Scripting

HdkApi.RepertoireEditor.UserInterface.Menu

Menu

Menu represents a menu in the Scene Editor UI, new items can be added to the menu by calling the AddMenuItem function, a new Menu can be created through MenuManager.CreateMenu

Namespace: HdkApi.RepertoireEditor.UserInterface
Assembly: Home.Editors.RepertoireEditor.Scripting.dll

Methods

AddMenuItem(menuItem)

Example

```
#<copyright>
#--------------------------------------------------------------------------------------------------
#   SCE CONFIDENTIAL
#   Home Development Kit (HDK) 1.65.0
#   Copyright (C) 2011 Sony Computer Entertainment Inc.
#   All Rights Reserved.
#--------------------------------------------------------------------------------------------------
#</copyright>
# Also add the System.Windows.Forms dll and import Keys so we can set a keyboard shortcut
clr.AddReference("System.Windows.Forms")
from System.Windows.Forms import Keys

class UiExample():
    def __init__(self):
        # Create a new menu
        menu = MenuManager.CreateMenu("My Hdk API Scripts")

        actionMenuItem = self.createActionMenu()
        behaviourMenuItem = self.createBehaviourMenu()

        # Add the MenuItem to the example menu
        menu.AddMenuItem(actionMenuItem)
        menu.AddMenuItem(behaviourMenuItem)

    def createActionMenu(self):
        # Create a new MenuItem
        itemDisplayText = "CreateAction"
        menuItem = MenuManager.CreateMenuItem(itemDisplayText)
        menuItem.Description = "Create an action with one state"
        menuItem.Toggleable = True
        menuItem.Shortcut = Keys.Control | Keys.Shift | Keys.A
        #menuItem.HasToolbarButton = True

        # Hook up the execute event to set what happens when the menu item is pressed
        menuItem.Execute += self.createAction

        # Hook up the can execute event to set whether the menu item is enabled
```

```
        menuItem.CanExecute += self.repertoireCanExecute

        return menuItem

    def createBehaviourMenu(self):
        # Create a new MenuItem
        itemDisplayText = "CreateBehaviour"
        menuItem = MenuManager.CreateMenuItem(itemDisplayText)
        menuItem.Description = "Create a behaviour with one state"
        menuItem.Toggleable = True
        menuItem.Shortcut = Keys.Control | Keys.Shift | Keys.B
        #menuItem.HasToolbarButton = True

        # Hook up the execute event to set what happens when the menu item is pressed
        menuItem.Execute += self.createBehaviour

        # Hook up the can execute event to set whether the menu item is enabled
        menuItem.CanExecute += self.repertoireCanExecute

        return menuItem


    def repertoireCanExecute(self):
        rep = RepertoireEditor.CurrentRepertoire
        if(rep is None):
            return False
        return True

    def createBehaviour(self, sender, args):
        rep = RepertoireEditor.CurrentRepertoire
        if(rep):
            behaviour = rep.CreateBehaviour("behaviour")
            if(behaviour):
                behaviour.Position = Point(500,100)

                state = behaviour.CreateState("state")
                if(state):
                    state.Position = Point(500,100)

        print("Created behaviour")

    def createAction(self, sender, args):
        rep = RepertoireEditor.CurrentRepertoire
        if(rep):
            action = rep.CreateAction("action")
            action.Position = Point(416,192)

            state = action.CreateState("state")
            if(state):
                state.Position = Point(500,100)
```

```
        print("Created action")

example = UiExample()
```

Associated Types

- HdkApi.RepertoireEditor.UserInterface.MenuManager
- HdkApi.RepertoireEditor.UserInterface.MenuItem

HdkApi.RepertoireEditor.UserInterface.MenuItem

MenuItem

MenuItem represents an item within a Menu in the Scene Editor UI, it provides access to properties about the menu item such as what icon it uses and whether it has a button associated with it, for a menu item to be visible it must be added to a Menu through AddMenuItem, to create a new MenuItem call MenuManager.CreateMenuItem

Namespace: HdkApi.RepertoireEditor.UserInterface
Assembly: Home.Editors.RepertoireEditor.Scripting.dll

Properties

DisplayText

Description

IconPath

Shortcut

Toggleable

HasToolbarButton

Events

Execute

CanExecute

Delegates

CanExecuteHandler

Example

```
#<copyright>
#----------------------------------------------------------------------------------------------
#   SCE CONFIDENTIAL
#   Home Development Kit (HDK) 1.65.0
#   Copyright (C) 2011 Sony Computer Entertainment Inc.
#   All Rights Reserved.
#----------------------------------------------------------------------------------------------
#</copyright>
# Also add the System.Windows.Forms dll and import Keys so we can set a keyboard shortcut
clr.AddReference("System.Windows.Forms")
from System.Windows.Forms import Keys

class UiExample():
    def __init__(self):
        # Create a new menu
        menu = MenuManager.CreateMenu("My Hdk API Scripts")
```

```
        actionMenuItem = self.createActionMenu()
        behaviourMenuItem = self.createBehaviourMenu()

        # Add the MenuItem to the example menu
        menu.AddMenuItem(actionMenuItem)
        menu.AddMenuItem(behaviourMenuItem)

    def createActionMenu(self):
        # Create a new MenuItem
        itemDisplayText = "CreateAction"
        menuItem = MenuManager.CreateMenuItem(itemDisplayText)
        menuItem.Description = "Create an action with one state"
        menuItem.Toggleable = True
        menuItem.Shortcut = Keys.Control | Keys.Shift | Keys.A
        #menuItem.HasToolbarButton = True

        # Hook up the execute event to set what happens when the menu item is pressed
        menuItem.Execute += self.createAction

        # Hook up the can execute event to set whether the menu item is enabled
        menuItem.CanExecute += self.repertoireCanExecute

        return menuItem

    def createBehaviourMenu(self):
        # Create a new MenuItem
        itemDisplayText = "CreateBehaviour"
        menuItem = MenuManager.CreateMenuItem(itemDisplayText)
        menuItem.Description = "Create a behaviour with one state"
        menuItem.Toggleable = True
        menuItem.Shortcut = Keys.Control | Keys.Shift | Keys.B
        #menuItem.HasToolbarButton = True

        # Hook up the execute event to set what happens when the menu item is pressed
        menuItem.Execute += self.createBehaviour

        # Hook up the can execute event to set whether the menu item is enabled
        menuItem.CanExecute += self.repertoireCanExecute

        return menuItem


    def repertoireCanExecute(self):
        rep = RepertoireEditor.CurrentRepertoire
        if(rep is None):
            return False
        return True

    def createBehaviour(self, sender, args):
        rep = RepertoireEditor.CurrentRepertoire
        if(rep):
            behaviour = rep.CreateBehaviour("behaviour")
            if(behaviour):
                behaviour.Position = Point(500,100)

                state = behaviour.CreateState("state")
                if(state):
                    state.Position = Point(500,100)

        print("Created behaviour")

    def createAction(self, sender, args):
        rep = RepertoireEditor.CurrentRepertoire
        if(rep):
            action = rep.CreateAction("action")
            action.Position = Point(416,192)

            state = action.CreateState("state")
            if(state):
```

```
state.Position = Point(500,100)
```

```
        print("Created action")

example = UiExample()
```

Associated Types

- HdkApi.RepertoireEditor.UserInterface.MenuManager
- HdkApi.RepertoireEditor.UserInterface.Menu

HdkApi.RepertoireEditor.UserInterface.MenuManager

MenuManager

MenuManager is used to add new menus and menu items to the Scene Editor UI, to add a new menu call CreateMenu, to add a new menu item call CreateMenuItem and then pass the returned MenuItem to a menu's AddMenuItem function

Namespace: HdkApi.RepertoireEditor.UserInterface
Assembly: Home.Editors.RepertoireEditor.Scripting.dll

Methods

HdkApi.RepertoireEditor.UserInterface.Menu MenuManager.CreateMenu(name)

HdkApi.RepertoireEditor.UserInterface.Menu MenuManager.GetMenu(name)

HdkApi.RepertoireEditor.UserInterface.MenuItem MenuManager.CreateMenuItem(displayText)

Example

```
#<copyright>
#----------------------------------------------------------------------------------------------------
#   SCE CONFIDENTIAL
#   Home Development Kit (HDK) 1.65.0
#   Copyright (C) 2011 Sony Computer Entertainment Inc.
#   All Rights Reserved.
#----------------------------------------------------------------------------------------------------
#</copyright>
# Also add the System.Windows.Forms dll and import Keys so we can set a keyboard shortcut
clr.AddReference("System.Windows.Forms")
from System.Windows.Forms import Keys

class UiExample():
    def __init__(self):
        # Create a new menu
        menu = MenuManager.CreateMenu("My Hdk API Scripts")

        actionMenuItem = self.createActionMenu()
        behaviourMenuItem = self.createBehaviourMenu()

        # Add the MenuItem to the example menu
        menu.AddMenuItem(actionMenuItem)
        menu.AddMenuItem(behaviourMenuItem)

    def createActionMenu(self):
        # Create a new MenuItem
        itemDisplayText = "CreateAction"
        menuItem = MenuManager.CreateMenuItem(itemDisplayText)
        menuItem.Description = "Create an action with one state"
        menuItem.Toggleable = True
        menuItem.Shortcut = Keys.Control | Keys.Shift | Keys.A
        #menuItem.HasToolbarButton = True
```

```
    # Hook up the execute event to set what happens when the menu item is pressed
    menuItem.Execute += self.createAction

    # Hook up the can execute event to set whether the menu item is enabled
    menuItem.CanExecute += self.repertoireCanExecute

    return menuItem

def createBehaviourMenu(self):
    # Create a new MenuItem
    itemDisplayText = "CreateBehaviour"
    menuItem = MenuManager.CreateMenuItem(itemDisplayText)
    menuItem.Description = "Create a behaviour with one state"
    menuItem.Toggleable = True
    menuItem.Shortcut = Keys.Control | Keys.Shift | Keys.B
    #menuItem.HasToolbarButton = True

    # Hook up the execute event to set what happens when the menu item is pressed
    menuItem.Execute += self.createBehaviour

    # Hook up the can execute event to set whether the menu item is enabled
    menuItem.CanExecute += self.repertoireCanExecute

    return menuItem


def repertoireCanExecute(self):
    rep = RepertoireEditor.CurrentRepertoire
    if(rep is None):
        return False
    return True

def createBehaviour(self, sender, args):
    rep = RepertoireEditor.CurrentRepertoire
    if(rep):
        behaviour = rep.CreateBehaviour("behaviour")
        if(behaviour):
            behaviour.Position = Point(500,100)

            state = behaviour.CreateState("state")
            if(state):
                state.Position = Point(500,100)

    print("Created behaviour")

def createAction(self, sender, args):
    rep = RepertoireEditor.CurrentRepertoire
    if(rep):
        action = rep.CreateAction("action")
        action.Position = Point(416,192)

        state = action.CreateState("state")
        if(state):
            state.Position = Point(500,100)
```

```
        print("Created action")

example = UiExample()
```

Associated Types

- HdkApi.RepertoireEditor.UserInterface.Menu
- HdkApi.RepertoireEditor.UserInterface.MenuItem

HdkApi.SceneEditor

Summary

Type
Description
Assembly
SceneEditor
Provides access to core functionality of the Scene Editor, such as getting the currently open scene.
Home.Editors.SceneEditor.Scripting

HdkApi.SceneEditor.SceneEditor

SceneEditor

Provides access to core functionality of the Scene Editor, such as getting the currently open scene.

Namespace: HdkApi.SceneEditor
Assembly: Home.Editors.SceneEditor.Scripting.dll

Methods

SceneEditor.OpenSceneProject(scprojPath)

System.Boolean SceneEditor.Export(validate)

Properties

CurrentScene

Events

OnSaving

OnSaved

OnOpened

OnExporting

OnExported

OnPackaging

OnPackaged

Delegates

SceneAction

Example

```
#<copyright>
#-------------------------------------------------------------------------------------------------
#   SCE CONFIDENTIAL
#   Home Development Kit (HDK) 1.70.0
#   Copyright (C) 2011 Sony Computer Entertainment Inc.
#   All Rights Reserved.
#-------------------------------------------------------------------------------------------------
#</copyright>

# First import the SceneEditor class
from HdkApi.SceneEditor import SceneEditor

# Import the HomeScene class
from HdkApi.Data.Scene import HomeScene

class SceneEditorExample():
    def __init__(self):
        # Get the current scene which is open in the Scene Editor
        openScene = SceneEditor.CurrentScene

        # Print the scene's filepath
        print openScene.FilePath

        # Export the currently open scene to a (.scene) file so it can be tested in game
        validate = True
        succeeded = SceneEditor.Export(validate)
        print succeeded

        self.RegisterCallbacks()

    def RegisterCallbacks(self):
        # Hook up to the OnOpened event called when a scene is opened in the Scene Editor
        SceneEditor.OnOpened += self.onOpened

        # Hook up to the OnSaving event called when a scene is about to be saved
        SceneEditor.OnSaving += self.onSaving

        # Hook up to the OnSaved event called when a scene has been successfully saved
        SceneEditor.OnSaved += self.onSaved

        # Hook up to the OnExporting event called when a scene is about to be exported from an .scproj
file to a .scene file
        SceneEditor.OnExporting += self.onExporting

        # Hook up to the OnExported event called when a scene has been successfully exported from an
.scproj file to a .scene file
        SceneEditor.OnExported += self.onExported

        # Hook up to the OnPackaging event called when a scene is about to be packaged
        SceneEditor.OnPackaging += self.onPackaging

        # Hook up to the OnPackaged event called when a scene has been successfully packaged
        SceneEditor.OnPackaged += self.onPackaged

    # Example of a callback function triggered when a scene is opened in the Scene Editor
    def onOpened(self, scene):
        print "Opened: {0}".format(scene.FilePath)

    # Example of a callback function triggered when the scene open in the Scene Editor is about to be
```

```
saved
    def onSaving(self, scene):
        print "Saving: {0}".format(scene.FilePath)

    # Example of a callback function triggered when the scene open in the Scene Editor has been
successfully saved
    def onSaved(self, scene):
        print "Saved: {0}".format(scene.FilePath)

    # Example of a callback function triggered when the scene open in the Scene Editor is about to be
exported
    def onExporting(self, scene):
        print "Exporting: {0}".format(scene.FilePath)

    # Example of a callback function triggered when the scene open in the Scene Editor has been
successfully exported
    def onExported(self, scene):
        print "Exported: {0}".format(scene.FilePath)

    # Example of a callback function triggered when the scene open in the Scene Editor is about to be
packaged
    def onPackaging(self, scene):
        print "Packaging: {0}".format(scene.FilePath)

    # Example of a callback function triggered when the scene open in the Scene Editor is about to be
```

```
packaged
    def onPackaged(self, scene):
        print "Packaged: {0}".format(scene.FilePath)
```

Associated Types

- HdkApi.Data.Scene.HomeScene

HdkApi.SceneEditor.Camera

Summary

Type
Description
Assembly
Camera
The Camera class is the basis for derived camera classes in the HdkApi such as SceneCamera.
Home.Editors.SceneEditor.Scripting
CameraManager
The CameraManager has access to Camera objects from the SceneEditor. Currently it provides access to the perspective camera that is used to navigate in the scene editor.
Home.Editors.SceneEditor.Scripting
SceneCamera
The SceneCamera class represents the perspective camera that is used to navigate the scene in the editor. You may use this class to retrieve position and look at point data.
Home.Editors.SceneEditor.Scripting

HdkApi.SceneEditor.Camera.Camera

Camera

The Camera class is the basis for derived camera classes in the HdkApi such as SceneCamera.

Namespace: HdkApi.SceneEditor.Camera
Assembly: Home.Editors.SceneEditor.Scripting.dll

Properties

Position

LookAt

Events

CameraChanged

Example

```
#<copyright>
#-------------------------------------------------------------------------------------------------
#   SCE CONFIDENTIAL
#   Home Development Kit (HDK) 1.70.0
#   Copyright (C) 2011 Sony Computer Entertainment Inc.
#   All Rights Reserved.
#-------------------------------------------------------------------------------------------------
#</copyright>

# First import the Camera classes
from HdkApi.SceneEditor.Camera import CameraManager, SceneCamera

class CameraExample():
 def __init__(self):
  self.camera = CameraManager.SceneCamera
  self.camera.CameraChanged += self.cameraUpdated

 def printPosition(self):
  str = "{0}{1}{2}".format(self.camera.Position.X, self.camera.Position.Y, self.camera.Position.Z)
  print(str)

 def printLookAt(self):
  str = "{0}{1}{2}".format(self.camera.LookAt.X, self.camera.LookAt.Y, self.camera.LookAt.Z)
  print(str)

 def cameraUpdated(self, sender, args):
  print("Camera Updated")
  self.printPosition()
  self.printLookAt()

camExample = CameraExample()
```

Associated Types

- HdkApi.SceneEditor.Camera.CameraManager
- HdkApi.SceneEditor.Camera.SceneCamera

HdkApi.SceneEditor.Camera.CameraManager

CameraManager

The CameraManager has access to Camera objects from the SceneEditor. Currently it provides access to the perspective camera that is used to navigate in the scene editor.

Namespace: HdkApi.SceneEditor.Camera
Assembly: Home.Editors.SceneEditor.Scripting.dll

Properties

SceneCamera

Example

```
#<copyright>
#--------------------------------------------------------------------------------------------------
#   SCE CONFIDENTIAL
#   Home Development Kit (HDK) 1.70.0
#   Copyright (C) 2011 Sony Computer Entertainment Inc.
#   All Rights Reserved.
#--------------------------------------------------------------------------------------------------
#</copyright>

# First import the Camera classes
from HdkApi.SceneEditor.Camera import CameraManager, SceneCamera

class CameraExample():
 def __init__(self):
  self.camera = CameraManager.SceneCamera
  self.camera.CameraChanged += self.cameraUpdated

 def printPosition(self):
  str = "{0}{1}{2}".format(self.camera.Position.X, self.camera.Position.Y, self.camera.Position.Z)
  print(str)

 def printLookAt(self):
  str = "{0}{1}{2}".format(self.camera.LookAt.X, self.camera.LookAt.Y, self.camera.LookAt.Z)
  print(str)

 def cameraUpdated(self, sender, args):
  print("Camera Updated")
  self.printPosition()
  self.printLookAt()

camExample = CameraExample()
```

Associated Types

- HdkApi.SceneEditor.Camera.Camera
- HdkApi.SceneEditor.Camera.SceneCamera

HdkApi.SceneEditor.Camera.SceneCamera

SceneCamera

The SceneCamera class represents the perspective camera that is used to navigate the scene in the editor. You may use this class to retrieve position and look at point data.

Namespace: HdkApi.SceneEditor.Camera
Assembly: Home.Editors.SceneEditor.Scripting.dll

Methods

ZoomToSelection( )

Properties

Position

LookAt

Events

CameraChanged

Example

```
#<copyright>
#--------------------------------------------------------------------------------------------------
#   SCE CONFIDENTIAL
#   Home Development Kit (HDK) 1.70.0
#   Copyright (C) 2011 Sony Computer Entertainment Inc.
#   All Rights Reserved.
#--------------------------------------------------------------------------------------------------
#</copyright>

# First import the Camera classes
from HdkApi.SceneEditor.Camera import CameraManager, SceneCamera

class CameraExample():
 def __init__(self):
  self.camera = CameraManager.SceneCamera
  self.camera.CameraChanged += self.cameraUpdated

 def printPosition(self):
  str = "{0}{1}{2}".format(self.camera.Position.X, self.camera.Position.Y, self.camera.Position.Z)
  print(str)

 def printLookAt(self):
  str = "{0}{1}{2}".format(self.camera.LookAt.X, self.camera.LookAt.Y, self.camera.LookAt.Z)
  print(str)

 def cameraUpdated(self, sender, args):
  print("Camera Updated")
  self.printPosition()
  self.printLookAt()

camExample = CameraExample()
```

Associated Types

- HdkApi.SceneEditor.Camera.Camera
- HdkApi.SceneEditor.Camera.CameraManager

HdkApi.SceneEditor.Selection

Summary

Type
Description
Assembly
SelectionManager
Provides events and data relating to the current selection in the Scene Editor
Home.Editors.SceneEditor.Scripting

HdkApi.SceneEditor.Selection.SelectionManager

SelectionManager

Provides events and data relating to the current selection in the Scene Editor

Namespace: HdkApi.SceneEditor.Selection
Assembly: Home.Editors.SceneEditor.Scripting.dll

Events

SelectionChanged

Example

```
#<copyright>
#--------------------------------------------------------------------------------------------------
#   SCE CONFIDENTIAL
#   Home Development Kit (HDK) 1.70.0
#   Copyright (C) 2011 Sony Computer Entertainment Inc.
#   All Rights Reserved.
#--------------------------------------------------------------------------------------------------
#</copyright>

# First import the Selection classes
from HdkApi.SceneEditor.Selection import SelectionManager

class SelectionExample():
    def __init__(self):
        # Hook up to the SelectionChanged event
        SelectionManager.SelectionChanged += self.selectionChanged

    def selectionChanged(self, sender, args):
        # Print a message when ever the selection in the Scene Editor changes
        print("Selection Changed")

selExample = SelectionExample()
```

HdkApi.SceneEditor.UserInterface

Summary

Type	Description	Assembly
Menu	Menu represents a menu in the Scene Editor UI, new items can be added to the menu by calling the AddMenuItem function, a new Menu can be created through MenuManager.CreateMenu	Home.Editors.SceneEditor.Scripting
MenuItem	MenuItem represents an item within a Menu in the Scene Editor UI, it provides access to properties about the menu item such as what icon it uses and whether it has a button associated with it, for a menu item to be visible it must be added to a Menu through AddMenuItem, to create a new MenuItem call MenuManager.CreateMenuItem	Home.Editors.SceneEditor.Scripting

MenuManager
MenuManager is used to add new menus and menu items to the Scene Editor UI, to add a new menu call CreateMenu, to add a new menu item call CreateMenuItem and then pass the returned MenuItem to a menu's AddMenuItem function
Home.Editors.SceneEditor.Scripting

HdkApi.SceneEditor.UserInterface.Menu

Menu

Menu represents a menu in the Scene Editor UI, new items can be added to the menu by calling the AddMenuItem function, a new Menu can be created through MenuManager.CreateMenu

Namespace: HdkApi.SceneEditor.UserInterface
Assembly: Home.Editors.SceneEditor.Scripting.dll

Methods

AddMenuItem(menuItem)

Example

```
#<copyright>
#-------------------------------------------------------------------------------------------------------
#   SCE CONFIDENTIAL
#   Home Development Kit (HDK) 1.70.0
#   Copyright (C) 2011 Sony Computer Entertainment Inc.
#   All Rights Reserved.
#-------------------------------------------------------------------------------------------------------
#</copyright>

# First import the UserInterface classes
from HdkApi.SceneEditor.UserInterface import MenuManager, Menu, MenuItem

# Also add the System.Windows.Forms dll and import Keys so we can set a keyboard shortcut
clr.AddReference("System.Windows.Forms")
from System.Windows.Forms import Keys

class UiExample():
    def __init__(self):
        # Create a new menu
        menu = MenuManager.CreateMenu("Example Menu")

        # Create a new MenuItem
        itemDisplayText = "Example Item"
        menuItem = MenuManager.CreateMenuItem(itemDisplayText)
        menuItem.Description = "Example menu item used in the UI Example"
        menuItem.Toggleable = True
        menuItem.Shortcut = Keys.Control | Keys.Shift | Keys.M
        menuItem.HasToolbarButton = True

        # Hook up the execute event to set what happens when the menu item is pressed
        menuItem.Execute += self.exampleExecute

        # Hook up the can execute event to set whether the menu item is enabled
        menuItem.CanExecute += self.exampleCanExecute

        # Add the MenuItem to the example menu
        menu.AddMenuItem(menuItem)

    def exampleCanExecute(self):
        return True

    def exampleExecute(self, sender, args):
        print("Example executed")

example = UiExample()
```

Associated Types

- HdkApi.SceneEditor.UserInterface.MenuManager
- HdkApi.SceneEditor.UserInterface.MenuItem

HdkApi.SceneEditor.UserInterface.MenuItem

MenuItem

MenuItem represents an item within a Menu in the Scene Editor UI, it provides access to properties about the menu item such as what icon it uses and whether it has a button associated with it, for a menu item to be visible it must be added to a Menu through AddMenuItem, to create a new MenuItem call MenuManager.CreateMenuItem

Namespace: HdkApi.SceneEditor.UserInterface
Assembly: Home.Editors.SceneEditor.Scripting.dll

Properties

DisplayText

Description

IconPath

Shortcut

Toggleable

HasToolbarButton

Events

Execute

CanExecute

Delegates

CanExecuteHandler

Example

```
#<copyright>
#-------------------------------------------------------------------------------------------------------
#   SCE CONFIDENTIAL
#   Home Development Kit (HDK) 1.70.0
#   Copyright (C) 2011 Sony Computer Entertainment Inc.
#   All Rights Reserved.
#-------------------------------------------------------------------------------------------------------
#</copyright>

# First import the UserInterface classes
from HdkApi.SceneEditor.UserInterface import MenuManager, Menu, MenuItem

# Also add the System.Windows.Forms dll and import Keys so we can set a keyboard shortcut
clr.AddReference("System.Windows.Forms")
from System.Windows.Forms import Keys

class UiExample():
    def __init__(self):
        # Create a new menu
        menu = MenuManager.CreateMenu("Example Menu")

        # Create a new MenuItem
        itemDisplayText = "Example Item"
        menuItem = MenuManager.CreateMenuItem(itemDisplayText)
        menuItem.Description = "Example menu item used in the UI Example"
        menuItem.Toggleable = True
        menuItem.Shortcut = Keys.Control | Keys.Shift | Keys.M
        menuItem.HasToolbarButton = True

        # Hook up the execute event to set what happens when the menu item is pressed
        menuItem.Execute += self.exampleExecute

        # Hook up the can execute event to set whether the menu item is enabled
        menuItem.CanExecute += self.exampleCanExecute

        # Add the MenuItem to the example menu
        menu.AddMenuItem(menuItem)

    def exampleCanExecute(self):
        return True

    def exampleExecute(self, sender, args):
        print("Example executed")

example = UiExample()
```

Associated Types

- HdkApi.SceneEditor.UserInterface.MenuManager
- HdkApi.SceneEditor.UserInterface.Menu

HdkApi.SceneEditor.UserInterface.MenuManager

MenuManager

MenuManager is used to add new menus and menu items to the Scene Editor UI, to add a new menu call CreateMenu, to add a new menu item call CreateMenuItem and then pass the returned MenuItem to a menu's AddMenuItem function

Namespace: HdkApi.SceneEditor.UserInterface
Assembly: Home.Editors.SceneEditor.Scripting.dll

Methods

HdkApi.SceneEditor.UserInterface.Menu MenuManager.CreateMenu(name)

HdkApi.SceneEditor.UserInterface.Menu MenuManager.GetMenu(name)

HdkApi.SceneEditor.UserInterface.MenuItem MenuManager.CreateMenuItem(displayText)

Example

```
#<copyright>
#-------------------------------------------------------------------------------------------------
#   SCE CONFIDENTIAL
#   Home Development Kit (HDK) 1.70.0
#   Copyright (C) 2011 Sony Computer Entertainment Inc.
#   All Rights Reserved.
#-------------------------------------------------------------------------------------------------
#</copyright>

# First import the UserInterface classes
from HdkApi.SceneEditor.UserInterface import MenuManager, Menu, MenuItem

# Also add the System.Windows.Forms dll and import Keys so we can set a keyboard shortcut
clr.AddReference("System.Windows.Forms")
from System.Windows.Forms import Keys

class UiExample():
    def __init__(self):
        # Create a new menu
        menu = MenuManager.CreateMenu("Example Menu")

        # Create a new MenuItem
        itemDisplayText = "Example Item"
        menuItem = MenuManager.CreateMenuItem(itemDisplayText)
        menuItem.Description = "Example menu item used in the UI Example"
        menuItem.Toggleable = True
        menuItem.Shortcut = Keys.Control | Keys.Shift | Keys.M
        menuItem.HasToolbarButton = True

        # Hook up the execute event to set what happens when the menu item is pressed
        menuItem.Execute += self.exampleExecute

        # Hook up the can execute event to set whether the menu item is enabled
        menuItem.CanExecute += self.exampleCanExecute

        # Add the MenuItem to the example menu
        menu.AddMenuItem(menuItem)

    def exampleCanExecute(self):
        return True

    def exampleExecute(self, sender, args):
        print("Example executed")

example = UiExample()
```

Associated Types

- HdkApi.SceneEditor.UserInterface.Menu
- HdkApi.SceneEditor.UserInterface.MenuItem

HdkApi.SceneEditor.Utilities

Summary

Type

Description
Assembly
SynchronizationObject
Provides a synchronization object for the main Scene Editor thread, this is useful for forcing a delegate to be executed on the main Scene Editor thread, for more information about synchronization objects see: http://msdn.microsoft.com/en-us/library/system.componentmodel.isynchronizeinvoke.aspx
Home.Editors.SceneEditor.Scripting

HdkApi.SceneEditor.Utilities.SynchronizationObject

SynchronizationObject

Provides a synchronization object for the main Scene Editor thread, this is useful for forcing a delegate to be executed on the main Scene Editor thread, for more information about synchronization objects see: http://msdn.microsoft.com/en-us/library/system.componentmodel.isynchronizeinvoke.aspx

Namespace: HdkApi.SceneEditor.Utilities
Assembly: Home.Editors.SceneEditor.Scripting.dll

Remarks

 SynchronizationObject is an advanced class which you should only have to use if you are dealing with multithreading, in general you should rarely need to use this class

Methods

System.ComponentModel.ISynchronizeInvoke SynchronizationObject.Get( )

Example

```
#<copyright>
#--------------------------------------------------------------------------------------------------
#   SCE CONFIDENTIAL
#   Home Development Kit (HDK) 1.70.0
#   Copyright (C) 2011 Sony Computer Entertainment Inc.
#   All Rights Reserved.
#--------------------------------------------------------------------------------------------------
#</copyright>

# First import the SynchronizationObject class
from HdkApi.SceneEditor.Utilities import SynchronizationObject

# We also use the communications classes in this example, so import them as well
from HdkApi.Core.Communication import TargetManager, Target, TargetResult, TargetException

class SyncObjectExample():
    def __init__(self):
        # Create a timer
        self.timerInterval = 30
        self.timer = Timer()
        self.timer.Interval = (1000/timerInterval)

        # Initialise the TargetManager so it is ready to be used
        TargetManager.Initialize()

        # Use the SynchronizationObject to force the Elapsed callback to run on the main Scene Editor
thread
        # Normally the Elapsed callback would execute on a separate thread, but by setting the
SynchronizingObject to
        # the main Scene Editor thread timerEvent will now be called on the Scene Editor thread
instead
        self.timer.SynchronizingObject = SynchronizationObject.Get()

        self.timer.Elapsed += self.timerEvent

    def timerEvent(self, sender, arg):
        # TargetManager can only be called on the thread it was created on (the main Scene Editor
thread),
        # by setting the SynchronizingObject we know that it will be called on the correct thread
        TargetManager.ConnectToDefault()
```

Samples

We have several samples you can download to help you when you're developing your spaces and objects.

We've divided them up into 7 sections:

Games

These samples demonstrate complete games, both minigames and arcade games, with multiple players, multiple win/loss conditions, game states and so on.

- Arcade Tutorial
- Advanced Arcade Game
- Simon Minigame
- Tic Tac Toe Minigame

Collision/Physics

These samples all demonstrate ways to implement Home's collision and physics system using Lua, Maya and Havok. Useful for any kind of interactive 3D game or scene.

- Action Forces
- Collision Callbacks
- Collision Layers
- Collision User IDs

- Entity States
- Raycasting

Art API

These samples demonstrate how our Lua libraries can be used in combination with the art tools to improve your scene's artwork by dynamically changing aspects of your scene.

- 2D Blending
- Blend Shader
- Dynamic Lighting
- Engine Settings
- Entity Animation Blending
- Entity Attachment
- Material API
- Scene Asset Loading
- Scripted Sky

Tools

These samples all show practical demonstrations of in-built tools systems and pipelines

- Baked Lighting
- Cloth Tool
- Companion
- Static LODs
- Environment dimensions and materials
- Object Instances
- Repertoires
- Simple Animations
- Water Effects

Media

These samples all show various media which you can create/access through the HDK

- Media Library
- Screens

Systems

These samples all show Client or PS3 systems which can be accessed via the HDK and Lua

- NP Ticketing
- OSD
- OSK
- Pad API
- Rewards

Full Scenes

These samples are all full scenes which demonstrate ways to combine all aspects of the HDK to create high-quality publishable scenes

- Day/Night Cycle
- The HTS Aeronautilus