
Lumberyard

User Guide

Version 1.6



Lumberyard: User Guide

Copyright ©

Table of Contents

What is Lumberyard?	1
Professional-Grade AAA Engine	1
Beautiful Worlds	1
Compelling Characters	1
Robust Networking	2
Real-time Gameplay Editing	2
Modular gems	2
Wwise LTX	2
and more...	2
Integrated with AWS	2
Amazon GameLift	2
Cloud Canvas	2
AWS SDK for C++	3
Integrated with Twitch	3
Twitch ChatPlay	3
Twitch JoinIn	3
Free, with Source	3
Lumberyard Systems	3
Lumberyard Editors and Tools	4
Lumberyard Asset File Types	5
Setting Up Lumberyard	9
System Requirements	9
Downloading Lumberyard	10
Using the Lumberyard Installer to Download Lumberyard	10
Upgrading Lumberyard	11
Upgrading Lumberyard with an Existing Version in Source Control	12
Upgrading Lumberyard without an Existing Version in Source Control	12
Upgrading Lumberyard without Source Control	12
Upgrading Your Game Projects	13
Files to Exclude When Upgrading Lumberyard	13
Using Lumberyard Setup Assistant to Set Up Your Development Environment	14
Running Lumberyard Setup Assistant	15
Using Lumberyard Setup Assistant Batch	16
Customizing Lumberyard Setup Assistant	18
Enabling a Firewall	20
Migrating Lumberyard Projects	21
Lumberyard 1.6	21
Migrating GridMate Service Sessions	21
Migrating from CryUnitTest to AzTest	22
Lumberyard 1.5	23
Migrating Your Project	23
Migrating Your Gems	24
Using Lumberyard Editor	33
Lumberyard Editor Interface	34
Viewport	34
Toolbars	35
Rollup Bar	35
Bottom Toolbar	35
Console	36
Using the Menu Bar in Lumberyard Editor	36
File Menu	37
File Configure Menu	37
Edit Menu	38
Modify Menu	39
Display Menu	39

AI Menu	40
Audio Menu	41
Clouds Menu	41
Game Menu	42
Physics Menu	42
Prefabs Menu	42
Terrain Menu	43
Tools Menu	43
View Menu	44
AWS Menu	45
Commerce Menu	45
Help Menu	45
Using the Top Toolbars	45
EditMode Toolbar	46
Object Toolbar	46
Editors Toolbar	46
Using the Bottom Toolbar	47
Status	47
Lock Selection	47
Coordinates/Transforms	47
Set Vector	48
Speed Control	48
Terrain Collision	48
AI/Physics	48
No Sync Player	48
Goto Position	48
Mute Audio	49
VR Preview	49
Using Shortcut Keys	49
Using the Viewport	51
Changing the View	51
Using the Rollup Bar	52
Objects Tab	52
Terrain Tab	52
Modeling Tab	54
Render/Debug Tab	54
Layers Tab	56
Using the Console Window	56
Configuring Console Variables	57
Customizing Your Workspace	58
Docking Windows and Toolbars	58
Customizing Toolbars and Menus	58
Changing Preferences	60
Restoring Default Settings for Lumberyard Editor	66
AI System	68
Spawning AI Agents	68
Using Flow Graph to Spawn AI Agents	69
Using Auto Disable for Agents	69
Debugging Agent Spawning Issues	70
AI Navigation	70
Multi-Layer Navigation Mesh (MNM)	71
Creating Navigation Areas	71
Selecting an AI Navigation Type	72
Setting Navigation Exclusion Areas	72
Adding Navigation Seed Points	73
Using Flow Graph for AI Navigation	73
Regenerating the Navigation Mesh	73
Off-Mesh AI Navigation	74

Tutorial: Basic AI Navigation	75
Debugging AI Navigation	76
Agent Perception	77
Using Flow Graph to Set Agent Perception	78
Using AI Anchors to Set Agent Perception	78
Using Console Variables to Set Agent Perception	79
Debugging AI Agent Perception Issues	80
AI Communications	80
Using Database View to Set AI Communication	80
Using AI Communication Channels	81
Using the CommConfig Property	82
Using GoalPipes to Trigger Communication	83
Using Voice Libraries for AI Speech	83
Using Flow Graph for Setting AI Communications	83
Using AI Signals Among Agents	83
AI Modular Behavior Tree	85
Standard MBT Nodes	86
Common AI MBT Nodes	90
Game AI MBT Nodes	105
Helicopter AI MBT Nodes	110
AI Agent Debugging	115
Using the AI Debug Recorder	115
Using the AI Debug Viewer	116
Using AI Debug Console Variables	118
Using AI Bubbles for Error Messaging	121
Using AILog and AISignals Files	122
Asset Pipeline	123
Asset Processor	124
Configuration	125
Batch Processing	125
Debugging	125
Live Reloading and VFS	125
Shader Compiler Proxy	126
Game Startup Sequence	126
Missing Asset Resolver Tool	127
Technical Information: Asset IDs and File Paths	127
Asset IDs and File Paths	127
Converting Asset IDs to Full Paths	128
Live Update Messages	129
Audio System	130
Audio System Architecture	131
Installing Audiokinetic Wwise LTX	131
Installing Wwise LTX	132
Running the Wwise LTX Authoring Tool	132
Using the Audio Controls Editor	132
Using Audiokinetic Wwise LTX	134
ATL Default Controls	135
do_nothing control	135
get_focus control	135
lose_focus control	135
mute_all control	135
object_speed control	135
object_velocity_tracking control	135
ObstructionOcclusionCalculationType control	136
unmute_all control	136
Audio PlayTriggers and StopTriggers	136
Placing Triggers in Game	136
PlayTrigger Set	136

StopTrigger Set	136
Both Triggers Set	137
Obstructing and Occluding Sounds	137
Obstructing Sounds	137
Sound Obstruction for Surface Types	138
Occluding Sounds	138
Raycasts	138
Debugging Raycasts	138
Audio Flow Graph Nodes	139
Adding Ambient Sounds to Levels	139
Setting Up the AudioAreaAmbience entity	140
Setting Up the AudioAreaEntity entity	141
Using Shape Priorities	142
Adding Reverb Effects to Levels	142
Setting Distance Values	143
Adding Collision Sounds to Levels	144
Adding Sound to Trackview Sequences	145
Adding Sound to Animations	145
Adding Sound to Mannequin	146
Audio Console Variables Commands	147
Characters and Animation	151
Working With Character Assets	151
Modeling Characters	152
Rigging Characters	154
Physicalizing Characters (Ragdoll)	157
Using Inverse Kinematics (IK)	168
Maya Export Tools	177
Accessing Maya Export Tools	178
Setting Time Working Units for Maya	179
Geometry Validation	179
Exporting Static Meshes	179
Exporting Characters	181
Exporting Materials	183
Exporting Animations	184
Exporting Blendshapes	185
Exporting Level of Details (LODs)	185
Exporting an Alembic Cache	187
Setting Export Options	188
3ds Max Export Tools	189
Exporting Static Meshes and Characters	189
Exporting Materials	191
Exporting Bones and Animations	191
Exporting Levels of Detail (LODs)	193
Configuring Log Options	194
Working with the FBX Importer	195
Importing Materials with the FBX Importer	197
Importing Physics Mesh for a Static Object	198
Additional FBX Importer Features and Settings	200
Using Rules	201
Using Multiple UV Streams	203
Using Geppetto	204
Geppetto Display Options	207
Creating a Character Definition	210
Character Attachments	211
Animating Characters	224
Mannequin System	247
Mannequin System Files	247
Creating a Mannequin Entity	250

Using Mannequin Editor	250
Synchronizing Multiple Characters	276
Using Flow Graph with Mannequin	277
Debugging Mannequin System Issues	277
Cinematics System	279
Cinematics Best Practices	279
Using Track View Editor	280
Track View Toolbars	280
Using Cutscene Animation Curves (Curve Editor)	281
Track View Nodes	281
Comment Node	281
Console Variable Node	282
Director (Scene) Node	282
Entity Nodes	284
Environment Node	286
Event Node	286
Material Node	287
Script Variable Node	288
Shadows Setup Node	288
Full Screen Effect Nodes	289
Creating Scenes	291
Setting Sequence Properties	291
Playing a Sequence	294
Changing Playback Speed	295
Managing Track Events	295
Linking Track View Events with Flow Graph	296
Cinematics Cameras	296
Moving a Camera	297
Setting Camera Focus	297
Creating Camera Shake	298
Blending a Camera	299
Pointing a Camera	301
Following with a Camera	301
Setting a First Person View Camera	302
Importing a Camera from Autodesk	302
Exporting a Camera to Autodesk	303
Cinematics Lighting	304
Animating a Light	304
Cinematic Lighting Best Practices	305
Animating Characters in Scenes	306
Importing and Exporting Transform Animations	306
Adding Geometry to a Sequence	307
Animated Character Tracks in Cutscenes	308
Moving an Entity in a Scene	308
Adding Scene Character Attachments	309
Using Look IK in Scenes	309
Blending Cinematic Animations	311
Using Track View Animation Curves	311
Pre-caching Cinematic Animations	312
Adding Player Interactivity	313
Looping and Jumping in a Scene	313
Pausing a Scene	314
Adding a Dead-Man Switch to a Scene	315
Setting Player Look Around	316
Adding Force Feedback	317
Using Layers for Scenes	318
Capturing Image Frames	318
Capturing Image Frames using Render Output	318

Capturing Image Frames using a Capture Track	318
Capturing Image Frames using Console Variables	319
Debugging Cinematic Scenes	319
Component Entity System	320
Component Palette	320
Component Palette Attributes	322
Entity Outliner	323
Parenting	323
Filtering	323
Slices	323
Entity Inspector	323
File Browser	324
Asset Drag and Drop	325
Filtering	326
File Operations	326
Component Reference	327
Attachment Component	328
Audio Environment Component	329
Audio Rtpc Component	330
Audio Switch Component	331
Audio Trigger Component	332
Behavior Tree Component	334
Camera Component	336
Camera Rig Component	336
Camera Target Component	340
Constraint Component	341
Decal Component	349
Event Action Binding Component	350
Flow Graph Component	353
Input Configuration Component	354
Lens Flare Component	358
Light Component	361
Lua Script Component	365
Mannequin Component	367
Mannequin Scope Context Component	373
Navigation Component	374
Particle Component	377
Character Physics Component	379
Physics Component	384
Mesh Collider Component	386
Primitive Collider Component	387
Rag Doll Component	387
Shapes Components	390
Simple Animation Component	392
Simple State Component	397
Skinned Mesh Component	399
Spawner Component	400
Static Mesh Component	402
Tag Component	404
Trigger Area Component	406
Working with Entities	410
Creating an Entity	410
Adding and Removing Components	410
Finding an Entity	411
Editing Component Properties	412
Working with Slices	413
Creating a Slice	413
Instantiating a Slice	413

Modifying a Slice and Pushing Changes	413
Cloning a Slice	414
Inheriting a Slice (Data Cascading)	414
Slice Reloading	415
Object and Entity System	416
Using the Designer Tool	416
Designer Tool Settings	417
Selection Tools	419
Shape Tools	420
Edit Tools	422
Modify Tools	422
Texture Tools	423
Miscellaneous Tools	425
Using the Measurement System Tool	425
Using the Object Selector	426
Finding an Object	426
Object Selector Table	428
Brushes	429
DrawLast	430
Prefabs	430
Common Parameters and Properties	432
Entity Properties	432
Entity Parameters	432
Scripting and Flow Graph Entity Parameters	433
Entity Links	434
Entity Events	435
Attached Entities	436
Shape Parameters	437
Entity Reference	438
Actor Entity	439
AI Control Objects	439
Anim Entities	444
Archetype Entity	444
Area Entities	445
Audio Entities	451
Boid Entity	457
Camera Entities	460
Geom Entities	460
Light Entities	461
Lightning Arc Entity	465
Miscellaneous Entities	467
Particle Entities	469
Physics Entities	470
Rain Entity	478
Render Entities	479
River Entity	480
Road Entity	481
Rope Entity	482
Snow Entity	484
Tornado Entity	484
Trigger Entities	485
Flow Graph System	487
Using Flow Graph Editor	487
Flow Graph Scripts	488
Level Flowgraphs	489
Global Flowgraphs	489
Flow Graph Prefabs	489
External Files	489

Managing Flow Graphs	489
Saving Flow Graphs	490
Grouping Flow Graphs	491
Importing and Exporting Flow Graphs	491
Using Flow Graph Nodes	491
Node Input/Output Ports	492
Adding Entity Nodes	493
Adding Component Nodes	493
Managing Nodes	493
Creating Flow Graph Nodes	494
Output Ports	495
Input Ports	496
Trigger Ports	500
Update Event	500
Flow Graph Node Reference	500
Actor Nodes	502
AI Nodes	506
AISequence Nodes	525
Animations Nodes	532
Audio Nodes	541
Camera Nodes	545
ComponentEntity Nodes	547
CustomAction Nodes	557
Debug Nodes	560
Dialog Nodes	572
Dynamic Response Nodes	574
Engine Nodes	577
Entity Nodes	579
Environment Nodes	593
FeatureTest Nodes	598
Game Nodes	600
Helicopter Nodes	606
Image Nodes	608
Input Nodes	619
Interpolate Nodes	638
Intersection Tests Nodes	644
Iterator Nodes	645
JSON Nodes	648
Kinect Nodes	650
Logic Nodes	652
Material Nodes	662
MaterialFX Nodes	665
Math Nodes	666
Mission Nodes	685
Module Nodes	689
Movement Nodes	693
Physics Nodes	696
Prefab Nodes	702
ProceduralMaterial Nodes	703
Stereo Nodes	714
String Nodes	715
System Nodes	719
Time Nodes	720
Twitch Nodes	730
Vec3 Nodes	738
Vehicle Nodes	745
Video Nodes	756
Weapon Nodes	757

XML Nodes	760
Using Flow Graph Links	772
Using Flow Graph Tokens	773
Managing Flow Graph Modules	773
Module Node Ports	774
Debugging Flow Graph	775
Using Flow Graph Debugger	775
Using Console Variables	775
Placing Cached Shadows	776
Recommended Settings	776
Related Console Variables	776
Gems	778
Modular Gems System	779
Gem Assets	779
Gem Code	779
Gem JSON File	780
Gem List File	781
Lumberyard Gems	782
Boids Gem	782
Camera Framework Gem	789
Cloud Gem	789
Cloud Canvas Gem	795
GameEffect Gem	796
GameLift Gem	796
Gestures Gem	797
Input Management Framework Gem	807
Lightning Arc Gem	807
Metastream Gem	814
Multiplayer Gem	818
Physics Entities Gem	822
Process Life Management Gem	822
Rain Gem	823
Snow Gem	827
Substance Gem	831
Tornadoes Gem	831
UiBasics Gem	836
UiDemo Gem	836
User Login Default Gem	836
Woodland Asset Collection Gem	836
Levels and Environment	838
Creating a New Level	838
Creating Terrain	839
Using the Terrain Heightmap	840
Using Terrain Texture Layers	843
Creating Landforms and Topography	848
Creating Bodies of water	851
Copying and Moving Terrain Areas	857
Importing and Exporting Terrain Blocks	858
Importing Splat Maps	858
Adding Sky Effects	860
Creating a Dynamic Daytime Sky	861
Creating a Dynamic Night Sky	863
Creating Time of Day Sky Effects	864
Creating a Static Sky (SkyBox)	867
Adding Weather Effects	868
Adding Wind Effects	868
Adding Clouds	870
Working with Layers	872

Managing Level Layers	872
Assigning Objects to Layers	873
Streaming and Switching Layers	874
Adding Vegetation	875
Vegetation Best Practices	876
Vegetation Recommendations	876
Vegetation Texture Mapping	876
Adding Trees and Bushes	877
Adding Grass	877
Adding Vegetation Bending Effects	878
Vegetation Parameters	880
Vegetation Debugging	881
Mobile Support	882
Android Support	882
Prerequisites	882
Setting Up Your PC	883
Setting Up Your Mac	883
Configuring Your Game Project for Android	885
Building Game Assets for Android Games	888
Building Shaders for Android Games	890
Building Android Games	892
Android Debugging	893
Deploying Android Games	894
Running Android Games	896
Using Virtual File System with Android	898
Using a Samsung Device with Lumberyard	900
Using Lumberyard with Android Studio	900
iOS Support	905
Prerequisites	906
Setting Up Your Mac	906
Building Game Assets for iOS Games	907
Building Shaders for iOS Games	908
Building and Deploying iOS Games	909
iOS Debugging and Troubleshooting	912
Creating iOS Games	913
Preparing Lumberyard iOS Games for Distribution	914
Using Virtual File System with iOS	914
Design Considerations for Creating Mobile Games Using Lumberyard	916
Input	916
Game Logic	917
Application Lifecycle	918
Adding IP Addresses to Allow Access to the Asset Processor and Remote Console	919
OS X Support	920
Prerequisites	920
Setting Up Your Mac	920
Building Game Assets for OS X Games	921
Sharing Game Assets Between PCs and Macs	922
Building Shaders for OS X Games	922
Building and Deploying OS X Games	923
OS X Debugging and Troubleshooting	925
Creating OS X Games	926
Particle Effects System	928
Particles Best Practices	929
Using the Particle Editor	929
Using the Preview Window	931
Customizing the UI	931
Using the Gradient Editor	932
Working with Color Gradients	933

Using Particle Editor Shortcut Keys	933
Managing Particle Libraries	934
Adding Particle Libraries	935
Importing Particle Libraries	935
Exporting Particle Libraries	936
Using Particle Libraries	936
Creating Custom Attribute Panels	937
Particle Trails	938
Particle Trail Parameters	939
Particle Trail Visibility	939
GPU Particles	940
Attribute Comment	940
GPU Emitter Attribute	940
GPU Particles Attribute	941
GPU Lighting Attribute	943
GPU Size Attribute	944
GPU Rotation Attribute	945
GPU Movement Attribute	945
GPU Particle Parameter Modifiers	946
Particle Level Of Detail (LOD)	947
Level Of Detail Panel	947
LOD Level Panel	949
Managing Emitters	950
Creating Emitters	950
Duplicating Emitters	950
Creating Child Emitters	951
Editing Emitters	951
Organizing Emitters in a Library	951
Reverting Changes to Emitter Attributes	952
Advanced Particle Techniques	952
Attaching Particle Effects to Basic Geometry Entities	952
Attaching Particles to Breakable Objects	952
Attaching Particles to Character Animations	953
Generating Particles from Surface Properties	953
Particle Entity Parameters and Properties	954
Particle Attributes and Parameters Reference	956
Using the Curve Editor	956
Attribute Comment	957
Emitter Attribute	957
Particles Attribute	962
Lighting Attribute	967
Size Attribute	968
Particle Rotation Parameters	970
Movement Attribute	971
Collision Attribute	973
Visibility Attribute	975
Advanced Attribute	977
Configuration Attribute	978
Audio Attribute	979
Particle Debugging	980
Physics System	981
Physics Proxies	981
Geometry Guidelines and Best Practices	982
Debugging Physics Proxy Issues	982
Sounds and Physics	983
Debugging Physics	984
Project Configurator	985
Creating and Launching Game Projects	986

Enabling Gems	986
Using Lmbr.exe	987
Project Commands	987
Gem Commands	987
Troubleshooting the Project Configurator	988
Rendering and Graphics	990
Materials and Shaders	990
Shader Rendering System	991
Shader Reference	1002
Selecting Material Surface Type	1034
Setting Material Opacity	1034
Setting Material Lighting and Color Settings	1034
Material ID Mapping in Autodesk 3ds Max	1035
Working with Textures	1044
Working with Substances	1055
Parallax Mapping	1057
Using Vertex Colors	1059
Customizing Post-Processing Effects	1059
Lighting and Shadows	1062
Environment Lighting	1062
Environment Shadows	1067
Voxel-based Global Illumination (SVOGI)	1069
Integration Modes	1070
Voxel GI Parameters	1070
Debugging	1071
Current Limits	1072
Render Cameras and Effects	1072
Fog Systems	1072
Rendering Cameras	1084
Sample Projects and Levels	1088
Samples Project	1091
Getting Started Project	1091
Samples Projects	1091
Twitch Chat Basics	1103
Multiplayer Sample Project	1103
MultiplayerLobby	1103
MultiplayerGame	1104
GameLiftLobby	1105
Legacy Sample Project (GameSDK)	1105
Beach City Sample Project	1106
Woodland Asset Package	1107
FeatureTests Project	1108
FeatureTest Controls	1109
FeatureTest Levels	1109
Testing, Profiling, and Debugging	1112
Using AZ Test Scanner	1112
Creating Unit and Integration Test Builds	1112
Running Unit and Integration Test Builds	1113
Statoscope Profiler	1116
User Interface	1116
Logging Data	1118
Filtering Data	1119
Data Groups	1120
Creating Data Groups	1125
Guidelines and Best Practices	1126
Debugging Issues	1127
Using Console Debug Views	1127
Troubleshooting	1129

Viewing Error Log	1129
Error Message Reference	1129
Art Assets Errors	1129
Twitch ChatPlay System	1130
Setting up a Twitch ChatPlay Channel	1131
Listening for Twitch Keywords	1132
Using Flow Graph with Twitch ChatPlay	1133
Twitch ChatPlay Voting	1133
Twitch ChatPlay Console Variables	1133
Generating and Setting a Twitch Client ID	1134
Generate a Client ID	1134
Set the Client ID	1134
Troubleshooting Twitch ChatPlay	1135
Twitch JoinIn	1135
Twitch API	1136
UI System	1137
Using the UI Editor	1137
Working with UI Canvases	1138
Navigating the Viewport	1139
Changing the Canvas Size	1140
Previewing Canvases	1140
Configuring Canvas Properties	1143
Associating Canvases with UI Flow Graph Nodes	1144
Loading Canvases in the Flow Graph Editor	1145
Loading Canvases in Lua	1147
Placing UI Canvases in the 3D World	1151
UI Elements	1152
Configuring UI Anchors and Offsets	1153
Using and Creating UI Prefabs	1153
UI Components	1154
Adding or Deleting Components	1155
Transform2D – Managing UI Anchors and Offsets	1155
Visual Components	1158
Interactive Components	1160
Layout Components	1170
Other Components	1171
Implementing New Fonts	1172
Adding New Fonts	1173
Creating Font Families	1173
Configuring Font Rendering Quality	1175
Using the Animation Editor	1176
Recording Animation Data	1177
Playing Animation Sequences	1178
Editing Animation Data	1178
UI Flow Graph Nodes	1185
UIe Flow Graph Nodes	1185
UI Flow Graph Nodes	1241
Virtual Reality	1305
Configuring your Project for Virtual Reality	1305
Configuring Required Console Variables	1306
Optional Console Variables	1307
Setting Up Virtual Reality with Flow Graph	1307
VR:ControllerTracking	1307
VR:DeviceInfo	1308
VR:TransformInfo	1308
VR:Dynamics:Controllers	1309
VR:Dynamics:HMD	1309
VR:OpenVR:Playspace	1310

VR:RecenterPose	1310
VR:VREnabled	1310
VR:SetTrackingLevel	1311
VR:TransformInfo	1311
VR:VREnabled	1311
Setting Up a Basic Virtual Reality Flow Graph	1311
Setting Up a Custom Playspace with Flow Graph	1313
Previewing your Virtual Reality Project	1316
Debugging your Virtual Reality Project	1316
Waf Build System	1318
Waf File System	1318
Waf File List (*.waf_files)	1319
Waf Branch Spec (waf_branch_spec.py)	1320
Waf Projects File (project.json)	1321
Waf Spec Files (*.json)	1324
Waf Module Files (wscript)	1326
Waf Default Settings (default_settings.json)	1328
Waf User Settings (user_settings.options)	1328
Waf Commands and Options	1334
Waf Configuration	1334
Build Configuration	1335
Multiplayer Configuration	1338
Waf Supported Platforms and Compilers	1338
Waf Project Settings	1339
Platform and Configuration Targeting	1341
Features	1341
Waf Extensions	1342
Compiling with Incredibuild	1342
Compiling with QT	1343
Compiling with Visual Studio	1344
Using Waf	1344
Adding a Game Project	1345
Adding a Spec	1347
Adding a Build Module	1349
Adding User Settings to Waf	1357
Getter Function	1357
Validator Function	1359
Hint Function	1360
Adding Qt 5 Content to Waf	1360
MOC (Meta-Object Compiler) Files	1361
QRC (QT Resource Collection) files	1361
UI Files	1361
Qt Linguist (TS) files	1362
Using Uber Files	1362
Configuring Waf	1363
Debugging Waf	1363
Game Builds	1365
Compiling Game Code	1366
Creating Release Builds for PC	1366
Running a Build from Visual Studio	1368
Creating Minimal Release Builds	1368
Using Visual Studio	1369
Compiling Shaders for Release Builds	1369
Adding Custom Game Icons	1369
Universal Remote Console	1370
Issuing Commands	1371
Glossary	1372
Lumberyard Blog, Forums, and Feedback	1377

Legal	1378
Lumberyard Redistributables	1378
Alternate Web Services	1380

What is Lumberyard?

Amazon Lumberyard is a free, cross-platform, 3D game engine that allows you to create the highest-quality games, connect your games to the vast compute and storage of the AWS cloud, and engage fans on Twitch. By starting game projects with Lumberyard, you can spend more of your time creating great gameplay and building communities of fans, and less time on the undifferentiated heavy lifting of building a game engine and managing server infrastructure.

Lumberyard includes everything a professional game developer would expect, from a full-featured editor, to native code performance and stunning visuals, and hundreds of other ready-to-use features like performant networking, character and animation editors, particle editor, UI editor, audio tools, and more. Additionally, Lumberyard unlocks huge scale with AWS and Twitch so that you can more easily build live multiplayer and community-driven games.

Professional-Grade AAA Engine

You can use Lumberyard to build rich, engaging, world-class games with the highest ceiling of quality through its comprehensive and proven toolset and runtime performance that has been highly optimized over many years. Lumberyard includes support for:

Beautiful Worlds

The visuals technology of Lumberyard can beautifully bring to life any virtual environment. Built on technology that has created award-winning graphical fidelity and benchmark-setting graphical performance, Lumberyard is capable of producing near-photorealistic environments and stunning real-time effects. Your artists get a powerful toolbox to create world-class visuals, such as physically based shaders, dynamic global illumination, a particle effects editor, vegetation tools, real-time dynamic water caustics, volumetric fog, and filmic features such as color grading, motion blur, depth of field, and integrated HDR lens flares.

Compelling Characters

Your artists can use Lumberyard to create believable characters and high-fidelity performances. Lumberyard's character tool, Geppetto, combines animation, attachments, and physics simulations with blendshape, blendspace, and animation layering. Combined with Lumberyard's animation tool, Mannequin, animators can bring believable characters and creatures to life with features that include animation sequencing, transitions, game logic procedures, ragdoll physics, and more.

Robust Networking

Lumberyard introduces GridMate, a robust and flexible networking solution designed for efficient bandwidth usage and low-latency communications. You can easily synchronize objects over the network with GridMate's replica framework. GridMate's session management integrates with major online console services and lets you handle peer-to-peer and client server topologies with host migration.

Real-time Gameplay Editing

Real-time gameplay editing enables you to iterate on gameplay and immediately see your results, without waiting for builds or leaving the editor. Lumberyard's asynchronous asset processing system automatically converts and optimizes your game assets in real time, so that you can import game objects, fine tune behavior, and play the game you have created.

Modular gems

Lumberyard's Modular Gems system gives you a library of pre-built features that can be used to quickly start new projects or prototype ideas. Modular gems give you increased control over which technologies you want to include in your game project. You can create your own modular gems or use any of the gems included with Lumberyard, such as weather effects, a boids-based ambient creature system, lightning effects, a camera framework, and more.

Wwise LTX

Lumberyard includes a version of Audiokinetic's advanced, feature-rich sound engine. With minimal dependency on engineers, sound designers and composers can work independently to author rich soundscapes for your games.

and more...

Additional discipline-specific toolsets provide the opportunity to create unique, thrilling, and differentiated content. With terrain tools, weather effects, input systems, cover systems, perception handling, Lua support, drillers, pathfinding, goal-driven planning, and more, Lumberyard provides the tools to help achieve your vision.

Integrated with AWS

Lumberyard is deeply integrated with AWS so you can build live and multiplayer games with dramatically less cost, time, and technical risk. AWS integrations include:

Amazon GameLift

Using Amazon GameLift, a new AWS service for deploying, operating, and scaling session-based multiplayer games, you can quickly scale high performance game servers up and down to meet player demand, without any additional engineering effort.

Cloud Canvas

You can build live, online game features, such as a community news feed, daily gifts, or in-game notifications, in minutes using Lumberyard's Cloud Canvas tool. With Cloud Canvas' drag-and-drop visual scripting interface, you can build gameplay that connects to AWS services, such as Amazon DynamoDB, AWS Lambda, and Amazon S3.

AWS SDK for C++

The AWS SDK for C++ provides C++ APIs for numerous AWS services including Amazon S3, Amazon EC2, Amazon DynamoDB, and more, with support for all major native platforms. You can use the SDK to quickly integrate AWS components into your game. For more information, see [AWS SDK for C++](#).

Integrated with Twitch

Lumberyard is integrated with Twitch so that you can build games that engage with more than 1.7 million monthly broadcasters and more than 100 million monthly viewers on Twitch.

Twitch ChatPlay

The Twitch ChatPlay feature within Lumberyard helps you build gameplay that interacts in real time with Twitch viewers. For example, you can build a game where viewers can vote on game outcomes, gift power-ups to their favorite players, or change the level based on the number of viewers watching the player. Using the Lumberyard Flow Graph visual scripting tool, you can create chat channel commands for your game. For example, you can build a multiplayer game where viewers can vote to drop grenades to the broadcaster by typing #boom in the Twitch chat channel.

Twitch JoinIn

The Twitch JoinIn feature within Lumberyard lets you build multiplayer games that allow Twitch broadcasters to instantly invite fans to join them side-by-side in the game. Once invited, a fan can jump into the broadcaster's game with a single click in the Twitch chat channel, while others continue to watch.

Free, with Source

Lumberyard is free, including source code. You can deeply customize Lumberyard for your team and vision for your project today, and for future projects in years to come. There are no seat fees, subscription fees, or requirements to share revenue. Only pay for the AWS services you choose to use. For more information, see the [Lumberyard Licensing FAQ](#).

Topics

- [Lumberyard Systems \(p. 3\)](#)
- [Lumberyard Editors and Tools \(p. 4\)](#)
- [Lumberyard Asset File Types \(p. 5\)](#)

Lumberyard Systems

Lumberyard consists of the following major systems:

- [AI System \(p. 68\)](#)
- [Asset Pipeline \(p. 123\)](#)
- [Audio System \(p. 130\)](#)
- [Characters and Animation \(p. 151\)](#)
- [Cinematics System \(p. 279\)](#)
- [Component Entity System \(p. 320\)](#)
- [Object and Entity System \(p. 416\)](#)

- [Flow Graph System \(p. 487\)](#) (Visual scripting)
- [Levels and Environment \(p. 838\)](#)
- [Materials and Shaders \(p. 990\)](#)
- [Modular Gems System \(p. 779\)](#)
- [Particle Effects System \(p. 928\)](#)
- [Project Configurator \(p. 985\)](#)
- [Shader Rendering System \(p. 991\)](#)
- [Twitch ChatPlay System \(p. 1130\)](#)
- [UI System \(p. 1137\)](#)
- [Virtual Reality \(p. 1305\)](#)
- [Waf Build System \(p. 1318\)](#)

Lumberyard Editors and Tools

Lumberyard provides the following suite of applications, editors, and tools for game development.

Tool Name	Description
AI Debugger	Debugs AI agent behaviors and consists of the AI Debug Recorder and AI Debug Viewer
Asset Browser	Displays all game assets available for use
Asset Processor	Runs in the background when you launch Lumberyard Editor, monitoring input folders for changes in source files and automatically generating platform-specific game assets as they change
Audio Controls Editor	Manages audio translation layer (ATL) controls and events for the Audio system
Geppetto	Manages character animations, attachments, and physics simulations along with blendspaces and animation layering
Component Palette	Lists available components for the component entity system
Console	Runs editor commands and lists available console variables
Database View	Displays various object libraries such as entities, particles, and prefabs
Entity Inspector	Displays the ID and name for component entity system objects
Entity Outliner	Displays all component entities used for a level
FBX Importer	Imports single static meshes and materials from FBX
Flow Graph	Implements complex game logic using a visual scripting system
Layer Editor	Creates and manages layers for levels
Lens Flare Editor	Creates and manages camera lens flare effects
Sun Trajectory Tool	Creates and manages dynamic sky lighting effects
LOD Generator	Generates geometry and material level of detail (LOD)
Lumberyard Editor	Acts as the main workspace editor and game viewport; loads the Rollup Bar and console by default

Tool Name	Description
Lumberyard Setup Assistant	Ensures you have the necessary runtime software and SDKs installed to successfully run Lumberyard
Lumberyard Tools	Exports static and skinned geometry, skeletons, materials, and animation
Mannequin Editor	Manages the high-level character Mannequin system and includes the FragmentID Editor, Fragment Editor, Tag Definition Editor, Transition Editor, Sequence Previewer, Animation Database Editor, and Context Editor
Material Editor	Applies final material setup, texture mapping, and shader parameters
Measurement System Tool	Measures the length of segmented objects like roads, rivers, and paths
Missing Asset Resolver	Searches for assets that have moved and references their new locations
Modular Gems System	Provides a library of prebuilt features that you can use to quickly start new projects or prototype ideas
Particle Editor	Creates and simulates explosions, fire, sparks, and other visual effects
Project Configurator	Standalone application used to tell the Waf build system which gems to include in the game build
Resource Compiler	Compresses and processes source game asset files and creates package files
Rollup Bar	Accesses and places objects, vegetation, modified terrain, and modeling tools; includes display options, profile tools, and layer controls
Script Terminal	Runs various scripts in a terminal window
Smart Objects Editor	Creates and manages smart objects, which are used to interact with other objects according to complex rules
Substance Editor	Imports substance <code>.sbsar</code> files, edits material properties, and exports them as textures
Terrain Editor	Generates terrain and sculpts terrain elements in your level
Terrain Texture Layers	Creates and paints terrain texture layers in your level
Time of Day Editor	Creates and manages day-night cycles and other dynamic sky effects
Track View Editor	Creates and manages cinematic scenes and sequences; consists of the Track Editor and Curves Editor
UI Editor	Creates, manages, and simulates user interface elements for your game, such as menus and heads-up displays (HUD)
Universal Remote Console	Used to connect to a remote instance of Lumberyard running on mobile devices

Lumberyard Asset File Types

The following tables list the various asset data file types supported or used in Lumberyard.

In addition, Lumberyard supports the `.xml` file format that is used for various purposes as well as the following image file formats:

- `.tif`
- `.png`
- `.jpg`
- `.tga`
- `.bmp`
- `.pgm`
- `.raw`
- `.r16`

3D Art Asset File Types

The following file formats are used for static geometry.

File Type	Where Created	Description
* <code>.cgf</code> (Static Geometry File)	DCC tool	Contains static geometry data, such as grouped triangles, tangent spaces, vertex colors, physics data, and spherical harmonics data.
* <code>.chr</code> (Character Asset File)	DCC tool	The base character used for animations.
* <code>.cdf</code> (Character Definition File)	Lumberyard	Defines the base character and associated attachments. This file is created using Geppetto and contains a reference to the <code>.chr</code> file.
* <code>.skin</code> (Character Skinned Render Mesh)	DCC tool	Contains skinned character data, including the mesh, vertex weighting, vertex colors, and morph targets.
* <code>.fbx</code> (Filmbox File)	DCC Tool	Contains mesh, material, camera, and animation data. Provides interoperability between DCC tools.
* <code>.scenestettings</code> (Scene Settings File)	Lumberyard	Contains configuration and rules settings from an <code>.fbx</code> file.
* <code>.abc</code> (Alembic Cache File)	DCC tool	Contains non-procedural, application-independent set of baked geometric data such as baked meshes and their materials.
* <code>.cax</code> (CAD/CAE Exchange File)	Lumberyard	Contains compressed game assets read from the <code>.abc</code> file and streamed in-game on demand from disk.
* <code>.trb</code> (Terrain Block File)	Lumberyard	Contains terrain data and associated level objects such as water and vegetation.

Material and Texture File Types

File Type	Where Created	Description
* <code>.mtl</code> (Material File)	DCC Tool	

File Type	Where Created	Description
*.dds (DirectDraw Surface)	DCC tool	Contains compressed source texture files.
*.sbsar (Substance Files)	Allegorithmic Substance Designer	Contains procedural materials.

Animation File Types

For more information on these file types, see [Character Animation Files \(p. 226\)](#).

File Type	Where Created	Description
*.adb (Animation Database File)	Lumberyard	Used by the Mannequin system to store fragments and transitions. This is typically referred to from the character Lua file and other systems such the hit death reaction system.
*.i_caf (Intermediate Character Animation File)	DCC tool	Contains the animated bone data for one or more characters in uncompressed format.
*.animsettings (Animation Settings File)	Lumberyard	Contains per-animation compression settings. This is a sidecar file that is stored next to the .i_caf file and describes how it should be compiled by the Asset Pipeline.
*.caf (Character Animation File)	Lumberyard	The compressed version of the intermediate .i_caf file. Contains on-demand asset data that is streamed in and out of the game as needed at runtime.
*.chrparams (Character Parameters File)	Lumberyard	Contains skeletal characters. Has the same name as the .chr file to which it refers to.
*.dba (Animation Database)	Lumberyard	Contains multiple compressed .caf animation files that are streamed in and out of the game together. Created by the Resource Compiler and defined in the .chrparams file.
*.animevents (Animation Events Database)	Lumberyard	Stores a list of assets with timed event markups. Geppetto is used to create this file, which gets mapped to the .chrparams file.
*.bspace (Blend Space File)	Lumberyard	Define how multiple animation assets are blended together. Blend spaces are parameterized at runtime with movement parameters such as movement speed, movement direction, turning angle, or slope.
*.comb (Blend Space Combination File)	Lumberyard	Combines multiple blend spaces into one, usually of a higher order, and represents a multidimensional blend space.
*.grp (Group Files)	DCC Tool	Exported animation sequences used for Track View sequences.

Audio Asset File Types

File Type	Where Created	Description
* .bnk (Soundbank File)	Audiokinetic Wwise	Contains compiled sound data and metadata.
* .wem (Encoded Media File)	Audiokinetic Wwise	Compiled streamable audio file.

Setting Up Lumberyard

Lumberyard supports the following platforms: PC, Xbox One, PlayStation 4, Android, iOS, and OS X. In order to develop games for the Xbox One or PlayStation 4, you must pass Microsoft and Sony's screening process, respectively. For information about console support, see [Developing Games for Xbox One](#) and [Become a Registered Developer](#) [for PlayStation]. For information about developing for mobile devices, see [Mobile Support \(p. 882\)](#). For information about developing for OS X, see [OS X Support \(p. 920\)](#).

Topics

- [System Requirements \(p. 9\)](#)
- [Downloading Lumberyard \(p. 10\)](#)
- [Upgrading Lumberyard \(p. 11\)](#)
- [Files to Exclude When Upgrading Lumberyard \(p. 13\)](#)
- [Using Lumberyard Setup Assistant to Set Up Your Development Environment \(p. 14\)](#)
- [Enabling a Firewall \(p. 20\)](#)

System Requirements

Lumberyard requires the following hardware and software:

- Windows 7 64-bit
- 3GHz minimum quad-core processor
- 8 GB RAM minimum
- 2 GB minimum DX11 or later compatible video card
- Nvidia driver version 368.81 or AMD driver version 16.15.2211 graphics card
- 60 GB minimum of free disk space
- Visual Studio 2013 Update 4 or later (required to compile Lumberyard Editor and tools)
- Visual C++ Redistributable Packages for Visual Studio 2013

If you do not already have Visual C++ Redistributable Packages for Visual Studio 2013 installed, do one of the following:

- After you have installed Lumberyard, run the installer from the following location: `\dev\Bin64\Redistributables\Visual Studio 2013`
- Download and run the installer directly from Microsoft: [Visual C++ Redistributable Packages for Visual Studio 2013](#)

For information about installing third-party software and SDKs, see [Using Lumberyard Setup Assistant to Set Up Your Development Environment \(p. 14\)](#).

Downloading Lumberyard

This topic includes information about downloading Lumberyard using the Lumberyard Installer, and links to more information about installing required third-party software. Upon completing the installation process, you will be able to use Lumberyard, Lumberyard Editor, and other engine tools.

Note

Be sure you have the hardware and software required to use Lumberyard. For information, see [System Requirements \(p. 9\)](#).

The Lumberyard directory includes the following folders and files:

- dev
 - `_WAF_` – Waf build system files
 - `Bin64` – Binaries directory and configuration files for the resource compiler
 - `Code` – Source files directory and solution files
 - `Editor` – Editor assets
 - `Engine` – Engine assets
 - `Gems` – Optional systems and assets
 - `MultiplayerProject` – Multiplayer sample project
 - `ProjectTemplates` – Configuration files, libraries, and scripts for the empty template
 - `SamplesProject` – Sample project
 - `Tools` – Third-party tools and plugins
- `3rdParty`
 - Third-party software required to use or compile Lumberyard
 - `Wwise LTX` – Software for authoring game audio
- `docs`
 - Release Notes
 - *Lumberyard Getting Started Guide*

Using the Lumberyard Installer to Download Lumberyard

The Lumberyard Installer provides a simpler way for you to download and install Lumberyard. After you specify the install location, the Lumberyard Installer extracts the Lumberyard files and adds shortcuts for the Lumberyard Setup Assistant, Project Configurator, and Lumberyard Editor on your desktop and in the Start menu. The installer also allows you to resume an interrupted download.

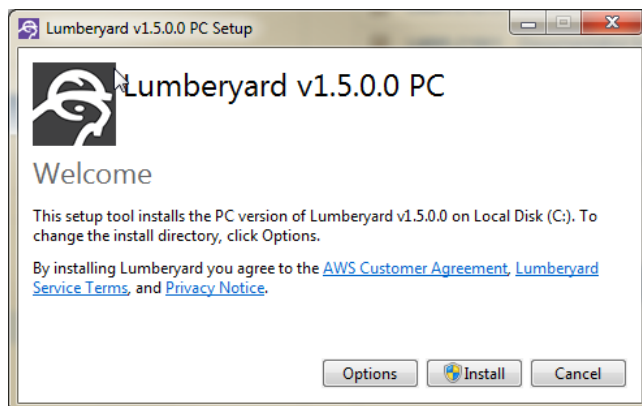
If you have an existing Lumberyard project, we recommend installing the latest version of Lumberyard in a new directory. For information, see [Upgrading Lumberyard \(p. 11\)](#).

Note

If you encounter errors during file extraction about `AssetProcessor.exe`, `AssetProcessor_temp.exe`, or `CrySystem.dll`, check whether your antivirus software is placing these files in quarantine and, if possible, grant exceptions for the affected files.

To download Lumberyard using the installer

1. On the [Lumberyard Downloads](#) page, under **Amazon Lumberyard**, click **Download Lumberyard**.
2. Run the Lumberyard Installer executable.
3. On the **Welcome** page, click **Install** to install to the default location. You can change the install location by clicking **Options**.



4. Follow the instructions onscreen to complete your installation.
5. On the **Installation Successfully Completed** page, click **Launch** to install required third-party software and SDKs using Lumberyard Setup Assistant. For information, see [Running Lumberyard Setup Assistant \(p. 15\)](#).

Upgrading Lumberyard

If you have an existing version of Lumberyard installed on your computer, you have several options for upgrading:

- Upgrade Lumberyard with an existing version in source control
- Upgrade Lumberyard without an existing version in source control
- Upgrade Lumberyard without source control

We recommend using source control, which allows relationships to be created between the installed versions of Lumberyard and the changes you make to your projects, among other benefits like revision history.

Note

When choosing a source control solution, keep in mind that Lumberyard provides plugins and tools for working with Perforce.

To set up Lumberyard in source control

1. Download and install Lumberyard. For information, see [Downloading Lumberyard \(p. 10\)](#).
2. Check into source control a pristine, unmodified version of Lumberyard. For information about file types to ignore, see [Files to Exclude When Upgrading Lumberyard \(p. 13\)](#). For information about the Lumberyard directory structure, see [Downloading Lumberyard \(p. 10\)](#).
3. In source control, create a new branch off the pristine Lumberyard branch to use for development.

4. Make changes to the new development branch only.

Upgrading Lumberyard with an Existing Version in Source Control

Before you begin upgrading, check into source control the previous pristine version of Lumberyard.

To upgrade Lumberyard with an existing version in source control

1. In Windows Explorer, locate the directory where you installed the previous pristine version of Lumberyard. Delete the contents of this directory to remove the files from source control.
2. Download and install the new version of Lumberyard to the empty directory. Ensure the directory structure is identical to the previous version.
3. Using source control, reconcile the files in the directory with the files in the pristine Lumberyard branch. For example, if you use Perforce, click **Actions, Reconcile Offline Work**.
4. Build and test the reconciled version locally to ensure it works.
5. Submit the reconciled version to the pristine Lumberyard branch as the new version of Lumberyard.
6. Integrate the updated, pristine Lumberyard branch into your development branch.

Upgrading Lumberyard without an Existing Version in Source Control

Follow these steps to prepare your source control to upgrade Lumberyard.

To upgrade Lumberyard without an existing version in source control

1. Check into source control the pristine version of Lumberyard that you used to create your game project.
2. Create a new branch off the pristine Lumberyard branch to use for development.
3. In Windows Explorer, locate the directory for the new development branch and delete the contents.
4. Copy the files from your existing game project to the empty directory.
5. Using source control, reconcile the files in the development branch directory with the files in source control. Accept your changes.
6. Follow the steps in [Upgrading Lumberyard with an Existing Version in Source Control \(p. 12\)](#).

Upgrading Lumberyard without Source Control

You can upgrade Lumberyard without using source control; however, we do not recommend this method.

To upgrade Lumberyard without source control

1. Download and install the latest version of Lumberyard to a location that will not overwrite any previous versions. For information, see [Downloading Lumberyard \(p. 10\)](#).
2. Use Lumberyard Setup Assistant to install the third-party software and SDKs required to run Lumberyard. For information, see [Running Lumberyard Setup Assistant \(p. 15\)](#).
3. Configure and compile the Samples Project to test your build environment.

Upgrading Your Game Projects

Once you have upgraded Lumberyard, you can upgrade each of your game projects.

To upgrade your game project

1. Copy your project's code (located in the `\dev\Code\[project name]` directory) and game folder (located in the `\dev\[project name]` directory) to the new Lumberyard directory.
2. Create a `project.json` file for your project with the following:

```
{
  "project_name": "[project name]",
  "product_name": "[project name]",
  "executable_name": "[project name]Launcher",
  "code_folder": "Code/[project name]",
  "modules" : ["[project name]"]
}
```

Replace all instances of `[project name]` with your project's name.

For example, if your project was called `MyProject`, the `project.json` file would include the following:

```
{
  "project_name": "MyProject",
  "product_name": "MyProject",
  "executable_name": "MyProjectLauncher",
  "code_folder": "Code/MyProject",
  "modules" : ["MyProject"]
}
```

3. Save the `project.json` file in the `\dev\[project name]` directory.
4. Run the Project Configurator (located in the `\dev\Bin64` directory) and set your game project as the default project. Close the Project Configurator when done.
5. Edit the `wscript` file (located in the `\dev\code\[project name]\Game` directory) to ensure the includes under `#Common` appear as follows:

```
#####
# Common
#####
    includes      = [ '.' ,
                    bld.Path('Code/CryEngine/CryCommon'),
                    bld.Path('Code/CryEngine/CryAction') ],
```

6. In a command line window, locate the new `dev` folder and run the following: `lmbr_waf configure build_win_x64_profile -p all`

Files to Exclude When Upgrading Lumberyard

When adding Lumberyard to source control, there are various files that you should exclude because they are generated, temporary, or developer-specific.

File types and folders in the entire repository to exclude

- *.ilk
- *.suo
- *.user
- *.o
- *.temp
- *.bootstrap.digests
- *.log
- *.exp
- *.vssettings
- *.exportlog
- *.mayaSwatches
- *.ma.swatches
- *.dds
- *.bak
- *.bak2
- *.options
- *.pyc
- *.db
- Solutions
- BinTemp
- Cache

File types and folders in the `\dev\Code` directory to exclude

- SDKs

File types and folders in each game folder (SamplesProject, MultiplayerProject, etc.) to exclude

- Compiled assets
 - *.dds
 - *.caf
 - *.\$animsettings
- Editor backup files – *.bak*
- Pak files that are exported from level files in the editor – *.pak

Using Lumberyard Setup Assistant to Set Up Your Development Environment

Use the Lumberyard Setup Assistant application to validate that you have installed the third-party software required to run Lumberyard.

Lumberyard Setup Assistant offers the following benefits:

- Ensures you have the required runtime software installed
- Ensures you have the required SDKs located in the source tree
- Provides plugins for certain programs detected
- Validates registry settings, paths, and libraries

You should run this application periodically and after you make any changes to your environment, to validate and repair settings and paths. You can also customize the application with a configuration file to easily integrate your specific directory structure.

Prerequisites

Lumberyard Setup Assistant is supported on the Windows operating system.

To use Lumberyard Setup Assistant, you need Visual Studio 2013 runtime. If you do not already have Visual Studio 2013 runtime installed, do one of the following:

- Install the runtime from the following location:

```
\3rdParty\Redistributables\VisualStudio2013
```

- Download and install the runtime directly from [Microsoft](#)

Topics

- [Running Lumberyard Setup Assistant \(p. 15\)](#)
- [Using Lumberyard Setup Assistant Batch \(p. 16\)](#)
- [Customizing Lumberyard Setup Assistant \(p. 18\)](#)

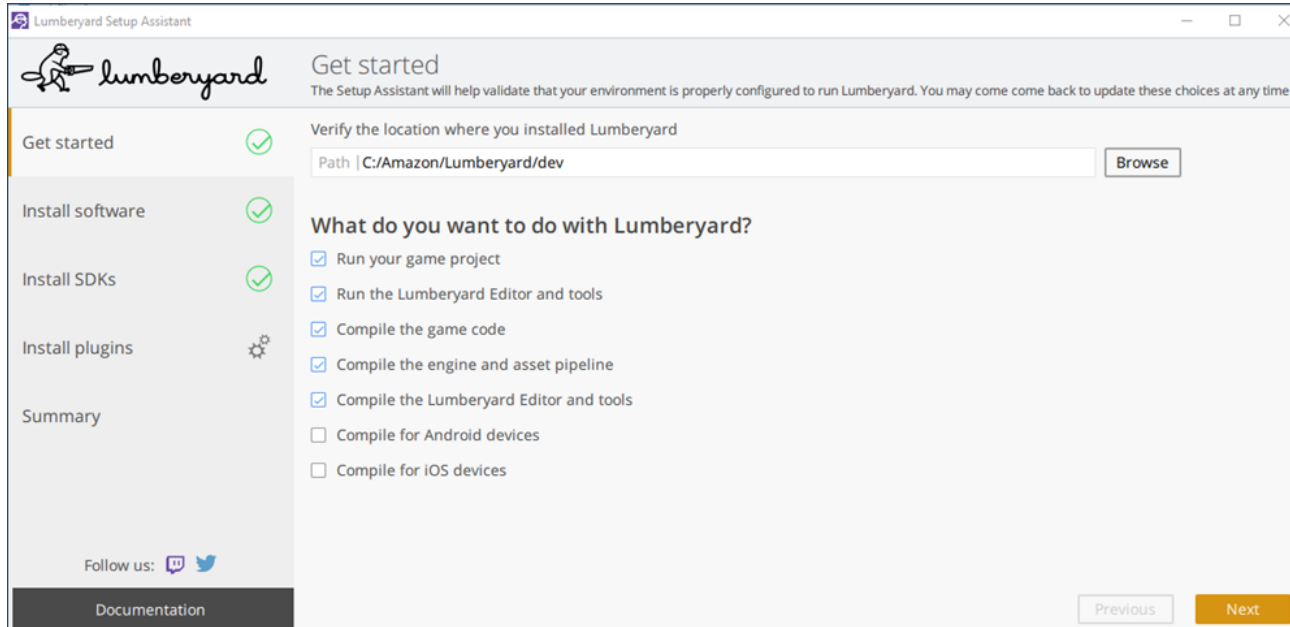
Running Lumberyard Setup Assistant

Before you run Lumberyard Setup Assistant, verify that `3rdParty.txt` appears in the `\3rdParty` directory and that `engineeroot.txt` appears in the `\dev` directory. These files are required for Lumberyard Setup Assistant to properly detect third-party software and SDKs.

To use Lumberyard Setup Assistant

1. Open the directory where you extracted Lumberyard. Run `SetupAssistant.bat`.
2. Verify that the engine root path is correct.
3. On the **Get started** page, select what you want to do:
 - **Run your game project**
 - **Run the Lumberyard Editor and tools** – Use Lumberyard Editor to create a game.
 - **Compile the game code*** – Compile the game code to include any changes you have made.
 - **Compile the engine and asset pipeline*** – Compile the engine code and asset pipeline to include any changes you have made.
 - **Compile the Lumberyard Editor and tools*** – Compile Lumberyard tools to include any changes you have made.
 - **Compile for Android devices***
 - **Compile for IOS devices***

*If you select any of the starred options, you may later see new dependencies in the **Install software** and **Required SDKs** pages. If so, follow the instructions to obtain each software and third-party SDK that you do not yet have installed.



4. Click **Next**.
5. Follow the instructions on each page.
6. When you have all the required software and SDKs installed for your implementation, click **Configure project** or **Launch Lumberyard**. For information, see [Project Configurator \(p. 985\)](#) or [Using Lumberyard Editor \(p. 33\)](#).
7. Log in to your existing Amazon account or create a new account to access the editor.

Using Lumberyard Setup Assistant Batch

The command line version of Lumberyard Setup Assistant is useful for server and build administrators and developers who would like to create a batch file to run the same configuration on multiple machines.

The command line version of Lumberyard Setup Assistant is provided in the `\dev\Bin64` directory as an executable file called `SetupAssistantBatch.exe`.

To use Lumberyard Setup Assistant Batch

1. Open a command prompt.
2. Change the directory to where you extracted Lumberyard.

Example: `cd D:\lumberyard-build\dev\Bin64`

3. Run the `SetupAssistantBatch.exe`.

Example: `D:\lumberyard-build\dev\Bin64\SetupAssistantBatch.exe`

4. Modify as needed. See the commands list below.

Commands

Command	Description
<code>--help</code>	Lists all commands and descriptions
<code>--3rdpartyath</code>	Sets the third-party directory to the specified parameter
<code>--sdkpath</code>	Sets the location of the Lumberyard SDK to the specified parameter Note This command expects a root where <code>Lumberyardroot.txt</code> is located.
<code>--disablecapability</code>	Disables the specified tasks (capabilities): <ul style="list-style-type: none">• Run your game project• Run the Lumberyard Editor and tools• Compile the game code• Compile the engine and asset pipeline• Compile the Lumberyard Editor and tools• Compile for Android devices• Compile for iOS devices Note Tasks are disabled by default.
<code>--enablecapability</code>	Enables the specified tasks: <ul style="list-style-type: none">• Run your game project• Run the Lumberyard Editor and tools• Compile the game code• Compile the engine and asset pipeline• Compile the Lumberyard Editor and tools• Compile for Android devices• Compile for iOS devices
<code>--all</code>	Enables all tasks
<code>--none</code>	Disables all tasks
<code>--no-modify-environment</code>	Prevents Lumberyard Setup Assistant from changing your environment variables

Examples

The following example sets the paths, clear all selected tasks, and set the selected task as "Run game":

```
setupassistantbatch.exe --3rdpartyath "d:\myLumberyard\3rdParty" --sdkpath  
"d:\myLumberyard\dev" --none --enablecapability runtime
```

If the command runs smoothly, the exit code for this program is 0.

The following example disables all tasks (capabilities) and enable only the compilation tasks. This is common for hosting a build server:

```
setupassistantbatch.exe --none --enablecapability compilegame --  
enablecapability compileLumberyard --enablecapability compilesandbox
```

Customizing Lumberyard Setup Assistant

The `\dev\Bin64` directory includes an external configuration file called `SetupAssistantConfig.json`. You can use the JSON file to customize Lumberyard Setup Assistant for your project. The settings in this file are prioritized above internal default settings.

Refer to the `SetupAssistantConfig.json` example file for example configuration data. You can copy and paste this information into your JSON file.

After you finish making your changes to the JSON file, run the `SetupAssistantBatch.exe` in a command line. This helps validate your changes for any syntax errors, for example a missing comma.

Enabling and Disabling Features

Based on your project requirements, you can enable or disable certain software and SDKs. The `SetupAssistantConfig.ini` file includes a list of commented code. Uncomment the lines to disable a specific feature.

```
;just uncomment the SDKs you want to disable  
;By default every SDK, software, and third-party plugin is enabled  
[DisabledSDKS]  
;boost="disabled"  
;python="disabled"  
;maya2013="disabled"  
;maya2014="disabled"  
;maya2015="disabled"  
;max2015="disabled"  
;photoshop="disabled"  
;mysql="disabled"
```

Adding New Third-Party SDKs

In addition to enabling or disabling certain software and SDKs, you can edit the `SetupAssistantConfig.json` file to add new, third-party SDKs to your project configuration.

When you add third-party SDKs to the `SetupAssistantConfig.json` file, which is loaded after the internal configuration file, the JSON file removes and replaces entries in the internal configuration. This allows you to customize your project configuration without having to recompile.

To add new tasks (capabilities)

- In the `SetupAssistantConfig.json` file, add the task(s) to the `Capabilities` section. Update the SDKs to include the appropriate tags.

To remove existing SDKs

- In the `SetupAssistantConfig.json` file, create a `remove` entry with the same identifier.

Note

When you specify the destination of your code directory, you can use `$CODEFOLDERNAME$` or specify the actual name. The code directory is the location where SDKs are expected and is relative to the SDK root. For example, you can change `CodeFolderName` to `myGame/A/b/c`.

SDK Fields

You may need to provide information for the following SDK fields.

identifier

Identifier that is not localized and can be used later to refer to the SDK. Must be one word and use only lowercase letters.

remove

Eliminates an existing entry if set to true. The `remove` and `identifier` fields are required to remove an entry.

name

Name of SDK; internal SDKs use `identifierName`, which is localized. Custom SDKs can use any name without any language restrictions.

description

Brief description of SDK; internal SDKs use `identifierDescriptionSummary`, which is localized. Custom SDKs can use any description. UTF-8 is supported.

detailedInstructions

(Optional) Detailed instructions to obtain the SDK.

tags

Tags to which the SDK applies. For example, if you need the SDK to run the game, you would add the `rungame` tag.

symlinks

List of symlink dictionaries for all junctions (symbolic links) to establish between the `3rdParty` directory and the code base. Each symlink uses the following form:

- `source` – Source directory, relative to the `3rdParty` directory
- `destination` – Destination directory, relative to the SDK root
- `exampleFile` – File that should be located in both the source and destination folders, to validate the link is established

Configuring Advanced Settings

The `SetupAssistantConfig.json` file has the following configuration settings in the root element (dictionary):

CodeFolderName

Location of the code directory, relative to `Lumberyardroot.txt`. You can specify relative paths such as `..` and `../..` (use forward slash marks) or relative paths with multiple components such as `code/mycode/stuff`.

ToolsFolderName

Location of the tools directory, relative to the `Lumberyardroot.txt` file. The default directory is `Tools`, but you can specify relative folders such as `../tools`.

RememberLumberyardRootFolder

Saves the Lumberyard root that the user browsed between sessions if set to true. Autodetects the Lumberyard root based on the executable location if set to false. The default value is false.

Remember3rdPartyFolder

Saves the third-party directory that the user browsed between sessions if set to true. Autodetects the third-party directory based on the executable location if set to false. The default value is false.

Customizing the Maya Environment

The `\Tools\Maya\Plugins` directory includes the Lumberyard Maya plugin, and the `\Tools\Maya\script` directory includes the MEL and Python scripts. To enable the Maya plugin functionality, Lumberyard Setup Assistant modifies your `Maya.ENV` to add the required variables to your Maya configuration.

If you use your own Maya tools in addition to the exporter and pipeline tools that Lumberyard provides, you can use the `SetupAssistantConfig.json` file to add your project-specific paths to the Maya ENV. Update the Maya paths in the `MayaEnvironments` tag in the JSON file.

Refer to the `SetupAssistantConfig.json` example file for example configuration data.

In the following example, `$TOOLS_FOLDER$` is a macro that is substituted with the appropriate tools directory; however, you can also use relative paths, relative to the game project's root directory that includes `Lumberyardroot.txt`:

```
"MayaEnvironments" :
[
  {
    "comment" : "an example entry showing how you can add a path to
MAYA_PLUG_IN_PATH in maya.env",
    "identifier" : "MAYA_PLUG_IN_PATH",
    "paths" : ["$TOOLS_FOLDER$/maya/plugins"]
  },
  {
    "comment" : "an example entry showing how you can add paths to
MAYA_SCRIPT_PATH in maya.env",
    "identifier" : "MAYA_SCRIPT_PATH",
    "paths" : ["%DHTECH_SCRIPT_PATH%\\%DHTECH_GAME_PATH%", "%DHTECH_SCRIPT_PATH
%\\animation"]
  },
],
```

Updating the Code or Tools Location

If your project requires moving the Lumberyard code or tools directory so that it's no longer located in a subfolder called `Code` or `Tools` relative to the Lumberyard root, you can edit the `SetupAssistantConfig.json` file to update the location of the directory. Ensure the updated directory includes the `Lumberyardroot.txt` file.

Enabling a Firewall

You can help protect your environment by enabling the firewall settings on all computers running the Asset Processor or Lumberyard Editor to do the following:

- Exclude external connections to ports 4600, 9432, 9433, and 45643 from untrusted IP addresses.
- Exclude connections from every address except 127.0.0.1.
- If you have multiple computers that work together (e.g. a PC and a Mac), you must allow connections to ports 4600, 9432, 9433, and 45643 from the IP addresses for these computers, but exclude all other connections.

Refer to the documentation for your operating system for how to manage your firewall settings.

Migrating Lumberyard Projects

If you are upgrading your projects to a newer version of Lumberyard, use the following guides to ensure the proper migration of earlier Lumberyard projects and components.

Lumberyard 1.6

If you are upgrading your projects to Lumberyard 1.6, use the following instructions to migrate GridMate service sessions and the framework for your tests.

Migrating GridMate Service Sessions

In Lumberyard 1.6, the way that GridMate handles session services has been refactored to enable multiple session services to co-exist. Previously only a single session service could be active and all requests were made through a generalized interface. The generalized interface has been removed and now E buses must be used to communicate with each session service. The following changes may require you to migrate built-in services to the new methods or to update any custom services that you have created.

RegisterService

`RegisterService` now uses `GridMateServiceId` to associate with the given service. The ID is used to unregister the service and should be unique to each instance of the service.

The following changes are required:

- Any calls to `RegisterService` must now pass along this ID.
- Any calls to the templated function `StartGridMateService` will work for the built-in session services (`LANSessionService`, `XBoneSessionService`, `PSNSessionService`, and `GameliftSessionService`).
- Any custom session services that are registered through the function must now implement the static function `GetGridMateServiceId` inside the class definition.
- The macro `GRIDMATE_SERVICE_ID` intakes the service name and creates the appropriate function. For example, `GRIDMATE_SERVICE_ID(MyCustomSessionService)`.

UnregisterService

`UnregisterService` now uses `GridMateServiceId` instead of `GridMateService*`.

The following changes are required:

- A new template function called `StopGridMateService` helps to standardize the helpers workflow. This helper will work for the built-in session services (`LANSessionService`, `XBoneSessionService`, `PSNSessionService`, and `GameliftSessionService`).
- Any custom session services that want to use this method must implement the static function `GetGridMateServiceId` inside the class definition.
- The macro `GRIDMATE_SERVICE_ID` intakes the service name and creates the appropriate function. For example, `GRIDMATE_SERVICE_ID(MyCustomSessionService)`.

HostSession, JoinSession, and StartGridSearch

`HostSession`, `JoinSession` (all varieties), and `StartGridSearch` have been removed from the `IGridMate` class.

The following changes are required:

- The removed methods no longer make sense when deciding which session service to use. Multiple session services offer the ability to accept different search parameters and implement a different subset of the join requests.
- Each built-in session service (`LANSessionService`, `XBoneSessionService`, `PSNSessionService`, and `GameliftSessionService`) now implements an EBus that exposes the specific methods for the session service. The EBus is identified by the `IGridMate*` instance to which the service is registered. Any calls to the `IGridMate*` methods must be replaced by service-specific EBus calls.

Host, Connect, and ListServers Nodes

The general purpose flow graph nodes for `Host`, `Connect`, and `ListServers` have been removed because the general purpose interface no longer exists.

The following changes are required:

- Flow graph nodes were created for each of the built-in session services (`LANSessionService`, `XBoneSessionService`, `PSNSessionService`, and `GameliftSessionService`). These service-specific nodes must be used in order to create a unified flow. For an example of how these nodes work in a multi-platform game, see the `MultiplayerLobby` level in the `Multiplayer Project`.

Migrating from CryUnitTest to AzTest

In Lumberyard 1.6, the `CryUnitTest` framework for writing tests is no longer available. The `AzTest` framework, which is built on top of `GoogleTest` and `GoogleMock`, replaces `CryUnitTest`. If you have tests written in `CryUnitTest`, follow these steps to use the `AzTest` framework for your tests.

To migrate from `CryUnitTest` to `AzTest`, you must:

- A. Modify tests to use `GoogleTest` macros
- B. Move tests into test build files
- C. Build and run tests

A. Modify Tests to Use GoogleTest Macros

Simply convert the `CryUnitTest` tests to `GoogleTest` tests by replacing the following macros. You must also replace `CryUnitTest.h` with `AzTest/AzTest.h` in your `.cpp` files to get the new macros.

CryUnitTest	GoogleTest
<code>CRY_UNIT_TEST_SUITE(SuiteName)</code>	No replacement. You can safely remove these from your code.
<code>CRY_UNIT_TEST(TestName)</code>	<code>TEST(SuiteName, TestName)</code>
<code>CRY_UNIT_TEST_FIXTURE(FixtureName)</code>	<code>class FixtureName : public ::testing::Test</code>
<code>CRY_UNIT_TEST_WITH_FIXTURE(TestName, FixtureName)</code>	<code>TEST_F(FixtureName, TestName)</code>
<code>CRY_UNIT_TEST_ASSERT(Condition)</code>	<code>ASSERT_TRUE(condition)</code>
<code>CRY_UNIT_TEST_CHECK_CLOSE(ValueA, ValueB, Epsilon)</code>	<code>ASSERT_NEAR(ValueA, ValueB, Epsilon)</code>
<code>CRY_UNIT_TEST_CHECK_EQUAL(ValueA, ValueB)</code>	<code>ASSERT_EQ(ValueA, ValueB)</code>
<code>CRY_UNIT_TEST_CHECK_DIFFERENT(ValueA, ValueB)</code>	<code>ASSERT_NE(ValueA, ValueB)</code>

B. Move Tests into Test Build Files

AzTest and GoogleTest are not included in normal builds, they are only included in test builds. Tests must be configured to only build in test builds. For more information about creating test builds and running tests, see [Using AZ Test Scanner \(p. 1112\)](#).

If your CryUnitTest tests are interspersed with regular engine code, you must move those tests into a separate `.cpp` file. You can then add the `.cpp` files to a test-specific `.waf_files` file. All modules and gems that are shipped with Lumberyard have a `*_test.waf_files` file. You can add new test files to these `*_test.waf_files` files. If your module or gem does not have a `*_test.waf_files` file, you can create one and reference it in the module's `wscript` file.

C. Build and Run Tests

After you complete these migration steps, you can build and run your new tests. For more information, see [Using AZ Test Scanner \(p. 1112\)](#).

If you encounter issues where your tests no longer work properly, your tests may be relying on in-engine code to pass. If this occurs, you must specify that your tests are integration tests by using the `INTEG_TEST` macro instead of `TEST`.

Lumberyard 1.5

If you are upgrading your projects to Lumberyard 1.5, use the following instructions to migrate your projects and convert your gems.

Migrating Your Project

Lumberyard 1.5 introduces application descriptor files, which list all modules used by a project. Each project requires two application descriptor files in its asset directory:

- `dev/<project_asset_directory>/Config/Game.xml`

- dev/`<project_asset_directory>/Config/Editor.xml`

Create these files by running the `Bin64\lmbd.exe projects populate-appdescriptors` command from the command line.

If you change gems using the Project Configurator, Lumberyard automatically updates the application descriptor files. If you manually edit a project's `gems.json` file, however, you must update these files by running the `Bin64\lmbd.exe projects populate-appdescriptors` command from the command line.

Migrating Your Gems

Beginning in Lumberyard version 1.5, gems with code should be built as [AZ modules](#). Gems built as AZ modules are better integrated with the Lumberyard's new [component entity system](#) (p. 320). As of Lumberyard 1.5, all gems that ship with Lumberyard have been migrated to be AZ modules.

Legacy gems built with Lumberyard 1.4 or earlier are still supported, but to avoid issues, we strongly recommend that you migrate them. If your custom gems make use of the component entity system, you should migrate your gems immediately.

To migrate a gem, you modify the initialization code and change the gem's public API to use [event buses](#). To accomplish this, you must:

- A. Rename your gem files
- B. Modify your gem code
- C. Edit your `gem.json` file
- D. Migrate your config files

Perform these procedures for each of your Lumberyard pre-1.5 custom gems.

A. Rename Your Gem.h File

In this step, you rename your `IGem.h` file.

To rename your gem.h file

1. Rename `Include\<GemName>\I<GemName>Gem.h` to `Include\<GemName>\<GemName>Bus.h` (Remove the `I` character and `Gem`, and add `Bus`).
2. Update your `<GemName>.waf_files` to account for the file that you renamed in the first step.

B. Modify Your Gem Code

To edit your gem code

1. Update the include statements that refer to the file that you renamed in the first procedure.
2. Make the following changes to `<GemName>Bus.h`.
 - a. Change the class name from `I<GemName>Gem` to `<GemName>Bus`.
 - b. Change the base class from `IGem` to `AZ::EBusTraits`.
 - c. Remove the `CRYINTERFACE_DECLARE` macro.
 - d. Add the following to the top of the class to make it a single handler bus:

```
public:  
////////////////////////////////////
```

```
// EBusTraits
static const AZ::EBusHandlerPolicy HandlerPolicy =
    AZ::EBusHandlerPolicy::Single;
static const AZ::EBusAddressPolicy AddressPolicy =
    AZ::EBusAddressPolicy::Single;
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
```

- e. Add the following after the class definition:

```
using <GemName>RequestBus = AZ::EBus<<GemName>Requests>;
```

An example of all of these changes is `ILightningArcGem.h` (now `LightningArcBus.h`), which before looked like this:

```
// ILightningArcGem.h
#pragma once
#include "IGem.h"

class CLightningGameEffect;
class CScriptBind_LightningArc;

class ILightningArcGem
    : public IGem
{
public:
    CRYINTERFACE_DECLARE(ILightningArcGem, 0xf1b3a17f9c61410a,
        0x8783fc1ff9854125);
public:
    virtual CScriptBind_LightningArc* GetScriptBind() const = 0;
    virtual CLightningGameEffect* GetGameEffect() const = 0;
};
```

And after looks like:

```
// LightningArcBus.h
#pragma once
#include <AzCore/EBus/EBus.h>

class CLightningGameEffect;
class CScriptBind_LightningArc;

class LightningArcRequests
    : public AZ::EBusTraits
{
public:

    //////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
    // EBusTraits overrides
    static const AZ::EBusHandlerPolicy HandlerPolicy =
        AZ::EBusHandlerPolicy::Single;
    static const AZ::EBusAddressPolicy AddressPolicy =
        AZ::EBusAddressPolicy::Single;

    //////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

    virtual CScriptBind_LightningArc* GetScriptBind() const = 0;
    virtual CLightningGameEffect* GetGameEffect() const = 0;
```

```
};  
using LightningArcRequestBus = AZ::EBus<LightningArcRequests>;
```

3. Convert all calls through the **GemManager** to your code with calls to `EBUS_EVENT(<GemName>RequestBus, etc.)`. For more information, see [Event Bus \(EBus\)](#). Here is an example:

```
// BEFORE: calling through the GemManager  
CLightningGameEffect* gameEffect =  
    GetISystem(>GetGemManager(>GetGem<ILightningArcGem>()->GetGameEffect());  
  
// AFTER: calling through the EBus  
CLightningGameEffect* gameEffect = nullptr;  
EBUS_EVENT_RESULT(gameEffect, LightningArcRequestBus, GetGameEffect);
```

4. Make the following modifications to the `<GemName>Gem.h` file.
 - a. Change the `<GemName>Gem` base class from `I<GemName>Gem` to `CryHooksModule`
 - b. In the `<GemName>Gem` class, add inheritance from `<GemName>RequestBus::Handler`.

Note

Other classes can implement the bus handler instead. For example, the OpenVR Gem creates a system component to handle bus requests.

- c. Replace the `GEM_IMPLEMENT_WITH_INTERFACE` line with one declaring type information. This requires the class name, a unique UUID (Visual Studio has a [tool](#) you can use to get unique values), and the module base class. For example:

```
AZ_RTTI(LightningArcGem, "{89724952-ADBF-478A-AFFE-784BD0952E2D}",  
    CryHooksModule);
```

- d. Declare a default constructor and destructor. These used to be declared by the `GEM_IMPLEMENT_WITH_INTERFACE` macro.

The following example, from `LightningArcGem.h` shows what the file looked like before the changes:

```
// LightningArcGem.h  
#ifndef _GEM_LIGHTNINGARC_H_  
#define _GEM_LIGHTNINGARC_H_  
#include <GameEffectSystem/IGameEffectSystem.h>  
#include "LightningArc/ILightningArcGem.h"  
  
class LightningArcGem  
    : public ILightningArcGem  
    , public GameEffectSystemNotificationBus::Handler  
{  
public:  
    GEM_IMPLEMENT_WITH_INTERFACE(LightningArcGem, ILightningArcGem,  
        0x8eccf081ff02476f, 0xb8ec7c4c20cc603c)  
    void OnSystemEvent(ESystemEvent event, UINT_PTR wparam, UINT_PTR  
        lparam) override;  
    void PostSystemInit();  
    void Shutdown();  
public:  
    CScriptBind_LightningArc* GetScriptBind() const override;  
    CLightningGameEffect* GetGameEffect() const override;  
protected:  
    CLightningGameEffect* m_gameEffect;
```

```

CScriptBind_LightningArc* m_lightningArcScriptBind;
int g_gameFXLightningProfile;

//////////////////////////////////////
// GameEffectSystemNotificationBus
void OnReleaseGameEffects() override;

//////////////////////////////////////
};
#endif // _GEM_LIGHTNINGARC_H_

```

Here is the LightningArcGem.h file after the changes are made:

```

// LightningArcGem.h
#ifndef _GEM_LIGHTNINGARC_H_
#define _GEM_LIGHTNINGARC_H_
#include <GameEffectSystem/IGameEffectSystem.h>
#include <LightningArc/LightningArcBus.h>

class LightningArcGem
: public CryHooksModule
, public LightningArcRequestBus::Handler
, public GameEffectSystemNotificationBus::Handler
{
public:
    AZ_RTTI(LightningArcGem, "{89724952-ADBF-478A-AFFE-784BD0952E2D}",
    CryHooksModule);
    LightningArcGem();
    ~LightningArcGem() override;
    void OnSystemEvent(ESystemEvent event, UINT_PTR wparam, UINT_PTR
    lparam) override;
    void PostSystemInit();
    void Shutdown();
public:
    CScriptBind_LightningArc* GetScriptBind() const override;
    CLightningGameEffect* GetGameEffect() const override;
protected:
    CLightningGameEffect* m_gameEffect = nullptr;
    CScriptBind_LightningArc* m_lightningArcScriptBind = nullptr;
    int g_gameFXLightningProfile;

    ////////////////////////////////////////
    // GameEffectSystemNotificationBus
    void OnReleaseGameEffects() override;

    ////////////////////////////////////////
};
#endif // _GEM_LIGHTNINGARC_H_

```

5. Perform the following steps to modify your `<GemName>Gem.cpp` file.
 - a. If your gem contains `AZ::Component` instances, register them in the constructor:

```

<GemName>Gem::<GemName>Gem()
{
    m_descriptors.insert(m_descriptors.back(), {
        <MyComponent>::CreateDescriptor(),
    });
}

```

```
...}
```

- b. If this class inherits from `<GemName>RequestBus::Handler`, connect to the bus in the constructor and disconnect in the destructor:

```
<GemName>Gem::<GemName>Gem()  
{  
    ...  
    <GemName>RequestBus::Handler::BusConnect();  
}  
  
<GemName>Gem::~<GemName>Gem()  
{  
    <GemName>RequestBus::Handler::BusDisconnect();  
    ...  
}
```

- c. If there is a call to `RegisterFlowNodes()`, replace it with `RegisterExternalFlowNodes()`.
- d. Perform the following steps to convert the `REGISTER_GEM` macro to `AZ_DECLARE_MODULE_CLASS`:
- Copy the all lower-case UUID from your `gem.json` file.
 - Replace `GEM_REGISTER(<GemName>Gem)` with:

```
AZ_DECLARE_MODULE_CLASS(<GemName>_<GemUUID>, fully qualified module  
class, usually <GemName>::<GemName>Gem)
```

The following example, `LightningArcGem.cpp`, shows what the file looked like before the changes:

```
// LightningArcGem.cpp  
#include "StdAfx.h"  
#include <platform_impl.h>  
#include <IEntityClass.h>  
#include "LightningArcGem.h"  
#include <FlowSystem/Nodes/FlowBaseNode.h>  
#include "LightningArc.h"  
#include "LightningGameEffect.h"  
#include "ScriptBind_LightningArc.h"  
  
LightningArcGem::LightningArcGem() {}  
  
LightningArcGem::~LightningArcGem() {}  
  
void LightningArcGem::PostSystemInit()  
{  
    REGISTER_CVAR(g_gameFXLightningProfile, 0, 0, "Toggles game effects  
system lightning arc profiling");  
    // Init GameEffect  
    m_gameEffect = new CLightningGameEffect();  
    m_gameEffect->Initialize();  
    // Init ScriptBind  
    m_lightningArcScriptBind = new CScriptBind_LightningArc(GetISystem());  
    // Init GameObjectExtension
```

```
// Originally registered with REGISTER_GAME_OBJECT(pFramework,
LightningArc, "Scripts/Entities/Environment/LightningArc.lua");
// If more objects need registered, consider bringing the macro back
along with the GameFactory wrapper.
IEntityClassRegistry::SEntityClassDesc clsDesc;
clsDesc.sName = "LightningArc";
clsDesc.sScriptFile = "Scripts/Entities/Environment/LightningArc.lua";
static CLightningArcCreator _creator;
GetISystem()->GetIGame()->GetIGameFramework()->GetIGameObjectSystem()-
>RegisterExtension("LightningArc", &_creator, &clsDesc);
}

void LightningArcGem::Shutdown()
{
    SAFE_DELETE(m_gameEffect);
    SAFE_DELETE(m_lightningArcScriptBind);
}

void LightningArcGem::OnSystemEvent(ESystemEvent event, UINT_PTR wparam,
UINT_PTR lparam)
{
    switch (event)
    {
        case ESYSTEM_EVENT_GAME_POST_INIT:
            IComponentFactoryRegistry::RegisterAllComponentFactoryNodes(*gEnv-
>pEntitySystem->GetComponentFactoryRegistry());
            GameEffectSystemNotificationBus::Handler::BusConnect();
            break;
        case ESYSTEM_EVENT_FLOW_SYSTEM_REGISTER_EXTERNAL_NODES:
            RegisterFlowNodes();
            break;
        // Called on ESYSTEM_EVENT_GAME_POST_INIT_DONE instead of
        ESYSTEM_EVENT_GAME_POST_INIT because the GameEffectSystem Gem
        // uses ESYSTEM_EVENT_GAME_POST_INIT to initialize, and this requires
        that has happened already.
        case ESYSTEM_EVENT_GAME_POST_INIT_DONE:
            PostSystemInit();
            break;
        case ESYSTEM_EVENT_FULL_SHUTDOWN:
        case ESYSTEM_EVENT_FAST_SHUTDOWN:
            GameEffectSystemNotificationBus::Handler::BusDisconnect();
            Shutdown();
            break;
    }
}

CScriptBind_LightningArc* LightningArcGem::GetScriptBind() const
{
    return m_lightningArcScriptBind;
}

CLightningGameEffect* LightningArcGem::GetGameEffect() const
{
    return m_gameEffect;
}

void LightningArcGem::OnReleaseGameEffects()
{
    Shutdown();
}
```

```
}  
  
GEM_REGISTER(LightningArcGem)
```

After the changes, `LightningArcGem.cpp` looks like this:

```
// LightningArcGem.cpp  
#include "StdAfx.h"  
#include <platform_impl.h>  
#include <IEntityClass.h>  
#include "LightningArcGem.h"  
#include <FlowSystem/Nodes/FlowBaseNode.h>  
#include "LightningArc.h"  
#include "LightningGameEffect.h"  
#include "ScriptBind_LightningArc.h"  
  
LightningArcGem::LightningArcGem()  
{  
    LightningArcRequestBus::Handler::BusConnect();  
}  
  
LightningArcGem::~LightningArcGem()  
{  
    LightningArcRequestBus::Handler::BusDisconnect();  
}  
  
void LightningArcGem::PostSystemInit()  
{  
    REGISTER_CVAR(g_gameFXLightningProfile, 0, 0, "Toggles game effects  
system lightning arc profiling");  
    // Init GameEffect  
    m_gameEffect = new CLightningGameEffect();  
    m_gameEffect->Initialize();  
    // Init ScriptBind  
    m_lightningArcScriptBind = new CScriptBind_LightningArc(GetISystem());  
    // Init GameObjectExtension  
    // Originally registered with REGISTER_GAME_OBJECT(pFramework,  
LightningArc, "Scripts/Entities/Environment/LightningArc.lua");  
    // If more objects need registered, consider bringing the macro back  
along with the GameFactory wrapper.  
    IEntityClassRegistry::SEntityClassDesc clsDesc;  
    clsDesc.sName = "LightningArc";  
    clsDesc.sScriptFile = "Scripts/Entities/Environment/LightningArc.lua";  
    static CLightningArcCreator _creator;  
    GetISystem()->GetIGame()->GetIGameFramework()->GetIGameObjectSystem()-  
>RegisterExtension("LightningArc", &_creator, &clsDesc);  
}  
  
void LightningArcGem::Shutdown()  
{  
    SAFE_DELETE(m_gameEffect);  
    SAFE_DELETE(m_lightningArcScriptBind);  
}  
  
void LightningArcGem::OnSystemEvent(ESystemEvent event, UINT_PTR wparam,  
    UINT_PTR lparam)  
{  
    switch (event)
```



```
{
    case ESYSTEM_EVENT_GAME_POST_INIT:
        IComponentFactoryRegistry::RegisterAllComponentFactoryNodes(*gEnv-
>pEntitySystem->GetComponentFactoryRegistry());
        GameEffectSystemNotificationBus::Handler::BusConnect();
        break;
    case ESYSTEM_EVENT_FLOW_SYSTEM_REGISTER_EXTERNAL_NODES:
        RegisterExternalFlowNodes();
        break;
    // Called on ESYSTEM_EVENT_GAME_POST_INIT_DONE instead of
    ESYSTEM_EVENT_GAME_POST_INIT because the GameEffectSystem Gem
    // uses ESYSTEM_EVENT_GAME_POST_INIT to initialize, and this requires
    that has happened already.
    case ESYSTEM_EVENT_GAME_POST_INIT_DONE:
        PostSystemInit();
        break;
    case ESYSTEM_EVENT_FULL_SHUTDOWN:
    case ESYSTEM_EVENT_FAST_SHUTDOWN:
        GameEffectSystemNotificationBus::Handler::BusDisconnect();
        Shutdown();
        break;
}
}

CScriptBind_LightningArc* LightningArcGem::GetScriptBind() const
{
    return m_lightningArcScriptBind;
}

CLightningGameEffect* LightningArcGem::GetGameEffect() const
{
    return m_gameEffect;
}

void LightningArcGem::OnReleaseGameEffects()
{
    Shutdown();
}

AZ_DECLARE_MODULE_CLASS(LightningArc_4c28210b23544635aa15be668dbff15d,
    LightningArcGem)
```

C. Edit Your Gem.json File

A simple change in your `gem.json` file signals Lumberyard that your gem is now an AZ module.

To edit the `gem.json` file

1. Open `gem.json`.
2. Increment `GemFormatVersion` to 3.

D. Migrate Your Config Files

The final step is to update your game's configuration files so that it recognizes your gem as a proper AZ module.

To migrate your config files

1. Open a command prompt and use the `cd` command to navigate to the `Bin64` directory.
2. Type the following command:

```
lmb.exe projects populate-appdescriptors
```

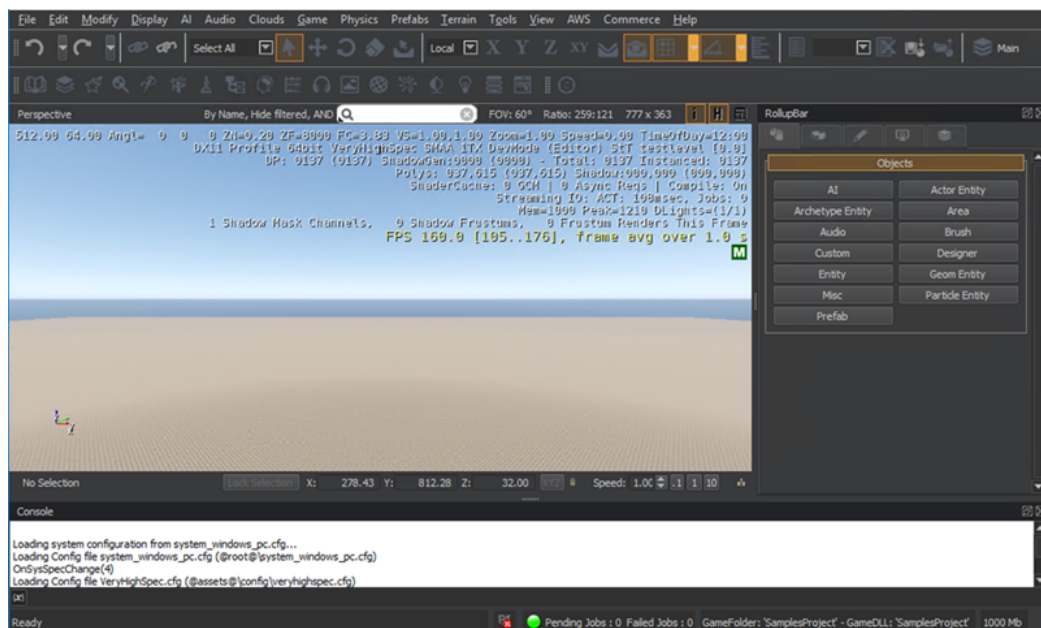
Using Lumberyard Editor

Lumberyard Editor is the primary workspace editor for Lumberyard and combines a running game with a full suite of tools to edit the game. You can access Lumberyard Editor by navigating to `\dev\Bin64` and double-clicking `Editor.exe`.

Note

When starting Lumberyard for the first time, if you encounter errors about `AssetProcessor.exe`, `AssetProcessor_temp.exe`, or `CrySystem.dll`, check whether your antivirus software is placing these files in quarantine and, if possible, grant exceptions for the affected files.

Lumberyard Editor consists of various menus, toolbars, and a viewport window. By default, Lumberyard Editor also opens the Rollup Bar for object selection and the console window for console variables.



Topics

- [Lumberyard Editor Interface \(p. 34\)](#)
- [Using the Menu Bar in Lumberyard Editor \(p. 36\)](#)
- [Using the Top Toolbars \(p. 45\)](#)

- [Using the Bottom Toolbar \(p. 47\)](#)
- [Using Shortcut Keys \(p. 49\)](#)
- [Using the Viewport \(p. 51\)](#)
- [Using the Rollup Bar \(p. 52\)](#)
- [Using the Console Window \(p. 56\)](#)
- [Customizing Your Workspace \(p. 58\)](#)
- [Restoring Default Settings for Lumberyard Editor \(p. 66\)](#)

Lumberyard Editor Interface

Lumberyard Editor has the following main panels and bars:

- **Viewport** – 3D viewport window that displays the game environment and allows you to view, create, and interact with assets
- **Rollup Bar** – Right pane that provides access to objects (AI and asset entities), terrain tools, modeling, and a layer organizer
- **Toolbars** – Top and bottom toolbars that provide quick access to the most commonly used functions and features; the toolbars can be customized based on your preferences
- **Console** – Console log window that shows console command variables

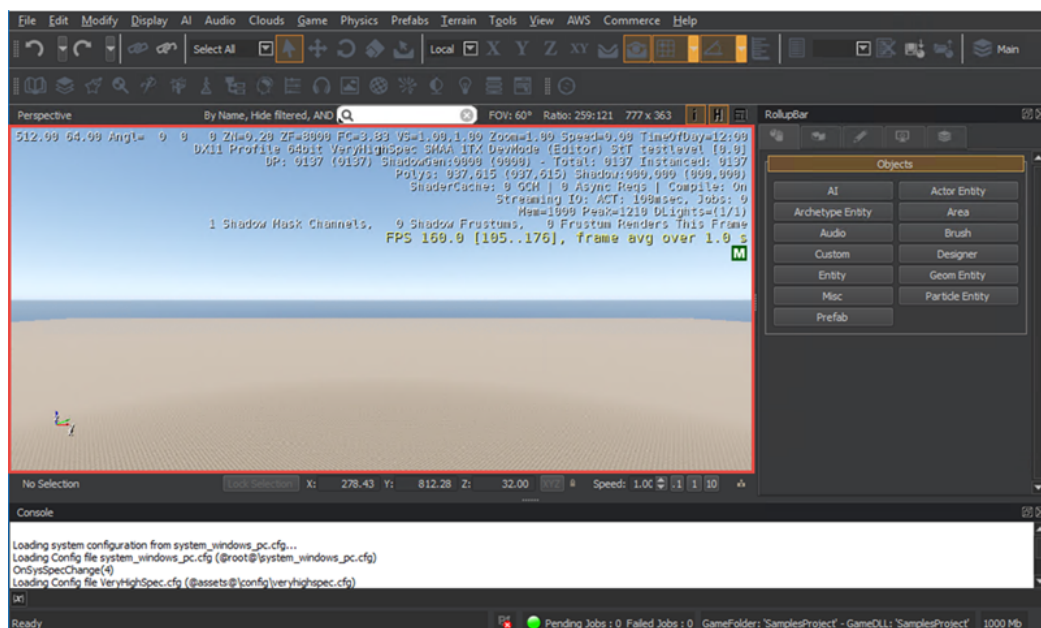
Tip

You can enter game mode by pressing **CTRL+G** on your keyboard. Press **Esc** to exit game mode and enter editing mode.

Viewport

To fly around your level, move your mouse cursor over the view port and hold down the right mouse button. With the right mouse button pressed, move the mouse to look around and press the **A**, **W**, **S** and **D** keys to move. Hold down the **Shift** key to move at 10x your normal speed.

For more information, see [Using the Viewport \(p. 51\)](#).

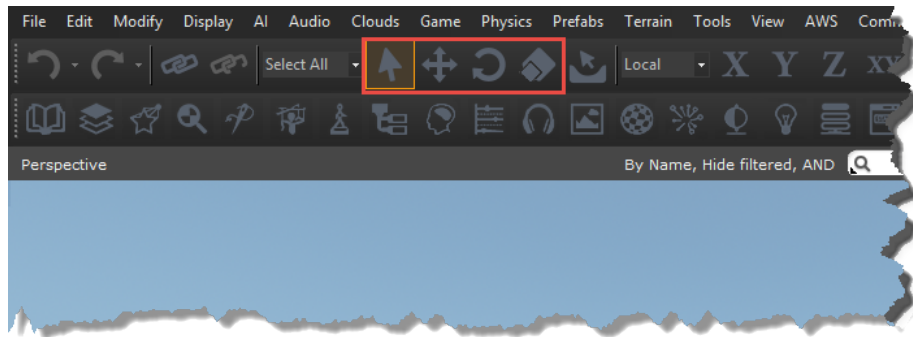


Toolbars

The toolbars at the top of Lumberyard Editor provide quick access to the most commonly used functions and features. Use the four buttons shown in the image to select, move, rotate, and scale objects. You can also use the hot keys 1, 2, 3, and 4 to accomplish these actions.

For example, you can move an object by clicking the move button, clicking the object you want to move in the viewport, and dragging the object to a new location with your mouse.

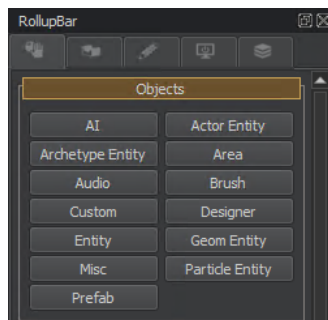
For more information, see [Using the Top Toolbars \(p. 45\)](#).



Rollup Bar

The **Rollup Bar** is a multi-tabbed window used for multiple purposes. Each section on the **Objects** tab has different objects you can add to your level.

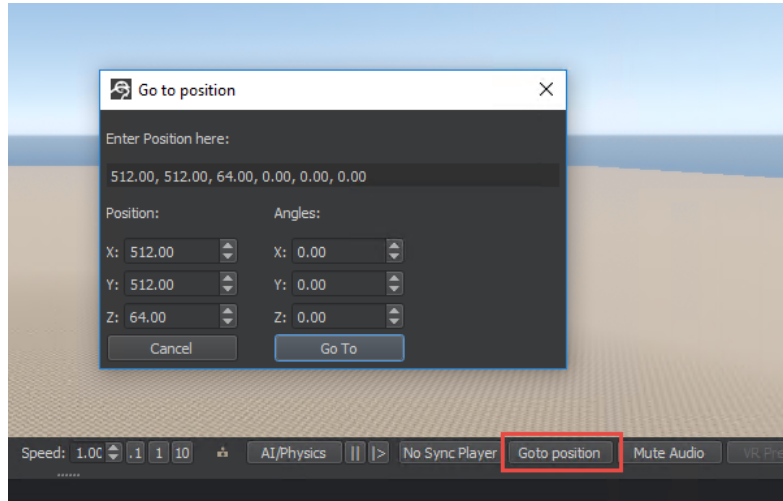
For more information, see [Using the Rollup Bar \(p. 52\)](#).



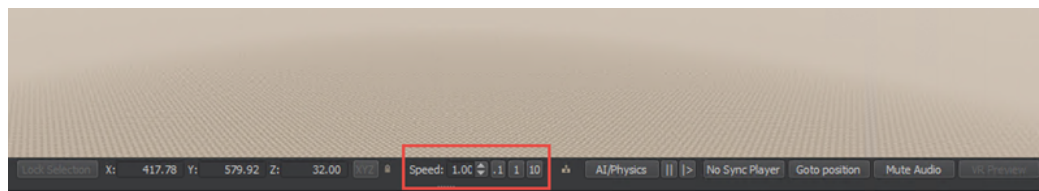
Bottom Toolbar

You can use the **Go to position** button in the bottom toolbar to navigate to a precise X, Y, and Z location in the viewport. To transport to a point above the terrain, you can change the X, Y, and Z values to 1024, 1024, and 34, respectively.

For more information, see [Using the Bottom Toolbar \(p. 47\)](#).



If it doesn't appear as if you're moving, you can select a faster move speed by entering a number in the **Speed** field or by clicking one of the preset speed buttons: **0.1**, **1**, or **10**.



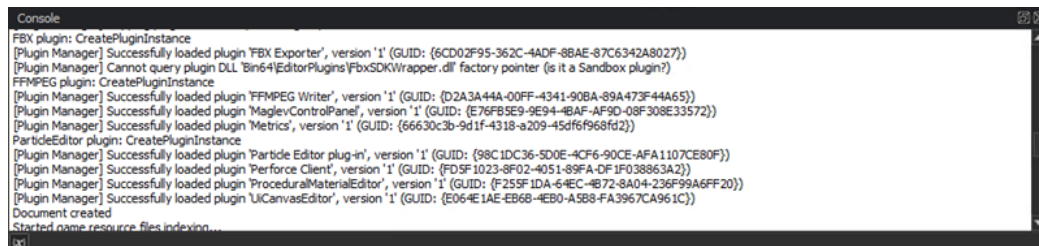
Console

Enter commands in the console window to change settings for your game or execute functionality, such as connecting to a server or banning a player. The console window also displays warnings and errors for your game level, such as missing textures and models.

For more information, see [Using the Console Window \(p. 56\)](#).

Tip

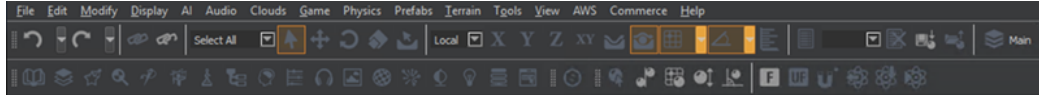
You can access the console while in-game by pressing the tilde (~) key.



Using the Menu Bar in Lumberyard Editor

You can use the main menu bar in Lumberyard Editor to access basic file operations and display options to more advanced features, such as terrain and level editing tools and AI settings.

You can perform many of these commands by using the toolbar buttons or keyboard shortcuts.



File Menu

The **File** menu includes commands for file handling, such as open and save level file, show log file, and a list of recently loaded levels.

Menu Item	Description
New	Creates a new level
Open	Opens an existing level
Save	Saves the level
Save As	Saves the level with a new name
Save Modified External Layers	Saves only the external layers that have been modified since the last save
Save Level Resources	Saves all assets used in the level
Load Objects	Loads objects from the game directory
Save Objects	Save objects to a <code>.grp</code> (group) file
Switch Projects	Close the current project
Configure Project	Configure Gems and settings for projects
Export to Engine	Exports the level data to the <code>level.pak</code> file so the level can be played in game mode
Export Selected Objects	Saves the selected geometry to an <code>.obj</code> or <code>.fbx</code> file
Export Occlusion Mesh	Exports the occlusion mesh
Show Log File	Shows the log file containing all text printed in the console
Global Preferences	Customize configuration, keyboard, and editor settings
Recent files list	Lists recently opened levels
Exit	Quits Lumberyard Editor, prompting you to save first if changes are detected

File Configure Menu

The **File, Global Preferences, Configure** menu allows you to quickly switch between predefined viewport quality settings to check memory footprints, visual quality differences, and features available in the different modes.

Menu Item	Description
Very High	Enables very high-resolution display settings (some DX11 specific)

Menu Item	Description
High	Enables high-resolution display settings
Medium	Enables medium-resolution display settings
Low	Enables low-resolution display settings
XBoxOne	Emulates Xbox One display setting
PS4	Emulates PlayStation 4 display settings
Android	Emulates Android display settings
iOS	Emulates Apple iOS display settings

Edit Menu

The **Edit** menu includes commands for object manipulation and selection.

Menu Item	Description
Undo	Reverts the last action
Redo	Applies the last action
Select	<ul style="list-style-type: none"> • All – Selects all visible, non-frozen objects • None – Deselects the objects that are currently selected • Invert – Inverts the selection, so the unselected object becomes selected and the objects currently selected become deselected • Objects – Opens the Select Objects tool • Lock Selection – Locks the selected object • Next Selection Mask – Selects the next selection mask
Hide Selection	Hides the selected object
Show Last Hidden	Shows the last hidden object
Unhide All	Unhides all hidden objects
Link	Creates hierarchies between objects
Unlink	Removes the link between linked objects
Group	Groups multiple selected objects together and draws a green box around those objects
Ungroup	Ungroups all grouped objects
Open Group	Opens the group so you can modify individual objects
Close Group	Closes the group
Attach to Group	Adds the selected object to the selected group
Detach from Group	Removes the selected object from the selected group
Freeze Selection	Freezes the selected object

Menu Item	Description
Unfreeze All	Unfreezes all frozen objects
Hold	Saves the current state
Fetch	Restores the saved state
Delete	Deletes the selected object
Clone	Duplicates the selected object

Modify Menu

The **Modify** menu includes commands for modifying and changing attributes and properties such as height, alignment, and material of objects and entities.

Menu Item	Description
Convert to	Converts the selected object to a brush, geom entity, designer object, StaticEntity, or GameVolume
Sub Object Mode	Selects and edits various geometry components, if an object is selected using the edit mesh function
Set Object(s) Height	Moves the object to a specified height (in meters) above the terrain
Rename Object(s)	Renames the selected object
Transform Mode	<ul style="list-style-type: none"> • Select Mode • Move • Rotate • Scale • Select Terrain
Constrain	Limits movement to the X, Y, Z axes, XY planes, or to the surface of the terrain and objects
Align	Aligns an object to the grid, to another object, or to the selected surface, which moves the pivot point of the object
Snap	Snaps an object to the grid or a rotational increment
Fast Rotate	Quickly rotates the selected object on the selected axis with the degree value specified for Rotate Angle

Display Menu

The **Display** menu allows you to toggle display features for level design, entity placement, and object manipulation. You can also access other commands, such as Remember/Goto Location and viewport navigation speed.

Menu Item	Description
Toggle Fullscreen MainWindow	Toggles the viewport to and from full screen mode when the viewport is not docked in Lumberyard Editor
Wireframe	Enables wireframe rendering view
Ruler	Enables the Ruler tool to measure distance
Grid Settings	Sets grid line spacing, angle snapping, and rotation and translation settings
Switch Camera	<ul style="list-style-type: none"> • Default Camera – Selects the default camera • Sequence Camera – Selects the camera used in a Track View sequence • Selected Camera Object – Selects the camera entity • Cycle Camera – Selects the next camera
Change Move Speed	Changes movement speed of all objects in the level
Goto Coordinates	Specifies the camera position in XYZ coordinates, and moves the camera to that position
Goto Selection	Jumps to the currently selected object in the viewport
Goto Location	Jumps to one of 10 predefined locations in the viewport
Remember Location	Saves up to 10 locations in the viewport
Configure Layout	Selects a preconfigured layout
Cycle Viewports	Changes the viewport to the next view type
Show/Hide Helpers	Shows or hides all helper objects

AI Menu

The **AI** menu includes commands for generating AI navigation and updating the AI system within a level.

Menu Item	Description
Generate All AI	Generates all AI navigation and performs the tasks listed below
Generate Triangulation	Generates triangulation of the navigation mesh used for outdoor levels
Generate 3D Navigation Volumes	Generates 3D navigation data for 3D volumes used by alien AI agents; volumes are defined by <code>AINavigationModifier</code> and a <code>Volume NavType</code>
Generate Flight Navigation	Generates 2.5D navigation data for volumes used by flying AI agents; volumes are defined by <code>AINavigationModifier</code> and a <code>Flight NavType</code>
Generate Waypoints	Generates links for indoor waypoints

Menu Item	Description
Validate Navigation	Checks navigation data for various problems (for example, bad object placement, overlapping forbidden areas, corruptions) and displays warnings if any problems are found
Clear All Navigation	Removes all navigation information from the level
Generate Spawner Entity Code	Looks for AI entity classes and generates an <code>.ent</code> file for each; associates an entity class name with the Lua base file for that entity
Generate 3D Debug Voxels	Generates debugging information for volume navigation regions when the <code>ai_DebugDraw</code> console variable is enabled
Create New Navigation Area	Creates a new navigation area
Request a full MNM rebuild	Performs a full rebuild of all MNM mesh data
Show Navigation Areas	Displays MNM navigation areas
Add Navigation Seed	Adds a navigation seed entity
Continuous Update	Toggles continuous MNM data updates; if disabled, mesh data will not update until a rebuild is requested
Visualize Navigation Accessibility	Displays inaccessible areas in red and accessible areas in blue
MediumSizedCharacters	Toggles the navigation debug display
Generate Cover Surfaces	Generates cover surface data
AIPoint Pick Link	Combines AI navigation modifier points
AIPoint Pick Impass Link	Restricts AI navigation modifier points so AI cannot walk on them

Audio Menu

The **Audio** menu includes commands for showing the music currently playing and accessing the sound and dialog browsers.

Menu Item	Description
Stop All Sounds	Silences all sounds in the level
Refresh Audio	Refreshes all sounds in the level

Clouds Menu

The **Clouds** menu allows you to create, open, close, and delete your custom cloud assets.

Menu Item	Description
Create	Creates a new cloud asset
Destroy	Deletes a created cloud asset

Menu Item	Description
Open	Opens the selected cloud asset
Close	Closes the selected cloud asset

Game Menu

The **Game** menu includes commands for enabling the game mode and testing newly created features.

Menu Item	Description
Switch to Game	Enables game mode (press Esc to exit game mode)
Enable Physics/AI	Enables physics and AI
Terrain Collision	Makes the camera collide with the terrain so you cannot fly under the terrain surface
Synchronize Player with Camera	Sets the player position relative to the camera position
Edit Equipment-Packs	Opens the Equipment Packs window
Toggle SP/MP GameRules	Toggles between SinglePlayer and TeamInstantAction game rules

Physics Menu

The **Physics** menu includes commands to make physics simulations.

Menu Item	Description
Get Physics State	Retrieves the current physics state
Reset Physics State	Resets the physics state to its original position
Simulate Objects	Makes objects respond to the force of gravity

Prefabs Menu

The **Prefab** menu includes commands to make prefabs from a selection, reload prefabs, and add selected objects to the prefab library.

Menu Item	Description
Create Prefab from Selected Object(s)	Creates a new prefab from selected objects
Add Selected Object(s) to Prefab	Adds selected objects to the prefab
Clone Selected Object(s)	Clones selected objects

Menu Item	Description
Extract Selected Object(s)	Extracts selected objects from the prefab
Open All	Opens all prefabs
Close All	Closes all prefabs
Reload All	Reloads all prefabs

Terrain Menu

The **Terrain** menu allows you to access view panes and tools that affect the game world and terrain appearance.

Menu Item	Description
Edit Terrain	Opens the Terrain Editor
Terrain Texture Layers	Opens the Terrain Texture Layers tool
Export/Import Megaterrain Texture	Exports or imports the megaterrain texture
Sun Trajectory Tool	Opens the Sun Trajectory Tool
Time Of Day	Opens the Time of Day Editor
Reload Terrain	Reloads the selected terrain
Export Terrain Block	Exports a section of the terrain to a terrain block <code>.trb</code> file
Import Terrain Block	Imports terrain from a saved <code>.trb</code> file
Resize Terrain	Opens the Terrain Resize tool
Terrain Modify	Opens the Terrain, Modify panel in the Rollup Bar
Edit Vegetation	Opens the Vegetation panel in the Rollup Bar
Paint Layers	Opens the Layer Painter panel in the Rollup Bar
Refine Terrain Texture Tiles	Divides the terrain tiles into smaller sections

Tools Menu

The **Tools** menu allows you to reload scripts, textures, geometry, and terrain. Other commands include user command configuration and check level for errors.

Menu Item	Description
Reload Scripts	Reloads all entities
Reload Textures/Shaders	Reloads all textures and shaders used in the level
Reload Geometry	Reloads geometries used in the level

Menu Item	Description
Reload Terrain	Reloads all terrain
Resolve Missing Objects/ Materials	Runs a check through the level and attempts to resolve all object and material issues
Enable file change monitoring	Enables monitoring of file changes
Clear Registry Data	Clears all custom toolbar registry data
Check Level for Errors	Checks the level for errors (e.g. duplicate objects and missing assets) and displays a list in the console window
Check Object Positions	Checks the positions of all objects in the level
Save Level Statistics	Saves level statistics to the <i>yourlevelname</i> .xml file in the TestResults folder
Advanced	<ul style="list-style-type: none"> • Compile Script – Compiles an entity script • Reduce Working Set – Reduces memory consumption • Update Procedural Vegetation – Updates all procedural vegetation
Configure Toolbox Macros	Opens the Tools Configuration window for creating shortcuts to the console commands
Toolbox Macros	Displays the shortcuts to the console and Lumberyard Editor commands, as specified in the Configure User Commands window
Script Help	Opens the Script Help window, which lists all commands, descriptions, and examples

View Menu

The **View** menu allows you to customize Lumberyard Editor and provides access to the various editors, user layouts, and skins.

Menu Item	Description
Open View Pane	Lists all Lumberyard Editor tools and sub-editors For more information about the tools and sub-editors, see Lumberyard Editors and Tools (p. 4) .
Show Rollup Bar	Displays the Rollup Bar panel
Show Console	Displays the console window
Show Quick Access Bar	Displays the quick access bar
Layouts	<ul style="list-style-type: none"> • Save Layout – Saves the current layout • Restore Default Layout – Restores the default layout

AWS Menu

The **AWS** menu allows you to sign up for an Amazon Web Services (AWS) account.

Menu Item	Description
Sign up for AWS	Goes to http://aws.amazon.com/

Commerce Menu

The **Commerce** menu allows you to learn how to submit your game to Amazon's Digital Software store.

Menu Item	Description
Merch by Amazon	Goes to https://merch.amazon.com/landing
Publishing on Amazon	Goes to https://developer.amazon.com/appsandservices/solutions/platforms/mac-pc

Help Menu

The **Help** menu includes a link to the online help documentation, support contact information, and Lumberyard Editor version information.

Menu Item	Description
Getting Started	Links to the online Lumberyard Getting Started Guide and Tutorials
Documentation	Links to the online Lumberyard Glossary, User Guide, GameLift Documentation, and Release Notes
GameDev Resources	Links to the GameDev Blog, Twitch Channel, Tutorials, Forums, and AWS Support
Give Us Feedback	Lists contact information for Lumberyard support
About Lumberyard	Displays the version of Lumberyard Editor

Using the Top Toolbars

Lumberyard Editor provides toolbars that allow you to quickly and easily access the various editors. You can configure and customize these toolbars to fit your needs. For example, you can right-click the toolbar to toggle the display of the following:

- EditMode Toolbar
- Object Toolbar
- Editors Toolbar
- Substance Toolbar

You can arrange toolbars horizontally at the top of the editor, vertically on the edges, or undocked from the editor.

In addition to using the toolbar, you can access the various editors by clicking **View, Open View Pane**.

For more information about the bottom toolbar, see [Using the Bottom Toolbar \(p. 47\)](#).

For more information about accessing functions by using keyboard shortcut keys, see [Using Shortcut Keys \(p. 49\)](#).

EditMode Toolbar



The **EditMode** toolbar includes various tools for general level editing:

- **A** – Reverts or applies the last command
- **B** – Links or unlinks the selected object
- **C** – Filters what you can select in the viewport: all, brushes, no brushes, entities, prefabs, areas, shapes, AI points, decals, solids, or no solids
- **D** – Translation tools used to select an object or object type, move an object, rotate an object, scale an object, select a terrain area, or rotate a terrain area
- **E** – Selects the reference coordinate system
- **F** – Specifies the axis constraint by locking on the X, Y, and Z axis
- **G** – Object placement tools used to follow the terrain, snap to objects, snap to grid, snap to angle, or show the ruler
- **H** – Asset organization tools used to open the object selector, create a selection, delete a selection, save selected objects, or load selected objects
- **I** – Selects the current layer

Object Toolbar



The **Object** toolbar includes various tools for object alignment and manipulation:

- **A** – Goes to the selected object
- **B** – Aligns the selection to an object by choosing the source object, clicking the tool, and then clicking the target object
- **C** – Aligns the object to the grid
- **D** – Sets the object's height
- **E** – Aligns the object to the terrain surface normal (hold **Ctrl** for object surface normal alignment)
- **F** – Freezes or unfreezes the selected object
- **G** – Vertex snapping for the selected object
- **H** – Resets the physics state for the selected object, get the physics state for the selected object, or simulate physics on the selected object

Editors Toolbar



The **Editors** toolbar includes 19 buttons that are used to access various tools:

- **A** – Opens the **Asset Browser**
- **B** – Opens the **Layer Editor**
- **C** – Opens the **LOD Generator**
- **E** – Opens the **Material Editor**
- **F** – Opens **Geppetto** (character and animation tools)
- **G** – Opens the **Mannequin Editor**
- **H** – Opens **Flow Graph**
- **I** – Opens the **AI Debugger**
- **J** – Opens the **Track View Editor**
- **K** – Opens the **Audio Controls Editor**
- **L** – Opens the **Terrain Editor**
- **M** – Opens the **Terrain Texture Layers Editor**
- **N** – Opens the **Particle Editor**
- **O** – Opens the **Time of Day Editor**
- **P** – Opens the **Sun Trajectory Tool**
- **Q** – Opens the **Database View**
- **R** – Opens the **UI Editor**
- **S** – Opens the **Substance Editor**

Using the Bottom Toolbar

Lumberyard Editor includes a bottom status/toolbar that is used for the purposes below.



Status

The **status** bar (1) displays the number of selected object(s) and provides functional hints for buttons or menu items in Lumberyard Editor. The status line is located at the bottom of the screen.

Lock Selection

The **Lock Selection** button (2) toggles selection locking, preventing you from inadvertently selecting something else in a level.

When your selection is locked, you can click or drag the mouse anywhere in the viewport without losing your selection. To deselect or alter your selection, click **Lock Selection** again to unlock the selection.

Coordinates/Transforms



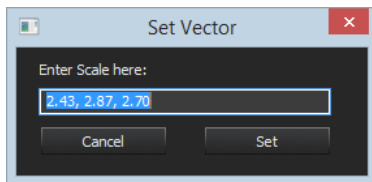
The **coordinates/transform** area (3) shows the position of the cursor or the status of a transform, and allows you to enter new transform values. The information in these fields vary based on your tasks:

- When creating an object or moving the mouse in the viewport, these fields show the cursor location in absolute world coordinates.
- When transforming an object by dragging it in the viewport, these fields show coordinates relative to the object's coordinates before the transformation started.

- While transforming an object, these fields change to spinners in which you can directly type values.
- When the transform button is active and a single object is selected, but you are not dragging the object, these fields show the absolute coordinates for the current transform.
- While the transform button is active and multiple objects are selected, these fields show the previous selection's transform coordinates.

Set Vector

The **Set Vector** button (4) allows you to set the vector scale for your selected object(s). You can lock the proportions by clicking the lock button.



Speed Control

The **Speed** button (5) allows you to change the speed of all movements in the viewport. The three buttons to the right of the **Speed** change the speed to 0.1, 1, or 10. You can also manually set the speed by entering your values into the fields or using the spinners to adjust the speed up or down.

Terrain Collision

The **Terrain Collision** button (6) toggles terrain collision. You can enable terrain collision to inhibit camera movement below the terrain surface.

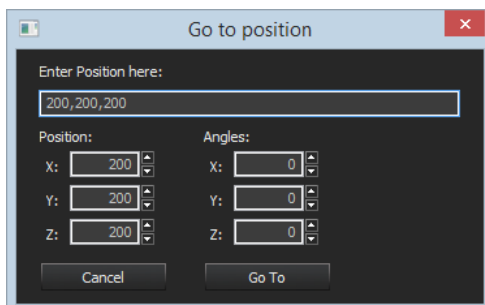
AI/Physics

The **AI/Physics** button (7) toggles physics simulation and AI, allowing you to test physics and AI behavior directly in the editor without entering game mode.

No Sync Player

The **No Sync Player** button (8) detaches the player entity from the camera. While in editor mode, a character entity is attached to the camera that is otherwise always synchronized. The No Sync Player function can be useful with AI or Physics enabled, when you don't want to activate triggers while navigating through a level.

Goto Position



The **Go to Position** button (9) opens the **Go to position** dialog box to jump to a specific location in the level. You can enter positional coordinates or use the spinners to specify values. If you click the **Go To** button, you immediately move the viewport to the specified coordinate.

Mute Audio

The **Mute Audio** button (10) mutes audio and all sounds in the level.

VR Preview

The **VR Preview** button (11) previews your game project in [virtual reality mode \(p. 1316\)](#) when a [virtual reality \(p. 1305\)](#) gem is enabled.

Using Shortcut Keys

Lumberyard supports the following keyboard shortcut keys.

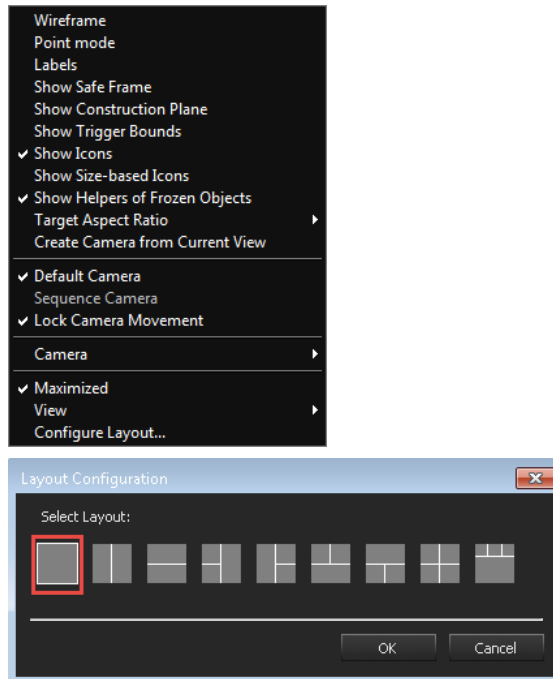
Shortcut key	Function
W	Move forward in the viewport
A	Move backward in the viewport
S	Move left in the viewport
D	Move right in the viewport
F	Freeze the selected object
G	Toggle snap-to-grid
H	Hide the selected object
M	Open the Material Editor
Q	Toggle camera or terrain collision
Z	Go to the selected object
F3	Toggle the wireframe view
Alt+middle mouse button	Rotate around the selected object
Ctrl+D	Duplicate the selected object
Ctrl+E	Export the level
Ctrl+F	Unfreeze all objects
Ctrl+G	Enter game mode (Esc to exit)
Ctrl+H	Unhide all hidden objects
Ctrl+O	Open a level
Ctrl+P	Enable AI or physics
Ctrl+S	Save the level

Shortcut key	Function
Ctrl+Z	Undo the last operation
Ctrl+Shift+Z	Redo the last operation
Ctrl+Alt+F	Restore the saved state
Ctrl+Alt+H	Save the current state
Ctrl+Shift+Space	Lock the selection
Ctrl+Tab	Cycle the viewport perspective
Ctrl+Shift+L	Load objects from the game directory
Ctrl+Shift+S	Save the selected object
Ctrl+F1 (or F2, F3, F4, F5, F6, F7, F8, F9, F10, F11, F12)	Save the viewport location
Shift+F1 (or F2, F3, F4, F5, F6, F7, F8, F9, F10, F11, F12)	Move to the saved viewport location
1	Select the object
2	Select and move the object
3	Select and rotate the object
4	Select and scale the object
5	Select the terrain area
Ctrl+1	Follow the terrain
Ctrl+2	Lock on the XY plane
Ctrl+3	Lock on the X axis
Ctrl+4	Lock on the Y axis
Ctrl+5	Lock on the Z axis
~	Open the console window
[Increase the brush radius size
]	Decrease the brush radius size
Shift+[Decrease the hardness shape of the fall-off curve between the inner and outer radius of the brush
Shift+]	Increase the hardness shape of the fall-off curve between the inner and outer radius of the brush
Shift+Spacebar	Show or hide helpers

Using the Viewport

The viewport window (called **Perspective** in Lumberyard Editor) displays the scene that is rendered by the engine. The viewport is where the majority of level design occurs, such as object placement, terrain editing, in-editor play testing, and asset creation and interaction. You can also use dynamic and flexible tools to understand the 3D relationships among objects in a level.

You can split the viewport into several sub-viewport to customize the layout and set viewing options. Right-click **Perspective** to access the context menu options available:



To the right of the Perspective header is a search field, field of view and screen ratio information, and options to show or hide debug information.



At the bottom of the viewport window is a navigation bar that you can use to input xyz coordinates and camera speed and toggle AI and physics.



Changing the View

You can change the perspective of the viewport by selecting **View, Open View Pane, Top/Front/Left/Perspective/Map**.

- **Top view** – Shows a top-down view of your level, including bounding boxes and line-based helpers. Terrain geometry is not shown.
- **Front view** – Shows a front view of your level, including bounding boxes and line-based helpers. Terrain geometry is not shown.
- **Left view** – Shows a view of your level from the left side, including bounding boxes and line-based helpers. Terrain geometry is not shown.
- **Perspective view** – Shows a view of your level using the default camera perspective, showing all level content. This is the most commonly used view.

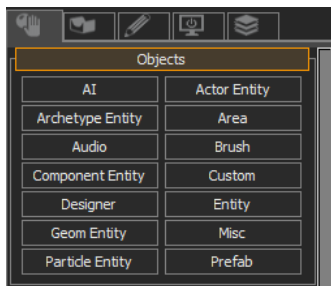
- **Map view** – Shows an overhead map of the level, including helper, terrain, and texture information.

Using the Rollup Bar

The **Rollup Bar** is a separate panel within Lumberyard Editor that provides display options; profile tools; layer controls; and object creation, vegetation, terrain modifying, and solid modeling tools. There are five tabs in the Rollup Bar.

Objects Tab

The **Objects** tab includes the majority of objects and entities, and the interfaces to various local object and brush databases.

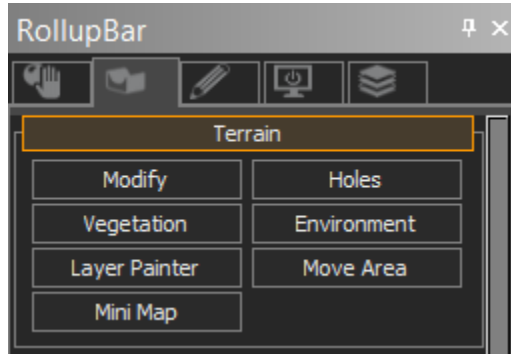


The following buttons are on this tab:

- AI
- Actor Entity
- Archetype Entity
- Area
- Audio
- Brush
- Custom
- Designer
- Entity
- Geom Entity
- Misc
- Particle Entity
- Prefab

Terrain Tab

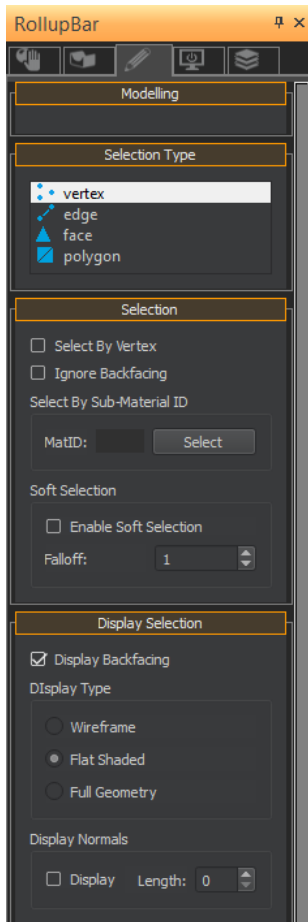
The **Terrain** tab includes the tools to modify terrain and vegetation.



The following buttons are on this tab:

- Modify
- Holes
- Vegetation
- Environment
- Layer Painter
- Move Area
- Mini Map

Modeling Tab

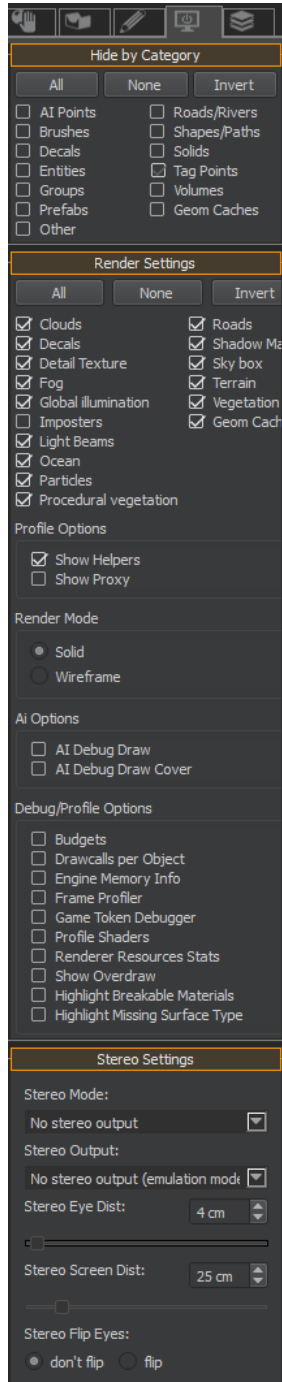


The following buttons are on this tab:

- Selection Type
- Selection
- Select by Sub-Material ID
- Soft Selection
- Display Selection
- Display Type
- Display Normals

Render/Debug Tab

The **Render/Debug** tab includes access rendering, display, and debug options. With the exception of **Hide Helpers**, **Virtual Memory Info**, and **Renderer Resources Stats**, you can also use a console variable to access these options from the console window.



The following sections and subpanels are on this tab:

- Hide by Category
- Render Settings
- Profile Options
- Render Mode
- Ai Options
- Debug/Profile Options

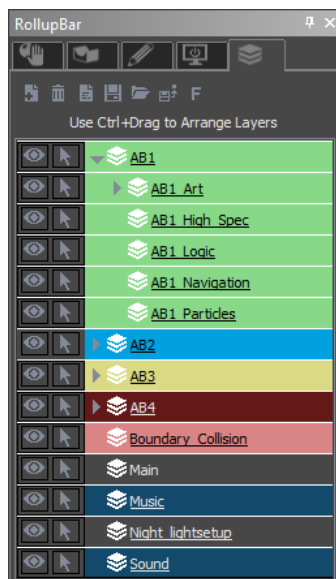
- Stereo Settings

Note

In the **Render Settings** section, the **Shadow Maps** setting must first be enabled/disabled before you can enable/disable the **Global Illumination** setting. These two settings work together.

Layers Tab

The **Layers** tab includes the tools to create and manage level layers.

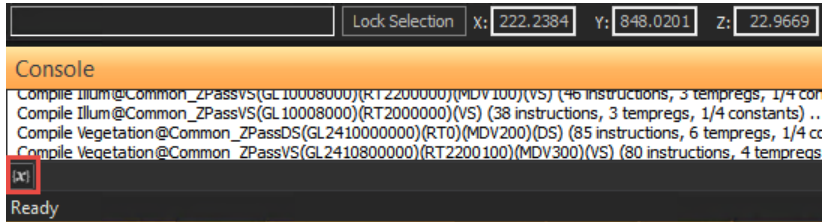


The following buttons are on this tab:

- New Layer
- Delete Layer
- Layer Settings
- Export Layer
- Import Layers
- Save All Modified External Layers
- Freeze Read-Only External Layers
- Hide/Show Layers (eye icon)
- Freeze/Unfreeze Layers (arrow icon)

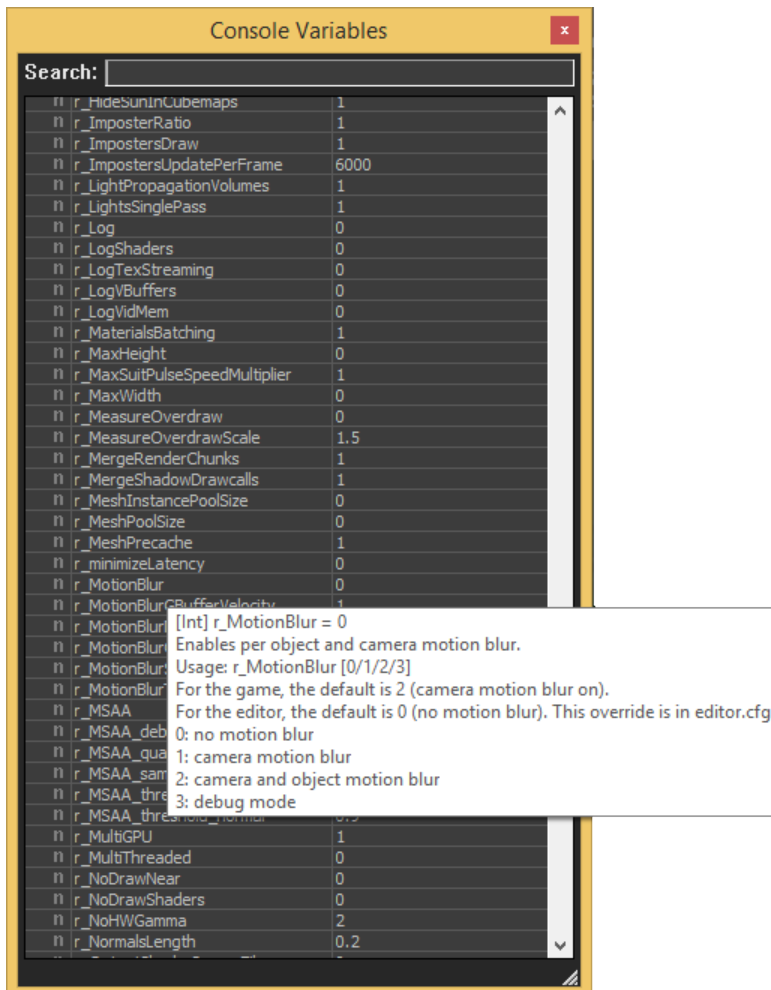
Using the Console Window

In Lumberyard Editor the console window displays a running list of all editor commands, processes, and output. Click the icon at the bottom left corner of the window to display a dialog box of all available console variables.



You can enter console commands in the **Console Variables** dialog box or in the text field to the right of the icon in the console window. For tips and instructions, hover over the console variable.

To export a list of all console variables, use the `DumpCommandsVars` console command. For more information about the console, see [CryConsole](#) in the Lumberyard Developer Guide.



Configuring Console Variables

Console variables (CVARs) are a type of variable that you can manipulate in Lumberyard's console interface.

CVARs can also be set in code, flow graphs, or specified in configuration files. CVARs are executed in the following order:

- Configuration files:
 - The `game.cfg` file in your project folder
 - The `lumberyardroot\dev\system_gamesystem.cfg` file for your game system
 - The `lumberyardroot\dev\engine\config\user.cfg` file
 - The `level.cfg` file in your project's level folder
- Code
- Flow graphs
- Console variables typed directly into the [console \(p. 56\)](#)

The order of execution is also the override order. That is, for example, CVARs set in flow graphs override any identical CVARs set in code, but CVARs set in code override those set in configuration files (and `level.cfg` overrides `user.cfg`, and so on). Finally, any CVARs typed directly into the console override all the other CVAR settings.

Customizing Your Workspace

You can customize the size setting for the toolbar icon (`ed_toolbarIconSize`) in the `Editor.cfg` file. By default, the toolbar icon size is set to 0 (32 pixels).

Docking Windows and Toolbars

Docking helpers automatically appear when you drag a window over another window or Lumberyard Editor itself.

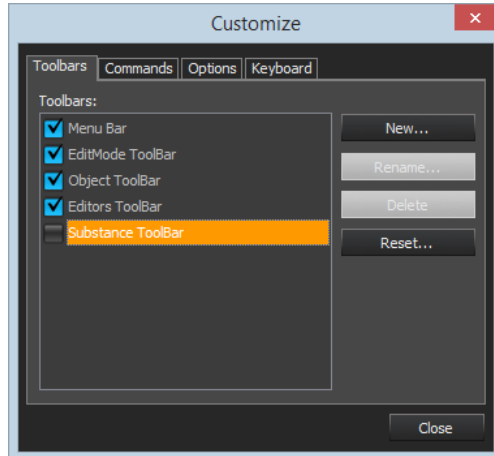
To dock a window to the left of the viewport, drag the window by its title bar and move it to the left of the main view window, onto the docking button. To undock a window, drag the title bar and move the selection window away. Avoid the docking buttons to prevent from accidentally redocking the window.

Customizing Toolbars and Menus

You can use the **Customize** dialog box to customize preset toolbars and create custom user toolbars and menus. To do so, right-click anywhere on the main toolbar and select **Customize**. You can create, rename, and delete any custom toolbars and menus, as well as reset them to their original settings. There are four tabs available.

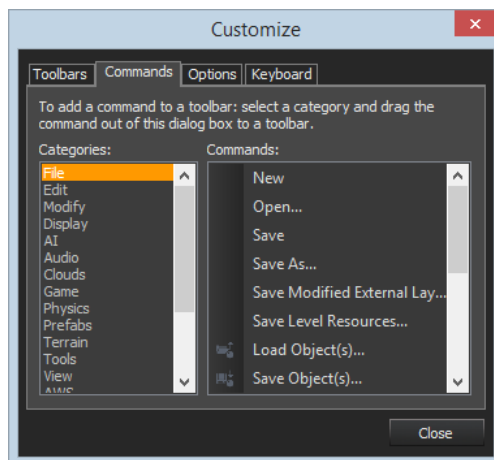
Customizing Toolbars

Docking helpers automatically appear when you drag a window over another window or Lumberyard Editor itself.



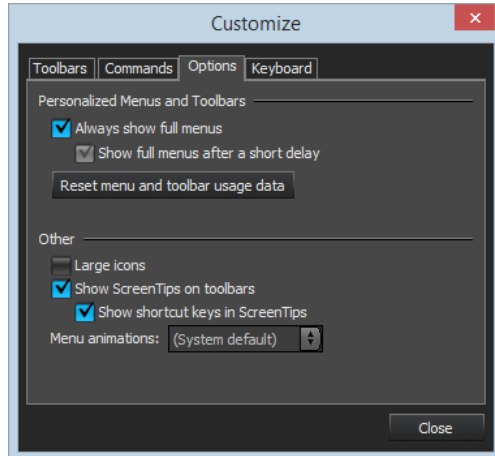
Commands Tab

You can use the **Commands** tab to drag and drop menu commands to any menu category.



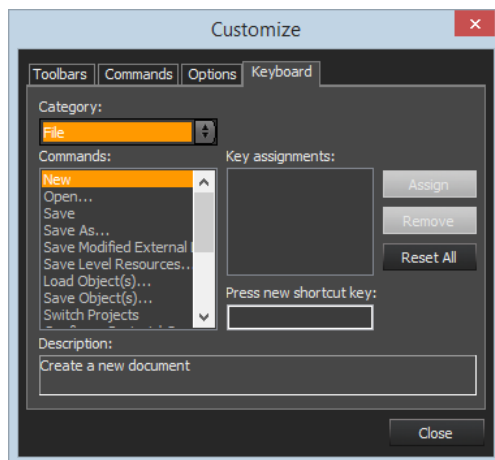
Options Tab

You can use the **Options** tab to set whether full menus are displayed immediately or after a short time delay. You can also customize the size of the menu icons, whether to display tool tips, and whether to employ animated effects in the menus.



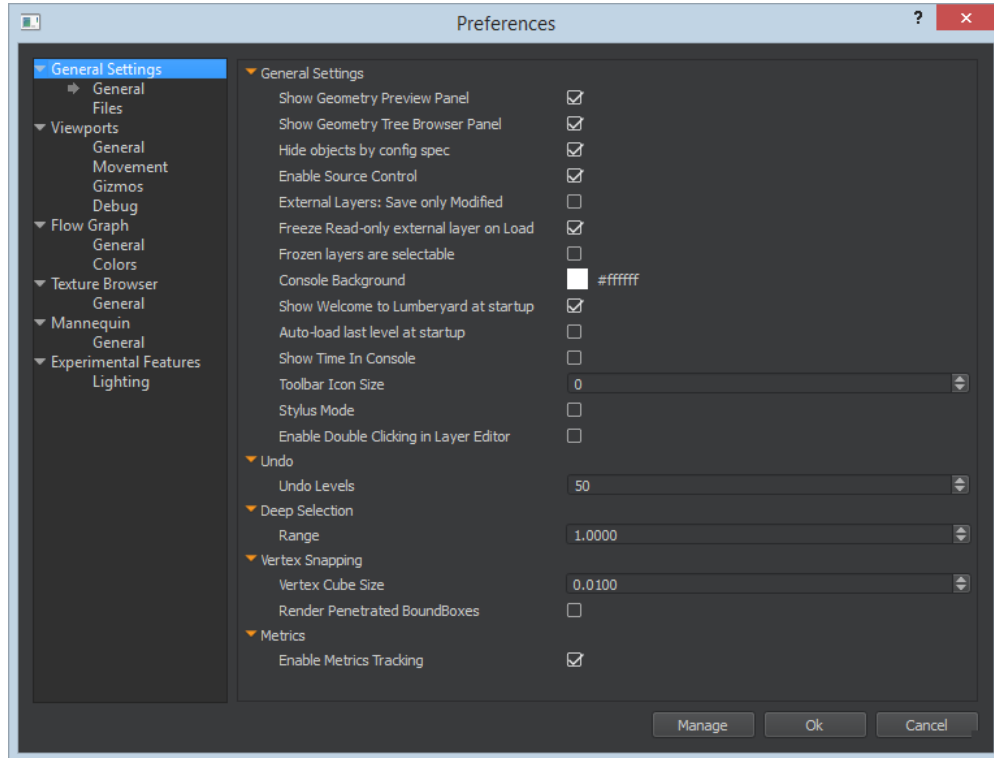
Keyboard Tab

You can use the **Keyboard** tab to select which categories to display in the main menu.



Changing Preferences

You can change the default preferences to customize the look and functionality of Lumberyard Editor. Open the **Preferences** window by selecting **File, Global Preferences, Editor Settings**.



General Settings

You can change the general Lumberyard Editor settings and file settings.

General Settings

Parameter	Description
Show Geometry Preview Panel	Display a preview window for the selected object
Show Geometry Tree Browser Panel	Display the geometry tree browser panel
Hide Objects by Config Spec	Hide objects as determined by the minimal specifications and configuration specifications
Enable Source Control	Enable Perforce version control
External Layers: Save Only Modified	Save only the modified external layers
Freeze Read-Only External Layer on Load	Freeze the read-only external layers when loading the level
Frozen Layers are Selectable	Allow objects in frozen layers to be selected
Console Background	Change the background color for the console
Show Welcome to Lumberyard Editor at Startup	Display the Welcome to Lumberyard Editor dialog box at startup
Autoload Last Level at Startup	Load the level that was last loaded

Parameter	Description
Show Time in Console	Display the time in the console window
Toolbar Icon Size	Adjust the toolbar icon size; default = 0 (32 pixels)
Stylus Mode	Enable stylus mode for tablets and other pointing devices
Enable Double-Clicking in Layer Editor	Allow double-clicking in the Layer Editor
Undo Levels	Specify the maximum number of times you can undo a level; default = 50
Range	Adjust the distance from the cursor to include objects in Deep Selection; default = 1
Vertex Cube Size	Adjust the vertex cube size
Render Penetrated BoundingBoxes	Render penetrated boundingboxes

File Settings

Parameter	Description
Backup on Save	Create a backup file when you save
Maximum Save Backups	Specify the maximum number of saved backups
Standard Temporary Directory	Specify the location of the default temporary directory to use; default = [root]\Temp
Autosave Camera Tag Points	Save the modified camera tag points
Scripts Editor	Specify the text editor to use for scripts
Shaders Editor	Specify the text editor to use for shaders
BSpace Editor	Specify the text editor to use for bspaces
Texture Editor	Specify the program to use for textures
Animation Editor	Specify the program to use for animations
Enable	Enable autobackup
Time Interval	Specify the frequency of autobackup (in minutes)
Maximum Backups	Specify the maximum number of autobackups
Remind Time	Specify the frequency of autobackup reminders (in minutes)

Viewport

You can change the default settings for the viewport.

General Settings

Parameter	Description
Synchronize 2D Viewports	Enable synchronization of 2D viewports to move and correspond with each other
Perspective View FOV	Specify the field of vision for the viewport
Perspective View Aspect Ratio	Specify the length of the aspect ratio for the viewport, where height = 1
Enable Right-Click Context Menu	Enable or disable the context menu that displays by right-clicking in the viewport
Show 4:3 Aspect Ratio Frame	Display a 4:3 aspect ratio frame to show what is visible in game mode
Highlight Selected Geometry	Highlight the selected geometry
Highlight Selected Vegetation	Highlight the selected vegetation
Highlight Geometry on Mouse Over	Highlight geometry on hover over
Hide Cursor when Captured	Show or hide the mouse cursor in the viewport
Drag Square Size	Specify the size of the drag square to prevent from accidentally moving objects when selecting
Display Object Links	Display entity links in the viewport
Display Animation Tracks	Display the animation path for any objects in track view; one line = one frame
Always Show Radii	Display the area of effect (radius) for certain entities
Always Show Prefab Bounds	Display the prefab boundary helpers
Always Show Prefab Objects	Display the prefab object helpers
Show Bounding Boxes	Display a boundary box around each object
Always Draw Entity Labels	Display entity names
Always Show Trigger Bounds	Display the trigger boundary helpers
Show Object Icons	Display object icons
Scale Object Icons with Distance	Scale object icons relative to distance
Show Helpers of Frozen Objects	Display the frozen object helper icons
Fill Selected Shapes	Highlight the inside area of a selected shape
Show Snapping Grid Guide	Display the grid in the viewport
Display Dimension Figures	Display the measurement dimensions of selected assets; you must enable helpers
Swap X/Y Axis	Reverse the x-axis and y-axis

Parameter	Description
Map Texture Resolution	Specify the resolution for the displayed map
Enabled	Display object names
Distance	Specify the visibility distance for text labels
Prefab Bounding Box	Specify the color for the prefab bounding box
Group Bounding Box	Specify the color for the group bounding box
Entity Bounding Box	Specify the color for the entity bounding box
Bounding Box Highlight Alpha	Specify the amount of alpha to add to the bounding box
Geometry Color	Specify the geometry color
Solid Brush Geometry Color	Specify the color of the solid brush geometry
Geometry Highlight Alpha	Specify the amount of alpha to add to the geometry
Child Geometry Highlight Alpha	Specify the amount of alpha to add to the child geometry

Movement Settings

Parameter	Description
Camera Movement Speed	Specify the speed of all movements in the viewport
Camera Rotation Speed	Specify the speed of the mouse while controlling the viewport camera
Fast Movement Scale	Specify the multiplier for the camera speed; e.g. a value of 2 doubles the movement speed of the camera
Wheel Zoom Speed	Specify the speed of the camera zoom when using the mouse wheel

Gizmo Settings

Parameter	Description
Size	Specify the size of the xyz-axes gizmo
Text Labels	Display the xyz-axes labels
Max Count	Specify the maximum number of xyz-axes gizmos that can display onscreen at one time
Helpers Scale	Specify the size of onscreen helpers, including AIAnchors, Tagpoints, and CoverSurfaces
Tagpoint Scale Multiplier	Specify the scale of the Tagpoint helper sphere and the base helper scale value
Ruler Sphere Scale	Specify the scale of the locator sphere size when using the Ruler tool

Parameter	Description
Ruler Sphere Transparency	Specify the transparency level of the locator sphere when using the Ruler tool

Debug Settings

Parameter	Description
Show Mesh Statistic	Display the level of detail information, such as tris and verts, for selectable objects
Warning Icons Draw Distance	Specify the distance to which to display warning icons in the viewport
Show Scale Warnings	Display an icon and warning text for objects that have been scaled
Show Rotation Warnings	Display an icon and warning text for objects that have been scaled

Flow Graph

You can change the default settings for Flow Graph.

General Settings

Parameter	Description
Automatic Migration	Update and reconnect port connection changes
Show NodeIDs	Display an ID for each node
Show Tooltip	Display a tooltip for each node on hover over
Edges on Top of Nodes	Enable edges on top of nodes
Highlight Edges of Selected Nodes	Highlight the incoming and outgoing edges for the selected nodes

Color Settings

Specify the colors to use for the following elements in the Flow Graph Editor:

- Arrows
- Highlight for the in and out arrows
- Highlight for the port edges
- Node outlines
- Node backgrounds
- Backgrounds for custom nodes
- Selected nodes
- Title text
- Text
- Backgrounds
- Grids

- Breakpoints
- Entity ports
- Quick search backgrounds and text
- Debug node backgrounds and titles

Mannequin

You can change the default settings for the Mannequin system.

General Settings

Parameter	Description
Default Preview File	Specify the preview file; the default file is <code>\Animations\Mannequin\Preview\playerPreview1P.xml</code>
Size of Tracks	Specify the height of the tracks for the dope sheet; minimum = 14, maximum = 32
Ctrl to Snap Scrubbing	Snap scrubbing by holding the Ctrl key
Timeline Wheel Zoom Speed	Specify the speed of the mouse wheel when zooming on the Mannequin timeline

Restoring Default Settings for Lumberyard Editor

If you have customized your workspace, you can reset the settings in Lumberyard Editor to the default settings at any time. To do so, select **View, Layouts, Restore Default Layout**.

If you require more granular control to restore Lumberyard Editor settings, you can delete the relevant keys in the Windows registry.

Important

Exercise caution when editing the Windows registry. Not following the instructions carefully may result in a corrupt Windows installation.

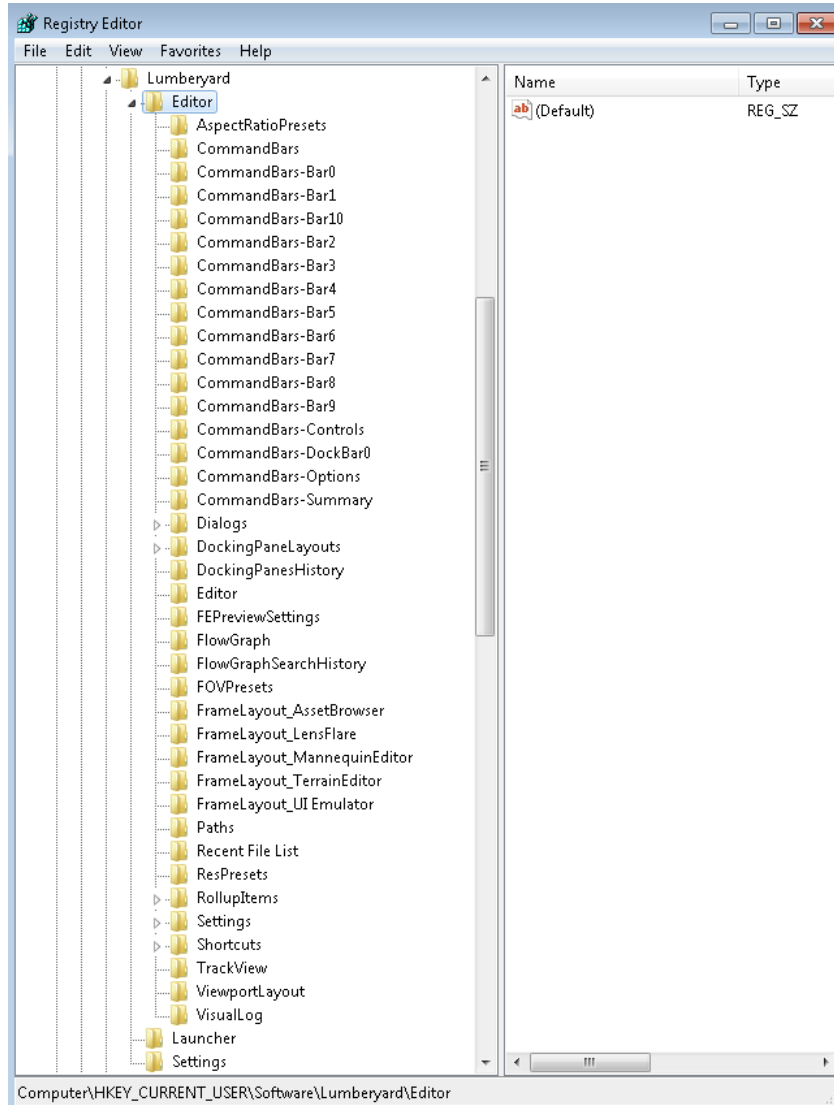
To edit the Windows registry

1. On your Windows desktop, click **Start** and type `regedit` in the search box.
2. In the **Registry Editor**, navigate to `HKEY_CURRENT_USER\Software\Amazon\Lumberyard\Editor`.
3. Right-click the applicable folder(s) and select **Delete**.

The default settings are restored the next time you start Lumberyard Editor.

Lumberyard User Guide

Restoring Default Settings for Lumberyard Editor

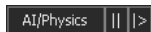


AI System

In the context of a game, AI refers to the technology or system used to endow seemingly-intelligent actions and behaviors to an agent or character, called the AI agent.

Specifically, an AI agent is a game entity that can use information to make decisions in pursuit of one or more goals. An AI agent can perceive its surroundings, navigate through its environment, interact with other objects, communicate with other agents or players, and exhibit a vast number of various actions and behaviors toward the end goal. Sophisticated AI behaviors can be triggered, event-driven or be scripted.

The selection strip at the bottom of Lumberyard Editor features controls to enable AI. The **AI/Physics** button turns AI simulation on and off, and allows you to test AI agent behavior directly without entering game mode.



The pause and next step buttons are used for stepping through the AI system one frame at a time for debugging. To use these correctly, first click the pause button, then click the **AI/Physics** button, then click the next step button.

Make sure to disable the pause button again to return to normal operation.

For information on AI entities, see [AI Control Objects \(p. 439\)](#).

Topics

- [Spawning AI Agents \(p. 68\)](#)
- [AI Navigation \(p. 70\)](#)
- [Agent Perception \(p. 77\)](#)
- [AI Communications \(p. 80\)](#)
- [AI Modular Behavior Tree \(p. 85\)](#)
- [AI Agent Debugging \(p. 115\)](#)

Spawning AI Agents

This section discusses how to spawn, activate, and deactivate one or more AI agents in your level.

Topics

- [Using Flow Graph to Spawn AI Agents \(p. 69\)](#)
- [Using Auto Disable for Agents \(p. 69\)](#)
- [Debugging Agent Spawning Issues \(p. 70\)](#)

Using Flow Graph to Spawn AI Agents

You can use the following AI flow graph nodes to spawn AI agents. An archetype entity is based on a regular entity and specifies individual parameter values for that entity. If the value of an archetype entity parameter is changed, all instances of that archetype entity in the level are updated automatically.

- **Entity:SpawnArchetype**
- **Entity:Spawn**

To access Flow Graph nodes

1. In Rollup Bar , click **AI, TagPoint** and enter a name.
2. In the current level, click to place the tag point.
3. Right-click the tag point, click **Create Flow Graph**, and enter a name.
4. In Lumberyard Editor, click **View, Open View Pane, Flow Graph**.
5. In **Flow Graph**, under **Flow Graphs**, select the new flow graph just created.
6. Right-click anywhere in the graph, click **Add Node**, and then click to create the following nodes:
 - a. **Game:Start**
 - b. **Entity:SpawnArchetype**
 - c. **Entity:EntityPos**
7. Drag node outputs to node inputs to create the following links:
 - a. **Game:Start** output links to **Entity:SpawnArchetype Spawn**
 - b. **Entity:EntityPos** pos links to **Entity:SpawnArchetype Pos**
 - c. **Entity:EntityPos** rotate links to **Entity:SpawnArchetype Rotate**
 - d. **Entity:EntityPos** scale links to **Entity:SpawnArchetype Scale**
8. For **Entity:SpawnArchetype**, click **Archetype** and make a selection from the menu.
9. For **Entity:EntityPos**, right-click **Choose Entity**, click **Assign graph, entity, <Graph Entity>**, and enter the name of the tag point created in step 6.

Using Auto Disable for Agents

You can save processor time by not updating distant AI agents. To control this on a global and on a per-agent basis, enable the **AutoDisable** property.

To enable AutoDisable using Rollup Bar

1. In Rollup Bar, on the **Objects** tab, click **Entity** and select your asset.
2. Under **Entity Properties2**, select the **AutoDisable** check box.

To enable AutoDisable using Flow Graph

1. In Lumberyard Editor, click **View, Open View Pane, Flow Graph**.
2. Under **Flow Graphs**, select your asset.

3. Right-click anywhere in the graph and then click **Add Node, AI, AutoDisable**.
4. In the **AI:AutoDisable** node, click **ON** to enable **AutoDisable**.

Note

You can also enable **AutoDisable** by setting the console variable **ai_UpdateAllAlways** value to 0.

Debugging Agent Spawning Issues

You can use the following console variables to debug AI entity pool issues:

Unless otherwise noted, variable type is Boolean and default value is 0.

- **ai_StatsDisplayMode 1** - Useful to check the number of currently active AI agents.
- **es_DebugPool 1** - Used to debug entity pools.
- **es_DebugPoolFilter** – Enter the name of the entity pool as the value.

The following represents different information that is available from debug output:

- **Bookmarked Entities** - Number of entities marked as being created through pools. It should be greater than 0 if you have AI agents in your level that should be marked.
- **Pool Name** - The entity pool whose information is about to follow. The name should be assigned to the **es_DebugPoolFilter** variable to get more information about the pool. The color highlight on this text means the following:
 - White means the pool has no issues.
 - Yellow means the pool has reached maximum capacity at some point during the level. It is a warning -the pool is still being used correctly, but at the maximum.
 - Red means the pool reached its maximum capacity and another entity was trying to be prepared from the pool, but failed.
- **Not In Use** - The current number of slots in the entity pool that are not in use.
- **In Use** - The current number of slots in the entity pool that are currently being used. Below this output, the AI that is currently prepared from the pool and exists is shown, followed by their EntityId.
- **Pool Definitions** - The entity classes that exist in the pool. Max count (size of the pool) is displayed on the first line. The color highlight of the class name displays information about how that class has been used with the pool, as follows:
 - White means no entities have been prepared from the pool yet of that class type.
 - Green means at least one entity has been prepared and all so far have been prepared successfully.
 - Red means at least one entity has been prepared but it failed when being prepared.

AI Navigation

Lumberyard has a robust set of tools and methods for moving AI agents around – from simple point-to-point navigation to complex sets of scripted navigation behaviors.

AI agents come in different sizes and with different physical properties that impact how they navigate through a game level. AI agent types that can navigate include animate entities such as humans and aliens, and vehicles such as cars, boats, and aircraft.

Each AI has its own navigation mesh that defines the 3D volume where it can move around in. This navigation mesh is called the Multi-Layer Navigation Mesh (MNM), and is comprised of 3D navigation areas, exclusion areas where it cannot move in, and navigation seed points.

You define where and how an AI agent moves around in the navigation mesh using Flow Graph logic. Flow Graph allows you to quickly create complex scripted movements and animations for AI agents as they navigate throughout the area.

AI agents can also move along defined paths between navigation meshes - this is called off-mesh navigation.

Topics

- [Multi-Layer Navigation Mesh \(MNM\) \(p. 71\)](#)
- [Creating Navigation Areas \(p. 71\)](#)
- [Selecting an AI Navigation Type \(p. 72\)](#)
- [Setting Navigation Exclusion Areas \(p. 72\)](#)
- [Adding Navigation Seed Points \(p. 73\)](#)
- [Using Flow Graph for AI Navigation \(p. 73\)](#)
- [Regenerating the Navigation Mesh \(p. 73\)](#)
- [Off-Mesh AI Navigation \(p. 74\)](#)
- [Tutorial: Basic AI Navigation \(p. 75\)](#)
- [Debugging AI Navigation \(p. 76\)](#)

Multi-Layer Navigation Mesh (MNM)

An MNM mesh is automatically created for each navigation area that is added to a level. During the mesh generation process, the terrain, voxels, static objects, and rigid bodies with zero mass are all accounted for in determining whether an AI agent can move through or must move around something.

When a navigation mesh is created, the navigation areas are split in small volumes called tiles, which have a fixed size of 8m x 8m x 8m. Tiles in turns consist of voxels. The smaller the voxel size, the more accurate (and more expensive) the generated mesh.

AI Pathfinding

Lumberyard uses the A* algorithm for pathfinding to search all the triangles of the navigation mesh, with the distance to the destination as the heuristic. The smaller the mesh, the faster the search.

The pathfinding algorithm is asynchronously time-sliced in that requests for paths are not processed immediately but are added to the queue, so it can take a few frames to get the result.

AI agents must stay within the navigation mesh to be able to follow a path defined by the pathfinding algorithm. If an agent gets to the boundary of the mesh, it tries to find the closest triangle within a certain range.

Creating Navigation Areas

For a navigation mesh to be generated, a navigation area needs to be first added to your level. The bottom plane of the navigation area must be underneath the lowest point of the terrain the AI traverses, and the top plane of the navigation area must be above the height of the AI agent placed at the highest point of the terrain, allowing for plenty of clearance. If this is not done, the navigation mesh fails. A successfully created mesh will be blue in color.

Note

The `ai_DebugDrawNavigation` console variable must be set to 1, 2, or 3 in order that the navigable surface is displayed.

To create a Navigation Area

1. In Lumberyard Editor, click **AI, Create New Navigation Area**.

2. In the Rollup Bar, under **NavigationArea**, edit the **Area** parameter to be a non-zero value.
3. Under **NavigationArea**, edit the **Height** parameter so that the area is tall enough to enclose any hills or valleys in the terrain, as needed.
4. Click **AI, Show Navigation Areas**.
5. In the level, drag and click to define a shape enclosing the area that the AI agent navigates through.
6. Double-click to complete the shape.

To edit a Navigation Area

1. In your level, hover over the where you want to make a change. Once the shape turns orange, click it.
2. In Rollup Bar, under **AI, NavigationArea, Edit Options**, click **Edit Shape**.
3. To create a new vertex in the navigation area, press **Ctrl** and click on a line in the area.
4. To delete a portion of the navigation area, double-click on a vertex in the area.

Selecting an AI Navigation Type

Each AI agent needs to have a navigation type assigned, either animate (human-based) or inanimate (vehicle-based). The following AI agent properties are relevant from a navigation perspective:

- **AgentType** - MediumSizedCharacters or VehicleMedium
- **voxelSize** - 0.125m x 0.125m x 0.125m minimum
- **radius** - agent radius, in voxels
- **climbableHeight** - maximum climbable height of maximum slope, in voxels
- **maxWaterHeight** - maximum walkable water depth, in voxels

To assign a navigation type for an AI agent

1. In Lumberyard Editor, click **View, Open View Pane, DataBase View**.
2. On the **Entity Library** tab, click the **Load Library** button and select your asset file.
3. Under **Class Properties** pane, for **Navigation Type**, make a selection. This sets the navigation type for all AI agents.
4. In Rollup Bar, under **Objects, AI, NavigationArea, NavigationArea Params**, make a selection.

Setting Navigation Exclusion Areas

If you don't want an AI agent to navigate through certain areas, you can set exclusion areas within the navigation mesh, as follows:

Exclusion areas are colored red. Besides exclusion areas, AI agents cannot navigate through walls, objects, and the terrain itself.

To set a navigation exclusion area

1. In your level, select a navigation area.
2. In Rollup Bar, click **AI, Navigation Area**.
3. Under **NavigationArea Params**, select the **Exclusion** check box.
4. In your level, click to position the desired exclusion area.

5. Double-click to complete defining the exclusion area shape.

Adding Navigation Seed Points

Navigation seed points are specific accessible locations within navigation meshes that are normally inaccessible due to terrain or other obstructions. Seed points notify the Lumberyard pathfinding system that these locations are accessible for AI agent navigation. For example, an AI agent located on an island could “teleport” to a seed point on an adjacent mountainous island.

To add a navigation seed point

1. In Lumberyard Editor, click **AI, Add Navigation Seed**.
2. In your level, click to position the seed.

Navigation seed point are represented by a seed icon. Areas of the mesh that are accessible by AI agents from navigation seed points are displayed in blue, all other areas are in red. You can use the console variable `ai_MNMCalculateAccessibility` to calculate accessibility.

Using Flow Graph for AI Navigation

Flow graphs are a visual way to define AI navigational logic by creating and linking navigation nodes together. Flow Graph is accessed from Lumberyard Editor by clicking **View, Open View Pane, Flow Graph**.

The navigation-related nodes are:

AISequence:Animation – Moves the AI to a location using a specified animation for the defined Stance, and plays an animation once the target has been reached.

AISequence:ApproachAndEnterVehicle – Moves the AI agent to and then inside a vehicle, using a specified animation for the supplied Stance.

AISequence:Move – Moves the AI to a location using a specified animation for the supplied Stance.

AISequence:MoveAlongPath – Moves the AI along a path indicated by the supplied PathName, using the appropriate animations for the supplied Stance.

Movement:MoveEntityTo – Moves the AI along a path indicated by the supplied PathName, using the appropriate animations for the supplied Stance.

Vehicle:DriveForward – Drives a vehicle forward at a specified time and speed.

Vehicle:FollowPath – Moves the vehicle along a defined off-mesh path at a specified speed.

Vehicle:ChaseTarget – Moves the vehicle along a defined off-mesh path, following a target vehicle and attempting to maintain a line of sight.

AI:RegenerateMNM – Regenerates the mesh at specified minimum and maximum positions. This is useful after the terrain has changed or an object has moved.

Regenerating the Navigation Mesh

There are situations where the navigation mesh must be dynamically updated in real time in order for an AI agent to make sense of its environment. For example, when an object is destroyed the AI agent can now navigate through the space.

Dynamically generating a navigation mesh could also place an AI agent outside of the mesh, leading to stuck or inconsistent behavior.

You can regenerate the entire mesh or a portion of it.

Complete Mesh Regeneration

If you want to regenerate the entire navigation mesh, do the following:

To completely regenerate the navigation mesh

- In Lumberyard Editor, select the mesh and then click **AI, Request a Full MNM rebuild**.

Partial Mesh Regeneration

There are two methods for regenerating a portion of a navigation mesh. Both methods only regenerate the relevant portion of the mesh. By not regenerating the entire mesh, performance is kept high.

The following method is a non-runtime generation of the mesh.

To partially regenerate the navigation mesh

- In Lumberyard Editor, click **AI** and enable **Continuous Update**.

You can also do a runtime partial regeneration of the mesh using the following Flow Graph nodes. Flow Graph is accessed from Lumberyard Editor by clicking **View, Open View Pane, Flow Graph**.

Entity:GetBounds – Obtains the bounding box size, in local or world-space coordinates, for any entity in the mesh. This gives information about the location inside the mesh that requires updating, such as where an object moved to and how big it is.

AI:RegenerateMNM – Specifies the minimum and maximum world-space coordinates of where the navigation mesh regenerates at run-time in response to geometry changes, such as a bridge collapsing or a path becoming blocked, for example.

Off-Mesh AI Navigation

Any AI agent navigation that does not occur inside an MNM mesh is referred to as off-mesh navigation. Off-mesh navigation can be implemented using AI Paths or Smart Objects.

Topics

- [Using AI Paths for Navigation \(p. 74\)](#)
- [Using Smart Objects for AI Navigation \(p. 75\)](#)

Using AI Paths for Navigation

An AI path is a control object that is used to guide an AI agent from point to point along a specified route in a level. AI paths are useful for AI agents that need to traverse between two navigation meshes.

To create an AI Path

1. In Rollup Bar, click **AI, AIPath**.
2. Under **AIPath Params**, set properties and parameter values as needed:

- a. **Road** – Used for **CRoadNavRegion::CreateRoad** and road navigation. Links with other nearby roads for land-based vehicles.
 - b. **ValidatePath** – If enabled, the path displays validation information when selected.
 - c. **Closed** – If true, the path is a loop.
3. Click **File, Export to Lumberyard**. This is a necessary step for the navigation system.

Unless absolutely necessary, AI path navigation should be Uninterruptable, meaning nothing should disrupt or block an AI agent moving along a path.

To set AI Path movement as uninterruptible

1. In Lumberyard Editor, click **View, Open View Pane, Flow Graph**.
2. Under **Graphs, Global, AI actions**, select the AI agent.
3. In the **AISequence:Start** node, clear the **Interruptible** check box.

You can add an AI Path to Flow Graph logic as follows:

To add an AI Path to Flow Graph

1. In Flow Graph, under **Graphs, Level, Entities**, select the applicable flow graph.
2. Right-click anywhere in the graph, and then click **Add Node, AISequence, MoveAlongPath**.
3. In the **AISequence:MoveAlongPath** node, for **PathName**, type the name of the AI Path value from the Rollup Bar.

Using Smart Objects for AI Navigation

Smart Objects are an advanced type of AI Control Object that are used to interact with other objects using rules. Smart Objects can be used for AI movements that would otherwise be impossible to navigate within a mesh. Smart Objects can be used to have AI Agents duck, jump, rappel and kick down doors.

As an example, a Smart Object could be used for an agent running alongside the top wall of a building (first mesh) and then leaping onto a lamp post below (second mesh).

For an AI agent to be able to use a Smart Object, its AgentType definition should list one or more SmartObjectUserClasses.

When using a Smart Object, make sure its flow graph entrance (**AI:SmartObjectHelper Start**) and exit (**AI:SmartObjectHelper End**) helper points are within the two connected navigation meshes. They then automatically connect two meshes together when positioned correctly.

To set AI agent movement using Smart Objects

1. In Rollup Bar, click **AI, SmartObject**.
2. Under **SmartObject Properties**, for **SmartObjectClass**, click the ... button.
3. In Smart Object Classes, select your asset, and then select the desired movements

Tutorial: Basic AI Navigation

This tutorial covers basic AI agent navigation through a level. A tagpoint is used to obtain the destination location within the navigation area.

The position coordinates for the TagPoint are dynamic, meaning you can move the TagPoint around and the AI updates its new destination coordinates accordingly.

To make an AI agent navigate

1. In Rollup Bar, on the Objects tab, click **AI, NavigationArea**.
2. In the level, click to define boundary nodes for the navigation area, then double-click to complete.
3. In Rollup Bar, click **AI, TagPoint**, then click to place it in the level.
4. In Rollup Bar, click **Entity, AI**, select your asset, then click to place it in the level.

Note

Use the legacy GameSDK sample project, which contains the AI assets, to see this folder in the UI. For more information, see [Legacy Sample Project \(GameSDK\) \(p. 1105\)](#).

5. In Rollup Bar, click **Entity, Default, FlowgraphEntity**, then click to place it in the level.
6. In the level, right-click the flow graph entity, click **Create Flow Graph**, and name it.
7. In Flow Graph, under **Flow Graphs**, select the flow graph entity.
8. Right-click anywhere in the graph, click **Add Node**, and create the following nodes:
 - a. **Game:Start**
 - b. **AISequence:Start**
 - c. **AISequence:Move**
 - d. **AISequence:End**
 - e. **Entity:EntityPos**
9. Click and drag to create links between the outputs and inputs of the nodes as follows:
 - a. **Game:Start** output to **AISequence:Start** Start
 - b. **AISequence:Start** Link to **AISequence:Move** Start
 - c. **AISequence:Move** Done to **AISequence:End** End
 - d. **Entity:EntityPos** pos to **AISequence:Move** Position
10. For each of the three **AISequence** nodes, do the following:
 - a. Select the entity in the entity tree to assign it.
 - b. Right-click the top bar of the node.
 - c. click **Assign Selected Entity, Choose Entity**.
 - d. Enter the name of the AI agent selected.
11. For the **Entity:EntityPos** node, do the following:
 - a. Select the tagpoint in the entity tree.
 - b. Right-click the entity top bar of the node.
 - c. Click **Assign Selected Entity, Choose Entity**.
 - d. Enter the name of the tagpoint.
12. Press Ctrl+G to test the AI agent navigation to the tagpoint.

Debugging AI Navigation

In addition to using the AI Debug Recorder and AI Debug Viewer, you can also use specific console variables to debug AI agent navigation issues.

Using Console Variables to Debug AI Navigation

There are a number of console variables that can be used for agent navigation mesh (MNM) debugging. Some statistics display at the top-right corner of the screen.

When debugging Smart Object navigation, make sure that all entities have the right classes assigned, and that the correct actions are set to execute.

ai_DebugDrawNavigation

General variable for AI navigation debugging.

Values: 1 =displays mesh and contour | 2 =also display triangles | 3 =also display tiles and external links

ai_DrawSmartObjects

Displays Smart Objects.

Values: 0 =hide | 1 =show

ai_debugMNMAgentType

Mesh agent type for which debugging information is displayed.

ai_MNMPathFinderQuota

Path finding quota per frame.

Units: seconds

ai_MNMPathFinderDebug

Displays pathfinder debugging statistics, including queue size, average and maximum number of A* search steps, and average and maximum search time.

Values: 0 =hide | 1 =show

ai_MNMPProfileMemory

Displays memory statistics.

Values: 0 =hide | 1 =show

ai_DrawPath

Draw path.

ai_DrawPathFollower

Draw path follower.

Debugging the Navigation Mesh

Use the following procedure as a start to debug the navigation mesh:

To debug the navigation mesh

1. Set the variable **ai_DebugDrawNavigation** value to 3.
2. Create and place a TagPoint with the name **MNMDebugLocator** within a tile of the mesh you want to debug.
3. Press Backspace to switch between the display of the different mesh generation steps.

Agent Perception

AI agents can perceive their environment. Specifically they can see objects in their vicinity, hear sounds, react to collisions, and understand speech.

Topics

- [Using Flow Graph to Set Agent Perception \(p. 78\)](#)
- [Using AI Anchors to Set Agent Perception \(p. 78\)](#)
- [Using Console Variables to Set Agent Perception \(p. 79\)](#)

- [Debugging AI Agent Perception Issues \(p. 80\)](#)

Using Flow Graph to Set Agent Perception

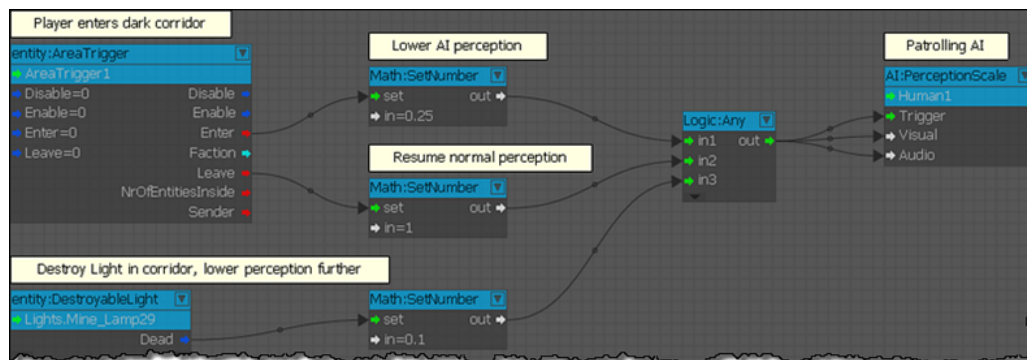
You can use the following AI flow graph nodes to affect agent perception. The perception scaling nodes are important as they control the degree to which AI agents can see or hear their surroundings.

- **AI:AIAwarenessToPlayer** – The degree to which an AI agent is aware of the player's faction. Red is the most aware, while green is the least aware.
- **AI:AIGlobalPerceptionScaling** – The degree of perception (as a percentage) for all AI agents or factions in a level.
- **AI:PerceptionScale** – The degree of perception (as a percentage) for a single AI agent.
- **AI:AlertnessState** – The degree to which any faction or AI agent type is aware of other factions or agent types.
- **AI:GroupAlertness** – Similar to AI:AlertnessState, but by group ID.
- **Input:SpeechRecognition** – Enables or disables Kinect speech recognition on the Xbox One.
- **Input:SpeechRecognitionEnabled** – Queries the availability of Kinect speech recognition on Xbox One.
- **Input:SpeechRecognitionListener** – Listens for a specific Kinect voice command.
- **AI:AttentionTarget** – Gets the target of an AI agent's attention as a position or entityID.

To access Flow Graph perception nodes

1. Open Flow Graph.
2. In the **Flow Graphs** pane, select your asset.
3. Right-click anywhere in the graph, then click **Add Node, AI** (or **Input**) to access the nodes.

You can use additional flow graph logic to fine-tune AI agent control and determine how agent awareness varies with the agent's surroundings, as shown below.



Using AI Anchors to Set Agent Perception

Another way to control AI agent perception is to use AI anchors. Unlike the Flow Graph method, which relies on logic, this method relies on object placement and level markup.

There are four AI anchors for controlling an AI agent's perception: LIGHTSPOT_LIGHT, LIGHTSPOT_MEDIUM, LIGHTSPOT_DARK, and LIGHTSPOT_SUPERDARK. As you might expect, LIGHTSPOT_SUPERDARK gives an agent the least amount of perception.

These settings limit the visibility for the AI agent inside the specified radius. If a player is inside this radius, the agent has a diminished perception of the player.

Note

To reduce demands on performance, use these AI anchors in place of visual light entities or the Sun.

To use AI Anchors for setting perception

1. In **Rollup Bar**, on the **Objects** tab, click **AI, AIAnchor**.
2. Under **AIAnchor Properties**, double-click **AnchorType**.
3. In **AI Anchors**, select one of the four anchors.
4. Select **Enabled**.
5. Click **radius** and enter a value in meters.

Using Console Variables to Set Agent Perception

You can also use console variables (cvars) to affect AI agent perception. Console variables are accessed by clicking the "..." button in the lower right corner of Lumberyard Editor.

Unless otherwise noted, the variable type is Boolean and the default value is 0.

- **ai_IgnorePlayer** – Determines the degree to which the agent ignores players. A setting of 1 is the same as 0% perception scale (agent ignores players).
- **ai_IgnoreBulletRainStimulus** – Determines whether AI agents perceive bullets passing near them.
- **ai_IgnoreVisibilityChecks** – Returns certain visibility checks as false.
- **ai_IgnoreVisualStimulus** – Notifies the Perception Handler to always ignore visual stimulus.
- **ai_IgnoreSoundStimulus** – Determines whether the agent ignores all sounds. Visual and tactile stimuli are not affected.
- **ai_SoundPerception** – Determines the degree to which the agent can hear sounds. A setting of 0 causes the agent to ignore all sounds (useful for debugging purposes when used in conjunction with `ai_DebugDraw`). Default value: 1
- **ai_EnablePerceptionStanceVisibleRange** – Determines the maximum perception range for AI based on the player's stance.
- **ai_CrouchVisibleRange** – Determines the perception range for AI agents when the player is crouching and `ai_EnablePerceptionStanceVisibleRange` is enabled. Default value: 15.0
- **ai_ProneVisibleRange** – Determines the perception range for AI agents when the player is prone and `ai_EnablePerceptionStanceVisibleRange` is enabled. Default value: 6.0

For the next three variables, if the `isAffectedByLight` property is true, this determines the scaling factor for the AI agent's visual perception range under the LIGHTSPOT lighting conditions.

- **ai_SightRangeDarkIllumMod** – Has the same effect as the LIGHTSPOT_DARK anchor type. Default value: 0.5
- **ai_SightRangeMediumIllumMod** – Has the same effect as the LIGHTSPOT_MEDIUM anchor type. Default value: 0.8
- **ai_SightRangeSuperDarkIllumMod** – Has the same effect as the LIGHTSPOT_SUPERDARK anchor type. Default value: 0.25

To set perception properties using Database View

1. In Lumberyard Editor, click **View, Open View Pane, DataBase View**.

2. On the **Entity Library** tab, click **Load Library** to select the applicable entity file.
3. Select the AI entity in the entity tree.
4. In the center pane, under **Perception**, enable properties and set parameter values as needed.

Debugging AI Agent Perception Issues

For debugging specific AI perception issues, use the following console variables. To debug generic AI issues, see [AI Agent Debugging \(p. 115\)](#).

Unless otherwise noted, variable type is Boolean and default value is 0.

- **ai_DebugGlobalPerceptionScale** – Displays global perception scale multipliers.
- **ai_DrawPerceptionIndicators** – Displays indicators showing the enemy's current perception level of player.
- **ai_DrawPerceptionDebugging** - Displays indicators showing how the enemy view intersects with perception modifiers.
- **ai_DrawPerceptionModifiers** - Displays perception modifier areas in game mode.
- **ai_DrawPerceptionHandlerModifiers** – Displays perception handler modifiers on a specific AI agent. Requires an AIName as the parameter.
- **ai_DebugPerceptionManager** – Displays perception manager performance overlay.
- **ai_DebugDrawLightLevel** – Displays the AI light level. Useful for debugging with lightspot anchors.
- **ai_DrawAgentFOV** – Displays the FOV cone for AI agents. Requires ai_DebugDraw to be enabled.
- **ai_DrawAgentStats** – Displays information about agents. Nadlt=name/alertness/distances/light level/target.
- **ai_DrawAttentionTargetPositions** – Displays position markers for the AI agent's current attention target.

AI Communications

AI agents can speak (or make sounds) at various times in the game and send signals to each other to affect their behaviors.

Note

Communications are not played if an AI agent is currently executing a smart object action.

Topics

- [Using Database View to Set AI Communication \(p. 80\)](#)
- [Using AI Communication Channels \(p. 81\)](#)
- [Using the CommConfig Property \(p. 82\)](#)
- [Using GoalPipes to Trigger Communication \(p. 83\)](#)
- [Using Voice Libraries for AI Speech \(p. 83\)](#)
- [Using Flow Graph for Setting AI Communications \(p. 83\)](#)
- [Using AI Signals Among Agents \(p. 83\)](#)

Using Database View to Set AI Communication

There are several communication-related properties and parameters that can also be set using the Database View tool.

To set communication properties using Database View

1. In Lumberyard Editor, click **View, Open View Pane, DataBase View**.
2. On the **Entity Library** tab, click the **Load Library** button to select the applicable entity file.
3. Select the AI entity from the entity tree.
4. In the center pane, under **Class Properties**, enable properties and set parameter values as needed:
 - a. **CommConfig** - select **Basic** or **Human**.
 - b. **Commrange** - enter the communication range as needed.

Using AI Communication Channels

AI communication channels are used to determine whether an AI agent can communicate at a given time, depending on whether the communication channel is occupied or free. Communication channels are XML-based and can be nested and this determines if a parent communication channel is occupied depending on whether a child communication channel is occupied.

A sample configuration file with multiple communication channels is shown below:

```
<!--ChannelConfig.xml-->
<Communications>
  <ChannelConfig>
    <Channel name="Global" minSilence="1.5" flushSilence="0.5"
type="global">
    <Channel name="Group" minSilence="1.5" flushSilence="0.5"
type="group">
      <Channel name="Search" minSilence="6.5" type="group"/>
    >
      <Channel name="Reaction" priority="2" minSilence="2"
flushSilence="0.5" type="group"/>
      <Channel name="Threat" priority="4" minSilence="0.5"
flushSilence="0.5" type="group"/>
    </Channel>
    <Channel name="Personal" priority="1" minSilence="2"
actorMinSilence="3" type="personal"/>
  </ChannelConfig>
</Communications>
```

Where,

- **minSilence** – Minimum time (in seconds) for which the Channel remains occupied after the Communication has completed.
- **flushSilence** – Time (in seconds) for which a Channel remains occupied after flushing the Channel. It is used to override the imposed silence time for the Channel which is no longer playing a Communication. If this attribute is not specified, the value of minSilence is used.
- **actorMinSilence** – Minimum imposed time (in seconds) to restrict AI actors from playing voice libraries after starting a Communication.
- **ignoreActorSilence** – Ignore (AI) actor Communication restrictions from the script.
- **type** – Personal, group, or global.
- **name** – Name of the channel.

- **priority** – Priority level.

Using the CommConfig Property

The `CommConfig` property (see [Using Database View to Set AI Communication \(p. 80\)](#)) determines which communications (and how) an AI agent can play. This property has a value of **Basic** or **Human**, whose properties and attributed are defined by two XML files.

A sample `BasicCommunications.xml` file is shown below:

```
<Communications>
  <!--Animation + Sound Event example (needs state using the action/signal
in the animation graph)-->
  <Config name="Surprise">
    <Communication name="comm_anim" finishMethod="animation" blocking="all"
forceAnimation="1">
      <Variation animationName="Surprise" soundName="sounds/
interface:player:heartbeat" />
    </Communication>
  </Config>

  <!--Sound Event example-->
  <Config name="Welcome">
    <Communication name="comm_welcome" finishMethod="sound" blocking="none">
      <Variation soundName="sounds/dialog:dialog:welcome" />
    </Communication>
  </Config>

  <!--Auto generated Voice Library examples-->
  <Config name="SDK_Example_NPC_01">
    <AutoGenerateCommunication voiceLib="npc_01_example" finishMethod="voice"
blocking="none" />
  </Config>

  <Config name="SDK_Example_NPC_02">
    <AutoGenerateCommunication voiceLib="npc_02_example" finishMethod="voice"
blocking="none" />
  </Config>
</Communications>
```

Where,

- **name:** Basic or Human, as specified by the `CommConfig` property.
- **choiceMethod:** Method used to choose a variation: Random, Sequence, RandomSequence, or Match.
- **responseChoiceMethod:** Method used to choose a variation: Random, Sequence, RandomSequence, or Match.
- **animationName:** Animation graph input value
- **lookAtTarget:** : Valid values are 1|0, true|false, or yes|no. Makes the AI look at the target.
- **finishMethod:** Any or all of: animation, sound, voice, timeout, all. It defines the way to determine when the communication is finished - after the animation is finished, or time interval has elapsed.
- **blocking:** movement, fire, all, none. It allows to disable the movement or firing of the AI.
- **animationType:** signal or action
- **voiceLib:** The name of the Voice Library to extract Communication names from.

Using GoalPipes to Trigger Communication

To trigger a Communication event, use the goalop "communicate" as follows:

```
<GoalPipe name="Cover2_Communicate">  
  <Communicate name="comm_welcome" channel="Search" expiry="0.5"/>  
</GoalPipe>
```

Where,

- **name** is the name of the actual communication (sound or voice). This is defined by the CommConfig property.
- **channel** is the name of the Communication Channel this AI is Using. The channel is defined in the Communication Channel file.
- **expiry** (expiry) is the maximum allowable delay in triggering the communication event when the Communication Channel is temporarily occupied. If the communication event couldn't be triggered within this time period, it is discarded.

Using Voice Libraries for AI Speech

Voice Libraries are XML-based Excel files used to support localized AI agent speech, sub-titles, and lip-syncing.

The specific voice library file is assigned in the Communication Configurations XML file using the `<AutoGenerateCommunication>` element and associated attributes. For more information, see [Using the CommConfig Property \(p. 82\)](#).

For each different AI signal, a specific sound file is used that plays a specific sound or speech snippet. AI agents are assigned a specific voice library file using the **esVoice** property.

Using Flow Graph for Setting AI Communications

There is one AI Flow Graph node used to effect agent communications, as follows:

To access Flow Graph communication nodes

1. In Lumberyard Editor, click **View, Available View Pane, Flow Graph**.
2. In Flow Graph, under **Flow Graphs**, select the applicable AI agent.
3. Right-click anywhere in the graph, then click **Add Node, AI, SetCommunicationVariable**.

Using AI Signals Among Agents

AI signals allow agents to communicate with each other. An AI signal is sent by one AI agent to another AI agent, to a subset of agents, or to itself. You can also specify how AI agents react to received signals.

Sending AI Signals

The method used to send an AI signal is as follows:

```
AI:Signal(signalfilter_, signal_type, *MySignalName*, sender_entity_id);
```

Where,

signalfilter_

Defines which AI agents receive the signal. It can be chosen among a fixed set of symbols that have the prefix SIGNALFILTER_. The list of available signal filters is shown below.

signal_type

Type of signal.

Values: 1 = The entity receiving the signal processes it only if it's enabled and it's not set to ignorant (see AI:MakePuppetIgnorant for details). | 0 = The entity receiving the signal processes it if it's not set to Ignorant. | -1 = The entity receiving the signal processes it unconditionally.

MySignalName

The signal identifier. It can be any non-empty string; for the signal recipient, it must be a function with the same name either in its current behavior, its default behavior, or in the DEFAULT.lua script file in order to react to the received signal.

sender_entity_id

The entity id of the signal recipient. This is usually the ID of the recipient, but can also be the entity ID of the sender if the signal will be sent to the sending agent.

Signalfilter Parameters

Parameter	Description
0	The entity specified with the entity_id parameter.
SIGNALFILTER_LASTOP	The entity's last operation target (if it has one).
SIGNALFILTER_TARGET	The current entity's attention target.
SIGNALFILTER_GROUPONLY	All the entities in the sender's group, i.e. the entities with its same group id, in the sender's communication range.
SIGNALFILTER_SUPERGROUP	All the entities in the sender's group, i.e. the entities with its same group id, in the whole level.
SIGNALFILTER_SPECIESONLY	All the entities of the same sender's species, in the sender's communication range.
SIGNALFILTER_SUPERSPECIES	All the entities of the same sender's species, in the whole level.
SIGNALFILTER_HALFOFGROUP	Half the entities of the sender's group (you cannot specify which entities).
SIGNALFILTER_NEARESTGROUP	The nearest entity to the sender in its group.
SIGNALFILTER_NEARESTINCOMM	The nearest entity to the sender in its group, if in its communication range.
SIGNALFILTER_ANYONEINCOMM	All of the entities in the sender's communication range.
SIGNALID_READIBILITY	This special signal is used to make the entity recipient perform a readability event (sound/animation).

Receiving AI Signals

A signal that is received by an AI agent can cause an agent to change its behavior, as follows:

```

Behavior1 = {
  OnEnemySeen = *Behavior1*,
}

```

```
OnEnemyMemory = *Behavior2*,  
MySignalName = *MyNewBehavior*,  
}
```

For example, if an AI agent is currently in Behavior1 and receives the signal MySignalName, after having executed the callback function above it will then switch its behavior to MyNewBehavior.

The MySignalName function is defined as follows:

```
MySignalName = function(self, entity, sender)
```

Where,

self is the AI agent behavior

entity is the AI agent

sender is the signal sender

This function is actually a callback which, similar to system events, can be defined in the recipient entity's current behavior, the default idle behavior (if it's not present in current behavior), or in the Scripts/AI/Behaviors/Default.lua script file (if not present in the default idle behavior).

Signal behaviors can be inherited, such as when a signal is used to initiate more than one behavior at a time.

AI Modular Behavior Tree

The Modular Behavior Tree (MBT) is a collection of XML-based nodes that describe rules, behaviors, and tasks for AI agents to follow. The Modular Behavior Tree Editor is used to create trees and nodes for AI agents, and is accessed from Lumberyard Editor by clicking **View, Open View Pane, Modular Behavior Tree Editor**.

AI signals are sent either from the MBT itself using the Signal node or from code. A signal sets a tree variable to true or false when it is triggered. Tree variables can then be used to make decisions in the tree. Timestamps are set when an AI signal comes in, and can be used to check how long ago something happened.

An example tree structure is shown here:

```
<BehaviorTree>  
  <Root>  
    <Sequence>  
      <Log message="Test" />  
      <WaitForEvent name="OnEnemySeen" />  
      <Move to="Target" speed="Walk" stance="Stand"  
fireMode="BurstWhileMoving" />  
      <Halt />  
    </Sequence>  
  </Root>  
</BehaviorTree>
```

Each node can have parameters to configure the behavior of its execution. When passing an unacceptable value the parsing of the node could fail and an error message could be found inside the Editor.log or Game.log files.

Topics

- [Standard MBT Nodes \(p. 86\)](#)
- [Common AI MBT Nodes \(p. 90\)](#)
- [Game AI MBT Nodes \(p. 105\)](#)
- [Helicopter AI MBT Nodes \(p. 110\)](#)

Standard MBT Nodes

The following standard Modular Behavior Tree nodes are supported. These nodes can be found at `Code\CryEngine\CryCommon\BehaviorTree\`.

Loop node

The Loop node runs one child multiple times or until the child fails to run. If the count is not specified, it is considered infinite.

Parameters

- **count** : The maximum number of times a child of the Loop node is run

Behavior

- **Success**: If the count is reached for the child
- **Failure**: If the child fails to run

Example

```
<Loop count="3">  
  <SomeChildNode />  
</Loop>
```

LoopUntilSuccess node

The LoopUntilSuccess node runs one child until it succeeds. A maximum number of attempts can be specified. If no maximum number of attempts is specified or if it's set to less than or equal to 0 then the node will attempt to run the child repeatedly until the child succeeds.

Parameters

- **attemptCount**: The maximum amount of possible attempts to make the child succeeding.

Behavior

- **Success**: If the child succeeds
- **Failure**: If the maximum amount of allowed attempts is reached.

Example

```
<LoopUntilSuccess attemptCount="5">  
  <SomeChildNode />  
</LoopUntilSuccess>
```


Parallel node

The Parallel node run its children in parallel. A maximum of 32 children are allowed. If success and failure limits are reached at the same time, the node will succeed.

Parameters

- **failureMode** : The mode used to evaluate when the node fails. Accepted values are any or all. Default value = any.
- **successMode**: The mode used to evaluate when the node succeeds. Accepted values are any or all. Default value = all.

Behavior

- **Success**: If any or all children succeed.
- **Failure**: If any or all children fail.

Example

```
<Parallel successMode="any" failureMode="all">
  <SomeChildNode1 />
  <SomeChildNode2 />
  <SomeChildNode3 />
</Parallel>
```

Selector node

The Selector node runs its children one at a time, stopping at the first one that succeeds.

Parameters

No parameters

Behavior

- **Success**: As soon as one of the children succeed. The remaining children are not run
- **Failure**: If all children fail.

Example

```
<Selector>
  <SomeChildNode1 />
  <SomeChildNode2ToExecuteIfSomeChildNode1Fails />
  <SomeChildNode3ToExecuteIfSomeChildNode2Fails />
</Selector>
```

Sequence node

The Sequence node runs its children one at a time in order. A maximum of 255 children are allowed.

Parameters

No parameters

Behavior

- **Success:** If all children succeed.
- **Failure:** If any children fail

Example

```
<Sequence>
  <SomeChildNode1 />
  <SomeChildNode2 />
  <SomeChildNode3 />
</Sequence>
```

StateMachine node

The StateMachine is a composite node allowed to have one or more children. The children of a StateMachine node must be of the type State.

Only one child at any given time is allowed to be run and the first one defined is the first one to be run.

The current status of a StateMachine node is the same as that of the child that is currently selected to be run.

Parameters

None

Behavior

- **Success:** If the child State node succeeds
- **Failure:** If the child State node fails

Example

```
<StateMachine>
  <State />
  <State name="State1" />
  <State name="State2" />
</StateMachine>
```

StateMachine:State node

The State node is the basic block of a StateMachine node. Each State node must have a BehaviorTree node and may also have a Transitions block.

A State node runs the content of its BehaviorTree node and can transition to another state (or itself) as specified in the Transitions block.

If a State node transitions into itself while running, it will first be terminated, re-initialized, and then updated again.

Parameters

- **name** : The name of the state. It must be unique for the scope of the StateMachine node.

Behavior

- **Success:** If the BehaviorTree node succeeds

- **Failure:** If the BehaviorTree node fails

Example

```
<State name="StateName" >
  <Transitions>
    <Transition onEvent="EventOrTransitionSignalName" to="OtherStateName" />
  </Transitions>
  <BehaviorTree>
    <SomeChildNode />
  </BehaviorTree>
</State>
```

The Transitions tag must have the following parameters:

- **onEvent:** Identifies the string name of the event that could cause the transition to happen
- **to:** Identifies the state name where transitioning to

SuppressFailure node

The SuppressFailure node owns and runs one child. It will succeed irregardless of the result of the child's execution.

Parameters

None

Behavior

- **Success:** As soon as the child finishes.

Example

```
<SuppressFailure>
  <SomeChildThatCanFail />
</SuppressFailure>
```

Timeout node

The Timeout node fails after a certain amount of time has passed.

Parameters

- **duration :** Number of seconds before the failure of the node occurs

Behavior

- **Failure:** if it runs for more than the amount of time specified by the duration parameter

Example

```
<Timeout duration=5" />
```

Wait node

The Wait node succeeds after a certain amount of time has passed.

Parameters

- **duration** : The amount of seconds before the failure of the node occurs
- **variation**: The extra amount of time that will be added on top of the specified duration. This allows random variations between different executions of the node

Behavior

- **Success**: As soon as it runs for more than the amount of time specified by the duration parameter plus the random variation

Example

```
<Wait duration="5" variation="1" />
```

Common AI MBT Nodes

The following common AI Modular Behavior Tree nodes are supported.

AnimateFragment node

This node plays an Mannequin animation fragment and waits until the animation finishes.

Parameters

- **name**: The name of the fragment to play.

Behavior

- **Success**: If the animation is correctly played or if no operation was needed.
- **Failure**: If an error occurs while trying to queue the request to play the specified fragment..

Example

```
<AnimateFragment name="SomeFragmentName" />
```

Bubble node

Used to display a message in a bubble above the agent.

- **message**: The message that should be shown in the speech bubble.
- **duration**: The number of seconds to show the message. Default = 0.
- **balloon**: Shows the message in a balloon about the AI agent. 1 will show the message in a balloon above the agent; 0 will not. Default = 1.
- **log**: Writes the message to a general-purpose log. 1 will write to the log; 0 will not Default = 1.

None.

Behavior

- **Success:** Succeeds immediately after having queued the message to be displayed.

Example

```
<Bubble message="MessageToBeDisplayedAndOrLogged" duration="5.0"  
  balloon="true" log="true" />
```

Move node

Used to move the agent from the current position to the specified destination. If the destination is a target then the end position is updated if not reached while the target moves.

Parameters

- **speed:** Movement speed, which can be any of the following: Walk, Run, or Sprint.
- **stance:** Stance, which can be any of the following: Relaxed, Alerted, or Stand. Default = Stand.
- **bodyOrientation:** Body orientation, which can be any of the following: FullyTowardsMovementDirection, FullyTowardsAimOrLook, or HalfwayTowardsAimOrLook. Default = HalfwayTowardsAimOrLook.
- **moveToCover:** True if the agent is moving into cover; otherwise false. Default = false.
- **turnTowardsMovementDirectionBeforeMoving:** True if the agent should first turn into the direction of movement before actually moving; false if not. Default = false.
- **strafe:** True if the agent is allowed to strafe; false if not. Default = false.
- **glanceInMovementDirection:** True if the agent is allowed to glance in the direction of movement; false if it should always look at its look-at target. Default = false.
- **to:** Movement destination, which can be one of the following: Target, Cover, RefPoint. or LastOp.
 - **Target:** The current attention target.
 - **Cover:** The current cover position.
 - **RefPoint:** The current reference position.
 - **LastOp:** The position of the last successful position related operation.
- **stopWithinDistance:** If within this distance from the target, stop moving. Default = 0.0.
- **stopDistanceVariation:** Additional random stopping distance, Default = 0.0.
- **fireMode:** Fire mode while moving: Default - Off.
 - **Off:** Do not fire.
 - **Burst:** Fire in bursts - living targets only.
 - **Continuous:** Fire continuously - living targets only.
 - **Forced:** Fire continuously - allow any target.
 - **Aim:** Aim target only - allow any target
 - **Secondary:** Fire secondary weapon.
 - **SecondarySmoke:** Fire smoke grenade
 - **Melee:** Melee.
 - **Kill:** No missing, shoot directly at the target, no matter what aggression/attackRange/accuracy is.
 - **BurstWhileMoving:** Fire in bursts, while moving and too far away from the target.
 - **PanicSpread:** Fire randomly in the general direction of the target.
 - **BurstDrawFire:** Fire in bursts, in an attempt to draw enemy fire.
 - **MeleeForced:** Melee, without distance restrictions.
 - **BurstSnipe:** Fire in burst, aiming for a head-shot.

- **AimSweep**: Keep aiming at the target, but not allowed to fire.
- **BurstOnce**: Fire a single burst.
- **avoidDangers**: 1 if dangers should be avoided while moving, 0 if they can be ignored. Default = 1.
- **avoidGroupMates**: 1 if group mates should be avoided while moving, 0 if they can be ignored. Default = 1.
- **considerActorsAsPathObstacles**: 1 if any actor should be considered a path obstacle that the path-finder should avoid, 0 if they can be ignored. Default = 0.
- **lengthToTrimFromThePathEnd**: The resulting path-finder path will be trimmed by the specified amount of distance. Positive values will trim from the end of the path; negative values will trim from the start of the path. Default = 0.0.

Behavior

- **Success**: If the destination is reached.
- **Failure**: If the destination is deemed unreachable.

Example

```
<Move to="DestinationType" stance="StanceName" fireMode="FiremodeName"  
speed="SpeedName" stopWithinDistance="3c " />
```

QueryTPS node

This node performs a Tactical Position System query and waits for a result.

Parameters

- **name**: The name of the TPS query to use.
- **register**: Where to store result of the TPS query: RefPoint or Cover. Default = Cover.

Behavior

- **Success**: If the TPS returns a tactical position.
- **Failure**: If the TPS does not find a tactical position.

Example

```
<QueryTPS name="NameOfTheQuery" register="NameOfTheRegister" />
```

IfTime node

This node executes the child node if the time condition is satisfied.

Parameters

- **since**: Name of the time stamp used for the condition.
- **isMoreThan**: Defines the condition to be a comparison if the value of the time stamp is more than this value (exclusive with the parameter 'isLessThan').
- **isLessThan**: Defines the condition to be a comparison if the value of the time stamp is less than this value (exclusive with the parameter 'isMoreThan').

- **orNeverBeenSet:** (Optional) Changes the behavior of the node in case the time stamp was never set, instead of failing the node will succeed.

Behavior

- **Success:** If orNeverBeenSet is true.
- **Failure:** If the time condition is not satisfied or if the time stamp was not previously set.

Example

```
<IfTime since="FragGrenadeThrownInGroup" isMoreThan="5.0" orNeverBeenSet="1">  
  <ThrowGrenade type="frag" />  
</IfTime>
```

WaitUntilTime node

This node executes the child node if the time condition is satisfied.

Parameters

- **since:** Name of the time stamp used for the condition.
- **isMoreThan:** Defines the condition to be a comparison if the value of the time stamp is more than this value (exclusive with the parameter 'isLessThan').
- **isLessThan:** Defines the condition to be a comparison if the value of the time stamp is less than this value (exclusive with the parameter 'isMoreThan').
- **orNeverBeenSet:** (Optional) Changes the behavior of the node in case the time stamp was never set, instead of failing the node will succeed.

Behavior

- **Success:** The time stamp was not set previously set and the parameter succeedIfNeverBeenSet is true. Otherwise, the node returns the result of the execution of its child node.

Example

```
<WaitUntilTime since="BeingShotAt" isMoreThan="7" />
```

AssertTime node

This node node succeeds if the time condition is satisfied.

Parameters

- **since:** Name of the time stamp used for the condition.
- **isMoreThan:** Defines the condition to be a comparison if the value of the time stamp is more than this value (exclusive with the parameter 'isLessThan').
- **isLessThan:** Defines the condition to be a comparison if the value of the time stamp is less than this value (exclusive with the parameter 'isMoreThan').
- **orNeverBeenSet:** (Optional) Changes the behavior of the node in case the time stamp was never set, instead of failing the node will succeed.

Behavior

- **Success:** If the time condition is true or the orNeverBeenSet parameter is true.
- **Failure:** If the time stamp was not previously set.

Example

```
<AssertTime since="GroupLostSightOfTarget" isLessThan="10"  
orNeverBeenSet="1" />
```

Priority:Case node

This node executes the child with the current highest priority. The priorities are derived from the order in which the children are defined and the satisfaction of their individual conditions, so that the highest priority goes to the first child to have its condition met.

The children's conditions must be specified with the use of Case nodes with the exception of the last child which is considered to be the default case, meaning that its condition is always true and cannot be specified.

Parameters

- **condition:** Specifies the condition of the child.

Behavior

The node returns the result of the execution of the child node.

Example

```
<Priority>  
  <Case condition="TargetInCloseRange and TargetVisible">  
    <Melee target="AttentionTarget" />  
  </Case>  
  <Case>  
    <Look at="Target" />  
  </Case>  
</Priority>
```

LuaGate node

This node executes the child node if the result from running a lua snippet is true.

Parameters

- **code:** The lua code to be executed.

Behavior

- **Failure:** If the lua code returns a value different from true. Otherwise, the node returns the result of the execution of its child node.

Example

```
<LuaGate code="return AI.GetGroupScopeUserCount(entity.id,  
'DeadBodyInvestigator') == 0">
```


RandomGate node

This node executes or not the child node based on a random chance.

Parameters

- **opensWithChance:** The chance of executing the child node (0.0 - 1.0).

Behavior

- **Failure:** If the child is not executed. Otherwise, the node returns the result of the execution of its child node.

Example

```
<RandomGate opensWithChance="0.5">
```

AdjustCoverStance node

This node updates the agent's cover stance based on the maximum height in which his current cover is still effective.

Parameters

- **duration:** (Optional) The amount of seconds the node will execute. Use 'continuous' for unlimited time.
- **variation:** (Optional) The extra random amount of seconds that will be added on top of the specified duration, in the range (0, variation).

Behavior

- **Success:** If the duration of execution elapses.
- **Failure:** If the child is not in cover.

Example

```
<AdjustCoverStance duration="5.0" variation="1.0"/>
```

SetAlertness node

This node sets the agent's alertness value.

Parameters

- **value:** The alertness value (0-2).

Behavior

The node succeeds immediately.

Example

```
<SetAlertness value="1" />
```

Log node

This node adds a message to the agent's personal log.

Parameters

- **message:** The message to be logged.

Behavior

The node succeeds immediately.

Example

```
<Log message="Investigating suspicious activity." />
```

Communicate node

This node requests the communication manager to play one of the agent's readabilities.

Parameters

- **name:** The name of the communication to be played.
- **channel:** The channel on which the communication is to be set.
- **waitUntilFinished:** (Optional) Specifies if the execution should wait for the end of the communication before finishing.
- **timeout:** (Optional) The threshold defining the maximum amount of seconds the node will wait.
- **expiry:** (Optional) The amount of seconds the communication can wait for the channel to be clear.
- **minSilence:** (Optional) The amount of seconds the channel will be silenced after the communication is played.
- **ignoreSound:** (Optional) Sets the sound component of the communication to be ignored.
- **ignoreAnim:** (Optional) Sets the animation component of the communication to be ignored.

Behavior

- **Success:** If the timeout elapses or when the readability is complete if the node is set to wait until the communication is finished.
- **Failure:**

Example

```
<Communicate name="Advancing" channel="Tactic" expiry="1.0"  
waitUntilFinished="0" />
```

Animate node

This node sets the agent to play an animation.

Parameters

- **name:** The name of the animation to be played.
- **urgent:** (Optional) Adds the urgent flag to the animation.

- **loop:** (Optional) Adds the loop flag to the animation.
- **setBodyDirectionTowardsAttentionTarget:** (Optional) Changes the body target direction to be facing the attention target.

Behavior

- **Success:** If the animation failed to be initialized or when it is finished.

Example

```
<Animate name="LookAround" loop="1" />
```

Signal node

This node sends a signal to the AI system.

Parameters

- **name:** The name of the signal to be sent.
- **filter:** (Optional) The filter to be applied to the signal in the AI system.

Behavior

The node succeeds immediately.

Example

```
<Signal name="StartedJumpAttack" />
```

SendTransitionSignal node

This node sends a signal destined for a state machine node on the behavior tree, with the explicit intent of causing a change of state.

Parameters

- **name:** The name of the signal to be sent.

Behavior

The node does not succeed or fail.

Example

```
<SendTransitionSignal name="LeaveSearch" />
```

Stance node

This node sets the stance of the agent.

Parameters

- **name:** The name of the stance to be set: Relaxed, Alerted, Crouch, Stand.

- **stanceToUseIfSlopeIsTooSteep**: (Optional) The alternative stance to be used in case the slope is too steep.
- **allowedSlopeNormalDeviationFromUpInDegrees**: (Optional) Defines how steep can the slope be for this stance.

Behavior

The node succeeds immediately.

Example

```
<Stance name="Crouch" allowedSlopeNormalDeviationFromUpInDegrees=" 30 "  
stanceToUseIfSlopeIsTooSteep="Stand" />
```

IfCondition node

This node executes the child node if the specified condition is satisfied.

Parameters

- **condition**: Specifies the condition to be checked.

Behavior

The node returns the result of the child's execution if the condition is true, otherwise it fails.

Example

```
<IfCondition condition="TargetVisible">  
  <Communicate name="AttackNoise" channel="BattleChatter" expiry="2.0"  
  waitUntilFinished="1" />  
</IfCondition>
```

AssertCondition node

This node succeeds if the specified condition is satisfied.

Parameters

- **condition**: Specifies the condition to be checked.

Behavior

The node succeeds if the condition is true, otherwise it fails.

Example

```
<AssertCondition condition="HasTarget" />
```

LuaWrapper node

This node executes the child node with the additional option of running a lua script on the start and/or end of that execution.

Parameters

- **onEnter:** (Optional) The code to be executed at the start.
- **onExit:** (Optional) The code to be executed at the end.

Behavior

The node returns the result of the child's execution.

Example

```
<LuaWrapper onEnter="entity:EnableSearchModule()"
onExit="entity:DisableSearchModule()">
  <Animate name="AI_SearchLookAround" />
</LuaWrapper>
```

ExecuteLua node

This node executes a lua script.

Parameters

- **code:** The code to be executed.

Behavior

The node always succeeds.

Example

```
<ExecuteLua code="entity:SetEyeColor(entity.EyeColors.Relaxed)" />
```

AssertLua node

This node executes Lua code and translates the return value of that code from true or false to success or failure. It can then be used to build preconditions in the Modular Behavior Tree.

Parameters

- **code:** The code to be executed.

Behavior

Succeeds if the Lua code returns value is true, otherwise it fails.

Example

```
<AssertLua code="return entity:IsClosestToTargetInGroup()" />
```

GroupScope node

This node tries to enter the agent in a group scope, which is limited by the specified amount of concurrent users. If the node succeeds to do that, then the child node is executed.

Parameters

- **name:** The name of the group scope to be entered.
- **allowedConcurrentUsers:** (Optional) The maximum number of simultaneous users of that can be in the specified group scope.

Behavior

The node fails if the agent could not enter the group scope, otherwise returns the result of the execution of the child.

Example

```
<GroupScope name="DeadBodyInvestigator" allowedConcurrentUsers="1">  
  <SendTransitionSignal name="GoToPrepareToInvestigateDeadBody" />  
</GroupScope>
```

Look node

This node adds a location for the agent to look at and clears it when the node stops executing.

Parameters

- **at:** The location to look at: ClosestGroupMember, RefPoint,Target.

Behavior

The nodes does not succeed or fail.

Example

```
<Look at="ClosestGroupMember" />
```

Aim node

This node sets the location where the agent should aim, clearing it when the node stops executing.

Parameters

- **at:** The location to look at: RefPoint,Target.
- **angleThreshold:** (Optional) The tolerance angle for the agent to be considered aiming in the desired direction.
- **durationOnceWithinThreshold:** (Optional) The amount of seconds to keep on aiming.

Behavior

- **Success:** If after aiming in the desired direction for the specified time, if the location is not valid or if the timeout elapses.

Example

```
<Aim at="Target" durationOnceWithinThreshold="2.0" />
```

AimAroundWhileUsingAMachineGun node

This node updates the aim direction of the agent for when he is using a mounted machine gun.

Parameters

- **maxAngleRange:** (Optional) The maximum amount to deviate from the original position.
- **minSecondsBetweenUpdates:** (Optional) The minimum amount of delay between updates.
- **useReferencePointForInitialDirectionAndPivotPosition:**

Behavior

The node does not succeed or fail.

Example

```
<AimAroundWhileUsingAMachineGun minSecondsBetweenUpdates="2.5"  
  maxAngleRange="30"  
  useReferencePointForInitialDirectionAndPivotPosition="1" />
```

ClearTargets node

This node clears the agent's targets information.

Parameters

None.

Behavior

The node succeeds immediately.

Example

```
<ClearTargets />
```

StopMovement node

This node sends a request to the Movement System to stop all the movements.

This may not always immediately physically stop the agent. The Movement System may be dependent on the influence of animations and physics, for example, which may result in a natural stop and not an immediate stop.

Parameters

- **waitUntilStopped:** 1 if the node should wait for the Movement System to have processed the request; 0 if not.
- **waitUntilIdleAnimation:** 1 if the node should wait until the Motion_Idle animation fragment started running in Mannequin, 0 if not.

Behavior

- **Success:** If the stop request has been completed.

Example

```
<StopMovement waitUntilStopped="1" waitUntilIdleAnimation="0" />
```

Teleport node

This node teleports the character when the destination point and the source point are both outside of the camera view.

Parameters

None.

Behavior

- **Success:** After the character is teleported.

Example

```
<Teleport />
```

SmartObjectStateWrapper node

This node executes the child node with the additional option of setting certain smart objects states on the start and/or end of that execution.

Parameters

- **onEnter:** (Optional) The smart object states to set at the start of the child's execution.
- **onExit:** (Optional) The smart object states to set at the end of the child's execution.

Behavior

The node returns the result of the execution of its child node.

Example

```
<SmartObjectStatesWrapper onEnter="InSearch" onExit="-InSearch">  
  <Animate name="LookAround" />  
</SmartObjectStatesWrapper>
```

CheckTargetCanBeReached node

This node checks if the agent's attention target can be reached.

Parameters

- **mode:** Defines the target to use: UseLiveTarget or UseAttentionTarget.

Behavior

- **Success:** If it can reach the target.

- **Failure:** If it cannot reach the target.

Example

```
<CheckIfTargetCanBeReached mode="UseLiveTarget" />
```

MonitorCondition node

This node continuously checks for the state of a specified condition.

Parameters

- **condition:** Specifies the condition to be checked.

Behavior

- **Success:** When the condition is satisfied.

Example

```
<MonitorCondition condition="TargetVisible" />
```

AnimationTagWrapper node

This node executes the child node, adding an animation tag for the agent on the beginning of that execution and clearing it on the end.

Parameters

- **name:** The animation tag to be set.

Behavior

The node returns the result of the execution of its child node.

Example

```
<AnimationTagWrapper name="ShootFromHip">  
  <Shoot at="Target" stance="Stand" duration="5" fireMode="Burst" />  
</AnimationTagWrapper>
```

ShootFromCover node

This node sets the agent to shoot at the target from cover and adjusts his stance accordingly.

Parameters

- **duration:** The number of seconds the node should execute.
- **fireMode:** The firemode to be used for shooting.
- **aimObstructedTimeout:** (Optional) The number of seconds the aim is allowed to be obstructed.

Behavior

- **Success:** If the duration of execution elapses.
- **Failure:** If the agent is not in cover, if there's no shoot posture or if the aim obstructed timeout elapses.

Example

```
<ShootFromCover duration="10" fireMode="Burst" aimObstructedTimeout="3" />
```

Shoot node

This node sets the agent to shoot at a target or a location.

Parameters

- **duration:** The number of seconds the node should execute.
- **at:** The location to shoot at: AttentionTarget, ReferencePoint, LocalSpacePosition.
- **fireMode:** The fire mode to be used for shooting.
- **stance:** The stance to be set while shooting: Relaxed, Alerted, Crouch, Stand.
- **position:** (Mandatory only if the target is a local space position). The local space position to be used as the target.
- **stanceToUseIfSlopeIsTooSteep:** (Optional) The alternative stance to be used in case the slope is too steep.
- **allowedSlopeNormalDeviationFromUpInDegrees:** (Optional) Defines how steep can the slope be for this stance.
- **aimObstructedTimeout:** (Optional) The amount of seconds the aim is allowed to be obstructed.

Behavior

- **Success:** If the duration of execution elapses.
- **Failure:** If the aim obstructed timeout elapses

Example

```
<Shoot at="Target" stance="Crouch" fireMode="Burst" duration="5"  
allowedSlopeNormalDeviationFromUpInDegrees="30"  
stanceToUseIfSlopeIsTooSteep="Stand" />
```

ThrowGrenade node

This node sets the agent to attempt a grenade throw.

Parameters

- **timeout:** The maximum amount of seconds the node will wait for the grenade to be thrown.
- **type:** The type of grenade: emp, frag, smoke.

Behavior

- **Success:** If a grenade is thrown before it times out.
- **Failure:** If a grenade is not thrown before it times out.

Example

```
<ThrowGrenade type="emp" timeout="3" />
```

PullDownThreatLevel node

This node sets the agent to lower his notion the target's threat.

Parameters

None.

Behavior

The node succeeds immediately.

Example

```
<PullDownThreatLevel to="Suspect" />
```

Game AI MBT Nodes

Game AI Modular Behavior Tree nodes are mostly used to offer specific game functionality. Each type of game may have multiple character types and each type may need to trigger specific logic to perform action in the game. Game-specific nodes are generally not suitable for general use and may need to be tweaked to fit the needs of your game.

Melee node

The Melee node will trigger a melee attack against an agent target. The Melee node succeeds regardless of whether the melee attack is performed and damages the target or not.

A melee attack is performed when the following conditions are satisfied:

- If the **failIfTargetNotInNavigationMesh** parameter is set, the target must be on a valid walkable position. Certain melee animations could move the character pushing it outside the navigable area while trying to melee a target outside the navigation mesh (MNM).
- If the target is not between the threshold angle specified by the entity lua value `melee.angleThreshold`.

Parameters

- **target**: The target of the melee. This parameter could be set as `AttentionTarget` or a generic `RefPoint`.
- **cylinderRadius**: The radius of the cylinder used for the collision check of the hit.
- **hitType**: The type of hit that is sent to the game rules. Default is `CGameRules::EHitType::Melee`.
- **failIfTargetNotInNavigationMesh**: Determines whether the node should not try to melee a target that is outside the navigation mesh. This will only cause the melee attack to not be performed - the Melee node will still succeed.
- **materialEffect**: The material effect used when the melee attack hits the target.

Behavior

- **Success:** Occurs irregardless of the actual execution of the melee attack.

Example

```
<Melee target="AttentionTarget" cylinderRadius="1.5"  
hitType="hitTypeName" materialEffect="materialEffectName" />
```

The following is an example lua file that defines the specific character in use:

```
melee =  
{  
  damage = 400,  
  hitRange = 1.8,  
  knockdownChance = 0.1,  
  impulse = 600,  
  angleThreshold = 180,  
},
```

The following table lists the various parameters one can use in the lua file:

Parameters

- **damage:** Defines the amount of damage the melee attack inflicts on the target.
- **hitRange:** Defines the height of the cylinder used to check if the melee attack can hit the target.
- **knockdownChance:** Defines the probability that a successful melee attack knocks down the player.
- **impulse:** Defines the amount of the impulse that is applied to the player in case of a successful melee attack.
- **angleThreshold:** Threshold between the agent direction and the direction between the agent and the target to allow a melee attack to be attempted.

KeepTargetAtADistance node

This node keeps the live target at a distance by physically pushing the target away if it is within the defined minimum distance.

This is useful when there's an action close to the player and you want to avoid clipping through the camera. This is preferable to increasing the capsule size since that will affect how the character can fit through tight passages. This node is mostly used in parallel with other actions that need to be performed while the player is not too close to the agent.

Parameters

- **distance:** The minimum distance allowed between the player and the agent.
- **impulsePower:** The power of the impulse used to keep the player at least at the minimum distance.

Behavior

The node never succeeds or fails. Once executed, it is always running until out of the scope of the executed nodes.

Example

```
<KeepTargetAtADistance distance="1.8" impulsePower="1.5" />
```

SuppressHitReactions node

This node enables and disables the hit reaction system for the agent during its execution.

Parameters

None.

Behavior

- **Success:** If the child succeeds
- **Failure:** If the child fails.

Example

```
<SuppressHitReactions>  
  <SomeChildNode />  
</SuppressHitReactions>
```

InflateAgentCollisionRadiusUsingPhysicsTricksTrick node

This node uses a feature of the physics system to inflate the capsule of the agent such that it has one radius for collisions with the player, and a different radius for collisions with the world.

Parameters

- **radiusForAgentVsPlayer:** The radius use to calculate the collision between the agent and the player.
- **radiusForAgentVsWorld:** The radius used to calculate the collision between the agent and the world.

Behavior

The node never succeeds or fails but always runs.

Example

```
<InflateAgentCollisionRadiusUsingPhysicsTrick radiusForAgentVsPlayer="1.0"  
  radiusForAgentVsWorld="0.5" />
```

ScorcherDeploy:RunWhileDeploying node

This node and the following one are special decorator nodes that the Scorcher uses to deploy and undeploy as part of the shooting phase. These two nodes rely on external Lua scripts and various signals to work properly. In this this way you don't have to explicitly expose more functionality from the AI system libraries.

This node must contain exactly one child node that runs while the Scorcher is in the processes of deployment getting ready for an attack. It can be used, for example, to control aiming before actually shooting.

Parameters

None.

Behavior

- **Success:** If the child node succeeds.
- **Failure:** If the child node fails.

Example

ScorcherDeploy:RunWhileDeployed node

This node must contain exactly one child node that controls the actual aiming and firing.

Parameters

None.

Behavior

- **Success:** If the child node succeeds. This will make the parent node start the undeployment sequence.
- **Failure:** If the child node fails.

Example

```
<ScorcherDeploy maxDeployDuration="1.0">
  <RunWhileDeploying>
    <SomeChildNode>
  </RunWhileDeploying>
  <RunWhileDeployed>
    <SomeOtherChildNode>
  </RunWhileDeployed>
</ScorcherDeploy>
```

HeavyShootMortar node

Used to control the shooting of heavy mortar. It tries to simplify and to centralize the check of the pre-condition and the initialization of the weapon plus the re-selection of the primary weapon.

Parameters

- **to:** (Optional) Defines the target of the shooting. Possible values: Target or RefPoint. Default is Target.
- **firemode:** (Optional) The Heavy X-Pak (or Mortar) has two different firemodes. Possible values: Charge or BurstMortar. Default is Charge.
- **timeout:** (Optional) Defines the maximum time the node can try to perform the shooting. Default value is 5.0 seconds.
- **aimingTimeBeforeShooting:** (Optional) Defines the time in which the Heavy will aim before starting the shooting. Default is 1.0 seconds. This amount of time must be larger than the global timeout.
- **minAllowedDistanceFromTarget:** (Optional) Defines the minimum distance from the Target to allow shooting. Default is 10.0 m.

Behavior

- **Success:** The node succeeds when the shooting succeeds.
- **Failure:** The node fails if the timeout is reached, if the Heavy is closer to the target than the **minAllowedDistanceFromTarget** value, or if there obstructions two meters in front of the Heavy

(a cylinder check is performed to avoid this condition in front of the mortar if there is an object the Heavy tries to shoot at.)

Example

```
<HeavyShootMortar to="RefPoint" fireMode="Charge"  
  aimingTimeBeforeShooting="2" timeout="7" />
```

SquadScope node

Used to enter a squad scope, which is limited by the specified amount of concurrent users. If the node succeeds to do that, then the child node is executed.

Parameters

- **name:** The name of the squad scope to enter.
- **allowedConcurrentUsers:** (Optional) Number of allowed concurrent users in the specified scope. Default value = 1.

Behavior

- **Success:** The node succeeds when the child succeeds.
- **Failure:** The node fails if it can't enter the specified scope or if the child fails.

Example

```
<SquadScope name="ANameForTheScope" allowedConcurrentUsers="5">  
  <SomeChildNode />  
</SquadScope>
```

SendSquadEvent node

Used to send an event only to the squad members.

Parameters

- **name:** Name of the event to be sent.

Behavior

The node succeeds after having sent the event. The node never fails.

Example

```
<SendSquadEvent name="ANameForTheEvent" />
```

IfSquadCount node

This node checks if a squad contains a specific amount of members and if so executes its child.

Parameters

One of the following parameters must be specified.

- **isGreaterThan:** (Optional) To succeed, checks if the number of members is greater than the specified number.
- **isLesserThan:** (Optional) To succeed, checks if the number of members is lesser than the specified number.
- **equals:** (Optional) To succeed, checks if the number of members is equal to the specified number.

Behavior

- **Success:** If the number of members satisfies the specified condition.
- **Failure:** If otherwise.

Example

```
<IfSquadCount isGreaterThan="1">  
  <SomeChildNode />  
</IfSquadCount>
```

Helicopter AI MBT Nodes

The following flying vehicle AI Modular Behavior Tree nodes are supported.

Hover node

Used to let a flying agent hover at its current position.

Parameters

None.

Behavior

This node never finishes by itself and will continue to hover the agent until it is forced to terminate.

Example

```
<Hover />
```

FlyShoot node

Used to let a flying agent shoot at its attention target, when possible from its current position. If the secondary weapon system is used, then the node will only open fire if the weapons are deemed to be able to hit close enough to the target. Otherwise normal firing rules are applied.

Parameters

- **useSecondaryWeapon:** 1 if the secondary weapon system should be used (these are often rocket launchers); 0 if not

Default value: 0

Behavior

This node never finishes by itself and the agent will continue shoot until it is forced to terminate.

Example

```
<FlyShoot useSecondaryWeapon="1" />
```

Fly node

Used to let an agent fly around by following a path. Paths should be assigned to the agent via a flow graph.

Upon arrival, the `ArrivedCloseToPathEnd` and `ArrivedAtPathEnd` events are emitted.

Parameters

- **desiredSpeed:** The desired speed to move along the path in meters/second.
Default value: 15.0
- **pathRadius:** The radius of the path in meters. The agent will try to stay within this distance from the line segments of the path.
Default value: 1.0
- **lookAheadDistance:** How far long the path, in meters, to look ahead for generating "attractor points" to fly to.
Default value: 3.0
- **decelerateDistance:** When nearing the end of the path, the agent will start to decelerate at the specified distance in meters.
Default value: 10.0
- **maxStartDistanceAlongNonLoopingPath:** When linking up with a non-looping path, this is the maximum distance in meters that the node is allowed to scan ahead to find the closest point to the path where to start at. This can be useful, for example, to prevent the agent from snapping to the path at a position that is seemingly closer but is actually behind a wall after a U-turn.
Default value: 30.0
- **loopAlongPath:** 1 if the agent should follow the path in an endless loop; 0 if not.
Default value: 0
- **startPathFromClosestLocation:** 1 if the agent should start following the path at its closest position; 0 if it should start following it from the very first path waypoint.
Default value: 0
- **pathEndDistance:** The distance towards the end of the path at which the node should start sending some arrival notification events.
Default value: 1.0
- **goToRefPoint:** 1 if the current reference point should be appended to the end of the path; 0 if not.
Default value: 0

Behavior

- **Success:** If the agent arrived at the end of the path.
- **Failure:** If no valid path was assigned to the agent.

Example

```
<Fly lookaheadDistance="25.0" pathRadius="10.0"  
decelerateDistance="20.0" pathEndDistance="1" desiredSpeed="15"  
maxStartDistanceAlongNonLoopingPath="30" loopAlongPath="0" goToRefPoint="1"  
startPathFromClosestLocation="1" />
```

Outputs

When	Lua variable	Overridden XML tag
Node activation	Helicopter_Loop	loopAlongPath
Node activation	Helicopter_StartFromClosestLocation	startPathFromClosestLocation
Each node tick	Helicopter_Speed	desiredSpeed

FlyForceAttentionTarget node

Used to keep forcing an attention target onto a flying vehicle. The attention target that should be enforced is acquired during each tick of the node from the `Helicopter_ForcedTargetId` Lua script variable.

Parameters

None.

Behavior

This node never finishes by itself and keeps forcing the attention target onto the agent. When the node is deactivated again, the `ForceAttentionTargetFinished` event is emitted.

Example

```
<FlyForceAttentionTarget />
```

FlyAimAtCombatTarget node

Used to aim a flying agent at its target, taking into account special aiming adjustments for weapons.

Parameters

None.

Behavior

This node never finishes by itself and keeps forcing agent to rotate its body towards its attention target.

Example

```
<FlyAimAtCombatTarget />
```

WaitAlignedWithAttentionTarget node

Used to wait until the agent is facing its attention target.

Parameters

- **toleranceDegrees:** The maximum allowed angle between the attention target and the forward direction of the agent, in the range of 0.0 to 180.0 degrees.

Default value: 20.0

Behavior

- **Success:** If the angle between the agent's forward direction and its attention target is small enough.
- **Failure:** If the agent has no attention target.

Example

```
<WaitAlignedWithAttentionTarget toleranceDegrees="40" />
```

HeavyShootMortar node

Used to control the shooting of a heavy mortar. The precondition and initialization of the weapon as well the reselection of the primary weapon is simplified and centralized.

Parameters

- **to:** (Optional) Defines the target of the shooting. Possible values: Target or RefPoint.

Default value: Target

- **firemode:** (Optional) The Heavy X-Pak (or Mortar) has two different firing modes. Possible values are Charge and BurstMortar.

Default value: Charge

- **timeout:** (Optional) Defines the maximum time in seconds that the node can try to perform the shooting.

Default value: 5.0

- **aimingTimeBeforeShooting:** (Optional) Defines the time in seconds in which the heavy mortar will aim before starting the shooting. This amount of time must be bigger than the global timeout.

Default value: 1.0

- **minAllowedDistanceFromTarget:** (Optional) Defines the minimum distance in meters from the Target to allow the shooting.

Default value: 10.0

Behavior

- **Success:** If the shooting succeeds.
- **Failure:** If if the heavy mortar is closer to the Target than the minimum distance, if there are obstructions 2 meters in front of the heavy mortar, or if the timeout is reached.

Example

```
<HeavyShootMortar to="RefPoint" fireMode="Charge"  
aimingTimeBeforeShooting="2" timeout="7" />
```

SquadScope node

Used to enter a squad scope, which is limited by the specified number of concurrent users. If the node succeeds to do that, then the child node is executed.

Parameters

- **name:** The name of the squad scope to enter.
- **allowedConcurrentUsers:** (Optional) Number of allowed concurrent users in the specified scope.

Default value: 1

Behavior

- **Success:** If the child succeeds
- **Failure:** If it can't enter the specified scope or if the child fails.

Example

```
<SquadScope name="ANameForTheScope" allowedConcurrentUsers="5">  
  <SomeChildNode />  
</SquadScope>
```

SendSquadEvent node

Used to send an event only to the squad members.

Parameters

- **name:** Name of the event to be sent.

Behavior

- **Success:** If the event is sent.
- **Failure:** Never fails

Example

```
<SendSquadEvent name="ANameForTheEvent" />
```

IfSquadCount node

Used to check if a squad contains a specific number of members and if so executes its child.

Parameters

- **isGreaterThan:** (Optional) To succeed the node will check if the number of members is greater than the specified amount.
- **isLesserThan:** (Optional) To succeed the node will check if the number of members is lesser than the specified amount.
- **equals:** (Optional) To succeed the node will check if the number of members is equal to the specified amount.

Behavior

- **Success:** If the number of members in the squad satisfies the specified comparison.
- **Failure:** the number of members in the squad does not satisfy the specified comparison.

Example

```
<IfSquadCount isGreaterThan="1">  
  <SomeChildNode />  
</IfSquadCount>
```

AI Agent Debugging

There are several tools available for debugging AI agent behaviors at the game level.

Topics

- [Using the AI Debug Recorder \(p. 115\)](#)
- [Using the AI Debug Viewer \(p. 116\)](#)
- [Using AI Debug Console Variables \(p. 118\)](#)
- [Using AI Bubbles for Error Messaging \(p. 121\)](#)
- [Using AI Log and AISignals Files \(p. 122\)](#)

Using the AI Debug Recorder

The AI Debug Recorder is a recording tool that logs all inputs, decisions, computations and other useful data for an AI agent in real-time while the game is being played. At the end of the game session, the recorder serializes all the data for future processing.

There are several ways to start or stop an AI debug recording session, as follows:

- **Automatically using the Console** – Use the `ai_Recorder_Auto` console variable to automatically begin recording whenever a new game session starts. Similarly, the recording stops and saves when the game session ends, by whatever means (except the game crashing).
- **Manually using the Console** – Use the `ai_Recorder_Start` and `ai_Recorder_Stop` console variables to start or stop a recording as needed.
- **Manually in Code** – Use the `IAIRecorder` interface to start or stop a recording as needed.

Recorder Output File

Regardless of which method is used to perform a recording, all recordings are saved within the `\Recordings` folder in the Lumberyard root directory (`\lumberyard\dev`). The file name of the recording is formatted as follows:

MapName_Build(A) Date(B) Time(C).rcd

- *MapName* – The name of the map in which the recording took place. The exception is if the recording took place in Lumberyard Editor, in which case the map name is EDITORAUTO as a suffix.
- *Build(A)* – Version of the build with which the recording was made. We recommended using the same build version to view the recording.

- *Date(B)* – Date the recording was made.
- *Time(C)* – The time the recording was saved.

Note

If you create a manual recording, you enter your own file name to use. If none is specified, the above format is used.

Recorder Data Streams

An AI Debug recording is comprised of many data streams that chronologically log a specific type of input, as follows. It is also possible to add a new stream to the recording if needed.

- E_RESET – When the agent is reset.
- E_SIGNALRECIEVED – When the agent receives a signal.
- E_SIGNALRECIEVEDAUX – When the agent receives an auxiliary signal.
- E_SIGNALEXECUTING – When the agent is executing a received signal (processing it).
- E_GOALPIPESELECTED – When the agent selects a new goal pipe.
- E_GOALPIPEINSERTED – When the agent inserts a new goal pipe.
- E_GOALPIPERESETED – When the goal pipe on the agent is reset.
- E_BEHAVIORSELECTED – When the agent selects a new behavior.
- E_BEHAVIORDESTRUCTOR – When the agent's current behavior has its destructor called.
- E_BEHAVIORCONSTRUCTOR – When the agent's current behavior has its constructor called.
- E_ATTENTIONTARGET – When the agent's attention target changes.
- E_ATTENTIONTARGETPOS – When the position of the agent's attention target changes.
- E_REGISTERSTIMULUS – When the agent receives a perception stimulus.
- E_HANDLERNEVENT – When the agent's mind handles an event.
- E_REFPOINTPOS – When the agent's reference point position changes.
- E_AGENTPOS – When the agent's position changes.
- E_AGENTDIR – When the agent's look direction changes.
- E_LUACOMMENT – When a Lua comment is made on the agent.
- E_HEALTH – When the agent's health changes.
- E_HIT_DAMAGE – When the agent receives hit damage.
- E_DEATH – When the agent is killed.
- E_SIGNALEXECUTEDWARNING – When the agent is taking too long to process a signal.
- E_BOOKMARK – When a bookmark is placed on the agent.

To record information for any of these events, use the `IAIRecordable` interface (which all AI Objects inherit from, and is used to link to the AI Debug Recorder itself).

All of the data streams listed are handled by the AI System with the exception of the Bookmark stream. This is a special stream that is used to mark areas of interest for easy debugging later.

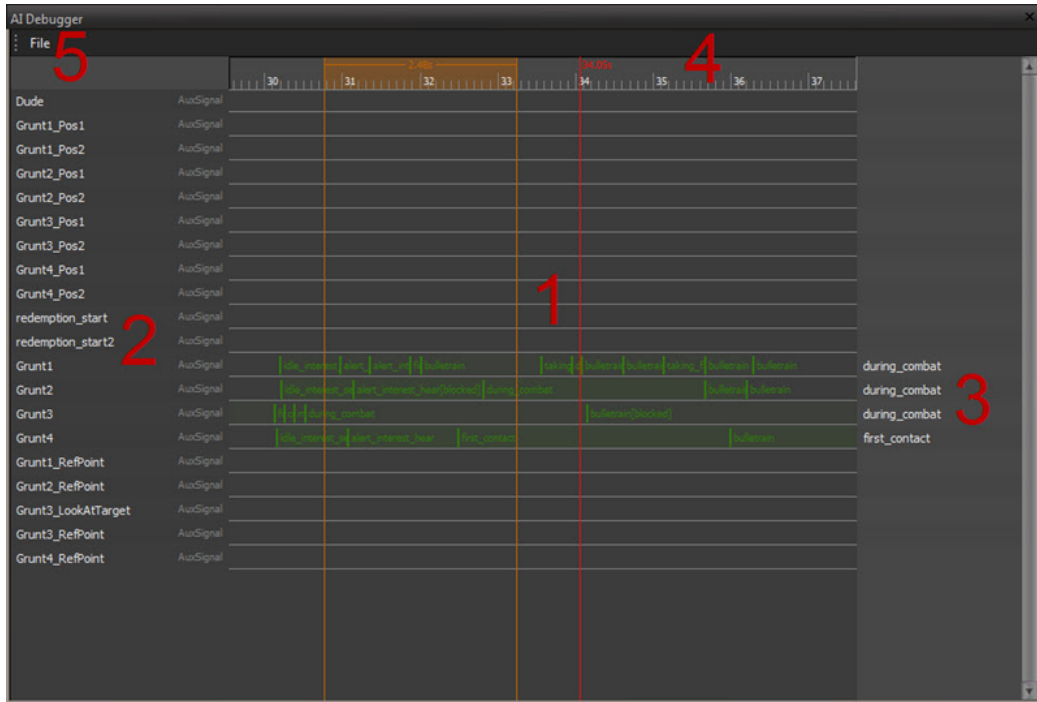
For example, a game project may connect a keyboard input to log event data on the Bookmark stream whenever a button is pressed, which informs the QA team to push the button whenever odd behavior from the AI is observed.

Using the AI Debug Viewer

AI Debug Viewer is the viewing utility that loads, parses, and displays the AI Debug Recorder file. This utility is accessed from Lumberyard Editor.

To view a AI Debug Recorder session

1. In Lumberyard Editor, click **View, Open View Pane, AI Debugger**.
2. In AI Debugger, click **File, Load** to view the last recorded session, or click **Load As** to view a prior pre-recorded session.



The timeline window can be broken down as shown below:

1. **Stream Window.** Displays the contents of all active streams for all of the AI agents who were recorded, along the timeline. By right-clicking in this window, a context menu is displayed, as follows:

Context Menu Items

Menu Item	Description
Copy Label	Copies the current label of the stream to the clipboard. See the Info Window for more details.
Find...	Finds the next occurrence of the label you specify along the timeline and sets the cursor to that point.
Goto Start	Sets the cursor to the starting time of the recording.
Goto End	Sets the cursor to the ending time of the recording.
Goto Agent Location	Sets the position of the camera in the Editor to the location of the agent who owns the stream.
Copy Agent Location	Copies the location of the agent who owns the stream to the clipboard.
Following Content	Contains all of the available streams for viewing. The check box next to the name of the stream means it is enabled. Enabling a stream causes its contents to be displayed in the Stream window.

2. Agent Window. Displays the name of the agent who owns the contents of the stream. Right-clicking this window displays the context menu, as follows:

- **Enable Debug Drawing** - Select to active debugging for this agent.
- **Set Editor View** - Select to focus the Editor's camera as centered on both the position and facing direction of the AI agent based on where the cursor is currently set. This allows you to see what that AI agent was seeing at any moment in time. By moving the cursor, you can replay the agent's movements and see what it was seeing.
- **Set Color** - Change the debug color used for representing this agent's information.

3. Info Window. Displays the last value of the stream based on the cursor position for the stream to the left. The references to **label** in various places throughout the debug viewer refer to the text you see in this pane.

4. Ruler Window. Displays the current time and can be used to select time ranges or manipulate the cursor.

The **cursor** is represented by a solid red line. Any information that occurred at the moment of time depicted by the cursor.

The **time range** is represented by a blue box. Click and hold anywhere in the ruler window to drag the time range box. Releasing the mouse button sets the time range box. All information that occurred during the highlighted time is drawn.

5. Menu Window. Contains configuration and command settings for the view pane.

Using AI Debug Console Variables

There are a number of console variables available for AI agent debugging. One of the most useful is the **ai_DebugDraw** console variable. Setting this variable to 1 results in debug information displayed above any active AI agent.

Note

Use the **ai_AgentStatsDist** variable listed below to set the distance above the AI agent that debug information displays.

To enable ai_DebugDraw

1. Open Lumberyard Editor and select **View, Show Console**.
2. At the bottom of the console window, type **ai_DebugDraw 1** or one of the other values, as needed:

ai_DebugDraw Values

Value	Description
-1	Only warnings and errors; no other information displays
0	Disables AI debug draw
1	Standard AI debug draw information displays
71	Draws all forbidden areas (including auto-generated ones)
72	Draws graph errors (problematic areas are highlighted with circles)
74	Draws the whole navigation graph

Value	Description
79	Draws the navigation graph close to the player (within 15m from the camera; faster than 74)
80	Draws tagged nodes (during A*)
81	Calculates (if necessary) and then draws 3D (volume) hidespots
82	Draws 3D (volume) hidespots
85	Draws steep slopes (determined by ai_steep_slope_up_value and ai_steep_slope_across_value)
90	Draws flight navigation within a 200m range of the player
179	Similar to 79, but also shows triangulation edges centers
279	Similar to 179, but also shows water depth information
1017	Visualizes navigation links of node that encloses entity "test"

Setting AI_DebugDraw to 1

Setting **ai_DebugDraw** to 1 enables the following console variables for debugging:

ai_AllTime

Displays the update times of all agents in milliseconds. Green indicates <1ms and white indicates 1ms-5ms.

Values: 0 = Disabled | 1 = Enabled

ai_DebugDrawNavigation 1

Displays the navigation mesh for the MNM system. Blue areas are navigable for AI agents to move around in. Red areas are cut off from the main mesh and are not reachable by AI agents.

Values: 0 = Disabled | 1 = Triangles and contour | 2 = Triangles, mesh, and contours | 3 = Triangles, mesh contours, and external links

ai_DrawBadAnchors

Toggles drawing out-of-bounds AI objects of a particular type for debugging AI. Valid only for 3D navigation. Draws red spheres at positions of anchors that are located out of navigation volumes. Those anchors must be moved.

Values: 0 = Disabled | 1 = Enabled

ai_DrawFormations

Draws all the currently active formations of the AI agents.

Values: 0 = Disabled | 1 = Enabled

ai_DrawModifiers

Toggles the AI debugging view of navigation modifiers.

ai_DrawNode

Toggles the visibility of named agent's position on AI triangulation. See also: ai_DrawNodeLinkType and ai_DrawNodeLinkCutoff.

Values: none = Disabled | all = Displays all agent nodes | player = Displays the player node | agent name = Displays the agent node

ai_DrawNodeLinkType

Sets the link parameter to draw with ai_DrawNode.

Values: 0 = Pass radius | 1 = Exposure | 2 = Maximum water depth | 3 = Minimum water depth

ai_DrawNodeLinkCutoff

Sets the link cutoff value in ai_DrawNodeLinkType. If the link value is more than ai_DrawNodeLinkCutoff, the number displays in green. If the link value is less than ai_DrawNodeLinkCutoff, the number displays in red.

ai_DrawOffset

Vertical offset during debug drawing.

ai_DrawPath

Draws the generated paths of the AI agents. ai_drawoffset is used.

Values: none = Disabled | squad = Squad members | enemy = Enemies | groupID = Group members

ai_DrawRadar

Draws a radar overlay at the center of the view.

Values: 0 = Disabled | *value* = size of radar (m)

ai_DrawRadarDist

AI radar draw distance in meters.

Default value: 20m

ai_DrawRefPoints

Toggles reference points and beacon view for debugging AI. Draws balls at AI reference points.

Usage: "all", agent name, group id

ai_DrawStats

Toggles drawing stats (in a table on top left of screen) for AI objects within a specified range. Displays attention target, goal pipe, and current goal.

ai_StatsDisplayMode 1

Displays information on the number of active AI agents, full AI updates per frame, and the number of TPS queries processed each frame.

ai_DrawTargets

Distance to display the perception events of all enabled puppets. Displays target type and priority.

ai_DrawType

Displays all AI object of a specified type. If object is enabled, it displays with a blue ball. If object is disabled, it displays with a red ball. A yellow line represents forward direction of the object.

Values: <0 = Disabled | 0 = Displays dummy objects | >0 = Object type to display

ai_DrawTrajectory

Records and draws the actual path taken by the agent specified in ai_StatsTarget. The path displays in the color aqua, and only a certain length displays. The old path gradually disappears as a new path is drawn.

Values: 0 = Disable | 1 = Enabled

ai_DebugTacticalPoints

Displays debugging information on tactical point selection system (TPS).

ai_Locate

Indicates the position and some base states of specified objects. Pinpoints the position of the agents; its name; its attention target; draw red cone if the agent is allowed to fire; draw purple cone if agent is pressing trigger.

Values: none = Disabled | squad = Squad members | enemy = Enemies | groupID = Group members

ai_ProfileGoals

Records the time used for each AI goal (approach, run, pathfind) to execute. The longest execution time displays onscreen.

Default value: 0 = Disabled

ai_StatsDisplayMode

Gives information on the number of active AIs, full updates, and TPS queries for every frame.

Values: 0 = Hide | 1 = Display

ai_StatsTarget

Displays the current goal pipe, current goal, subpipes, and agent stats information for the selected AI agent. A long, green line represents the AI forward direction. A long, red or blue line represents the AI view direction if the AI is firing or not firing.

Values: *AI name*

ai_SteepSlopeAcrossValue

Indicates the maximum slope value that is borderline walkable across the slope. Zero (0.0) value indicates flat (no slope). Must be set to a value greater than ai_SteepSlopeUpValue.

Default value: 0.6

ai_SteepSlopeUpValue

Indicates the maximum slope value that is borderline walkable up the slope. Zero (0.0) value indicates flat (no slope). Must be set to a value smaller than ai_SteepSlopeAcrossValue.

Default value: 1.0

Other AI_Debug Variables

There are a number of other ai_DebugDraw console variables that can be accessed. Click the ... button at the bottom right corner of the console, and then enter `ai_debug` in **Search**.

Using AI Bubbles for Error Messaging

The AI Bubbles System is used to collect and display AI agent error messages for level designers. Debugging wrong behavior for an AI agent can take lots of time as it is difficult to track down which system is connected with the problem and which console variables need to be enabled to retrieve important information.

Game developers are encouraged to enter important error messages into the AI Bubbles system.

Error messages can be displayed as speech bubbles above an AI agent, displayed in a pop-up window, or displayed in the Console window.

The following console variables are used to control if and how alert messages are displayed:

ai_BubblesSystem

Enables or disables the AI Bubbles system.

Values: 0 =disable | 1 =enable

ai_BubblesSystemDecayTime

Specifies the number of seconds a speech bubble remains onscreen before the next bubble is displayed.

Units: seconds

ai_BubblesSystemAlertnessFilter

Specifies the type and level of messages displayed.

Values: 0 =none | 1 =logs | 2 =bubbles | 3 =logs and bubbles | 4 =blocking popups | 5 =blocking popups and logs | 6 =blocking popups and bubbles | 7 =all notifications

ai_BubblesSystemUseDepthTest

Specifies if the message will be occluded by game objects.

ai_BubblesSystemFontSize

Defines the font size of the message displayed.

Using AILog and AISignals Files

The AILog.log file can be used to log various AI agent events and the AISignals.csv file can be used to store AI signals for debugging purposes.

Note

These are only available if CryAISystem (and CryAction in the case for AISignals.csv) were built in Debug Mode.

The following AI events can be logged to the AILog.log file:

- AI Action started
- AI Action ended
- AI Action suspended
- AI Action resumed
- Signal received
- Auxiliary Signal received
- Goalpipe selected
- Goalpipe inserted
- Goalpipe reset
- RefPoint position set
- Stimulus registered
- AI System reset
- OnEnemyHeard
- OnEnemyMemory
- OnEnemySeen
- OnInterestingSoundHeard
- OnLostSightOfTarget
- OnMemoryMoved
- OnNoTarget
- OnObjectSeen
- OnSomethingSeen
- OnSuspectedSeen
- OnSuspectedSoundHeard
- OnThreateningSeen
- OnThreateningSoundHeard
- AI Signal executing
- Behavior constructor called
- Behavior destructor called
- Behavior selected

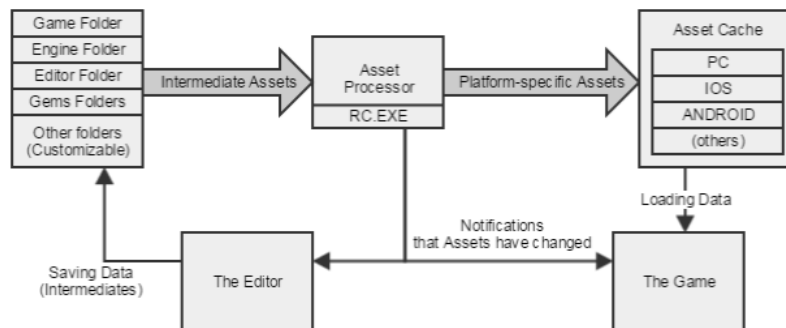
Asset Pipeline

The Asset Pipeline converts source art and other assets into platform-specific, game ready data. To prepare your game to ship, build all your game assets with the Asset Pipeline and package them with your game for your supported platforms.

The Asset Processor (AP) is a service that runs in the background and monitors a configurable set of input folders for changes in files. When changes are detected, it uses configurable rules to determine what needs to be done. The objective is to end up with game-ready versions of all assets for each platform and each game directory in a location called the asset cache. The asset cache is kept separate from your input directory and can be automatically rebuilt entirely from your source assets by the Asset Processor.

Note

The asset cache should not be added to your source control.



Folders that contain input assets are monitored for changes, with the game directory being the highest priority. This allows you to put assets in the game directory and have them override assets with the same path in Lumberyard or other folders with lower priority.

Each output directory in the asset cache represents a full image of all files (except for executables and related files) needed to run the game. The Asset Processor curates the directory to keep it up to date, ensuring that new files are ready to use in the game and Lumberyard Editor as soon as possible. Game runtimes load assets only from the asset cache and never directly from your input source folders.

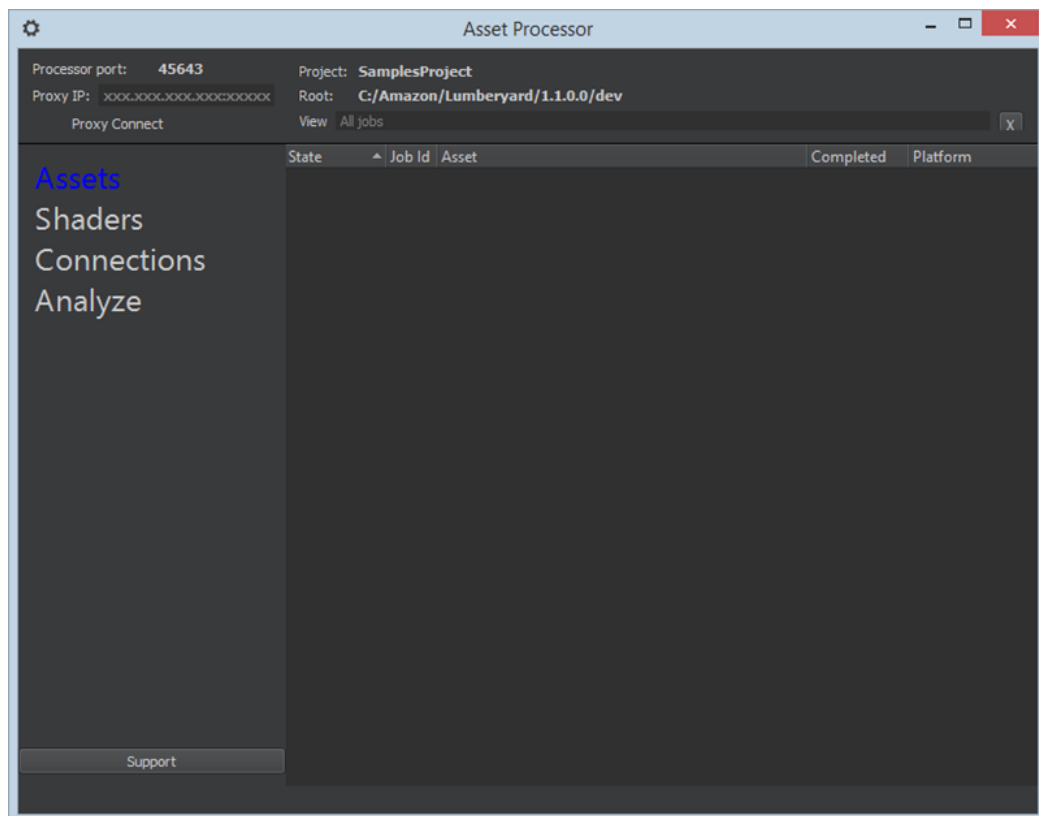
Topics

- [Asset Processor \(p. 124\)](#)
- [Live Reloading and VFS \(p. 125\)](#)
- [Shader Compiler Proxy \(p. 126\)](#)

- [Game Startup Sequence](#) (p. 126)
- [Missing Asset Resolver Tool](#) (p. 127)
- [Technical Information: Asset IDs and File Paths](#) (p. 127)

Asset Processor

The Asset Processor is a utility that automatically detects new or modified asset files, launches the Resource Compiler (`Rc.exe`), and then automatically processes the assets and places them in the cache. Afterward, the Asset Processor communicates with all running game or tool instances to inform them that the asset has been updated. The game can then reload the asset.



The Asset Processor can also allow games to run on other game platforms without deploying assets to that platform. Instead, the assets are accessed from the asset cache on a connected PC. With this feature, you can also run PC-based games using someone else's assets.

By proxying requests through itself, the Asset Processor can also communicate with an iOS or Android shader compiler server through a USB cable on iOS and Android.

The Asset Processor starts automatically if you run Lumberyard Editor with automatically maintained connections. It also restarts automatically if you modify any of the data files it needs to operate or get a new version of it. You do not have to close the Asset Processor when getting latest from source control. Nor do you have to wait for it to finish processing your assets before you start Lumberyard Editor. However, if you aren't using the game or Lumberyard Editor, you can quit the Asset Processor by right-clicking its icon in the Start bar notification icon area.

The Asset Processor can also serve files directly to running console games so that the assets don't have to be present on the game device. This is called virtual file system (VFS) and is required for live reloading to work on those platforms.

Configuration

The `AssetProcessorPlatformConfig.ini` configuration for the Asset Processor is stored in the root Lumberyard installation directory. If you need to perform any of the following tasks, see this file.

- Add new file types for the Asset Processor to feed to the Resource Compiler, to copy into the cache, or to alter existing file type rules.
- Alter the ignore list.
- Alter which platforms are currently enabled; the default is PC only.
- Add additional folders for the Asset Processor to watch. For example, if you want to share particle libraries and associated textures between projects.
- Alter what files trigger related files to be rebuilt. This is called metafile fingerprinting.

If you want to add game-specific overrides, add a file called `AssetProcessorGamePlatformConfig.ini` to your game assets directory. This file is read after the root configuration file. It can have additional game-specific settings for the folders to watch, the ignore list, platforms, and file types.

Batch Processing

`AssetProcessorBatch.exe` is a command line driven batch file version of the Asset Processor. When you run `AssetProcessorBatch.exe`, it compiles all assets for the current project and enabled platforms. It then exits with a code of 0 if it succeeded without errors. It can be used as part of your build system for automation.

`AssetProcessorBatch.exe` currently accepts the following command line parameters for overriding the default behavior.

- `/platforms=comma separated list`
- `/gamefolder=name of game folder`

Example usage:

```
AssetProcessorBatch.exe /platforms=pc,ios /gamefolder=SamplesProject
```

Debugging

Use the following techniques to debug Asset Processor issues:

- Quit Asset Processor and then restart it from the project or branch you're currently working in. You may need to close it in the system notification area to actually quit the Asset Processor. Pressing the close button merely hides it.
- Clear the asset cache by deleting the `Cache` folder located in the Lumberyard root directory when the Asset Processor is not running. Then restart it to rebuild all assets.

Live Reloading and VFS

On the PC platform, live reloading does not require virtual file system (VFS), since the PC that is running the game is presumably also running the Asset Processor.

On non-PC platforms, VFS is required for live reloading to work, because otherwise assets would need to be deployed onto the game device as part of live reloading, incurring platform-specific costs and different asset pipelines. VFS enables the same behavior across all platforms using the same

workflow. For debugging purposes, you can also enable VFS on a PC and point it at a remote Asset Processor to serve assets.

To enable VFS, you use the `bootstrap.cfg` configuration file.

The game runtimes and all tools can communicate with the Asset Processor through simple interfaces. Communication involves the following:

- Notification when assets are built and change, so as to reload them if possible.
- Request an immediate compilation of an asset, blocking until processing has completed.
- Request asset status, blocking until the status is known.
- Query the location of an asset source file, given an asset ID.
- Query the destination asset ID, given an asset source file name and path.

Not all asset types can live reload. If you are developing new asset types, keep the following guidelines in mind:

- When an asset loads, be prepared to substitute it for a temporary asset while it is compiling.
- If an asset is missing, query the status of the asset from the Asset Processor. This can determine whether the asset really is missing or whether it is in the queue for processing. Querying also moves the asset to the front of the queue for processing.
- If your asset is essential and it cannot live reload, use the blocking synchronous asset build request to make it build immediately. This moves the asset to the front of the queue and prevents the call from returning until the asset is compiled.
- Do not discard the original requested name when an asset is missing.
- Connect to the notification bus to learn when assets change and reload them when that happens.

Shader Compiler Proxy

Some mobile devices may be connected via a USB TCP/IP tunnel and may not have direct network access to a shader compiler server. The shader compiler proxy component in Lumberyard allows such devices to forward shader compiler requests through the Asset Processor connection.

This proxy connection only works for connecting to the shader compiler server on that protocol. It is not a general purpose network bridge or tunnel. To use the shader compiler proxy, open the `system_assetsplatform.cfg` file and modify the following values:

- `r_ShaderCompilerServer` = *IP address of shader compiler server* – Sets the location of the shader compiler server as seen from the computer running `AssetProcessor.exe`. For example, **localhost** could be used if both the Asset Processor and the shader compiler server are running on the same computer.
- `r_ShadersRemoteCompiler` = 1 – Compiles shaders remotely.
- `r_AssetProcessorShaderCompiler` = 1 – Routes shader compiler through the Asset Processor. If not set to 1, the device attempts to directly connect to the shader compiler server through the IP set.

Game Startup Sequence

Compiled Lumberyard games start up in the following sequence:

1. The game reads the `bootstrap.cfg` file, which must contain the following information at a minimum:

- Name of the game, and optionally, the name of the game DLL, if it differs from the game name.
 - Whether or not to connect to the Asset Processor on startup or listen for an incoming connection instead.
 - Whether or not to wait for an established connection before proceeding.
 - Whether or not to enable the virtual file system (VFS), which allows you to read assets remotely from a connected computer instead of having to deploy them to the game device. This also is required for live reloading to function on non-PC platforms.
 - Which kind of assets to load. For example, you could configure the Android runtime to load `es3` assets, or `pc` assets, or `metal` assets. This determines which directory the game looks in for the assets so that the appropriate directory is also used for VFS.
2. The `lyconfig_default.xml` file is read.
 3. VFS is started and enabled. All file access then goes through the VFS system. Besides the `bootstrap.cfg` file, executable files, DLL files, and associated platform files, nothing else needs to be deployed to the device. Instead, they can all be accessed remotely.
 4. The `system_game_platform_assets.cfg` file is read, where `assets` are the assets specified in the `bootstrap.cfg` file.

Missing Asset Resolver Tool

The Missing Asset Resolver helps you find asset files in a level that have been moved, and will display where the missing file used to be located and where it is now located.

To use the Missing Asset Resolver

1. In Lumberyard Editor, choose **View, Show Console** to open the **Console** window.
2. In the **Console** window text box, type `ed_MissingAssetResolver 1`.
3. In Lumberyard Editor, choose **View, Open View Pane, Missing Asset Resolver**.
4. Click **File, Open**, select the level that contains the missing asset, and click **Open**.
5. In the **Missing Asset Resolver** window, right-click the applicable asset, and then click **Accept all resolved files**.

The asset file is now referenced from its correct location.

Technical Information: Asset IDs and File Paths

Consult this section if you are a developer who needs to port older game code or develop new code or tools.

Asset IDs and File Paths

All files accessed for the game runtime go through an interface that supports aliasing of file paths by name. For example, the alias `%ROOT%` refers to the root directory where the `bootstrap.cfg` file is located. If you need to open a file in the root directory, do not go to the root directory or use the current working directory. Instead, use the file name, such as `%root%/filename.cfg`. The various Lumberyard subsystems correctly resolve the alias.

Other aliases available include the following:

- `%log%` – For storing forensic data, such as crashes, logs, traces, performance drops, and unit test output.

- `%cache%` – For storing data that can be cleaned out at any time and does not need to persist.
- `%user%` – For storing data that needs to persist between users. Note that some platforms may back up this data to the cloud, such as for user preferences.
- `%assets%` – The location of the asset cache. If no alias is specified, this is assumed, so it is almost never necessary to specify this.
- `%devroot%` – The root of your development tree where the editable `engineeroot.txt` file is located. This file is shared by many game projects and used by the editor and other tools.
- `%devassets%` – The root of your source asset directory for your game, which is used by the editor and tools.

The following are examples of asset IDs:

```
textures/mytexture.dds
objects/rain/droplet.cgf
gamedata.xml
levels/mainlevel/mainlevel.xml
```

The following examples are file paths and not assets IDs:

```
%assets%/textures/mytexture.dds
%root%/system.cfg
C:\dev\mystuff.txt
\\networkdrive\somefile.dat
```

The following example is invalid as it mistakenly assumes that the asset cache has the same name as the game and that it is a child folder of the root directory. This isn't true on all platforms:

```
%root%/GameName/textures/mytexture.dds
```

When referring to assets during runtime, always use the asset ID. Do not prefix asset IDs with `%assets%` or the game name, and do not concatenate them with custom strings. Treat asset IDs as immutable data that is not a string and refers to a specific asset. For example, you would store `textures/mytexture.dds` and not `gems/rain/mytexture.tif`.

You can use the `FileIO` interface, which is accessible through `gEnv->pFileIO`, to resolve aliased names to full paths, if you want to point to an external disk loading tool such as `Qt QFile()`. This should almost never be necessary during runtime. If you do use this, however, your system cannot use remote asset access nor support live reloading.

Converting Asset IDs to Full Paths

If you are writing a new editor tool or porting an existing one from a legacy system, keep in mind the separation between game code and editor code. Game code cannot manipulate asset IDs, and therefore it is invalid to retrieve the game path or concatenate game names with path names. The game code and game modules also have no access to source control, so relying on the game to find out where to save files will not work.

Instead, develop your editor code in such a way that the editor decides where files are saved, and optionally loaded from, and correctly interfaces with source control and the asset processing system. (Source control and asset processing are overhead that is governed by the editor tool, not the game.)

The following utilities and guidelines are provided to make this easier:

- Store only asset IDs for all source assets. For example, if you are writing a file that refers to other files, do not store `C:\lumberyard\dev\MyGame\myasset.txt` in the file's data, for example. Instead, just store `myasset.txt`, its asset ID.
- If you are in an editor tool, link to `EditorCore`, and then do the following:
 - `#include <PathUtil.h>`
 - Call `Path::FullPathToGamePath(string)` to convert any full path into a game asset ID automatically.
 - Call `Path::GamePathToFullPath(string)` to convert any asset ID into a full source asset name.
 - Call `Path::GetEditingGameDataFolder` to see where to save files that do not exist yet, such as for a **File Save** dialog.
- If you are working in a new system that does not rely on legacy systems, you can use an EBus, which has the same functionality as described above. For more information about the EBus, see [Event Bus \(EBus\)](#) in the [Amazon Lumberyard Developer Guide](#).
 - `#include <AzToolsFramework/API/EditorAssetSystemAPI.h>`
 - Call EBus messages `ConvertFullPathToRelativeAssetPath` and `ConvertRelativeAssetPathToFullPath` to convert back and forth.
 - Call EBus messages `GetAbsoluteDevGameFolderPath` to get the game directory for **File Save** dialogs. Use this only when you do not have an asset ID already, such as in the case of new files.

As an example, the following steps code a tool that provides a list of all available assets of type `sprite`:

To make a list of available sprite assets

1. Use the `gEnv->pCryPak` file-finding functions to search for all asset IDs. Usually, since `%assets` is assumed, just the directory name or extensions are all that is required, but aliases are accepted.
2. Once you have the asset ID list, call `GamePathToFullPath` or `ConvertRelativeAssetPathToFullPath` to convert the list to full source names.
3. Display the appropriate name in the UI, either the real source name or the output name.
4. When a user wants to edit the file, use the source name to check it out from source control.
5. When a user saves the file, make sure to write it to the source name, not the target name.
6. When the asset compiler recompiles the asset, it notifies you using the asset ID. Make sure you compare the incoming name to this asset ID.

Live Update Messages

If you are on a PC or you are connected to VFS, you can listen for live update messages from the Asset Pipeline and reload your assets when you get them.

To do this, do the following:

- `#include <IAssetSystem.h>`
- Subscribe a listener to the `AssetSystemBus`. Subscribers connect via the `crc` of the file extensions they are interested in. Search for "AssetChanged" to see examples in various systems.

Here is an example: `BusConnect(AZ_CRC("dds")); // be notified of all DDS file changes.`

Once you get your live reload notification, it contains an asset ID. Consider queuing the request for later if you are in a thread-sensitive module.

Audio System

Lumberyard uses an audio translation layer (ATL) to interface between Lumberyard and third party audio middleware, so you can change your audio implementation without affecting the game logic. ATL events trigger in-game sounds, which then trigger audio implementation events that notify the audio middleware to play the specified sounds.

Lumberyard supports Audiokinetic Wave Works Interactive Sound Engine (Wwise), an audio pipeline solution with which you can create compelling soundscapes for your game.

Lumberyard also supports a free “compact” version called Wwise LTX. The runtime SDK for it comes pre-configured with Lumberyard.

The audio system consists of the following elements:

- Sound banks – Compiled sound files and metadata
- Project files – All files related to your project for the middleware authoring tool
- Game audio libraries – XML files that define the mappings (both global and level-specific) between game-side ATL audio controls and middleware data

For information on audio entities, see [Audio Entities \(p. 451\)](#).

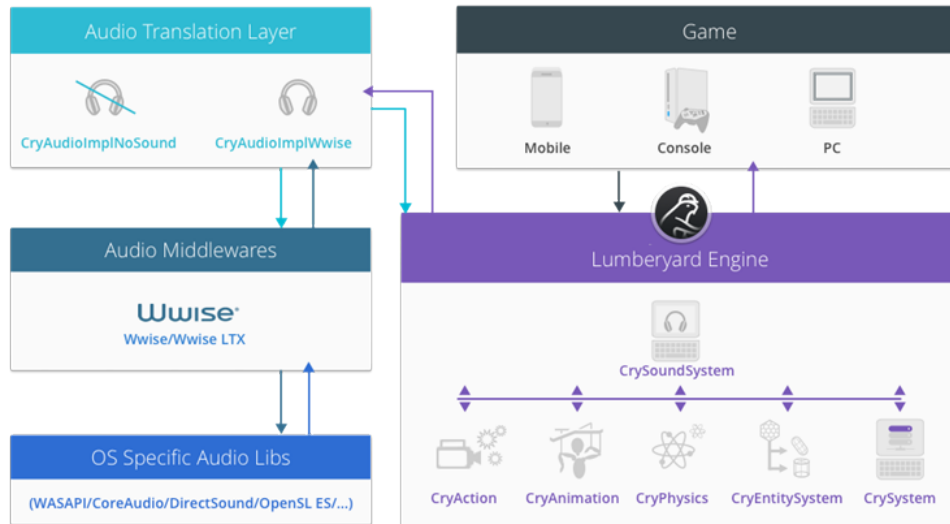
Topics

- [Audio System Architecture \(p. 131\)](#)
- [Installing Audiokinetic Wwise LTX \(p. 131\)](#)
- [Using the Audio Controls Editor \(p. 132\)](#)
- [ATL Default Controls \(p. 135\)](#)
- [Audio PlayTriggers and StopTriggers \(p. 136\)](#)
- [Obstructing and Occluding Sounds \(p. 137\)](#)
- [Audio Flow Graph Nodes \(p. 139\)](#)
- [Adding Ambient Sounds to Levels \(p. 139\)](#)
- [Adding Reverb Effects to Levels \(p. 142\)](#)
- [Adding Collision Sounds to Levels \(p. 144\)](#)

- [Adding Sound to Trackview Sequences \(p. 145\)](#)
- [Adding Sound to Animations \(p. 145\)](#)
- [Audio Console Variables Commands \(p. 147\)](#)

Audio System Architecture

Lumberyard Audio EcoSystem



The Lumberyard Audio system consists of three largely independent layers:

CAudioSystem: Represents the Audio system interface to the outside world. It holds methods for looking up or reserving IDs for various objects and the `PushRequest` method, which is the only way to request an action from the Audio system. This class contains the message queues and handles the scheduling and dispatch of the incoming requests. It also manages the Main Audio thread.

CAudioTranslationLayer: Keeps track of the Audio system's current state, including registered `AudioObjects`, `AudioListeners`, and active `AudioEvents`, and processes the requests submitted through the `PushRequest` method.

IAudioSystemImplementation: Represents an interface to an audio middleware system. While processing incoming requests, `CAudioTranslationLayer` calls the appropriate method of `IAudioSystemImplementation` and, if the call succeeds, records all of the resulting changes in the `AudioSystem` state.

Installing Audiokinetic Wwise LTX

Lumberyard includes an exclusive, free version of the Audiokinetic Wwise audio system for PC games: Wwise LTX. Sound designers and composers can use Wwise LTX to work independently from the engineering team and author rich soundscapes for your games.

If your game requires the feature set of the full version of Wwise, Lumberyard provides a simple migration path. By replacing the Wwise LTX SDK with the full version of the Wwise SDK and rebuilding

your game, you can take advantage of the advanced features offered by Audiokinetic's full product range.

To access the Wwise LTX documentation once Wwise LTX is installed, press the F1 key. Alternatively, click **Help**, **Wwise Help** in the application menu.

Installing Wwise LTX

To author sounds with Wwise LTX for your game, you must do the following:

To install Audiokinetic Wwise LTX

1. Run Lumberyard Setup Assistant, located at `engine_root_folder\SetupAssistant.bat`.
2. Click **Install software**.
3. Look for the **Audiokinetic Wwise LTX Authoring Tool** entry, and click the **Install it** link. This will install the Wwise Launcher and run it.
4. If prompted to sign in to your Audiokinetic account, enter your details and click **Sign In**, or else click **Skip sign in**.
5. On the next page, you can select the desired installation components and settings for Wwise LTX, or else accept the default. Once done, click the **Install** button.
6. If prompted with License Terms, you can review the End-User License Agreement. Once done, click the **Accept** button. Installation will commence.
7. Once the installation has successfully completed, click the **Launch Wwise (64-bit)** button under the Wwise LTX entry to run the Authoring Tool. If desired, click the wrench icon to create a desktop shortcut for later use.
8. Close the Wwise Launcher and return to Lumberyard Setup Assistant. It should now show that Wwise LTX is installed.

Running the Wwise LTX Authoring Tool

To run the Wwise LTX Authoring Tool, you must first open or create a project. The SamplesProject includes a Wwise LTX project you can use.

To run the Wwise LTX Authoring Tool

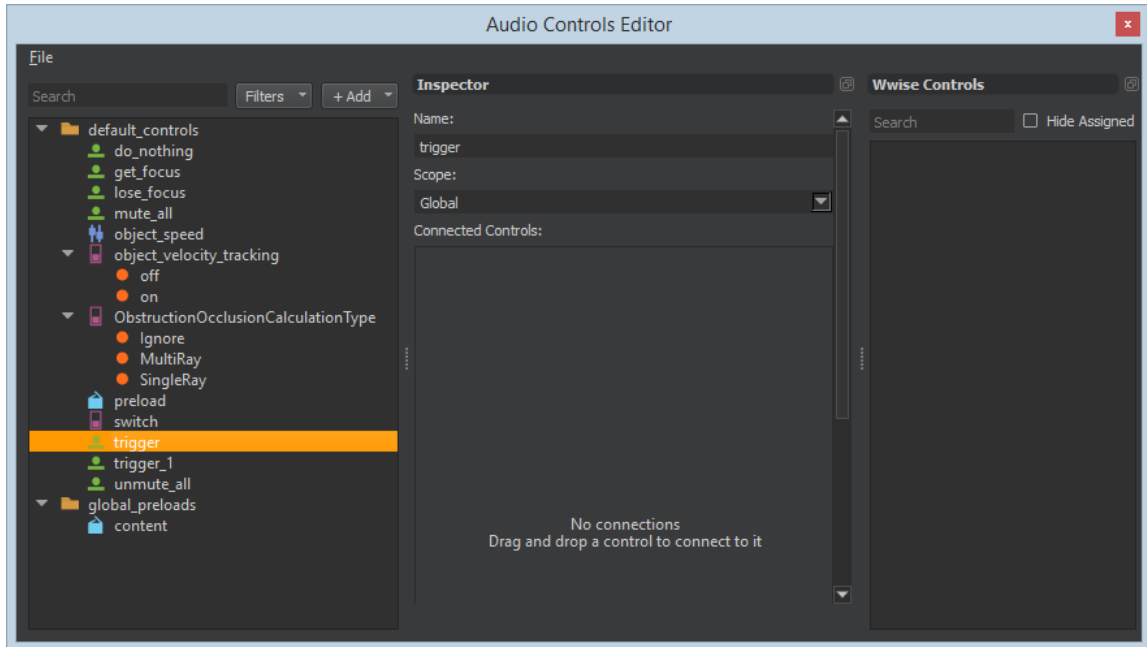
1. Run **Wwise Launcher** from the **Programs** menu.
2. Click the **Wwise** tab, then in the Wwise LTX installation, click **Launch Wwise (64-bit)**. Alternatively, if you created a desktop shortcut earlier, you may use that.
3. If this is your first time running Wwise LTX, you will be prompted again to review and accept the End-User License Agreement (EULA). After accepting the EULA, click **Open Other**.
4. Browse to the `.wproj` file located in `engine_root_folder\SamplesProject\Sounds\wwise_project\` and click **Open**.
5. Alternatively, if you are not using SamplesProject, you may create a new Wwise LTX project by clicking **New**. The following page titled **Using the Audio Controls Editor** will contain more information about setting up Wwise for your game project.

Using the Audio Controls Editor

All actions, events, and parameters from your game are communicated to the Audio system using Audio Translation Layer (ATL) controls that are mapped to one or more controls inside your selected

middleware. The connection between the ATL controls and the middleware controls, as well as the creation of the controls themselves, are done using the Audio Controls Editor.

Select **View, Open View Pane, Audio Controls Editor**. The editor consists of three panels: the ATL controls panel, Inspector panel, and middleware-specific Controls panel.



The following tables list various controls and properties available in the different panes of the Audio Controls Editor. The controls available in the middleware Controls panel are by definition middleware-specific.

Audio Controls Table

Audio Control	Description
Trigger	Containers that execute all audio controls that are connected to them. You can preview a trigger by right-clicking it and then selecting Execute Trigger , or by pressing the keyboard spacebar.
RTPC	Real-Time Parameter Control (RTPC) is typically a floating-point variable that is updated continuously over time to alter a parameter's value, which the audio middleware can use to drive corresponding effects.
Switch	A variable that can be in one of several states that can be set using Flow Graph or by code. For example, a SurfaceType switch might have values of <i>Rock</i> , <i>Sand</i> , or <i>Grass</i> .
Environment	Environments can be set on areas such as AreaBoxes, AreaShapes, and AreaSpheres, which allow for driving environmental effects such as reverb and echo.
Preload	A preloaded sound bank, which is an audio file that includes packaged audio data that contains both a signal and metadata.

The Inspector panel allows you to edit all the properties of the control currently selected in the ATL controls panel, including making connections to any matching middleware-specific controls.

Inspector panel table

Property	Description
Name	The name of the control. This can also be edited in the ATL controls panel.
Scope	Controls can exist for a global or on a per-level scope. A control with a global scope exists as long as the game is running and regardless of whether the control is used in the current level. When a specific level is defined as the scope, they exist only when that level is loaded. This setting is very useful in low-memory systems because controls are only loaded in levels where they are needed.
Auto Load	If Auto Load is set, the elements preloaded with this control will be reference counted—that is, only one copy of them is created that is shared between all users. (Only available for preloads)
Preloaded Soundbanks	The soundbanks connected with a preload can be different for different platforms. Different soundbanks can be added to different groups and then in the Platforms field you can choose which group to load for each platform you are targeting. (Only available for preloads.)
Platforms	Allows you to set which group of soundbanks to load for each platform. You can share a group between several platforms. (Only available for preloads.)
Connected Controls	Contains all the middleware controls connected to your control.

To create new connections between ATL controls and middleware-specific controls, just drag the control from the middleware controls panel to the **Connected Controls** area of the Inspector panel. A middleware control can also be dragged directly to the ATL controls panel; doing so creates a new control with the name of the middleware control and automatically connect both of them.

Note

After creating a new control, in the Audio Controls Editor, click **File, Save All**, and then click **Audio, Refresh Audio** to be able to preview the control.

Using Audiokinetic Wwise LTX

If you use Audiokinetic Wwise LTX, the project must be in a location where the Audio Controls Editor can detect it. The `.wproj` project is located at `\[game folder]\Sounds\wwise_project\`. You will need to configure Wwise LTX to build soundbanks to the following location: `..\wwise\`.

With the Audio Controls Editor tool, you can edit the following audio-related data files:

- `\SamplesProject\Libs\gameaudio\wwise\config.xml`
- `\SamplesProject\Libs\gameaudio\wwise\global_preloads.xml`
- `\SamplesProject\Libs\gameaudio\wwise\default_controls.xml`

Soundbanks for Audiokinetic Wwise LTX are located in the `\Sounds` directory:

- `\SamplesProject\Sounds\wwise\Content.bnk`
- `\SamplesProject\Sounds\wwise\Init.bnk`

ATL Default Controls

The Lumberyard audio system uses an Audio Translation Layer (ATL) to control when and how sounds play in your level. Wwise LTX controls are then connected to the ATL controls. The following ATL default controls are automatically created by the Audio Controls Editor and are located in the `default_controls` folder:

- `do_nothing`
- `get_focus`
- `lose_focus`
- `mute_all`
- `object_speed`
- `object_velocity_tracking`
- `ObstructionOcclusionCalculationType`
- `unmute_all`

do_nothing control

You can define both a `PlayTrigger` and a `StopTrigger`. The `do_nothing` control is used as a blank event in cases where `StopTrigger` functionality should not be used. This trigger should not be connected to any event in your audio middleware.

get_focus control

This trigger is called when the application window in Lumberyard Editor gains focus.

lose_focus control

This trigger is called when the application window in Lumberyard Editor loses focus.

Note

If you don't want to pause or resume audio when gaining or losing focus, use the console command `s_ignorewindowfocus = 1`. This bypasses the `get_focus` and `lose_focus` events from being called when gaining or losing focus.

mute_all control

This trigger is called when the **Mute Audio** button is selected, located on the lower menu bar of Lumberyard Editor.

object_speed control

This is an RTPC control that is updated according to the speed of the associated entity in the level. The calculation of the speed can be enabled on a per-entity basis with the `object_velocity_tracking` control.

object_velocity_tracking control

This is a switch used to enable or disable the calculation of the `object_speed` value on a per-entity basis. This switch does not need to be connected to the audio middleware as it is communicating Lumberyard-specific data.

ObstructionOcclusionCalculationType control

This is a switch used to set the obstruction and occlusion calculation method of an entity. The switch state values are `Ignore`, `SingleRay`, and `MultiRay`. This switch does not need to be connected to the audio middleware as it is communicating Lumberyard-specific data.

unmute_all control

This trigger is called when the **Mute Audio** button is deselected, located on the lower menu bar of Lumberyard Editor.

Audio PlayTriggers and StopTriggers

You can define both PlayTriggers and StopTriggers for entities in your level. These are accessed from the Rollup Bar in the **Properties** panel for the entity.

Placing Triggers in Game

In order to hear a sound in the game, first place the audio entity that executes the audio trigger during gameplay in the level. For each level, we recommended creating a dedicated audio layer that contains all your audio data. In the Rollup Bar, click the **Layer** tab, click the **New layer** icon, and then name the layer. Select the audio layer to ensure that all entities that you are placing in the level are included in this layer.

On the **Objects** tab in the Rollup Bar, click **Audio, AudioTriggerSpot**, then drag and click to place the entity in the level.

PlayTrigger Set

On activation, this executes the PlayTrigger and, on deactivation, stops the PlayTrigger. The audio system automatically stops the PlayTrigger when the corresponding StopTrigger is activated.

This can be useful, as it automatically stops sounds without the need to create any additional stop functionality inside of your audio middleware. This also assures that any looping sounds are stopped when the associated entity is disabled.

You can also bypass the automatic stopping of a StartTrigger by using the default `do_nothing` ATL control. When it is set on the StopTrigger, the audio system behaves as explained under the Both Triggers Set section. As this control is not connected to any functionality within your middleware, it will not affect the audio, but bypasses the automatic stop functionality of the audio system.

For example, when setting an audio trigger on a shooting animation of a gun, you would want to hear the end of the gunfire even after the animation has finished. However, if no trigger has been set in the StopTrigger field, the PlayTrigger would be terminated and therefore the sound of the gunfire ending would be cut off. To prevent this, place the `do_nothing` control in the StopTrigger field. This bypasses the automatic stopping functionality and lets the PlayTrigger execute completely.

Note

Remember that, with the above setup, any looping sounds that you have set as a PlayTrigger will not be stopped.

StopTrigger Set

On deactivation, the StopTrigger is executed and, on activation, nothing happens. As no trigger is defined under the PlayTrigger, nothing happens when the PlayTrigger is executed. However, if a trigger is set for the StopTrigger, it plays when the StopTrigger executes.

An audio trigger can also execute the playback of audio in your middleware when it is placed as a StopTrigger.

Both Triggers Set

With this configuration, the StartTrigger is executed on activation. The StopTrigger is activated upon deactivation and without stopping the StartTrigger. This is because the audio trigger has been defined as a StopTrigger.

If you need to stop the PlayTrigger with another audio trigger that is set as a StopTrigger, then you need to set up stop functionality inside of your audio middleware.

As a general rule, it is always useful to use the automatic stop behavior contained in the audio system when you just want to simply stop a sound on entity deactivation or on the ending of an animation. When creating more complex events, such as fade outs or triggering additional audio samples while stopping the StartTrigger, create the stop functionality inside your audio middleware and set the connected ATLControl as the StopTrigger.

Obstructing and Occluding Sounds

Sound *obstruction* means that the direct path to the audio is blocked but the sound might still be audible due to the sound reflecting off the obstruction and other objects. Sound *occlusion*, on the other hand, refers to the degree to which sound is lost or affected by intervening objects. You can enable different obstruction and occlusion settings for the **AudioTriggerSpot**, **AudioAreaEntity**, and **AudioAreaAmbience** object types. Using these settings correctly helps you to create a game world where sound is realistically filtered and attenuated according to the surrounding environments.

You can set the **SoundObstructionType** property for the **AudioTriggerSpot**, **AudioAreaEntity**, and **AudioAreaAmbience** in their respective properties panels in the **Rollup Bar**. All audio object types default to **Ignore** as their **SoundObstructionType** setting.

Obstructing Sounds

Lumberyard uses raycasting, or ray-to-surface intersection testing, to get information about the objects with which the line intersects. If the occlusion value of a raycast's center ray differs from the average of the occlusion values of the outer rays from the same raycast, Lumberyard applies obstruction to the sound source. Therefore, obstruction is calculated only when the **SoundObstructionType** is set to **MultipleRays** on the object type, since a single ray does not provide enough information to differentiate between obstruction and occlusion.

Obstruction is applied to the sound after occlusion and in addition to it. If the center ray of a raycast has reached the listener without being blocked, and the outer rays are fully or partially blocked by game objects, then the obstruction value is set to zero and only the occlusion value is positive. In addition, obstruction is only applied to the dry signal; it has no effect on the signal sent to the environment auxiliary buses.

Obstruction is also affected by the distance of the raycasting entity to the listener. As the distance increases, the obstruction value decreases and the difference is transferred to the occlusion value. This reflects the fact that, with increasing distance, the contribution of the direct line-of-sight sound path in the overall sound perception becomes progressively smaller.

The console variable `s_FullObstructionMaxDistance` sets the maximum distance after which the obstruction value starts to decrease with distance. For example, `s_FullObstructionMaxDistance = 5` means that, for the sources that are farther than 5 meters away from the listener, the obstruction value is lower than the actual value calculated from the raycast. In this case, an object 10 meters away has half the obstruction value of the similarly obstructed source located 5 meters away.

Sound Obstruction for Surface Types

You can define how much each different material type affects the sound passing through it. The `sound_obstruction` physics property is a value between 0 and 1. For each raycast from a sound source, the ray's occlusion value increases by the `sound_obstruction` value of each surface it intersects.

Values for each surface type can be set in the `\Libs\MaterialEffects\SurfaceTypes.xml` file. The exact effect that this value has on the audio content of your game is defined in your specific audio middleware.

For a material with `sound_obstruction = 0.5`, the maximum obstruction and occlusion value that is reached in the game is 0.5. Therefore, if the sound is fully occluded by one object with this surface type, the occlusion value passed to the middleware is 0.5. If the sound is also obstructed, the combined values of obstruction and occlusion would be summed to 0.5. However, their sum would never exceed this value, as it is defined as the maximum obstruction or occlusion value in the material's `sound_obstruction` property.

Occluding Sounds

Occlusion is applied to a sound source that is completely or partially hidden from the listener by the other game object(s).

A nonzero occlusion value is set for a sound source whenever at least one ray that is cast from that source encounters a surface with non-zero `sound_obstruction` value. The `sound_obstruction` values from the surfaces struck by the ray are accumulated, and the total values are averaged over time for each ray to produce this ray's occlusion value, as shown in the ray label enabled with `s_DrawAudioDebug` flag. With the **SingleRay** selected for **SoundObstructionType**, the audio object occlusion value is equal to its only ray's occlusion value. With **MultipleRays** selected for **SoundObstructionType**, the audio object occlusion value is the average of the occlusion values for all the rays.

You can use the console variable command `s_OcclusionMaxDistance` to set a maximum distance beyond which the sound obstruction and occlusion calculations are disabled. For example, for `s_OcclusionMaxDistance = 150`, Lumberyard calculates the obstruction and occlusion values for every active audio object with **SoundObstructionType** set to **SingleRay** or **MultipleRays**, providing they are located within 150 meters of the sound's listener.

Raycasts

When Lumberyard performs a raycast, it can calculate the occlusion and obstruction values either synchronously or asynchronously. In synchronous calculations, all occlusions of an individual ray are available immediately in the same frame as the one that requested the raycast. In asynchronous calculations, the individual ray data is received over the next few frames and processed once all of the rays have reported back. Synchronous raycasts are much more responsive, but they also require more processing resources and can hurt performance if a large number of raycasts are performed in a single frame. In order to save resources and avoid performance hits, Lumberyard switches between synchronous and asynchronous raycasts based on the distance between the sound source and the listener. For the sources close to the listener, Lumberyard uses synchronous raycasts to provide a maximum responsive environment. For sources further away, asynchronous raycasts are used.

Debugging Raycasts

The `s_DrawAudioDebug` console variable has three flags that show you the values calculated by the raycasts:

- **b** – Shows text labels for active audio objects, including obstruction (`Obst`) and occlusion (`Occl`) value.
- **g** – Draws occlusion rays.
- **h** – Shows occlusion ray labels.

Audio Flow Graph Nodes

There are a number of Flow Graph nodes you can use to control different aspects of the Audio system. For more information, see [Audio Nodes \(p. 541\)](#).

Adding Ambient Sounds to Levels

Lumberyard has two audio entities that you can use to add ambient sounds to levels as well as [Adding Reverb Effects to Levels \(p. 142\)](#)—**AudioAreaAmbience** and **AudioAreaEntity**. Both entities are linked to a specified shape in a level that defines the area in which ambient sounds are triggered from.

You use these two entities to set multiple attributes with which you can define a **PlayTrigger** and **StopTrigger**, an environment, and a radius around the shape where your ambient sound starts to fade in and out.

To make use of the **AudioAreaAmbience** or **AudioAreaEntity** entity in a level, you must first create a new shape.

Note

The distance that is output by the **AudioAreaAmbience** and **AudioAreaEntity** entities is always scaled from 0 to 1 from the maximum range set in the **RtpcDistance** property. Therefore, the range of the RTPC (real time parameter control) value used by your middleware needs to be only 0 to 1.

To define an area shape for an audio entity

1. In the **Rollup Bar**, on the **Objects** tab, click **Area**.
2. Under **Object Type**, select either **AreaBox**, **AreaSphere**, or **Shape**. Then do the following:
 - In the case of **AreaBox**, click in your level to place it, and then under **AreaBox Params**, specify values for **Width**, **Length**, and **Height**.
 - In the case of **AreaSphere**, click in your level to place the shae, and then under **AreaSphere Params, Radius**.
 - In the case of **Shape**, click in your level to create points that define the boundaries of the shape. When finished, double-click the last point to complete the shape.

Note

The event listener, which is attached to the player character by default, needs to be inside an area shape for a sound to play. Set the shape's **Height** value to at least 15 to ensure there is room for the ambient sound to play even if the character jumps.

You can change an area shape by adding, removing, and moving points.

To edit a shape

1. Select the shape in your level.
2. In the **Rollup Bar**, under **Shape Parameters**, click **Edit Shape**.
3. ect the applicable point on the shape and do the following:

- Drag the point to move it to another location.
 - Press the **Delete** key to remove the point.
 - Press **Ctrl+click** to add a point to the shape
4. When done, under **Shape Parameters**, click **Reset Height** to flatten the shape. This is useful when creating shapes over hilly terrain.

Note

If the **Follow Terrain** option was not selected, the area shape that you created may be located under the terrain. If so, select the Move tool and drag the shape up by clicking on the yellow Z-axis arrow and dragging it up.

Setting Up the AudioAreaAmbience entity

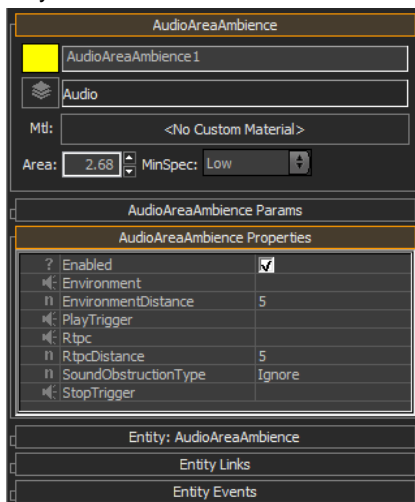
The **AudioAreaAmbience** entity is the main audio entity for defining which ambient sound should play and how without the need to use the Flow Graph editor for advanced sound effects or behaviors.

To set up and link the AudioAreaAmbience entity

1. In Rollup Bar, on the **Objects** tab, click **Audio**.
2. Under **Object Type**, click **AudioAreaAmbience**. Then click in your level to place the object type.
3. Under **AudioAreaAmbience Properties**, select the applicable property, and then click the folder icon that appears on the right. Do this for the **PlayTrigger**, **StopTrigger**, and **Rtpc** properties.
4. In the **Choose** window, expand **default_controls**, select an ATL control to use for the property, and then click **OK**.
5. For **RtpcDistance**, enter a value that represents the distance in meters at which sounds starts fading in volume for the player.
6. Under **Entity Links**, select **AudioAreaAmbience** and click the **Pick Target** button. The object type is now linked to the shape that you created earlier.
7. Press **Ctrl+G** to test the ambient sound.

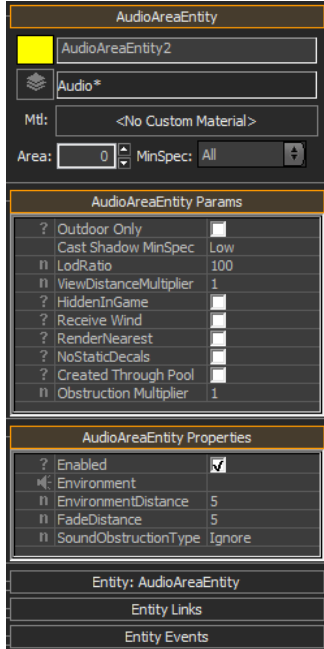
The selected **PlayTrigger** control is called whenever the character is the distance away from the shape specified by the **RtpcDistance** value.

As the character moves closer towards or further away from the shape, the volume of the ambient sound increases or decreases in volume in accordance to the setup in your audio middleware. As long as your character is within the area shape, the sound plays without volume attenuation.



Setting Up the AudioAreaEntity entity

The **AudioAreaEntity** entity functions like the **AudioAreaAmbience** entity but requires manual setup in the Flow Graph editor to trigger the ATL controls. This extra step gives you access to multiple parameters and more advanced setup possibilities than the **AudioAreaAmbience** entity.

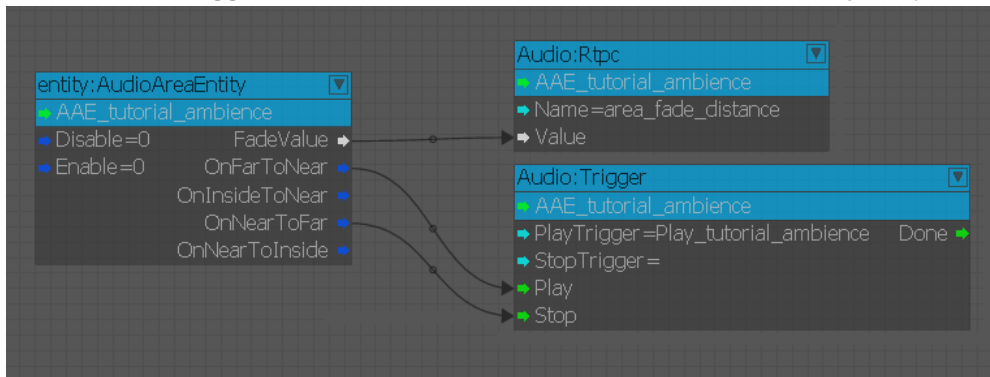


The **AudioAreaEntity** entity includes a **FadeDistance** parameter, which behaves like the **RtpcDistance** parameter except that it can be connected to any object in the Flow Graph editor and not simply to a default RTPC audio control.

Notice that the **Properties** panel does not include **PlayTrigger** or **StopTrigger**, as they are manually set up in the Flow Graph editor.

The **AudioAreaEntity** flow graph node does not have any playback functionality itself; Instead it triggers output when the character enters or leaves either the area shape or its outer values as defined by the fade distance. It also sends out a value that can be used to control any RTPC audio control with the **Audio:Rtpc** node. To use the **AudioAreaEntity** to enable playback of an ambient sound for your area, you need to add the **Audio:Trigger** and **Audio:RTPC** nodes to the flow graph.

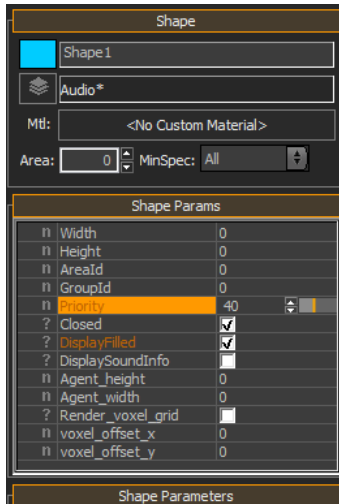
After adding both nodes to the flow graph, right-click on each and select **Assign Selected Entity**. Now both the **Audio:Trigger** and the **Audio:RTPC** are set for the **AudioAreaEntity** entity.



A **StopTrigger** is not needed as the audio system automatically stops the **PlayTrigger** control if no **StopTrigger** is assigned. Ambient sounds that are set up using the **AudioAreaEntity** entity do not play by default as they are triggering the controls in the flow graph. To preview the **AudioAreaEntity** when not in game mode, click **AI/Physics** at the bottom of Lumberyard Editor.

Using Shape Priorities

When using multiple shapes in a level, you can set shape priorities to define how audio behaves when a character moves from one shape to another. You can set the priority per shape under **Area**, **Object Type**, **Shape** in the **Rollup Bar**.



When transitioning from one shape to another, the shape with the higher priority overrides the **RtpcDistance** and **FadeDistance** properties for lower priority **AudioAreaAmbience** and **AudioAreaEntity** entities respectively.

Note

With helpers enabled, pressing the spacebar on your keyboard displays pivots for all entities in a level. You can also do this to simplify the selection of areas that are nested together.

Select the **DisplaySoundInfo** check box to indicate any sound obstructions for the sides of an **AreaBox** or a **Shape**. Sound-obstructed sides appear in red and do not calculate ambient sound updates for that segment. Non-obstructed sides are displayed in green and do calculate ambient sound updates.

Adding Reverb Effects to Levels

Lumberyard has two audio [entities](#) that you can use to add reverberation effects (as well as ambient sounds) to a level —**AudioAreaAmbience** and **AudioAreaEntity**. Both entities are linked to a specified shape in a level that defines the area in which reverb effects are triggered from.

To setup Wwise LTX for reverb effects

1. In Audiokinetic Wwise LTX, click the **Audio** tab in **Project Explorer**.
2. Under **Master-Mixer Hierarchy**, create an auxiliary bus.
3. In the **Auxiliary Bus Property Editor**, on the **General Settings** tab, assign a **Wwise RoomVerb** effect to the bus.
4. Click **Edit** to edit the settings of the effect.

5. Under **Actor-Mixer Hierarchy**, for any sounds or sound containers that you want to run through the effects bus, do the following:
 - a. On the **Property Editor** pane, click the **General Settings** tab and select **Game-Defined Auxiliary Sends**.
 - b. Select the **Use game-defined auxiliary sends** check box.
 - c. Save the project and select **Generate soundbank**.
6. Open Lumberyard Editor, and then click **View, Open View Pane, Audio Controls Editor**.
7. In **Audio Controls Editor**, click **Add, Environment**.
8. Link the Wwise auxiliary bus to the audio translation layer (ATL) environment.

To setup Wwise for reverb effects

1. In AudioKinetic Wwise, click the **Audio** tab in **Project Explorer**.
2. Under **Master-Mixer Hierarchy**, create an auxiliary bus.
3. In the **Auxiliary Bus Property Editor**, on the **Effects** tab, assign a **Wwise RoomVerb** effect to the bus.
4. Click **Edit** to edit the settings of the effect.
5. Under **Actor-Mixer Hierarchy**, for any sounds or sound containers that you want to run through the effects bus, do the following:
 - a. On the **Property Editor** pane, click the **General Settings** tab and select **Game-Defined Auxiliary Sends**.
 - b. Select the **Use game-defined auxiliary sends** check box.
 - c. In **SoundBank Manager Layout (F7)**, select **Generate soundbank**.
6. Open Lumberyard Editor, and then click **View, Open View Pane, Audio Controls Editor**.
7. In **Audio Controls Editor**, click **Add, Environment**.
8. Link the Wwise auxiliary bus to the ATL environment.

After you have setup reverb effects, you next create and define an area shape, set the **AudioAreaAmbience** or **AudioAreaEntity** entity properties, and then link the entity to the area shape. The process for doing this is very similar to that for adding ambient sounds to a level. For more information, see [Adding Ambient Sounds to Levels \(p. 139\)](#).

Setting Distance Values

The **FadeDistance** property for the **AudioAreaEntity** entity and the **RtpcDistance** property for the **AudioAreaAmbience** entity specify the maximum distance over which these values and the reverb level value are updated.

In order for the reverb level values to be updated correctly for players approaching and leaving an area shape, in most cases the **EnvironmentDistance** value is set lower or equal to the **FadeDistance** and **RtpcDistance** values in order to create realistic reverb (and ambient sound) effects. If it is necessary to have a greater **EnvironmentDistance** value, use two separate audio entities to control the reverb effect and then play the sound in the linked area shape.

After you set the **AudioAreaEntity** or **AudioAreaAmbience** entity properties, the audio is sent to the connected auxiliary bus in Wwise LTX when player approaches the area shape.

You can also assign sound volume values to a **GameParameter**, which you can control using an **Audio:RTPC** Flow Graph node.

Adding Collision Sounds to Levels

You can add physics-based collision sounds to your level using the `materialeffects.xml` spreadsheet file located in the `\dev\SamplesProject\libs\materialeffects\` folder. This file requires Microsoft Excel for editing, but you can preview it using any software that opens this file format.

The following figure shows a portion of a sample `materialeffects.xml` spreadsheet file for collisions with a rubber material. As you can see, most of the effects for rubber material use the `collisions_rubber_default` sound effect when rubber collides with various other surface types.

	mat_rubber
mat_metal	collisions:rubber_default
mat_metal_nofric	collisions:rubber_default
mat_metal_thick	collisions:rubber_default
mat_metal_RayProxy	collisions:metal_rayproxy_default
mat_metal_shell	collisions:metal_shell_default
mat_metal_barbwire	collisions:rubber_default
mat_ladder	collisions:rubber_default
mat_glass	collisions:rubber_default
mat_glass_breakable_safetyglass_large	collisions:rubber_default
mat_glass_breakable_safetyglass_small	collisions:rubber_default
mat_glass_breakable_thin_large	collisions:rubber_default
mat_glass_breakable_thin_small	collisions:rubber_default
mat_glass_unbreakable	collisions:rubber_default
mat_wood	collisions:rubber_default
mat_wood_RayProxy	collisions:rubber_default
mat_wood_breakable	collisions:rubber_default
mat_snow	collisions:snow_default
mat_ice	collisions:ice_default
mat_water	collisions:water_default
mat_concrete	collisions:rubber_default
mat_rock	collisions:rubber_default
mat_rock_landslide	collisions:rubber_default
mat_rock_dusty	collisions:rubber_default
mat_soil	collisions:rubber_default

To change the collision sound effect for a spreadsheet entry (such as for `rubber_default` for example) in the `materialeffects.xml` file, you edit the `collisions.xml` file.

To change a collision sound effect

1. Open the `collisions.xml` file in the `\dev\SamplesProject\libs\MaterialEffects\FXLibs\` folder for editing.
2. Specify which audio trigger plays when an effect is triggered by adding the appropriate code between the `START` and `END` markers for a material.

The following code example specifies that when the `rubber_default` effect is triggered, the `Play_cannonball_wall_impact` audio trigger is executed.

```
<Effect name="rubber_default">
<Audio trigger="Play_cannonball_wall_impact" />
</Effect>
```

The following shows a sample `collisions.xml` file.

```
<FXlib type="collision">
<!-- START mat_rubber START -->
```

```
<Effect name="rubber_impact">
  <Audio trigger="Play_cannonball_wall_impact" />
</Effect>
<!-- END mat_rubber END -->
.
.
.
</FXLIB>
```

Note

Sounds that end when an object impacts something, such as for bullet projectiles, use the `bulletimpacts.xml` file to define their effect instead.

Adding Sound to Trackview Sequences

This topic describes how to play sounds in Trackview sequences using the Audio Controls Editor.

Note

Sound tracks for **Director** nodes can play only 2D sounds because there is no associated entity for this node. Entity nodes can play both 2D and 3D sounds.

To add audio to a TrackView sequence

1. In Lumberyard Editor, click **View, Open View Pane, Track View** to open the Track View Editor.
2. In Track View Editor, click **Sequence** and select the sequence to which you want to add audio.
3. In the tree pane, select the entity that should play the sound. If that node does not already contain a **Sound** track, right-click it, then click **Add Track, Sound**.
4. In the center pane timeline, double-click on a time location on the sound track to add a key. The key can be dragged to another time or the time can be entered manually under **Key Properties** in the right pane.
5. Right-click the sound key in the center pane, click **Edit on Spot**, and adjust key properties as follows:
 - **StartTrigger** – The trigger name that triggers on the key.
 - **StopTrigger** – The trigger name that triggers after the time set in **Duration**.
 - **Duration** – The time after the key position when the **StopTrigger** is triggered. If the **Duration** value is 0, this will not get triggered.
 - **CustomColor** – Changes the color of the duration in the sound track.

Adding Sound to Animations

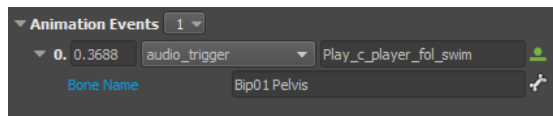
Sound effects contribute to a game by adding a sensory experience to characters, objects, weather, and more. You can add sound effects to animations by using Geppetto. This requires that an `.animevents` file has been created for the character and its animations before you can proceed.

You can also add sound by editing the XML file to reference an ATL (Audio Transition Layer) event.

To add sound effects by using Geppetto

1. Open Lumberyard Editor and click **View, Open View Pane, Geppetto**.
2. In **Geppetto**, under the **Assets** panel, double-click the character to which you want to add sound.
3. In the **Assets** pane, under **Animations**, choose an animation to which you want to add sound. The animation's properties load in the **Properties** panel.

4. In the **Properties** panel, for **Animation Events**, click the drop-down list and click **Add**.
5. For the new animation event, select **sound** from the drop-down list.
6. Enter a value for the time that the sound should play during the animation, or click on the animation event in the **Playback** timeline and drag it to where you want it on the animation.
7. Alternatively, you can double-click anywhere on the **Playback** timeline of the animation to add a new animation event, which is then displayed under **Animation Events** in the **Properties** panel for the animation.
8. Click on the field next to the drop-down list for **sound**, select the sound you want to assign to the event, and click **OK**.
9. You can achieve more precise timing of the sound by attaching the sound to a particular bone on the character. Under **Animation Events**, double-click the animation event, then for **Joint Name**, click the bone icon. In the **Choose Joint** window, choose a bone and click **OK**.



10. When done adding audio, click on the **Save** icon in the **Properties** panel to save the changes to the animation. The information is saved to the *.animevents file for the character.

To add sound effects by editing the XML file

1. Navigate to \SamplesProject\Objects\Characters*character* and use a text or XML editor to open the *.animevents file.
2. Add or edit the following event:

```
<event name="audio_trigger" time="0" endTime="0" parameter=""/>
```

3. Add or edit the `parameter` attribute with the ATL event.

Example: `parameter="Play_KatanaSwing"`

Adding Sound to Mannequin

You can control audio in Mannequin by adding procedural clips to fragments and setting their type to Audio in the procedural clip properties.

In turn, fragments are played on scopes. It is common to set up a Mannequin character in such a way that specific audio scopes are reserved exclusively for the placement of audio triggers on them. Using the Mannequin FragmentID Editor, you can enable a scope for a fragmentID to edit its default scope mask. When editing a fragmentID, you can select which scopes it should use by default.

The Mannequin system determines which fragments it triggers via tag states. This allows flexibility in supporting a variety of animations with sound.

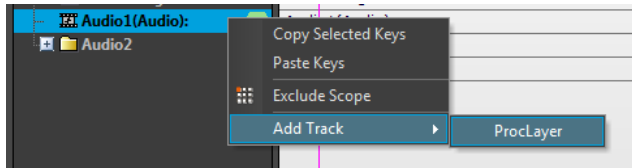
By adding tags to a fragment, you can also specify what needs to occur in the game or with the character for that specific fragment to be selected.

Adding a ProLayer Track

Once you have determined on which scope you want to place the audio triggers, a **ProLayer** track is first added to the scope. You can add any number of ProLayers to a scope, which can help better organize the fragment.

To add a trigger to a ProLayer track

- In **Mannequin Editor**, right-click on the applicable scope and click **Add Track, ProLayer**.

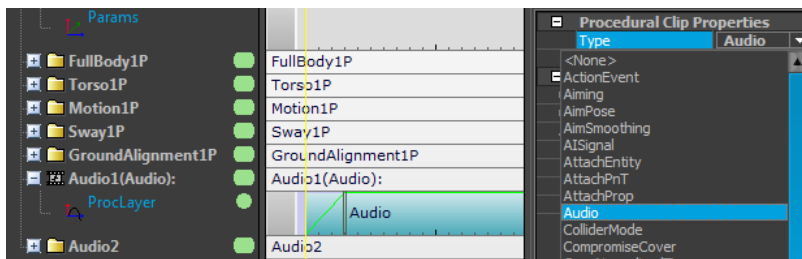


You can also add procedural clips to any ProLayer on any scope. These might, however, be saved to a different Animation Database (ADB) file, depending on your setup.

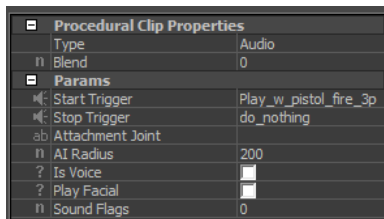
Adding a Trigger to a ProLayer Track

To add a trigger to a ProLayer track

- In **Mannequin Editor**, double-click in the new ProLayer timeline to add a procedural clip. To move the clip, you can drag its starting point.
- Under **Procedural Clip Properties**, click **Type** and select **Audio**.



- Under **Params**, select a **Start Trigger** and **Stop Trigger** as needed to define the sound behavior. To keep the sound playing, select **do_nothing** for **Stop Trigger**.



Audio Console Variables Commands

The following console variable commands can be used with the Lumberyard Audio system.

s_ATLPoolSize

Specifies in KB the size of the memory pool to be used by the audio translation layer (ATL).

Default values: PC = 8192, Xbox One = 8192, PS4 = 8192, Mac = 8192, Linux = 8192, iOS = 8192, Android = 4096

s_AudioEventPoolSize

Sets the number of preallocated audio events.

Default values: PC = 512, XboxOne = 512, PS4 = 512, Mac = 512, iOS = 128, Android = 128

s_AudioLoggingOptions

Toggles the logging of audio-related messages.

Default values: 0 (disabled), a = Errors, b = Warnings, c = Comments

s_AudioObjectsDebugFilter

Allows for filtered display of audio objects by a search string.

Default value: "" (all)

s_AudioObjectPoolSize

Sets the number of preallocated audio objects and corresponding audio proxies.

Default values: PC = 2048, XboxOne = 2048, PS4 = 2048, Mac = 2048, iOS = 256, Android = 256

s_AudioProxiesInitType

Can override on a global scale. If set, it determines whether AudioProxies initialize synchronously or asynchronously. This is a performance variable, as asynchronously initializing AudioProxies has a greatly reduced impact on the calling thread. When set to initialize asynchronously, audio playback is delayed.

Values: 0 = AudioProxy-specific initialization; 1 = Initialize synchronously; 2 = Initialize asynchronously.

Default value: 0 (all platforms)

s_AudioSystemImplementationName

Name of the AudioSystemImplementation library to be used without extension.

Default value: CryAudioImplWwise

s_AudioTriggersDebugFilter

Allows for filtered display of audio triggers by a search string.

Default value: "" (all)

s_DrawAudioDebug

Draws AudioTranslationLayer related debug data to the screen.

Values:

- 0: No audio debug info on the screen
- a: Draw spheres around active audio objects
- b: Show text labels for active audio objects
- c: Show trigger names for active audio objects
- d: Show current states for active audio objects
- e: Show RTPC values for active audio objects
- f: Show Environment amounts for active audio objects
- g: Draw occlusion rays
- h: Show occlusion ray labels
- i: Draw sphere around active audio listener
- v: List active Events
- w: List active Audio Objects
- x: Show FileCache Manager debug info

s_ExecuteTrigger

Executes an Audio Trigger. The first argument is the name of the audio trigger to be executed, the second argument is an optional audio object ID. If the second argument is provided, the audio trigger is executed on the audio object with the given ID; otherwise, the audio trigger is executed on the global audio object.

s_FileCacheManagerDebugFilter

Allows for filtered display of the different AFCM entries such as Globals, Level Specifics, and Volatiles.

Values: Default = 0 (all); a = Globals; b = Level Specifics; c = Volatiles

s_FileCacheManagerSize

Sets the size in KB that the AFCM allocates on the heap.

Default values: PC = 393216, Xbox One = 393216, PS4 = 393216, Mac = 393216, Linux = 393216, iOS = 2048, Android = 73728

s_FullObstructionMaxDistance

For sounds whose distance to the listener is greater than this value, the obstruction value is attenuated with distance.

Default value: 5 m

s_IgnoreWindowFocus

If set to 1, the sound system continues to play when the Editor or Game window loses focus.

Default value: 0 (off)

s_OcclusionMaxDistance

Obstruction/Occlusion is not calculated for the sounds whose distance to the listener is greater than this value. Set this value to 0 to disable obstruction/occlusion calculations.

Default value: 500 m

s_OcclusionMaxSyncDistance

Physics rays are processed synchronously for the sounds that are closer to the listener than this value, and asynchronously for the rest (possible performance optimization).

Default value: 10 m

s_PositionUpdateThreshold

An audio object has to move by at least this amount to issue a position update request to the audio system.

Default: 0.1 (10 cm)

s_SetRtpc

Sets an Audio Rtpc value. The first argument is the name of the audio RTPC, the second argument is the float value to be set, the third argument is an optional audio object ID. If the third argument is provided, the RTPC is set on the audio object with the given ID. Otherwise, the RTPC is set on the global audio object.

s_SetSwitchState

Sets an audio switch to a provided state. The first argument is the name of the audio switch, the second argument is the name of the switch state to be set, the third argument is an optional audio object ID. If the third argument is provided, the audio switch is set on the audio object with the given ID; otherwise, the audio switch is set on the global audio object.

s_ShowActiveAudioObjectsOnly

When drawing audio object names on the screen, this variable is used to choose between all registered audio objects or only those that reference active audio triggers.

Default value: 1 (active only)

s_StopTrigger

Stops an audio trigger. The first argument is the name of the audio trigger to be stopped, the second argument is an optional audio object ID. If the second argument is provided, the audio trigger is stopped on the audio object with the given ID; otherwise, the audio trigger is stopped on the global audio object.

s_VelocityTrackingThreshold

An audio object has to change its velocity by at least this amount to issue an `object_speed` RTPC update request to the audio system.

Default value: 0.1 (10 cm/s)

Characters and Animation

Most game projects require an animated character to move around in the environment. This may be a character that the player controls, or an AI-driven entity that interacts with the level.

The character animation system combines skeletal-based deformation of meshes with morph-based vertex deformation to allow for complex animation. Character movements appear much more realistic by playing and blending animation sequences, controlling facial expressions, and applying damage effects. Characters can play scripted movements, employ AI navigation, or use the Mannequin system to play complex, fully interactive animation sequences, either alone or in concert with other characters.

The recommended animation frame rate is 30 fps. If you are creating animations in Maya, there are additional supported frame rates of 15 fps, 60 fps, 120 fps, and 240 fps.

Topics

- [Working With Character Assets \(p. 151\)](#)
- [Maya Export Tools \(p. 177\)](#)
- [3ds Max Export Tools \(p. 189\)](#)
- [Working with the FBX Importer \(p. 195\)](#)
- [Using Geppetto \(p. 204\)](#)
- [Mannequin System \(p. 247\)](#)

Working With Character Assets

To work with character assets, first create your art assets, skeletal meshes, and animations using a third-party digital content creation (DCC) package such as Autodesk 3ds Max or Autodesk Maya. Then export your skeletal meshes and animations into Lumberyard.

Topics

- [Modeling Characters \(p. 152\)](#)
- [Rigging Characters \(p. 154\)](#)

- [Physicalizing Characters \(Ragdoll\) \(p. 157\)](#)
- [Using Inverse Kinematics \(IK\) \(p. 168\)](#)

Modeling Characters

The workflow for modeling characters is to model the characters in a digital content creation (DCC) tool, such as Autodesk Maya and Autodesk 3ds Max. You then export the characters to Lumberyard, where you apply material and shader settings.

As part of this process, you set up and create the following character modeling elements in a DCC tool:

- Asset structure
- 3D rendering mesh
- Pivot positions
- Scaling information
- Vertex colors
- Hierarchical structures
- Helper nodes
- Physics settings
- Breakability setup
- Skeletons and weighting

For best results, learn the best practices, asset file types, and export steps to ensure that your characters are imported into Lumberyard correctly and efficiently, as described in the topics following.

Topics

- [Character Modeling Best Practices \(p. 152\)](#)
- [Character Asset Files \(p. 153\)](#)
- [Using Character-Specific Shaders \(p. 153\)](#)
- [Debugging Character Skeleton Issues \(p. 153\)](#)

Character Modeling Best Practices

Consider the following best practices when modeling a character for later export to Lumberyard:

- Make sure all character geometry corresponds to the proportion and alignment of the skeleton.
- Select a pose that suits the widest range of motion that the character needs to perform.
- To improve the deformation of the character, make sure that all arm, shoulder, and leg joints are slightly angled for the selected pose.
- Add enough polygons to the joints to ensure a smooth deformation.
- If the character is used as an AI, make sure that the physics settings and inverse kinematics (IK) limit settings are correctly set.
- Make sure the character geometry is facing the positive Y-axis for Autodesk 3ds Max, or the Z-axis for Autodesk Maya.
- Make sure the base mesh and all morphs share the same vertex count and vertex IDs and have pivots in the same relative space.
- For character skinning best practices, see [Character Rigging Best Practices \(p. 154\)](#).

Character Asset Files

You can export the following character file types for use in Lumberyard.

Character File (*.chr)

You create the `.chr` file in a DCC tool. This file contains the base character.

Character Definition File (*.cdf)

You create the `.cdf` file in Geppetto. This file contains the base character, plus all attachments.

Character Skinned Render Mesh (*.skin)

You create the `.skin` file in a DCC tool. This file contains skinned character data. This data can be any asset that is animated with bone-weighted vertices, such as humans, aliens, ropes, lamps, heads, and parachutes. The `.skin` file includes the mesh, vertex weighting, vertex colors, and morph targets.

Using Character-Specific Shaders

Lumberyard provides the following shaders for use with characters:

- **Eye Shader (p. 1006)** – Renders realistic eyes that take sclera, cornea, iris, and eye moisture properties into account.
- **Hair Shader (p. 1011)** – Renders all character hair, giving the hair different color, stranding, and animation effects.
- **HumanSkin Shader (p. 1013)** – Renders character skin and its various physical properties, including color, oiliness, pores, stubble, and wrinkles.

Debugging Character Skeleton Issues

You can use the console variable `p_draw_helpers` to determine whether a character's physical skeleton is set up and working correctly.

You can display the following entity and helper types in the view port of Lumberyard Editor. To indicate the entity and helper types to display, enter options after the console variable `p_draw_helpers`. A list of possible options is shown following.

For example, if you enter `p_draw_helpers larRis_g` in the **Console** window, the window displays geometry for living, static, sleeping, active, independent entities, and areas in the view port.

Entity Types to Display

```
t - show terrain
s - show static entities
r - show sleeping rigid bodies
R - show active rigid bodies
l - show living entities
i - show independent entities
g - show triggers
a - show areas
y - show rays in RayWorldIntersection
e - show explosion occlusion maps
```

Helper Types to Display

```
g - show geometry
c - show contact points
b - show bounding boxes
l - show tetrahedra lattices for breakable objects
j - show structural joints (will force translucency on the main geometry)
t(#) - show bounding volume trees up to the level #
f(#) - only show geometries with this bit flag set (multiple f's stack)
```

Note

If the skeleton is in the default pose, you might need to choose **AI/Physics** in the bottom toolbar of the view port in Lumberyard Editor.

Rigging Characters

Before you can export a character to and animate it in Lumberyard, it must first be bound to a skeleton of bones and joints for bending and posing in your DCC tool. A *character rig* consists of this skeleton bound to the 3D character mesh.

For a character rig to work properly, the bones and joints must follow a logical hierarchy, starting with the root joint. Each subsequent joint is connected to the root joint either directly or indirectly through another joint. To help prevent unrealistic movements, we recommend that you set up joint constraints in your DCC tool.

Lumberyard Editor's scene axis is oriented with the Z-axis up and the Y-axis forward, which matches the orientation in Autodesk 3ds Max. However, Autodesk Maya's axis is oriented with the Y-axis up and the Z-axis forward by default. One option for using Autodesk Maya is to change the world coordinate setting from from **Y Up** axis to **Z Up** axis. To do this in Maya, choose **Windows, Preferences**, and then choose **Settings, World Coordinate System, Up axis**. Another option for Maya, if you want to keep the default axis orientation, is to use a `SceneRoot` node when exporting assets.

The general workflow for rigging a character model character rig using Autodesk 3ds Max or Maya is as follows:

- Set to zero all transform values for controllers.
- Orient all joints appropriately.
- Align a biped skeleton to the character model.
- Set up the `Locator_Locomotion` node as needed for animations.
- Skin your character. For 3ds Max, use **Skin modifier**. For Maya, use **Quaternion skinning**.
- Paint weight intensity values on the character's skin as needed.

Topics

- [Character Rigging Best Practices \(p. 154\)](#)
- [Character Skinning \(p. 155\)](#)
- [Painting Skin Vertex Weights \(p. 156\)](#)

Character Rigging Best Practices

Consider the following guidelines and best practices when you rig your characters in your DCC tool.

- Make sure the root node, `SceneRoot` node, and `Locator_Locomotion` node all share the same orientation, with the Z-axis up and Y-axis forward (in the direction the character is facing). For more information, see [Locomotion Locator Animation Best Practices](#) (p. 246).
- Make sure no position, rotation, or scale transformations are applied to control objects in rigs. If so, set them all to 0, 0, 0.
- If the model was sculpted to match an existing skeleton, make sure that it lines up and that all joints match.
- Characters must be in their bind pose, or the pose that is the reference pose for skin weights.
- Dual quaternion skinning must be used in all skin-binding procedures. Any other method used results in abnormalities when you import the character into Lumberyard.
- If you use Lumberyard's integrated IK system, you must set up joint orientations the same way. In addition, the naming of the joints must match those defined in the `.chrparams` file.
- Use the **Cryskin** and **Skin** modifiers in 3ds Max for skin weights. Do not use **Physique**.
- For Autodesk Maya, change the world coordinate setting from **Y Up** axis to **Z Up** axis. To access this in Maya, choose **Windows, Preferences**, and then choose **Settings, World Coordinate System, Up axis**.
- To check proportions, increase the transparency of the material to better see the bones inside the character.
- Collapse all list controllers if possible.
- Use the level of detail settings **LOD1**, **LOD2**, and **LOD3** for characters.
- Use rig elements inside the hierarchy sparingly, because they are exported as null bones.

Character Skinning

Set character skinning parameters in Maya and 3ds Max as follows.

Character Skinning in Maya

After all the bones and joints for your character rig have been added in Maya, set the correct skinning parameters as follows.

To set character skinning parameters in Maya

1. In Maya, choose **Skin, Smooth Bind**.
2. In **Smooth Bind Options**, for **Skinning Method**, choose **Dual quaternion**.
3. For **Max Influences**, we recommend that you choose **4**.

Note

If you need more skin weights, Lumberyard supports up to eight. To use eight skin weights, select the **8 Weights (skin only)** check box when exporting your skin.

Character Skinning in 3ds Max

After all the bones and joints for your character rig have been added in 3ds Max, set the correct skinning parameters as follows.

To set character skinning parameters in 3ds Max (version 2015 Extension 2, Service Pack 3 and later)

1. In 3ds Max, choose the **Modify** tab.
2. For **Modifier List**, for **OBJECT-SPACE-MODIFIERS**, choose **Skin**.

3. In the **Parameter** panel, for **Dual Quaternion**, choose **DQ Skinning Toggle**.

To set character skinning parameters in 3ds Max (versions 2014 and 2015)

1. In 3ds Max, choose the **Modify** tab.
2. For **Modifier List**, for **OBJECT-SPACE-MODIFIERS**, choose **CrySkin**. Choosing this option causes the proper deformations to display in 3ds Max and Lumberyard.
3. In the **Parameter** panel, for **Dual Quaternion**, choose **DQ Skinning Toggle**.

Painting Skin Vertex Weights

You can use DCC tools such as Autodesk Maya and 3ds Max to paint skin vertex weights on your character model. Although the controls differ from one application to another, the concepts are similar. You can copy, mirror, scale, blend, and assign numeric values to selected vertex weights.

Copying smooth skin weight information between characters can save a lot of time if your project involves setting up several similar characters. Just focus your painting efforts on one character, then copy those weights to the other characters.

If you plan on copying skin weights between characters, ensure that the skeletons on each character have the same structure and pose. If the orientation of the joints is not similar, the copying can lack precision, forcing you to touch up the results.

When mirroring weights from one side of the character to the other, make sure the character and rig are aligned and symmetrical along the X-, Y-, and Z-axes as applicable. Rotate and scale joints as needed to make the skeletons better match.

For information on how to perform specific tasks, see the documentation for your DCC tool.

Painting Weights in Maya

To paint weight intensity values on the current smooth skin, use the Paint Skin Weights tool in Maya. To set individual skin point weights to specific values, use the Component Editor.

Reflection is disabled by default for the Paint Skin Weights tool. To reflect skin weights, use the Mirror Skin Weights tool. To use this tool, choose **Skin, Edit Smooth Skin, Mirror Skin Weights** in Maya.

To paint character vertex weights in Maya

1. In Maya, choose **Skin, Paint Skin Weights Tool**.
2. Assign vertex weights as needed for your character rig.

Painting Weights in 3ds Max

Autodesk 3ds Max includes various tools for skin vertex painting, as follows:

- Skin envelopes
- Weight table
- VertexPaint

To paint character vertex weights in 3ds Max

1. In 3ds Max, choose **Modifiers, Mesh Editing, Vertex Paint**.

2. For **VertexPaint**, assign vertex weights as needed for your character rig.

Physicalizing Characters (Ragdoll)

Characters can have two skeletons: the main (alive) skeleton and the ragdoll (injured or dead) skeleton. Use Lumberyard to physicalize the ragdoll skeletons that you created in your DCC tools.

The ragdoll skeleton is what the main skeleton swaps to when it is inflicted with enough damage. It can be a more simplified version and often utilizes the use of capsules on limb joints for more accurate simulation. It features a similar physics mesh for joint limits and spring attributes as the main skeleton. For the most part, the ragdoll skeleton will be quite similar to the main skeleton and any differences are used to fine-tune the ragdoll simulation.

Topics

- [Ragdoll Best Practices \(p. 157\)](#)
- [Ragdoll Skeleton DCC Setup \(p. 157\)](#)
- [Creating Joint Mesh Proxies \(p. 158\)](#)
- [Using physParentFrames \(p. 161\)](#)
- [Applying Simulation Settings to Ragdoll Joints \(p. 162\)](#)
- [Lumberyard Proxy Tool \(Experimental\) \(p. 165\)](#)
- [Adding Mesh Proxy Materials \(p. 166\)](#)
- [Ragdoll Physics \(p. 166\)](#)

Ragdoll Best Practices

There are a few guidelines and best practices to follow when creating physicalized ragdoll characters:

- Use simplified geometry for your phys mesh proxies such boxes and capsules whenever possible to help with performance.
- In Maya, the physics skeleton does not use the `SceneRoot` node for determining the Up-axis of the scene.
- The root joint and first or highest hierarchical ragdoll joint, such as the pelvis/hip, needs to be oriented to match Z-up. In Maya, you can alternatively use a group node with the same orientation for Z-up as the parent of the first joint with a ragdoll phys mesh proxy, which may be the better choice for skeletons with existing animations that are not oriented as Z-up.
- For self-collision to work correctly, there is a current restriction with naming conventions for the following skeleton joints (these are case sensitive currently):
 - Pelvis
 - Spine
 - Head
 - UpperArm
 - Forearm
 - Thigh
 - Calf

Ragdoll Skeleton DCC Setup

This topic discusses how to setup your ragdoll skeleton in a DCC tool such as Maya or 3ds Max.

To setup a ragdoll skeleton

1. Create phys mesh proxies for the main joints that need to ragdoll. They should match the orientation of their respective joint. The first or highest hierarchical ragdoll joint and proxy (such as the pelvis and hip) should have the Z-up orientation.
2. Name the phys mesh proxies based on the joint they represent and with the appropriate suffix. For example:

For Maya, a joint named `def_l_thigh` would have a `physParentFrame` named `def_l_thigh_phys`.

For 3ds Max, a joint named `Bone C SpineA` would have a `physParentFrame` named `Bone C SpineA Phys`.

3. Create any necessary `PhysParentFrame` groups or nodes for joints that need to rotate more than the Y-axis limit range of -90 to 90 degrees and name them based on the joint they represent with the appropriate suffix. For example:

For Maya, a joint named `def_l_thigh` would have a `physParentFrame` named `def_l_thigh_physParentFrame`.

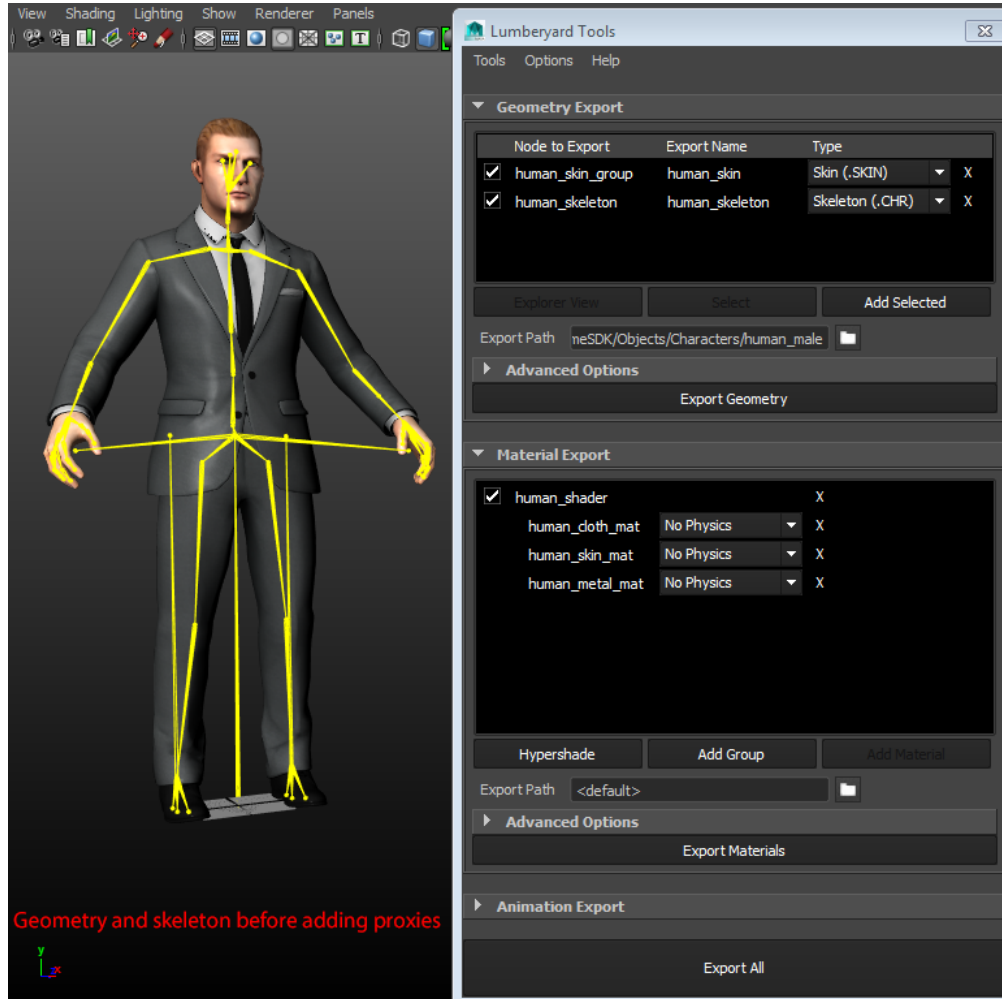
For 3ds Max, a joint named `Bone C SpineA` would have a `physParentFrame` named `Bone C SpineA Phys ParentFrame`.

4. Parent each phys mesh proxy and `PhysParentFrame`. If using a `PhysParentFrame`, parent the phys mesh proxy under the `PhysParentFrame`, and the `PhysParentFrame` under the joint it belongs to. If there is only the phys mesh proxy, parent the proxy under the joint it belongs to.
5. Create and assign phys mesh materials to the appropriate phys mesh as needed. For example, a material for the left arm assigned to phys mesh proxies that are part of the left arm.
6. Add rotation limit values to your skeleton joints that will be used in the ragdoll. You will still need to add some rotation limit information to joints that do not have a phys mesh proxy but are in the hierarchy of the ragdoll joints. For example, a clavicle needs some rotation limits even though it typically will not have a phys proxy mesh.
7. Export the skeleton's `.chr` file and the material group or multi-material that contains the phys mesh materials.

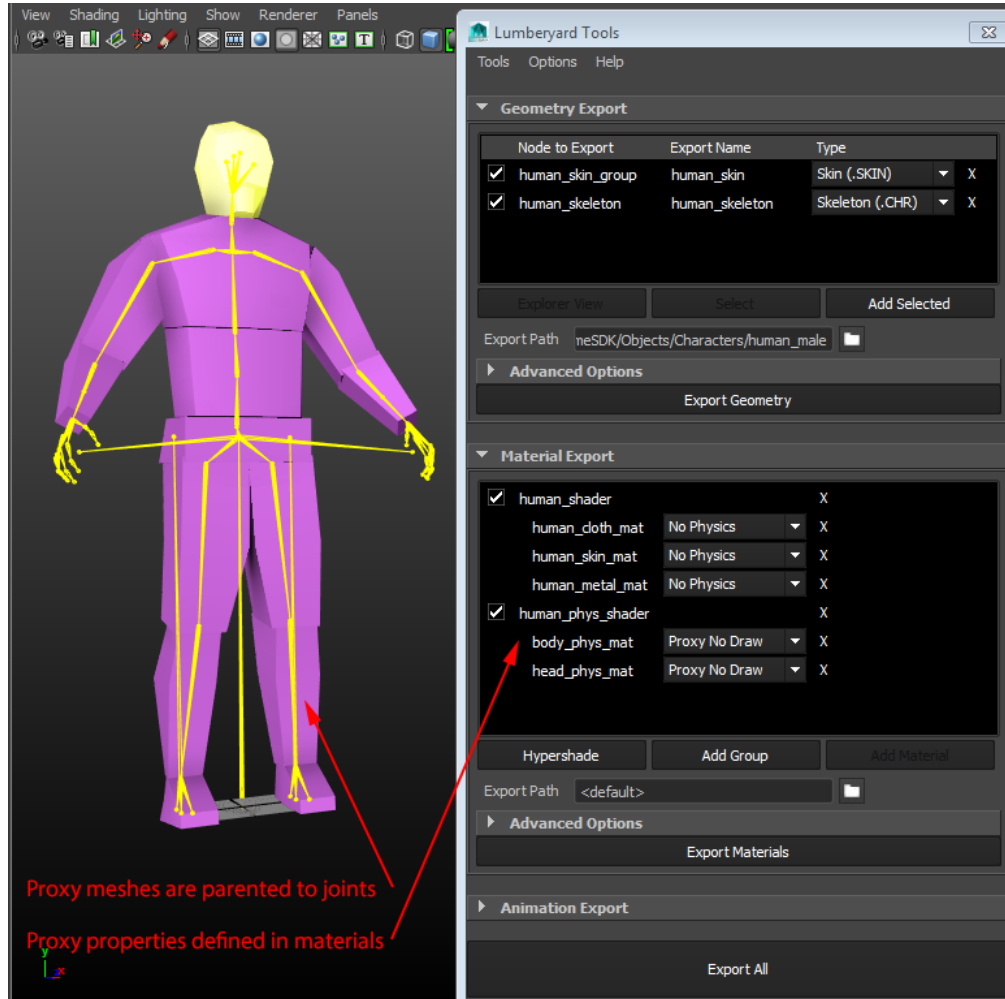
Creating Joint Mesh Proxies

To help define body masses for character physics and collisions, you need to use a joint mesh proxy. To create a joint proxy mesh, observe the following guidelines:

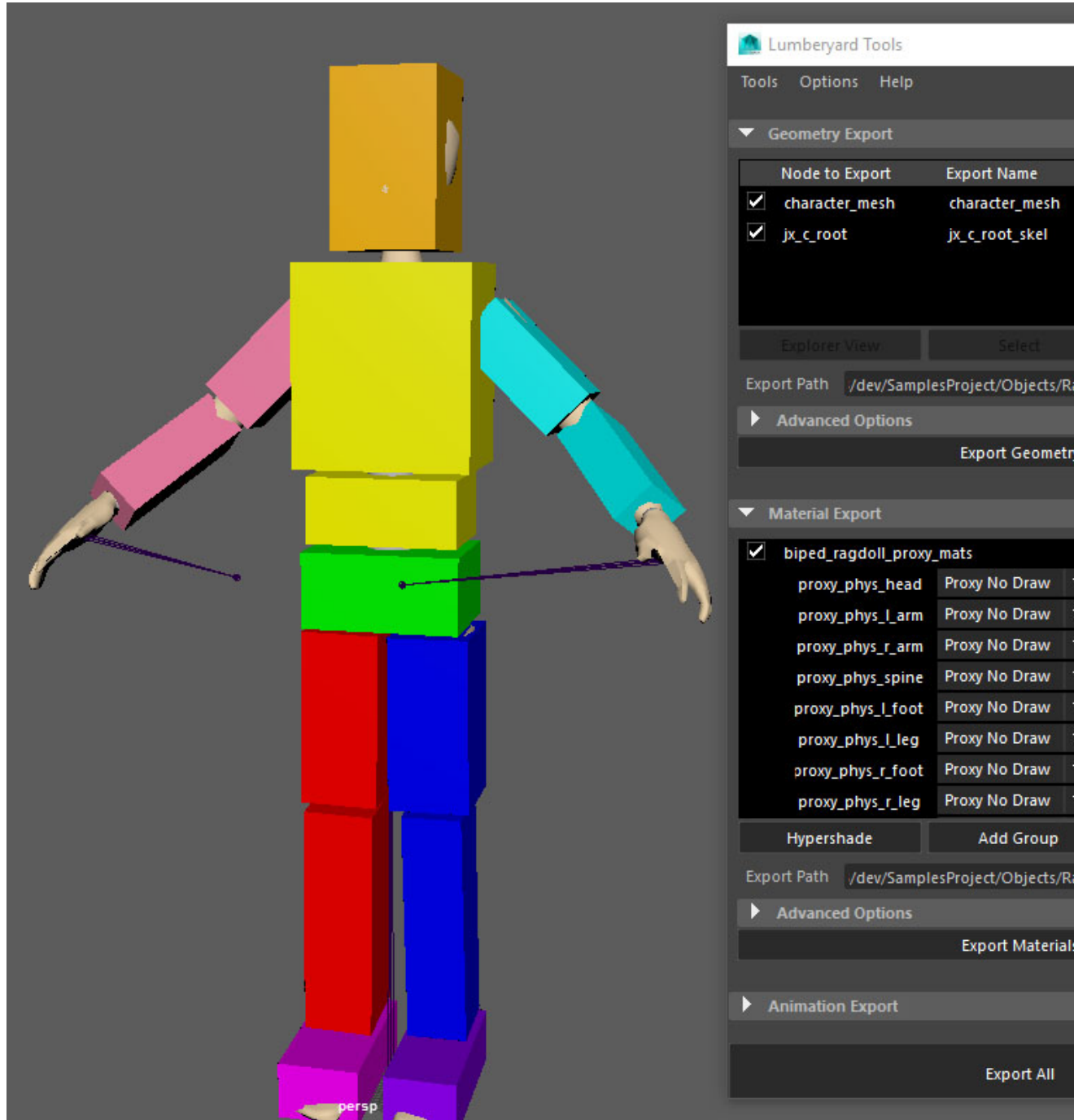
- Model meshes around the geometry that needs to be detectable. Meshes with lower polygon counts will perform better.
- Create meshes that use a generalized area instead of getting too granular with a phys mesh per joint to help with performance. For example, you could simplify a biped chest phys mesh proxy to cover the area of multiple spine joints instead of a proxy for each spine joint.
- Parent a proxy mesh to its corresponding joints. The proxies will then be exported with the `.chr` (skeleton) files.
- Proxy mesh naming needs to match the name of the joint it gets parented under with the addition of the following suffix:
 - For Maya, add the `_Phys` suffix (not case sensitive)
 - For 3ds Max, add the `Phys` suffix (not case sensitive)
- Meshes are designated as proxies by assigning a material to them and changing the material type to **Proxy No Draw**. It is recommended to keep your ragdoll phys mesh materials in a separate Material Group.



In the following figure, two different proxy materials are applied to the proxy mesh. Lumberyard uses the different materials to detect different parts of the body. In this case, the separate head material allows the engine to distinguish if the head is interacting with an object, as opposed to the rest of the body interacting with an object. For example, if a character gets hit in the head, you might want a special animation reaction to play, as opposed to the character getting hit in the body.



Ragdoll characters can have more proxy materials to define specific areas of the body. For example, you can have a proxy material for the head, spine, hip, left arm, right arm, left leg, right leg, left foot, and right foot.



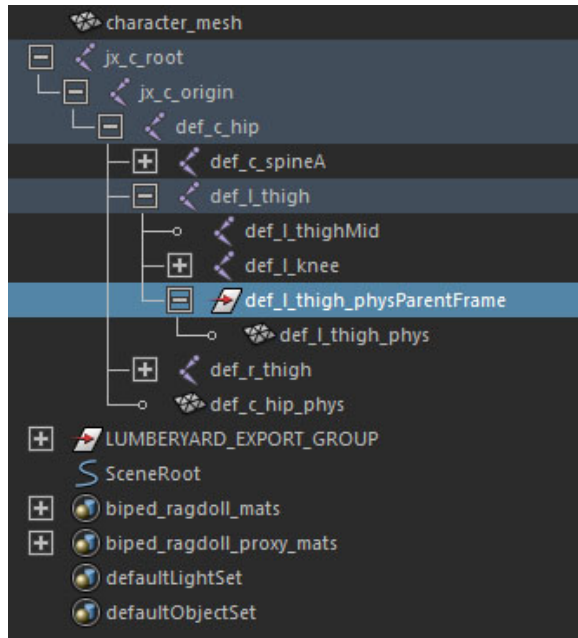
Using physParentFrames

If a character joint needs to rotate beyond the -90 to 90 degree Y-axis limit range, a `physParentFrame` node can be created. For example, if you need a joint that needs to rotate in the -120 to 120 degree Y-axis limit range, you would create a `physParentFrame` node or group for the joint. The `physParentFrame` node could have a -50 to 50 degree Y-axis limit range and the `phys` mesh could have a -70 to 70 degree Y-axis limit range to have the joint combine to a -120 to 120 degree Y-axis limit range.

You do not need to use another mesh for the `physParentFrame`. You can use a group node (for Maya) or dummy (for 3ds Max) as the `physParentFrame`.

In Maya, the naming convention for a `physParentFrame` is `joint name + _physParentFrame` suffix (not case sensitive).

In 3ds Max, the naming convention for a `physParentFrame` is `joint name + physParentFrame` suffix (not case sensitive).



The `physParentFrame` would be parented under the joint, and the `phys` mesh proxy would be parented under the `physParentFrame`.

Applying Simulation Settings to Ragdoll Joints

You will want to apply rotation limit values to your joints used for ragdoll depending on what ranges you want the joint to have. The default values are set to 0 degrees, but the range is -90 to 90 degrees, with the lowest range value being set in the rotation minimum and the highest range value being set in the rotation maximum. If you need more than the -90 to 90 degree Y-axis range, you will need to create a `physParentFrame` node. For more information, see [Using physParentFrames \(p. 161\)](#).

Any joint that has 0 degrees set for all the rotation limit ranges and is not in the Active and Limited states will be treated as non-physicalized. For this reason, it is a good idea to set some random limit ranges on joints that do not have a `phys` mesh proxy, but have child joints that do have a `phys` mesh proxy so the rest of the chain will still be physicalized. You will not need to enable the Active and Limited states for these joints either. For example, if you have a clavicle joint with no `phys` mesh proxy that is the parent of your shoulder joint that had a `phys` mesh proxy, you will want to add some values to the clavicle joint so the shoulder will still exhibit ragdoll behavior, but do not set any of the rotations in the Active/Limited states.

To apply simulation settings using Maya

1. Open Maya and select the root joint for your skeleton.
2. Select **Lumberyard Tools** from the Lumberyard Shelf.
3. Select **Tools, Add Attributes**.
4. In the **Attribute Editor**, scroll down to the **Extra Attributes** panel for the root joint.

You will see the ragdoll simulation settings that have been applied. This will be the case for every joint in the hierarchy.



5. Apply the desired simulation values for your ragdoll skeleton joints.

Select the **Rot Limited** check boxes for your coordinates to limit rotation to the specified values. Clear the check boxes for unlimited rotation.

6. Place the lowest range value in the **Rot Limit Min** field and place the highest range value in the **Rot Limit Max** field for the X, Y, and Z axes. For example, a joint in the -70 to 0 degree range for the Y-axis would have -70 in the **Rot Limit Min** (second-column) field and 0 in the **Rot Limit Max** (second-column) field.

Note

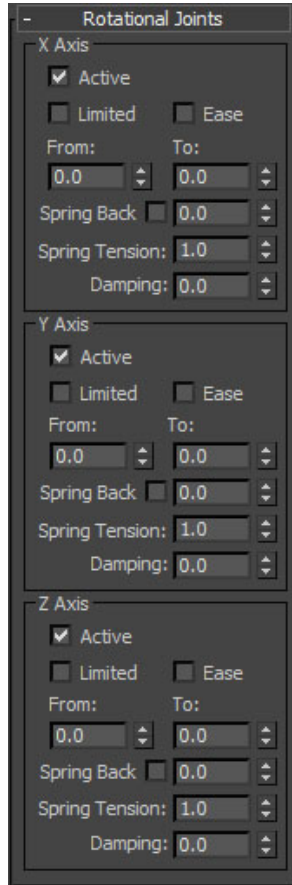
For Maya users, there is one exception to the **Rot Limited** checkboxes in the case of the pelvis/hip joint. You will want to apply some values for the **Rot Limit Min** and **Rot Limit Max** fields for the pelvis/hip joint, but keep the **Rot Limited** checkboxes unchecked.

7. After simulation settings have all been applied, export the character skeleton `.chr` file and the material group or multi-material that contains the phys mesh materials.
8. In Lumberyard, use the character skeleton `.chr` file as part of a `.cdf` as normal for character assembly.
9. Open Geppetto and preview your phys mesh proxies by enabling **Display Options**, **Physics**, **Physical Proxies**, and view **Ragdoll Joint Limits**.

To test your ragdoll, use either the Ragdoll component entity or the legacy DeadBody entity. For more information, see [Ragdoll Physics \(p. 166\)](#)

To apply simulation settings using 3ds Max

1. Open 3ds Max and select any skeleton joint.
2. Click the **Hierarchy** tab.
3. Click the **IK** button under the name of your joint.
4. Scroll down to the **Rotational Joints** panel.



5. Apply the desired simulation values for your ragdoll skeleton joints.

When using the X, Y, Z rotations for ragdoll, enable the **Active** and **Limited** checkboxes for the axes you are using and disable the **Active** checkbox for the axes you are not using.

Place the lowest range value in the **From** field and place the highest range value in the **To** field.

6. Set **Damping** to 1 for an active axis.
7. After simulation settings have all been applied, export the character skeleton `.chr` file and the material group or multi-material that contains the phys mesh materials.
8. In Lumberyard, use the character skeleton `.chr` file as part of a `.cdf` as normal for character assembly.
9. Open Geppetto and preview your phys mesh proxies by enabling **Display Options**, **Physics**, **Physical Proxies**, and view **Ragdoll Joint Limits**.

To test your ragdoll, use either the Ragdoll component entity or the legacy DeadBody entity. For more information, see [Ragdoll Physics \(p. 166\)](#)

Other Parameters

The **Spring Tension** parameter controls the stiffness of an angled spring at a joint. The default value of 1 means the acceleration of 1 radian/second² (1 radian = 57 degrees).

The **Damping** parameter controls how loose the joint will be in the ragdoll simulation. The default value of 1 is recommended because it corresponds to fully-damped oscillations for the joint.

Lumberyard Proxy Tool (Experimental)

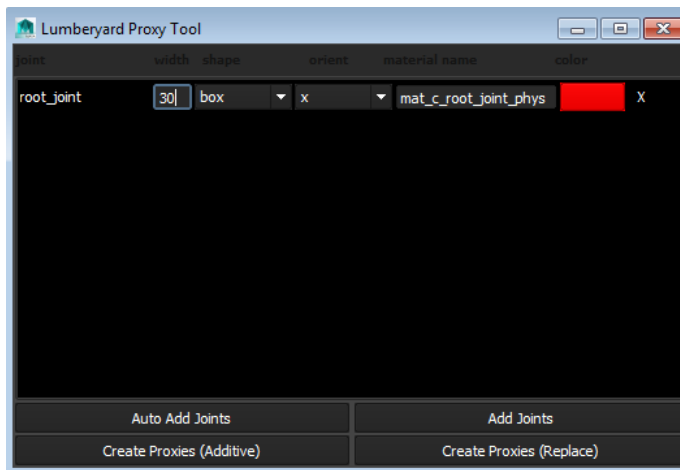
Creating individual meshes for each character body part can be time-consuming. The Lumberyard Proxy Tool automates the process of building simple joint proxy meshes and adding materials.

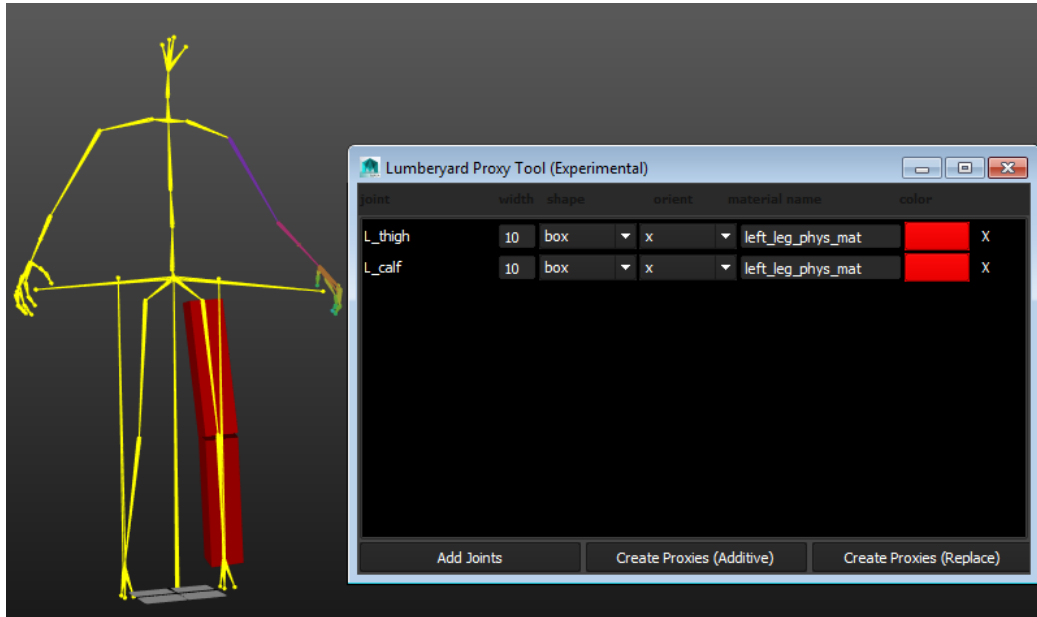
Note

This tool is in the experimental phase of development.

To create a joint proxy

1. In the Maya scene, select a joint you want to add a proxy to, and then choose **Add Joints**. Joint and proxy settings are displayed in the list window.
2. Choose the drop-down lists and adjust the following parameters as needed:
 - **Width** – Adjusts the width and depth of the proxy
 - **Shape** – Adjusts the shape of the proxy. Options are **box**, **capsule**, and **sphere**.
 - **Orient** – Adjust to match the orientation axis of the joint as it points to its child.
 - **Material name** – Name of the proxy material.
3. Choose one of the following:
 - **Create Proxies (Additive)** – Creates the joint proxies
 - **Create Proxies (Replace)** – Deletes all current proxies before creating the new joint proxies.





Adding Mesh Proxy Materials

For the materials to be used for the ragdoll phys mesh proxies, you assign a surface type to each sub-material. The shader type should already be set to **Nodraw** if the materials were exported from your DCC with the **Proxy No Draw** material physics type, but note that the **Surface Type** field in the Lumberyard Material Editor will be empty.

You can choose to set **Surface Type** to a few options depending on if you need to detect specific phys mesh proxies or if you do not need any additional special behavior. The **nodraw** type is a good default if you only want to use phys mesh proxies for your ragdoll skeleton. Otherwise, you can use the following settings:

- arm_left
- arm_right
- foot_left
- foot_right
- hand_left
- hand_right
- head
- leg_left
- leg_right
- torso

Ragdoll Physics

You can add physics to your ragdoll skeleton using either the **Ragdoll** component entity or by using the legacy **DeadBody** entity.

Using the Ragdoll Component Entity

Using the **Ragdoll** component entity is an easy way to test out your ragdoll asset. This requires that you have created a `.cdf` file using the character's skeleton `.chr` file that was exported with ragdoll simulation attributes on the joints and with phys mesh proxies.

To set up a Ragdoll component entity

1. In the Lumberyard Editor, right-click in your level and select **Create Component Entity**.
2. Select **View, Open View Pane, Entity Inspector (PREVIEW)** to view your component entity's settings, if the window is not open already.
3. In the **Entity Inspector (PREVIEW)** window, click the **Add Component** button.
4. Select **Physics, Ragdoll**.

You will see a new **Skinned Mesh** component and a **Ragdoll** component on your entity. The skinned mesh component was automatically added because it is required for the ragdoll component.

5. Under the Skinned Mesh component, expand **Rendering**.
6. For the **Skinned asset** parameter, click the ... button and locate your character's .cdf file for the ragdoll and assign it to the entity.

You can also use the `bidped.cdf` file located for the SamplesProject at `Objects/Tutorials/Biped/` to test if you do not have a ragdoll character.

7. Under the **Ragdoll**, select the **Enabled initially** checkbox.
8. Click the **AI/Physics** button at the bottom of the Lumberyard Editor window to view the ragdoll physics for your character. Click the button again to reset the character.
9. To make additional adjustments to your Ragdoll component settings, see [Rag Doll Component \(p. 387\)](#) for more information.

Note

To change the character joint rotations, you will need to change the simulation values on the skeleton in your DCC tool and then re-export the skeleton.

Using the (Legacy) DeadBody Entity

You may wish to use the legacy **DeadBody** entity for your ragdoll skeleton instead of using the Ragdoll component. The DeadBody entity is located on the Rollup Bar under **Entity, Physics**.

When using the DeadBody entity, ragdoll skeletons may collapse in unpredictable ways. To counter this, adjust the values of the following parameters in the **PhysParams** and **Properties** panels in Rollup Bar to the following:

- `ExtraStiff = 1` (enabled)
- `Mass = 80`
- `Stiffness = 100`

When using the DeadBody entity, the ragdoll skeleton will have the following characteristics:

- The ragdoll skeleton bones act as switches, activating physicalization of the corresponding bone in the main skeleton.
- The IK limits and dampening used in the physics mesh are read and used in ragdoll physics to limit and dampen the movement of any given joint.
- Each node in the ragdoll skeleton stores physical properties for its corresponding bone in the deforming hierarchy, as stored in the phys bone IK properties.
- The `ExtraStiff` parameter turns off constraints and attempts to maintain shape by pulling the bones toward an animation pose.

Fall-and-Play Movement

Fall-and-Play movement is activated when a character is a ragdoll (`RelinquishCharacterPhysics`) with a greater than zero stiffness. This activates angular springs in the physical ragdoll that attempt to bring the joints to the angles specified in the current animation frame. When a character is still a ragdoll it's also possible to turn the stiffness off with a `GoLimp` method.

The character tries to select an animation internally based on the current Fall-and-Play state. If there are no or very few physical contacts, it will be a falling animation. Otherwise, the first frame of a standup animation will correspond to the current body orientation.

Whenever there is an animation with a name that starts with `Standup_`, it's registered as a standup animation. Standup is initiated from outside the animation system through the appropriately named function. During the standup, the character physics is switched back into an alive mode, with the final physical pose blended into a corresponding standup animation. This is selected from a standup animation list that best matches this pose.

You can control which type to use by `_CSkeletonPose::SetFnPAnimGroup()` methods. On run-time, Lumberyard checks the most similar standup animation registered to the current lying pose and starts blending.

Using Inverse Kinematics (IK)

Inverse kinematics (IK) involves calculating the rotations of the joints in a character skeleton so that a specific part of the skeleton (the end effector) reaches a defined target point. Use IK when an animation calls for a terminating joint to be placed very precisely. All IK systems must be defined in the character's `.chrparams` file.

The following lists the order in which Lumberyard's animation system processes forward kinematics (FK) and IK tasks:

1. Aim IK and look IK
2. Animation-driven IK
3. Foot IK and ground alignment
4. Limb IK
5. Individual joint overrides

Topics

- [Aim IK \(Aim Poses\) \(p. 168\)](#)
- [Look IK \(Look Poses\) \(p. 172\)](#)
- [Animation-Driven IK \(p. 175\)](#)
- [Foot IK and Ground Alignment \(p. 176\)](#)
- [Limb IK \(p. 176\)](#)

Aim IK (Aim Poses)

Having a character aim a weapon at a target location is a somewhat complex but common movement required in a game. For example, aiming a weapon requires the weapon pointing at some specific location, the hands of the character firmly holding the weapon, and the character looking through the scope at all times. In many cases, other nuances are added to the character while aiming.

Lumberyard provides a parametric directional blending system that allows you to create a set of poses for characters aiming in different directions. At run-time, these poses are layered on top of the currently

playing animation so that the character aims towards a point in space requested by the game code, while retaining the style present in the original authored poses as much as possible. In this way, characters exhibit a realistic range of motion. Continuous 360 degree aiming around a pivot point is not supported however.

A number of poses of your character aiming in several directions is required so that they can be blended together to achieve poses in any intermediate direction. Create a set of 15 aiming poses for best results. When creating aimposes, it is common to use an underlying pose as a starting point, such as standing idle. The aimposes created from such a starting animation is applied on top of this animation. If the underlying animation currently playing for a character is different enough, it might be necessary to create aimposes for that specific case to achieve better quality.

Aim IK can be called using Flow Graph, Track View, the AI system, or from code.

Skeleton Setup

The system requires certain joints, listed following, to figure out where a character is aiming at. Sometimes you can use joints already present in the skeleton, but you might need to add some extra joints to make your setup work well. The aim IK bone should be a child of the head bone. Make sure your eye bones are also children of the head bone.

- `ParameterJoint` – A value that indicates the direction aimed in, with the Y-axis forward.
- `StartJoint` – A value that indicates the positional center of the aiming. Because only position information is used from this joint, its orientation is not important. For more stable results, consider using a joint that is not heavily animated, and that is not overly influenced by animation from other joints, such as a joint that is parented to the root joint.
- `ReferenceJoint` (optional) – A value that indicates the forward direction of the character, with the Y-axis forward. When no value is specified, the joint at index 1 (usually the pelvis) is used. This joint is used mainly for characters in cinematics, because they might have an offset on top of the root joint.
- `AnimToken` – A substring that needs to be matched to some part of the name of an animation to be processed as a aim pose with the current configuration for parameter, start, and reference joints.

Note

The joint names referenced for the attributes should match the names of the joints in your skeleton, but these names don't have any specific naming requirements.

.Chrparams File Setup

Aim IK parameters are stored in the .chrparams file, whose format is shown in the following example. You can have at most one `<AimIK_Definition>` tag block within an `<IK_Definition>` tag block. Within a `<AimIK_Definition>` tag block, you can have at most one of each of the following blocks: `PositionList`, `RotationList`.

```
<Params>
  <IK_Definition>
    <AimIK_Definition>
      <DirectionalBlends>
        <Joint AnimToken="AimPoses" ParameterJoint="Bip01 CustomAim"
StartJoint="Bip01 CustomAimStart" ReferenceJoint="Bip01 Pelvis"/>
      </DirectionalBlends>
      <RotationList>
        <Rotation Additive="1" Primary="1" JointName="Bip01 Pelvis"/>
        <Rotation Additive="1" Primary="1" JointName="Bip01 Spine"/>
        <Rotation Additive="1" Primary="1" JointName="Bip01 Spine1"/>
        <Rotation Additive="1" Primary="1" JointName="Bip01 Spine2" /
      >
    >
  >
</IK_Definition>
</Params>
```

```
CustomAim" />
    <Rotation Additive="0" Primary="1" JointName="Bip01
CustomAimStart" />
    <Rotation Additive="0" Primary="0" JointName="Bip01
LHand2Aim_IKTarget" />
    <Rotation Additive="0" Primary="0" JointName="Bip01
LHand2Aim_IKBlend" />
    ...
    <Rotation Additive="0" Primary="0" JointName="Bip01
RHand2Weapon_IKBlend" />
    </RotationList>
    <PositionList>
        <Position Additive="1" JointName="Bip01 Pelvis" />
        <Position Additive="0" JointName="Bip01 CustomAim" />
        ...
        <Position Additive="0" JointName="Bip01
RHand2Weapon_IKBlend" />
    </PositionList>
    </AimIK_Definition>
</IK_Definition>
    ...
<Params>
```

DirectionalBlends section

The `DirectionalBlends` section specifies a combination of parameter, start, and reference joints to use for aim poses. An animation is processed as a aim pose with this specific configuration when the `AnimToken` is found somewhere in its name. For example, any animation processed for a skeleton that contains the substring `AimPoses` anywhere in its path is considered a aim pose with `Bip01 Aim` as a parameter joint, `Bip01 Aim` as a start joint, and `Bip01 Pelvis` as a reference joint.

You can specify more than one `DirectionalBlends` section.

RotationList section

The list in the `RotationList` section is used by the run-time code to identify the joints that contribute their orientation to aim poses. Any joint not in this list will be ignored for the purposes of calculating and blending the aim pose.

Primary joints should be specified at the start of the rotation list. All primary joints must appear in the list before any of their children that are also marked as primary.

`AimPoses` can only have one rotation list, so all joints used by all aim poses should appear in this list, and the list should be valid for all of them.

- `JointName` – The name of the joint.
- `Additive` – The blend mode, where 1 is additive blending and 0 (zero) is override blending.
- `Primary` – A value that specifies if the joint is part of the hierarchical chain that goes from the root joint up to the parameter joint.

PositionList section

The list in the `PositionList` section is used by the run-time code to identify the joints that contribute their position to aim poses. Any joint not in this list is ignored for the purposes of calculating and blending the aim pose.

Aim poses can only have one position list, so all joints used by all aim poses should appear in this list.

Animation File Setup

The system requires a number of poses for a character aiming in several directions so that it can blend between the poses to aim in any intermediate direction. The system works with 9 or 15 poses. Although 9 poses might be enough for many cases, we recommend that you use 15 poses for better visual results. When you provide 9 poses, the system extrapolates from the provided ones to create 15 poses.

The poses are exported as an animation file, with one pose for each frame. Naming for this file is important. Some part of its name should match the AnimToken provided in the definition.

The order of the poses in the animation is also important.

When creating aim poses, commonly you use an underlying animation pose as a starting point (such as standing idle). The aim poses created from such a starting animation must be applied on top of similar animations. If the underlying animation currently playing for a character is different enough (such as crouching), you might need to create aim poses for that specific case to achieve better quality.

Try to make the poses as extreme as possible, even though they might aim unnatural. Limits can then be set using the game code. The middle pose (frame 4 of 9) needs to point forward. The other poses are centered around the middle pose. The angle between the middle pose and the remaining aim poses should be approximately 70 degrees.

Debugging Aim IK

The easiest way to verify that aim poses are working properly is to look at them in Geppetto with animation layers.

To view animation layers in Geppetto

1. Load your character in Geppetto.
2. Start an animation, and assign it to the base animation layer.
3. In the **Scene Parameters** panel, choose **Animation Layers**, and then choose **Add**. A new animation layer is added that has no animation assigned to it yet. This layer will become your active layer.
4. Select the aim pose animation to assign it to the new animation layer.
5. The aim pose animation is now layered on top of the base animation. Move the camera around in the Geppetto viewport, and observe the character aiming towards the camera.
6. Under the aim pose animation layer, adjust the direction of aiming, offset, and time-smoothing as needed.

Set the `ca_DrawAimIKVEGrid` console variable to 1 to display the grid for your aim poses. The green rectangle shows your individual aim pose frame extremes. As you move the camera around in the Geppetto viewport, you will see a red cube move around the grid to indicate which blend of the aim poses is being used. If you don't see a green rectangle or are running into other issues, recheck the setup for the aim poses in the `.chrparams` file and the orientation of the joints in the skeleton.

You can use the `ca_UseAimIK` console variable to enable or disable aim poses on a global level for debugging.

To see the current state of a character in the animation system during gameplay debugging, you can use the `es_debugAnim EntityName` console variable. Because this variable contains information on all animations that are being played, you can get information on which aim poses and aim poses play with which base animations. The combination of the aim pose with the base animation might explain why certain aim poses aim broken, for example if the combination doesn't match.

The base layer also displays information on the blend weights and final influences of the aim IK and look IK, and whether it is being requested by the game or not.

Look IK (Look Poses)

Lumberyard supports parametric blending for automated look IK that you can use to make characters look at specific targets, even in different locomotion cycles. A character with look IK tries to look at the target as long as possible and then turns its head away. The spine, head, eyelids, and eyeballs are all animated to make the character look in the target direction. This functionality is useful in cutscene animations to make sure characters makes eye contact with the player.

Look IK can be called using Flow Graph, Track View, the AI system, or from code.

Topics

- [Skeleton Setup \(p. 172\)](#)
- [.Chrparams File Setup \(p. 172\)](#)
- [Animation File Setup \(p. 174\)](#)
- [Debugging Look IK \(p. 174\)](#)

Skeleton Setup

The system requires certain joints, listed following, to figure out where a character is looking toward. Sometimes you can use joints already present in the skeleton, but you might need to add some extra joints to make your setup work well. The look IK bone should be a child of the head bone. Make sure your eye bones are also children of the head bone.

- `ParameterJoint` – A value that indicates the direction looked in, with the Y-axis forward.
- `StartJoint` – A value that indicates the positional center of the looking. Because only position information is used from this joint, its orientation is not important. For more stable results, consider using a joint that is not heavily animated, and that is not overly influenced by animation from other joints, such as a joint that is parented to the root joint.
- `ReferenceJoint` (optional) – A value that indicates the forward direction of the character, with the Y-axis forward. When no value is specified, the joint at index 1 (usually the pelvis) is used. This joint is used mainly for characters in cinematics, because they might have an offset on top of the root joint.
- `AnimToken` – A substring that needs to be matched to some part of the name of an animation to be processed as a look pose with the current configuration for parameter, start, and reference joints.

Note

The joint names referenced for the attributes should match the names of the joints in your skeleton, but these names don't have any specific naming requirements.

.Chrparams File Setup

Look IK parameters are stored in the `.chrparams` file, whose format is shown in the following example. You can have at most one `<LookIK_Definition>` tag block within an `<IK_Definition>` tag block. Within a `<LookIK_Definition>` tag block, you can have at most one of each of the following blocks: `LEyeAttachment`, `REyeAttachment`, `PositionList`, `RotationList`.

```
<Params>
  <IK_Definition>
<LookIK_Definition>
  <LEyeAttachment Name="eye_left" />
  <REyeAttachment Name="eye_right" />

  <DirectionalBlends>
```

```
        <Joint AnimToken="LookPoses" ParameterJoint="Bip01 Look"  
StartJoint="Bip01 Look" ReferenceJoint="Bip01 Pelvis"/>  
    </DirectionalBlends>  
  
    <RotationList>  
        <Rotation Additive="1" Primary="1" JointName="Bip01 Pelvis" />  
        <Rotation Additive="1" Primary="1" JointName="Bip01 Spine" />  
        <Rotation Additive="1" Primary="1" JointName="Bip01 Spine1" />  
        <Rotation Additive="1" Primary="1" JointName="Bip01 Spine2" />  
        <Rotation Additive="1" Primary="1" JointName="Bip01 Spine3" />  
        <Rotation Additive="0" Primary="1" JointName="Bip01 Neck" />  
        <Rotation Additive="0" Primary="1" JointName="Bip01 Head" />  
        <Rotation Additive="0" Primary="1" JointName="Bip01 Look" />  
    </RotationList>  
  
    <PositionList>  
        <Position Additive="1" JointName="Bip01 Pelvis" />  
    </PositionList>  
</LookIK_Definition>  
</IK_Definition>  
    ...  
</Params>
```

DirectionalBlends section

The `DirectionalBlends` section specifies a combination of parameter, start, and reference joints to use for look poses. An animation is processed as a look pose with this specific configuration when the `AnimToken` is found somewhere in its name. For example, any animation processed for a skeleton that contains the substring `LookPoses` anywhere in its path is considered a look pose with `Bip01 Look` as a parameter joint, `Bip01 Look` as a start joint, and `Bip01 Pelvis` as a reference joint.

You can specify more than one `DirectionalBlends` section.

RotationList section

The list in the `RotationList` section is used by the run-time code to identify the joints that contribute their orientation to look poses. Any joint not in this list will be ignored for the purposes of calculating and blending the look pose.

Primary joints should be specified at the start of the rotation list. All primary joints must appear in the list before any of their children that are also marked as primary.

`LookPoses` can only have one rotation list, so all joints used by all look poses should appear in this list, and the list should be valid for all of them.

- `JointName` – The name of the joint.
- `Additive` – The blend mode, where 1 is additive blending and 0 (zero) is override blending.
- `Primary` -- A value that specifies if the joint is part of the hierarchical chain that goes from the root joint up to the parameter joint.

PositionList section

The list in the `PositionList` section is used by the run-time code to identify the joints that contribute their position to look poses. Any joint not in this list is ignored for the purposes of calculating and blending the look pose.

Look poses can only have one position list, so all joints used by all look poses should appear in this list.

LEyeAttachment and REyeAttachment

These optional parameters specify the names of the left and right eyeball attachments. These parameters are used during skeleton post-processing to orient those attachments toward the target location. These parameters are relevant only if you use attachments for the eyes.

Animation File Setup

The system requires a number of poses for a character looking in several directions so that it can blend between the poses to look in any intermediate direction. The system works with 9 or 15 poses. Although 9 poses might be enough for many cases, we recommend that you use 15 poses for better visual results. When you provide 9 poses, the system extrapolates from the provided ones to create 15 poses.

The poses are exported as an animation file, with one pose for each frame. Naming for this file is important. Some part of its name should match the AnimToken provided in the definition.

The order of the poses in the animation is also important.

When creating look poses, commonly you use an underlying animation pose as a starting point (such as standing idle). The look poses created from such a starting animation must be applied on top of similar animations. If the underlying animation currently playing for a character is different enough (such as crouching), you might need to create look poses for that specific case to achieve better quality.

Try to make the poses as extreme as possible, even though they might look unnatural. Limits can then be set using the game code. The middle pose (frame 4 of 9) needs to point forward. The other poses are centered around the middle pose. The angle between the middle pose and the remaining look poses should be approximately 70 degrees.

Debugging Look IK

The easiest way to verify that look poses are working properly is to look at them in Geppetto with animation layers.

To view animation layers in Geppetto

1. Load your character in Geppetto.
2. Start an animation, and assign it to the base animation layer.
3. In the **Scene Parameters** panel, choose **Animation Layers**, and then choose **Add**. A new animation layer is added that has no animation assigned to it yet. This layer will become your active layer.
4. Select the look pose animation to assign it to the new animation layer.
5. The look pose animation is now layered on top of the base animation. Move the camera around in the Geppetto viewport, and observe the character looking towards the camera.
6. Under the look pose animation layer, adjust the direction of aiming, offset, and time-smoothing as needed.

Set the `ca_DrawAimIKVEGrid` console variable to 1 to display the grid for your look poses. The green rectangle shows your individual look pose frame extremes. As you move the camera around in the Geppetto viewport, you will see a red cube move around the grid to indicate which blend of the look poses is being used. If you don't see a green rectangle or are running into other issues, recheck the setup for the look poses in the `.chrparams` file and the orientation of the joints in the skeleton.

You can use the `ca_UseLookIK` console variable to enable or disable look poses on a global level for debugging.

To see the current state of a character in the animation system during gameplay debugging, you can use the `es_debugAnim EntityName` console variable. Because this variable contains information on all animations that are being played, you can get information on which aim poses and look poses play with which base animations. The combination of the look pose with the base animation might explain why certain look poses look broken, for example if the combination doesn't match.

The base layer also displays information on the blend weights and final influences of the look IK and aim IK, and whether it is being requested by the game or not.

Animation-Driven IK

Lumberyard supports animation-driven IK that can retarget limbs on the fly and that is controlled by the animation. You begin by controlling and animating the blend weight of this IK in your DCC tool.

An additional `_IKTarget` bone and `_IKBlend` weight bone inside a character's skeleton defines the IK target and the blend weight. These weights ensure that a limb reaches a specific destination regardless of animations in higher layers that modify the skeleton. For example, you might create a weapon reload animation that always brings the character's hand to the pocket at the belt, regardless of upper body animations rotating the torso and arms. You can also blend from one IK target to another, such as blending the left hand from a weapon to the magazine and back again.

Animation-driven IK can save memory and asset creation. For example, you can use the same aim pose for different guns by simply moving the IK target to the correct location on the new weapon.

You define the IK solver for a character inside the `.chrparams` file. Each entry in the file specifies which solver (2-bone, 3-bone, or CCD IK) to use with a chain of bones, the `_IKTarget` bone, and the `_IKBlend` weight bone.

You can animate both the `_IKTarget` bone and the `_IKBlend` weight bone. If the `_IKBlend` weight bone indicates that the IK should be blended in, Lumberyard uses the `_IKTarget` bone to apply the IK solver listed in the `.chrparams` file to the bone chain.

The end effector of the bone chain is aligned with the target bone and matching its rotation. In this way, you can also control hand orientation.

Blend weight is determined by the distance (in centimeters) of the `_IKBlend` weight bone from its parent along the X-axis. The distance is limited to values from 0 to 100 to avoid potential problems from blending multiple animations that might affect the same blend bones.

For best visual results, animate the character to get the end effector close and use the IK only to fix the deviation instead of doing all movement with the IK bones alone.

To make Lumberyard aware of the new IK bones and link them to a solver, open the `.chrparams` file and add a new line for each to the `<Animation_Driven_IK_Targets>` section, which lists every bone-controlled IK setup the character uses, as shown in the following example:

```
<Animation_Driven_IK_Targets>

  <ADIKTarget Handle="LftArm01" Target="Bip01 Chin_IKTarget" Weight="Bip01
  Chin_IKBlend"/>

</Animation_Driven_IK_Targets>
```

Each entry to the `<Animation_Driven_IK_Targets>` section specifies which bones to use for the target and the blend weight and includes a handle that points to an IK solver. These handles are listed in the `<LimbIK_Definition>` section of the `.chrparams` file, which links a solver and a bone chain.

Note

You cannot retarget animations between different skeletons.

Bones without rotation controllers are ignored for optimization purposes.

Foot IK and Ground Alignment

Lumberyard can automatically adjust a character's legs and feet to match the surface of the terrain the character is walking on. This adjustment includes foot alignment to the direction of the slope, in addition to adjusting the legs to different ground heights.

Leg and foot IK setup is defined in the character `.chrparams` file. Both legs must be added to the file as follows:

```
<LimbIK_Definition>
  <IK EndEffector="Right_Foot" Handle="RgtLeg01" Root="Right_Thigh"
  Solver="2BIK"/>
  <IK EndEffector="Left_Foot" Handle="LftLeg01" Root="Left_Thigh"
  Solver="2BIK"/>
</LimbIK_Definition>
```

The `Handle` name for the right and left legs must be `"RgtLeg01"` and `"LftLeg01"` respectively. You can use any naming for the calf, foot, and thigh as long as they are defined in the `.chrparams` file. For more information, see [Chrparams File Elements \(p. 231\)](#).

The bones listed following must be named as shown in the list and are required for ground alignment. The last four bones listed are all children of the foot bone.

- `Bip01 pelvis` – The character's hip joint.
- `Bip01 planeWeightLeft` – For 3ds Max, this bone shares the same X and Y position but is approximately 100 cm. above the foot on the Z-axis. For Maya, this bone shares the same X and Z position but is approximately 100 cm. above the foot on the Y-axis.
- `Bip01 planeTargetLeft` – For 3ds Max, this bone shares the same X and Y position and is aligned to 0 on the Z-axis. For Maya, this bone shares the same X and Z position and is aligned to 0 on the Y-axis.
- `Bip01 planeWeightRight` – For 3ds Max, this bone shares the same X and Y position but is approximately 100 cm. above the foot on the Z axis. For Maya, this bone shares the same X and Z position but is approximately 100 cm. above the foot on the Y axis.
- `Bip01 planeTargetRight` – For 3ds Max, this bone shares the same X and Y position and is aligned to 0 on the Z axis. For Maya, this bone shares the same X and Z position and is aligned to 0 on the Y axis.

The `PlaneTarget` and `PlaneWeight` bones are set up to give an absolute offset limit. The aligned pose drives the `PlaneTarget` node to align to the `PlaneWeight` node and no further.

Debugging Ground Alignment Poses

You can use the following console variables for debugging:

- `a_poseAlignerEnable 1` – Enables alignment.
- `a_poseAlignerDebugDraw 1` – Enables debug drawing of plane weight, target, and root offsets.
- `a_poseAlignerForceWeightOne 1` – Forces the weight to 1, which causes the limb to always experience automatic adjustments.

Limb IK

You can set up limb IK chains for characters. When a limb IK chain is active, Lumberyard calculates values for the joints that are part of the chain so that the end effector reaches the specified target position.

The behavior for each chain and the number of joints supported depends on the IK solver used: 2BIK for two-bone IK, 3BIK for three-bone IK, and CCDX for cyclic coordinate descent with x joints.

Systems that use limb IK chains include animation-driven IK, foot and leg ground alignment, and game code.

The following summarizes the attributes that you must define for each IK element:

- **EndEffector** – The joint that reaches the target location.
- **Handle** – The limb IK definition. No more than 8 characters are allowed, and the handle must be unique.
- **Root** – The starting joint for the IK chain.
- **Solver** – Code that calculates the joint values.

Note

The joint names referenced for the attributes should match the names of the joints in your skeleton, but these names don't have any specific naming requirements.

The limb IK parameters are stored in the `.chrparams` file with the following format:

```
<Params>
  <IK_Definition>
    <LimbIK_Definition>
      <IK EndEffector="Bip01 L Hand" Handle="LftArm01" Root="Bip01 L
UpperArm" Solver="2BIK"/>
      <IK EndEffector="Bip01 R Hand" Handle="RgtArm01" Root="Bip01 R
UpperArm" Solver="2BIK"/>
      <IK EndEffector="Bip01 L Foot" Handle="LftLeg01" Root="Bip01 L
Thigh" Solver="2BIK"/>
      <IK EndEffector="Bip01 R Foot" Handle="RgtLeg01" Root="Bip01 R
Thigh" Solver="2BIK"/>
    </LimbIK_Definition>
  </IK_Definition>
</Params>
```

Maya Export Tools

Lumberyard Tools is a plugin for Autodesk Maya 2014, 2015, and 2016 that exports geometry, animated geometry, skinned geometry, and skeletons (joint hierarchies) from Maya into Lumberyard.

To install the Lumberyard Tools plugin

1. Navigate to the Lumberyard root directory (`\\lumberyard\dev`) and run `Lumberyard Setup Assistant`.
2. On the **Install plugins** page, install **Autodesk Maya**.

Topics

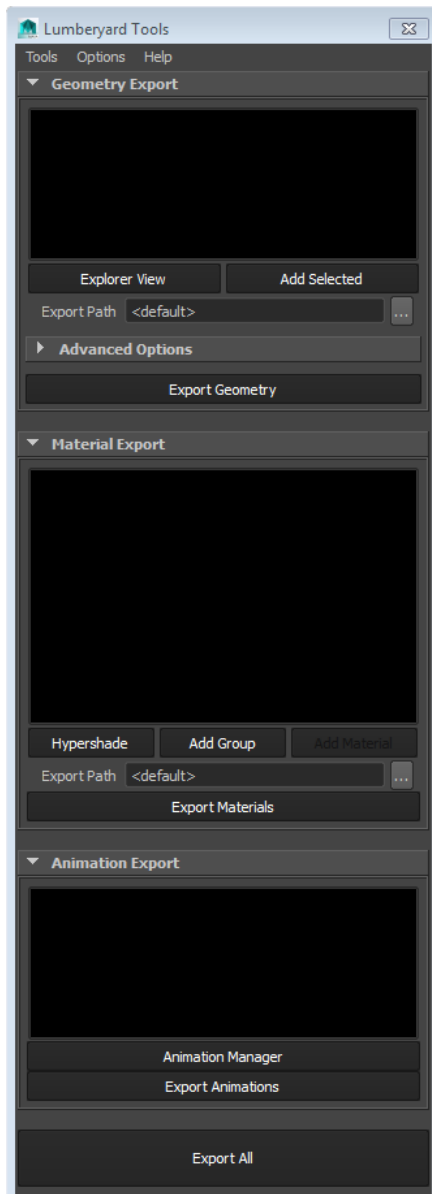
- [Accessing Maya Export Tools \(p. 178\)](#)
- [Setting Time Working Units for Maya \(p. 179\)](#)
- [Geometry Validation \(p. 179\)](#)
- [Exporting Static Meshes \(p. 179\)](#)
- [Exporting Characters \(p. 181\)](#)
- [Exporting Materials \(p. 183\)](#)

- [Exporting Animations \(p. 184\)](#)
- [Exporting Blendshapes \(p. 185\)](#)
- [Exporting Level of Details \(LODs\) \(p. 185\)](#)
- [Exporting an Alembic Cache \(p. 187\)](#)
- [Setting Export Options \(p. 188\)](#)

Accessing Maya Export Tools

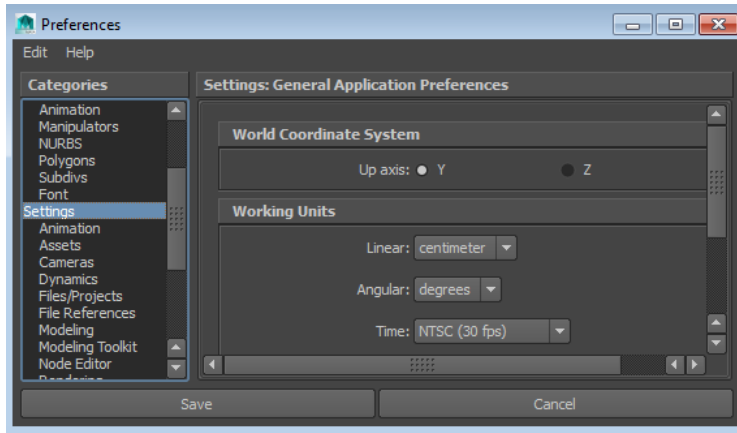
To install this plugin, run `SetupAssistant.bat`. On the **Install Plugins** page, install **Autodesk Maya**. After it is installed, the **Lumberyard** tab is available in the user interface of Maya. This tab presents a series of options, including the Lumberyard Tools beaver icon.

After the Lumberyard Tools dialog box opens, the following is shown:



Setting Time Working Units for Maya

We recommend that you use the **NTSC (30 fps)** setting for animations, but there are additional supported frame rates of 15 fps, 60 fps, 120 fps, and 240 fps.



To change time working units to NTSC

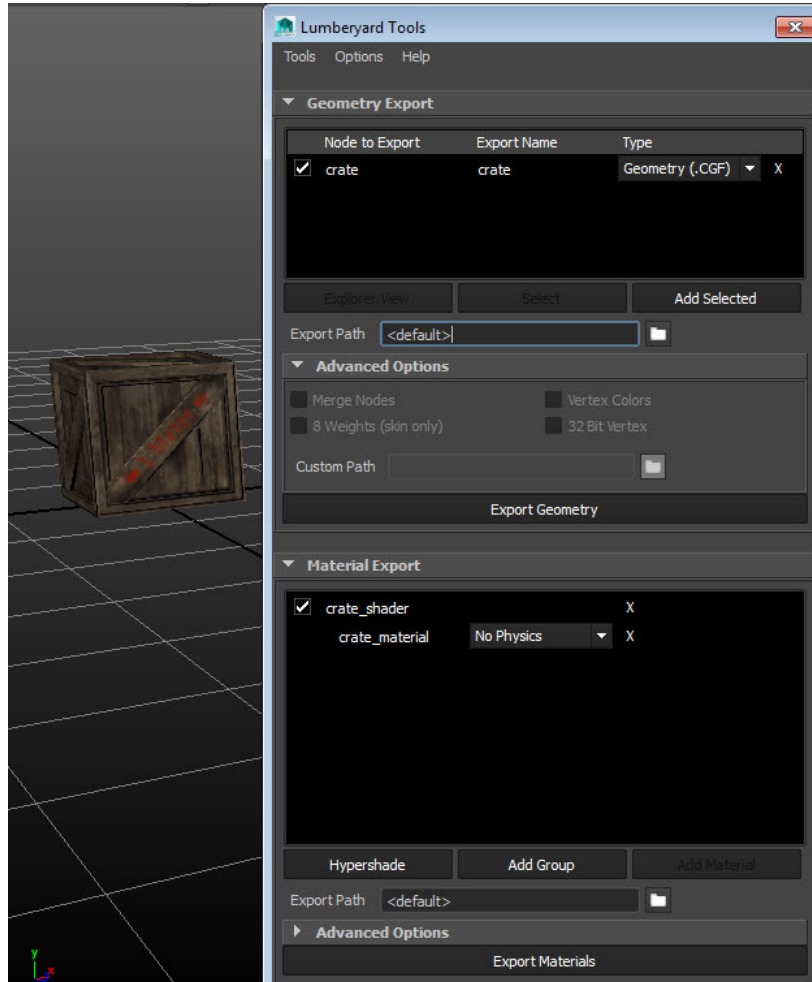
1. In Maya, choose **Window, Settings/Preferences, Preferences**.
2. In the **Preferences** dialog box, choose **Settings**.
3. Under **Working Units**, we recommend that you choose **NTSC (30fps)**, but **NTSC Field (60fps)** and **Film (240fps)** are also valid options.
4. Choose **Save**.

Geometry Validation

Before export, the plugin validates your character geometry. Be sure to resolve any errors that are displayed in the **Lumberyard Validation** window. For each error listed, choose **Focus** for more information about the error, as displayed in the **Transform Attributes** panel of the **Attribute Editor** for Maya. Errors are displayed on red backgrounds and warnings are on yellow backgrounds.

Exporting Static Meshes

To export static geometry, do the following steps. Make sure you save your scene before you export geometry.



To export static geometry

1. In Maya, choose the **Lumberyard** tab, and then choose the Lumberyard Tools beaver icon.
2. Select a geometry or group node in Maya, then choose **Add Selected** to add the node or group to the **Geometry Export** list window. You can only add one node or group (can be a group with children groups also) to the export list at a time. Select the check box to add the node or group for export. Choose the **X** to remove the node from the **Geometry Export** list. Choose the node name to edit the text as needed.

Note

Choose **Select** to see the node in Maya that corresponds to the export node in the **Geometry Export** list.

3. In the drop-down list, select **Geometry (.CGF)**.
4. For **Export Path**, choose the folder icon and select a directory path. By default, this path is the same as the directory of the current Maya file and all nodes are exported to this directory. Choose **Explorer View** to view the directory.
5. Expand **Advanced Options** and choose the following options as needed for the export node you selected:
 - **Merge Nodes** – Compiles geometry from multiple nodes into a single flattened mesh. Only supported for non-skinned geometry.
 - **8 Weights (skin only)** – Exports up to eight weights per skinned vertex. Generally used for faces or blend shapes.

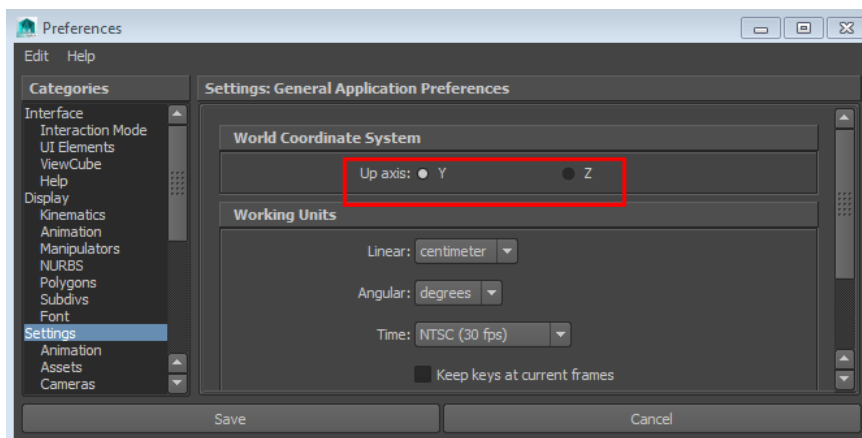
- **Vertex Colors** – Exports vertex colors.
- **32 Bit Vertex** – Enabling this will add 32-bits of precision to position each vertex accurately when the mesh is located far from its pivot. Note however that Playstation platforms only support 16-bit precision.

When working in centimeter units, 32-bit vertex precision is useful when the geometry is more than 10 meters from the pivot. When working in meter units, 32-bit vertex precision is useful when geometry is more than 100 meters from the pivot.

- For **Custom Path**, choose the folder icon and select a specific file path for your geometry. You can save each geometry to an individual location. This path overwrites the **Export Path** from the previous step.
6. Repeat as needed for each node you want to export. Make sure the check box is selected for each node you wish to export; otherwise that node will not be exported.
 7. With the desired node or group selected in the Maya scene, in the **Material Export** section, choose **Add Group**. This creates a material group and adds all of the materials that were applied to the mesh.
 8. When finished, choose **Export Geometry**.

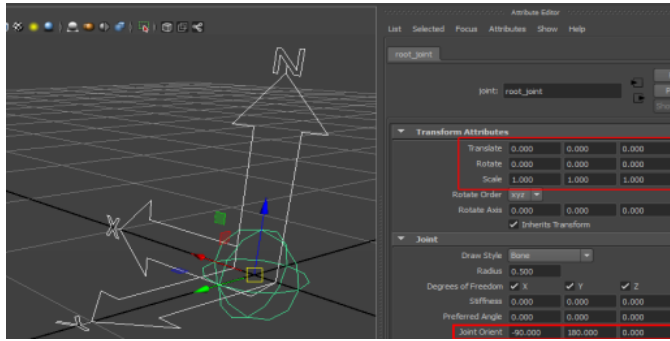
Exporting Characters

Before you can use the Lumberyard Tools plugin for exporting character geometry from Maya, you must check the **Up Axis** setting under **World Coordinate System** for your scene. By default, this setting is **Y** for Maya. To check this setting, click **Windows, Settings, Preferences**. In the **Preferences** window, under **Categories**, click **Settings**.



If the **Up Axis** is set to **Y**, you must ensure the following is true:

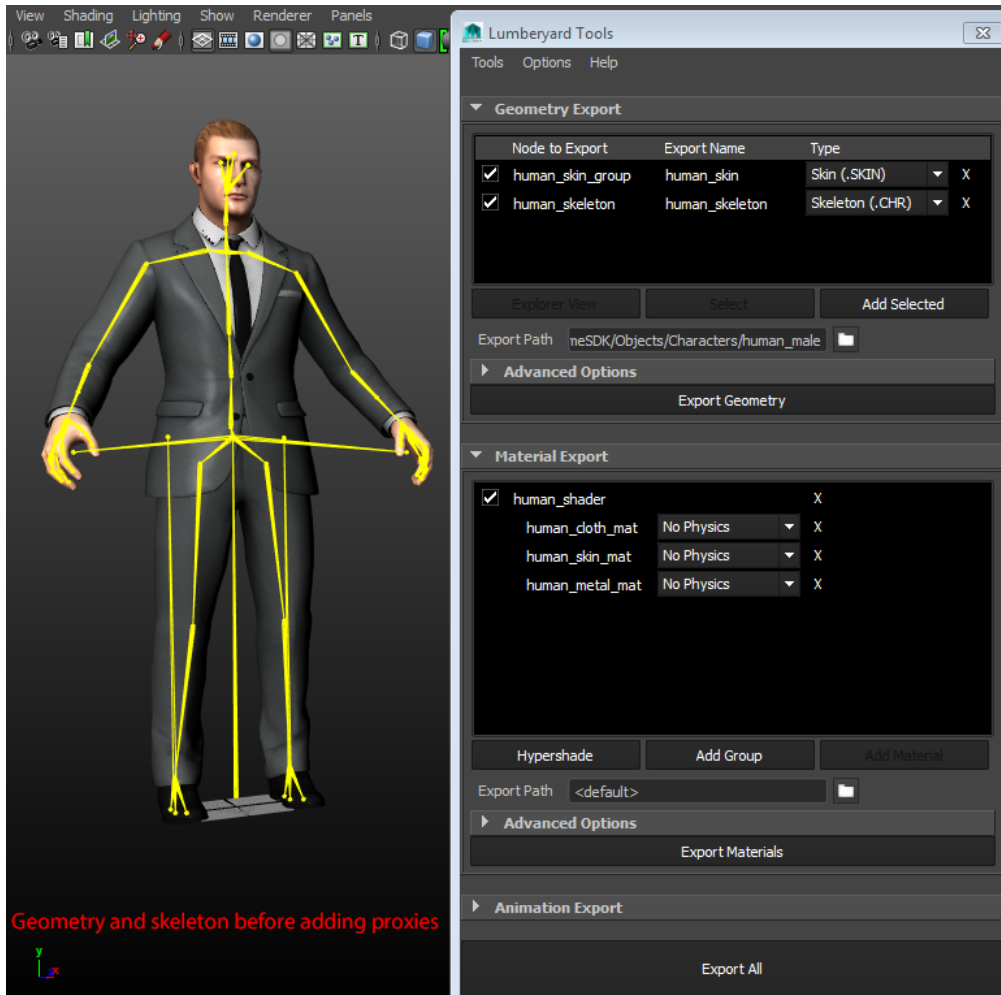
- The root joint of the character is positioned at the origin of the scene at $0, 0, 0$.
- The root joint of the character is oriented to **Z up** and **Y forward**.
- The **Joint Orient** attribute for the root joint is set to $-90, 180, 0$.
- A **SceneRoot** node exists for your scene. If this node does not exist, create it by choosing **Tools, Add Scene Root**.



If the **Up Axis** is set to **Z**, you must ensure the following is true:

- The root joint of the character is positioned at the origin of the scene at $0, 0, 0$.
- The root joint of the character is oriented to **Z up** and **Y forward**.
- The **Joint Orient** attribute for the root joint is set to $0, 0, 0$.
- A **SceneRoot** node does not exist for your scene.

The following procedure is very similar to the procedure on exporting static geometry, with many of the same options and advanced options. Refer to the previous procedure for explanation.



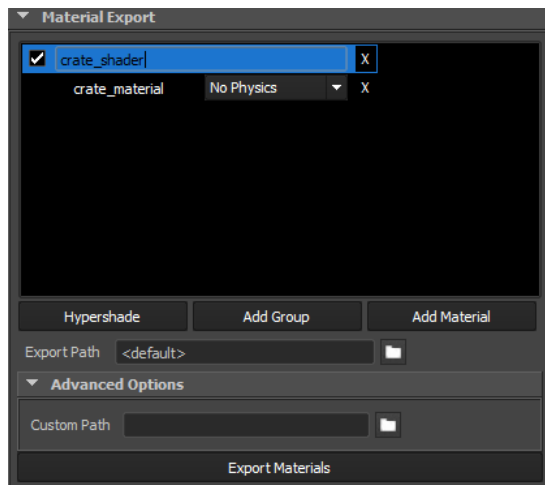
To export character geometry

1. In Maya, select the root joint node of the character.
2. In Lumberyard Tools, select the root joint node, then choose **Add Selected**. Be sure that it is set to **Skeleton (.CHR)** in the drop down list.
3. Select the geometry that is skinned to the joints and choose **Add Selected**. Be sure that it is set to **Skin (.SKIN)** in the drop down list.
4. Add the skinned geometry's materials to the **Material Export** list.
5. Choose **Export Geometry**.

Exporting Materials

There are a couple of ways to export material (.mtl) files. All exported materials must be contained in a material group as shown in the following. Be sure to save your scene before you export your materials.

Lumberyard also uses material information to drive physics properties.



To export character materials (Method 1)

1. In your Maya scene, choose the geometry that has the desired materials applied.
2. In Lumberyard Tools in the **Material Export** section, choose **Add Group**. This creates a new material group and automatically adds all applied materials to it.
3. In the **No Physics**, choose from the following options:
 - **No Physics** - Material contains no physics attributes (default setting).
 - **Default** - Render geometry is used as a physics proxy. This is expensive for complex objects, so use this only for simple objects like cubes or if you need to fully physicalize an object.
 - **ProxyNoDraw** - Mesh is used exclusively for collision detection and is not rendered.
 - **No Collide** - Proxy is used to detect player interaction, such as for vegetation touch bending.
 - **Obstruct** - Used for "Soft Cover" to block AI agent views, such as for dense foliage.
4. For **Export Path**, choose the folder icon and select a directory path. By default, this path is the same as the directory of the current Maya file, and all nodes will be exported to this directory. To export to a custom directory, choose **Advanced Options**, **Custom Path**, choose the folder icon, and select a specific file path for your materials. You can save each material to an individual location. This path overwrites the **Export Path** from the previous step.
5. Make sure the check box for each material you wish to export is selected, then choose **Export Materials**.

To export character materials (Method 2)

1. With nothing selected in the Maya scene, in Lumberyard Tools, choose **Add Group** to create an empty material group.
2. Select the newly created material group. Only material groups that are selected are exported. Choose the **X** to remove a material group as needed.
3. In Maya, select the materials in the **Hypershade** window you wish to add to this material group. Alternatively, you can select meshes that have the desired materials applied.

Note

Use the **Hypershade** button in Lumberyard Tools to display the material or group in the Maya **Hypershade** window for a selected material in the Lumberyard Tools **Material Export** window.

4. Choose **Add Material**.
5. In **No Physics**, choose from the following options:
 - **No Physics** - Material contains no physics attributes (default setting).
 - **Default** - Render geometry is used as a physics proxy. This is expensive for complex objects, so use this only for simple objects like cubes or if you need to fully physicalize an object.
 - **ProxyNoDraw** - Mesh is used exclusively for collision detection and is not rendered.
 - **No Collide** - Proxy is used to detect player interaction, such as for vegetation touch bending.
 - **Obstruct** - Used for "Soft Cover" to block AI agent views, such as for dense foliage.
6. For **Export Path**, choose the folder icon and select a directory path. By default, this path is the same as the directory of the current Maya file, and all nodes will be exported to this directory. If you want to export to a custom directory, choose **Advanced Options, Custom Path**, choose the folder icon, and select a specific file path for your materials. You can save each material to an individual location. This path overwrites the **Export Path** from the previous step.
7. Make sure the check box for each material you wish to export is selected, then choose **Export Materials**.

Tip

The order of materials listed can be changed by clicking on a material with the middle mouse button and dragging the material to the desired placement within the material group. This does not allow you to move a material to a different material group, however.

Exporting Animations

Lumberyard Tools uses the the Lumberyard Tools Animation Manager to specify various settings for each animation you want to export. New fields added to Lumberyard Tools Animation Manager also update the **Animation Export** window.

Be sure to save your Maya scene before you export animations.

Animation layers can be used to toggle animation key frames on a node. By default all animations are on a BaseAnimation layer. If new animation layers are added to a Maya file, they are reflected in the **Lumberyard Layers** drop-down list in **Lumberyard Animation Manager**. If an animation layer is selected, key frames on the animation layer will be exported. If an animation layer is not selected, the key frames on those layers will not be exported.

To export character animations

1. In **Lumberyard Tools**, choose **Animation Manager**.

2. In the **Animation Manager** dialog box, choose the **+** button and then specify the following properties:
 - a. For **Start** and **End**, enter values for the starting and ending frames for the animation, as defined in the Maya **Range Slider** settings. Choose the **< >** button to populate the start and end fields with the Maya time range slider start and end values.
 - b. For **Name**, type a name for the animation.
 - c. For **Root Node** select the root joint for animation and choose the **+** button.
 - d. Under **Animation Layers**, select **Selected1** from the drop down list and then select a layer. Select or deselect **BaseAnimation** as applicable if the animation is primary or secondary (additive).
 - e. For **Export Path**, choose the folder icon and select a directory path.
 - f. To delete an animation from the list, choose the **x** button next to it.
3. Repeat [Step 2 \(p. 185\)](#) as needed for each animation you want to export.
4. Make sure the check box is selected for each animation you want to export, then choose **Export Animations**.

Note

To export all static geometry, materials, and animated geometry that are listed and selected in each of the three lists at once, choose **Export All**.

Exporting Blendshapes

The following requirements must be observed when exporting a blend shape to Lumberyard.

- Select the skinned mesh with the blend shape nodes and add it to the **Geometry Export** list in the Lumberyard Tools. Be sure that it has been assigned the **.SKIN** extension type.
- Assign the appropriate materials to your skinned blend shape meshes. It should be identical to the materials used on your main skinned mesh.
- Add a material group for the **.SKIN** in **Materials Export** if you haven't already.
- Export the `.skin` file.

Exporting Level of Details (LODs)

Level of detail (LOD) techniques are used to increase performance and reduce draw calls by displaying progressively lower detailed objects the further they are from the camera. Generally, each LOD should have its vertices reduced 50% from the previous level and a reduction in the number of materials used. Lower LODs should also have multiple textures combined into one texture to further reduce draw calls.

Lumberyard supports up to six LODs to be used per group node in Maya. LOD number is from 0 (highest level of detail) to 5 (lowest level of detail).

LOD Naming

The following naming conventions for LODs must be used.

`_lod0_ through _lod5_` (prefix)

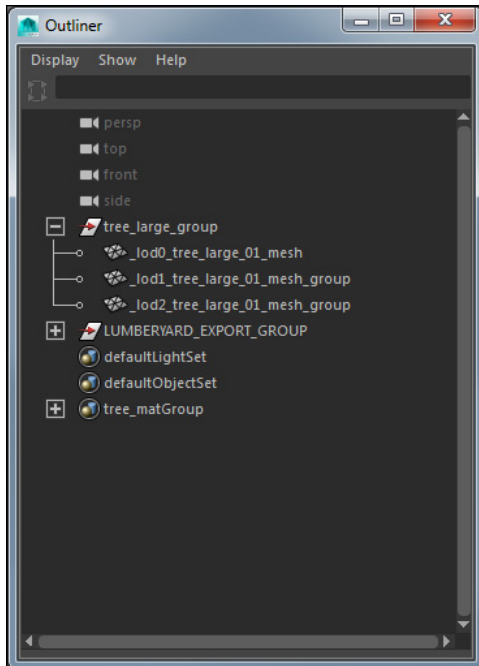
`_group` (suffix)

`_helper` (suffix)

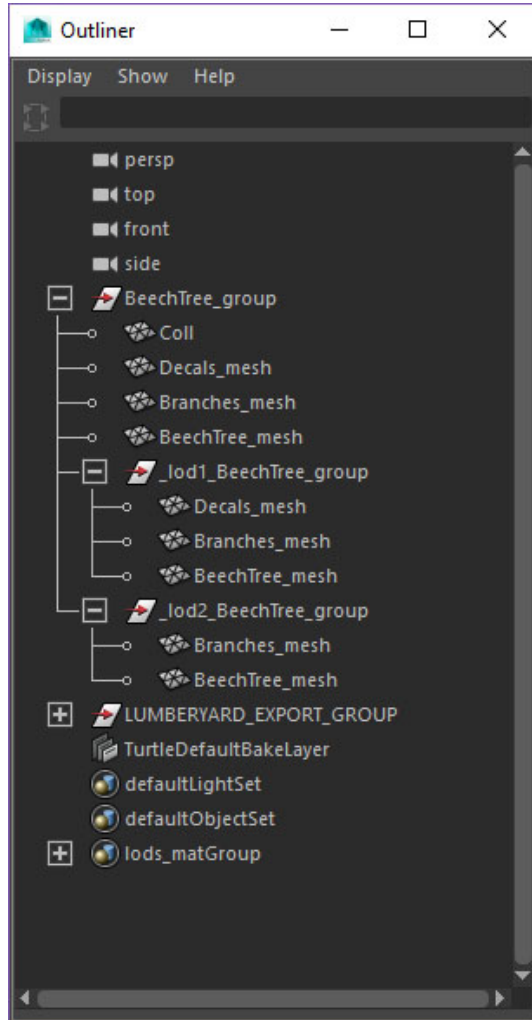
Any LOD that is not 0 must have the `_group` suffix or the LOD will not work in Lumberyard.

LOD Setup

Basic LOD Setup: All LOD meshes need to be under a group node in Maya. You will need to add the `_group` suffix at the end of the name for your group node. The following example shows assets that have no animated parts and small assets that do not need to be split up for culling.



Advanced LOD Setup: Each set of LOD meshes needs to be under a group node in Maya. You will need to add the `_lod#_` prefix at the beginning and `_group` suffix at the end of the name for your group nodes that contain these sets. The following figure shows an asset that has multiple meshes and a collision mesh that must be split into LODs that can be culled. The lowest LOD does not contain a `Decals_mesh` because by this LOD it will not be noticeable and the removal of it will save on performance.



Note

When exporting, under the **Advanced Options** panel, ensure that the **Merge Nodes** checkbox is not selected.

Debugging LODs

The following console variables can be used for debugging LODs:

- `e_DebugDraw = 1` - Name of the used cgf, polycount, and the used LOD.
- `e_LodCompMaxSize = 6` - (default value) Lower values force the LODs to load sooner.
- `e_LodRatio = 40` - (default value) LOD distance ratio for objects. Lower values force LODs to load sooner.
- `r_Stats = 6` - Displays the drawcall count for each object instance in the scene. The numbers above each object are broken into total DP, zpass, general, transparent, shadows, and misc.

Exporting an Alembic Cache

Alembic distills complex, animated scenes into a non-procedural, application-independent set of baked geometric results. Specifically, it handles baked meshes and their materials, but not rigs or other procedural networks of computations.

Lumberyard allows you to export Alembic (.abc) cache files from Maya. Lumberyard then compiles them into compressed game asset (.cax) files using the Resource Compiler and imports them into the game using the **GeomCache** entity. In-game, the .cax files are then streamed off disk.

Note

Deforming meshes can be exported along with their UVs and material assignments. However, multiple UV sets are not exported; only the default UV set is exported.

To export an Alembic cache from Maya

1. In Maya, rename each material using a unique integer ID. Material names are scanned from left to right and the first integer found is used. For example: mat01_sphere, mat02_sphere, mat03_cube.
2. In Lumberyard Tools, choose **Tools, Prepare Alembic Materials**.
3. In Lumberyard Tools, under **Material Export**, choose **Add Group** and then enter a name. The name of this material group (.mtl) file must match the name of the exported Alembic (.abc) cache file.
4. Set the export path to any folder within your game directory, and then choose **Export Materials**.
5. In Maya, select the geometry objects you want to export, and then in Lumberyard Tools, choose **Tools, Export Select to Alembic**.
6. In **Export Alembic for Geomcache**, navigate to the same directory used to export the materials to, enter the same name used for the material group, and then choose **Save**.

Lumberyard imports Alembic caches using the **GeomCache** entity found in the Rollup Bar.

To import an Alembic cache to Lumberyard

1. In Lumberyard Editor, choose **New** and then enter a name for the new level.
2. In the Rollup Bar, on the **Objects** tab, choose **Entity**.
3. Under **Browser**, expand **Render**. Select **GeomCache**, drag it into the level, then click to position the entity.
4. Under **Entity Properties**, choose the folder icon for **File**, select the Alembic (.abc) cache file previously exported, and then choose **Open**.
5. In **Compile Alembic**, change preset, compilation, and compression settings as needed, and then choose **OK**.
6. In **Running Resource Compiler**, review and resolve any errors, and then choose **Close**.

Note

Lumberyard automatically changes the **File** property to point to the compiled .cax file. If you modify the Alembic (.abc) cache file later, you'll need to recompile it into a .cax file. To do this from Lumberyard Editor, change the **File** property to point to the .abc file instead of the .cax file. You will then be prompted to repeat the steps in this section.

Setting Export Options

Lumberyard has a number of options to customize the export process. To apply them, select a geometry node from the list, choose **Tools**, and select from the following as needed.

Add Scene Root

Creates a scene node that re-oriens exported nodes relative to the displayed orientation.

Move Origin to Pivot

Sets a selected object's transform as an offset from the origin. If the Center Pivots check box is enabled, it will also center the pivot of the selected object.

Zero Joint Rotations

Removes any rotations on the selected joint and sets the value to zero.

Add Attributes

Exposes Lumberyard variables to joints and materials.

User Defined Properties

Opens a dialog box to add custom properties that is most commonly used for assigning a defined collision shape (sphere, box, or capsule) to override the existing collision mesh shape.

Polygon Check

Checks for degenerate faces.

Prepare Alembic Materials

Slightly modifies a scene to work around limitations in the Maya Alembic Exporter by changing the scene's shading engines and shading groups to enable the export of faceset information, which is used for the transport of the material assignments.

Export Selected to Alembic

Exports geometry caches that allow storing and playing arbitrarily animated geometry.

Joint Proxy Editor (Experimental)

Opens the Lumberyard Proxy Tool, which is used to create physics proxies for characters to be physicalized.

Validator

Runs the validation process.

3ds Max Export Tools

Lumberyard has a plugin for Autodesk 3ds Max 2014–2016 to simplify exporting static geometry, character geometry, and materials to Lumberyard. To install this plugin, go to the Lumberyard root directory (`\lumberyard\dev`) and start `Lumberyard Setup Assistant`, choose **Integrated tools**, and then choose **Autodesk Max**.

Topics

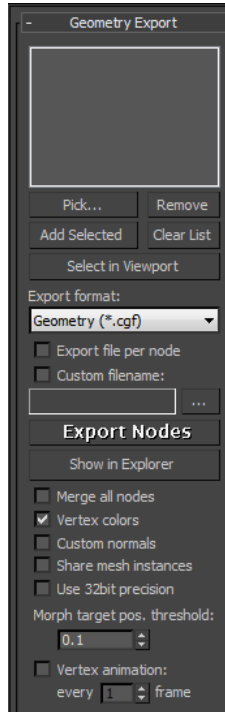
- [Exporting Static Meshes and Characters \(p. 189\)](#)
- [Exporting Materials \(p. 191\)](#)
- [Exporting Bones and Animations \(p. 191\)](#)
- [Exporting Levels of Detail \(LODs\) \(p. 193\)](#)
- [Configuring Log Options \(p. 194\)](#)

Exporting Static Meshes and Characters

Use the following procedure to export geometry and character geometry. You can specify which nodes in the scene to export, and other options regarding how they are exported. If nodes have any children, the child nodes are also exported.

Note

If you are exporting multiple Proxy No Draw meshes, they will need to be children of a single object, such as a dummy object. This will ensure that the exported meshes have collision functionality in Lumberyard.



To set geometry export options for 3ds Max

1. In 3ds Max, click the **Utilities** tab (hammer icon), and then choose **More**.
2. In **Utilities**, double-click **Lumberyard Exporter**.
3. In **Geometry Export**, choose the node in the viewport, and then choose **Add Selected**. Repeat as needed.
4. Choose the desired options as listed in the following table and then choose **Export Nodes**.

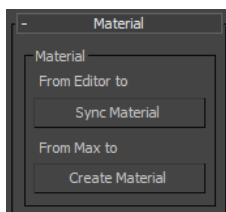
Geometry Export Options

Option	Description
Export Format	Specifies the file format for the exported file. Geometry export file formats include geometry (*.cgf), character (*.chr), character skeleton (*.skel), and character skin (*.skin).
Export file per node	Exports each node in the export list as a separate file. The filename is generated from the node name.
Custom filename	Overrides the default export filename if Export File per Node is not selected.
Merge All Nodes	Compiles non-skinned geometry from multiple nodes into a single node.
Vertex Colors	Exports vertex colors.
Use 32-bit precision	Enabling this will add 32-bits of precision to position each vertex accurately when the mesh is located far from its pivot. Note however that Playstation platforms only supports 16-bit precision.

Option	Description
Morph target pos. threshold	Vertices that don't move at least the specified distance (in meters) are ignored when the morph target is exported.
Vertex animation	Not supported at this time.

Exporting Materials

Use the following procedure to export materials.



To export materials

1. In 3ds Max, choose the **Utilities** tab (hammer icon), and then choose **More**.
2. In **Utilities**, double-click **Lumberyard Exporter**.
3. In **Materials**, do one of the following:
 - To update 3ds Max material settings to match those used in the Lumberyard material `.mtl` file for the object, choose **Sync Material**.
 - To create a material `.mtl` file with settings that match those used for the 3ds Max material, choose **Create Material**.

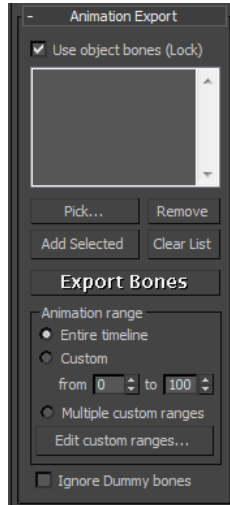
Exporting Bones and Animations

Animation Export contains the settings for the export of the skeleton and animations for skinned character models. When a node is added to the **Geometry Export** list, its skeleton root bone is also added to the **Animation Export** list. As a result, you typically don't need to configure the **Animation Export** settings. However, sometimes it is helpful to be able to directly edit this list (for example, when a user wants to export animations for only the upper body).

Note

You must export your animations using the **30 FPS** frame rate setting, otherwise the Asset Processor will fail. This is set in the **Time Configuration** dialog under **Frame Rate**.

Use the following procedure to export character skeleton bones. If bones have any children, the child bones are also exported.



To set bone export options for 3ds Max

1. In 3ds Max, choose **Utilities** tab (hammer icon), and then choose **More**.
2. In **Utilities**, double-click **Lumberyard Exporter**.
3. In **Geometry Export**, choose the node in the viewport, and then choose **Add Selected**. Repeat as needed.
4. In **Animation Export**, choose the desired options as listed in the following table, and then choose **Export Bones**.

Bone Export Options

Option	Description
use object bones (Lock)	Uses the bone of the geometry target listed in Geometry Export .
Ignore Dummy bones	Prevents any dummy bones that are in the bone hierarchy from being exported.

5. (Optional) In **Animation range**, you can also specify the animation range for a character's skeleton using the various parameter options listed in the following table.

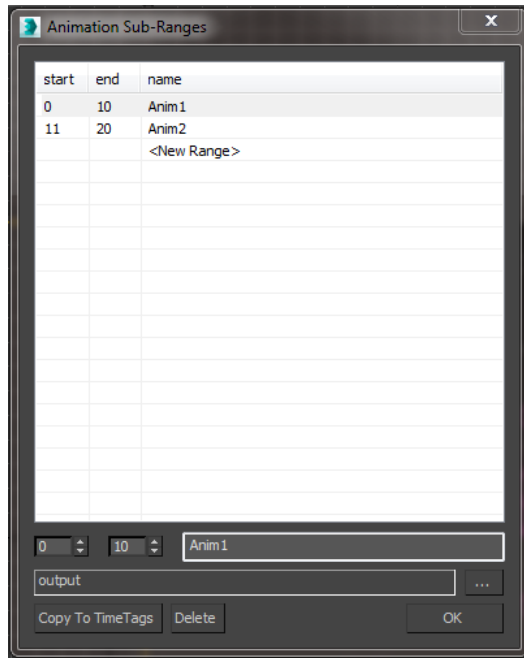
Animation Range parameters

Parameter	Description
Entire timeline	Uses the full timeline length.
Custom	Uses the customized length by specifying the start and end frames.
Multiple custom ranges	Uses specified multiple animation ranges (for details, see the following procedure).

To edit multiple custom animation ranges

1. In **Animation Export**, choose **Edit custom ranges**.
2. In **Animation Sub-Ranges**, double-click **<New Range>** and then type a name.
3. Use the arrows to specify the start and end frames.

4. Choose the ... button and then choose an export file path for the animation range.



Exporting Levels of Detail (LODs)

Level of details (LODs) is a technique that increases performance and reduces draw calls by displaying progressively lower detailed objects the further they are from the camera. Generally, each LOD should have its vertices reduced 50% from the previous level and a reduction in the number of materials used. Lower LODs should also have multiple textures combined into one texture to further reduce draw calls.

Lumberyard supports up to six LODs to be used per group node in 3ds Max. LOD number is from 0 (highest level of detail) to 5 (lowest level of detail).

LOD Naming

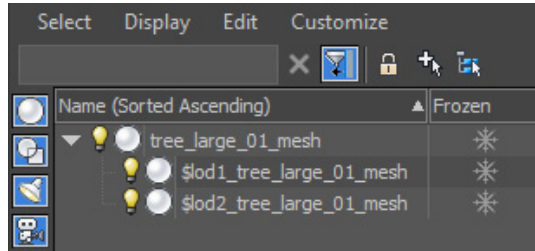
LOD naming conventions are very important with respect to prefixes and suffixes. You must use the following naming conventions:

The highest LOD mesh (LOD 0) does not need a prefix.

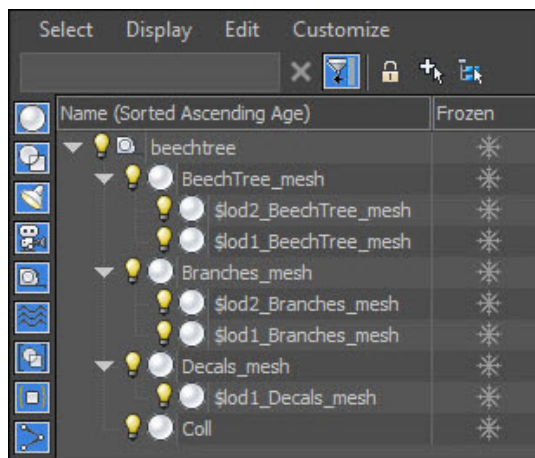
`$lod1_` through `$lod5_` (prefix)

LOD Setup

Basic LOD Setup: All LOD meshes with the appropriate prefix need to be parented under the main render mesh (LOD0). Refer to the example below for assets that have no animated parts or for small assets that do not need to be split up for culling.



Advanced LOD Setup: When you have LOD subobject meshes, the same rule applies as the basic setup where the all LOD meshes with the appropriate prefix need to be parented under their respective main render mesh (LOD0). The LOD0 mesh for the subobjects should be parented under the main object LOD0 mesh. Refer to the example below for assets that have animated parts or that are large and need to be split into multiple objects that can be culled.



Debugging LODs

The following console variables can be used for debugging LODs:

- `e_DebugDraw = 1` – Name of the used cgf, polycount, and the used LOD.
- `e_LodCompMaxSize = 6` – (default value) Lower values force the LODs to load sooner.
- `e_LodRatio = 40` – (default value) LOD distance ratio for objects. Lower values force LODs to load sooner.
- `r_Stats = 6` – Displays the drawcall count for each object instance in the scene. The numbers above each object are broken down into total DP, zpass, general, transparent, shadows, misc.

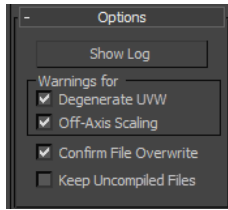
Configuring Log Options

There are several options for configuring what is logged during export.

To set exporter log options for 3ds Max

1. In 3ds Max, choose the **Utilities** tab (hammer icon), and then choose **More**.
2. In **Utilities**, double-click **Lumberyard Exporter**.
3. In **Geometry Export**, choose the node in the viewport, and then choose **Add Selected**. Repeat as needed.

4. In **Options**, choose the desired options as listed in the following table, and then choose **Show Log**.



Other Options

Option	Description
Degenerate UVW	Checks for degenerate texture coordinates and issues a warning if they exist, otherwise, silently exports them. Degenerate coordinates arise when two vertices on a triangle have the same (or very nearly the same) UVs.
Off-axis scaling	Checks whether the node is scaled along a non-primary axis. The node can still be exported, but the scale won't match the object in 3ds Max.

Working with the FBX Importer

FBX Importer is in preview release and is subject to change.

You can use the **FBX Importer** to import static FBX meshes and materials into Lumberyard.

Note

FBX files are required to be located in your game project directory. The **FBX Importer** will not import an `.fbx` file from a location outside of the game project directory.

When you import an `.fbx` file, Lumberyard creates a `.assetinfo` file with the `.fbx` file. The `.assetinfo` file stores the configuration and rules settings that are applied when processing. When you load an `.fbx` file that already has an `.assetinfo` file, the data automatically appears in the **FBX Importer**.

Note

The `.assetinfo` file is an updated version of the `.scenestettings` file. If you are using earlier versions of the FBX Importer, you may need to remove the `.scenestettings` file and reimport your FBX meshes and materials.

To see a sample `.fbx` file, go to `\dev\SamplesProject\Objects\Tutorials\Fbx`.

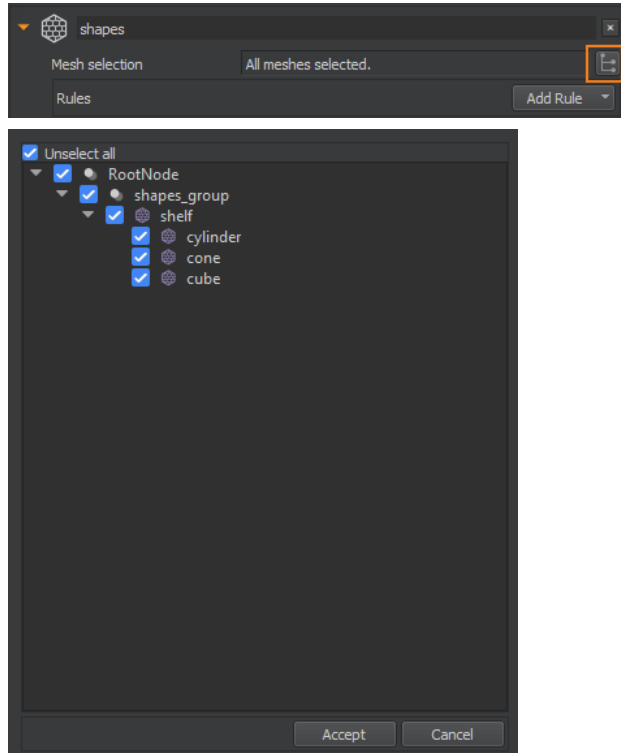
The following figure shows an example of the **FBX Importer** window when you first open it.



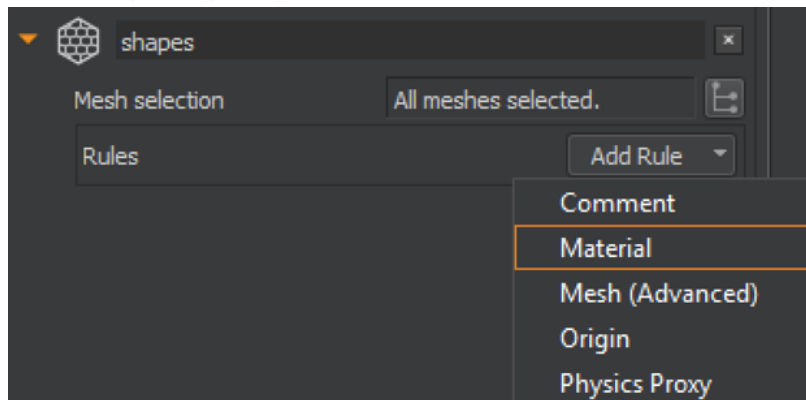
To use FBX Importer

1. In Lumberyard Editor, select **View, Open View Pane, FBX Importer**.

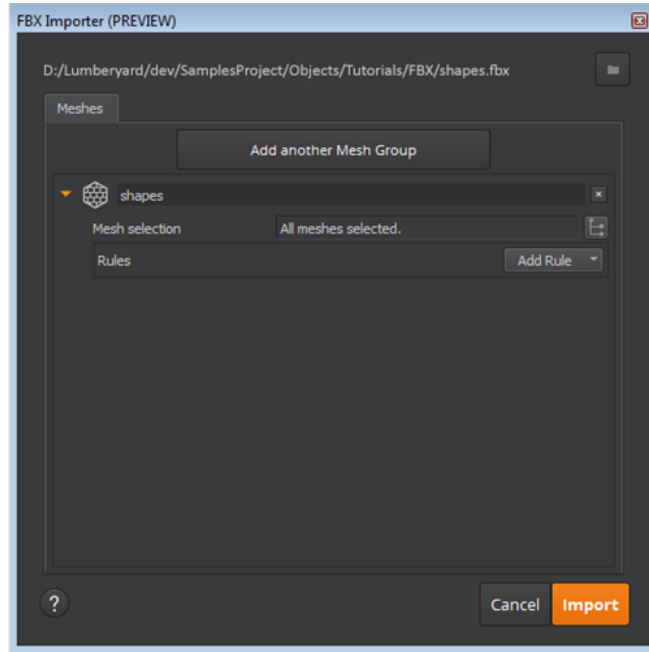
2. Click the folder icon at the upper right of the tool window and select the desired FBX asset. The .fbx file and associated texture must exist in your Lumberyard project.
3. Choose which mesh(es) you want to have imported from your FBX. These should be mesh(es) that you want to have rendered. For example, if you have a physics proxy mesh that is different from your render mesh, you would not want to include it under the general meshes to be imported.



4. By default, materials that are assigned to the mesh(es) set for import will be imported automatically with the mesh. If you add the Materials Rule, you can select a checkbox that allows you to turn **Enable Materials** on or off.
5. Click **Add Rules** to add the **Material Rule**. Then deselect the **Enable Materials** option to import the meshes without their associated materials. For more information, see [Importing Materials with the FBX Importer \(p. 197\)](#).

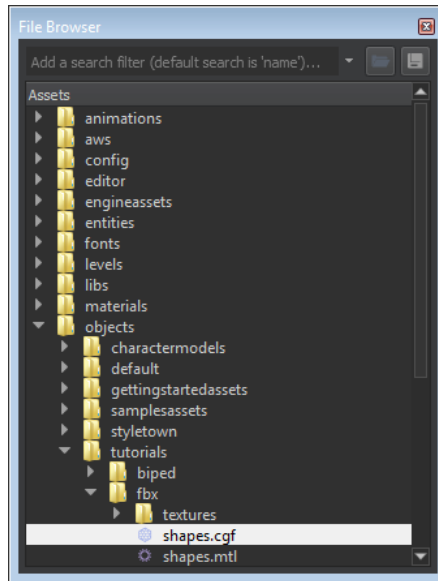


6. Click **Import** to open the mesh (.cgf) and material group files and save asset metadata into a new .scenestettings file. The mesh is processed into the cache and is available in the **File Browser**. If a material file (.mtl) does not exist for the associated mesh, Lumberyard creates one and places it in the same location as the mesh (.cgf) file.



To access the File Browser

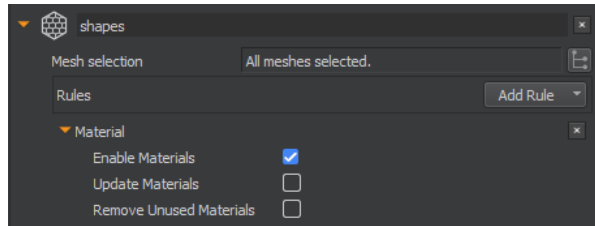
- In Lumberyard Editor, click **View, Open View Pane, File Browser**.



Importing Materials with the FBX Importer

By default, the first time you import an `.fbx` file, by default the FBX Importer automatically imports the materials associated with the file's meshes. Any materials that are assigned to the meshes get listed as submaterials within a single material (`.mtl`) file. The `.mtl` file inherits the same name as the imported mesh from the `.fbx` file.

You can use a **Material Rule** to control how materials in the `.fbx` is imported. You can reset the existing material (`.mtl`) file with the **Reset File** check box, and use the **Enable Materials** check box to import materials from the `.fbx` file. The steps are described in [Working with the FBX Importer](#) (p. 195).



When re-importing an FBX asset, the default behavior is to not overwrite an existing material `.mtl` file. The reason is that after importing a material (`.mtl`) file for the first time, users may have used the **Material Editor** to add texture maps or change lighting settings. You can use the **Reset File** check box if you need to reimport the materials when you reimport the `.fbx` file.

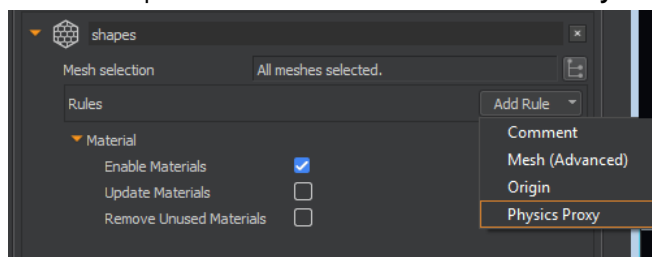
Importing Physics Mesh for a Static Object

You need a physics mesh for static objects that require collision detection or avoidance. Any mesh within an FBX can be used as a physics mesh. You also need to add the physics proxy material to the material (`.mtl`) file.

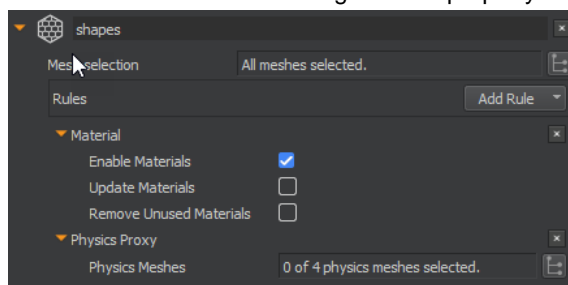
To see a sample `.fbx` file, go to `\dev\SamplesProject\Objects\Tutorials\Fbx`.

To import a physics mesh for a static object

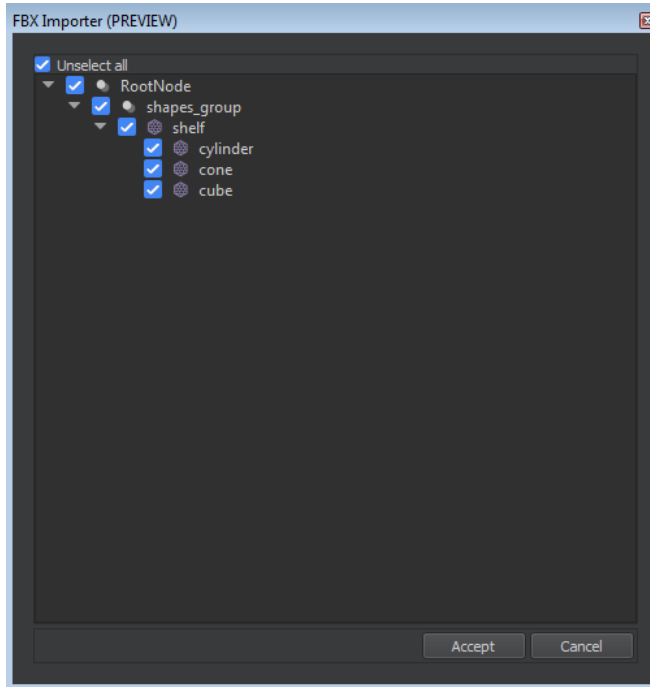
1. In Lumberyard Editor, click **View, Open View Pane, FBX Importer**.
2. Click the folder icon at the upper right of the tool window and select the desired FBX asset. The `.fbx` file and associated texture must exist in your Lumberyard project.
3. Click the drop-down menu for **Add Rule** and select **Physics Proxy**.



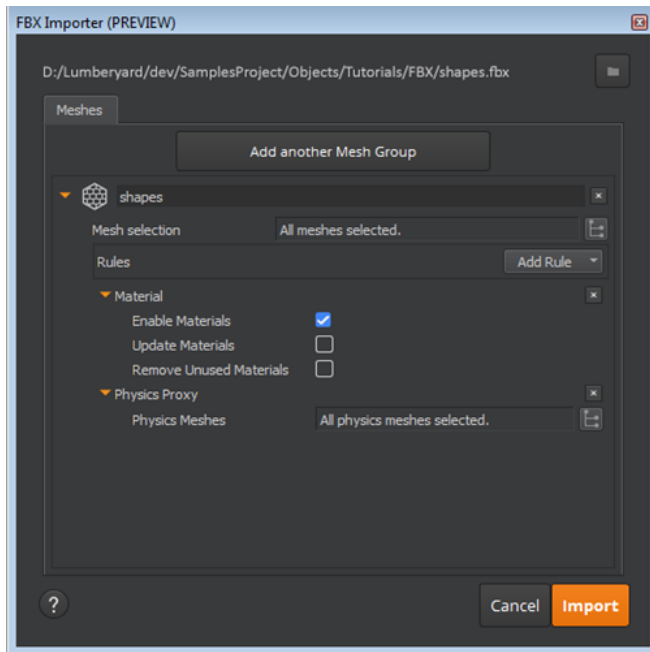
4. Click the **Add Rule** button to add the **Physics Rule**.
5. There is a **Physics Meshes** property under the **Physics Rule** named **all meshes selected**. Click on the icon on the far right of this property.



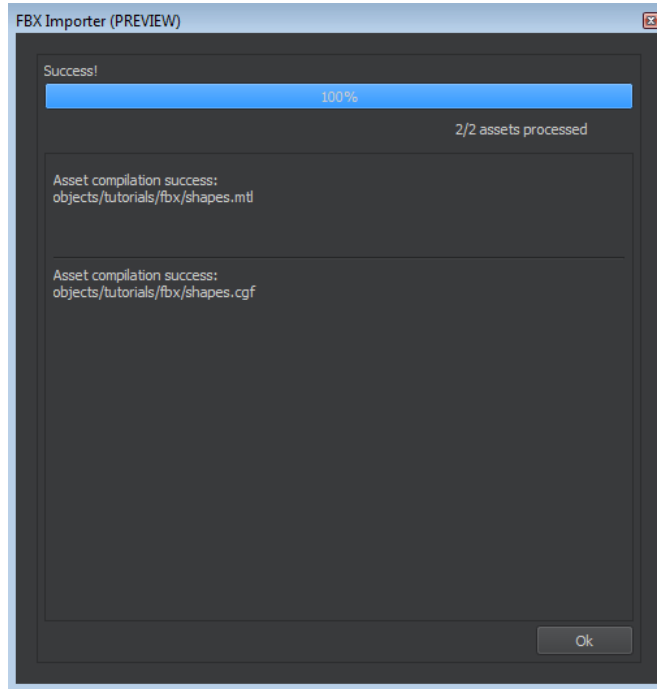
6. In the node selection window, select the meshes that you want to use for your physics proxy.



7. When you have selected the meshes you want, click **Accept** at the bottom. The property shows the number of meshes being used as physics meshes.



8. Click **Import**. The progress bar will appear with a report on the status, and if there are issues with the import, they will display in the window.



Additional FBX Importer Features and Settings

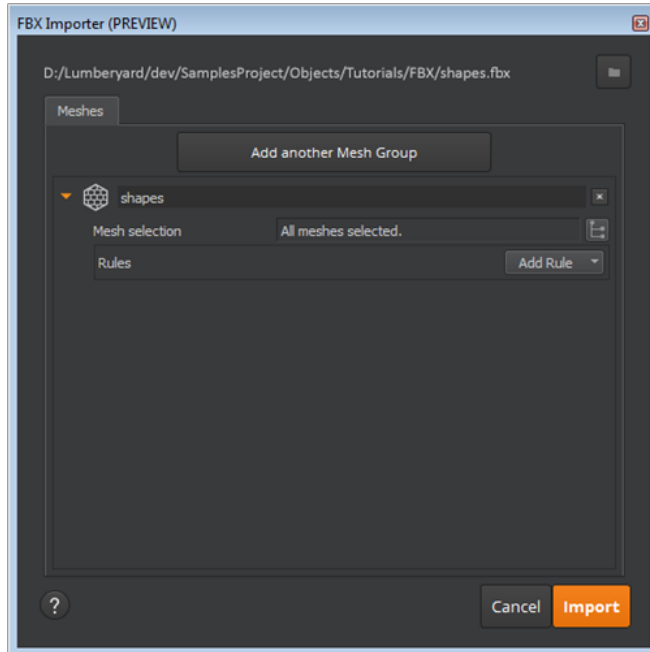
The FBX Importer includes the following features:

Add Group Configuration

You can use a single `.fbx` file to create multiple mesh files (`.cgf`). Click **Add Group Configuration** to define additional `.cgf` assets from a single `.fbx` file.

Selecting a Mesh to Import

By default the FBX Importer imports all meshes in an FBX scene. To include or exclude a mesh, click the **all meshes selected** icon to the far right of the file name to see the available meshes in selection mode.



Meshes that are selected (checked) are included in the processed mesh file. Meshes that are not selected are ignored. When you have selected or deselected the desired import meshes, click **Select**.

Using Rules

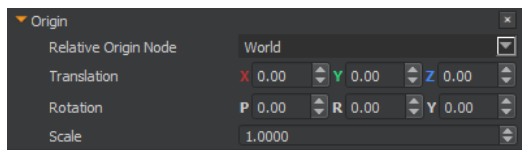
Rules provide useful, extensible, and flexible ways to affect the game data produced by processing a group.

To add a rule

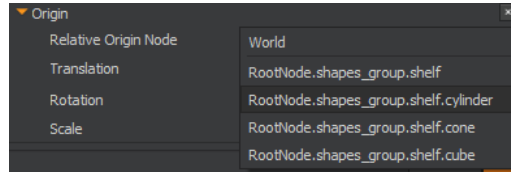
1. In Lumberyard Editor, click **View, Open View Pane, FBX Importer**.
2. In the FBX Importer, browse to the `.fbx` file that you want to import.
3. Click **Add Rule** and select the type of rule: **Comment, Materials, Advanced, or Origin**.
4. Click **Add Rule**.

Origin Rule

You can use the **Origin Rule** to change the position (**Translation**), orientation (**Rotation**), and **Scale** of a mesh relative to how it was authored.



By default the mesh origin is placed at the scene origin of the `.fbx` file. By selecting **World** in the **Relative Origin Node** menu, you can select the transform of any node in the scene. Lumberyard imports the mesh relative to the selected transform.



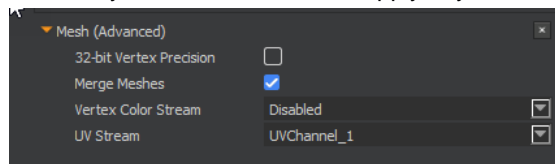
Mesh (Advanced) Rule

You can use the **Mesh (Advanced) Rule** to process mesh data in the asset pipeline.

Use the **Use 32-bit Vertex Precision** option for higher precision vertex data.

When you merge meshes, this action combines all sub-meshes into a single mesh for optimization.

Lumberyard supports the ability to import vertex coloring. If available, the vertex color stream is accessible in the **Vertex Color Stream** menu. If the `.fbx` file contains multiple vertex color streams, you can choose one to apply to your mesh and material settings.



Material Rule

`.fbx` files can include materials used for the mesh. When an `.fbx` file is imported, an `.mtl` file is generated to represent the materials inside the engine. Also reflected within the `.mtl` file are the Physics Rule and option for using the vertex color streams from the Advanced Mesh Rule.

The following restrictions apply for the Material Rule:

- `.fbx` files control only texture maps in the `.mtl` file.
- Materials in the `.fbx` file are matched to the materials in the `.mtl` file by the **Name** field. Duplicate name fields are not supported.

The Material Rule controls the following:

- The use of materials for the meshes that are imported by the FBX Importer.
- The ability to update materials.
- The ability to merge behavior.

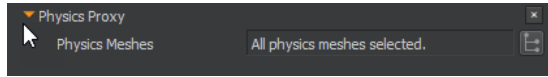
If you do not explicitly add a rule, the FBX Importer functions as if **Enable Materials** is selected and both **Update Materials** and **Remove Unused Materials** are deselected.

You can customize the Material Rule using the following options:

- **Enable Materials** – When selected, an `.mtl` file is generated and linked with the `.cgf` file that the FBX Importer produces. If an existing `.mtl` file is found, new materials in the `.fbx` file will be added but no existing materials will be updated or modified.
- **Update Materials** – When selected, an existing `.mtl` file is updated to use the relevant settings from the `.fbx` file. This feature is limited to the texture map file names.
- **Remove Unused Materials** – When selected, any material that is present in an existing `.mtl` file but not present in the `.fbx` file is removed from the `.mtl` file.
- **Disable Materials** – If the Material Rule is deleted, materials are not processed. Use the **Add Rule** button to add material rules to groups that support materials.
- **Physics Rules** – If present, an `.mtl` file is generated and a physics material is added. Removing a Physics Rule removes the physics material from the `.mtl` file.
- **Mesh (Advanced)** – When added, a vertex color stream can interact with existing `.mtl` files.

Physics Rule

Use the **Physics Rule** to import physics mesh(es) for your object. You can select which mesh(es) from the `.fbx` file to use for physics.

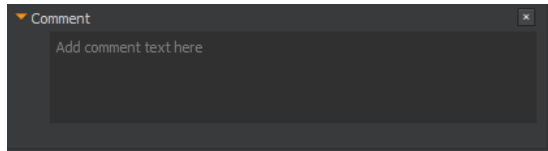


When you use this rule, you must also add a physics proxy material to the material (`.mtl`) file. If a material already exists before you re-import your object with physics mesh(es), a window will pop up after clicking **Import** to confirm if you want to overwrite your existing material. You will want to confirm this action by clicking **Overwrite** so your material has the appropriate physics proxy **no draw** material added so your physics mesh gets treated as a non-rendered collision mesh.

If you choose **Save New**, **FBX Importer** generates a new `.mtl` file with an incremented numerical suffix in the file name. You must manually merge the new file with the previous `.mtl` file if there is a significant amount of editing on the previous file and you don't want to lose this information.

Comment Rule

Use the **Comment Rule** to leave notes and edits for your team members.



Using Multiple UV Streams

The FBX Importer allows you to import multiple UV streams and then select one UV stream to apply to the selected mesh or multiple meshes. If you import multiple meshes, they must all use the same UV stream. The `.fbx` file uses the first UV stream detected by default. You can choose to apply another UV stream to a mesh.

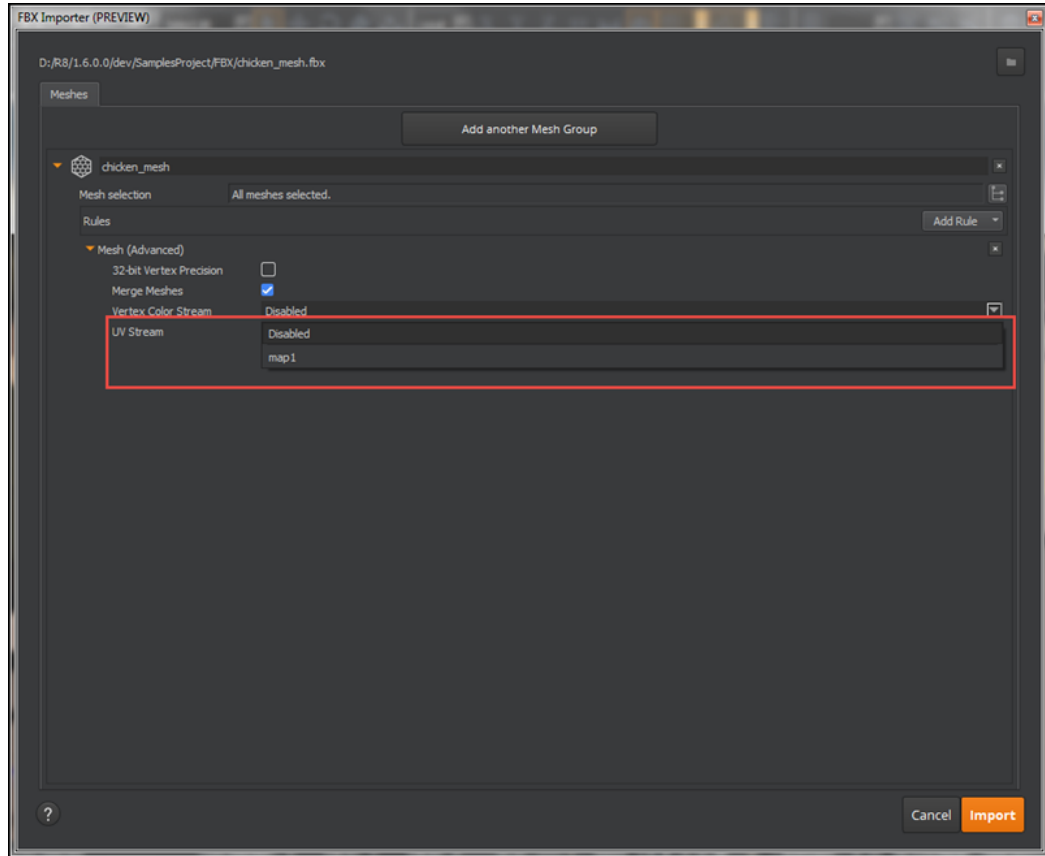
To apply a UV stream to a mesh

1. In Lumberyard Editor, click **View**, **Open View Pane**, **FBX Importer**.
2. In the FBX Importer, browse to the `.fbx` file that you want to import.
3. To select a different UV stream, click **Add Rule**, **Mesh (Advanced)**.
4. Under **Mesh (Advanced)**, select an option from the **UV stream** drop-down list.

Note

Select **Disabled** to set all UV coordinates to 0,0.

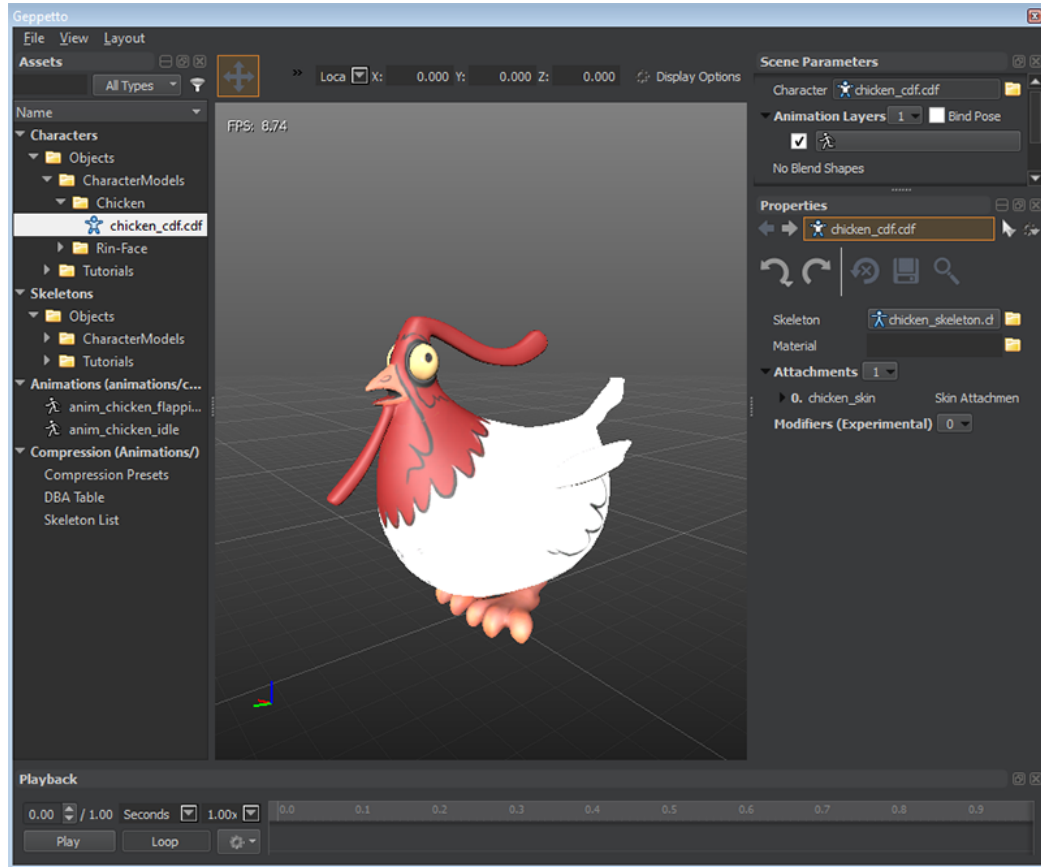
5. Click **Import** to create the `.cgf` mesh with the selected UV mapping.



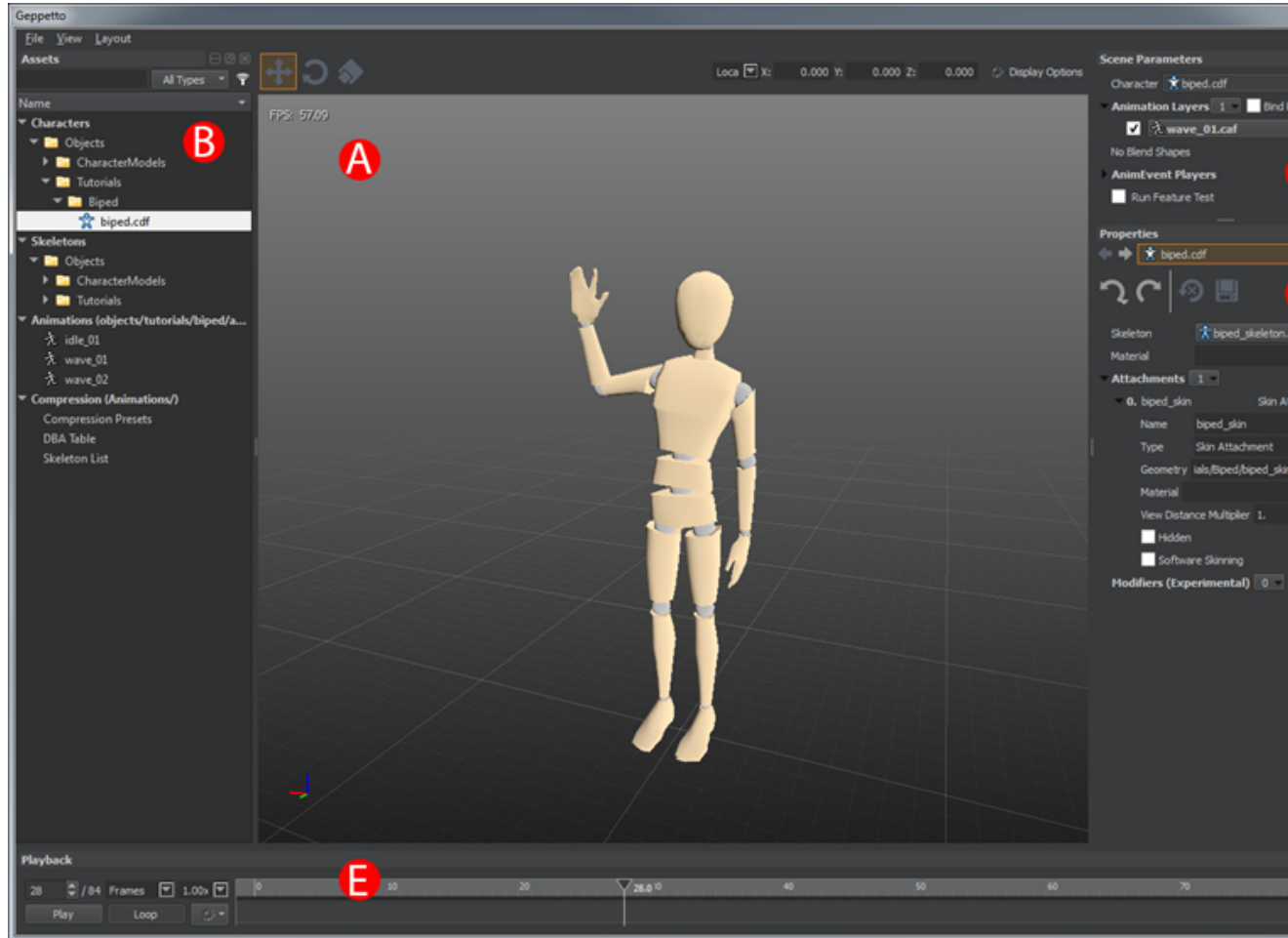
Using Geppetto

Geppetto is used to set up fully animated characters in Lumberyard, in preparation for use with either custom game code to select, play, and blend animations, or with the Mannequin animation controller system. In Geppetto you build a character by associating one or more skinned models with an animation skeleton (built in a DCC like 3DS Max or Maya), and specifying a list of animations (built in a DCC like 3DS Max or Maya) to use with that character.

Animations can also be combined together into blend spaces, which are collections of similar animations that can smoothly blend together to vary travel speed, turning speed, travel angle, slope, turn angle, and travel distance. Blend spaces allow you to easily author natural, complex locomotion for characters. You can use Geppetto to add attachments to the character, such as weapons or other props, including physically simulated attachments that are connected by springs, pendulums, and strings of joints, allowing you to model clothing attachments, capes, and large scale movement of hair. Geppetto also allows you to preview animations and blends between animations on the characters you define, set compression settings for game ready animation data, and compare compressed and uncompressed animations.



To access Geppetto from Lumberyard Editor, choose **View, Open View Pane, Geppetto**. Geppetto has the following UI:



A. Viewport window

Displays the loaded character. Use the WASD keyboard for movement and the right mouse button for camera rotation.

B. Assets pane

Lists all character assets, skeletons, animations, and compression settings. Each asset item has a context menu with available options. When an asset is selected, its properties are displayed in the **Properties** panel.

There are multiple ways to filter the tree in the **Assets** panel:

- By name. It is possible to specify multiple strings separated by a space to look for substrings. For example, **walk relaxed** looks for any name that contains both "walk" and "relaxed".
- By type
- Using advanced filtering options, like presence of events or location of file.

You can have multiple instances of the **Assets** window open. To create a new instance, choose **Split Pane Assets** navigation bar.

C. Scene Parameters panel

This panel is used for previewing purposes and consists of the following:

- **Character name** – Used to select and load a new character by clicking the folder icon. When a character is loaded, you can use the button to select a CDF so you don't have to locate it in the Assets tree every time.
- **Animation Layers** – Location where the played animations are set up. Whenever you select an animation in the **Assets** panel, one is assigned to the active animation layer, which is highlighted with bold text. Add new animation layers using the button next to it. Remove animation layers through the context menu. Blend spaces, aimposes, and lookposes expose additional settings.
- **Blend shape** – Shows blend shape sliders when the character contains blend shapes.
- **Audio Setup** – Used to preview sound foleys and footsteps.
- **Run Feature Test** – Used to add and run project-specific tests.

D. Properties panel

Lists character definitions, skeleton, and animation properties.

E. Playback panel

Displays the animation timeline and playback options, such as looping and speed.

All panels can be moved and are dockable.

Note

Because hot reloading of character-related assets is not supported in Lumberyard Editor, you need to close and restart Lumberyard Editor if you modify any characters that pre-exist in a level. This is not necessary for characters that later spawn into a level. This does not apply if you are previewing changes in Geppetto.

Topics

- [Geppetto Display Options \(p. 207\)](#)
- [Creating a Character Definition \(p. 210\)](#)
- [Character Attachments \(p. 211\)](#)
- [Animating Characters \(p. 224\)](#)

Geppetto Display Options

The following is a list of the various display option settings in Geppetto. In the upper-right corner of the Geppetto viewport, choose **Display Options** to access the various settings.

Animation

Movement

Choose between **In Place (Only grid moves)**, **Repeated**, and **Continuous (Animation Driven)** in response to when the character's root joint moves in world space during an animation.

Compression

Choose between **Preview Compressed Only** and **Side by Side (Original and Compressed)** for what to preview for animations.

Animation Event Gizmos

Enables and disables the visibility of animation event gizmos that are tied to a skeleton joint.

Locomotion Locator

Enables and disables the visibility of the locomotion locator for the character, to indicate which direction the root motion or locomotion locator are pointing during an animation.

DCC Tool Origin

Enables and disables the transform display on the DCC origin for the skeleton and also displays the rotation and position information near the top of the viewport.

Reset Character

Allows you to reset the character by forcing it back to bind pose, setting it to viewport origin, and removing any current animations on the character, including the removal of animation layers.

Rendering

Edges

Enables and disables the display of all edges for polygons on meshes. It also displays information regarding the mesh data at the top of the viewport.

Wireframe

Enables and disables the wireframe mode for meshes. If used in combination with **Edges**, it uses a flat colored wireframe instead of the material wireframe.

Framerate

Enables and disables the display of the frame rate for the viewport.

Skeleton

Joint Filter

Allows you to enter text to help filter what joints are displayed so joints are only displayed that have the text somewhere in the joint name. Should be used with **Joints** enabled.

Joints

Enables and disables the display of skeleton joints.

Joint Names

Enables and disables the display of skeleton joint names.

Bounding Box

Enables and disables the display of the bounding box for the character created by the skeleton joints.

Camera

Show Viewport Orientation

Enables and disables the display of the viewport orientation.

FOV

Slider to adjust the camera's FOV.

Near Clip

Slider to adjust the camera's near clip plane.

Move Speed

Slider to adjust the movement speed of the camera, currently capped at 3. The default is 0, not restraining the camera at all. If this parameter is set to an odd number, it does not allow the use of rotation for the camera.

Rotation Speed

Slider to adjust the rotation speed of the camera.

Movement Smoothing section

Position

- Slider for adjusting smoothing for the camera translation.

Rotation

Slider for adjusting smoothing for the camera rotation.

Follow Joint

Joint

Joint that the camera will follow. The default is null so that you can manipulate the camera.

Align

Enables and disables the alignment of the camera to the specified joint based on **Position** and **Orientation**.

Position

Enables and disables the position of the joint to influence the camera.

Orientation

Enables and disables the orientation of the joint to influence the camera.

Secondary Animation

Dynamic Proxies

Enables and disables the display of dynamic proxies.

Auxiliary Proxies

Enables and disables the display of auxiliary proxies.

Physics

Physical Proxies

Enables and disables the display of physics proxies.

Ragdoll Joint Limits

Enables and disables the display of the ragdoll joint limits on the skeleton.

Grid

Show Grid

Enables and disables the display of the grid. There are additional settings for setting the grid main line and middle line color and transparency.

Spacing

Sets the scale of the grid based on meters. The default is 1.

Main Lines

Sets the display of the number of grid main sections.

Middle Lines

Sets the display of the number of middle sections within the grid main sections.

Origin

Enables and disables the display of the viewport origin. When enabled, this parameter gives additional options for adjusting the color and transparency of the origin.

Lighting

Brightness

Sets the brightness of the light. You also have control over the color and transparency of the light.

Rotate Light

Enables and disables the rotation of the light in world space.

Light Multiplier

Sets the multiplier for the light.

Light Spec Multiplier

Sets the multiplier for the specular for the light. You also have control over the color and transparency of the specular for the light.

Background

Use Gradient

Enables and disables the use of gradient with the colors assigned below. If disabled, only one color is available to adjust.

Creating a Character Definition

Using Geppetto, you can create a character definition. The character definition `.cdf` file consists of a skeleton `.chr` file, an animation list that is referenced in a `.chrparams` file, and attachments.

Character Definition File

The XML-based character definition file (`.cdf`) combines different character parts such as skeletons, meshes, materials, and attachments.

Before proceeding, make sure you have the following assets exported from your DCC tool:

- Character skeleton `.chr` file
- Skinned geometry `.skin` file
- One or more character animations

To create a character definition file

1. In Geppetto, choose **File, New Character**, type a file name and path, then choose Save. An empty file is created, but without a skeleton or attachment yet.
2. In the **Properties** panel, choose the folder icon next to **Skeleton**, select the skeleton `.chr` file, and choose **Open** to load the skeleton. This assigns the skeleton to the `.cdf` file.
3. Choose **Display Options** to reveal the **Skeleton** section in the UI.
4. Expand **Skeleton** and choose **Joints**. The skeleton is displayed in the viewport.

Character Skeleton List

Make sure the skeleton is added to the `SkeletonList.xml` file using the following procedure.

To add the skeleton to the list

1. In the **Assets** panel under **Compression (Animations)**, choose **Skeleton List**.
2. In the **Properties** panel under **Aliases**, make sure the skeleton `.chr` file is in the list. If not, do the following:

- a. Choose the number button next to **Aliases** and **Add**.
- b. Choose the folder icon next to the new entry, then select a suitable skeleton.
- c. Name the added skeleton alias. This name is used to refer to the skeleton.

Character Animation List

The character animation list is specified in the `.chrparams` file.

To specify the animation list

1. In the **Asset** panel, expand **Skeletons, Characters** and select the skeleton `.chr` file.
2. In the **Properties** panel, choose the number button next to **Animation Set Filter** and **Add**.
3. Select the folder icon for the new row, open the context (double-click) menu for **Animations**, and then choose **Select Folder**.

Character Attachments

In order to attach something to a character, a socket is needed. Sockets provide the connection between the character and the attachment. For more information, see [Attachment Sockets \(p. 211\)](#).

After a socket has been created and defined, an attachment can be created and connected to the socket. For more information, see [Character Attachments \(p. 211\)](#).

Character Attachments

Attachments are separate objects that are attached to characters, respond to real-world physics, and can be attached, detached, or replaced at runtime in the game.

Lumberyard allows for various skinned, animated, or physicalized attachments to the skeleton or polygonal faces of a character. Attachments are hierarchical, and each attachment can have its own morph targets and animations. You can use skin attachments for entire body parts such as heads, hands, or upper and lower body.

To add or change a character attachment, the character must first be loaded into Geppetto.

Topics

- [Attachment Sockets \(p. 211\)](#)
- [Joint Attachments \(p. 212\)](#)
- [Face Attachments \(p. 213\)](#)
- [Pendula Row \(PRow\) Attachments \(p. 214\)](#)
- [Proxy \(Collision\) Attachments \(p. 216\)](#)
- [Skin Attachments \(p. 217\)](#)
- [Collision Detection and Response \(p. 218\)](#)
- [Secondary Animations \(Simulations\) \(p. 221\)](#)

Attachment Sockets

To attach something to a character, you must first create an attachment socket. A socket is an empty attachment without assigned geometry. Sockets have a name, position/orientation (for joint and face attachments), and attachment type. Attachment sockets can be used by game code to attach objects to characters at runtime, such as replacing weapons attached to a hand. After a socket is created, you can plug a `.cgf` attachment into it.

Tip

To display all empty sockets for a character, use the `ca_DrawEmptyAttachments=1` console variable.

You can also use sockets to achieve simulated motion of joint and face attachments. This type of animation is always a reaction to a primary character motion, and are called secondary animations. Such animations can simulate the movement of attached objects. For more information, see [Secondary Animations \(Simulations\)](#) (p. 221).

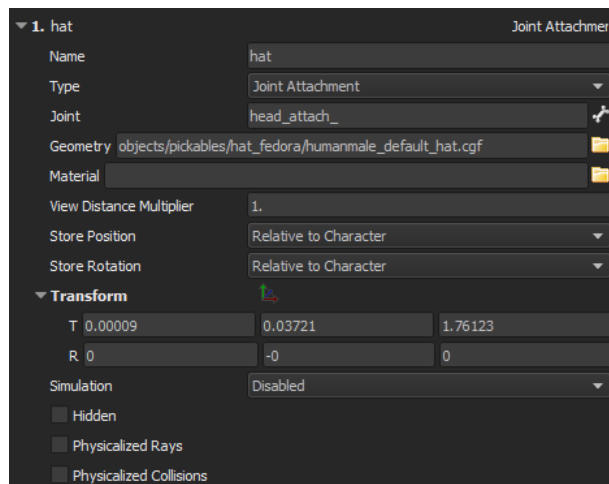
Joint Attachments

Joint attachments require an attachment socket that provides a connection point between the attachment and the character. Use the move and rotate tool to position and orient the socket relative to a bone joint.

The socket is attached to a joint and moves with the joint when the skeleton is animated. Secondary animations can be enabled on a socket and provide additional motions based on a real-world physical simulation and generated in response to the movements of the character. This has the effect of making loosely-attached objects behave more realistically when the character is undertaking fast movements.

These secondary animations can also be redirected to the skeleton of the character to apply the simulated motion to all vertices that are part of the skinned mesh and weighted to the joint. This is very useful when animating hair and cloth. By enabling collision detection, such attachments can also interact with the character.

You can simulate the motion of hair braids and dangling straps using joint attachments. A chain or rope of pendula can be created by attaching a pendulum at each link. When the motion simulation is activated, each parent joint transfers motion to the children. In this case, the primary motion is not coming from an animation, but from a previous motion simulation. Collision detection and response is used to limit the motion of the attachment from moving through the body of the character.



To create a joint attachment

1. In Geppetto, in the **Properties** panel, choose the number next to **Attachments** and **Add or Insert**.
2. For **Name**, enter a name for the attachment.
3. For **Type**, choose **Joint Attachment**.
4. For **Joint**, choose the bone icon, then open the applicable joint to place the socket on.
5. For **Geometry**, choose the folder icon and select the desired * .cgf file for the attachment.
6. For **Material**, choose the folder icon and select the desired * .mtl file for the attachment.
7. Adjust the values of attachment parameters for the desired result, as listed in the following table.

Joint Attachment Parameters

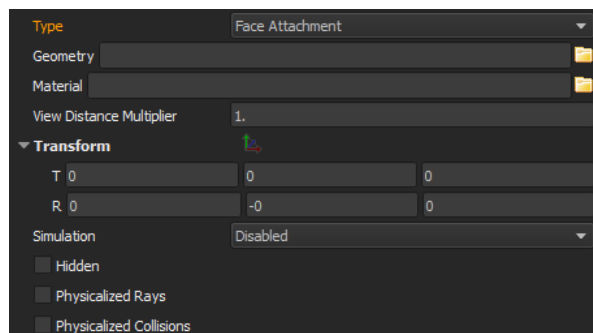
Parameter	Description
View Distance Multiplier	Multiplier to the computed fade-out camera distance to apply on the attachment.
Store Position	Stores position data relative to either the character or to a joint.
Store Rotation	Stores rotation data relative to either the character or to a joint.
Transform	Transform-Translation (T) and Rotation (R) vectors for the X, Y, and Z axes in relation to Store Position and Rotation .
Simulation	Type of simulated motion. Disabled is on by default, but types consist of: Pendulum Cone, Pendulum Hinge, Pendulum Half Cone, Spring Ellipsoid, and Translational Projection. For more information, see Secondary Animations (Simulations) (p. 221).
Hidden	Hides the attachment.
Physicalized Rays	Enables hit ray detection if a physics proxy is available.
Physicalized Collisions	Enables collision detection if a physics proxy is available.

Face Attachments

Face attachments require an attachment socket that provides a connection point between the attachment and the character. The socket is attached to a specific triangle on the mesh surface and moves along with the triangle when the skeleton is animated and the mesh gets deformed. The location of the face attachment can be relative to the triangle and it is possible to assign face attachments to all skinned meshes of a character.

It is recommended that the character be first put into its bind pose. To do so, in Geppetto, in the **Scene Parameters** panel, choose **Bind Pose** next to **Animation Layers**.

When you move the socket using the using the gizmo tool in the viewport, it automatically connects to the closest triangle in the mesh.



To create a face attachment

1. In Geppetto, in the **Properties** panel, choose the number next to **Attachments** and **Add** or **Insert**.
2. For **Name**, enter a name for the attachment.
3. For **Type**, choose **Face Attachment**.
4. For **Geometry**, choose the folder icon and select the desired * .cgf file for the attachment.

- For **Material**, choose the folder icon and select the desired *.mtl file for the attachment.
- Adjust the values of attachment parameters for the desired result, as listed in the following table.

Face Attachment Parameters

Parameter	Description
View Distance Multiplier	Multiplier to the computed fade-out camera distance to apply on the attachment.
Transform	Transform-Translation (T) and Rotation (R) vectors for the X, Y, and Z axes in relation to Store Position and Rotation .
Simulation	Type of simulated motion. Disabled is on by default, but types consist of: Pendulum Cone, Pendulum Hinge, Pendulum Half Cone, Spring Ellipsoid, and Translational Projection. For more information, see Secondary Animations (Simulations) (p. 221).
Hidden	Hides the attachment.
Physicalized Rays	Enables hit ray detection if a physics proxy is available.
Physicalized Collisions	Enables collision detection if a physics proxy is available.

Pendula Row (PRow) Attachments

To create a pendula row attachment

- In Geppetto, in the **Properties** panel, choose the number next to **Attachments** and **Add or Insert**.
- For **Name**, enter a name for the attachment.
- For **Type**, choose **PRow Attachment**.
- For **Joint Row Name**, choose the bone icon, then open the applicable joint to place the socket on.
- Adjust the values of attachment parameters for the desired result, as listed in the following table.

Pendula Row Parameters

Parameter	Description
Clamp Mode	Used to select the movement bounding volume of the pendula row: Cone, Half Cone, Hing, or Translational Projection.
Debug Setup	When enabled, displays a green bisected spherical shape that represents the bounding volume for the simulated object's pivot.
Debug Text	Enable to display debugging text in the viewport.
Activate Simulation	Enable to activate the physics simulation for springs and pendula.
Simulation FPS	Used to specify the frame rate of the physics simulation updates. A value of 30 indicates 30 updates per second. The valid value range is 10–255 fps, with a recommended range of 30–60 fps. This value should ideally be the same as the game frame rate.
Mass	Used to specify the mass of pendula bobs. If the value of the Joint Spring parameter is zero, the Mass value has no impact on the oscillation period.

Parameter	Description
Gravity	Used specify the force of gravity on pendula. While the mass of a bob has no effect on the oscillation of a pendulum, the force of gravity does. The default value of 9.81 represents Earth's gravitational force.
Damping	Used to specify a velocity-dependent force such as air resistance. The faster that pendula move, the more force that is encountered, decelerating the pendula at a rate proportional to the velocity. Greater damping values result in pendula coming to rest more quickly.
Joint Spring	Used to simulate position dependent forces, and is a value between 0-999 applied to the spherical joint. The further the pendulum swings away from the axis of the spring target , then the harder it tries to return.
Cone Angle	Used to specify the pendula starting movement angle for cone, half-cone, and hinge-planes bounding volumes. Valid range is from 0-179 degrees, where values greater than 90 degrees form an inverse cone.
Cone Rotation	Used to specify the amount of rotation relative to joints along the X, Y, and Z axes for cone, half-cone, and hinge-planes.
Rod Length	Used to specify the length of pendula row rods, which impacts swinging frequency. The longer the rods, the longer the pendula oscillations.
Spring Target	Used to specify two planes of rotation around the X-axis of the joints.
Turbulence	Used to control frequency and amplitude of noise added to PRow joints to simulate wind and similar effects on cloth.
Max Velocity	Used to clamp the velocity of the PRow pendula bobs in order to control large impulse spikes from character movements.
Cycle	Select to attach the last joint in the pendula row to the first joint to form a horizontal circle. Used for cloth skirts.
Stretch	Used to define the horizontal distance between pendula row joints, which defines how much cloth can stretch or shrink horizontally. A value of 0.2 indicates a stretching or shrinking of 20%.
Relax Loops	Used to iteratively keep pendula row joints together horizontally. Each iteration brings the joints closer together for each frame. A value between 2-4 is recommended.
Capsule	Defines the length and radius values for the capsules used for the dynamic (blue) proxies connected to each joint in the entire pendula row. Used for collision detection.
Projection Type	Choose Shortarc Rotation to activate collision detection.

Proxy (Collision) Attachments

Collision detection and response involves the realistic animation of attachments that collide with the body of a living character to simulate real-world physics. To handle this, a special attachment called a collision proxy is used. Collision proxies are normal attachments that are linked to joints and move with the skeleton. Using a collision proxy is more efficient than undertaking all the necessary computation required for collision detection and response with a polygonal mesh.

Two different types of collision proxies are used:

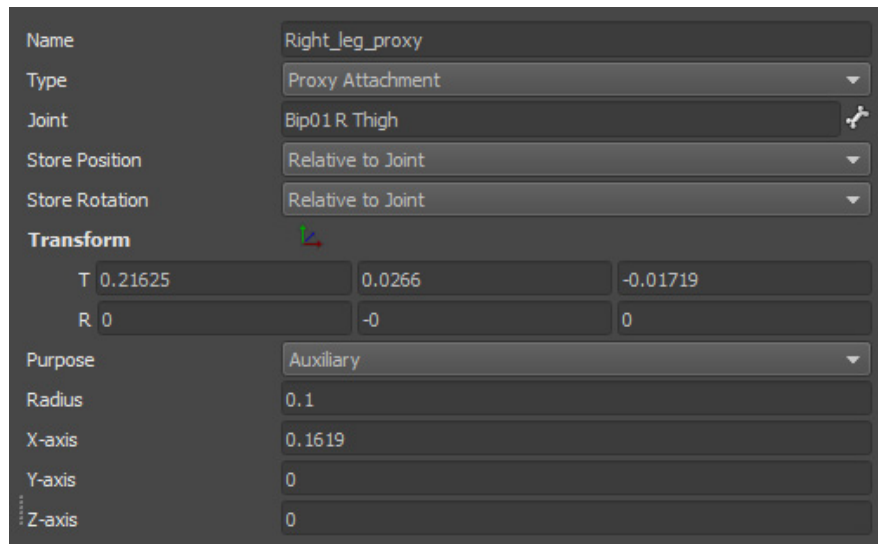
- **Auxiliary proxies (lozenges)** – Called lozenges, these are represented in gray by simple geometric objects linked to joints that move with the skeleton, and represent an approximation of a body shape. Gray proxies handle collision detection and response with the character and are normal attachments.
- **Dynamic proxies** – These are represented in blue by capsules and spheres and are a property of a socket. Blue proxies handle collision detection and response between gray proxies. Blue proxies are dynamic collision proxies, which means that gray proxies always push blue proxies away.

Collision detection is detecting when an overlap occurs between an auxiliary proxy and a dynamic proxy. For both proxy types, you can tweak the size, shape, and other physical parameters interactively while a character animation is running and see the effect immediately.

Auxiliary Proxies (Lozenges)

An auxiliary proxy lozenge is defined by a radius and scaling values for the X, Y, and Z axes. Using these four numbers, points, line-segments, rectangles, boxes, spheres, 1D lozenges (capsules), 2D lozenges, and 3D lozenges can be created. These eight shapes are used to approximate the shape of arms, legs, and torso of a living character.

The following figure shows a capsule shape defined for the thigh joint on the right leg of a character.



To set up an auxiliary proxy (lozenge)

1. In Geppetto, choose **Display Options** to reveal the **Secondary Animations** section, then select the **Auxiliary Proxies** check box.
2. In the **Properties** panel, choose the number next to **Attachments** and then choose **Add** or **Insert**.
 - For **Type**, choose **Proxy Attachment**.

- For **Joint**, choose the bone icon; in the **Choose Joint** window, select the joint to attach the lozenge to and choose **OK**.
- For **Purpose**, choose **Auxiliary**.
- For **Radius**, enter a value in meters.
- For **X-axis**, enter a value in meters.
- For **Y-axis**, enter a value in meters.
- For **Z-axis**, enter a value in meters.

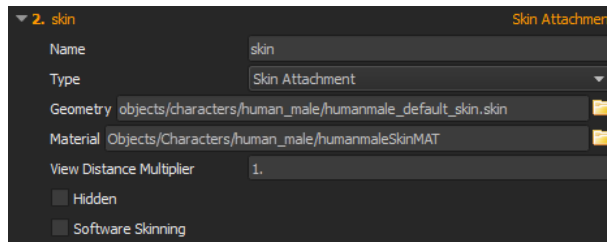
The axes scale in both directions, so entering values of 0,1,1,1 creates a box of 2x2x2 meters.

Dynamic Proxies

Dynamic (blue) proxies handle collision detection and response between gray proxies. Blue proxies are dynamic collision proxies, which means that gray proxies always push blue proxies away. For more information, see [Collision Detection and Response \(p. 218\)](#).

Skin Attachments

Skin attachments have a skeleton of their own, making it possible to replace body parts such as heads, hands, or upper and lower body parts. Furthermore, these parts are automatically animated and deformed by the base skeleton. The use of skinned attachments that have more joints and different joints than the base skeleton is also supported using *Skeleton-Extensions*. It is also possible to merge different types of skeletons together, even skeletons from totally different characters.



To create a skin attachment

1. In Geppetto, in the **Properties** panel, choose the number next to **Attachments** and then choose **Add** or **Insert**.
2. For **Name**, enter a name for the attachment.
3. For **Type**, choose **Skin Attachment**.
4. For **Geometry**, choose the folder icon and select the desired *.skin file for the attachment.
5. For **Material**, choose the folder icon and select the desired *.mtl file for the attachment.
6. Adjust the values of attachment parameters for the desired result, as listed in the following table.

Skin Attachment Parameters

Parameter	Description
View Distance Multiplier	Multiplier to the computed fade-out camera distance to apply on the attachment.
Hidden	Hides the attachment
Software Skinning	If enabled, the mesh gets skinned on the CPU instead of the GPU. Software skinning is required for blendshapes and to have tangent frames recalculated every frame.

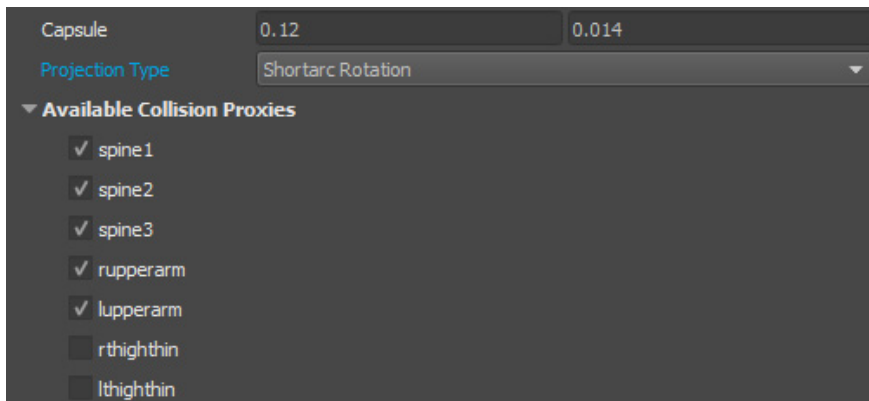
Collision Detection and Response

Collision detection and response involves the realistic depiction of attachments that collide with the body of a living character that simulate real-world physics. To do this, a collision proxy is used to approximate parts of a character body such as the legs and torso with a simple geometry shape. Using a collision proxy is more efficient than undertaking all the necessary computation required for collision detection and response with a polygonal mesh.

For information about how to set up a collision proxy attachment, see [Proxy \(Collision\) Attachments](#) (p. 216).

Collision Detection

Collision detection involves checking to see if a blue proxy capsule/sphere overlaps (collides) with a gray proxy lozenge. For pendulums, a blue proxy is always connected to an attachment socket (pivot) at one end.



Collision Response

Collision response is handled by projections. If a dynamic (blue) proxy capsule/sphere collides with an auxiliary (gray) proxy lozenge, the blue proxy is projected or moved away until it no longer overlaps (collides with) the gray proxy lozenge. This means projecting (moving) the blue proxy capsule/sphere perpendicularly from the lozenge surface or rotating it out of the lozenge.

Lumberyard performs two consecutive constraint checks for collision detection. First, the blue proxy capsule/sphere is moved out of the gray proxy lozenge, and second the spring particle or pendulum rod movement is clamped to the shape of the bounding volume: spring ellipsoid, pendulum cone or half-cone, pendulum hinge, or translation projection.

After these two checks, the blue proxy capsule/sphere should be outside of the gray proxy lozenge but inside of the bounding volume. However, if the bounding volume is too small, the collision response may happen successfully only to have the bounding volume push the capsule/sphere back inside the lozenge.

There are four different projection methods used to move proxies to a non-colliding state, depending on the bounding volume, in addition to **No Projection**:

Topics

- [Spring Ellipsoid Response](#) (p. 219)
- [Pendulum Cone and Half-Cone Response](#) (p. 219)
- [Pendulum Hinge Response](#) (p. 219)
- [Translational Projection Response](#) (p. 219)

If **No Projection** is selected, collisions are ignored and no response is initiated.

Spring Ellipsoid Response

Selecting **Shortvec Translation** moves a gray proxy sphere away from a blue proxy lozenge using the shortest distance possible. For springs, only gray proxy spheres (and not capsules) are supported with spring motions.

Pendulum Cone and Half-Cone Response

Selecting **Shortarc Rotation** rotates a gray proxy capsule out of a blue proxy lozenge using the smallest angle possible.

Pendulum Hinge Response

Selecting **Shortarc Rotation** rotates a gray proxy capsule out of a blue proxy lozenge using the shortest direction possible. For hinges, there are only two ways for a capsule to rotate out of a lozenge.

Selecting **Directed Rotation** rotates a gray proxy capsule out of a blue proxy lozenge along the (green) direction of the hinge-plane.

Translational Projection Response

In the case of rotations (**Shortarc Rotation** and **Directed Rotation**), the pivot for a blue proxy capsule must lie outside of a gray proxy lozenge. The pivot is the spherical portion of the capsule that is connected to the attachment socket.

If the capsule pivot lies inside of a lozenge, collisions cannot be resolved and the proxies remain in an overlapping (collided) state. This can occur for secondary animations on characters where the simulation update is triggered after the animation update and it happens that the animation itself moves proxies into each other or creates invalid proxy configurations that break the simulation. To handle these cases, **Translational Projection** type is used, which defines the direction of movement. There are two types of translational projections:

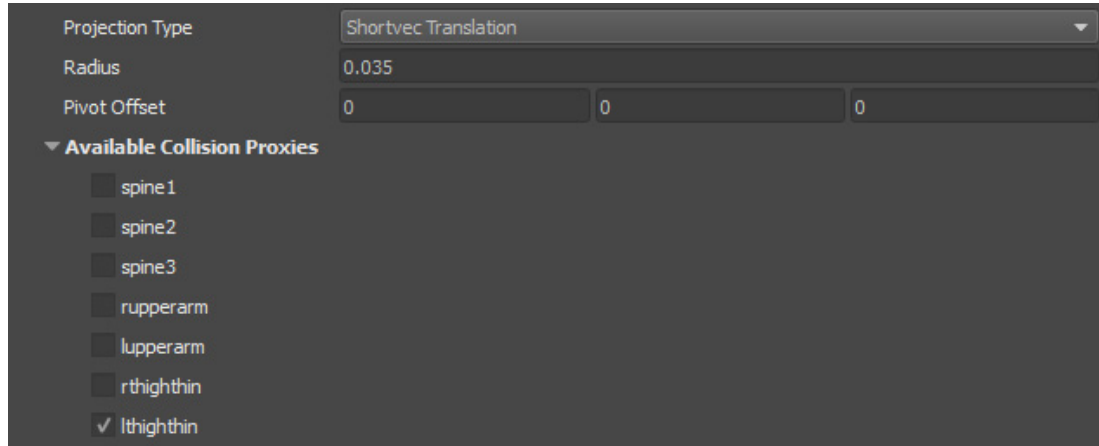
- **Shortvec Translation**
- **Directed Translation**

Note

It is important that the new socket is on the same joint where you want to perform the translation and appears in the list of attachments ahead of the pendulum attachment that you want to move out. You can change the order of attachments in Geppetto. This order defines the order of execution, so the translation operation moves the joint out of the proxies before the pendulum attachment is executed.

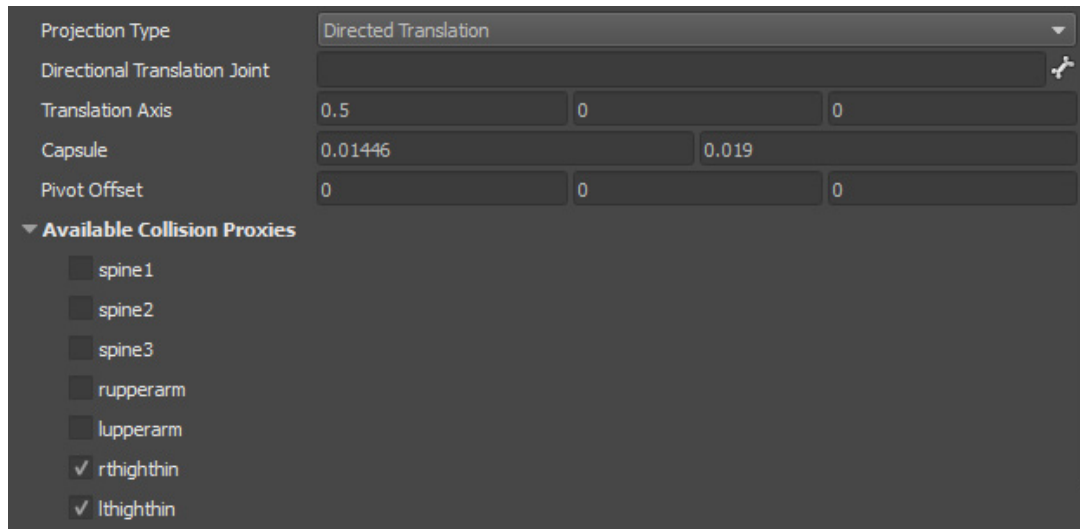
Selecting **Shortvec Translation** moves a blue proxy capsule out of a gray proxy lozenge along the shortest vector from the surface of a sphere enclosing the joint, where the radius of the sphere is specified. This type should be used in cases where there are only a few lozenges, due to potential unpredictable and undesirable movements.

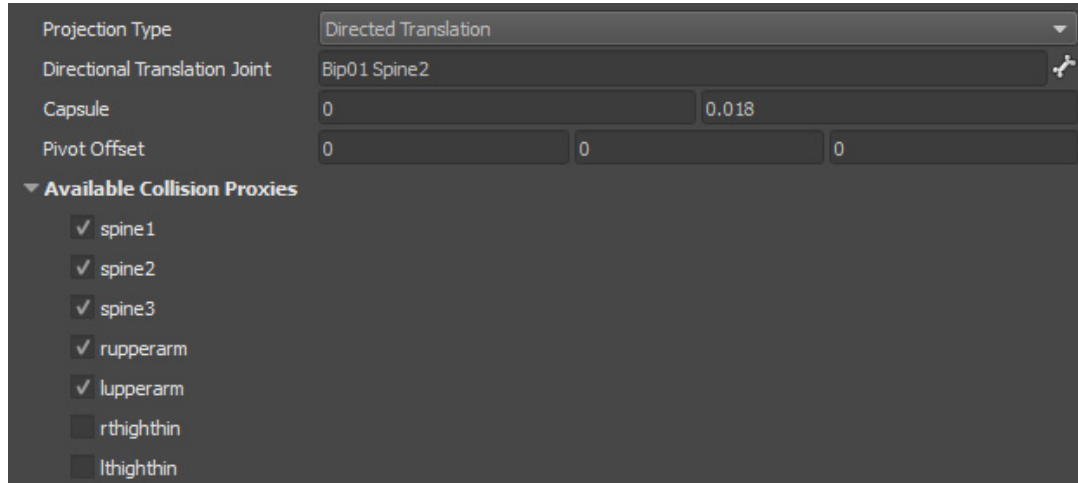
If an overlap is detected, the sphere is translated out of the lozenge along the shortest vector to the surface. This method of translation is only recommended for setups with just a few lozenges where the results are predictable. Otherwise, issues may arise where the first shortvec translation moves a capsule out of the first lozenge and directly into a second lozenge. These issues are very likely with complex setups where many lozenges are close together or overlap. It can also happen that it projects out in the wrong direction and produces undesired "tunneling" effects.



Choosing **Directed Translation** moves a blue proxy capsule out of a gray proxy lozenge along either a **Translation Axis** (defined relative to a joint and socket orientation) in its negative direction, or moves it out relative to a selected **Directional Translation Joint**, which defines the translation axis between the joint and socket. Optionally, you can select a joint, which forms a translation axis between the location of the joint and the socket.

Both options allow you to specify a capsule in the direction of the translation axis; however, the capsule is always projected out in the predefined direction even if the capsule is behind the lozenge, which makes "tunneling" unlikely.





Secondary Animations (Simulations)

You can also use sockets to produce realistic movements of joint and face attachments. This type of animation is always a reaction to a primary (character) animation, and are called secondary animations or motion simulations. Such animations can simulate the movement of attached static objects such as weapons and holsters, muscles, and fat.

In addition, it is also possible to create complex setups to simulate the motions of swinging hair braids, tentacles, chains, ropes, necklaces, clothing, and other loose or dangling objects on a character. Chains can have branching strings and different physical properties for each link.

However, such motions are just approximations of real-world physical movements. In Lumberyard, the physical properties of springs and pendula are used to approximate (simulate) the physical movement of dangling or swinging objects attached to characters.

- **Pendulum:** A bob connected to a rigid rod that experiences simple harmonic motion as it swings back and forth. The equilibrium position of an unconstrained pendulum is hanging directly downward. The swing is specified by physical parameters such as stiffness and stiffness target, and movement is constrained by cone, half cone, or hinge plane bounding volumes.
- **Spring Ellipsoid:** A bob connected to an elastic rod. Unlike a helical spring, a spring ellipsoid can stretch in any direction. The movement of the spring is constrained to by sphere, ellipsoid, half sphere, flat plane, or line bounding volumes.

Moving springs and pendula have different motion bounding volumes that constrain the movement of objects attached to characters.

While the type, size, and shape of the attachment has no impact on its actual motions, it does determine which type of simulation is selected as the movements of a corresponding real-world physical object must be simulated. In this way, the socket and attached object realistically react to the movements of the character.

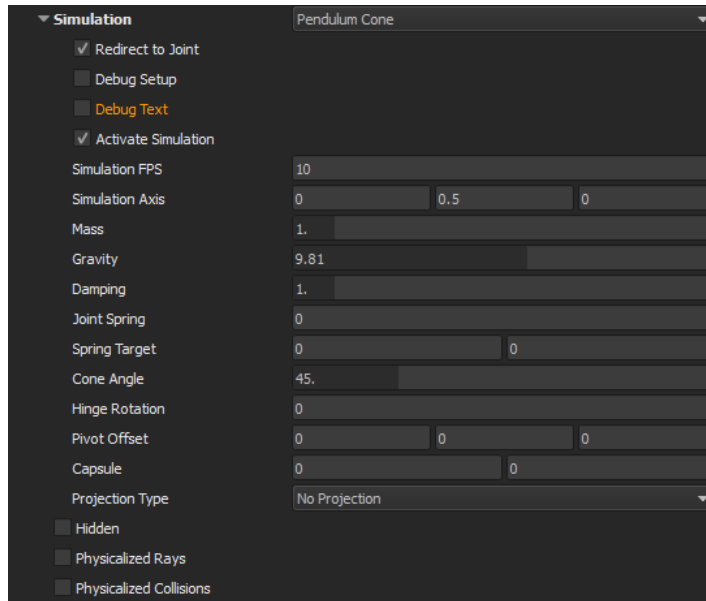
In addition, because moving attachments may collide with the character, this must be accounted for. For more information, see [Proxy \(Collision\) Attachments \(p. 216\)](#) and [Collision Detection and Response \(p. 218\)](#).

Topics

- [Pendulum Cone Simulation \(p. 222\)](#)
- [Pendulum Half-Cone Simulation \(p. 222\)](#)
- [Pendulum Hinge Simulation \(p. 223\)](#)

- [Spring Ellipsoid Simulation \(p. 224\)](#)

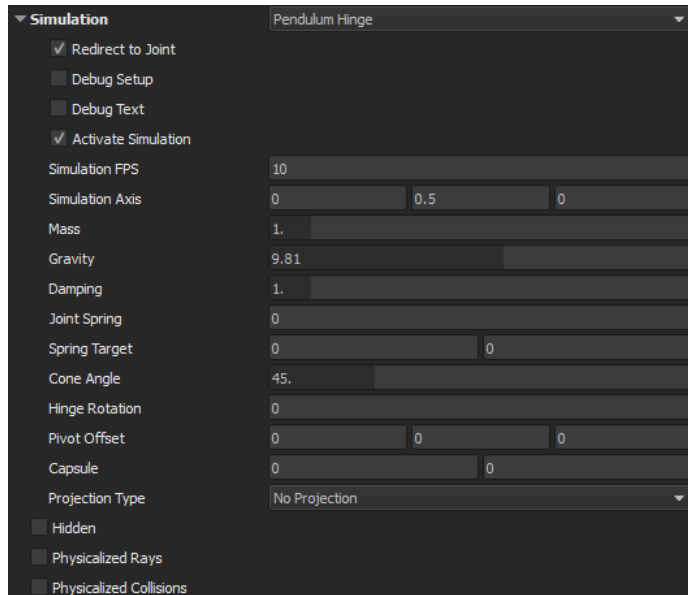
Pendulum Cone Simulation



Pendulum Half-Cone Simulation



Pendulum Hinge Simulation



Pivot Offset

This feature is identical for both spring and pendula simulations. Pivot Offset allows you to offset the location of the attached render object. Note that this is purely a visual feature with no impact on the simulation itself and only adds an offset to the attached object at the rendering stage. Adding or changing an offset value doesn't change the position of the socket; it only renders the attached geometry at another location that can be outside of the bounding volume.

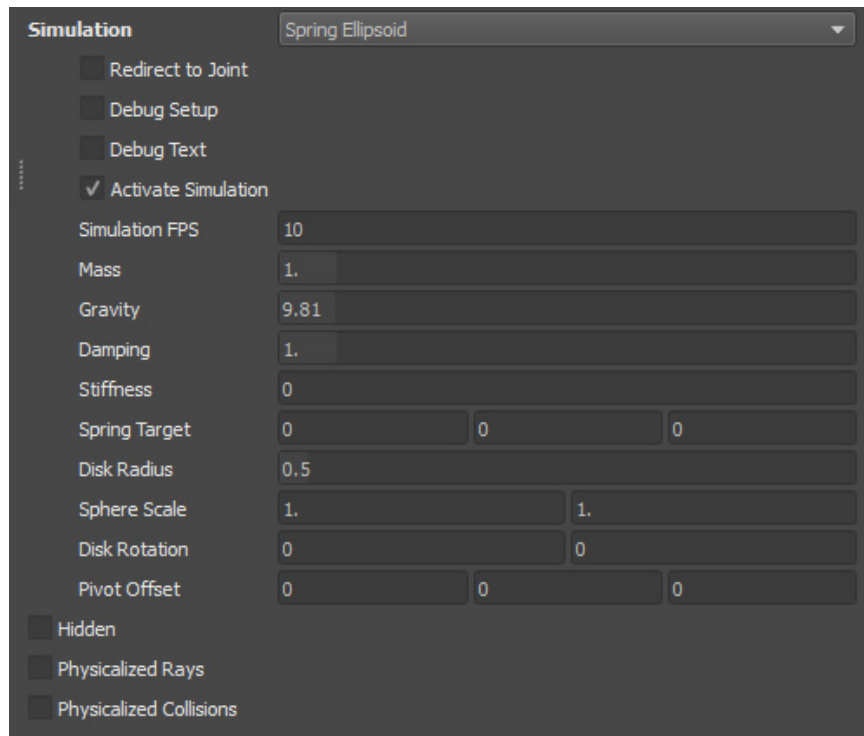
By default, it is the pivot of the model (offset = 0,0,0) and those three values are an x,y,z axes offset that translates the rendered geometry in the direction of the socket axes.

If **Redirect to Joint** is also enabled, then the pivot offset changes the location of the joint and all its children, as discussed next.

Redirect to Joint

If enabled, the relative motion of the simulated object is transferred to the joint that it is attached to, which means that the relative motion of the pendulum is added to the joint. So as long as the pivot offset is (0,0,0) then we only modify the orientation of the joint and this moves all vertices that are part of the mesh and weighted to this joint.

Spring Ellipsoid Simulation



Pivot Offset

This feature is identical for both spring and pendula simulations. Pivot Offset allows you to offset the location of the attached render object. Note that this is purely a visual feature with no impact on the simulation itself and only adds an offset to the attached object at the rendering stage. Adding or changing an offset value doesn't change the position of the socket, it only renders the attached geometry at another location that can be outside of the bounding volume.

By default, it is the pivot of the model (offset = 0,0,0) and those three values are an x,y,z axes offset that translates the rendered geometry in the direction of the socket axes.

If **Redirect to Joint** is also enabled, then the pivot offset changes the location of the joint and all its children, as discussed next.

Redirect to Joint

If enabled, the relative motion of the simulated object is transferred to the joint that it is attached to, which means that the distance between the spring particle and the joint is added together. For spring simulations, only the translation of the joint is changed, which moves all vertices that are part of the mesh and weighted to the joint.

Animating Characters

Skeleton-based animation is the most flexible animation technique used today, and includes playback and blending of animation data as well as IK-based poses. Procedural algorithms like CCD-IK, analytic IK, example-based IK, or physical simulations are all used to augment pre-authored animations. To provide realism when combining artificial and captured animations, a warping technique preserves the style and content of the base motion.

However, skeleton-based animation is not the ideal solution for animating muscles and tendons of the human body or face. Although it is possible to use skeleton-based animation for this, the number of joints involved is high and animation setup is difficult.

Generally, the combination of morph-based animation along with skeletal-based animation provides the greatest flexibility. The number of vertices that change in each morph target is very limited and the targets can be clearly defined. Morph targets are ideal for creating facial animations. Morph-based animation can even be used to generate entire animation sequences.

At the highest level, you can use Flow Graph, Lua scripts, or C++ code to request character animations. These methods invoke the Mannequin system, which in turn invokes the core Lumberyard animation system for animation clips, animation events, and procedural clips. Procedural clips can include IK, ragdoll, sounds, particle effects, and game logic.

Geppetto is a tool used to add character attachments, preview animations, and test blending features. It provides a visual interface to the underlying animation system.

You can add character `.cdf` and geometry `.cgf` assets in the Track View cinematic cutscene animations.

Topics

- [Types of Character Animations \(p. 225\)](#)
- [Character Animation Files \(p. 226\)](#)
- [Chrparams File Setup Using Geppetto \(p. 228\)](#)
- [Chrparams File Elements \(p. 231\)](#)
- [Character Skeletons \(p. 235\)](#)
- [Importing Character Animations \(p. 235\)](#)
- [Compressing Character Animations \(p. 236\)](#)
- [Working with Additive Animations \(p. 239\)](#)
- [Character Animation Layers \(p. 240\)](#)
- [Working with Blend Shapes \(Morphs\) \(p. 241\)](#)
- [Working with Blend Spaces \(Bspaces\) \(p. 242\)](#)
- [Animation Events \(p. 244\)](#)
- [Locomotion Locator Animation Best Practices \(p. 246\)](#)
- [Streaming Character Animations \(p. 246\)](#)

Types of Character Animations

You can produce three major types of animation in Lumberyard:

Cutscene Animations

Cutscenes are cinematic sequences in a game that involve no gameplay. Also known as linear or cinematic animation, cutscene animations are the easiest animation to create, as the animator controls every aspect. Camera angle, lighting, keyframes, and character pose are all fixed. You create cutscene animations with Track View Editor.

Scripted Flow Graph Animations

More complex than cutscene animations are scripted animations in which characters follow a predefined path. The quality is such that it appears to be interactive, but it is not. Characters cannot engage with, or respond to, the player.

You can create scripted animations using animation Flow Graph nodes and can also include AI nodes for more complicated animations. An example would be a character who changes his walking gait over uneven or hilly terrain, or to avoid a vehicle that is in the line of the walking path.

You can use Flow Graph to start and stop animations, trigger animations based on time, synchronize two animations, and coordinate multiple animations based on various parameters.

Interactive Animations

The most complex character animation to create are fully interactive, nonlinear animations where characters respond automatically to their environment, other characters, player inputs, AI behaviors, and other in-game variables. It is common to have a character perform multiple movements and tasks simultaneously, displaying different emotions, and respond differently to different events.

In such an environment, character movements and actions are unpredictable. A crucial feature of interactive animation involves the automatic synthesis of high quality character motions and good AI rules for behavior based on a variety of different game events, all while keeping performance high and asset count as low as possible.

Interactive animations fall into two categories: player controlled and AI controlled.

In player-controlled animations, the player determines the movement and all other actions of the character; the animation system takes the player input and translates it on the fly to skeleton movements using procedural and data-driven methods. For player control, high responsiveness is a key feature.

In AI-controlled animations, the AI system controls the movement and actions of the character. All motion and behaviors are dictated based on a series of rules and parameters that defines a character's actions in response to in-game events. These actions are not fully predictable as there are an almost unlimited number of different game permutation possibilities.

To help you achieve high quality interactive character animations, Lumberyard provides the following tools:

- **Geppetto** – Lower level system that manages short animation clips, poses, procedural and parameterized movements, transitions, and layers. For more information, see [Using Geppetto \(p. 204\)](#).
- **Mannequin Editor** – High-level system that manages animation variations, transitions, sequences, blends, layers, and procedural logic. For more information, see [Using Mannequin Editor \(p. 250\)](#).

Character Animation Files

To create character animation files, you start by animating character skeletons and hierarchies in a DCC tool. You then use your DCC tool to export these elements to the intermediate `.i_caf` file format. They are then compressed and optimized to the `.caf` before Lumberyard can use them.

Lumberyard's animation system uses the following files to create animations for your characters.

Character Asset File (*.chr)

The character used for animations is defined in a `.chr` file. For animation, the two important aspects of a character are the morph targets and the skeleton.

Character Definition File (*.cdf)

Characters are usually combinations of a primary model and several attachments. In particular, the head is often considered a skin attachment that is a separate model attached to the body. This composite model is defined in the `.cdf` file and contains a reference to the `.chr` file and its attachments.

Intermediate Character Animation File (.i_caf)

The intermediate character animation file contains the animated bone data for a specific character. This file can be used with multiple characters with similar bone structures. The file is created by a DCC tool and stores animation data in uncompressed format. It is usually used with a skinned mesh.

Animation Settings File (.animsettings)

The animation settings file contains per-animation compression settings. This is a sidecar file that is stored next to the .i_caf file and describes how it should be compiled by the Asset Pipeline. This file is created using Geppetto for importing animations.

Skeleton Alias File (SkeletonList.xml)

This file provides a table that maps skeleton aliases used in the .animsettings file to skeleton file names. This file contains skeleton structure information that is needed during animation compression.

Character Animation File (*.caf)

Assets, such as bones, are stored in .caf files. Because they are considered on demand assets, these files are streamed in and out as needed. This file is the compressed version of the intermediate .i_caf file and uses lossy compression. Character animation files are created by Lumberyard Editor during the asset build, and are loaded by the game at runtime.

Character Parameters File (*.chrparams)

Skeletal character parameters are defined in the XML .chrparams file. This file has the same name as the .chr character file to which it refers.

Animation Database (.dba)

A .dba file consists of multiple animations (character, player, AI, weapons) that are streamed in and out together. These files are typically smaller and take up less memory than individual animations (.caf files). Single .caf files are no longer needed unless they are on-demand assets.

If an animation is in a .dba file, it will not be available anymore as an individual .caf file. If the game tries to play one of these animations, the database containing that animation loads instead. As this can take a while, make sure the .dba is preloaded.

When two animations in the same .dba file have exactly the same animation for a joint, the data for that animation is stored once. This can provide significant memory savings.

The .dba files are created by the Resource Compiler after compressing the individual animations (.cafs), according to the DbTable.xml file. The .dba file must be defined in the .chrparams file.

Typical animations that get stored in the .dba include:

- Animations that need to be individually loaded and unloaded.
- Animations that need to be accessed once on demand, such as track view (cinematic) animations. These animations are preloaded a couple of seconds before starting.

Note

Aimposes, Lookspace, .bspace, and .comb files cannot be stored in a .dba database.

Animation Database Table (DbTable.xml)

The animation database table contains a list of .dba files, which the resource compiler uses to determine which .caf animations to put in which .dba files. Here is an example:

```
<DBAs>
  <DBA Path="Animations\human\male\hits_1p.dba">
    <Animation Path="Animations\human\male\hits\1p
\stand_tac_hit_generic_add_1p_01.caf"/>
    <Animation Path="Animations\human\male\hits\1p
\stand_tac_hit_knockDown_1p_01.caf"/>
  </DBA>
</DBAs>
```

```
<Animation Path="Animations\human\male\hits\lp
\stand_tac_idle_reactExplosion_3p_01.caf"/>
</DBA>
<DBA Path="Animations\human\male\locomotion.dba">
  <Animation Path="Animations\human\male\locomotion\kneel
\kneel_tac_AimPoses_idle_01.caf"/>
  <Animation Path="Animations\human\male\locomotion\kneel
\kneel_tac_death_01.caf"/>
  <Animation Path="Animations\human\male\locomotion\kneel
\kneel_tac_idle_01.caf"/>
  <Animation Path="Animations\human\male\locomotion\kneel
\kneel_tac_stepRotate_90_lft_01.caf"/>
</DBA>
</DBAs>
```

Animation Events Database (.animevents)

This database stores a list of assets with timed event markups. For example, it might store footstep sounds. You use the Geppetto to create this database, which gets mapped to the .chrparams file.

Blend Space (.bspace)

Blend spaces (Bspaces) define how multiple animation assets are blended together. Blend spaces are parameterized at runtime with movement parameters such as movement speed, movement direction, turning angle, or slope.

BlendSpace Combination (.comb)

This file combines multiple blend spaces into one, usually of a higher order, and represents a multidimensional blend space.

Group Files (*.grp)

Group files are exported animation sequences in XML format that are used for track view animation sequences. Data stored in a sequence includes everything from audio positions to skeletal animations and camera paths used.

Chrparams File Setup Using Geppetto

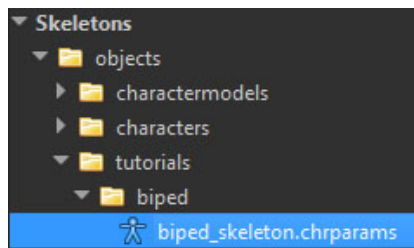
The .chrparams file is used to map animations to a specific character skeleton. With it, you can setup the mapping of Animations, Animation Events, and Database Animations using Geppetto. The sections below explain how to map these to the .chrparams file.

Note

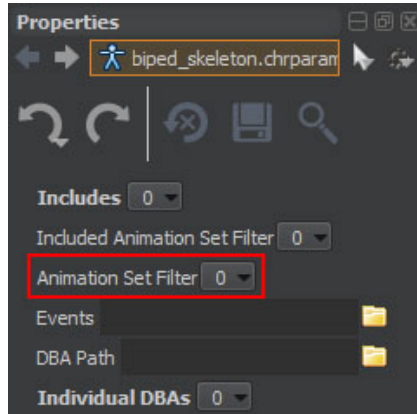
Currently, if you want to do anything else such as IK definitions, you will have to manually edit those items using a text editor.

To add an Animation path to a .chrparams file

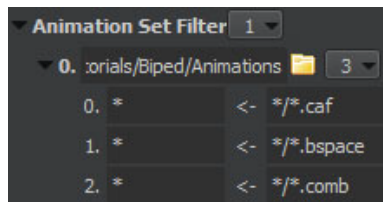
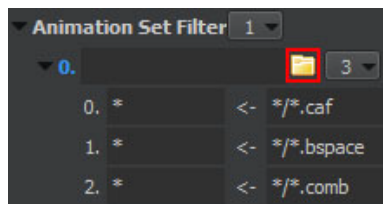
1. In Geppetto, in the **Assets** panel, expand **Skeletons**, navigate to the character's skeleton (*.chrparams), and select it.



2. In the **Properties** panel, find the **Animation Set Filter** with a # (will display a 0 if this is a new setup) next to it.



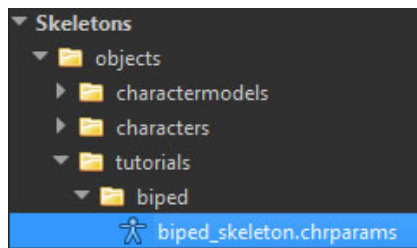
3. Click the # drop-down list and select **Add**.
4. Click the folder icon next to the empty property field and assign a directory where the animations for this character's skeleton will be located.



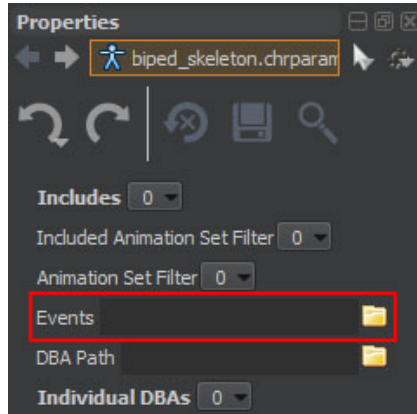
5. After assigning an animation directory, click **Save** in the **Properties** panel to save your changes to the .chrparams file.
6. Add additional animation directories as needed and modify the sub-fields if you need specific filters set for your animations.
7. When loading a .cdf file in Geppetto using the skeleton, look for the animations from the assigned directory listed under the **Animations** section of the **Assets** panel.

To add an AnimEvents path to a .chrparams file

1. In Geppetto, in the **Assets** panel, expand **Skeletons**, navigate to the character's skeleton (*.chrparams), and select it.



2. In the **Properties** panel, find the **Events** field.



3. Click the folder icon next to the empty property field and assign a directory where the `.animevents` file will be located for this character's skeleton.

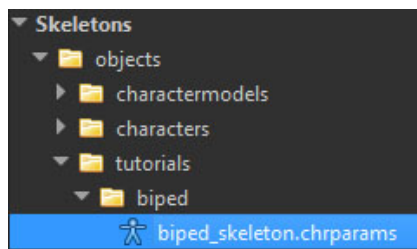
Note

Only one `.animevents` file can be assigned per `.chrparams` file.

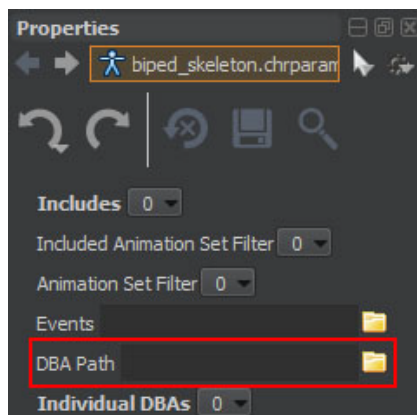
4. After assigning the `.animevents` file, click **Save** to save your changes to the `.chrparams` file.

To add a DBA path to a `.chrparams` file

1. In Geppetto, in the **Assets** panel, expand **Skeletons**, navigate to a character's skeleton (`*.chrparams`), and select it.



2. In the **Properties** panel, find the **DBA Path** field.



3. Click the folder icon next to the empty property field and assign a directory where the `.dba` files will be located.
4. After assigning the `.dba`, click **Save** in the **Properties** panel to save your changes to the `.chrparams` file.

Chrparams File Elements

All parameters for a character in Lumberyard is stored in various element sections of the `.chrparams.xml` file. You can use any text editor to edit this XML file.

Animations

The `.chrparams` file contains a single `<AnimationList>` element. This element lists all animation asset files that the character uses. See the following example.

```
<AnimationList>
  <Animation name="$AnimEventDatabase" path="animations\human\male
\events.animevents"/>
  <Animation name="$Include" path="animations\human\male\male.chrparams"/>
  <Animation name="$TracksDatabase" path="animations\human\male\hits.dba"/>
  <Animation name="$TracksDatabase" path="animations\human\male
\locomotion.dba" flags="persistent"/>
  <Animation name="#Filepath" path="animations\human\male"/>
  <Animation name="*" path="*\*.caf"/> <!-- includes all cafs in #Filepath
and subfolders -->
  <Animation name="_*" path="*\*.bspace"/> <!-- includes all bspace in
#Filepath and subfolders and prepend with _ -->
  <Animation name="_*" path="*\*.comb"/> <!-- includes all comb in
#Filepath and subfolders and prepend with _ -->
</AnimationList>
```

Bone LODs

The `.chrparams` file contains a single `<Lod>` element section, which lists all joints that the character uses. See the following example:

```
<Lod>
  <JointList level="0">
    <Joint name="Bip01 Pelvis"/>
    <Joint name="Bip01 Spine"/>
  </JointList>
  <JointList level="1">
    <Joint name="weapon_bone"/>
    <Joint name="joint_12"/>
  </JointList>
</Lod>
```

IK Definition

The `.chrparams` file contains a single `<IK_Definition>` element section, which defines the joint that are used for the different IK methods, such as AimIK, LookIK, LimbIK and Animation-Driven IK.

Limb

The `.chrparams` file contains a single `<LimbIK_Definition>` element section within `<IK_Definition>`. This section lists all the joints that are used for Limb IK, along with the root bone, end effector, and solver. See the following example section:

```
<IK_Definition>
  <LimbIK_Definition>
```

```
        <IK EndEffector="Bip01 R Hand" Handle="RgtArm01" Root="Bip01 R
UpperArm" Solver="2BIK" />
        <IK EndEffector="Bip01 L Hand" Handle="LftArm01" Root="Bip01 L
UpperArm" Solver="2BIK" />
        <IK EndEffector="Bip01 R Foot" Handle="RgtLeg01" Root="Bip01 R Thigh"
Solver="2BIK" />
        <IK EndEffector="Bip01 L Foot" Handle="LftLeg01" Root="Bip01 L Thigh"
Solver="2BIK" />
    </LimbIK_Definition>
</IK_Definition>
```

Anim Driven

The .chrparams file contains a single <Animation_Driven_IK_Targets> element section within <IK_Definition>. This section lists all joints used for Animation-driven IK, along with target bones, blend bones, and weights. See the following example section:

```
<IK_Definition>
  <Animation_Driven_IK_Targets>
    <ADIKTarget Handle="LftArm01" Target="Bip01 Chin_IKTarget" Weight="Bip01
Chin_IKBlend" />
  </Animation_Driven_IK_Targets>
</IK_Definition>
```

Foot Lock

The .chrparams file contains a single <FeetLock_Definition> element section within <IK_Definition>. This block lists all joints used for foot step alignment and lock effects. See the following example section:

```
<IK_Definition>
  <FeetLock_Definition>
    <RIKHandle Handle="RgtLeg01" />
    <LIKHandle Handle="LftLeg01" />
  </FeetLock_Definition>
</IK_Definition>
```

Recoil

The .chrparams file contains a single <Recoil_Definition> element section within <IK_Definition>. This block lists all weapon joints and impact joints used for recoil effects, along with weights and delay times. See the following example section:

```
<IK_Definition>
  <Recoil_Definition>
    <RIKHandle Handle="RgtArm01" />
    <LIKHandle Handle="LftArm01" />
    <RWeaponJoint JointName="weapon_bone" />
    <ImpactJoints>
      <ImpactJoint Arm="3" Delay="0.3" Weight="0.2" JointName="Bip01
Pelvis" />
      <ImpactJoint Arm="3" Delay="0.2" Weight="0.3" JointName="Bip01 Spine"
/>
      <ImpactJoint Arm="3" Delay="0.1" Weight="0.5" JointName="Bip01
Spine1" />
      <ImpactJoint Arm="3" Delay="0.0" Weight="1.0" JointName="Bip01
Spine2" />
    </ImpactJoints>
  </Recoil_Definition>
</IK_Definition>
```

```

    <ImpactJoint Arm="3" Delay="0.0" Weight="1.0" JointName="Bip01
    Spine3" />
    <ImpactJoint Arm="3" Delay="0.0" Weight="1.0" JointName="Bip01
    Neck" />

    <ImpactJoint Arm="3" Delay="0.10" Weight="0.10" JointName="Bip01 R
    Thigh" />
    <ImpactJoint Arm="3" Delay="0.05" Weight="0.05" JointName="Bip01 R
    Calf" />
    <ImpactJoint Arm="3" Delay="0.10" Weight="0.10" JointName="Bip01 L
    Thigh" />
    <ImpactJoint Arm="3" Delay="0.05" Weight="0.05" JointName="Bip01 L
    Calf" />

    <ImpactJoint Arm="2" Delay="0.0" Weight="1.0" JointName="Bip01 R
    Clavicle" />
    <ImpactJoint Arm="2" Delay="0.00" Weight="0.50" JointName="Bip01 R
    UpperArm" />

    <ImpactJoint Arm="1" Delay="0.01" Weight="0.7" JointName="Bip01 L
    Clavicle" />
    <ImpactJoint Arm="1" Delay="0.00" Weight="0.50" JointName="Bip01 L
    UpperArm" />
  </ImpactJoints>
</Recoil_Definition>
<IK_Definition>

```

Look

The `.chrparams` file contains a single `<LookIK_Definition>` element section within `<IK_Definition>`. This block lists all joints used for Look IK, along with eye attachments, limits, and rotations. See the following example section:

```

<IK_Definition>
  <LookIK_Definition>
    <LEyeAttachment Name="eye_left"/>
    <REyeAttachment Name="eye_right"/>

    <DirectionalBlends>
      <Joint AnimToken="LookPoses" ParameterJoint="Bip01 Look"
      StartJoint="Bip01 Look" ReferenceJoint="Bip01 Pelvis"/>
    </DirectionalBlends>

    <RotationList>
      <Rotation Additive="1" Primary="1" JointName="Bip01 Pelvis"/>
      <Rotation Additive="1" Primary="1" JointName="Bip01 Spine"/>
      <Rotation Additive="1" Primary="1" JointName="Bip01 Spine1"/>
      <Rotation Additive="1" Primary="1" JointName="Bip01 Spine2" />
      <Rotation Additive="1" Primary="1" JointName="Bip01 Spine3" />
      <Rotation Additive="0" Primary="1" JointName="Bip01 Neck" />
      <Rotation Additive="0" Primary="1" JointName="Bip01 Head" />
      <Rotation Additive="0" Primary="1" JointName="Bip01 Look" />

      <Rotation Additive="0" Primary="0" JointName="def_r_brow_A" />

      <Rotation Additive="0" Primary="0" JointName="def_r_brow_B" />
      <Rotation Additive="0" Primary="0" JointName="def_r_brow_C" />
      <Rotation Additive="0" Primary="0" JointName="def_r_upperEyeLid" />
    </RotationList>
  </LookIK_Definition>
</IK_Definition>

```

```

    <Rotation Additive="0" Primary="0" JointName="def_r_lowerEyeLid" />

    <Rotation Additive="0" Primary="0" JointName="def_l_brow_A" />

    <Rotation Additive="0" Primary="0" JointName="def_l_brow_B" />
    <Rotation Additive="0" Primary="0" JointName="def_l_brow_C" />
    <Rotation Additive="0" Primary="0" JointName="def_l_upperEyeLid" />
    <Rotation Additive="0" Primary="0" JointName="def_l_lowerEyeLid" />
  </RotationList>

  <PositionList>
    <Position Additive="1" JointName="Bip01 Pelvis" />

    <Position Additive="0" Primary="0" JointName="def_r_brow_A" />

    <Position Additive="0" Primary="0" JointName="def_r_brow_B" />
    <Position Additive="0" Primary="0" JointName="def_r_brow_C" />
    <Position Additive="0" Primary="0" JointName="def_r_upperEyeLid" />
    <Position Additive="0" Primary="0" JointName="def_r_lowerEyeLid" />

    <Position Additive="0" Primary="0" JointName="def_l_brow_A" />

    <Position Additive="0" Primary="0" JointName="def_l_brow_B" />
    <Position Additive="0" Primary="0" JointName="def_l_brow_C" />
    <Position Additive="0" Primary="0" JointName="def_l_upperEyeLid" />
    <Position Additive="0" Primary="0" JointName="def_l_lowerEyeLid" />
  </PositionList>
</LookIK_Definition>
<IK_Definition>

```

Aim

The .chrparams file contains a single <AimIK_Definition> element section within <IK_Definition>. This block lists all joints required for Aim IK, along with positions, rotations, and procedural adjustment joints. See the following example section:

```

<IK_Definition>
  <AimIK_Definition>
    <DirectionalBlends>
      <Joint AnimToken="AimPoses" ParameterJoint="weapon_bone"
      StartJoint="Bip01 R UpperArm" ReferenceJoint="Bip01" />
    </DirectionalBlends>

    <RotationList>
      <Rotation JointName="Bip01 Spine"           Primary="1" Additive="0" />
      <Rotation JointName="Bip01 Spine1"          Primary="1" Additive="0" />
      <Rotation JointName="Bip01 Spine2"          Primary="1" Additive="0" />
      <Rotation JointName="Bip01 Spine3"          Primary="1" Additive="0" />
      <Rotation JointName="Bip01 Neck"            Primary="1" Additive="0" />
      <Rotation JointName="Bip01 Head"            Primary="0" Additive="0" />

      <Rotation JointName="Bip01 R Clavicle"      Primary="1" Additive="0" />
      <Rotation JointName="Bip01 R UpperArm"      Primary="1" Additive="0" />
      <Rotation JointName="Bip01 R ForeArm"       Primary="1" Additive="0" />
      <Rotation JointName="Bip01 R Hand"          Primary="1" Additive="0" />
      <Rotation JointName="weapon_bone"           Primary="1" Additive="0" />

      <Rotation JointName="Bip01 L Clavicle"      Primary="0" Additive="0" />
    </RotationList>
  </AimIK_Definition>
</IK_Definition>

```

```
<Rotation JointName="Bip01 L UpperArm" Primary="0" Additive="0" />
<Rotation JointName="Bip01 L ForeArm" Primary="0" Additive="0" />
<Rotation JointName="Bip01 L Hand" Primary="0" Additive="0" />
</RotationList>

<PositionList>
  <Position JointName="Bip01 R Clavicle"/>
  <Position JointName="weapon_bone"/>
  <Position JointName="Bip01 L Clavicle"/>
</PositionList>

<ProcAdjustments>
  <Spine JointName="Bip01 Pelvis"/>
  <Spine JointName="Bip01 Spine"/>
  <Spine JointName="Bip01 Spine1"/>
  <Spine JointName="Bip01 Spine2"/>
  <Spine JointName="Bip01 Spine3"/>
</ProcAdjustments>
</AimIK_Definition>
<IK_Definition>
```

Character Skeletons

Use the following procedure to add a character skeleton to the `SkeletonList.xml` file.

To add a character skeleton to the skeleton list

1. In the **Assets** panel under **Compression (Animations)**, click **Skeleton List**.
2. In the **Properties** panel under **Aliases**, make sure the skeleton `.chr` file is in the list. If not, do the following:
 - a. Click the number button next to **Aliases** and click **Add**.
 - b. Click the folder icon next to the new entry, then select a suitable skeleton.
 - c. Name the added skeleton alias - this name is used to refer to the skeleton.

Skeleton Aliases

This file provides a table that maps skeleton aliases used in the `.animsettings` file to skeleton file names. This file contains skeleton structure information that is needed during animation compression.

Importing Character Animations

You can easily import character animations using Geppetto. The character's skeleton needs to be part of the skeleton list before you can start importing animations. Have your character loaded in Geppetto before following the steps below.

To import character animations

1. In Geppetto, in the **Assets** panel, under **Animations**, select the animation to import. All unimported animations become unavailable.
2. In the **Properties** panel, click **Import**.

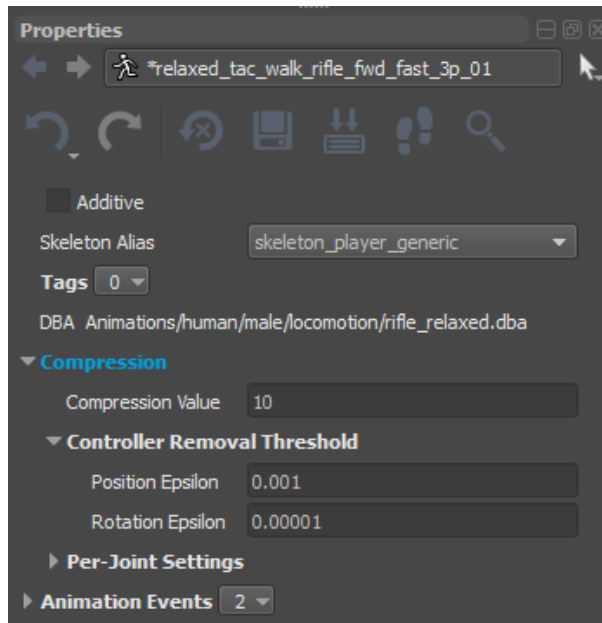
Note

You may need to select **Skeleton Alias** in the menu in the event that the loaded character could not be matched to the skeleton alias.

Compressing Character Animations

For best results, try to employ character assets that use the least amount of memory but are animated at the highest possible quality. An uncompressed animation contains a key for every frame in the animation and for each joint that has been exported. The goal is to reduce the amount of joints and keys to minimize the size. There are separate channels for rotation keys and position keys per joint.

For maximum compression, remove from the animation any joints that don't contribute much to the animation. To know whether a joint contributes to an animation, use the Resource Compiler, which determines how much the joint moves during the animation and compares it to the provided epsilon values. If the joint moves less than what the epsilon specifies, the keys will be removed for the joint. Use higher epsilon values to remove more joints. Use **Position Epsilon** for the position channel and **Rotation Epsilon** for the rotation channel.



Removing Joints Automatically

The two epsilon values are global values for the entire animation. Additive animations have smaller movements, so small values are used for the epsilon values.

Either all the keys are retained for a channel (position and rotation), or they are deleted.

To remove joints automatically

1. In Geppetto, in the **Properties** panel, expand **Compression**, **Controller Removal Threshold**.
2. Change the values for **Position Epsilon** and **Rotation Epsilon** as needed.

Removing Joints Manually

By default, each joint uses two epsilon values to determine whether the joint is removed.

To remove individual joints manually

1. In Geppetto, in the **Properties** panel, expand **Compression**, **Per-Joint Settings** and then select the check box next to the joint to delete it. Both the **Position** and **Rotation** channels are removed for the selected joint.

2. Enter a value in the box next to the joint to change the multiplier that is applied to the compression value. By default this value is 1.

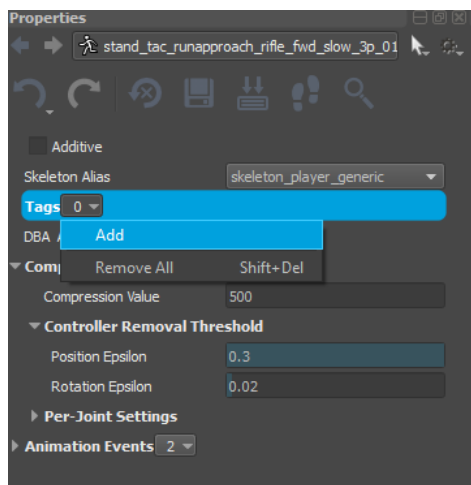
Animation Tags

Each animation can have a list of tags associated with it. You can use tags to accomplish the following:

- Select animations that have to go into a specific DBA table by means of an animation filter.
- Apply compression to a group of animation files by means of compression presets

Tags are located in the **Properties** panel when you select an animation in the **Assets** panel.

To add a new tag, click the number besides **Tags** and click **Add**.

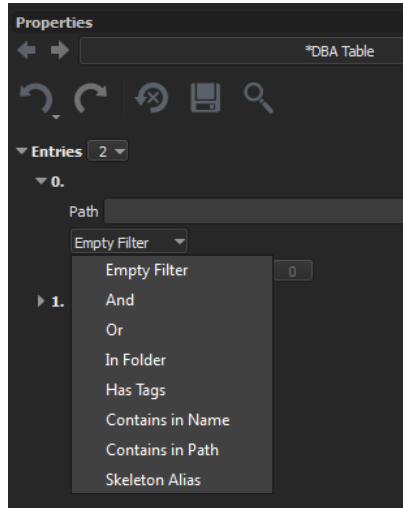


Animation Filters

Use an animation filter to choose a set of animation files for specific DBA or compression preset. An animation filter is defined as a tree of condition nodes.

DBA files are bundles of animations that can be streamed in and out as one. They are typically smaller and take up less memory than individual animations. DBAs are created using the same filters as compression presets. You can define a combination of criteria such as location, name, or tags to select animations for a specific DBA.

DBA descriptions are saved to `Animations/DBATable.json`. The resource compiler uses this `.json` file at build time to create the actual DBA files. The DBA Table can be found under **Compression (Animations/)** in the **Assets** panel.



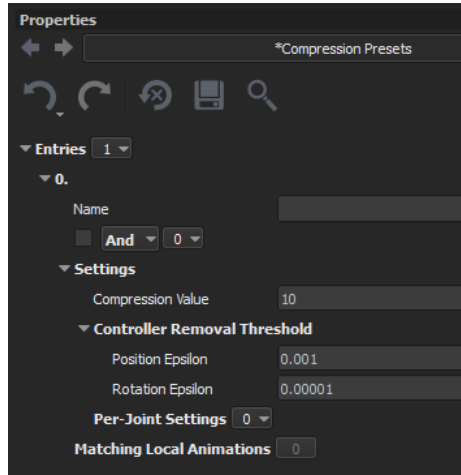
DBA Table Options

Conditions	Description
Empty filter	No conditions applied
And	Succeeds when all of the child conditions succeed.
Or	Succeeds when at least one of the child conditions succeeds.
In folder	Checks whether animation is located within a specific directory.
Has tags	Checks whether animation has all of the listed tags. Tags are stored in ANIMSETTINGS and can be set in Animation Properties.
Contains in name	Checks for a substring within an animation name.
Contains in path	Checks whether animation is located within a specific file path.
Skeleton alias	Checks whether animation uses a specific skeleton alias. Skeleton aliases are defined in the skeleton table.

Compression Presets

You can use compression presets to apply the same set of compression rules to multiple animations at once. Presets are listed under **Compression (Animations/)** in the **Assets** panel.

Each compression preset entry defines a filter that can match animations according to a certain filter. Filter criteria can include a folder, file name, or tags. You can use logical operations to combine these criteria into a complex condition like "in folder and doesn't contain specific tag but has substring in name." When multiple presets match the same animation, only the first one is used. You can preview which compression setting entry was applied to animation in the **Properties** panel by selecting a specific animation in the **Assets** panel.



Working with Additive Animations

Additive animations are animations that can be added as layers on top of a base animation. The additive animation is usually a partial-body animation, so it can be applied to a base full-body animation without interfering with joint controllers and other important parts of the base animation. With additive animations, you can reuse the same full body-animations and add lots of variation to it.

An additive animation preserve the underlying animation and style and as such is great for adding poses and animations to the upper body. Since the underlying animations are not overwritten, this can reduce the overall asset count greatly, add a lot of variation to the animations, and reduce the monotonous look.

For example, you can use additive animations for breathing, looking around, flinching, and posture changes. To prevent foot sliding, additive animations cannot modify bones below the character's hips.

You start an additive animation like a regular animation. Lumberyard automatically recognizes it after it has been processed by the resource compiler.

Creating Additive Animations

To create an additive animation, you start with a typical base pose and then animate only those bones and other parts that you want to include in the additive animation. The first frame (frame 0) is the base pose and the rest of the animation becomes the delta. Bones that do not differ from the base pose are not used. The resource compiler subtracts the first frame during export; it is not part of the final animation.

Importing Additive Animations

To import an additive animation, select **Additive** check box for the `.i_caf` animation in the **Properties** panel in Geppetto.

Testing Additive Animations

You can test an additive animation in just a few steps

To test an additive animation

1. In Lumberyard Editor, click **View, Open View Pane, Geppetto**.
2. Click **File, Open** and load the applicable character `.cdf` file.
3. Make sure a full body animation is playing on the first animation layer under **Animation Layers** in the **Scene Parameters** panel.

4. Add a new animation layer in the **Scene Parameters** panel by clicking on the number next to **Animation Layers**, and then click **Add**.
5. Select an additive animation from the **Animation** list in the **Assets** panel to add it to the new animation layer. Adjust the weights of the additive animation as needed by changing the value (0 to 1) next to the new animation layer.

Character Animation Layers

By layering animations, you can apply an animation to only a few select bones, rather than to the whole skeleton. Lumberyard has a maximum of 16 virtual layers available for use. Layer 0 is the primary base layer and contains the base full-body animations, joints, and blend spaces. Higher levels contain additive partial-body animations and *overwrite animations*, meaning that animations in higher layers overwrite animations in lower layers. As long as they don't share the same joints, these animations won't interfere. You can combine all layers into a single layer, which applies them to a character simultaneously.

If an animation played in layer 0 has no controller for a specific bone, the default transformation from the character rig is used instead. Layer 0 is the only layer that supports the root bone and the locomotion locator.

Each layer can play and blend animations and has its own transition queue that handles the blending in and out of animations in the layer. The default behavior for animations in a layer is as follows:

1. Play animation once; then blend it out (weights decrease to 0).
2. Remove animation from the queue when the weight reaches 0.
3. Blend in the next animation (weight increases from 0 to 1).

In Geppetto, only one animation layer is active by default for previewing animations. Any time you select an animation, it plays on the default base layer. You can find the **Animation Layers** listed in the **Scene Parameters** panel of Geppetto.

To add animation layers using Geppetto

- 1.
2. Click on the drop down menu next to **Animation Layers** and click on **Add**. The newly added layer becomes the active layer for you to select a new animation from the **Animations** list in the **Asset** panel to assign to the new layer.
3. Repeat this step for as many animation layers as you need. At any point, you can click on a specific animation layer to make it active in order to change the animation playing on that layer.
4. Adjust the blend weight (0 to 1) for each layer.
5. Enable and disable layers using the checkboxes next to each layer.

You can enable on-screen debug information to see which animations are queued and playing, as well as information about the applied pose modifiers and IK.

Accessing Animation Layers using Code

To access animation layers via code, use the `ISkeletonAnim` object. In the example below, a looping animation starts on layer 2 and is fully blended in 0.5 seconds.

```
ISkeletonAnim& skeletonAnim = ...;  
CryCharAnimationParams params;  
params.m_nLayerID = 2;
```

```
params.m_nFlags |= CA_LOOP_ANIMATION;  
params.m_fTransTime = 0.5f;  
  
// Starting the animation by id. Alternatively use StartAnimation to start an  
// animation by name.  
skeletonAnim.StartAnimationById(animationId, params);
```

To smoothly blend out animations in a layer, use the `StopAnimationInLayer` function:

```
ISkeletonAnim& skeletonAnim = ...; // Blend out all animations in layer 2  
in  
    0.5 seconds: skeletonAnim.StopAnimationInLayer(2, 0.5f);
```

To force the transition queue in a specific layer to immediately clear all animations:

```
ISkeletonAnim& skeletonAnim = ...;  
skeletonAnim.ClearFIFOLayer(layerId);
```

To force transition queues in all layers to clear immediately, use `StopAnimationsAllLayers`, as follows:

```
ISkeletonAnim& skeletonAnim = ...;  
skeletonAnim.StopAnimationsAllLayers();
```

Working with Blend Shapes (Morphs)

Animated blend shapes, also known as morph target animation, is a method that stores a deformed version of a mesh as a series of vertex positions. In each keyframe of an animation, the vertices are then interpolated between these stored positions.

Blendshape animations are created by adding bones to the base skeleton. This involves explicit name matching between bone names and the blend shape controls.

For blend shape export requirements, see [Exporting Blendshapes \(p. 185\)](#).

Blend Shape Authoring Requirements in Maya

As blend shapes only work for `.skin` attachments, use Maya to create a base `.chr` like a cube or triangle that is skinned to the export skeleton.

See the following requirements and guidelines when creating a blend shape scene in Maya:

- All blend shape meshes must exist be in the same world space location as the skinned base mesh. Move your blend shape meshes on top of the skinned base mesh.
- **Smooth bind** at least one joint to the blend shape base mesh.
- Make sure the `root` joint of the skeleton hierarchy has no (zero) rotations.
- Create an empty `SceneRoot` group node and a `root` joint as the top-level node of the deforming skeleton. Do not skin the root joint into your character mesh.
- Set the `SceneRoot` group node and the `root` joint both looking forward with their Z-axes aligned to the world Y-axis and their Y-axes aligned to the world Z-axis.
- For each blend shape mesh, create a joint in the origin and name it `blend_shape_mesh_name_blendWeightVertex`
- The `_blendWeightVertex` joints should be parented under the root joint for the skeleton hierarchy.

- Manually create the `blendWeightVertex` joints and connections. Connect and map the `weight` output range (0 to 1) of the blend shape node to (0 to 100) to the `translateX` attribute of these helper joints.
- Nonrigid deformations require real-time tangent updates to get correct shading. Because such tangent updates are expensive, in order to minimize CPU cost, use vertex colors to transfer a blue (0, 0, 255) painted mask in your DCC tool to mark the most important facial parts.
- Tangent updates only work with 8-weight CPU skinning. To implement that, open the `.cdf` file and add `flags=8` on the line that lists the applicable skin attachment. This skinning makes the morphs expensive to use, so use it sparingly.

Blend Shape Setup in Lumberyard

Use the following procedure when setting up a blend shape in Lumberyard using Geppetto:

To set up a blend shape in Geppetto

1. Create a `SkeletonList.xml` file and place it in the `\Animations` directory and add the following skeleton element block to the file:

```
<SkeletonList>
  <Skeleton name="base_skel" file="exported_character_path_filename.chr" />
</SkeletonList>
```

2. Add a `.skin` attachment to the skeleton `.chr` file.
3. By default, Geppetto will add a **Joint Attachment**. Change this to a **Skin Attachment** and browse for the `.chr` file you exported earlier.
4. Enable **Software Skinning** for the blend shape to work.
5. Create a `.chrparams` file.
6. Add an **Animation Set Filter** and point it at the directory of the exported animation file.
7. Browse the directory containing the exported animation file and select the `default` animation.
8. Add a new `.animsettings` file and save it.
9. Browse for the saved `.cdf` file. Select and double-click the `default` animation.

Working with Blend Spaces (Bspaces)

Lumberyard supports blend spaces, also known as locomotion groups or LMGs, which are related motion parameters that you use to create motion clips. Specifically, an asset's kinematic, physical, and other high-level motion-related parameters are mapped onto corresponding features that are stored in the animation clips. By storing such motion as parameters, you can create controllable interactive animations.

With blend spaces, animation blending is treated as geometry. The structure of a blend space is similar to a character mesh with a vertex buffer and index buffer. Each animation clip represents a point on a coordinate system. Specifically, each animation is associated with a 1D, 2D, or 3D location in a blend space. You can play blend spaces on any layer, and they can contain additive or partial body animation.

Blend spaces (`.bspace` file format) in Lumberyard are XML-based file maps of animation blends that the Mannequin system uses. A `.comb` file represents a multidimensional blend space. Geppetto supports hot-loading of these XML files. This means you can change the XML file with a text editor, and Lumberyard updates it automatically and renders the result. This makes it ideal for prototyping and experimentation. Almost all parameters are identical for 1D, 2D, and 3D blend spaces.

Lumberyard supports blend space control of the following parameters:

- Move Speed
- Turn Speed
- Travel Angle
- Slope
- Turn Angle
- Travel Distance
- Blend Weight

Displaying Blend Spaces

The best way to get a feeling how blend spaces work internally, is to start a simple 2D-BSpaces, visualize it in Geppetto and a play around with the different debug options.

To display blend spaces

1. Open Geppetto and load a character that has a blend space file.
2. Click **View, Enable Blend Space Preview**. This displays the **Blend Space Preview** window on the right side of the **Geppetto** window between the **Scene Parameters** and **Properties** panels.
3. Detach the **Blend Space Preview** window from the **Geppetto** window by clicking the **Toggle Floating** button. Once detached, adjust the size of the **Blend Space Preview** window by grabbing it's corners.
4. Under the **Assets Panel**, under **Animations**, select the blend space file. The character displays in the preview window at each point on the grid that represents the blend space. The character in the **Geppetto** window is also animated based on the blend space controls.
5. Use the same viewport controls to navigate within the **Blend Space Preview** window as you would in the **Geppetto** window.
6. To adjust what part of the blend space is being displayed in **Geppetto** window, go to the **Scene Parameters** panel and expand the blend space animation layer to use the sliders to change the blend space's dimensions, such as travel speed and angle.
7. Adjust the blend space dimensions, examples, and annotations listed under the **Properties** panel as needed.

1D Blend Spaces

For 1D blend spaces, you can control a single character parameter X, such as movement speed.

In 1D blend spaces all points are on line segments. It is important that p0 points to the lower parameter and p1 points to the higher parameter. If the order is reversed or both parameters are identical, an error results.

Make sure that the line has no uncovered gaps and no overlapping line segments. At runtime, Lumberyard checks whether the input parameter is inside a line segment and then interpolates the value between the two animations.

2D Blend Spaces

2D blend spaces involve changing two parameters, X and Y, independently. This means when one parameter is changed, the other parameter stays constant and vice versa. An example of a 2D blend space is a character that moves at different speeds while also turning while moving. When the speed is changed, the turn radius (body angle) stays the same; and when the turn radius is changed, the speed is not affected.

In 2D blend spaces all points are on planar triangles and quads. Looking down onto the blend space, annotations occur counterclockwise for triangles and quads.

Make sure that the plane has no overlapping triangles and quads and no gaps or holes that are not covered with a face. At runtime, Lumberyard checks whether the input parameters fall inside a plane, and then interpolates the values between the three animations (for triangles) or four animations (for quads).

3D Blend Spaces

For 3D blend spaces, three separate parameters X, Y, and Z are changed independently. For example, character speed, turn radius, and travel angle can be changed.

In 3D blend spaces all points are inside of volume tetrahedrons, pyramids, and prisms. All have a ground plane (3 or 4 points) and a tip (1 or 2 points). If the tip points up, the vertices on the ground plane must be annotated counterclockwise. If the tip points down, the vertices are annotated clockwise.

Make sure that the space has no overlapping volumes and no holes that are not covered with a volume. At runtime, Lumberyard checks whether the input parameters are inside of one those volumes and then interpolates the values between those animations.

3D blend spaces are more difficult to debug, even with a very structured design. Fortunately, many higher dimensional blend spaces are a combination of simple lower dimensional blend spaces. This relationship makes it possible to combine two 2D blend spaces into a 3D space and two 3D blend spaces into a 4D blend space.

Number of Assets for Movement

Four assets are the minimum, but eight are the recommended minimum for realistic 360-degree movement. Diagonal blends usually don't look as good as forward and sideways motions. Specifically, diagonal blends can create foot-sliding, foot-crossing, and foot dipping through the ground if you only use four.

Another issue is hip rotation. Usually the hips point to the right when sidling right and to the left when sidling left. However, doing quick left to right side steps looks like Samba dancing. For best results, keep hip orientation static in each blend space, create a new blend space for each hip rotation, and play an explicit transition to adjust the gait. In this situation, 16 assets may be needed.

Debug Information

The following information is provided in the **Blend Space Preview** window:

- All animation files available the blend space, as a mini version of the character model. The size of the model can be controlled by the slider near the top right of the preview window.
- Each model has either a white or red spinning cube at its root joint.
- Each cube has an index. This is the order of the animation clips how they appear in `.bpspace XML` file, including all the pseudo examples.
- There is also a green wireframe quat. This shows which assets are currently considered in the blend. In a 2D blend space there are either triangles (blends between 3 assets) or quats (blends between 4 assets)
- Inside of a triangle or quat there is always a red flashing cursor. You can control it with the blend-space sliders and see at any time which assets contribute to the final blend.
- The current dimension values are displayed in the window, which corresponds with the current slider values set by the **Scene Parameters** animation layer.

Animation Events

Using Geppetto, you can add character animation events by double-clicking in the **Playback** timeline window. If you can right-click in the timeline, you can jump to previous and next events. Each animation can have multiple events specified.

If you need to create a large number of animation events, click **View** and select **Animation Event Presets**. This creates a new **Animation Event Presets** panel above the **Properties** panel.

This provided you a set of quickly-accessible animation events, which you can add to the playback timeline with a double-click. Keys with events corresponding to the presets are colored the same in the timeline.

Animation events are also accessible from the **Properties** pane for an animation. These are stored in an `.animevents` file, which is referenced from the `.chrparams` file, which contains lists of animation events per animation.

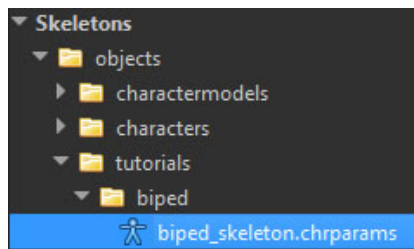
You will need to create an `.animevents` file per character skeleton unless the character shares skeletons and animations.

Creating the `.animevents` file

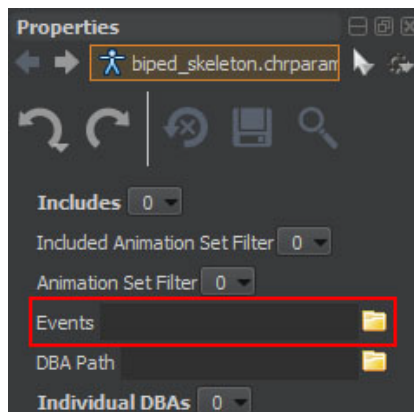
1. Create a new `.xml` file using a text editor.
2. In the new file, add the following tags: `<anim_event_list></anim_event_list>`.
3. Name the file using the `.animevents` extension and save it to the same directory as the animations it will apply to.

Updating the `.chrparams` file

1. In Geppetto, in the **Assets** panel, expand **Skeletons**, navigate to the character's skeleton (`*.chrparams`), and select it.



2. In the **Properties** panel, find the **Events** field.



3. Click the folder icon next to the empty property field and assign a directory where the `.animevents` file will be located for this character's skeleton.

Note

Only one `.animevents` file can be assigned per `.chrparams` file.

4. After assigning the `.animevents` file, click **Save** to save your changes to the `.chrparams` file.

Locomotion Locator Animation Best Practices

The locomotion locator, or `Locator_Locomotion` bone, is a node that is required for nonlinear or nonuniform character motions, such as a start or stop transition that has peaks and troughs in acceleration. For best results, consider doing the following:

- This bone must have the same orientation as the root joint and the `SceneRoot` node, which is the positive Y-axis in the local coordinate system. Otherwise animations are rotated to match the orientation of the locomotion locator bone. This only affects the animation and not the skeletal orientation.
- Just as the first and last keyframe of your animation cycle should match, the locomotion locator position relative to the character on the first keyframe should match the position relative to the character on the last keyframe. For complicated character animations like turns, you must animate this locator needs.
- The orientation of the locator in an idle-to-move transition should remain looking forward until keyframe 10.
- Make sure that orientation changes (left, right, left reverse, or right reverse) occur in the following 6 frames so the new orientation is complete at keyframe 16.
- When changing the orientation 180 degrees for reverse transitions, make sure you rotate the locator 0.1 degrees back to its original orientation to avoid flipping the character.
- For swimming transitions or vehicle transitions, the locator can be a straight blend between the ground position of 0,0,z and end at the `Bip01` location and forward-looking direction (positive Y-axis) of the character.
- For animation loops, set keys for the start and end of the animation only if you need to add a locator to them. They are technically not needed but can be useful for batch processing.

Streaming Character Animations

Animation can be a very memory-intensive resource. Limited memory budgets, a high number of animated joints and high animation quality requirements makes it undesirable to have all animations loaded in memory all the time. Lumberyard alleviates this issue by streaming asset files in and out as needed.

Animation data is divided into header data and controller data. Given the extreme size difference between controller and header data, only controller data is streamed in and out. The header data for all animations is kept in memory at all times.

Animation Header Data

The header contains generic information for an animation such as filename, duration, and flags. Header data is stored in `.CAF` files and in the `animations.img` file.

CAF files contain header information for a single animation, while the `Animations.img` file contains header information for all animations in the build. The `Animations.img` file is obtained as a result of processing all the animations using the Resource Compiler.

Animation Controller Data

The controller contains animation curves for each joint's position and orientation values needed to play the animation. Even when compressed, controller data can easily take up more than 95% of the total memory required for an animation.

The controller data for animations is stored in CAF files, which contains controller information for a single animation, and a DBA file, which contains controller information for a group of animations.

Mannequin System

Mannequin is in preview release and is subject to change.

Mannequin builds on top of the Geppetto tool to make it easier to construct complex, interactive character animations. Mannequin provides animation layering, blending, additive animations, and partial body animations.

The core of Mannequin is the ability to define families of movements that are variations on a theme (e.g. running injured, running exhausted, running slow, etc.), and to author smooth transitions between those families. Each variation in a family is called a fragment. Fragments are grouped together into families by sharing a fragment ID. Each fragment can carry one or more tags (e.g. tired, injured, gun-in-hand) that selects fragments from within a family during playback, allowing easy authoring of highly varied and situation-specific animation sequences.

With Mannequin you can simplify complex animation code and avoid manually constructing this degree of realism. You can also author preview sequences using your fragments and transitions, reducing iteration time and allowing you to retest scenarios as your animation setup evolves. The Mannequin runtime allows you to play sequences of fragments that smoothly transition from one to the other under the control of C++ code or the flow graph visual scripting system.

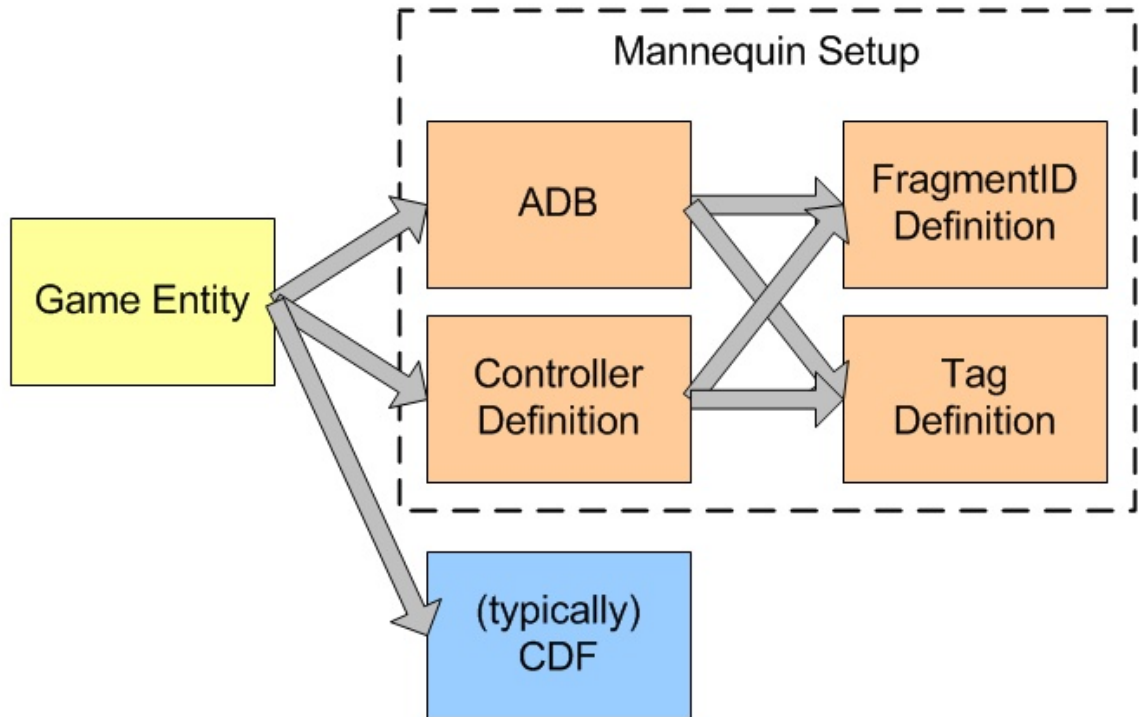
Topics

- [Mannequin System Files \(p. 247\)](#)
- [Creating a Mannequin Entity \(p. 250\)](#)
- [Using Mannequin Editor \(p. 250\)](#)
- [Synchronizing Multiple Characters \(p. 276\)](#)
- [Using Flow Graph with Mannequin \(p. 277\)](#)
- [Debugging Mannequin System Issues \(p. 277\)](#)

Mannequin System Files

Mannequin is in preview release and is subject to change.

The Mannequin system uses a variety of file types.



With the exception of the *.Sequence.xml file, all other .xml files must be created manually using a text editor. Example files are shown as follows. These files must be saved in the Animations \Mannequin directory. You can also create subfolders by character if desired.

Controller Definition File (*ControllerDefs.xml)

Used by the game and by Mannequin Editor to define a mannequin setup. This file is typically referred to from the character Lua file and Mannequin Preview file.

```
<ControllerDef>
  <Tags filename="Animations/Mannequin/Sample/Character_Tags.xml" />
  <Fragments filename="Animations/Mannequin/Sample/
Character_FragmentIDs.xml" />
  <FragmentDefs>
  </FragmentDefs>
  <ScopeContextDefs>
    <Char3P />
  </ScopeContextDefs>
  <ScopeDefs>
    <FullBody3P layer="0" numLayers="3" context="Char3P"/>
    <Additive layer="9" numLayers="3" context="Char3P"/>
  </ScopeDefs>
</ControllerDef>
```

Animation Database File (*.adb)

Used by the game and by Mannequin Editor to store fragments and transitions. This is typically referred to from the character Lua file and other systems such the hit death reaction system.

Tag Definition File (*Tags.xml)

Used by the game and by Mannequin Editor to store tag definitions. The controller definition and animation database files refer to this file.

```
<TagDefinition version="2">
</TagDefinition>
```

FragmentID Definition File (*FragmentIDs.xml)

Used by the game and by Mannequin Editor to store FragmentID definitions. The controller definition and animation database files refer to this file.

```
<TagDefinition version="2">
</TagDefinition>
```

Character Definition File (*.cdf)

Used by the game and by Mannequin Editor to store the main character (.chr) as well as any attachment definitions.

Preview Setup File (*Preview.xml)

Used by Mannequin Editor to determine which controller definition file, animation database file, and character to load.

```
<MannequinPreview>
  <controllerDef filename="Animations/Mannequin/Sample/
Character_ControllerDefs.xml"/>
  <contexts>
    <contextData name="MainCharacter" enabled="1" database=""
context="Char3P" model=""/>
  </contexts>
  <History StartTime="0" EndTime="0">
  </History>
</MannequinPreview>
```

Sequence File (*Sequence.xml)

Used by Mannequin Editor to store animation sequences.

Setting up Mannequin files

Some Mannequin files must be manually setup and edited by hand. Once these files have been set up, you can go into Mannequin Editor and verify the character is displayed in the viewport by selecting **File, Load Preview Setup**, then selecting the *Preview.xml file.

Note

Be sure and select **File, Save Changes** whenever a change is made.

Setting up the *ControllerDefs.xml file

This file name should match the name of the character so it's easier to recognize. This name should also be referenced appropriately in the *Preview.xml file.

To setup the *ControllerDefs.xml file

1. Set the `Tags` filename path to point to the `Tags.xml` file.
2. Set the `Fragments` filename path to point to the `FragmentIDs.xml` file.
3. Save the file.

Setting up the *Preview.xml file

This file name should match the name of the character so it's easier to recognize.

To setup the *Preview.xml file

1. Open the `Character_Preview.xml` file in a text editor.
2. Set the `controllerDef filename` path to point to the appropriate Controller Definition file.
3. Set the `contextData model` path to point to the character model `.cdf` file you want to use in Mannequin.
4. Save the file.

Setting up the .adb file

You will also need to set up the Animation Database (`.adb`) file and assign it to your `*Preview.xml` file. Once the `.adb` file is assigned to the `*Preview.xml` file, Mannequin fragments can be added.

To setup the .adb file

1. In Mannequin Editor, choose **File, Context Editor**.
2. Select the **MainCharacter** entry.
3. Click the **Edit** button.
4. For the **Database** field **<no animation database (adb)>** entry, click the + (Add) button.
5. In the **Edit Context** window, enter the name of the `.adb` file. Click **OK** when done.
6. Verify the **Database** field in the **Edit Context** window points to the `.adbfile`.
7. Click **OK** in the **Edit Context** window.
8. Click **OK** in the **Context Editor** window.

Creating a Mannequin Entity

Mannequin is in preview release and is subject to change.

You can use the Mannequin system to control complex characters, which are often created by code, and you can use the Mannequin object entity type to create a character that can host a Mannequin setup and support any feature of that system.

To create a Mannequin entity

1. In Lumberyard Editor, in the Rollup Bar, click **Entity** and then select **Anim\MannequinObject**.
2. Drag the Mannequin object to the viewport.
3. In the **Entity Properties**, click each of the following to assign the specific files:
 - **ActionController** – Select a `*ControllersDef.xml` file.
 - **AnimDatabase3P** – Select an `*.adb` database file.
 - **Model** – Select a character `*.cdf` file.

Using Mannequin Editor

Mannequin is in preview release and is subject to change.

Mannequin Editor is the primary tool for creating and managing complex character animations.

To open Mannequin Editor

- In Lumberyard Editor, click **View, View Open Pane, Mannequin Editor**. You can also open **Mannequin Editor** from its icon on the main toolbar for Lumberyard Editor.

Fragments Browser

The Fragments browser occupies the left pane of Mannequin Editor by default, and contains the FragmentID Editor tab. The Fragments Browser lists all fragments stored in the animation `.adb` database file. You use the Fragments browser tab in conjunction with the Fragment Editor tab to create fragments, change fragment tags, and create FragmentIDs.

To access the Fragments Browser, click the **Fragments** tab at the bottom left of Mannequin Editor.

You use the FragmentID Editor to edit FragmentID names and fragment definition properties that are stored in the controller definition `ControllerDefs.xml` file.

Fragment Editor

The Fragment Editor occupies the central pane of Mannequin Editor. You use the Fragment Editor to edit mannequin fragments and animation clip properties.

To access the Fragment Editor, click the **Fragment Editor** tab at the bottom of Mannequin Editor. You can also start the editor by double-clicking a fragment in the Fragments browser.

Tag Definition Editor

You use the Tag Definition Editor to create and edit tag definition files (`*.Tags.xml`), which are used for labeling fragments and transitions. To open the Tag Definition Editor, choose **File, Tag Definition Editor**. You can also access it by clicking on the **Tag Definition Editor** button in the FragmentID Editor.

Transitions Browser

The Transitions browser occupies the left pane of Mannequin Editor. The Transitions browser lists all transitions stored in the animation `.adb` database file. You use the Transitions browser in conjunction with the Transition Editor to create transitions.

To access the Transition Editor, click the **Transitions** tab at the bottom left of Mannequin Editor.

Transition Editor

The Transition Editor occupies the central pane of Mannequin Editor. You use it to edit and display mannequin transitions.

To access the Transition Editor, click the **Transition Editor** tab at the bottom of Mannequin Editor. You can also start it access it by double-clicking a transition in the Transitions browser.

Sequences Browser

The Sequences browser occupies the left pane of Mannequin Editor. The Sequences browser lists all the XML sequence files that are stored in the default sequences directory. You use the Sequences browser to select the sequences that you want to open in the Sequence Previewer.

To open the Sequences browser, click the **Sequences** tab at the bottom left of Mannequin Editor.

Sequence Previewer

The Sequence Previewer occupies the central pane of Mannequin Editor. You use the Sequence Previewer to edit and view mannequin sequences from an XML sequence file or to test a new sequence of fragments before saving it to a file.

To access the Sequence Previewer, click the **Previewer** tab at the bottom of Mannequin Editor.

Animation Database Editor

You use the Animation Database Editor to create .adb files and to edit the rules that determine which fragments are stored in a specified .adb file.

To open the Animation Database Editor, choose **File, Animation Database Editor**.

Context Editor

You use the Context Editor to edit the preview setup (*Preview.xml) file. Mannequin Editor needs the preview setup file to determine which controller definition (*ControllerDefs.xml) file to load, which animation database (.adb) file to use, and which characters to use in specific scope contexts.

To open the Context Editor, choose **File, Context Editor**.

Mannequin Error Report

The Mannequin Error Report displays the validation results for any files opened in Mannequin. Validation is performed every time you open a new Mannequin-related file, with errors and warnings listed for each fragment. You can copy validation results to the clipboard, emailed, or open them in Microsoft Excel.

To see the Mannequin Error Report, click the **Mannequin Error Report** tab at the bottom of Mannequin Editor.

Mannequin Fragments (Clips)

Mannequin Editor is in preview release and is subject to change.

The fragment is the basic building block within the Mannequin system. A fragment is a layered collection of time-sequenced animation clips and procedural clips such as poses, attachments, and sounds. You can transition from one clip to another, speed up clips, loop them, or cut them up. This is similar to other nonlinear animation tools. Instead of starting a specific animation directly, the fragment containing the animation is called first. Fragments are defined by their FragmentIDs and tags.

FragmentIDs represent an animation state, such as crouching, idling, or aiming. You use the FragmentID to request rragments. Note that multiple fragments often share the same FragmentID.

You use tags to label fragments with easy-to-remember names, such as *blink*, *yawn*, or *step*. If multiple fragments share the same FragmentID and tag, the fragments are designated as options.

Animators create the animation clips and fragments, while game developers define the FragmentIDs and tags used in Mannequin.

Topics

- [Managing Mannequin Fragments \(p. 253\)](#)
- [Fragment Selection Process \(p. 253\)](#)
- [Using Animation Clips in Fragments \(p. 253\)](#)

- [Using Procedural Clips in Fragments \(p. 254\)](#)
- [Adding Layers to a Fragment \(p. 265\)](#)
- [Managing Fragment Preview Sequences \(p. 265\)](#)

Managing Mannequin Fragments

Mannequin Editor is in preview release and is subject to change.

Use the Mannequin Editor to create, copy, and delete fragments.

To create, copy, or delete a fragment

Open [Mannequin Editor \(p. 250\)](#), choose the **Fragments** tab at the bottom, and do the following. The Fragments browser (panel) is displayed on the left.

- To create a fragment, select the applicable FragmentID, and then click **New**.

Tip

Alternatively, you can also drag the corresponding animation from within Geppetto and drop it onto the FragmentID.

- To copy a fragment, drag it to the desired location.
- To delete a fragment, select it and then choose **Delete**.

Fragment Selection Process

Mannequin Editor is in preview release and is subject to change.

The following process determines which fragment gets selected for use when a game request is made.

- Determine FragmentID for fragment
- Determine scope mask for FragmentID
- Determine scopes assigned to FragmentID
- Determine scope context assigned for each scope
- Determine scope tags assigned for each scope
- Find best matching fragments in the animation database .ADB file assigned to the scope context for each scope. A matching fragment must contain all the scope tags for a scope.
- Ranking matching fragments using tag priorities. Fragments are displayed in the Fragments panel according to rank.
- If there are multiple options with the same tags, the option index is used to select the fragment.

First, the Mannequin system determines which scopes are assigned to the requested fragmentID by looking up the scope mask for the fragmentID. Typically the fragmentID determines the scope mask by itself, but it is possible to specify 'overrides' and select different scope masks based on the global tagstate and requested fragtags. See the file format section in the article on the controller definition file for more on how this is set up. Also, if the calling action requests a specific SubContext, the scope mask and global tags coming from this SubContext's definition extends the ones from the original request. Finally, the scope mask can optionally be extended by the action's 'forced scope mask'.

Using Animation Clips in Fragments

Mannequin Editor is in preview release and is subject to change.

You can easily add animation clips to a fragment and move them around the fragment timeline as desired.

To add an animation clip to a fragment

1. In Mannequin Editor, from the **Fragment Editor** pane, select the applicable fragment or create a new fragment.
2. Add an animation layer to the fragment by right-clicking on the scope for the fragment in the Fragment Editor, going under **Add Layer**, and clicking **AnimLayer**. It is not possible to add animation clips until there is an animation layer available.
3. Open Geppetto, select the animation from the **Animation list**, and then dragging the animation to the desired location in the timeline window for the fragment.
4. Add an empty animation clip by double-clicking on the timeline. With the empty clip selected, you can assign an animation by clicking on the folder icon for the **Animation** property under **Anim Clip Properties**.

Understanding Fragment Clip Zones

The timeline window contains various locations and zones. Understanding them and their effect on fragments can help you add animation clips to a fragment.

The timeline shows various aspects of a clip:

- Blend-in period of the first clip.
- The period where the first clip is playing.
- After the first clip has finished, and last key is repeated by default.
- Blend-in period of the second clip.
- The period where the second clip is playing.

Normally, the second clip is positioned toward the end of the first clip so there aren't any repeating frames. You can also increase or decrease the blend-in time by dragging the vertical bars.

Moving and Snapping Animation Clips

You can drag a clip to move it along the fragment timeline. The default dragging behavior is to snap to the beginning, end, or blend time of a clip.

To snap the clip to the timeline, begin dragging the clip and then press **Shift** as you continue to drag. This snaps the clip to the timeline markers and ignores the other animation clips.

To disable snapping, begin dragging and then press **Ctrl** as you continue to drag. You can now drag the clip to any point on the timeline without snapping to the other clips or to the timeline.

Using Procedural Clips in Fragments

Mannequin is in preview release and is subject to change.

Procedural clips are code snippets that you insert into fragments and run alongside animation clips in that fragment. Like animation clips, procedural clips can be started, stopped, and blended. They are grouped into Lumberyard (CryAction) and game (GameDLL) types.

Procedural clips can range from playing sounds, controlling joints on a character, or aligning an entity to a location. Procedural clips communicate with game code by means of parameters or procedural contexts.

To edit procedural animation clips, you use the Fragment Editor within Mannequin Editor.

To add a procedural clip to a fragment

1. In Mannequin Editor, from the **Fragments Browser**, select the applicable fragment or create a new fragment.
2. Add a procedural layer to the fragment by right-clicking on the scope for the fragment in the Fragment Editor, going under **Add Layer**, and clicking **ProCLayer**. It is not possible to add procedural clips until there is a procedural layer available.
3. Double-click in the timeline on the **ProCLayer** to add a new empty procedural clip.
4. Set the procedural clip **Type** property in the **Procedural Clip Properties** pane.

CryAction Procedural Clips

The following are classified as CryAction procedural clips.

ActionEvent clip

Sends a CryMannequin event to the action controlling this fragment. Specifically, calls `IAction::OnActionEvent`.

Event Name

The name of the event to send.

AimPose clip

Low-level clip to start an AimPose asset. Uses the `AimTarget` parameter as the target, if it exists. If not specified, the target is 10m in front of the entity.

Animation

The Aimpose asset.

Blend

The fade-in duration.

Blend Time

The smoothing time for the spherical coordinates. Higher numbers mean the longitude and latitude have faster smooth aiming at the target.

Animation Layer

The layer (0–16) to play the Aimpose on.

Note

This works differently than the layer parameter inside the LookPose procedural clip, which is a layer index relative to the scope's first animation layer. For more information on scopes, see [Mannequin Scopes \(p. 266\)](#).

AI Signal clip

Sends an AI signal directly to the AI actor interface of the entity on which the clip is playing.

EnterAndExitSignalNames

Signal names sent on the start and finish of the clip, separated by a `|` character.

AttachEntity clip

Attaches an entity to a specific attachment point, and then detaches it on exit.

Attachment Name

The name of the attachment point.

Param Name with EntityId

The name of the parameter that stores the EntityID of the entity to attach.

[AttachProp clip](#)

Attaches a `.chr`, `.skel`, or `.cga` to a specific attachment point (and detaches on exit).

Object Filename

Name of the `.chr`, `.skel`, or `.cga` to attach.

Attachment Name

The name of the attachment point.

[Audio clip](#)

Runs the audio translation layer (ATL) triggers.

Start Trigger

(Optional) ATL trigger to execute at the start.

Stop Trigger

(Optional) ATL trigger to execute at the end.

Attachment Joint

(Optional) name of a joint on which to execute the trigger.

Play Facial

Requests facial animation to match the sound.

Sound Flags

(Reserved)

[FlowGraphEvent clip](#)

Sends events to the flow node `Actor ProcClipEventListener`.

Enter Event Name

Name of the event to send at start.

Exit Event Name

Name of the event to send at end.

[HideAttachment clip](#)

Hides an attachment for the duration of the clip.

Attachment Name

Name of the character attachment to hide.

[IKLayerWeight clip](#)

Controls the weight of an animation layer by a joint's X value.

Joint Name

The joint whose X value controls the layer weight.

Scope Layer

The index of the layer within this scope that this clip should control.

Invert

Use $(1.0 - \text{value})$ as the weight.

LayerAnimSpeed clip

Controls the speed of an animation that is playing in the same scope as this procedural clip through code. The Blend value is not used.

LayerAnimSpeedParam

The name of the floating point parameter that stores the speed value (0 by default).

ScopeLayer

The layer index within the scope of the animation that you want to control.

Invert

Uses $(1.0 - \text{value})$ as the speed.

LayerManualUpdate clip

Controls the (normalized) time of an animation that is playing in the same scope as this procedural clip through code.

Param Name

The name of the floating point parameter that stores the normalized time value (0 by default).

Scope Layer

The layer index within the scope of the animation that you want to control.

Invert

Uses $(1.0 - \text{value})$ as the normalized time.

LayerWeight clip

Controls the weight of an animation layer through code.

Layer Weight Param

The name of the floating point parameter that stores the weight to apply to the layer

Scope Layer

The layer index within the scope of the layer that you want to control.

Invert

Uses $(1.0 - \text{value})$ as the normalized time.

LookPose clip

Low-level clip to start an LookPose asset. Uses the `LookTarget` parameter as the target, if it exists. If not specified, the target is 10m in front of the entity.

Animation

The Lookpose asset.

Blend

The fade-in duration.

Blend Time

The smoothing time for the spherical coordinates. Higher numbers mean the longitude and latitude have faster smooth movement toward the target.

Scope Layer

The layer to play the Lookpose on, relative to the scope's first animation layer.

Note

This works differently than the layer parameter inside the AimPose procedural clip, which is the actual layer number (0–16).

ManualUpdateList clip

Controls the normalized time of animations playing in multiple layers through code.

Param Name

The name of the parameter of type `SWeightData` (four floating-point weights), where the parameter stores the segment normalized time values for the layers.

Scope Layer

The layer index within the scope of the first layer that contains animation that you want to control. All layers after that within this scope are also controlled (up to four layers).

Invert

Use $(1.0 - \text{value})$ as the weight.

ParticleEffect clip

Plays a particle effect.

Effect Name

Name of the particle effect to spawn.

Joint Name

Optional joint to attach the emitter to.

Attachment Name

Optional attachment interface name to attach the emitter to

Position Offset, Rotation Offset

Local-space offset of the emitter. If `Joint Name` or `Attachment Name` is given, the offset is relative to the host entity.

Clone Attachment

If `Attachment Name` is given, create a copy of the given interface instead of using it directly. This allows for more than one effect to play on the same attachment. Disabled by default.

Kill on Exit

Explicitly remove all spawned particles instead of letting them die out on their own. Disabled by default.

Keep Emitter Active

Keep emitter alive after the procedural clip has ended. Disabled by default.

Note

Use with care - if the particle effect goes away on its own, there is no other way to get rid of the effect after it started.

PositionAdjust clip

Procedurally moves the entity towards a target position over time. The target position is taken from the `TargetPos` parameter, which must be set for the clip to play. Used to align characters to ledges.

Blend

Duration of the adjustment.

Offset, Yaw

Additional offset on top of the target position.

Ignore Rotation

Checks to ignore rotation.

Ignore Position

Checks to ignore position.

PositionAdjustAnimPos clip

Moves the entity from the source position (its origin in the DCC tool) of the animation to the target position. If the animation contains movement, this clip might not behave as expected as the delta is only calculated at the start of the animation. In this case, use the `PositionAdjustAnimPosContinuously` clip instead. The target position is taken from the `ParamName` parameter.

Blend

Duration of the adjustment.

Param Name

(Optional) Name of the parameter to use. If not specified, uses the `TargetPos` parameter.

Ignore Rotation

Check to ignore rotation.

Ignore Position

Check to ignore position.

PositionAdjustAnimPosContinuously clip

Moves the entity from the source position (its origin in the DCC tool) of the animation to the target position. The target position is taken from the `TargetPos` parameter, which must be set for the clip to play.

Blend

Duration of the adjustment.

PositionAdjustTargetLocator clip

Takes the character assigned to the specified scope, typically a dependent scope, and moves the entity towards the location of a specific joint of this character.

Blend

Duration of the adjustment.

Target Joint Name

Name of the joint to align to.

Target Scope Name

The scope that has the dependent character attached that you want to align to.

Target State Name

Not used.

SetParam clip

Sets a float parameter to a certain value.

Param Name

The name of the parameter.

Blend

The time it takes to reach the target value.

Target

The target value.

Exit Target

The value to go to after the clip ends.

WeightedList clip

Controls the weight of consecutive layers through code.

Param Name

The name of the parameter of type `SWeightData` (four floating-point weights), which stores the weights for the layers.

Scope Layer

The layer index within the scope of the first layer that you want to control. All layers after that within this scope are also controlled (up to four layers).

Invert

Uses $(1.0 - \text{value})$ as the speed.

Game Procedural Clips

The following are classified as GameDLL procedural clips.

Aiming clip

Requests that the Aimpose be enabled.

Blend

Fade-in duration for the Aimpose.

AimSmoothing clip

Relies on Aimpose or Aiming scope setup. Controls smoothing parameters for the polar coordinates while moving toward or following a target.

Smooth Time Seconds

The "smoothing time" for the spherical coordinates. Higher numbers mean the longitude or latitude have faster smooth movement towards the target.

Max Yaw Degrees Per Second

Maximum degrees per second in the yaw direction.

Max Pitch Degrees Per Second

Maximum degrees per second in the pitch direction.

AttachPnt clip

Attaches the pick-and-throw weapon.

Attachment Point

Name of the attachment interface to use.

ColliderMode clip

Overrides the ColliderMode for the character.

Valid values:

- Undefined (give up control)
- Disabled (no collisions)
- GroundedOnly

- Pushable
- NonPushable
- PushesPlayersOnly
- Spectator

CompromiseCover clip

Tells the AI system that cover has been compromised.

CopyNormalizedTime clip

Synchronizes animation within two layers by automatically copying over the segment normalized time from an animation in one layer to an animation in another layer.

Source Scope

The scope from which to copy.

Source Layer

The layer within the source scope to look for the source animation.

Layer

The layer within the current scope that contains the animation that you want to synchronize

FacialSequence clip

Plays a facial sequence.

Filename

The facial animation sequence .fsg file to play

Continue After Exit

Whether to continue playing the sequence after the clip ends. Ignored when looping the sequence, in which case the default behavior is used, so the sequence stops playing when the clip ends.

Looping

Whether to loop the sequence.

Looking clip

Relies on Lookpose or Looking scope setup. Requests the Lookpose to be enabled. Blend-in time is used as fade-in time for the Lookpose.

Blend

Fade-in duration for the Lookpose.

MovementControlMethod clip

Override the movement control method of the character.

Horizontal

Horizontal movement control method. Valid values:

- 0: Undefined (no override)
- 1: Entity driven
- 2: Animation-driven
- 3: Animation-driven with collision in the horizontal plane

Vertical

Vertical movement control method. Valid values:

- 0: Undefined (no override)
- 1: Entity-driven
- 2: Animation-driven

Ragdoll clip

Makes a character turn into a ragdoll and optionally blend back to animation.

Blend

Defines the time range during which the character starts randomizing.

Sleep

When set to 0, the AI exhibits ragdoll behavior. When set to 1, the AI stays alive during the ragdoll phase and blends back to animation.

Stiffness

Determines how much the ragdoll behavior follows the animation.

Note

The `Sleep` parameter is only used by the blend-from-ragdoll game code, which is triggered by calling `CActor::Fall()`.

This triggers the `CAnimActionBlendFromRagdollSleep`, which makes the character exhibit ragdoll behavior: It plays the fragment with fragmentID `BlendRagdoll` and tags containing `standup+blendin+ragdoll`. This fragment has to contain a `Ragdoll` clip with the `sleep` value set to 1.

For standing up, a `CAnimActionBlendFromRagdoll` is started after the ragdoll phase has ended. This action relies on all possible standup animations to be an option for the fragmentID `BlendRagdoll` and tags containing `standup+blendout`. The best matching animation is chosen based upon the first frame of these.

SetStance clip

Tells an AI character it is in a certain stance. It does not trigger stance-change animation. This is useful to annotate an animation that ends up in a stance other than it started in, such as in a scripted sequence that can be interrupted. When the sequence is interrupted, the game knows the AI is in another stance.

Stance

Stance name. Valid values:

- Null
- Stand
- Crouch
- Prone
- Relaxed
- Stealth
- Alerted
- LowCover
- HighCover
- Swim
- Zero-G

SwapHand clip

Temporarily move an attachment from the right hand to the left. This is hardcoded to use the attachment names `weapon` and `left_weapon`.

TurretAimPose clip

Controls aiming and aimpose of the turret entity.

Blend

The fade in time of the Aimpose.

Animation

The Aimpose asset to use.

Blend Time

Unused.

HorizontalAimSmoothTime

The smoothing time for the yaw direction.

VerticalAimSmoothtime

The smoothing time for the pitch direction.

Max Yaw Degrees Per Second

Maximum degrees per second that the turret rotates in the yaw direction.

Max Pitch Degrees Per Second

Maximum degrees per second that the turret rotates in the pitch direction.

WeaponBump clip

First-person weapon bump animation that occurs when the player lands.

Time

The amount of time that the bump animation plays.

Shift

How much the weapon moves on screen after the player lands.

Rotation

How much the weapon rotates.

WeaponPose clip

Places the weapon on a specific location on the screen. It has three modes: right hand, left hand, and zoom. Only one of these modes can be active at a time; however, more than one clip can run in parallel.

Pose Type

The default is 0, which means right hand. This changes the weapon's position on screen starting from the idle pose position. A value of 1 means zoom, which places the weapon on the screen when the player decides to zoom in. A value of 2 means left hand, which can be used to modify the original base pose to accommodate underbarrel attachments.

Zoom Transition Angle

The default is 0, which defines the angle that the weapon rotates during a zoom transition. `Zoom Transition Angle` is only read if `Pose Type` is set to 1 (zoom). Otherwise this parameter is totally ignored.

Position, Rotation

Defines the pose itself as an offset to the base pose. Rotation is defined in angles.

WeaponRecoil clip

Activates the recoil behavior on the weapon. It triggers a recoil animation every time the weapon fires.

Damp Strength

How quickly the weapon comes back to rest pose after a kick.

Fire Recoil Time

Attack time of the recoil kick. A value of 0 applies the kick in a single frame, which is not recommended, since it can make the animation look jerky.

Fire Recoil Strength First, Fire Recoil Strength

The kick strength. Fire Recoil Strength First has the same behavior as Fire Recoil Strength but is applied to the first shot only. For best results in rapid fire modes, make Fire Recoil Strength First much higher than Fire Recoil Strength.

Angle Recoil Strength

The degree of deviation the weapon experiences after each shot.

Randomness

The overall organic feeling of the recoil animation.

[WeaponSway clip](#)

This clip activates the laziness effect on the player's moving hands. Careful setup of the clip simulates different weight feelings for different weapons. After the clip is activated, it starts reading the player movement and computes weapon offsets in real time.

Ease Factor Inc, Ease Factor Dec

How much it takes for the look poses to blend in (Inc) or out (Dec) when player looks around

Velocity Interpolation Multiplier

Fine tune control for strafing.

Velocity Low Pass Filter

The filter applied to the player movement to make the sway more reactive or intensive.

Acceleration Smoothing

Helps make strafe poses less linear and more realistic.

Acceleration Front Augmentation

The degree to which it makes more sense for the strafe poses to move back and forth as opposed to left and right.

Vertical Velocity Scale

Changes the look poses behavior when player is going up or down a ramp.

Sprint Camera Animation

Do not use.

Look Offset

The degree to which the weapon moves around the screen while player looks around.

Horiz Look Rot

The rotation applied to the weapon when the player looks left and right.

Vert Look Rot

The rotation applied to the weapon when player looks up and down.

Strafe Offset

The degree to which the weapon moves when player moves around.

Side Strafe Offset

The rotation of the weapon when the player starts strafing either to the left or to the right.

Front Strafe Rot

The rotation of the weapon when the player starts moving forward or backward.

[WeaponWiggle clip](#)

Activates weapon wiggling and shaking.

frequency

Shake frequency.

intensity

Shake intensity.

Adding Layers to a Fragment

Mannequin is in preview release and is subject to change.

You can add multiple layers of animation clips to one fragment. In these layers, you can place additive or override animations to add variation to the base layer's animation. In some instances, the number of layers you can add may be limited by the scope. For information about scope, see [Creating and Editing Scopes \(p. 266\)](#).

To add a layer to a fragment

1. In Mannequin Editor, from the **Fragment Editor** pane, right-click the fragment scope, and then click **Add Track, AnimLayer**.
2. If you're adding a procedural clip layer instead of an animation layer, when you right-click on the fragment scope, go to **Add Track** and click on **ProcLayer**. Currently, when you add a new layer, it is added directly below the lowest layer. You cannot change the order of layers at this time, instead, just reorganize the clips as necessary.

Managing Fragment Preview Sequences

Mannequin is in preview release and is subject to change.

You can save, load, and view fragment preview sequences in Mannequin Editor.

To save a fragment preview sequence

1. In Mannequin Editor, in the **Sequences** browser, under **Sequences**, select the sequence.
2. Click **Previewer, Save Sequence**. Name the sequence and click **Save**.

To load a fragment preview sequence

1. In Mannequin Editor, in the **Sequences** browser, under **Sequences**, select the sequence.
2. Click **Previewer, Load Sequence**.

You can preview how fragment sequences look without actually running the game. This is useful for debugging sequences and previewing what-if scenarios, such as how the game would look if requesting the `Move after Idle while Kneeling` fragment sequence, for example.

To view a fragment preview sequence

1. In Mannequin Editor, click the **Previewer** tab at the bottom.
2. Select the sequence and click the start button. You can also rewind and fast forward through the sequence.

Mannequin Fragment IDs (Animation States)

Mannequin is in preview release and is subject to change.

A FragmentID is the main label under which a fragment is stored.

FragmentIDs are character animation states, such as moving, idling or firing. Game code uses a FragmentID to access a fragment. Typically, a number of different fragments may be assigned to the same FragmentID. For example, the animation could include several different moving fragments, such as moving while standing, moving while crouching, or moving plus some random variation.

Typically, a game developer creates a different FragmentID for every basic character animation state, while animators create animation clips and the associated fragments for those FragmentIDs.

You can create and edit FragmentIDs in Fragment Editor within Mannequin Editor. You store the FragmentIDs in a FragmentID definition file (`*Actions.xml`), which is referred to from the controller definition file (`*ControllerDefs.xml`).

If animations are required between FragmentIDs, you can use a transition.

Mannequin Scopes

Mannequin is in preview release and is subject to change.

Typically, individual portions of a character's body will be in different animation states. Scopes are animation channels assigned to the parts of a character's body on which fragments are triggered and played. For example, one animation fragment can be played for the entire body, another fragment for the lower body, another fragment for the torso, and another fragment for the head. These scoped animations can be played independently or synchronized together.

To create and edit scopes, you modify the following parts of the controller definition file (`*ControllerDefs.xml`).

- Primary entity (character)
- Attached entities (head, weapon)
- Animation layers
- Animation database for fragments (.adb)

Topics

- [Creating and Editing Scopes \(p. 266\)](#)
- [Creating and Editing Scope Contexts \(p. 267\)](#)
- [Using Scope Masks \(p. 268\)](#)
- [Playing Fragments on Scopes \(Actions\) \(p. 268\)](#)

Creating and Editing Scopes

Mannequin is in preview release and is subject to change.

Mannequin scopes are stored in the controller definition `*ControllerDefs.xml` file, which contains the setup of a mannequin character.

The following shows an example `*ControllerDefs.xml` file. You use FragmentID Editor in Mannequin Editor to edit the scope masks and related flags. To edit the remaining sections, you need a text editor. The FragmentID Editor appears when you create a new FragmentID in the Fragments pane.

```
<ControllerDef>
  <Tags filename="Animations/Mannequin/ADB/sampleTags.xml" />
  <Fragments filename="Animations/Mannequin/ADB/sampleFragmentIds.xml" />
  <SubContexts/>
  <FragmentDefs>
    <move scopes="FullBody+Torso" flags="Persistent" />
    <burst_fire scopes="Torso+Weapon">
      <Override tags="heavyMortar" fragTags="boosted" scopes="Torso" />
    </burst_fire>
  </FragmentDefs>
  <ScopeDefs>
    <FullBody layer="0" numLayers="3" context="MainContext" />
    <Torso layer="3" numLayers="3" context="MainContext" />
    <Face layer="6" numLayers="0" context="MainContext" Tags="scope_face" />
    <Weapon layer="0" numLayers="2" context="WeaponContext" />
  </ScopeDefs>
</ControllerDef>
```

The controller definitions file can include a number of different tags:

- `Tags` – References the scope's tag definition (`*Tags.xml`) file.
- `Fragments` – References the scope's FragmentID definition (`*Actions.xml`) file.
- `FragmentDefs` – Contains one entry for each FragmentID specified in the FragmentID definition file. For each FragmentID, a `scopes` attribute defines the scopemask, optional `flags` attributes that control fragment play, and the `override` attribute that overrides the scopemask when certain tags and fragtags are matched.
- `SubContexts` – Lists all subcontexts available.
- `ScopeDefs` – Defines the scopes and scope contexts used. Each element defines a scope.

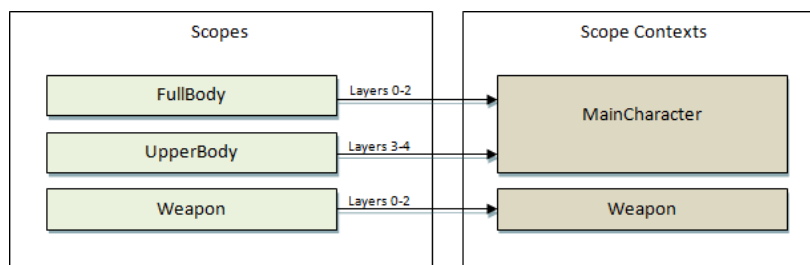
Creating and Editing Scope Contexts

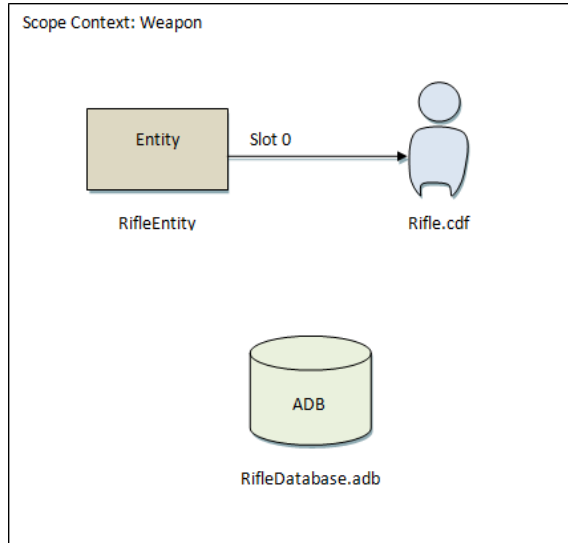
Mannequin is in preview release and is subject to change.

A scope context defines which entity, character, and animation database to use. You can use the same scope context for multiple scopes. Because every scope is attached to a scope context, at least one scope context is needed for each character.

Scope context properties may change during runtime, so it is possible to swap the entity, character instance, or animation database at any time. You can use this technique to change weapons or attach other characters to the player during a synchronized animation, for example.

Scope contexts are defined in the controller definition file (`*ControllerDefs.xml`).





The implementation of the animated character game object extension is hardcoded to support the scope contexts Char1P, Char3P, and Audio.

The controller definitions file must use the Char3P scope context when using Mannequin object or the actions and layers will not play, as shown below:

```
<ScopeContextDefs>
  <Char3P />
</ScopeContextDefs>
<ScopeDefs>
  <FullBody layer="0" numLayers="3" context="Char3P"/>
</ScopeDefs>
```

Using Scope Masks

Mannequin is in preview release and is subject to change.

A scope mask is the set of scopes that a fragmentID runs on. Each fragmentID has a scope mask associated with it, as defined in the Controller Definition File using the FragmentID Editor. When an action requests a fragmentID, the action owns the scopes in the FragmentID scope mask and starts playing fragments on these scopes.

For example, a Fire Weapon fragmentID could have a scope mask containing the weapon scope for animating the weapon as well as the torso scope. It doesn't need to contain the other scopes of the character because it can control the torso independently of the rest of the body using additive and partial-body animations.

Playing Fragments on Scopes (Actions)

Mannequin is in preview release and is subject to change.

Scopes are defined portions of a character's body where fragments are triggered and played. By playing different sequences of animations (fragments) on specific parts of a character's body (scopes), realistic movements and motions can be achieved. This process is called a mannequin action.

One fragment can play on the full-body scope (walking), while another fragment plays on the torso scope (rotating), and yet another fragment plays on the head scope (looking at target), all simultaneously.

Fragments use Flow Graph nodes or game code to play on scopes.

Mannequin Tags (Animation Contexts)

Mannequin is in preview release and is subject to change.

When multiple fragments are assigned to a single FragmentID, such fragments are simply variations of ideas expressed in that FragmentID. With Tags, you can label fragments for more specific character contexts like crouched, shooting, or scared.

The game looks for tags based upon the state of the game character. For example, when a character is crouching, the game starts looking for fragments tagged as `crouched`. And when the character is using a machine gun, the game looks for fragments tagged as `machineGun`. If the game is looking for both of these tags at the same time, it first looks for a fragment with both tags. Next, the game looks for fragments labeled either `machineGun` or `crouched`. Finally, it looks for a fragment with an empty set of tags that acts as a fallback. Fragments with other tags such as `swimming` are not selected.

Multiple fragments can have the same set of tags and FragmentID. In this case, the game automatically assigns each fragment an option index. By default a random option index is chosen, but you can have the game select a specific one if needed, such a particular fragment for animation streaming. For example, if you have 20 variations (options) but want to stream in only one of them, you can override the random selection process and make sure that the specific variation you streamed in is selected.

When working with tags, it's useful to know the following terms:

- **Tag Definition** – A collection of tags.
- **Tag Group** – A mutually-exclusive set of tags.
- **Tag State** – A combination of tags, such as `crouching+pistol`.

Topics

- [Using Tag Definitions \(p. 269\)](#)
- [Using Tag State Keys \(p. 270\)](#)
- [Using FragmentID Tags \(FragTags\) \(p. 270\)](#)
- [Assigning Fragment Tags \(p. 271\)](#)

Using Tag Definitions

Mannequin is in preview release and is subject to change.

Tag definitions define a collection of fragment tags. You use Tag Definition Editor within Mannequin Editor to create tag definitions and store them in a tag definition (`*Tags.xml`) file, or you can create the tag definitions file manually in a text editor.

Each tag must have a unique name within a tag definition file. Tag definition files can include (nest) other tag definition files. To edit a nested tag definition, you manually edit the tag Definition (`*Tags.xml`) file. For all other tag definitions, you can use the Tag Definition Editor, which you access from the Fragments pane.

Note that Lumberyard ignores the casing of tags.

Using Tag State Keys

Mannequin is in preview release and is subject to change.

A tag state is a combination of tags from a tag definition. Tag states are represented by a list of tags separated by + characters. For example `crouching+pistol` defines a tag state combining the tags `crouching` and `pistol`.

A game can set global tags describing the current state of the character, or the global tag state. This typically contains global state information like character type, stance, and weapon attachment for example.

The global tag state is the `tags` member of the ActionController `SAnimationContext`, which is found with `IActionController::GetContext()`.

Study the following numbered fragment timeline screen shots to understand the use of tag state keys:

- Select the `{kneeling+tired}` tag state key.
- Disable the `tired` tag in the key.
- Note the tag state key changes to `{kneeling}`

The FragmentID (below the tag state key) selected is the default `Idle{<default> - 0}`. This fragment represents the best match for the game's request.

For the `{kneeling+tired}` tag state key, select the `tired` tag check box again.

Now drag the `{kneeling+tired}` tag state key to the right in the timeline.

This simulates a situation where the game requests `{kneeling+tired}` after requesting the `Idle{<default> - 0}` fragmentID. This means that at the moment `Idle` is requested, the tags are not set, and the default FragmentID is selected.

The order in which game requests arrive in the Mannequin system has an influence on which fragments get selected eventually. For example, if you want to move a certain fragment around, you need to select both the FragmentID and the tag state key above it.

Using FragmentID Tags (FragTags)

Mannequin is in preview release and is subject to change.

FragmentID-specific tags, also known as fragtags, are tags that are assigned only to fragments with a specific fragmentID.

Many fragment tags don't have to be available to all fragments. For example, there might be a `hit` fragmentID that groups fragments containing hit reaction animations. The actual type of hit, such as `headshot` or `explosion` would then be encoded in tags. But such tags are only useful in the context of the `hit` fragmentID, so such tags are considered fragmentID-specific.

Fragtags are created by creating a new tag definition using the Tag Definition Editor in Mannequin Editor. This new tag definition is then assigned to a FragmentID using the FragmentID Editor.

Each fragmentID can have only one tag definition containing fragtags, but for more complicated cases you can import other tag definition files hierarchically from the main tag definition.

Fragtags are stored in separate tag definition files that are linked to from the fragmentID definition file as sub-tag definitions.

Assigning Fragment Tags

Mannequin is in preview release and is subject to change.

Tags are added to fragments to limit which fragments can get selected. For example, a "tired" tag can be assigned to a fragment so it only gets selected when the character is tired. Or for example, other fragments can be assigned "kneeling" or "standing" tags to create different "stance" variations for the same animation.

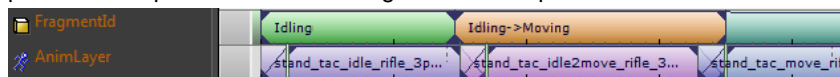
For this example, "stance" is considered a tag group. Some tags are inside tag groups, some other tags, like "tired", are not. Putting tags in a group ensures sure you can only select one of the tags in the group at the same time. So a character cannot be both "standing" and "kneeling" at the same time for example, but can be both "kneeling" and "tired". The various tags within a tag group are called tag options.

The order in which the fragments are listed in the Fragments browser reflects the order in which they are selected. If there are multiple equivalent matches, the first match in the list is selected. For example, you might have a tag called "tired" and a tag called "scared." You have one fragment tagged "tired" and another fragment tagged "scared." The game looks for a fragment for a character that is both "tired" and "scared." If "tired" and "scared" have the same priority, it is undefined which fragment is chosen, but the Mannequin Editor shows you the fragments in the selection order.

Mannequin Animation Transitions

Mannequin is in preview release and is subject to change.

Animation transitions blend together multiple fragments in a specified sequence. Specifically, game code requests multiple FragmentIDs sequentially, and those associated fragments need to be blended together. With Mannequin, you can specify complex transitions between the fragments, such as specifying exactly how individual layers within fragments are combined, or the ability to add new procedural clips in between existing animation clips.



Transitions are stored with their associated fragments in the XML-based animation database .adb file. The `FragmentBlendList` element contains the transitions, as the following shows.

```
<FragmentBlendList>
  <Blend from="" to="idlePose">
    <Variant from="" to="">
      <Fragment selectTime="0" enterTime="0">
        <AnimLayer>
          <Blend ExitTime="0" StartTime="0" Duration="0"/>
        </AnimLayer>
      </Fragment>
    </Variant>
  </Blend>
</FragmentBlendList>
```

Topics

- [Creating and Editing Transitions \(p. 272\)](#)

- [Setting Transition Parameters \(p. 272\)](#)
- [Cyclic Transitions \(p. 273\)](#)

Creating and Editing Transitions

Mannequin is in preview release and is subject to change.

Without transitions, a character's motion snaps between two fragment clips using the default blend time specified for the beginning of the second fragment. Add custom transitions for more realistic motion.

To add a new transition between two fragments

1. In Mannequin Editor, click the **Transitions** tab and then click the **New** button near the bottom left.
2. Select the first fragment in **Fragment ID From** and select any associated tags.
3. Select the second fragment in **Fragment ID To** and select any associated tags.
4. View the new transition in the **Transitions** list and the **Transitions Preview** timeline window. The transition is colored orange.

By default, the transition duration is the default blend time. You can easily change the transition duration time.

To change transition duration time

- In the **Transitions Preview** timeline window, drag the vertical divider line to the right or the left for the transition to increase or decrease the duration.

To add an animation to a transition

1. In the **Transitions Preview** timeline window, double-click after the start of the orange transition block.
2. Select an animation clip in **Animation**.
3. Drag the new clip in the **Transitions Preview** timeline window until the blend time of the second clip overlaps with the end of the new transition clip.
4. Right-click the first fragment and click **Insert Transition**.

The default transition behavior for a nonlooping fragment is to wait until the end of the fragment to begin. You can adjust a transition so that a second fragment does not start playing until the first fragment is finished playing (and not immediately when requested).

To delay transition start time

1. Select any key on the transition.
2. Under **Transition Properties**, adjust the value of the **Earliest Start Time** property. This value is relative to the end of the previous fragment.

Setting Transition Parameters

Mannequin is in preview release and is subject to change.

There are two broad types of parameters that can be edited using Mannequin Editor – action parameters and motion parameters.

Action Parameters

These are parameters the game uses when playing actions and procedural clips. Some examples include providing a target position when aligning an entity, providing a weight value when fading an animation in or out, or providing a sound parameter.

All action parameters have a name and a value.

Motion Parameters

These are parameters that get passed to the blend spaces (bspaces) parametric animation system. You can preview how these parameters influence animation by adding keys for them on the Params track in Mannequin Editor.

Cyclic Transitions

Mannequin is in preview release and is subject to change.

To set up a transition from a looping or parametric animation, set the transition **Select Time** value relative to one cycle (or segment) of the animation clip. If the fragment changes duration, the time would automatically adjust in the proper proportion. You do this by selecting **Cyclic Transition** under **Transition Properties**. This turns the select time into a value between 0 and 1 instead of a value in seconds.

The following fragment shows:

- The first fragment is looping.
- **Cyclic Transition** is selected
- The select time is 0.5, and this translates into 50% along the cycle. Also displayed is the range of the select time, in this case it runs all the way to the end of the cycle. After that the second transition with select time of 0 is selected.

Unless marked as being locked, cyclic transitions always trump the previous fragment, regardless of action priority. The **Earliest Start Time** value is thus effectively ignored.

It is possible to delay transitions in an animation using the **Earliest Start Time** value. By default, this value is relative to the end of the previous fragment. For fragments with no clear ending, such as fragments with looping end clips, this is handled by "locking" the cycling so that transitions are triggered when preceding animations are a certain portion of their run cycle.

In this case, select both **Cyclic Transition** and **Cyclic Locked**. This enables the **Earliest Start Time** value to be stored cyclically in that the time restarts at zero after each cycle.

Mannequin Animation Actions

Mannequin is in preview release and is subject to change.

An action is a programmatic construct that used to control animations and synchronize them with the game, combining game code with simple high-level animation control.

When an action is installed, it "owns" one or more scopes and can request FragmentIDs to play on those scopes. Each scope can be controlled by only a single action. Many actions can be running in parallel as long as they all control different scopes.

Although each action can only request one FragmentID at a time, it can nonetheless sequence multiple such requests in a row. If you want to implement an animation state machine, either you queue multiple actions that each push a single FragmentID and you handle the state machine externally, or you queue

a single action that has an internal state machine that requests the appropriate FragmentIDs. The latter is typically how Lumberyard handles basic locomotion state machines.

The Mannequin ActionController (IActionController) is the root object that controls a character mannequin. You configure it in a controller definition (*ControllerDefs.xml) file, which defines the FragmentIDs, scopes, and scope contexts. It also installs actions onto scopes and holds the global tag state.

Topics

- [Creating Mannequin Actions \(p. 274\)](#)
- [Mannequin Action Queuing \(p. 274\)](#)
- [Using Action Subcontexts \(p. 275\)](#)

Creating Mannequin Actions

Mannequin is in preview release and is subject to change.

You may want to create a new action class or simply use a generic one for simple cases.

With this constructor, you can do the following:

- Set the relevant FragmentID, which is the first FragmentID that gets requested.
- Set any FragmentID-specific tags (frag tags).
- Set the action priority, which is used to manage overlapping actions and actions that want to own the same scope. Higher numbers indicate higher priority.

The following shows a sample code snippet that creates an action that plays the `Idle` FragmentID.

```
const FragmentID idleFragmentId = m_pAnimationContext->controllerDef.m_fragmentIDs.Find( "Idle" );
const int actionPriority = 0;

IActionPtr pAction = new TAction< SAnimationContext >( actionPriority,
idleFragmentId );
```

Mannequin Action Queuing

Mannequin is in preview release and is subject to change.

Actions are queued onto the target Mannequin ActionController(IActionController), which is the root object that controls the character mannequin.

For actors, the ActionController is accessible via the AnimatedCharacter extension (`IAnimatedCharacter::GetActionController()`).

A queuing statement looks like the following: `pActionController->Queue(pAction);`

This is a priority queue where higher priority actions are selected first. For each frame, the Mannequin system checks whether queued actions can be installed on the applicable scopes. Specifically, the FragmentID is retrieved and associated scope mask is determined.

If an action has higher priority than all the actions currently owning those scopes, it is installed immediately and skips any waiting times in transitions. This is called trumping. Otherwise the candidate action waits for those actions to finish or for a suitable transition to gracefully stop the current action.

When an action gets selected from the queue, it gets installed on its scopes, and its fragmentID is pushed on and updated before the next batch of animations are sent off for processing.

Actions that get pushed away are stopped unless the `interruptible` flag is set, in which case they get pushed back to the queue and return when they can. The `interruptible` flag is typically used for actions controlling Movement or Idling actions. These are low-priority interruptible actions that run by default on certain scopes but get pushed back by more specific actions.

Using Action Subcontexts

Mannequin is in preview release and is subject to change.

Subcontexts are a way for programmers to explicitly refer to a single logical role (out of multiple such roles) when requesting an action. Subcontexts are a convenience when dealing with FragmentIDs whose scope mask encompasses multiple scope contexts, where each context refers to a different role. For example, a car could have multiple seats, each one with its own scope and unique associated tag. Subcontexts do not affect fragments but rather provide additional contextual information when dealing with actions that involve multiple independent scope contexts.

Subcontexts are defined in the controller definition (`*ControllerDefs.xml`) file. Each subcontext has a unique name and exposes a scope mask and global tag state. Using the car example, the following code shows how the car's controller definition file could define different subcontexts for different seats, each seat having its own set of scopes.

```
<SubContexts>
  <Driver scopeMasks="Driver+DoorDriver" tags="Driver"/>
  <Passenger scopeMasks="Passenger+DoorPassenger" tags="Passenger"/>
</SubContexts>
```

Upon entering the car, a character typically gets enslaved to either the Driver or Passenger scope context. When requesting a FragmentID that is local to one of the seats (for entering or leaving the vehicle), the game needs to state the correct subcontexts. This is done by requesting the subcontext in a mannequin action. The following snippet shows an action installed on a subcontext:

```
// Driver just entered the vehicle, already enslaved to it
IAActionController* pVehicleActionController;
IAAction* pEnterVehicleAction;

// ...

// Queue the "EnterVehicle" FragmentID with the suitable SubContext
pEnterVehicleAction->SetFragment(EnterVehicle);
pEnterVehicleAction->SetTagContext(isDriver ? Driver : Passenger); // Change
  SubContext based on which role the enslaved character is supposed to have
pVehicleActionController->Queue(pEnterVehicleDriverAction);
```

This results in automatically adding the matching scope mask and global tags to the default state during the fragment selection process for this action. In this example, with the proper setup, Mannequin would then know which character and which door to animate when processing this action. As such, the FragmentID can be queried and resolved to different scope masks and ultimately fragments based on the given subcontext.

Adding Mannequin Audio

Mannequin is in preview release and is subject to change.

Sound is added in the Mannequin system by inserting audio procedural clips to fragments. Sound effects can be very granular, with different sounds used for different weapons in different states of firing for example. The general process is as follows:

- Reserve a scope just for audio, and place an ATL-Trigger on it.
- Edit the scope mask to include the audio scope.
- Add a ProcLayer track for the audio scope.
- Add a procedural clip, and set the type to `Audio`.
- Set the appropriate start and stop triggers as well as other parameters to affect the sound's properties.

Synchronizing Multiple Characters

Mannequin is in preview release and is subject to change.

Synchronizing multiple animated characters is a common task. Practical examples include animating a weapon in sync with a character's body when reloading or firing, or synchronized actions across multiple characters, such as for stealth kills.

This can be achieved with Mannequin through the use of scope contexts and the concept of coupling or enslavement.

The first step required to synchronize a secondary character with a primary one is to add an extra scope and scope context in the host character's Controller Definition `*ControllerDefs.xml` file. The secondary character is then attached to the newly-created scope context. The following is an example `ControllerDefs.xml` file:

```
<ControllerDef>
...
<ScopeDefs>
  <FullBody1P layer="0" numLayers="3" context="Char1P"/>
  ...
  <FullBody3P layer="0" numLayers="3" context="Char3P"/>
  ...
  <Weapon layer="0" numLayers="3" context="Weapon"/>
  ...
  <AttachmentTop layer="0" numLayers="3" context="attachment_top"/>
  <AttachmentBottom layer="0" numLayers="3" context="attachment_bottom"/>
  <SlaveChar layer="0" numLayers="3" context="SlaveChar" Tags="slave"/>
  <SlaveObject layer="0" numLayers="3" context="SlaveObject" Tags="slave"/>
</ScopeDefs>
</ControllerDef>
```

This example shows seven scopes using seven different contexts, which means that fragments can be synchronized for up to seven different characters.

Parameters

Scope	Scope Context	Layers
FullBody1P	Char1P	0, 1, 2
FullBody3P	Char3P	0, 1, 2
Weapon	Weapon	0, 1, 2

Scope	Scope Context	Layers
AttachmentTop	attachment_top	0, 1, 2
AttachmentBottom	attachment_bottom	0, 1, 2
SlaveChar	SlaveChar	0, 1, 2
SlaveObject	SlaveObject	0, 1, 2

The `Actor:EnslaveCharacter` Flow Graph node can be used to couple characters together in order to play synchronized animations.

When coupling a character, you can optionally use a different Animation Database .ADB file if needed, depending on setup in the Mannequin Editor. If left empty, fragments will be queried from the host character's .ADB file.

Using Flow Graph with Mannequin

Mannequin is in preview release and is subject to change.

Some Mannequin system functionality is available using the `Actor:PlayMannequinFragment` and `Actor:PlayMannequinFragment` Flow Graph nodes.

The `Actor:PlayMannequinFragment` node looks for a fragment to play using the provided `FragmentID` and `TagState`. This fragment is in a Mannequin Action and queued with the given priority. The `Actor:PlayMannequinFragment` node can also stop this action using the `ForceFinishLastQueued` input, or pause/resume the entire Mannequin ActionController.

Some guidelines and best practices for using this node include the following:

- Make sure that querying fragments do not conflict with AI, player, or game logic if the entity being targeted is also driven by other game code
- Select priority based on what you want to interrupt. Movement fragments run at priority 4, hit reactions at priority 5, and death reactions at priority 6.
- You cannot start an action on one node and stop it with another node. Actions are not shared across nodes.

Debugging Mannequin System Issues

Mannequin is in preview release and is subject to change.

Lumberyard offers a number of methods for debugging Mannequin system issues. In addition to the ones listed below, you can also analyze an error report.

Topics

- [Using Console Variables \(p. 277\)](#)

Using Console Variables

Mannequin is in preview release and is subject to change.

Use the following console variables for debugging the Mannequin system.

- `mn_allowEditableDatabasesInPureGame mn_DebugAI` – Do not store editable databases.
- `mn_listAssets` – Lists all the currently referenced animation assets.
- `mn_reload` – Reloads animation databases.
- `mn_sequence_path` – Default path for sequence files.

Cinematics System

Cinematics, also known as sequences or cutscenes, are interactive movie animations with time-dependent control over objects and events. You can use Lumberyard to add cutscenes to your game.

You can also add scripted events so that a sequence of objects, animations, and sounds are triggered in the game. The player can view these sequences from their own (first person) or another's (third person) perspective.

Sequences consist of the following elements (listed in hierarchical order), which are created and managed from Track View editor:

- **Node** – Each sequence comprises a top-level director (scene) node, one or more camera nodes, image effects nodes, and entity nodes.
- **Track** – Depending on the type, each node consists of multiple tracks, such as position, animation, sound, lighting, text, and events. Tracks are displayed in the track timeline pane.
- **Key** – A key is a setting for a property at a specific time. As the sequence plays, keys are interpolated based on their in and out tangent values set in Track View Graph Editor.

Topics

- [Cinematics Best Practices \(p. 279\)](#)
- [Using Track View Editor \(p. 280\)](#)
- [Track View Nodes \(p. 281\)](#)
- [Creating Scenes \(p. 291\)](#)
- [Managing Track Events \(p. 295\)](#)
- [Cinematics Cameras \(p. 296\)](#)
- [Cinematics Lighting \(p. 304\)](#)
- [Animating Characters in Scenes \(p. 306\)](#)
- [Adding Player Interactivity \(p. 313\)](#)
- [Using Layers for Scenes \(p. 318\)](#)
- [Capturing Image Frames \(p. 318\)](#)
- [Debugging Cinematic Scenes \(p. 319\)](#)

Cinematics Best Practices

Consider adopting the following recommended guidelines and best practices when working with cinematics:

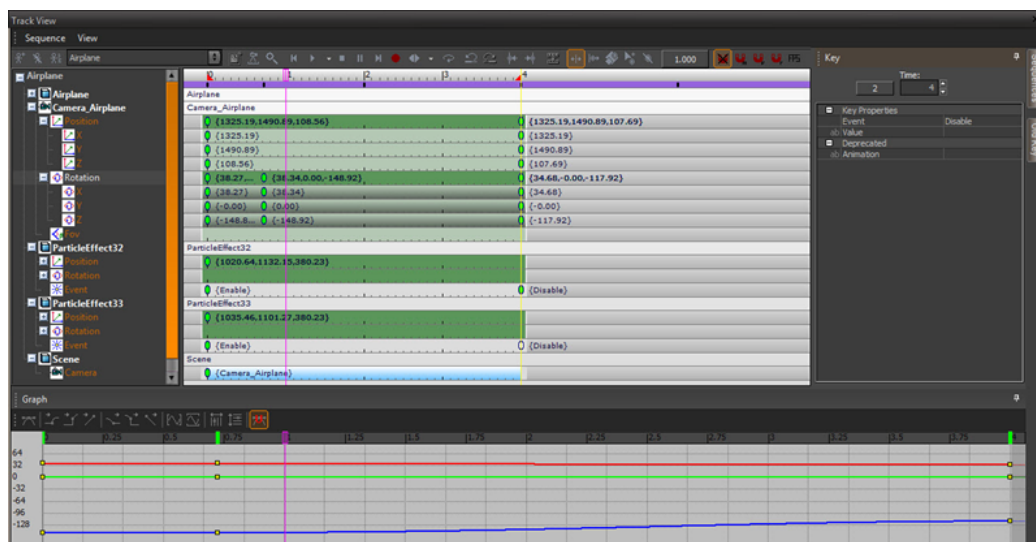
- Use **AnimObjects** for characters, vehicles, and other entities that are animated. In Rollup Bar, click **Entity, Physics\AnimObject**.
- Use **BasicEntity** for brushes and static entities that are simply updated with position or rotation movement, or are hidden and unhidden.
- To help with performance, whenever possible, disable the **Pushable by Player** and **Rigid Body** entity settings.
- Hide entities on game start. Do not use Flow Graph to hide or unhide entities.
- Disable sounds and particle effects on game start.
- Use camera depth of field (focus) whenever possible as it can hide background scene imperfections. Use lower levels of detail for better performance.
- Minimize the use of simultaneous multiple effects, full-screen image, or HUD effects. Make sure to disable them afterwards.
- Use animation precaching to avoid having characters appear in a T-pose when starting a scene in a game.
- Enable **Snapping** whenever possible.

Using Track View Editor

Track View Editor is the primary tool for creating and managing cinematic sequences. It is accessed from Lumberyard Editor by clicking **View, Open View Pane, Track View**. Track View Editor consists of the following components:

Track View Toolbars

- **Node browser** – Tree pane of all nodes and associated tracks.
- **Curve Editor** – Pane for controlling keys and their interpolation for all sequence entities.
- **Track Editor** – Track timeline of all sequence tracks. Each row in the timeline corresponds to a track listed in the accompanying node browser.
- All of the buttons in the Track View editor have descriptions of their use that are visible when you hover the mouse over them.



Using Cutscene Animation Curves (Curve Editor)

Select a key frame to see the associated tangent handles and then drag the boxes at the key frames or the ends of the tangent handles (including unify tangents and automatic tangents) to manipulate them. When moving key frames, hold down `Shift` to constrain the movement to time only, and `Alt` to scale the selected key frames around the play head location.

Track View Nodes

Track View Editor offers a variety of nodes for specific purposes. The top-level node in the tree view is the sequence - all other nodes are listed below the sequence.

Topics

- [Comment Node](#) (p. 281)
- [Console Variable Node](#) (p. 282)
- [Director \(Scene\) Node](#) (p. 282)
- [Entity Nodes](#) (p. 284)
- [Environment Node](#) (p. 286)
- [Event Node](#) (p. 286)
- [Material Node](#) (p. 287)
- [Script Variable Node](#) (p. 288)
- [Shadows Setup Node](#) (p. 288)
- [Full Screen Effect Nodes](#) (p. 289)

Comment Node

Use the **Comment** node to add comments to your track view sequence. This is mostly used for production purposes and is not rendered in the game.

To add a Comment node in Track View

1. In the **Track View** editor, right-click either the sequence (top node) or the **Director** node in the tree as applicable, and then click **Add Comment Node**.
2. For each of the keys listed below, click the applicable key listed under the **Comment** node.
3. To position a key, double-click the preferred location on its highlighted row in the timeline. Double-click the green marker, and then under **Key Properties** type a value for **Value**.

Comment Node Key Properties

Property	Description
Unit Pos X	position of the text horizontally
Unit Pos Y	position of the text vertically
Text	<ul style="list-style-type: none">• Comment – Text string• Duration – Length of time the node is active• Size – Font size• Color – Font color

Property	Description
	<ul style="list-style-type: none"> • Align – Text alignment (Center, Left, Right) • Font – Font type (default, console, hud)

Console Variable Node

Use the **Console Variable** node to use and animate console variables in a track view sequence.

To add a Console Variable node in Track View

1. In the **Track View** editor, right-click either the sequence (top node) or the **Director** node in the tree as applicable, and then click **Add Console Variable**. Type a name for it and click **OK**.
2. At the bottom of **Lumberyard Editor**, right-click the text box in the **Console** window, which opens up the **Console Variables** window that displays a list of all available console variables.
3. Pause on the desired console variable to get a tool tip that gives a description and valid values to use.
4. In the **Track View** editor select the **value** key listed under the console variable node.
5. To position a key, double-click the preferred location on its highlighted row in the timeline. Double-click the green marker, and then under **Key Properties** type a value for **Value**.

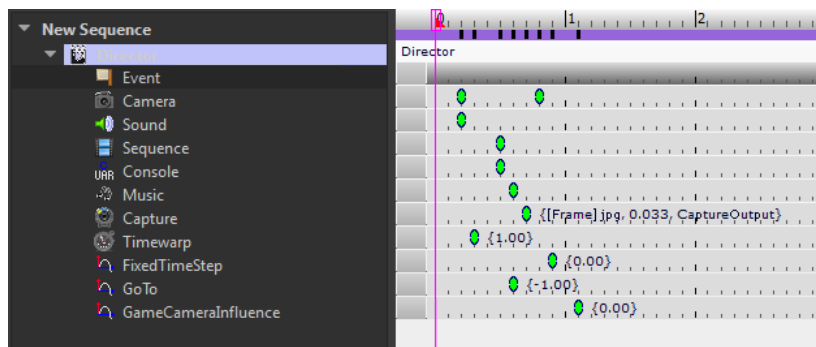
To animate a console variable

1. In the **Track View** editor click **View, Curve Editor**.
2. Click **Set In Tangent To Step** button (located third button from the left above the timeline window) to set the keyframes for the console variable.

Director (Scene) Node

To add a Director node in Track View

- In the **Track View** editor, right-click the sequence (top node), and then click **Add Director (Scene) Node**.



The Director (Scene) node contains a camera track that specifies which camera is active during a sequence. Additionally, sequence-specific nodes, such as a Depth of Field node or Comment node, can be added under Director nodes to optionally override any of the same nodes set at the sequence level.

You can add multiple Director nodes in a scene, but only one can be active at a time. To set a Director node to be active, right-click on the node and click **Set as Active Director**.

When a Director node is inactive, all child node animations are deactivated. This is useful for enabling and disabling animation for specific objects for the same shot for offline rendering.

The **Capture** track can be used to record frames to disk; however, often a more straightforward approach is to use the **Render Output** option under **Tools** in **Track View**.

You can add the following tracks to the Director (Scene) node:

Director Node Tracks and Key Properties

Track	Key Property	Description
Create Folder		Folders can be optionally used to organize Director tracks.
Camera	Camera	Specifies the sequence camera.
	Blend time	Number of seconds to use to blend between sequential cameras in the track
Capture	Duration	Image capture duration in seconds.
	Time Step	Used to set a fixed time step. Units are in 1 fps (frames per second), so a time step value of .0333 results in a game frame rate of 30 fps.
	Output Format	Specifies the image output file format.
	Output Prefix	Inserts a prefix in the image file name. For best results, use the same prefix as the sequence for clarity.
	Output Folder	Specifies the directory where the image output is stored.
	Buffer(s) to capture	Specifies the image capture format. The following are valid values: <ul style="list-style-type: none"> • Frames&misc – Outputs .tga and .hdr image files. • Just frame – Outputs normal images in the format specified. • Stereo – Captures stereo 3D images (one frame per eye).
Console	Command	Console command to execute
Event		Triggers events in the Director node LUA script.
FixedTimeStep		Sets a fixed time step in order to modify the game speed. Units are in 1 fps, so a <code>fixed_time_step</code> value of .0333 results in a game frame rate of 30 fps.
GoTo		Jumps forward or backward in time in a sequence. Used primarily for key framing time shifts and to turn parts of a sequence into a loop. This key automatically applies animation

Track	Key Property	Description
		blending on all currently playing animations in the sequence.
Music	Mood(T) or Volume(F)	The first Boolean specifies whether the key should change mood or volume according to the corresponding mood or time property of that key.
	Mood (if Mood)	Changes mood.
	Time (if Volume)	Changes time.
Sequence	Sequence	Sequence to play at the specified keyframe.
	Override Start/End Times	Check to override the sequence start and end times.
	Start Time	Specifies the start time to override.
	End Time	Specifies the end time to override.
Sound		Adds sound effects to a sequence.
Timewarp		Creates a slow-motion effect using a curve. This applies only to visuals; sounds are not slowed down.

Entity Nodes

Entity nodes are used to communicate between Track View and Lumberyard Editor. They are created by selecting the entity in Lumberyard Editor and using the Sequence or Director node .

To add an Entity node in Track View

1. In the Track View editor, right-click either the sequence (top node) or the **Director** node in the tree as applicable, and then click **Add Selected Entity(s)**.
2. For each of the tracks listed below, click on the applicable track listed under the **Entity** node, then double-click to position it on its highlighted row in the timeline, click the green marker, then under **Key Properties**, enter a value for **Value**.

Entity Node Tracks and Key Properties

Track	Key Property	Description
Animation	Animation	Opens the Animation Browser to select an animation to apply at this key.
	Loop	Sets whether to loop (repeat) the animation until the next key
	Blend Gap	When there is a gap in time between animation clips, this blends the end frame of the first clip to the beginning frame of the second clip. To use it, enable 'Blend Gap' for the first animation.
	Unload	Unloads the animation after the sequence is finished

Track	Key Property	Description
	In Place	If set, do not change the entity's base position and orientation
	Start Time	Sets the time, in seconds, within the clip for when to start playing the animation. 0 indicates the start of the saved clip. Start Time can never be greater than End Time.
	End Time	Sets the time, in seconds, within the clip for when to stop playing the animation. 0 indicates the end of the saved clip. Start Time can never be greater than End Time.
	Time Scale	Factor with which to scale time. Values larger than 1 will result in a sped up appearance, values smaller than 1 will result in a slow motion appearance.
Event	Event	A pull-down of all possible events supported by the Entity script
	Value	Sets the value to send with the Event
	No trigger in scrubbing	Disables sending of event triggers when scrubbing in Track View
LookAt	Entity	Entity to look at
	Target Smooth Time	Transition time, in seconds, over which to smooth the look rotational change
Mannequin	mannequin fragment	The mannequin fragment to play at the key frame
	fragment tags	Fragment tags to use with the fragment
	priority	TBD
Noise		Adds noise to the position and rotation of the entity if and only if Position and Rotation tracks respectively have keys in them
Physicalize		Track to enable and disable Physics simulation on an entity
PhysicsDriven		Sets the position and rotation to be driven by physics for non-static physics entities
Position		The X,Y,Z position of the entity
Procedural Eyes		This track is deprecated and will be removed in an upcoming release.
Rotation		The X, Y, Z Euler rotation angles of the entity
Scale		The X, Y, Z scale factors
Sound	StartTrigger	Audio Control for starting the sound. Refer to Audio System/Using the Audio Controls Editor for more information on Audio Controls.

Track	Key Property	Description
	StopTrigger	Audio Control for stopping the sound. Refer to Audio System/Using the Audio Controls Editor for more information on Audio Controls.
	Duration	Length of time to play the sound, in seconds
Visibility		Toggles visibility of the entity

Environment Node

You can use the **Environment** node to set the sun's longitude and latitude in a scene.

To add an Environment node in Track View

1. In the Track View editor, right-click either the sequence (top node) or the **Director** node in the tree as applicable, and then click **Add Environment Node**.
2. For each of the keys listed below, click the applicable key listed under the **Environment** node.
3. To position a key, double-click the preferred location on its highlighted row in the timeline. Double-click the green marker, and then under **Key Properties**, type a value for **Value**.

Environment Node Key Properties

Property	Description
Sun Longitude	Sets the sun's longitude.
Sun Latitude	Sets the sun's latitude.

Event Node

An Event node is used to trigger and send values to Flow Graph. It is used in tandem with a **TrackEvent** Flow Graph node. Track Events are created using the **Edit Events** window located in the context menu for a Sequence node or Director node. To trigger a Track Event, use an Event node and create a key frame. When this key is played, the event is triggered.

These Track View events will appear as Flow Graph node outputs on **TrackEvent** Flow Graph nodes that points to the corresponding sequence.

To add an Event node in Track View

1. In the Track View editor, right-click either the sequence (top node) or the **Director** node in the tree as applicable, and then click **Add Event**.
2. Click the **Track Event** track under the **Event** node, then double-click to position it on it's highlighted row in the timeline, click the green marker, then under **Key Properties**, enter a value for **Value**. To add Track View events, choose **Edit Events** in the Sequence or Director context menus to add, remove, or edit Track View events.

Material Node

Material nodes help you animate a number of commonly used material properties that you would normally set in the **Material Editor**. You can add **Material** nodes through a sequence or from the director node context menu.

The name of the **Material** node must be the full path of the material that you want to animate, as shown in the **Material Editor**. A recommended workflow is to use the **Mtl** button located in the **Entity** pane of **Rollup Bar** with the entity whose material you want to animate selected. This opens the **Material Editor** with the material selected.

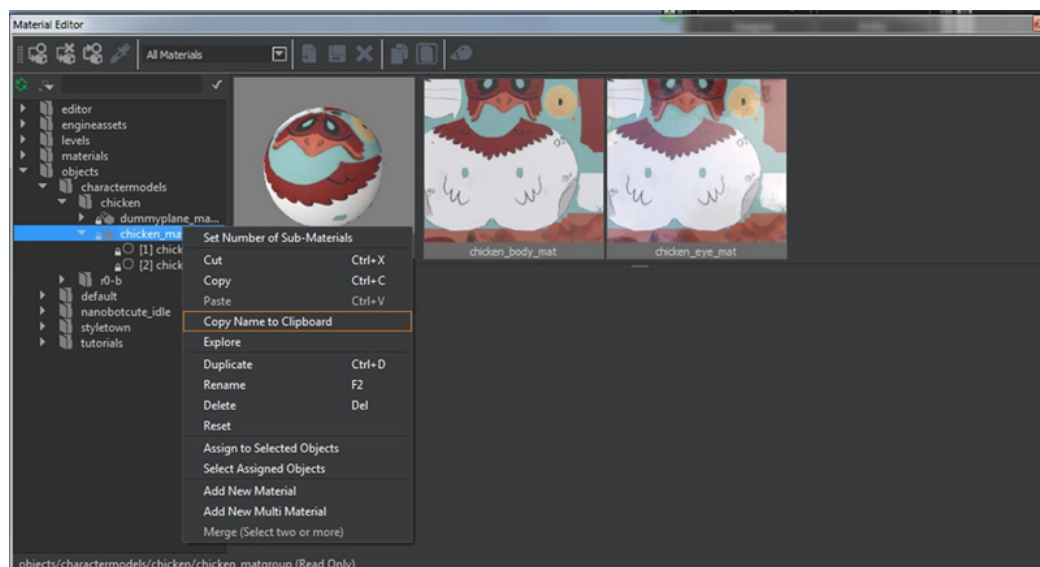
To add a Material node in Track View

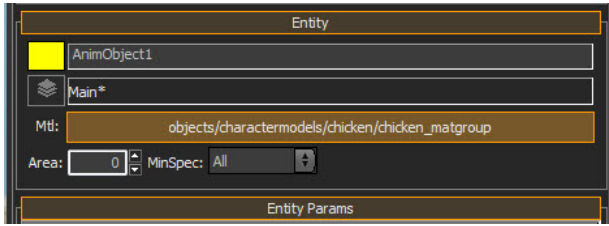
1. In Lumberyard Editor, select the the entity whose material you want to animate.
2. In the **Rollup Bar**, select the **Mtl** button in the **Entity** pane.
3. In the **Material Editor**, right-click the selected material and click **Copy Name to Clipboard**. If the material is in a material group, select the group and copy the group name to the clipboard for this step.
4. Click **View, Open View Pane, Track View**. In the **Track View** editor, use the sequence selector on the toolbar to choose the sequence or director that you to contain the animation. (Or click the **Add Sequence** icon to create a new one.)
5. Right-click in the node tree, and choose **Add Material Node**. When the dialog box opens, press **Ctrl+V** to paste the full path to the material that you copied in step 3.

If the material is in a material group, right-click the created material node in the **Track View** editor. At the bottom of the context menu you select the material that you want to animate.

The material node should appear up in non-colored text. If it shows up red it means that the **Track View** editor wasn't able to find a match for the material node name.

6. Right-click the node you added in the previous step and select **Add Track/Diffuse**, for example. Add two key frames with different colors, then scrub. Your material's diffuse color should be animated.





Material Node Tracks

Track	Key Property	Description
Diffuse		Type RGB values to specify the base color of a material.
Emission		Type RGB values to enables objects to emit light and be visible in the dark.
Glossiness		The acuity or sharpness of the specular reflection. Values of 10 or less create a scattered reflection, while values greater than 10 yield a sharp reflection.
Opacity		The degree of transparency. Values below 50 fall more to the white end of the alpha channel map. Values above 50 fall more to the black end of the alpha channel map.
SSSIndex		Controls subsurface scattering profile and amount. Valid value ranges are 0.01 to 0.99 for marble; 1.00 to 1.99 for skin.
Specular		The reflective brightness and color of a material when light shines on the object. The greater the value, the shinier the material. To apply reflections in degrees of black and white, make the R, G, and B values the same. For colored reflections, use varied RGB values.

Script Variable Node

Script Variable nodes create LUA variables using the name of the script variable, which can include '.' to specify variables within tables. Only floating-point variable values can be set.

Shadows Setup Node

You can use the **Shadows Setup** node to add or remove sun shadow maps over several frames in a sequence.

To add a Shadows Setup node in Track View

1. In the **Track View** editor, right-click either the sequence (top node) or the **Director** node in the tree as applicable, and then click **Add Shadows Setup Node**.
2. Click the **GSMCache** key under the **ShadowsSetup** node.
3. To position a key, double-click the preferred location on its highlighted row in the timeline. Double-click the green marker, and then under **Key Properties** type a value for **Value**.

Full Screen Effect Nodes

Full Screen Effect nodes create post-processing effects for a sequence. They are added by using the context menu for a Sequence or Director node.

Topics

- [Radial Blur Node \(p. 289\)](#)
- [Color Correction Node \(p. 289\)](#)
- [Adding a Depth of Field Node \(p. 290\)](#)
- [Screen Fader Node \(p. 290\)](#)

Radial Blur Node

You use the Radial Blur node to blur the animation radially outward from a center point.

To add a Radial Blur node in Track View

1. In the **Track View** editor, right-click either the sequence (top node) or the **Director** node in the tree as applicable, and then click **Add Radial Blur Node**.
2. For each of the keys in the following list, click the applicable key listed under the **RadialBlur** node. Then double-click the preferred location its highlighted row in the timeline. Double-click the green marker and then under **Key Properties** enter a value for **Value**.

Radial Blur Node Key Properties

Property	Description
Amount	Intensity of the blur effect. Range is 0 to 1.
ScreenPosX	X-axis position of the effect's center. Range is -1 to 1, with 0.5 being the center of the screen.
ScreenPosY	Y-axis position of the effect's center. Range is -1 to 1, with 0.5 being the center of the screen.
Blurring Radius	Size of the blur effect. Range is 0 (not visible) to 1 (covers the entire screen).

To make the blur intensity dynamically change based on a variable (such as the player's health for example), you can use the **Image:FilterRadialBlur** flow graph node.

Color Correction Node

You use the **Color Correction** node to change the CMYK, brightness, contrast, saturation, and hue in a scene. Most color correction properties don't update smoothly. For this reason, you should hide stronger color correction changes should by cuts or fading between scenes.

To add a Color Correction node in Track View

1. In the **Track View** editor, right-click either the sequence (top node) or the **Director** node in the tree as applicable, and then click **Add Color Correction Node**.
2. Click the applicable key listed under the **ColorCorrection** node.
3. To position a key, double-click the preferred location on its highlighted row in the timeline. Double-click the green marker, and then under **Key Properties** type an applicable value for **Value**.

To have change correction dynamically based on a variable, you can use the **Image:ColorCorrection** flow graph node.

Adding a Depth of Field Node

You can use the Depth of Field (DOF) node to add realism to scenes by simulating the way a real-world camera works. You can use a broad depth of field to focus on the entire scene, or use a shallow depth of field to have sharp focus only on objects that are a specific distance from the camera.

To add a Depth of Field node in Track View

1. In the **Track View** editor, right-click either the sequence (top node) or the **Director** node in the tree as applicable, and then click **Add Depth of Field Node**.
2. For each of the keys listed below, click the applicable key listed under the **DepthOfField** node.
3. To position a key, double-click the preferred location on its highlighted row in the timeline. Double-click the green marker, and then under **Key Properties** type a value for **Value**.

Depth Of Field Node Key Properties

Property	Description
Enable	Enables or disables depth of field effect
FocusDistance	Distance the focus is from the camera. Positive values are in front of the camera while negative values are behind the camera.
FocusRange	Distance toward and away from the camera until maximum blurriness is reached. By default, this value is twice the FocusDist value.
BlurAmount	Maximum blurriness value.

If you have a scene with full player control, setting depth of field using the **Image:EffectDepthOfField** flow graph node can be a good option. In addition, you can use the **Interpol:Float** node to smoothly fade the focus in and out. Use this sparingly as it can be difficult to track where and what the player is looking at.

Screen Fader Node

Use the Screen Fader node to fade the screen in and out in a scene.

To add a Screen Fader node in Track View

1. In the Track View editor, right-click either the sequence (top node) or the **Director** node in the tree as applicable, and then click **Add Screen Fader**.
2. Click the **Fader** key under the **ScreenFader** node
3. To position a key, double-click the preferred location on its highlighted row in the timeline. Double-click the green marker, and then under **Key Properties**, enter a value for the following properties:

Screen Fader Node Key Properties

Property	Description
Type	Select either FadeIn or FadeOut values.

Property	Description
ChangeType	For this transition type select from Cubic Square , Linear , Sinus Curve , Square , or Square Root .
Color	Specify the RGB value used for fading.
Duration	Specify how long it takes to fade in or out the screen.
Texture	Specify a texture file to use as a screen overlay. An alpha texture is commonly used for effects like dirt or blood.
Use Current Color	Select to ignore the Color property and use the color of the previous key instead.

You can also create a screen fader effect by using the **Image:ScreenFader** flow graph node.

Creating Scenes

Cinematic scenes, also known as sequences, consist of multiple nodes, tracks, and track events.

You create a sequence in Track View Editor by clicking **Sequence**, **New Sequence**, and naming it. A sequence is always the top (parent) node in the tree view.

Topics

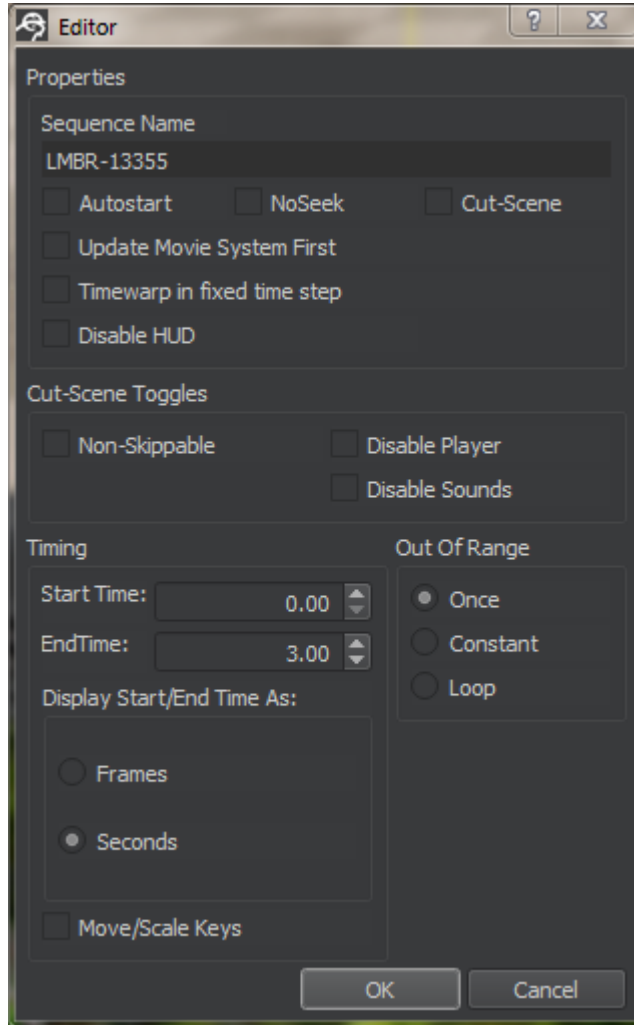
- [Setting Sequence Properties \(p. 291\)](#)
- [Playing a Sequence \(p. 294\)](#)
- [Changing Playback Speed \(p. 295\)](#)

Setting Sequence Properties

You can set various sequence properties in the Track View editor as follows:

To set sequence properties

1. In the Track View editor, select the applicable sequence and click the **Edit Sequence** button, which is the third button in the **Sequence/Node** toolbar row.
2. In the **Sequence Properties** dialog box, set properties as shown and listed in the following image:



Scene Properties

Property	Description	When to use
Autostart	Plays the scene on game start.	Use for testing purposes only. For scenes that must always play on game start, use triggers instead.
NoSeek	Disables random seeks in a scene, such as jumping to a certain time.	
CutScene	Used to enable various scene toggles. When selected, the following options are available: <ul style="list-style-type: none"> • Disable Player – disables the player (required for all camera-controlled scenes) • Non-Skippable – prevents the player from skipping the scene for important events 	Required for all scenes that are camera-controlled.

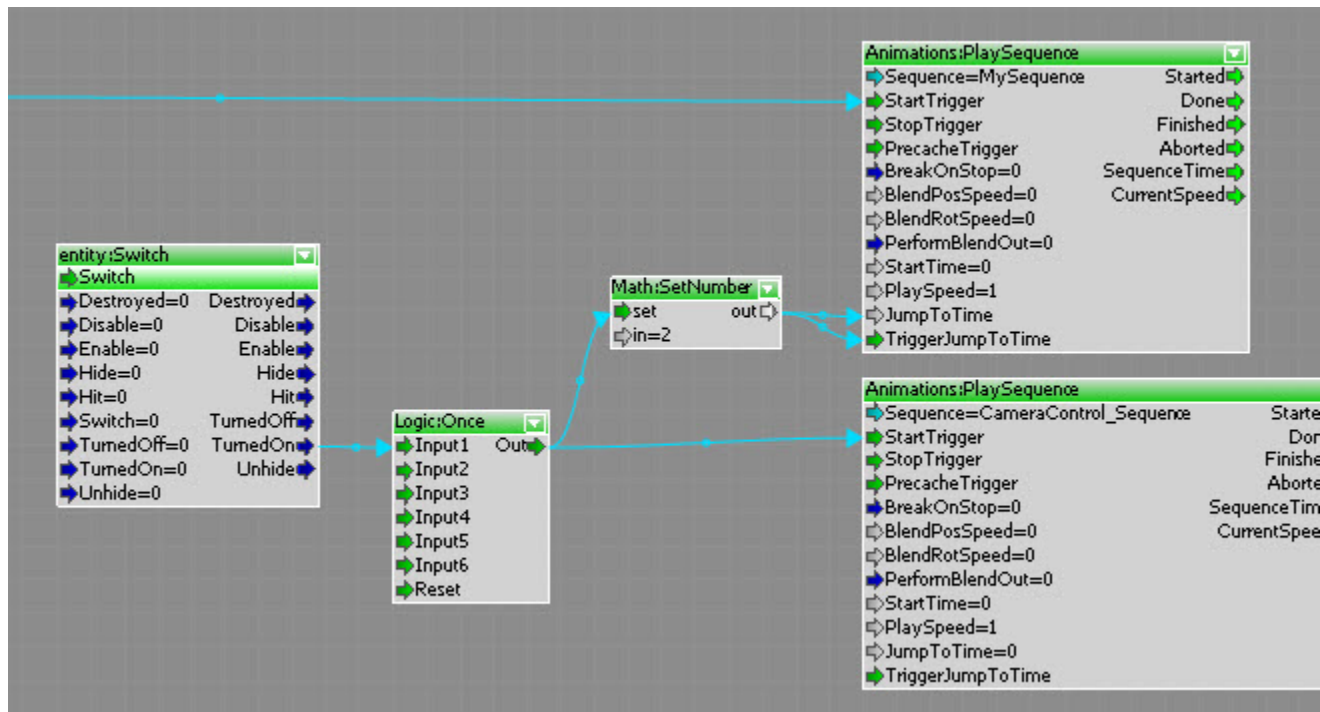
Property	Description	When to use
	<ul style="list-style-type: none"> • Disable Sound – disables all sounds not in the scene 	
Update Movie System First	Typically, the movie system updates before the entity system updates. This reverses that order.	Used to fix bone-attached entities that lag behind the parent movement. This problem typically occurs if the parent locator position is animated in Track View.
Timewarp in fixed time step	Modifies the fixed time step value instead of the time scale value.	Used for capturing scenes in fixed time step that use at least one Timewarp so that Timewarp is correctly captured (fixed time step overrules the time scale value).
Disable HUD	Disables the HUD	Only if the HUD isn't supposed to be shown at all.
Timing	Sets the start and end times of the sequence	
Display Start/End Time As:	Displays the start and end times in this dialog in frames or seconds. This is a display-only option; the times are always stored in seconds	
Move/Scale Keys	When set on, animation curves are scaled over the time line when the Start or End times are changed.	When you want to lengthen or shorten a sequence and have the animations slow-down or speed-up to fill the same relative percentage of the sequence time line
Out of Range	<p>Changes the movie time behavior when it passes the end of the sequence:</p> <p>Once - movie time continues past the end of the sequence</p> <p>Constant - move time is held at the end of the end of the sequence</p> <p>Loop - movie time loops back to the beginning time of the sequence when it reaches the end</p>	

Changing Scene Toggles Mid-Sequence

The **Cut-Scene Toggles** properties listed above can be changed mid-sequence by starting another sequence that runs in parallel. This is primarily used to briefly turn on camera control in a sequence that allows free player movement.

For example, the main sequence allows free player movement and enters a loop at second 1. A Flow Graph **Entity:Switch** node (shown in the following image) makes the sequence jump to second 2, which starts a short camera-controlled second sequence.

In this example, all the main sequence properties would be disabled (unselected), while the second sequence would have the **Cut Scene**, **Disable Player**, and the **Non-Skippable** properties enabled (selected).



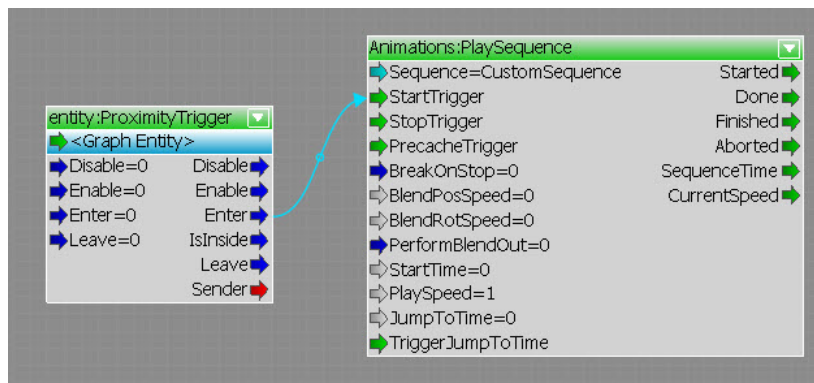
Playing a Sequence

The easiest way to play Track View sequence is to attach it to a Flow Graph proximity trigger that can be positioned in the level. To access Flow Graph from Track View, trigger entities are used to send events to Flow Graph where various nodes are then triggered. When a track event is triggered from the scene, its corresponding output in a Flow Graph node is activated.

Note

To use the default game camera in a sequence, add a keyframe under the **Director** node on the **Camera** track, leaving the camera **Key** property blank. Using this as the last keyframe on the **Director** node **Camera** track in your sequence transitions the last used sequence camera to the default game camera when the keyframe is played.

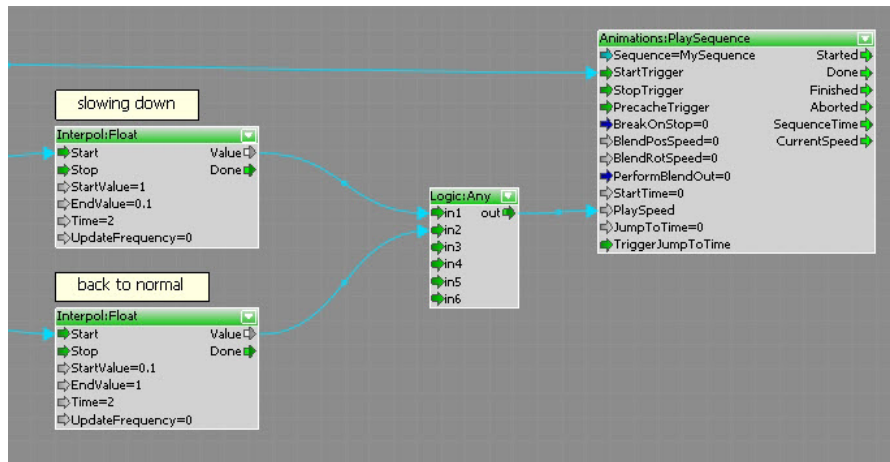
Specifically, the **entity:ProximityTrigger** node output is connected to the **StartTrigger** input of the **Animations:PlaySequence** node. When a player enters the trigger in the game, the sequence starts.



Changing Playback Speed

Using the **Animations:PlaySequence** Flow Graph node, you can control the playback speed of the sequence by simply changing the value of the **PlaySpeed** input.

If you want a fixed slow-motion or fast-forward effect instead, use the **TimeWarp** track of the Director (Scene) Node.



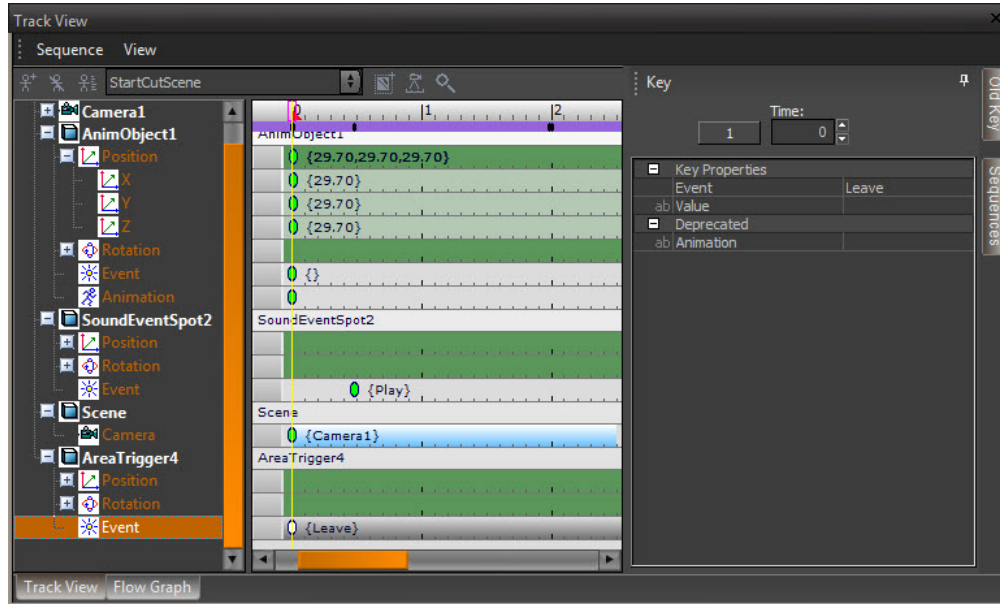
Managing Track Events

A track event is a trigger that allows you to integrate Flow Graph logic with a Track View scene. When a track event is triggered from the scene, its corresponding output in a Flow Graph node is activated. A scene can contain a number of track events that are grouped under a Track Event Node. Each track event can have multiple keys assigned to it.

Track events can also be used to change the time of day in terrain level.

To add a track event

1. In the **Track View** editor, right-click the applicable scene. Click **Edit Events**.
2. Click **Add**, and then enter a name. Close the dialog.
3. Under the track event node, click the **Track Event** track, then double-click to place a key in the timeline row adjacent to it.
4. In **Key Properties/Value**, enter a value.

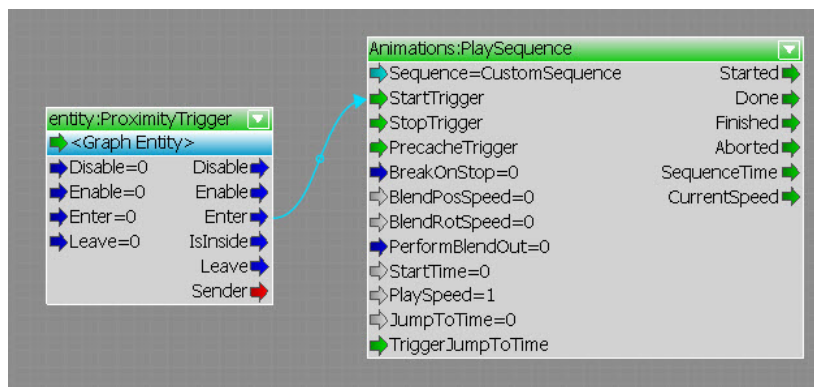


Linking Track View Events with Flow Graph

The Track Events you create in Track View can be used in Flow Graph by adding a Track Event node in Flow Graph and setting its **Sequence** property to the Track View sequence triggering the event. The Track Event Flow Graph node has outputs for each event in that sequence.

Certain features required for creating cinematic effects are available only in Flow Graph. To access these features, you need a link between Track View and Flow Graph. Specifically, Track View trigger entities are used to send events to Flow Graph where various nodes are then triggered.

Specifically, the **entity:ProximityTrigger** node output is connected to the **StartTrigger** input of the **Animations:PlaySequence** node. When a player enters the trigger in the game, the sequence starts.



Cinematics Cameras

Cameras present scenes from particular points of view. Cameras are added using the Rollup Bar (on the **Objects** tab under **Misc, Camera**) in Lumberyard Editor.

Topics

- [Moving a Camera \(p. 297\)](#)
- [Setting Camera Focus \(p. 297\)](#)
- [Creating Camera Shake \(p. 298\)](#)
- [Blending a Camera \(p. 299\)](#)
- [Pointing a Camera \(p. 301\)](#)
- [Following with a Camera \(p. 301\)](#)
- [Setting a First Person View Camera \(p. 302\)](#)
- [Importing a Camera from Autodesk \(p. 302\)](#)
- [Exporting a Camera to Autodesk \(p. 303\)](#)

Moving a Camera

To move, rotate, or animate a camera in Track View, use the Viewport Camera controls.

To move a camera view

1. Select the applicable camera in the viewport. In Track View editor, right-click the applicable sequence and click **Add Selected Entity(s)**.
2. Click the red Record button.
3. In the viewport, right-click the camera and uncheck **Lock Camera Movement**.
4. In the viewport, right-click the camera and select your camera under **Camera**.
5. In **Track View**, with the **Camera Node** selected, press the Record button.
6. Navigate the viewport using the mouse and keyboard. Notice the values being recording as key frames in **Track View**.

Setting Camera Focus

Camera focus, or depth of field (DoF), is used to add realism to scenes by simulating the way a real-world camera works. You can use a broad depth of field to focus on the entire scene, or use a shallow depth of field to have sharp focus only on objects that are a specific distance from the camera.

Here are some guidelines and best practices when setting up camera focus:

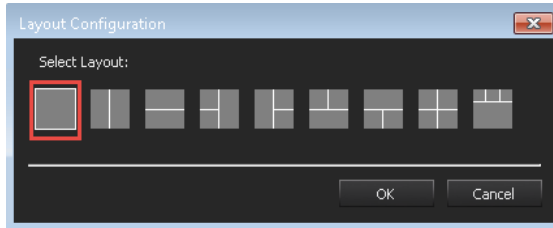
- Always keep characters in focus.
- Shift focus slowly and deliberately.
- Don't overdo it.
- Do not use depth of field for scenes that are far away. Rather, it works best for differentiating between closeups and the background.
- Use your eyes to focus at different distances and see what is sharp and what is blurred (use your thumb as a helper). This should give you a sense of how it should look in a scene.

DoF is rendered only for a single view pane layout (the default) in the viewport in Lumberyard Editor. If you are using a multiple view pane layout and the sequence camera is not in the active pane, DoF does not render. If you need to set this, complete the following procedure.

To set the view port for a single view pane layout

1. In Lumberyard Editor, right-click the **Perspective** title bar in the viewport, then click **Configure Layout...**

2. In the **Layout Configuration** dialog box, click the single view pane (the left-most option), then click **OK**.
3. Right-click the **Perspective** title bar again, then click **Sequence Camera**.



To add a Depth of Field node

- In the Track View editor, right-click the **Director** node or any camera node, and then click **Add Depth of Field Node**.

Camera nodes take precedence over the Director node. Use the Director DoF node if you want the same DoF setup for multiple cameras. Most of the time, however, you want separate, specific DoF setups for each camera for more control.

You can add as many keys as you want, and use the curve editor to further tweak DoF settings to change over time.

Creating Camera Shake

Most moving cameras in the real-world have some degree of shake. You can add shake to your cameras for more realism.

Unlike the amplitude parameter in the ViewShakeEx Flow Graph node, camera shake involves separate overlapping and accumulating values and multipliers of amplitude and frequency parameters in both the Rollup Bar and the Track View editor to achieve the final effect.

The following guidelines can be followed to achieve realistic camera shake effects:

- Keep shaking restrained, don't overdo it.
- Vary the amplitude and frequency values.
- Edit curve and key values appropriately.
- Try to mimic the corresponding effect in the real-world for what is happening in the scene.

You can adjust the following static parameters in Rollup Bar (under **Camera Params**) for a camera entity for the desired effect. These parameters are the primary, non-animating parameters which you can further tweak in the Track View editor.

Camera Shake Parameters

Parameter	Description
Amplitude A	Intensity of the camera shake
Amplitude A Mult.	Multiplier for Amplitude A value
Frequency A	How rapidly the camera changes orientation

Parameter	Description
Frequency A Mult.	Multiplier for Frequency A value
Noise A Amp. Mult.	Multiplier for Noise A Amp value
Noise A Freq. Mult.	Multiplier for Noise A Freq value
Time Offset A	Delay time for camera shake
Amplitude B	Intensity of the camera shake
Amplitude B Mult.	Multiplier for Amplitude B value
Frequency B	How rapidly the camera changes orientation
Frequency B Mult.	Multiplier for Frequency B value
Noise B Amp. Mult.	Multiplier for Noise B Amp value
Noise B Freq. Mult.	Multiplier for Noise B Freq value
Time Offset B	Delay time for camera shake

To achieve realistic camera shake, it is important to edit the fCurves using the Curves editor in Track View. When you add a shake keyframe, the default fCurve values have wide tangents which cause extreme easing in and out time values. However, most of the time, the goal is to have an immediate shake effect, such as for punches or explosions. In this case, the curve must be edited to have very rapid build up time, as shown below.

Blending a Camera

You can blend a camera in and out of a camera-controlled sequence.

Note

To use the default game camera in a sequence, add a keyframe under the **Director** node on the **Camera** track, leaving the camera **Key** property blank. Using this as the last keyframe on the **Director** node **Camera** track in your sequence transitions the last used sequence camera to the default game camera when the keyframe is played.

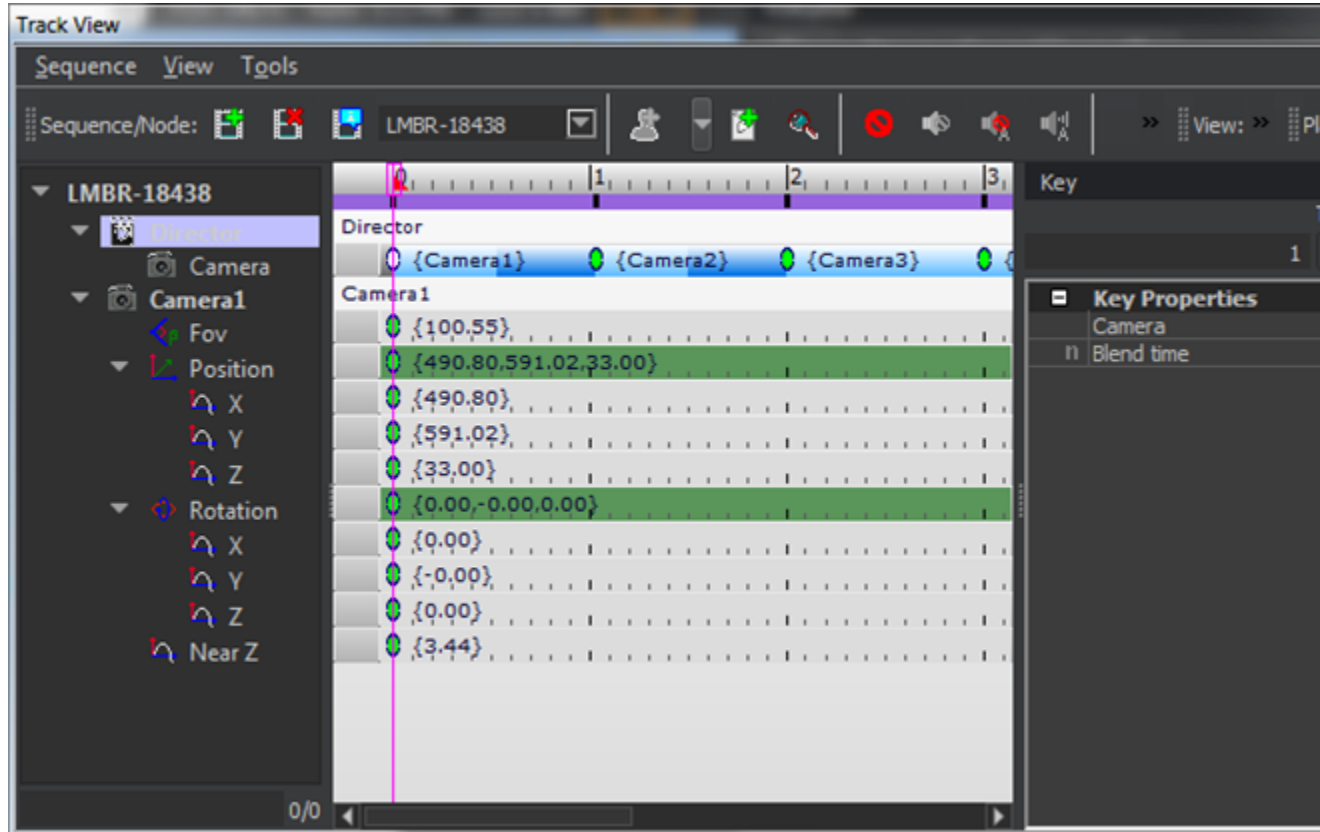
Blending within a sequence

To create a series of jump cuts from one camera to another in a sequence, place key frames on the **Director** node **Camera** track. The **Camera** key property for each key frame is what the **Director** node uses to determine which camera to switch to at that time.

Blended camera key frames will blend the position, rotation, field of view, and near-Z properties of the current camera into the next camera on a **Director** node **Camera** track, which will make the cut appear as a continuous single camera motion rather than an abrupt jump cut.

To create a blended camera key, select the key frame on the first camera of the blend and set the **Blend time** key property to a value greater than zero. This is the time in seconds over which the blend will occur.

The following figure shows an example for **Camera1**.



Note

If you have added the first camera as a **Track View** node in this example for **Camera1**, you must set at least one key frame for **Position** and **Rotation** when using blended sequence cameras.

Blending into a sequence

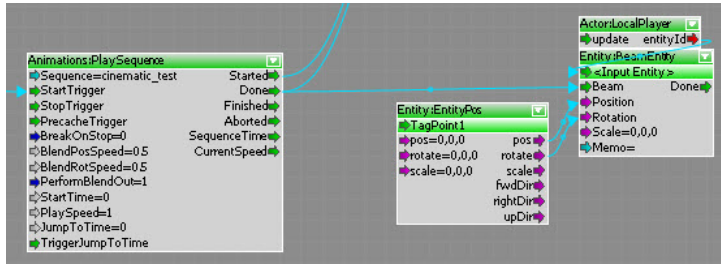
When blending into a sequence using Flow Graph, place the camera inside of a large trigger that encloses the entire sequence, otherwise snapping occurs when starting the sequence. For best results, in the Flow Graph **Animation:Play Sequence** node used to start the sequence, use values between 0.5 and 2.0 for **BlendPosSpeed** and **BlenRotSpeed**.

Using the **Animations:Play Sequence** Flow Graph node, slow player motions down so that jumps, sprints, and slides transition more smoothly.

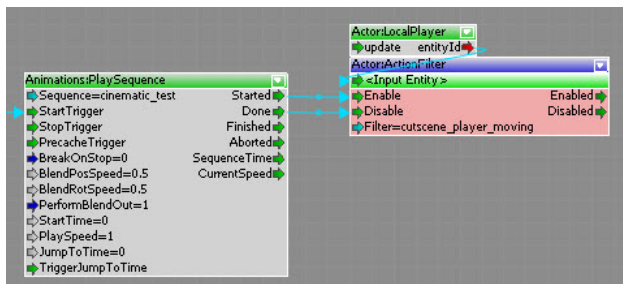
Blending out of a sequence (for first-person games)

When blending a camera out of a sequence, use the **Entity:BeamEntity** Flow Graph node to set the player to the end of the sequence. Position the **Entity:Entity Pos** TagPoint right below or slightly behind the last camera position for a good transition. Be sure to connect either the **Done** or **Finished** outputs for the **Animations:PlaySequence** Flow Graph node directly to the **Beam** input of the **Entity:BeamEntity**.

Do not use game tokens or other logic in between transitions, otherwise a previous player position may be visible for a few frames before beaming to the final position.



Sometimes, the player's last movement input is stored and remains active after the sequence, causing the player to continue to walk though no key is pressed. To prevent this, use the **Actor:ActionFilter** Flow Graph node and enable the **Filter=cutscene_player_moving** input at the start of the sequence and disable it at the end of the sequence.



Transitioning to the Active Game Camera in Track View

In the Track View Editor, the sequence camera is set by adding a **Director** node and adding keys to the **Camera** track. If a **Camera** track is left blank, the sequence camera uses the active game camera.

Pointing a Camera

You can have a camera always point at a selected target in the viewport in Lumberyard Editor. The camera target can be created (designated) only when placing a new camera in the viewport.

1. In Lumberyard Editor, in the Rollup Bar, click **Misc, Camera**.
2. Click in the viewport where you want the camera located, continuing pressing the left mouse button, and then release the mouse button where you want the camera target located.

The camera now always points at the camera target, which can be animated in the Track View editor. As such, the camera's rotation can no longer be modified independently.

Following with a Camera

You can have a camera follow an object and also rotate around (orbit) the object by first linking the camera to a TagPoint entity and then linking the TagPoint to the object. The TagPoint acts as a pivot and, by animating its rotation in Track View, the camera now rotates around the object.

However, if the object itself is rotating, it can cause unwanted effects on the camera. You can minimize this issue by adjusting the TagPoint pivot rotation. For example, if the object has an X-axis rotation of -15° , set the pivot's X-axis rotation to $+15^\circ$ to cancel out the values.

Another way to minimize this issue is to make the TagPoint's rotation independent of the object you want to follow. To do this, use one "root" TagPoint for animating the position, then link your followed object to it and animate only its rotation. Next, link the pivot TagPoint and the attached camera to

the root TagPoint as well. Using this method, you can rotate your target and the camera orbit pivot independently of each other.

Setting a First Person View Camera

There are several ways of setting up a first-person view (FPV) camera:

- Link the camera to the character
- Link the camera to the character's camera bone
- Link the camera to a TagPoint

Linking a camera to a character

Linking a camera to a character works well for rough blocking, where the character has no animation applied in Track View. By attaching the camera to the character and positioning it close to the camera bone (at eye level), you can test it from the character's point of view and the pace of the scene.

Linking a camera to a character's camera bone

You can attach a camera to the character's camera bone so that the camera follows the camera bone of the character. This method is good for referencing, but not for the final process, as the information from the camera bone can be very rigid and often clips through the character's body, especially if animation is derived from motion capture. It is also impossible to manipulate the camera this way.

Linking a camera to a character and a tagpoint

The best way to set up an FPV camera is to attach it to both the character and a tagpoint. This method allows the camera and the character to be animated independently. The tagpoint acts as an anchor that connects the character and camera together, which makes it easy to move the character around after you have finished adjusting the scene.

By using a second camera that links to the character's camera bone, you can easily adjust and match your main camera to the second (referencing) camera to get the right movement. This camera tracks the character's head movement.

To set up an FPV camera

1. Link the main camera to a tagpoint.
2. Link the second camera to the character's camera bone.
3. Position the second camera to 0, 0, 0.
4. Assign the main camera in the track view.

Importing a Camera from Autodesk

Cinematic camera transformations can be imported to Lumberyard from Autodesk Maya or 3ds Max.

Topics

- [Importing a Camera from Maya \(p. 302\)](#)
- [Importing a Camera from 3ds Max \(p. 303\)](#)

Importing a Camera from Maya

Use the following process when importing a camera from Autodesk Maya to Lumberyard.

To import a camera from Maya

1. In Maya, click **Window, Settings/Preferences, Preferences**. In the **Settings** dialog box, click **Settings**. For **Time**, select **NTSC (30 fps)**.
2. In Maya, in **Film Back settings**, change **Camera Aperture** to 0.581 0.327 and **Film Aspect Ratio** to 1.78.
3. Select the camera you wish to export. It must have the same name as the Lumberyard camera to which you want to import the camera's animation.
4. Click **File, Export Selection**.
5. In **Select File to Export**, select the **FBX** format.
6. In **FBX Export, Advanced Options**, for **Up Axis**, select **Z**.
7. Set **Scale Factor** to **1,0**.
8. Save the `.fbx` file to a suitable location.
9. In Track View editor, right-click a camera node in the applicable sequence and then click **Import FBX File**.
10. Browse to the `.fbx` file and click **Open**.
11. In the **FBX Import – Select Nodes to Import** dialog box, select the name of the camera you exported in Step 3, which should match the name of the Lumberyard camera in step 9.

Importing a Camera from 3ds Max

Use the following process when importing camera transformations and Field Of View (FOV) from Autodesk 3ds Max to Lumberyard.

To import a camera from 3ds Max

1. In 3ds Max, click **Customize, Units Setup**. In the dialog box, under **Display Unit Scale**, select **Metric, Meters**.
2. Select the camera you wish to export. It must have the same name as the Lumberyard camera to which you want to import the camera's animation.
3. Click the **MAX** toolbar icon at the upper left, then click **Export, Export Selected**.
4. In **Select File to Export**, select **Autodesk (*.FBX)** in **Save as type**, then enter a file name.
5. In **FBX Export, Advanced Options**, for **Axis Conversion, Up Axis**, select **Z-up** and select **Units, Automatic**.
6. In Track View editor, right-click a camera node in the applicable sequence and then click **Import FBX File**.
7. Browse to the `.fbx` file and click **Open**.
8. In the **FBX Import – Select Nodes to Import** dialog box, select the name of the camera you exported in Step 3, which should match the name of the Lumberyard camera in step 9.

Exporting a Camera to Autodesk

Cinematic camera transformations can be exported to Autodesk Maya or 3ds Max from Lumberyard.

Topics

- [Exporting a Camera to Maya \(p. 304\)](#)
- [Exporting a Camera to 3ds Max \(p. 304\)](#)

Exporting a Camera to Maya

Use the following process when exporting a camera from Lumberyard to Autodesk Maya. Transformation tracks and animated FOV data are supported for export.

Upon export, cameras are re-oriented to fit the Maya standard of pointing down in the Z-axis as opposed to the Lumberyard standard of cameras pointing in the Y-axis.

To export a camera to Maya

1. In the Track View editor, right-click a camera node and then click **Export FBX File**.
2. Select a file path, and then set **Save as type to FBX (*.fbx)**.
3. In **FBX Export Settings**, ensure **Convert Cameras/Axes for Max/Maya** is selected. The remaining parameters are all optional.
4. In Maya, click **File, Import** toolbar icon, click **Import, Import**, then select the file you exported in step 3 for import.

Exporting a Camera to 3ds Max

Use the following process when exporting a camera from Lumberyard to Autodesk 3ds Max. Transformation tracks and animated FOV data are supported for export.

Upon export, cameras are re-oriented to fit the 3ds Max standard of pointing down in the Z-axis as opposed to the Lumberyard standard of cameras pointing in the Y-axis.

To export a camera to 3ds Max

1. In the Track View editor, right-click a camera node, then click **Export FBX File**.
2. Select a file path, and then set **Save as type to FBX (*.fbx)**.
3. In **FBX Export Settings**, ensure **Convert Cameras/Axes for Max/Maya** is selected. The remaining parameters are all optional.
4. In 3ds Max, click the **MAX** toolbar icon, click **Import, Import**, then select the file you exported in step 3 for import.

Cinematics Lighting

Creating lighting for cinematic scenes involves a different process than that used for creating environment lighting for a level.

Animating a Light

Light entities are animated in the Track View Editor by creating a light animation set, which is a sequence containing **Light Animation** nodes. A Light entity then references these nodes with the **Light Animation** property in the Rollup Bar.

To create a new light animation set

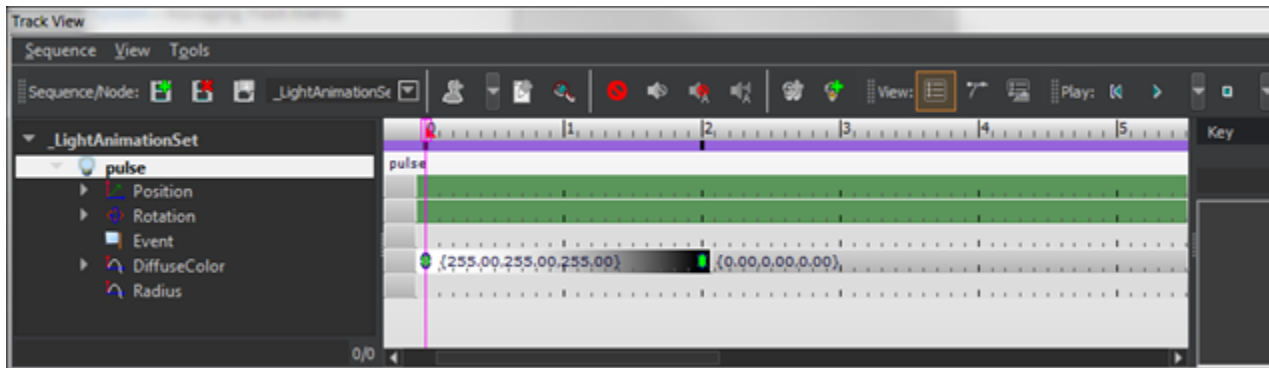
1. In Track View Editor, click the **Create Light Animation Set** button as shown below. This only needs to be done once per level.



2. In the left pane, select **_LightAnimationSet**, click the **Add Light Animation Node** button, then name the node **Pulse**.



3. Under **DiffuseColor**, add two key frames as the animation.



4. In Rollup Bar, create a light entity by clicking **Entity, Lights**, then double-click **Light**.
5. In the **Entity Properties** panel, under **Style**, select **LightAnimation**, then click the ... button to access the **Select Light Animation** dialog.
6. Select the **Pulse** node and select **OK**.

Your light entity will play the animation in the **LightAnimationSet\pulse** node in a loop.

Cinematic Lighting Best Practices

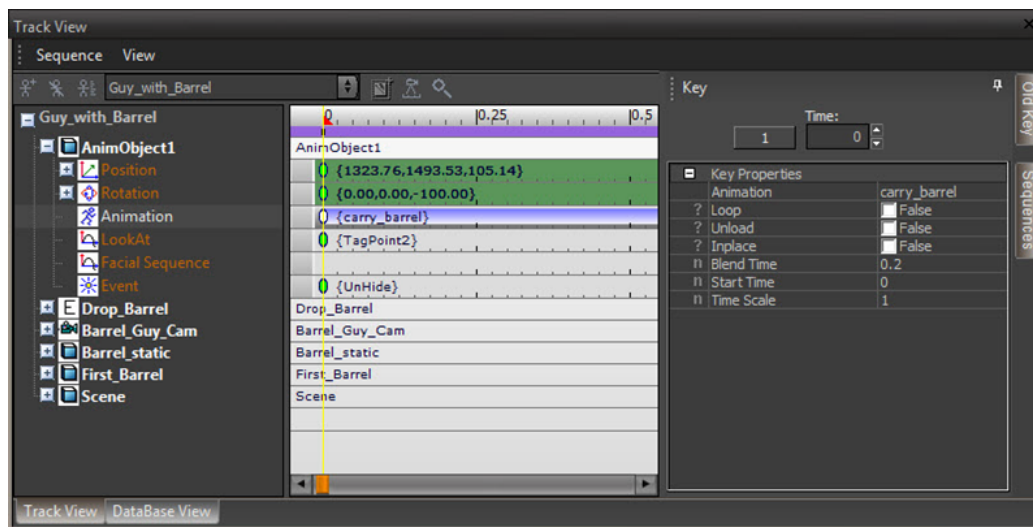
The following represents recommended guidelines and best practices for cinematic lighting.

- Cinematics lighting should be on its own layer.
- Enable lights for a shot, then disable them when the camera cuts.
- Disable gameplay and cubemap lights as needed for shots to avoid interference.
- For pre-rendered cinematic scenes, use the console variable `e_timeofday` to trigger the correct time of day.
- For real-time cinematics, use a Track Event node to trigger the correct time of day.
- For pre-rendered cinematic scenes, use `ShadowsSetup` to enable High Quality shadows mode.
- For pre-rendered cinematic scenes, because performance isn't an issue, you should always enable shadow casting and use as many spotlights as needed. Projector textures should be used as much as possible for spotlights. The `SpecularMultiplier` value should always be 1.
- Shadowmap quality from point lights is greatly improved when the `ProjectorFOV` value is as low as possible. To soften shadows, you can increase the `ProjectorFOV` value slightly, but this also decreases the accuracy of the shadowmap.
- Don't use ambient lights as they can weaken contrast and illuminate unwanted areas. Instead, use cubemaps to make the deepest shadow as dark as possible, and then add lights to increase the overall illumination.
- Lights should be turned on and off while in the Track View editor. If lights are off by default, they won't accidentally render in-game or interfere with a scene shot. When editing a light, keep the Active track flag enabled. Once done, disable the flag. Add keyframes on the Active track to ensure that the light is shown only when needed.

Animating Characters in Scenes

Character .CGF and geometry .CDF assets can be added to Track View sequences for animation and interactions by using the **AnimObject** entity.

In the case of a static asset, the **BasicEntity** entity is used instead.



Topics

- [Importing and Exporting Transform Animations \(p. 306\)](#)
- [Adding Geometry to a Sequence \(p. 307\)](#)
- [Animated Character Tracks in Cutscenes \(p. 308\)](#)
- [Moving an Entity in a Scene \(p. 308\)](#)
- [Adding Scene Character Attachments \(p. 309\)](#)
- [Using Look IK in Scenes \(p. 309\)](#)
- [Blending Cinematic Animations \(p. 311\)](#)
- [Using Track View Animation Curves \(p. 311\)](#)
- [Pre-caching Cinematic Animations \(p. 312\)](#)

Importing and Exporting Transform Animations

Lumberyard supports the import and export of translation and rotation transform animations between Track View and DCC tools that support FBX file export and import, such as 3ds Max and Maya for example.

Importing Transform Animations to Track View

FBX translation and rotation transform animations can be imported from any DCC tool that supports the export of FBX animations. Such animations can then be imported and applied to entities in Lumberyard.

To import transform animations to Track View

1. In your DCC tool, ensure the nodes containing the animation you wish to export are top-level nodes, and are named exactly the same as the Track View nodes to which you would like to import and apply the animation to.

2. In your DCC tool, export the node animations to an FBX file format. Ensure that the **Animation** option is enabled in the export settings. Also ensure that the FBX option for the **Up Axis** setting matches that of your DCC scene. For example, in Maya, if your scene's **World Coordinate System** value is set to **Y**, then the **Up Axis** setting value should also be set to **Y**.

To check this setting, click **Windows, Settings, Preferences**. In the **Preferences** window, under **Categories**, click **Settings**. Under **World Coordinate System**, check the **Up Axis** setting to ensure it is set to **Y**.

3. In Track View, right-click the node that will receive the animation import and choose **Import FBX File**.
4. Browse to the FBX file saved in step 2 and click **Open**.
5. Under **FBX Import, Select Nodes to Import**, select the node that you exported in step 2, which should match the name of the node selected in step 3, then click **OK**.

Exporting Track View Transform Animations

Track View translation and rotation transform animations can be exported from Lumberyard to any DCC tool that supports the import of FBX animations.

To export transform animations from Track View

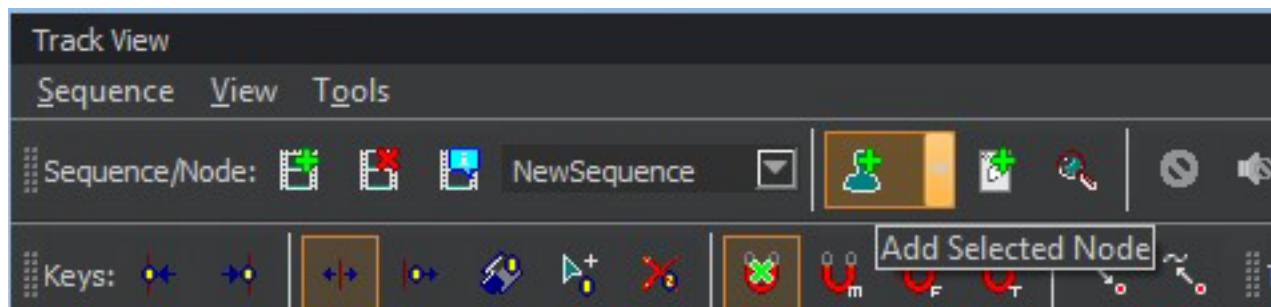
1. In Track View, right-click the node with the transform animation you wish to export and choose **Export FBX File**.
2. Choose a file name and click **Save** in the **Export Selected Nodes To FBX File**.
3. In your DCC tool, create a top-level node and name it exactly the same as the node selected in step 1.
4. In your DCC tool, import the FBX file, ensuring that the **Animation** option is enabled in the import settings.

Adding Geometry to a Sequence

In order to import an asset into a cinematic sequence, first add an AnimObject into the sequence.

To add geometry to a sequence

1. In Lumberyard Editor, in the Rollup Bar, on the **Objects** tab, click **Entity, Physics**, and then double-click **AnimObject**.
2. Under **Entity Properties**, click **Model** and then click the folder icon.
3. In the **Preview** dialog box, select the applicable asset, and then click to place it in the viewport where desired.
4. In the Rollup Bar, under **Entity Properties**, do the following:
 - Select **AlwaysUpdate** under **Animation**
 - Unselect (disable) **RigidBody** under **Physics**
 - Unselect (disable) **PushableByPlayers** under **Physics**
5. In Track View Editor, create a new sequence by clicking the **Add Sequence** button, or by clicking **Sequence, New Sequence**. With the AnimObject selected in the main editor, add the entity to Track View Editor using the **Add Selected Node** button.



6. With the **AnimObject** selected in Rollup Bar, add the entity to Track View Editor by clicking the **Add Selected Node** button.

Animated Character Tracks in Cutscenes

The AnimObject entity is used to animate characters and other objects in cinematic scenes. The Track View Editor has a number of tracks that can be set to customize and fine-tune character animation.

To add AnimObject tracks

1. In the Track View editor, right-click the applicable **AnimObject** node, click **Add Track**, then click a track.
2. Select the track in the tree pane, then double-click in the timeline window to place a key.
3. Click the green marker, then under **Key Properties**, adjust the values of the track key properties.

Animated Character Properties in Cutscenes

The Track View Editor also has a number of properties that can be set to customize and fine-tune character animation.

To set AnimObject properties

1. In the Track View editor, right-click the applicable **AnimObject** node, click **Add Track, Properties**, then click a property.
2. Select the property in the tree pane, then double-click in the timeline window to place.
3. Click the green marker, then under **Key Properties**, adjust the values of the key properties.

Moving an Entity in a Scene

You can use the Track View editor to move or rotate any entity in a scene.

To move a character in a scene

1. In the Track View editor, add the character to the desired sequence, then click the Red record button.
2. In the Lumberyard viewport, click the character, then move or rotate as desired. This automatically updates keys at the current position of the slider in the Track View sequence timeline slider.
3. Double-click the key to access **Key Properties**, then adjust values as needed.
4. Click the **Curve Editor** button, then select the tracks where the curve needs to be adjusted.

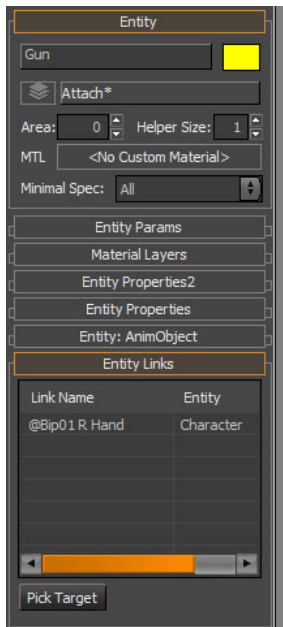
5. Drag a selection box around all the keys you want to change.
6. Click the **Set In/Out Tangents to Auto** button.

Adding Scene Character Attachments

You can add attachments to characters by creating a bone attachment link. This is useful when your character is picking or moving objects and then placing them back in the scene.

The bone attachment link is controlled using the **Link Object** and **Unlink Selection** buttons in Lumberyard Editor. Keep the following in mind when adding character attachments for cutscenes:

- Ensure the characters are properly named to prevent any errors when linking attachments.
- The @ prefix for the Link Name is essential, and is used to identify the link as a bone attachment link.
- Attachments do not need to be precisely placed as they can be adjusted after the link is created.
- If the character has a skeletal mesh, pressing the `Shift` key displays the list of bones.
- Once a link has been created, turn the **Link Object** button off.



Using Look IK in Scenes

Lumberyard supports parametric-blending for automated LookIK that can be used to make characters look at targets at specific locations, even in different locomotion cycles. LookIK can be called using Flow Graph, Track View, the AI system, or from code. Track View is mostly used for camera-controlled scenes, while Flow Graph is used in most player-controlled scenes.

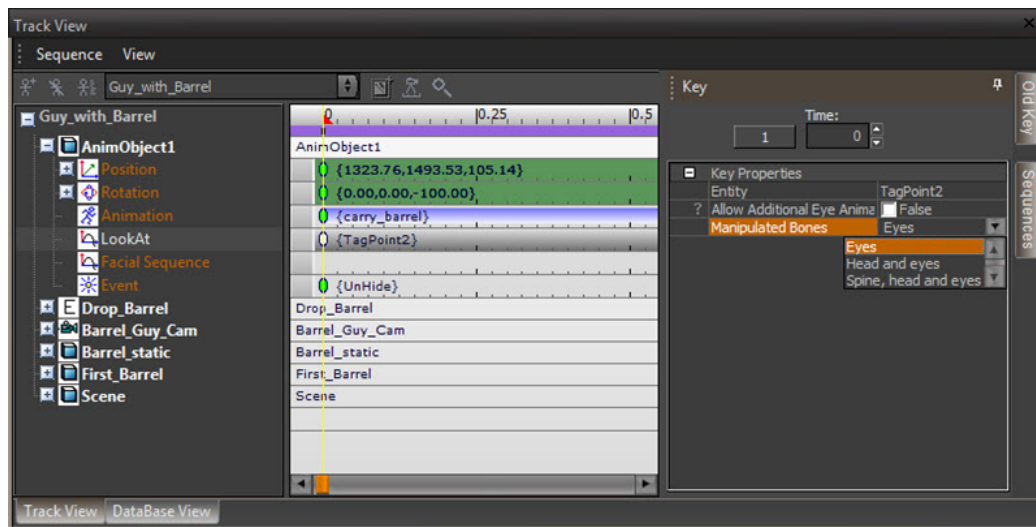
The character with LookIK tries to look at the target as long as possible, then turns its head away. The spine, head, eyelids, and eyeballs are all animated to make the character look in the target direction.

To use LookIK in a scene

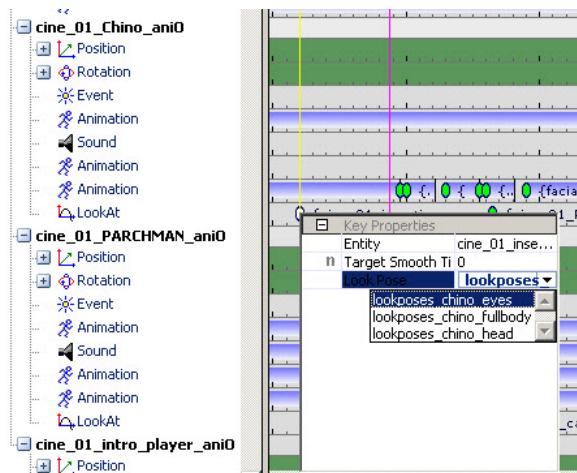
1. In the Track View editor, right-click the applicable AnimObject character node, then click **Add Track, LookAt**.

2. Double-click the timeline row for LookAt, then click the green marker.
3. In **Key Properties**, do the following:
 - In **Entity**, select a target from the list.
 - In **Target Smooth Time**, enter a value. Good values for eyes are 0.1-0.2, for head 0.3-0.5, and for full body 0.7-0.9.
 - In **Look Pose**, select which part of the body aligns with the target.

To use LookIK in Track View, the **LookAt** track is added to **AnimObject** node for the applicable character.

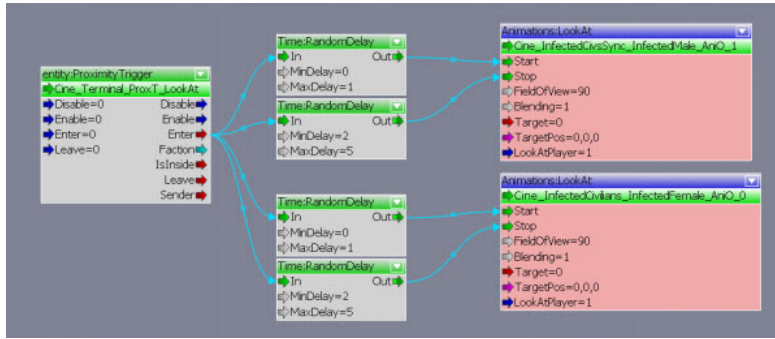


You can add multiple LookAt track keys. As soon as the timeline hits one key, the character aligns to the next key. If you want to reset LookIK, place an empty key in the timeline.



Using Flow Graph for Look IK in Scenes

You can also use the **Animations:LookAt** Flow Graph node to make a character look at a specific target or the player. Assign the character to the node and a target entity or set the **Animations:LookAt** node **LookAtPlayer** input to 1 and trigger the **Start** input to force LookIK on a character.



Blending Cinematic Animations

There are two different types of blending that can be used between two animation sequences in Track View: cross-fade blending and gap blending.

Cross-Fade Animation Blending

Cross-fade blending automatically starts if Lumberyard detects that two cinematic animation sequences overlap. The blending affects the whole section where the two animations intersect, with the weight of the second animation steadily increasing towards the end of the intersection. Specifically, at the start of the second animation, the weights for the first/second animation is 100%/0% and shifts linearly until the end of the first animation to 0%/100%.

Gap Animation Blending

Gap blending is used to blend from the end frame of the first animation to the starting frame of the second animation if a time gap exists between the two. This only works if the **End Time** property of the first animation is less than the full animation clip time.

To enable gap animation blending

1. In the Track View editor, in the sequence timeline, click the first animation.
2. In **Key Properties**, select the **Blend Gap True** check box.

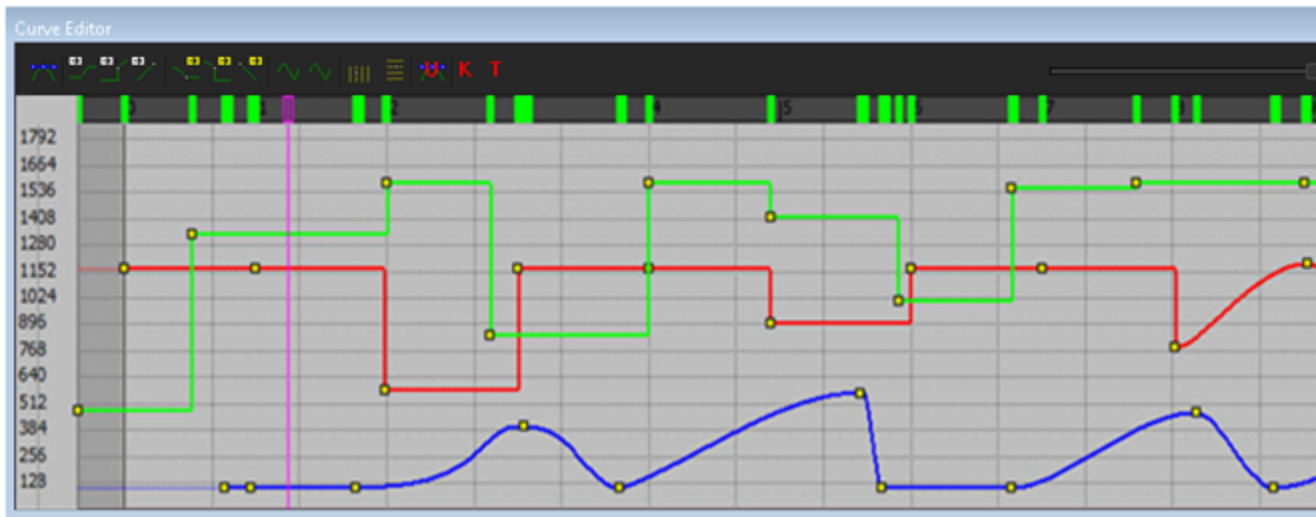
Using Track View Animation Curves

The Curve Editor enables precise animation control for entities within the Track View editor. Position, rotation, and scale can be independently controlled.

To use the Curve Editor for a scene

1. In the Track View editor, select the desired sequence.
2. Click **View, Both**.
3. In the **Graph** pane, click a top row button to change the shape of the graph as follows. Repeat as needed for each of the three graphs.
 - Sets the in/out tangents to auto
 - Sets the in tangent to zero
 - Sets the in tangent to step
 - Sets the in tangent to linear

- Sets the out tangent to zero
 - Sets the out tangent to step
 - Sets the out tangent to linear
 - Fits the splines to the visible width
 - Fits the splines to the visible height
4. To fine-tune the shape of the curve, double-click a point on the graph and drag it to the desired new value. Repeat for other points as needed for each of the three graphs.

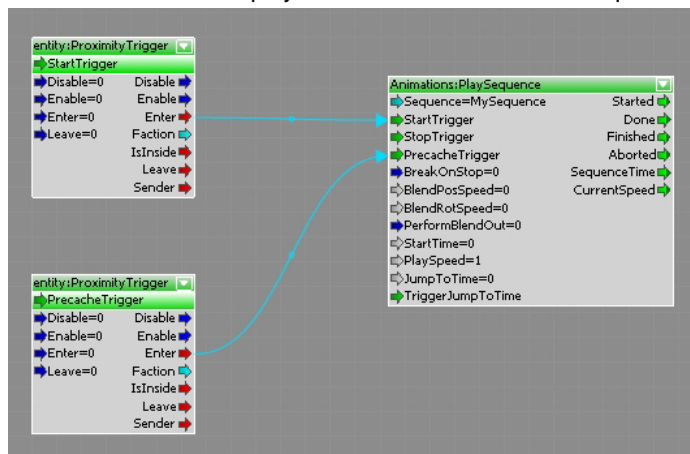


Pre-caching Cinematic Animations

Pre-caching is used to avoid animation streaming problems at the start of a sequence. The pre-cached animations remain in memory until a scene is played. Once playing, a sequence automatically pre-caches the next two seconds of needed animation data.

Optimally, pre-caching is triggered about 4-5 seconds before the sequence starts playing. However, in some cases, shorter pre-caching times work just as well. The slowest platform is the deciding factor for determining the time that is needed.

A simple pre-caching setup using two **entity:ProximityTrigger** Flow Graph nodes is shown below. The **PrecacheTrigger** input on the **Animations:PlaySequence** Flow Graph node pre-caches all animation data that is needed to play the first two seconds of a sequence.



If the **Start Time** value of a sequence has been changed to be larger than 0, pre-caching takes this into account and does not load any animation data that is not needed.

Adding Player Interactivity

There are multiple ways to create player interactivity in your cinematic scenes.

Topics

- [Looping and Jumping in a Scene \(p. 313\)](#)
- [Pausing a Scene \(p. 314\)](#)
- [Adding a Dead-Man Switch to a Scene \(p. 315\)](#)
- [Setting Player Look Around \(p. 316\)](#)
- [Adding Force Feedback \(p. 317\)](#)

Looping and Jumping in a Scene

You can jump ahead or back in time, as well as use looping, in a sequence using Track View GoTo track keys or using Flow Graph.

Scene Jumping using GoTo Track Keys

The GoTo track allows you to jump ahead or back in time while the sequence is running. It is primarily used to turn parts of a sequence into a loop.

Using a GoTo track key to jump to a different point in time automatically applies animation blending on all currently playing animations in the sequence. If animation blending is not desired for a scene, use the **Loop** property instead.

GoTo track keys placed at the end of a sequence never trigger. Instead, the sequence simply stops playing. To resolve this issue, slightly extend the end time of the sequence.

To jump in a scene using a GoTo track key

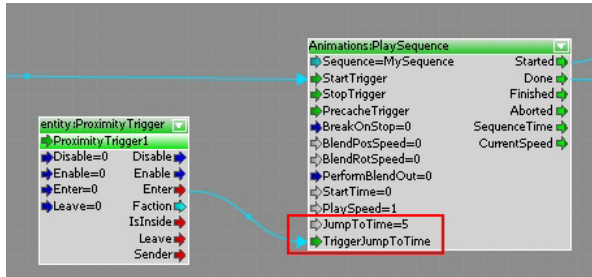
1. If applicable, in the Track View editor, right-click the top node and click **Add Director(Scene) Node**.
2. Right-click the applicable node and then click **Add Track, GoTo**.
3. Add a key in the GoTo track where you want the jump to occur.
4. In the timeline, right-click the key, and in **Key Properties**, adjust the value of the **GoTo Time** parameter.

If the duration of a sound overlaps into a GoTo track loop, the last portion is played repeatedly. In most cases, this behavior is not desired and the sound key must get moved further away in time from the target of the GoTo jump so as not to overlap it.

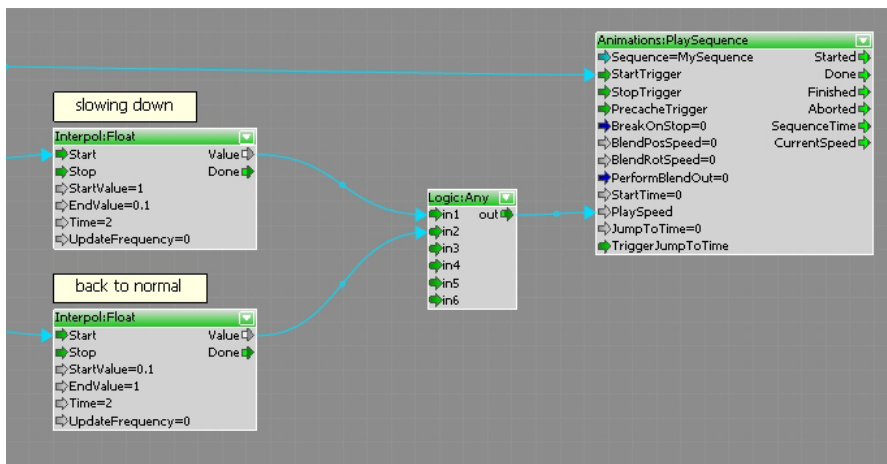
Using the Loop property instead of a GoTo track key is useful for moving mechanical parts (such as helicopter rotor blades) or when the animations are pose-matched and do not require blending. GoTo tracks could be used, but the effect does not look good.

Scene Jumping using Flow Graph

Using the **Animations:PlaySequence** Flow Graph node, you can activate the **Trigger Jump To Time** to make the sequence jump to the specified time while the sequence is playing.



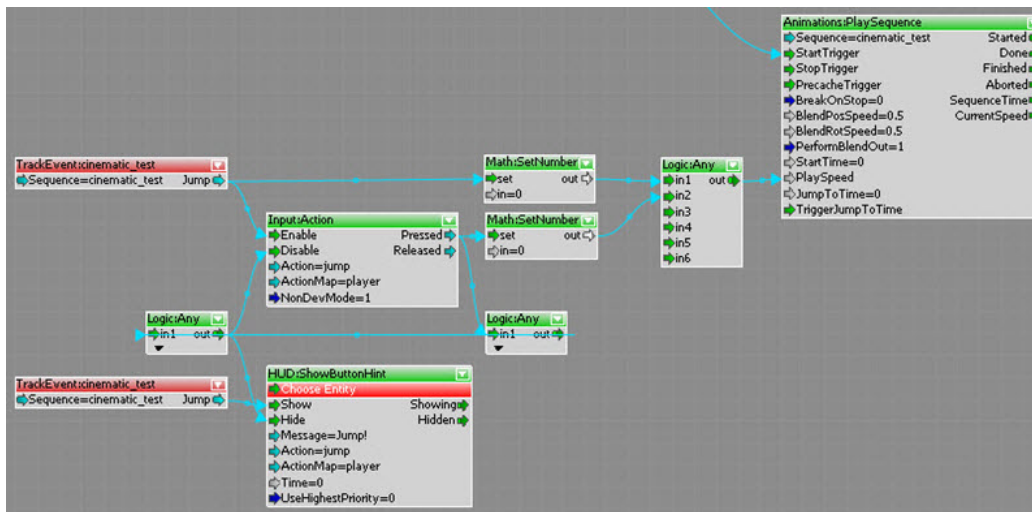
You can also set up multiple triggers that jump to different times using **Math:SetNumber** nodes and a **Logic:Any** node, as shown in the following image.



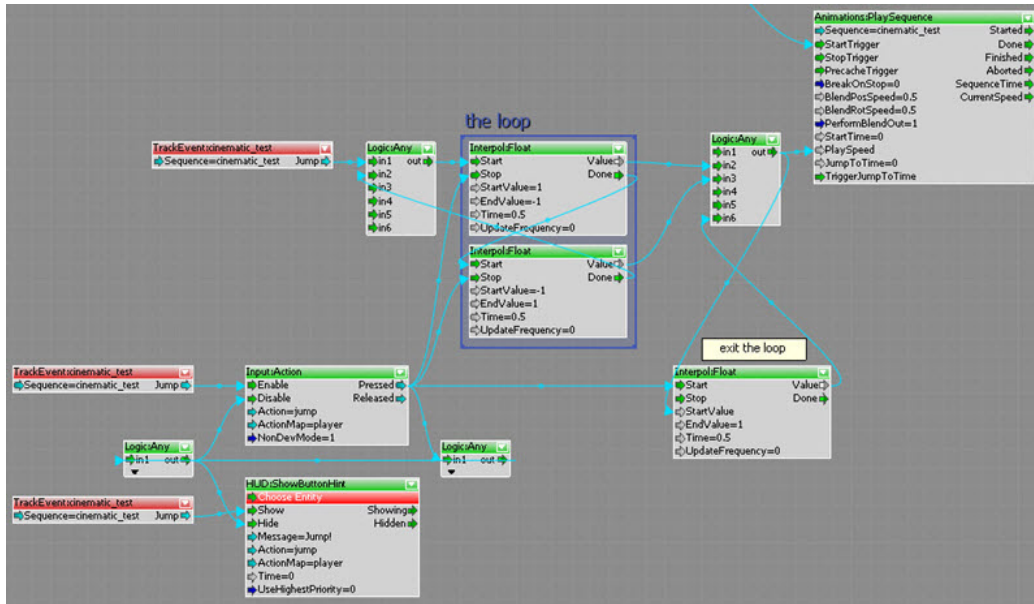
Pausing a Scene

Using Flow Graph, you can pause a sequence and keep it in a loop until the player presses a button. This can be useful when the player picks something up, moves forward, or jumps, for example.

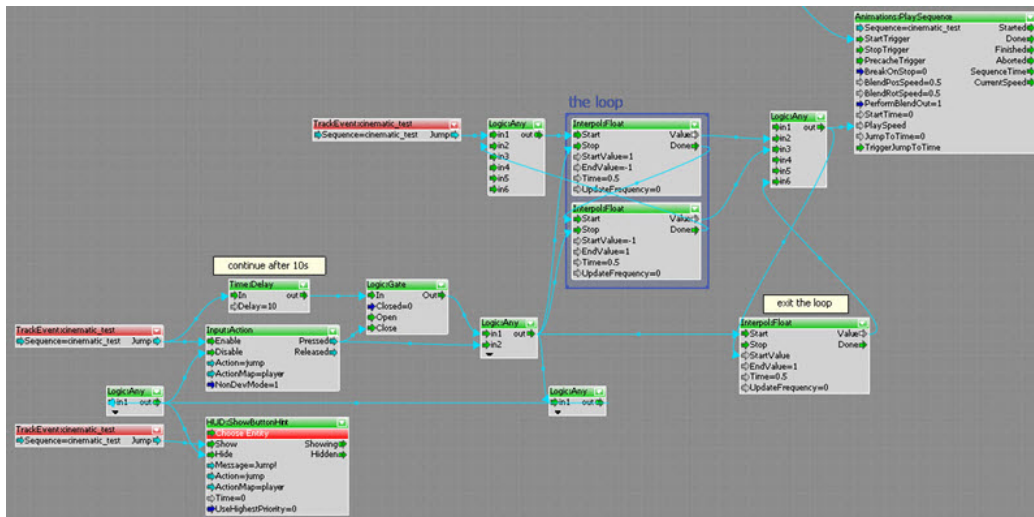
A simple implementation would be to add a track event to a sequence when the pause should happen. Then set the **PlaySpeed** input value to 0 in the **Animations:PlaySequence** Flow Graph node, and then to 1 when the player presses the required button, as shown below.



With this method, however, the sequence stops and continues suddenly and is completely static; there is no movement at all during the pause. A better method would be to add several **Interpol:Float** nodes to slow down the play speed, and a small loop to keep some movement in the scene.

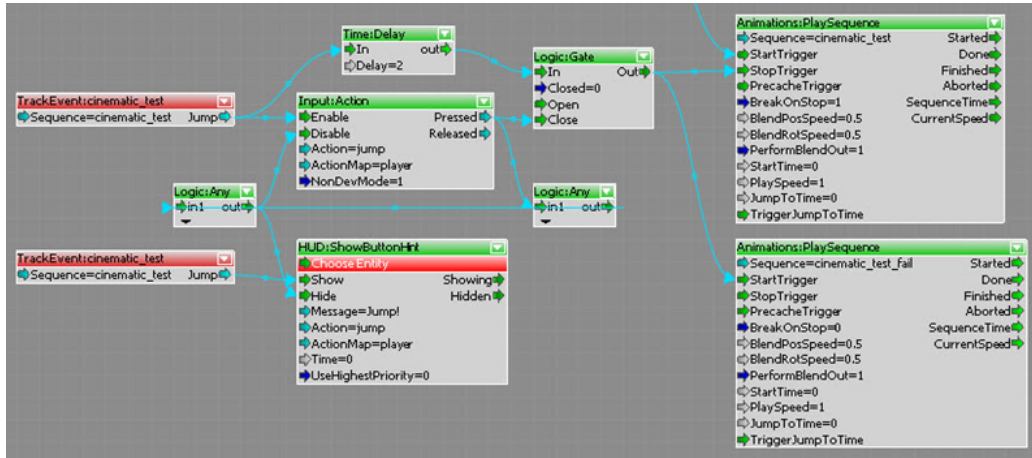


However, this would create an infinite pause of the sequence. To make it continue automatically after a certain amount of time, add a **Time:Delay** node as an optional path to the **Input:Action** node. Note the use of the **Logic:Gate** node that is used in the following example to prevent the sequence from continuing twice.

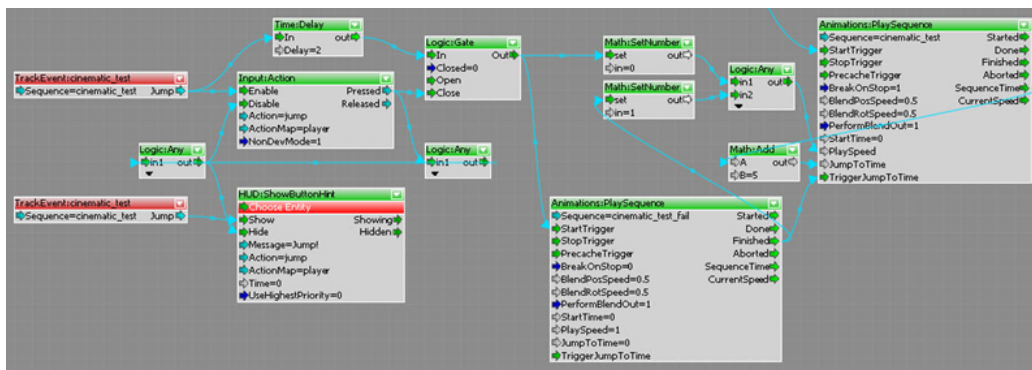


Adding a Dead-Man Switch to a Scene

Using Flow Graph, a dead-man switch can be implemented. When a player fails to perform a specified action by a certain time, such as a button push, the sequence stops and the player dies. An example is shown below.



A less strict implementation can be set up where, for example, instead of the player dying, the sequence continues and the player may just stumble. The following image shows an example of how to use a separate sequence that runs parallel to the main sequence.



Setting Player Look Around

If the **Cutscene** flag is enabled, the player can look around by rotating the cutscene camera within a certain range.

To set up player look around

1. In the Track View editor, right-click the main sequence node and click **Add Script Variable**. Name it something like `Cinematic_CameraLookUp`.
2. Repeat Step 1 three times, naming each script variable for a direction, such as `Cinematic_CameraLookDown`, `Cinematic_CameraLookLeft`, and `Cinematic_CameraLookRight`, for example.
3. For each script variable, click **Value**, then under **Key Properties**, enter a value, which represents the number of degrees the camera can be moved beyond its default position for the respective frame.

If desired, it is possible to slowly decrease these values to zero to make it less obvious that it gets disabled at a certain point.

Adding Force Feedback

Force feedback (also known as haptics) refers to the activation of gyros and actuators in game controllers and rumble chairs. This can be used for anything ranging from subtle heartbeats, to earthquake rumbling, to weapon recoil, to explosions.

Using Track View for Force Feedback

You can add force feedback rumble tracks to your cutscene using the Track View editor. Two variables are needed—one for the low-frequency motor, and one for the high-frequency motor of the game device.

To add force feedback using Track View

1. In the Track View Editor, right-click in the tree pane, click **Add Script Variable**, and name it something like `Cinematic_Rumble_Low`.
2. Repeat Step 1, giving the second script variable a different name, such as `Cinematic_Rumble_High`.
3. Select the applicable sequence, then click the **Edit Sequence** button.
4. In the Sequence Properties box, select the **Cut Scene** check box.
5. For each variable, adjust the values by moving the sliders in the graph from **0** (off) to **1** (maximum).

Note

Lumberyard clamps the value to 1, even though the slider goes higher.

You can also use the Curve Editor to fine tune the rise and fall of the rumble effect. Keep in mind that external device gyros and actuators need a bit of time to get going and to fully stop.

Using Flow Graph for Force Feedback

You can also use Flow Graph to add rumble effects using the following nodes:

- **Game:ForceFeedback**
- **Game:ForceFeedbackTweaker**
- **Game:ForceFeedbackTriggerTweaker**

To add force feedback using Flow Graph

1. In the Flow Graph editor, expand the **Game** node.
2. Right-click in the graph, then click **Add Node, Game, ForceFeedbackTweaker**.
3. Adjust the values of the **LowPass** and **HighPass** inputs. Valid values range from **0** (off) to **1** (maximum).

Note

Lumberyard clamps the value to 1, even though the slider goes higher.

You can also use the Curve Editor to fine tune the rise and fall of the rumble effect. Keep in mind that external device gyros and actuators need a bit of time to get going and to fully stop.

Using Layers for Scenes

You should create a new layer for each cinematic scene. Use the following procedure to create a layer for a scene.

To create a layer for a scene

1. In Lumberyard Editor Rollup Bar, click the **Layers** tab.
2. Click the **New Layer** button and enter a name. Ensure the **Visible** and **Use In Game** check boxes are selected.

Capturing Image Frames

There are two methods you can use to capture image frames.

Capturing Image Frames using Render Output

The easiest way to capture image frames is to simply click **Tools, Render Output** from the Track View editor, change the dialog properties to the settings you wish, click **Add** to add the capture as an item in the **Batch** list, then select **Start** to start the capture.

The aspect ratio for captured image frames is set by the **Perspective View Aspect Ratio** value, which is 1.3333 by default. You can change this value in Lumberyard Editor as follows:

To change the aspect ratio of image frame captures

1. In Lumberyard Editor, choose **File, Global Preferences, Editor Settings**.
2. In the **Preferences** window under **Viewports**, click **General**.
3. Under **General Viewport Settings**, change the value for **Perspective View Aspect Ratio**.

Capturing Image Frames using a Capture Track

This method captures images when a sequence is played in game-mode only.

To capture images using a capture track

1. Using the context menu on the **Director Node** you wish to capture, add a **Capture Node**.
2. Double-click the created track to add a **Capture Key Frame**. Refer to the table below to set the **Key Properties**.
3. Set up a flow graph to play the sequence on game start. For specific instructions, see [Playing a Sequence \(p. 294\)](#).

Key Properties

Property	Description
Duration	Sets the capture duration in seconds.
Time Step	Forces a fixed frame rate in seconds by using a specified time step, where time step = 1/number of frames.
Output Format	Selects output to various file formats.

Property	Description
Output Prefix	Selects a prefix to apply to the image file names.
Output Folder	Specifies the directory where the image files are stored under <code>\directory Cache\project_name\pc\project_name\</code> .
Buffer	Frames#misc outputs <code>.tga</code> files and <code>.hdr</code> information. Just frame outputs normal image data in the format you set. Stereo captures stereo 3D so 1 frame per eye is captured. This needs a proper stereo 3D setup before it can be used.
Just 1 Frame	Chooses between single or multi-frame image capture.

Capturing Image Frames using Console Variables

You can also use the following console variables for image frame capture:

fixed_time_step

Lowers the game speed to achieve a constant frame rate throughout the sequence. The default time step is `0.0`, while a time step value of `0.04` specifies a 25 fps gameplay speed, for example.

capture_frames

A value of `1` enables frame capture.

capture_file_format

Sets the output format for the images. Valid values are `.jpg`, `.tga`, and `.tif`.

capture_file_prefix

Sets a file name prefix to use for captured frames. `'Frame` is used if this is not set.

capture_buffer

Sets the type of buffer to capture. `0` = Color (rgb pixels), `1` = Color with Alpha (rgba pixels where the alpha channel is set to 255 where geometry exists, 0 otherwise).

Debugging Cinematic Scenes

Use the following console variables when profiling a scene:

- `r_displayinfo 3` – Gives you basic performance information. It also gives you a warning when you exceed texture streaming memory.
- `p_profile_entities 1` – Runs your scene and looks for fluctuations. Any entity causing large peaks should be investigated.
- `r_stats 6` – Finds assets with large draw calls or excessive materials, where shadows can be disabled, etc.
- `r_stats 15` – Prints detailed frame timings for specific render passes like static geometry or lighting. Blue = Within budget. Red = Over budget.
- `e_debugdraw 2 | 3` – Value of `2` shows Polycount and value of `3` shows current LOD of selected entity.
- `e_CameraFreeze 1` – Locks your current view and allows you to look around without redrawing any elements. This allows you to see where the problems are and fix them.
- `mov_debugEvents 1` – Shows the names of all actively playing sequences in-game.

Component Entity System

component entity system is in preview release and is subject to change.

The component entity system provides a modular and intuitive construction of game elements. The component entity system works at both the system level and the entity level, and employs reflection, serialization, messaging using the event bus (EBus), fully cascading prefabs (slices), and the ability to drag-and-drop and edit component objects in Lumberyard Editor.

This section describes how to add and customize the components available in Lumberyard Editor. For information on creating your own custom components programmatically, see [Component Entity System](#) in the [Amazon Lumberyard Developer Guide](#).

The following Lumberyard Editor tools are used to improve workflow for the component entity system.

- Component Palette
- Entity Outliner
- Entity Inspector
- File Browser

Note

The component entity system replaces the existing [Object and Entity System \(p. 416\)](#) in Lumberyard at a future date.

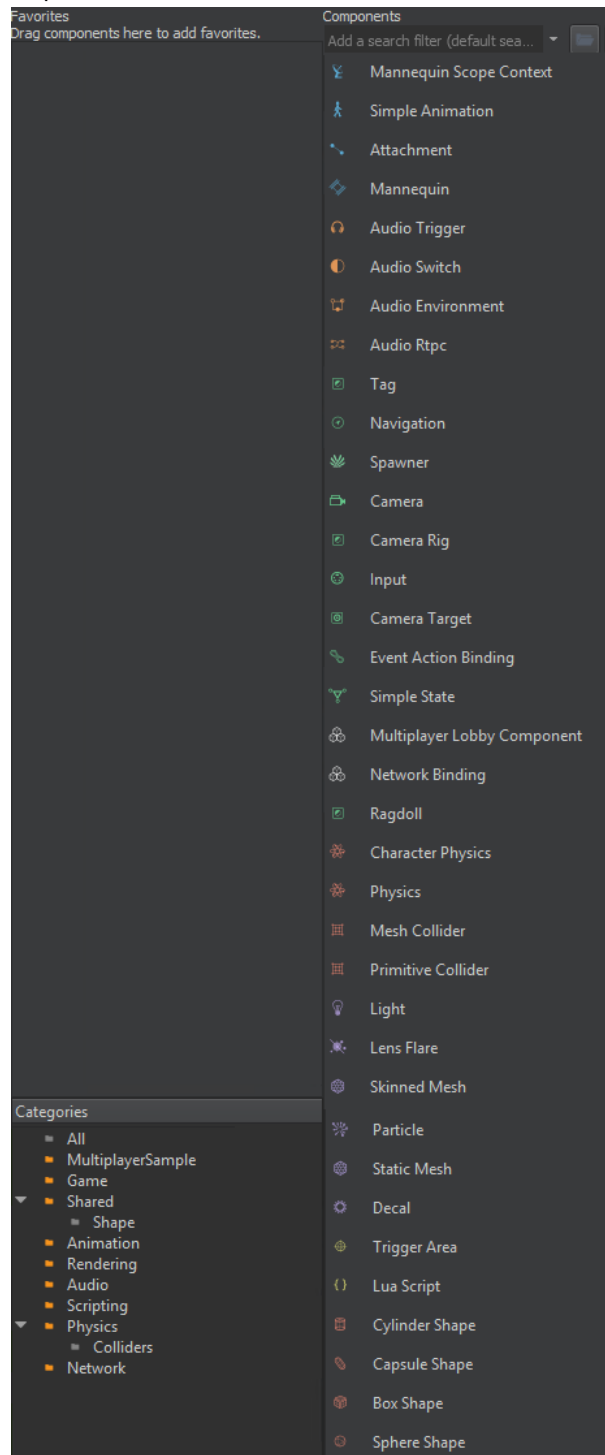
Topics

- [Component Palette \(p. 320\)](#)
- [Entity Outliner \(p. 323\)](#)
- [Entity Inspector \(p. 323\)](#)
- [File Browser \(p. 324\)](#)
- [Component Reference \(p. 327\)](#)
- [Working with Entities \(p. 410\)](#)
- [Working with Slices \(p. 413\)](#)

Component Palette

component entity system is in preview release and is subject to change.

Component Palette is used to find available components to create or add to existing entities. Component Palette provides drag-and-drop support to the Lumberyard Editor viewport and to the Entity Inspector.



To open Component Palette

- In Lumberyard Editor, choose **View, Open View Pane, Component Palette**.

The **Component Palette** features three panels:

- **Favorites** – Customizable list of components that you frequently use. Drag components from the **Components** panel to add a favorite component. To remove a component from favorites, right-click and click **Remove**.
- **Categories** – Component categories. Click a category to display only the components in that category.
- **Components** – List of components. In addition to limiting components by category in the **Categories** panel, you can also use the search filter at the top of this panel.

You can create entities using several different methods, as listed below:

To create entities

1. Drag one or more components from **Component Palette** or your **Favorites** panel into the Lumberyard Editor viewport.
2. Drag one or more components onto an entity in the **Entity Outliner**.
3. Drag one or more components onto an entity in the **Entity Inspector**.

Component Palette Attributes

The **Component Palette** is configured using the reflected data of the **Editor Component**.

For example:

```
AZ::EditContext* editContext = serializeContext->GetEditContext();
if (editContext)
{
    editContext->Class<EditorParticleComponent>("Particle", "")->
        ClassElement(AZ::Edit::ClassElements::EditorData, "")->
            Attribute(AZ::Edit::Attributes::Category, "Rendering")->
            Attribute(AZ::Edit::Attributes::Icon, "Editor/Icons/Components/
Particle")->
            Attribute(AZ::Edit::Attributes::ViewportIcon, "Editor/Icons/
Components/Viewport/Particle.png")
            // ...
            ;
}
```

You can modify the following attributes.

- `AZ::Edit::Attributes::Category` – Name of category for the component; organizes components. To specify a subcategory, use the forward slash '/' character. For example, `Attribute("Category", "Physics/Colliders")`.
- `AZ::Edit::Attributes::Icon` – Path to the icon that will be displayed in the Component Palette's list of components.
- `AZ::Edit::Attributes::ViewportIcon` – Although not part of the Component Palette, **ViewportIcon** refers to the icon that will be displayed on the entity in the viewport.

If a category is not provided, the component is displayed as **Uncategorized**.

If an `Icon` or `ViewportIcon` attribute is not provided, a default icon is used.

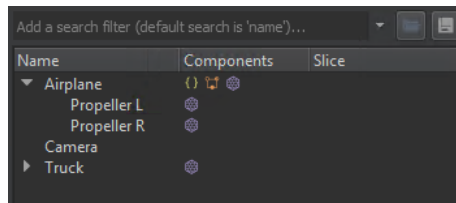
Entity Outliner

component entity system is in preview release and is subject to change.

The Entity Outliner shows all entities in the world, including key data about each entity, including its components and the slice to which it belongs. Entity Outliner is useful for scene searching, hierarchical viewing, and at-a-glance preview of slice and component information.

To open Entity Outliner

- In Lumberyard Editor, choose **View, Open View Pane, Entity Outliner**.



Parenting

Entities with a transform parent appear nested in the Entity Outliner.

To make one entity the transform parent of another, drag and drop the entity's name onto its desired parent.

Filtering

Enter text in the filter field to find specific entities. Any entity whose name does not match is hidden in the Entity Outliner.

Delete all text to resume showing all entities.

Slices

The **Entity Outliner** displays the slice from which the entity was instantiated.

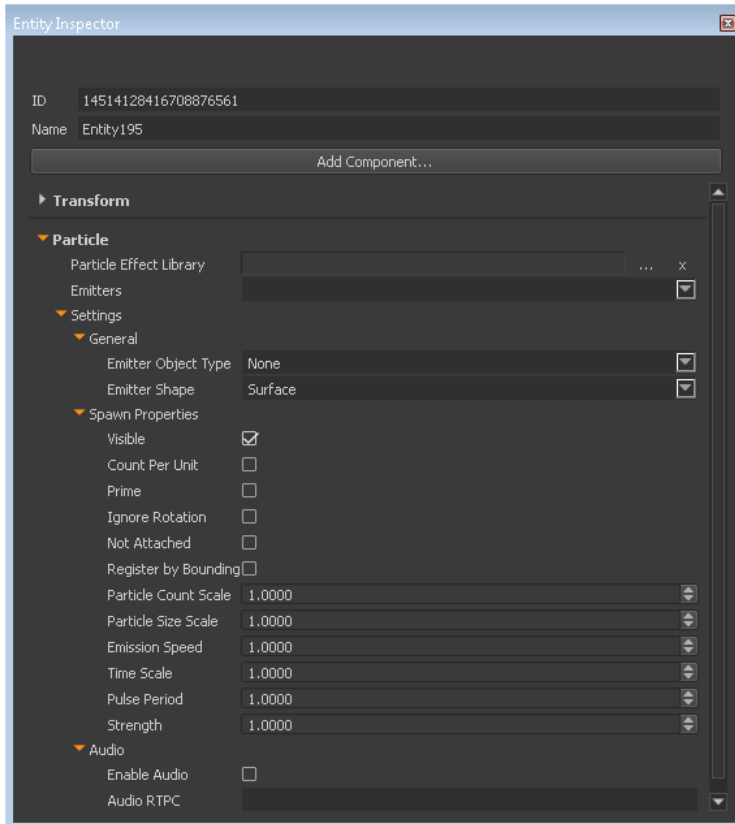
The color next to the slice designates the instance of the slice to which the entity belongs. Multiple entities that belong to the same instance of a slice share the same color. Entities that come from different instances of a common slice have different colors.

Entity Inspector

component entity system is in preview release and is subject to change.

The **Entity Inspector** is used to add component entities and modify their settings and properties.

For a list of component entities available, see [Component Reference \(p. 327\)](#).



To use Entity Inspector

1. In Lumberyard Editor, choose **View, Open View Pane, Entity Inspector**.
2. In the Lumberyard Editor viewport, select an entity.
3. In Entity Inspector, click **Add Component**.

File Browser

component entity system is in preview release and is subject to change.

File Browser is used to create and populate entities. It can be used with the **Entity Inspector** and **Entity Outliner** to improve your workflow. **File Browser** displays your assets in a tree view that mirrors your assets directory. When **File Browser** detects an asset that is associated with a single component type, it displays the associated icon if possible.

File Browser provides drag-and-drop support to the Lumberyard Editor viewport, the **Entity Inspector**, and to component **Asset** fields.

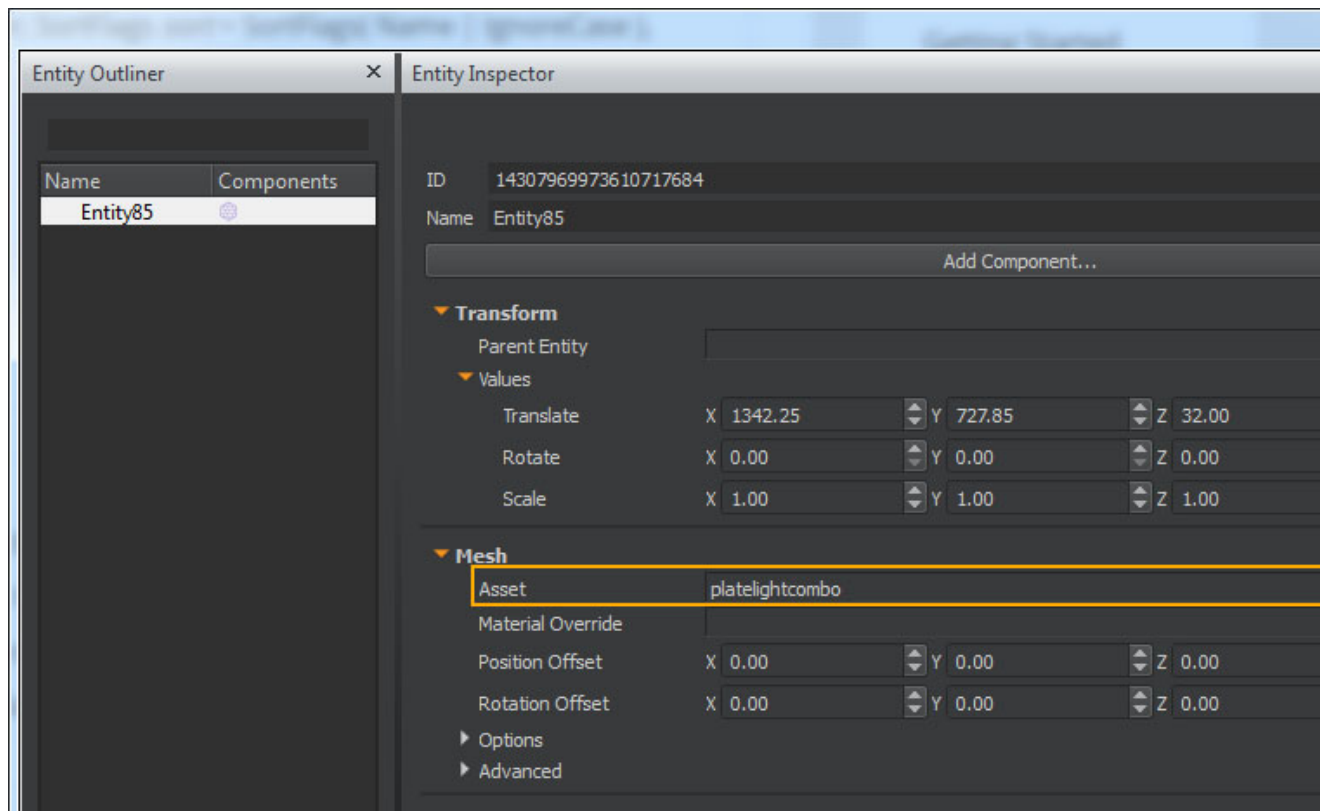
To open File Browser

- In Lumberyard Editor, click **View, Open View Pane, File Browser**.

Asset Drag and Drop

File Browser supports dragging and dropping assets into multiple windows. In many cases, it behaves like the **Component Palette**, except that it uses information about a specific asset to skip some steps in component entity creation. If an asset is associated with a single component entity type, which is denoted by its associated icon, then do the following:

- Drag an asset into the Lumberyard Editor viewport to create a new component entity at the cursor's location, add the associated component, and assign that asset into that component. For example, dragging a mesh asset (*.cgf in the figure below) creates a new component entity, adds a mesh component, and assigns the dragged asset into the **Asset** field.
- Drag an asset directly into the **Entity Inspector** to add the associated component to the selected entity(s) and assign the asset.
- Drag over the name of an entity in **Entity Outliner** to add the associated component and assign the asset to that entity.

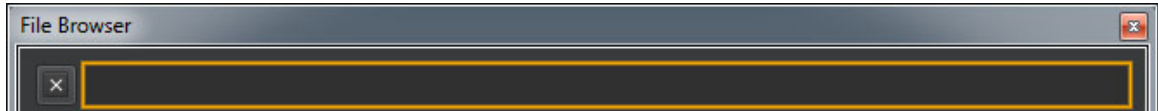


Entity Inspector also supports typed fields. These fields (such as the highlighted **Asset** field in the figure above) contains a dialog that allows you to search for assets of the correct type. Since dragged assets contain their type information, that information can also be used by asset fields to check for a valid asset. So for example, materials can be dropped on material fields but cannot be dropped in mesh fields.

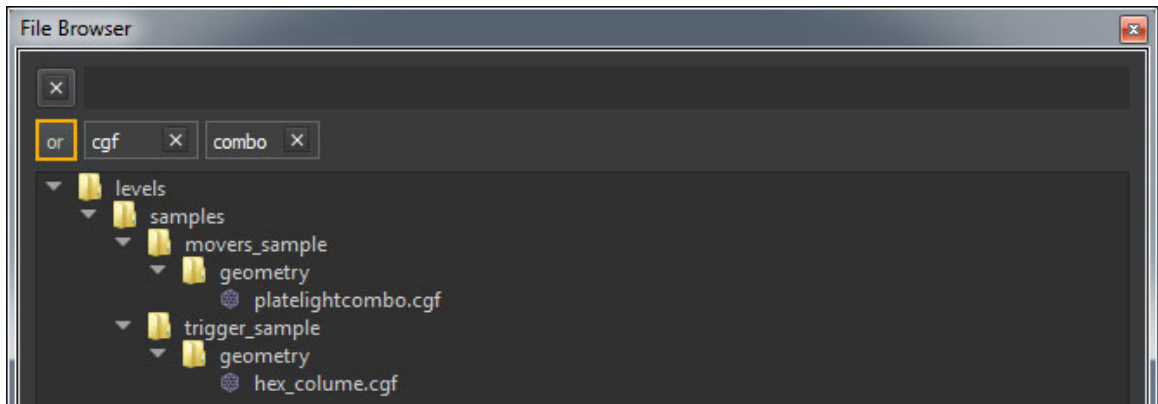
Finally, dragged assets also contain basic file name information, and that can be used by any untyped fields that support text drops.

Filtering

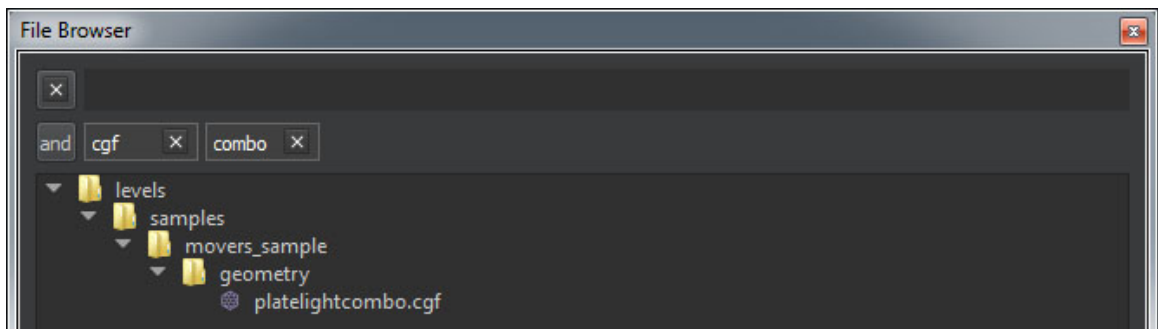
You can filter assets in the **File Browser** by typing search criteria at the top of the window.



Every search you enter creates a search criteria widget. These widgets can be individually removed from your search, or you can remove them all by clicking the **X** to the left of the search box. While search criteria also finds folders, any directory that contains a matching asset also remains visible. Adding multiple search criteria causes your window to look similar to the following picture. This search shows all assets that contain either **cgf** or **combo**.

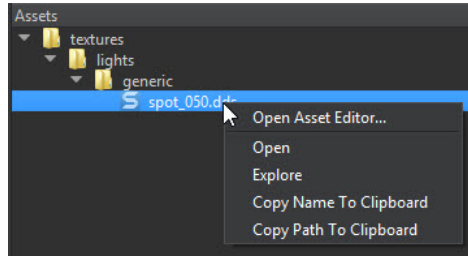


Clicking the **X** on a criteria removes it from the window and re-filters the results. As long as you have one criteria, the filter type button on the left is visible. Toggling that button switches your filtering criteria from match any (**or**) to match all (**and**). Toggling the above search to **all** changes the results to the following:



File Operations

Right-click on any entry to display a context menu that allows you to open the file, search for it on disk or copy its name to the clipboard. If source control is enabled, there are additional options to do source control operations, such as checking files in or out, or showing the history on a file.



Component Reference

component entity system is in preview release and is subject to change.

The following sections describe the various components that are available in the component entity system.

Topics

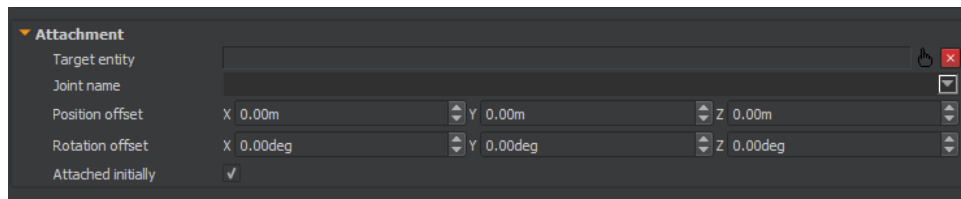
- [Attachment Component \(p. 328\)](#)
- [Audio Environment Component \(p. 329\)](#)
- [Audio Rtpc Component \(p. 330\)](#)
- [Audio Switch Component \(p. 331\)](#)
- [Audio Trigger Component \(p. 332\)](#)
- [Behavior Tree Component \(p. 334\)](#)
- [Camera Component \(p. 336\)](#)
- [Camera Rig Component \(p. 336\)](#)
- [Camera Target Component \(p. 340\)](#)
- [Constraint Component \(p. 341\)](#)
- [Decal Component \(p. 349\)](#)
- [Event Action Binding Component \(p. 350\)](#)
- [Flow Graph Component \(p. 353\)](#)
- [Input Configuration Component \(p. 354\)](#)
- [Lens Flare Component \(p. 358\)](#)
- [Light Component \(p. 361\)](#)
- [Lua Script Component \(p. 365\)](#)
- [Mannequin Component \(p. 367\)](#)
- [Mannequin Scope Context Component \(p. 373\)](#)
- [Navigation Component \(p. 374\)](#)
- [Particle Component \(p. 377\)](#)
- [Character Physics Component \(p. 379\)](#)
- [Physics Component \(p. 384\)](#)
- [Mesh Collider Component \(p. 386\)](#)
- [Primitive Collider Component \(p. 387\)](#)
- [Rag Doll Component \(p. 387\)](#)
- [Shapes Components \(p. 390\)](#)
- [Simple Animation Component \(p. 392\)](#)

- [Simple State Component](#) (p. 397)
- [Skinned Mesh Component](#) (p. 399)
- [Spawner Component](#) (p. 400)
- [Static Mesh Component](#) (p. 402)
- [Tag Component](#) (p. 404)
- [Trigger Area Component](#) (p. 406)

Attachment Component

component entity system is in preview release and is subject to change.

The attachment component lets an entity attach to a bone on the skeleton of another entity. Specifically, the transform of the target bone is checked each frame and if the target bone is not found, then the target entity transform origin is followed.



Attachment Component Properties

The **Attachment** component has the following properties:

Target Entity

Entity to attach to.

Joint Name

Attach to this joint on the target entity. If none is chosen then attach to the target's world transform.

Position Offset

Local position offset from the target in meters.

Rotation Offset

Local rotation offset from the target in degrees.

Attached Initially

Whether to attach to the target upon activation.

EBus Request Bus Interface

Use the following request functions with the EBus interface to communicate with other components of your game.

For more information about using the Event Bus (EBus) interface, see [Event Bus \(EBus\)](#).

Attach

Causes entity to change its attachment target. Entity detaches from its previous target.

Parameters

`targetEntityId` – ID of entity to attach to.

`targetBoneName` – Name of bone on entity to attach to. If bone is not found, then attach to target entity's transform origin.

`offsetTransform` – Attachment's offset from target.

Detach

Causes entity to detach from its target.

Parameters

None

SetAttachmentOffset

Update entity's offset from target.

Parameters

None

EBus Notification Bus Interface

Use the following notification functions with the EBus interface to communicate with other components of your game.

For more information about using the Event Bus (EBus) interface, see [Event Bus \(EBus\)](#).

OnAttached

Indicates that the entity has attached to the target.

Parameters

`targetEntityId` – ID of the target being attached to.

OnDetached

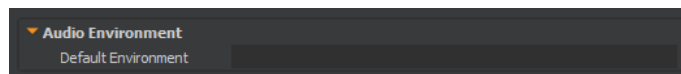
Indicates that the entity is detaching from its target.

Parameters

`targetEntityId` – ID of the target being detached from.

Audio Environment Component

The **Audio Environment** component provides access to features of the [Audio Translation Layer \(ATL\)](#) (p. 135) environments. Environments are used to apply environmental effects such as reverb or echo.



Audio Environment Properties

The **Audio Environment** component has the following property:

Default Environment

Type the name of the audio environment to use by default when setting amounts.

EBus Request Bus Interface

Use the following request functions with the EBus interface to communicate with other components of your game.

For more information about using the Event Bus (EBus) interface, see [Event Bus \(EBus\)](#).

SetAmount

Sets the amount of environmental **'send'** to apply to the default environment, if set.

Parameters

`amount` – Float value of the amount to set

Return

None

Scriptable

Yes

SetEnvironmentAmount

Sets the amount of environmental **'send'** to apply to the specified environment.

Parameters

`environmentName` – Name of ATL Environment to set an amount on

`amount` – Float value of the amount to set

Return

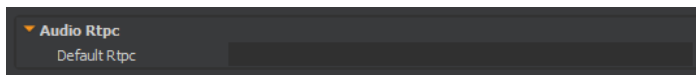
None

Scriptable

Yes

Audio Rtpc Component

The **Audio RTPC** component provides basic [Real-Time Parameter Control \(RTPC\)](#) (p. 135) functionality. An RTPC is a named variable that the audio system can interpret in many different ways. It allows game developers to set the value from the game at run time to produce real-time tweaking of sounds.



Audio RTPC Component Properties

The **Audio RTPC** component has the following property:

Default Rtpc

Type the name of the audio RTPC to use by default. You can associate any RTPC name with the entity, typically one that is meant to affect a particular trigger.

EBus Request Bus Interface

Use the following request functions with the EBus interface to communicate with other components of your game.

For more information about using the Event Bus (EBus) interface, see [Event Bus \(EBus\)](#).

SetValue

Sets the value of the default RTPC.

Parameters

`value` – Float value of the RTPC

Return

None

Scriptable

Yes

SetRtpcValue

Sets the value of the specified RTPC.

Parameters

`rtpcName` – Name of the RTPC to set

`value` – Float value to set

Return

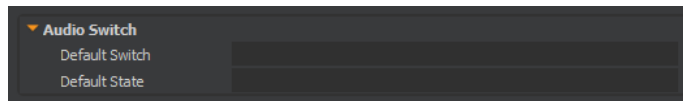
None

Scriptable

Yes

Audio Switch Component

The **Audio Switch** component provides basic [Audio Translation Layer \(ATL\)](#) (p. 135) switch functionality. With switches (and switch states), you can specify the state of an entity. The audio middleware interprets states, modifies the behavior of sounds, and plays the appropriate sounds.



Audio Switch Properties

The Audio Switch component has the following properties:

Default Switch

Type the name of the audio switch to use by default. You can associate any audio switch with the entity.

Default State

Type the name of the audio switch state to use by default. Use the [Audio Controls Editor](#) (p. 132) to assign the state to the switch. When this component is activated, the default switch is set to the default state.

Play immediately

Select this option to run upon component activation the audio **'play'** trigger.

EBus Request Bus Interface

Use the following request functions with the EBus interface to communicate with other components of your game.

For more information about using the Event Bus (EBus) interface, see [Event Bus \(EBus\)](#).

SetState

Sets the specified state of the default switch.

Parameters

`stateName` – Name of the state to set

Return

None

Scriptable

Yes

SetSwitchState

Sets a specified switch to a specified state.

Parameters

`switchName` – Name of the switch to set

`stateName` – Name of the state to set

Return

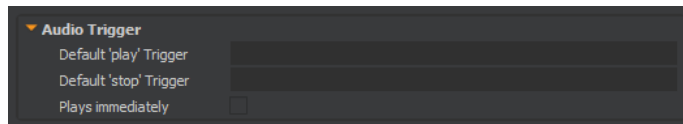
None

Scriptable

Yes

Audio Trigger Component

The **Audio Trigger** component provides basic play and stop features so that you can set up [Audio Translation Layer \(ATL\)](#) (p. 135) play and stop triggers that can be executed on demand. With an audio trigger, you can also enable the player to run or stop audio triggers by name on entities.



Audio Trigger Properties

The Audio Trigger component has the following properties.

Default 'play' Trigger

Type the name of the audio trigger that this component runs when **'play'** is called. You can change this property to specify a different default audio trigger.

Default 'stop' Trigger

Type the name of the audio trigger that this component runs when **'stop'** is called. You can specify any trigger here; you do not need to specify a **'stop'** trigger in order to stop audio, but it is a best practice to pair the two triggers. If you leave this setting blank, the **'stop'** trigger simply stops the audio trigger specified for **'play'**.

Play immediately

Select this option to run upon component activation the audio **'play'** trigger.

EBus Request Bus Interface

Use the following request functions with the EBus interface to communicate with other components of your game.

For more information about using the Event Bus (EBus) interface, see [Event Bus \(EBus\)](#).

Play

Runs the default **'play'** trigger, if set.

Parameters

None

Return

None

Scriptable

Yes

Stop

Runs the default **'stop'** trigger, if set. If no **'stop'** trigger is set, kills the default **'play'** trigger.

Parameters

None

Return

None

Scriptable

Yes

ExecuteTrigger

Runs the specified audio trigger.

Parameters

`triggerName` – Name of the audio trigger to run

Return

None

Scriptable

Yes

KillTrigger

Cancels the specified audio trigger.

Parameters

`triggerName` – Name of the audio trigger to cancel

Return

None

Scriptable

Yes

KillTrigger

Cancels all audio triggers that are active on an entity.

Parameters

None

Return

None

Scriptable
Yes

SetMovesWithEntity

Specifies whether triggers should update position as the entity moves.

Parameters

`shouldTrackEntity` – Boolean indicating whether triggers should track entity's position.

Return

None

Scriptable

Yes

EBus Response Bus Interface

Use the following response functions with the EBus interface to communicate with other components of your game.

For more information about using the Event Bus (EBus) interface, see [Event Bus \(EBus\)](#).

OnTriggerFinished

Informs all listeners about an audio trigger that has finished playing (the sound has ended).

Parameters

`triggerId` – Id of trigger that was successfully executed

Return

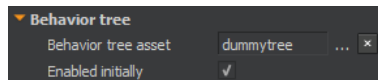
None

Scriptable

Yes

Behavior Tree Component

Use the **Behavior Tree** component to load and run a [behavior tree](#) for the attached entity.



Behavior Tree Component Properties

The **Behavior Tree** component has the following properties:

Behavior tree asset

Select an XML file that contains a behavior tree definition.

Enabled initially

When selected, the behavior tree is loaded and activated with the entity.

EBus Request Bus Interface

Use the following request functions with the event bus (EBus) interface, `BehaviorTreeComponentRequestBus`, to communicate with other components of your game.

For more information about using the EBus interface, see [Event Bus \(EBus\)](#).

StartBehaviorTree

Starts an inactive behavior tree associated with this entity.

Parameters

None

Return

None

Scriptable

Yes

StopBehaviorTree

Stops an active behavior tree associated with this entity.

Parameters

None

Return

None

Scriptable

Yes

GetVariableNameCrcs

Gets a list of all `crc32s` of the variable names.

Parameters

None

Return

`AZStd::vector<AZ::Crc32>`

Scriptable

Yes

GetVariableValue

Gets the value associated with a variable.

Parameters

`AZ::Crc32 variableNameCrc`

Return

`bool`

Scriptable

Yes

SetVariableValue

Sets the value associated with a variable.

Parameters

`AZ::Crc32 variableNameCrc`

`bool newValue`

Return

None

Scriptable
Yes

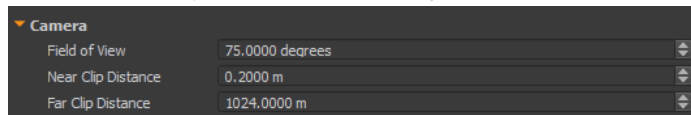
The following is an example of script using the **Request Bus Interface**.

```
behaviortreescript =  
{  
  Properties =  
  {  
    Target = {default=EntityId()},  
  },  
}  
  
function behaviortreescript:OnActivate()  
  self.behavior = BehaviorTreeComponentRequestBusSender(self.entityId)  
  self.behavior:StartBehaviorTree()  
  self.behavior:SetVariableValue(Crc32("HasTarget"),  
  self.Properties.Target:IsValid())  
end
```

Camera Component

component entity system is in preview release and is subject to change.

The camera component allows an entity to be used as a camera.



Camera Component Properties

The **Camera** component has the following properties:

Field of View

Vertical field of view in degrees.

Near Clip Plane Distance

Distance to the near clip plane of the view frustum.

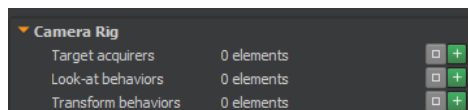
Far Clip Plane Distance

Distance to the near far plane of the view frustum.

Camera Rig Component

component entity system is in preview release and is subject to change.

Use the **Camera Rig** component to add and remove behaviors to drive your camera entity.



Camera Rig Component Properties

The **Camera Rig** component has the following properties:

Target acquirers (p. 337)

Array of behaviors that define how a camera selects a target.

Look-at behaviors (p. 337)

Array of behaviors that modify the look-at target transform.

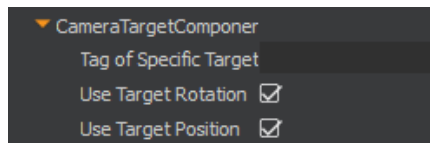
Transform behaviors (p. 339)

Array of behaviors that modify the camera transform based on the look-at target transform.

Target Acquirers

Target Acquirers identify valid targets and acquire their transforms for use in other rig behaviors.

CameraTargetComponentAcquirer



The **CameraTargetComponentAcquirer** has the following properties:

Tag of Specific Target

Filters available camera targets that have a tag.

Use Target Rotation

If selected, uses the target's rotation when determining camera behavior.

Use Target Position

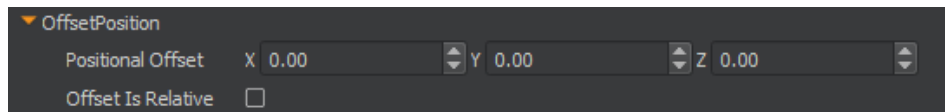
If selected, uses the target's position when determining camera behavior.

Look-at Behaviors

Look-at Behaviors changes the target transform to modify camera behavior.

OffsetPosition

Use **OffsetPosition** to change the position of the target's transform. Positions are often determined from the base of a model. But suppose, for example, that you want to determine its position 1.8 meters up from its base. You can use this property to achieve that positional offset.



Look-at Behaviors has the following properties:

Positional Offset

Vector displacement of the target transform's position.

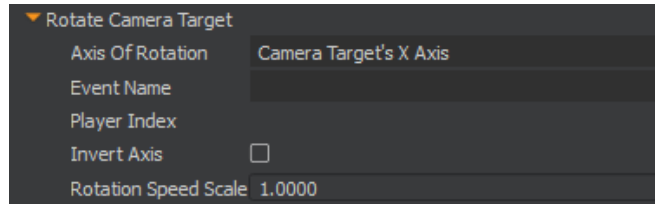
Offset Is Relative

If selected, uses local coordinates. If deselected, uses world-basis vectors for the offset.

Rotate Camera Target

Use **Rotate Camera Target** to rotate the target separately from its source target. For example, you may want your character to look up and down without pitching.

Rotate Camera Target has the following properties:



Axis of Rotation

The target cardinal's axis around which the camera rotates. Select the **X**, **Y**, or **Z** axis.

Event Name

Name of event that provides the values for the rotation.

Player Index

Index of the player (input device).

Invert Axis

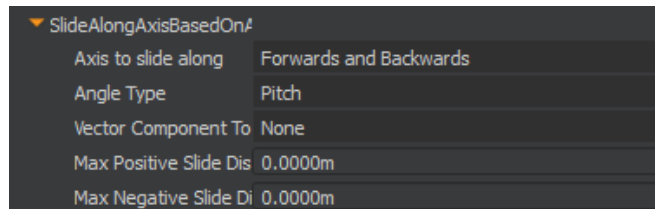
If selected, inverts the axis of rotation.

Rotation Speed Scale

Multiplier for new input values to scale the speed of rotation.

SlideAlongAxisBasedOnAngle

Use **SlideAlongAxisBasedOnAngle** to modify the position of the look-at target based on an angle. For example, say that you set the target to slide along the forward and backward axis based on pitch. As the target pitched down, then the position would move ahead of the target. If the target is attached to the character, then every time the target looked down, it would be ahead of the character. Every time it looked up, it would be behind the character.



SlideAlongAxisBasedOnAngle has the following properties:

Axis to slide along

Select an axis along which the target slides:

- **Forwards and Backwards**
- **Right and Left**
- **Up and Down**

Angle Type

Select an angle type on which to base the slide:

- **Pitch**
- **Yaw**
- **Roll**

Vector Component to Ignore

Select a vector component to ignore: **None**, **X**, **Y**, or **Z**.

Max Positive Slide Distance

The maximum slide along the axis when the angle reaches 90 degrees.

Max Negative Slide Distance


The maximum slide along the axis when the angle reaches -90 degrees.

Transform Behaviors

Transform Behaviors are a critical component of how the camera responds to the target. For example, you can set the camera to face the target, follow from a distance, or follow the target at a specific angle.

FaceTarget

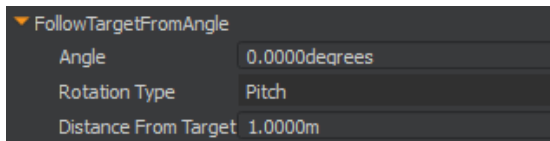
FaceTarget causes the camera to change the rotation of its transform to look at the target. To use this feature, simply add it. There are no additional properties to configure.



FaceTarget

FollowTargetFromAngle

FollowTargetFromAngle causes the camera to follow the target from a specified angle. This feature works well for top-down, isometric, and side scrolling cameras.



Follow Target from Angle has the following properties:

Angle

Angle at which to follow the target.

Rotation Type

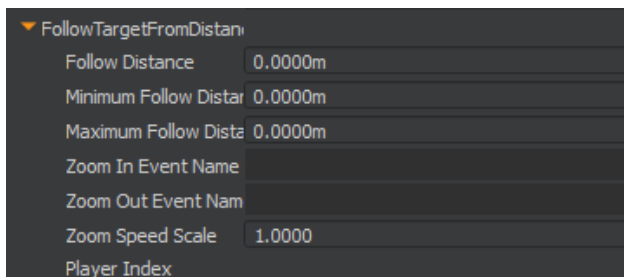
Rotation type of the angle for following the target: yaw, pitch, or roll.

Distance from Target

The distance in meters from which the camera follows the target.

FollowTargetFromDistance

FollowTargetFromDistance causes the camera to follow the target from a specified distance. You can also set named events to trigger the camera to zoom in on or out from a target.



FollowTargetFromDistance has the following properties:

Follow Distance

The distance in meters from which the camera follows the target.

Minimum Follow Distance

Minimum distance from which the camera follows the target.

Maximum Follow Distance

Maximum distance from which the camera follows the target.

Zoom In Event Name

Event name that reduces the current follow distance, in effect zooming in.

Zoom Out Event Name

Event name that increases the current follow distance, in effect zooming out.

Zoom Speed Scale

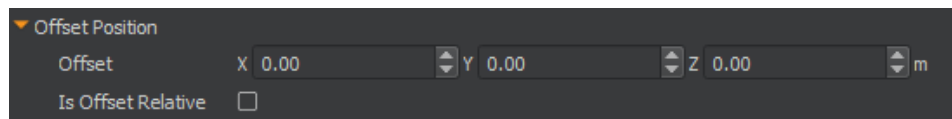
Scale amount for the incoming zoom value.

Player Index

The index of the player (device index) that this feature supports.

Offset Camera Position

Offset Camera Position sets the camera's position to the target's position with an offset.



Offset Camera Position has the following properties:

Offset

The vector offset in meters from the target.

Is Offset Relative

If selected, local basis vectors are used. If deselected, worldbasis vectors are used.

Rotate

Use **Rotate** to rotate a camera about one of its axes (**X**, **Y**, or **Z**).

Rotate has the following properties:

Angle

Angle in degrees to rotate the camera.

Axis

Axis about which to rotate the camera.

Camera Target Component

component entity system is in preview release and is subject to change.

The **Camera Target** component registers itself with listeners as a potential camera target.



Note

Orthographic cameras are not supported for primary 3D scene rendering passes, but they are supported for special case rendering passes, including UI and full screen effects.

Camera Target Component Properties

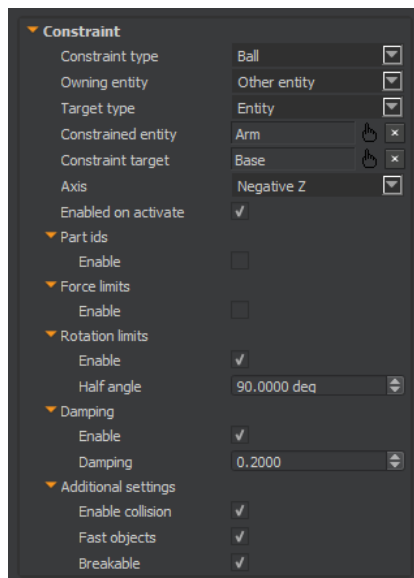
The **Camera Target** component has the following property:

Tag

Filters camera targets with the specified tag.

Constraint Component

The **Constraint** component creates a physical limitation or restriction between an entity and its target. The target is either another entity or a point in world space. When a constraint is set, the physics system applies an impulse to the entity or entities until they reach the constraint pivot. The entity owning the constraint (**Owning entity**) then moves about its pivot point according to the specified **Constraint type** (behavior).



The **Owning entity**—the entity that has the constraint component applied—must have a [physics component \(p. 384\)](#) with **Rigid Body** behavior (movable body). If its target entity is another entity, that entity must also have a physics component with either rigid body (movable) or static body (unmovable) behavior. Both entities must also have either a [primitive collider \(p. 387\)](#) or [mesh collider \(p. 386\)](#) component.

There are many ways to use a constraint component. Here are some examples:

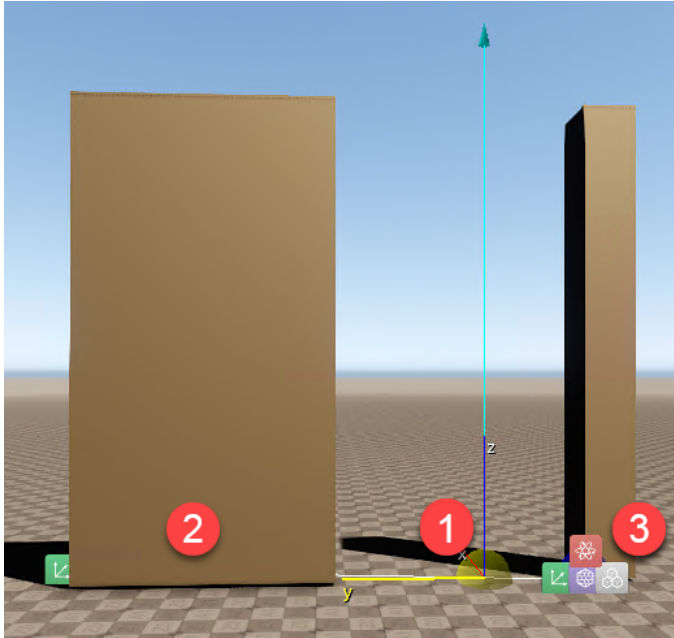
- Create two entities, and then place the constraint component on the movable entity.
- Create two entities, and then place the constraint component on a new entity between the two entities.
- Create one movable entity, and then place the constraint component on it. Set the **Target type** as **World space**.
- Create one movable entity, and then place the constraint component on a new entity near the movable entity. Set the **Target type** as **World space**.

An example procedure for using a constraint component between two entities is described below.

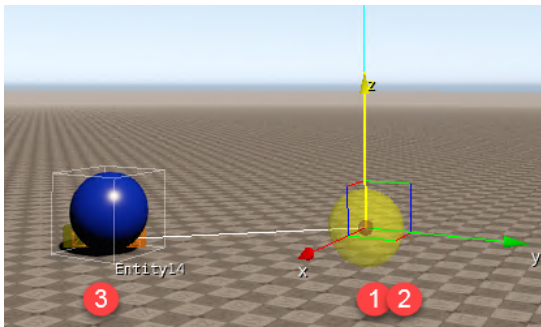
To place two entities and then place the constraint component entity between them

1. To place the first (movable) entity, and then add necessary mesh and physics components to it, follow these steps:
 - a. [Create an entity.](#) (p. 410)
 - b. Open the [Entity Inspector](#) (p. 323), and then click **Add Component, Rendering, Static Mesh** (p. 402) or **Skinned Mesh** (p. 399). Next to **Static asset** or **Skinned asset**, click the ellipsis button to select an asset.
 - c. Click **Add Component, Physics, Physics** (p. 384).
 - d. To the right of **Behavior**, click the **+** to add a [Rigid Body](#) (p. 384) behavior element. Set rigid body properties as appropriate.
 - e. Click **Add Component, Physics, Colliders, Primitive Collider** (p. 387) or [Mesh Collider](#) (p. 386).
2. To place the second entity, repeat the previous set of steps for a new entity. If you want this entity to be stationary (unmovable), however, set its physics behavior element as [Static Body](#) (p. 386) (instead of rigid body).
3. To place the constraint component on a new entity, which acts as the pivot point, do the following:
 - a. [Create an entity.](#) (p. 410)
 - b. Open the [Entity Inspector](#) (p. 323), and then click **Add Component, Physics, Constraint**.
 - c. For **Owning entity**, select **Other entity**.
 - d. For **Target type**, select **Entity**.
 - e. Next to **Constrained entity**, click the object picker (hand icon), and then in the viewport, click the first entity you created.
 - f. Next to **Constraint target**, click the object picker (hand icon), and then in the viewport, click the second entity you created.
4. Set [Constraint Component Properties](#) (p. 343) as appropriate.

The following picture shows an example of a constraint entity (1) between the constraint owner (2) and the constraint target (3). Here, the constraint target (3) is a static body (unmovable), and the constraint owner (2) is a [rigid body](#) (p. 384) (movable).



The following picture shows an example of a constraint entity (1) set with a constraint target (2) of **World Space**. The constraint owner (3) is a [rigid body](#) (p. 384) (movable).



Constraint Component Properties

The primary **Constraint** component property is the **Constraint type**. This property defines how the constraint owner moves in relation to the constraint target and constraint entity (the entity with the constraint component on it).

Constraint Type

When setting constraint properties, select one of the following **Constraint types**.

Hinge

The constraint owner moves around the selected **Axis**. The entity with the constraint component on it (constraint entity) acts as the pivot point. This constraint type provides one degree of freedom (X, Y, or Z) between the constraint owner and the constraint target.

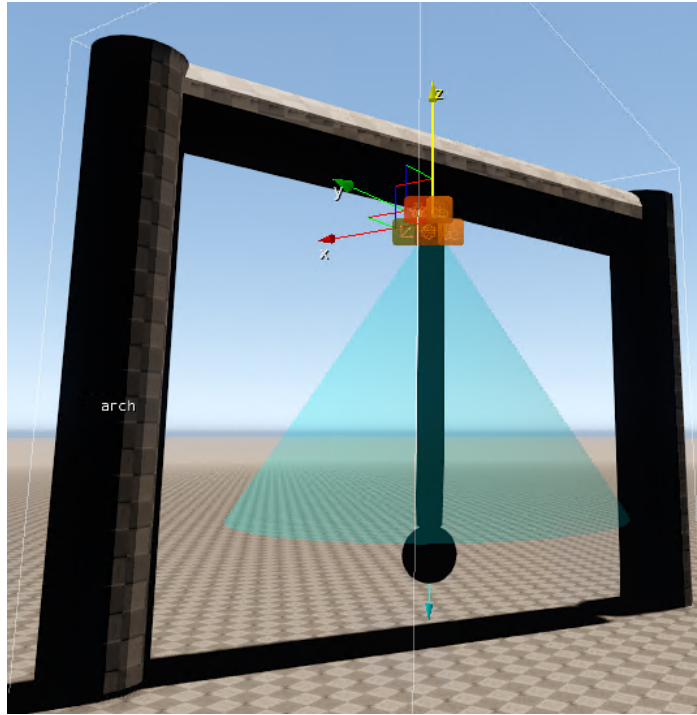
An example of a hinge is a door, which rotates about the Z axis.

Ball

The constraint owner moves around the Z axis as if attached at the pivot (constraint entity) by a ball socket or ball joint. This constraint type provides two degrees of freedom between the constraint owner and the constraint target.

An example of a ball constraint is a bell's clapper (the swinging part inside the bell housing).

This constraint type always moves about the Z axis. If you want to use this constraint type, but need it to move in a different direction (for example, attaching at the bottom and moving at the top), you must change the orientation of the Z axis. To do this, rotate the constraint owner entity.



Slider

The constraint owner slides along one axis. If rotation is also enabled, then the constraint owner acts like a movable hinge about the sliding axis.

An example of a slider constraint is a sliding door.

Plane

The constraint owner moves along one plane (XY, YZ, or XZ). Select the X axis to move along the YZ plane, the Y axis to move along the XZ plane, and the Z axis to move along the XY plane.

An example of a plane constraint is a hockey puck sliding along a smooth surface.

Magnet

The constraint owner moves towards the constraint entity, which serves as the pivot point for the magnet constraint.

Fixed

The fixed constraint must be applied to two rigid body (movable) entities. This constraint type constrains the two entities with no rotation relative to one another (they can rotate together). The distance and orientation between the constraint owner and its target are preserved.

An example of a fixed constraint is the two sides of a dumbbell—with an invisible and intangible bar fixing the two sides together.

Free

The constraint owner can move around its target anywhere in space and any distance from its target, but the orientation of the owner and its target are locked to each other.

Constraint Properties

The following properties are common to all of the constraint types, except where noted.

Constraint type

Select from hinge, ball, slider, plane, magnet, fixed, or free.

Owning entity

Select either **self** or **Other entity**.

Only select **self** if the constraint entity is on the entity that has a static or skinned mesh, and is movable (has a physics rigid body component). Otherwise, if the constraint component is on its own entity between the intended constraint owner and its target, select **Other entity**.

Target type

Select either **Entity** or **World space**.

Select **Entity** if you want the constraint owner to move in relation to another entity, whether movable (rigid body) or unmovable (static body). Select **World space** if the constraint owner should move in relation to its starting position.

Constrained entity

Visible when **Owning entity** is set to **Other entity**. Click the picker (hand icon) and, in the viewport, select the entity that is to be the constraint owner.

Constraint target

Visible when **Target type** is **Entity**. Click the picker (hand icon) and, in the viewport, select the entity that is to be the constraint target.

Axis

Visible when **Constraint type** is set to hinge, slider, or plane.

For the hinge constraint, the constraint owner rotates about the defined axis. For example, a typical door rotates on the Z axis. A typical pet door would rotate about the X or Y axis.

For the slider constraint, the axis specifies the direction that the constraint owner slides. For example, choose the Z axis if you want an up-and-down slider (like an elevator).

For the plane constraint, the axis you choose is perpendicular to the plane upon which the constraint owner moves. For example, choose X to move along the YZ plane, Y to move along the XZ plane, and Z to move along the XY plane.

Enable on activate

If selected, the constraint is enabled upon activation.

Part ids

If selected, you can specify the **Owner part id** and the **Target part id**.

Force limits

If selected, you can specify the **Max pull force** in Newtons (N) and the **Max bend torque** in Newtons per meter (NM). If the constraint is set to **Breakable**, and a force exceeding those values is exerted upon the constraint, then the constraint is removed (broken). If the **Breakable** setting is not selected, **Force limits** control how much force the constraint applies to keep the constrained entities together.

Rotation limits

Visible when constraint is set to hinge, ball, or slider.

When selected and set to hinge constraint, the **Min** setting defines the limit of the clockwise rotation. Use a negative value to set a limit clockwise. For example, -90 degrees would be at the 3 o'clock marker. -180 degrees would be at 6 o'clock. The **Max** setting defines the rotation counter-clockwise. Use a positive value. For example, 90 degrees would be at the 9 o'clock marker. 180 degrees would be at the 6 o'clock marker. You can reverse these settings by selecting the negative axis (for example, **Negative Y**).

When selected and set to ball constraint, you can set the **Half angle**. Visualize the rotation area as a cone; the half angle represents the distance between the middle of the cone and the edge. When you set the half angle, a blue cone appears in the viewport to represent the rotation area.

When selected and set to slider constraint, use a positive number for the **Max** setting to define how far in the positive direction the object can move from its starting point. Use a negative number for the **Min** setting to define how far the object can move from its starting point in the opposite direction. A blue arrow appears in the viewport when a constraint entity is selected.

Damping

If selected, you can specify the [damping \(p. 1373\)](#) of a constraint movement.

Use this setting to prevent perpetual motion, or for constraints that don't properly come to rest. Start with small values, such as 0.2 to 0.3. Values above 0.5 may make objects seem unnatural and overly-dampened.

Enable collision

If selected, enables collision between constrained entities.

Breakable

If selected, the constraint is removed (broken) if force limits are exceeded.

EBus Request Bus Interface

Use the following request functions with the EBus interface to communicate with other components of your game. For more information about using the event bus (EBus) interface, see [Event Bus \(EBus\)](#).

SetConstraintEntities

Sets which entities are affected by this constraint.

Parameters

AZ::EntityId *owningEntity* – The entity that owns the constraint (constrained entity).

Return

None

Scriptable

Yes

SetConstraintEntitiesWithPartIds

Sets which entities are affected by this constraint and which of their bones are constrained.

Parameters

AZ::EntityId *owningEntity* – The entity that owns the constraint (constrained entity).

int *ownerPartID* – Constraint owner's part ID of the bone to constrain to.

AZ::EntityId *targetEntity* – The target entity. If the target type is world space, this can be `Invalid EntityId`.

int *targetPartId* – The constraint target's part ID of the bone to constrain to.

Return

None

Scriptable

Yes

EnableConstraint

Enables all constraints on this entity.

Parameters

None

Return

None

Scriptable

Yes

[DisableConstraint](#)

Disables all constraints on this entity.

Parameters

None

Return

None

Scriptable

Yes

[EBus Notification Bus Interface](#)

Use the following notification functions with the constraint component notification bus EBus interface to communicate with other components of your game.

For more information about using the event bus (EBus) interface, see [Event Bus \(EBus\)](#).

[OnConstraintEntitiesChanged](#)

Notifies that one or both of the constrained entities has changed.

Parameters

AZ::EntityId `oldOwner` – The prior owner of the constraint.

AZ::EntityId `oldTarget` – The prior target of the constraint.

AZ::EntityId `newOwner` – The new owner of the constraint.

AZ::EntityId `newTarget` – The new target of the constraint (can be `Invalid EntityId` if constraint target is world space).

Return

None

Scriptable

Yes

[OnConstraintEnabled](#)

Notifies that the constraint has been enabled.

Parameters

None

Return

None

Scriptable

Yes

[OnConstraintDisabled](#)

Notifies that the constraint has been disabled.

Parameters

None

Return

None

Scriptable

Yes

Example Script

The following is a script sample that is intended to be placed on an entity with a constraint component of **Magnet** type, with its owner set as **Other entity**. This script alternates between three different entities to be constrained for *Transition Interval* seconds.

```
constraintexample =
{
  Properties =
  {
    ConstrainedEntity1 = EntityId(),
    ConstrainedEntity2 = EntityId(),
    ConstrainedEntity3 = EntityId(),
    TransitionInterval = 3,
  },
}

function constraintexample:OnActivate()
  self.TransitionCountDown = self.Properties.TransitionInterval
  self.ConstrainedIdx = 1
  self.ConstrainedEntities = { self.Properties.ConstrainedEntity1,
self.Properties.ConstrainedEntity2, self.Properties.ConstrainedEntity3 }
  self.busSender = ConstraintComponentRequestBusSender(self.entityId)
  self.tickBusHandler = TickBusHandler(self, 0)
  self.constraintHandler = ConstraintComponentNotificationBusHandler(self,
self.entityId)
  Debug.Log("ConstraintComponent activated for entity: " ..
self.entityId.id)
end

function constraintexample:OnTick(deltaTime, timePoint)
  self.TransitionCountDown = self.TransitionCountDown - deltaTime
  if (self.TransitionCountDown < 0.0) then

self.busSender:SetConstraintEntities(self.ConstrainedEntities[self.ConstrainedIdx],
self.entityId)
    self.ConstrainedIdx = (self.ConstrainedIdx + 1) %
table.getn(self.ConstrainedEntities) + 1
    self.TransitionCountDown = self.Properties.TransitionInterval
  end
end

function constraintexample:OnDeactivate()
  self.tickBusHandler:Disconnect()
  self.constraintHandler:Disconnect()
end

function constraintexample:OnConstraintEntitiesChanged(oldOwner, oldTarget,
newOwner, newTarget)
```

```
    Debug.Log("Constraint Changed - old owner:" .. tostring(oldOwner) .. "
old target:" .. tostring(oldTarget))
    Debug.Log("          new owner:" .. tostring(newOwner) .. " new
target:" .. tostring(newTarget))
end

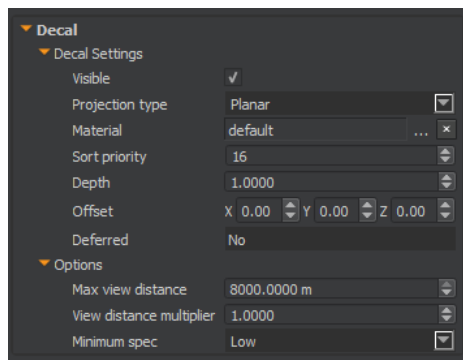
function constraintexample:OnConstraintEnabled()
    Debug.Log("Constraint Enabled: " .. tostring(self.entityId))
end

function constraintexample:OnConstraintDisabled()
    Debug.Log("Constraint Disabled: " .. tostring(self.entityId))
end
```

Decal Component

component entity system is in preview release and is subject to change.

Use the **Decal** component to place a component on an entity.



Decal Component Properties

The **Decal** component has the following properties:

Visible

If selected, shows the decal.

Projection Type

Specifies the type of decal projection: **Planar**, **On Terrain**, or **On Terrain and Static Objects**.

Material

The decal's material file.

Sort Priority

Sort priority relative to other decals in the system.

Depth

Projection depth for deferred decals.

Offset

Allows offsetting the decal relative to the entity's position.

Deferred

If true, enables deferred decals.

Max view distance

The furthest distance at which this decal can be viewed.

View distance multiplier

Multiplier to the automatically computed fade-out camera distance.

Minimum spec

Minimum spec for the decal to be active.

EBus Request Bus Interface

Use the following request function with the EBus interface to communicate with other components of your game.

For more information about using the event bus (EBus) interface, see [Event Bus \(EBus\)](#).

SetVisibility

Sets an explicit value (true/false) on the decal's visibility.

Parameters

true or false

Show

Shows the decal.

Parameters

None

Hide

Hides the decal.

Parameters

None

The following is an example of script using the **Request Bus Interface**.

```
function example:OnActivate()  
    self.decalBusSender = DecalComponentRequestBusSender(self.entityId);  
    self.decalBusSender:Hide();  
    self.decalBusSender:Show();  
    self.decalBusSender:SetVisibility(false);  
end
```

Event Action Binding Component

component entity system is in preview release and is subject to change.

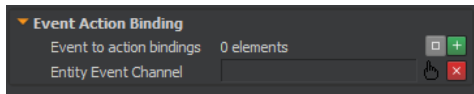
The **Event Action Binding** component converts events to actions. For example, a jump event can be converted to an **Add Physics Impulse** action.

The following actions are available:

- **Rotate Entity Action** – Rotates an entity

- **Add Physics Impulse Action** – Applies an impulse to a character
- **Move Entity Action** – Moves an entity

When you first add the **Event Action Binding** component, it looks like this:

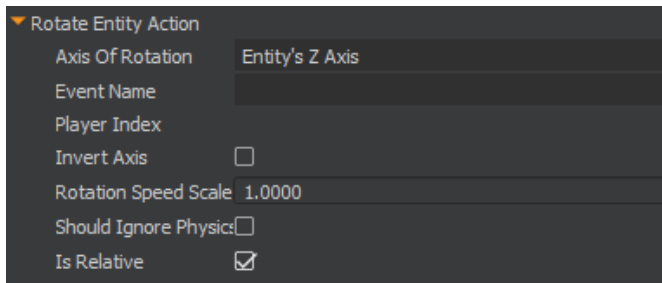


To add an action

1. Click the plus sign (+) next to **Event to action bindings**.
2. Select an action from the list.
3. Click the hand icon next to **Entity Event Channel** and then select an entity in the viewport.

Rotate Entity Action

Use the **Rotate Entity Action** to rotate an entity.



Rotate Entity Action has the following properties:

Axis of Rotation

Select from the list to specify the axis around which the entity rotates.

Event Name

Name (*string*) of the expected event.

Player Index

If selected, entity rotates along a negative axis.

Invert Axis

Scale the incoming direction's **Z** component.

Rotation Speed Scale

Use a value greater than 1 to speed up the entity. Use a value between 0 and 1 to slow it down.

Should Ignore Physics

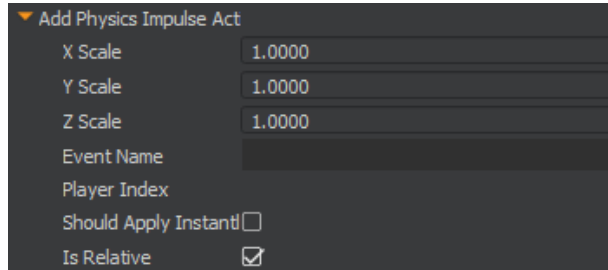
If selected, applies action only to the entity's transform component.

Is Relative

If selected, the entity's transform component forms the basis for the movement.

Add Physics Impulse Action

Use the **Add Physics Impulse Action** to apply an impulse to a character. For example, you can use this action on spacecraft and watercraft and to create jumping actions.



Add Physics Impulse Action has the following properties:

X Scale

Scale the incoming direction's **X** component.

Y Scale

Scale the incoming direction's **Y** component.

Z Scale

Scale the incoming direction's **Z** component.

Event Name

Name (*string*) of the expected event.

Player Index

The index (*u8*) of the player attached to this event.

Should Apply Instantly

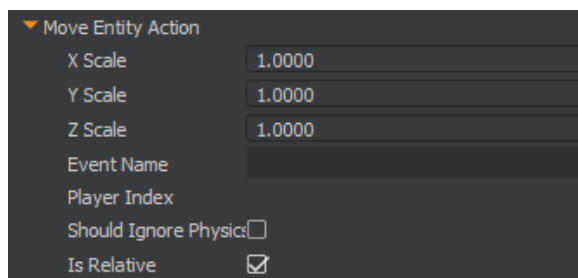
If selected, applies impulse instantly. Leave deselected to send continuous impulses.

Is Relative

If selected, the entity's transform position forms the basis for the movement.

Move Entity Action

Use the **Move Entity Action** to move an entity.



Move Entity Action has the following properties:

X Scale

Scale the incoming direction's **X** component.

Y Scale

Scale the incoming direction's **Y** component.

Z Scale

Scale the incoming direction's **Z** component.

Event Name

Name (*string*) of the expected event.

Player Index

The index (*u8*) of the player attached to this event.

Should Ignore Physics

If selected, applies action only to the entity's transform component.

Is Relative

If selected, the entity's transform position forms the basis for the movement.

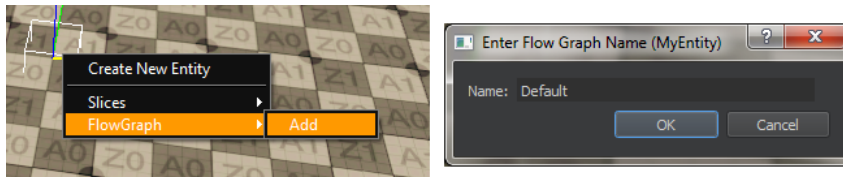
Flow Graph Component

component entity system is in preview release and is subject to change.

Component entities support some flow graphs using the context menu on selected component entities.

The following flow graph nodes are supported for component entities:

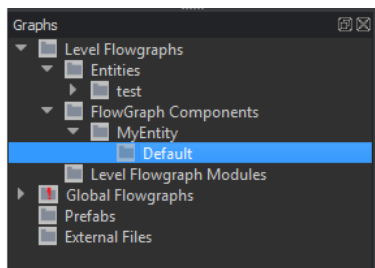
- **Movement:RotateEntity** – Applies a rotation velocity to an entity.
- **Movement:MoveEntityTo** – Moves the entity to the specified location.
- **ComponentEntity:TransformComponent:GetEntityPosition** – Returns the entity's position.
- **ComponentEntity:TransformComponent:GetEntityRotation** – Returns the entity's orientation.
- **ComponentEntity:TransformComponent:SetEntityPosition** – Specifies the entity's position.
- **ComponentEntity:TransformComponent:SetEntityRotation** – Specifies the entity's rotation.
- **ComponentEntity:TriggerComponent:EnterTrigger** – Triggers event notification on entry or exit.



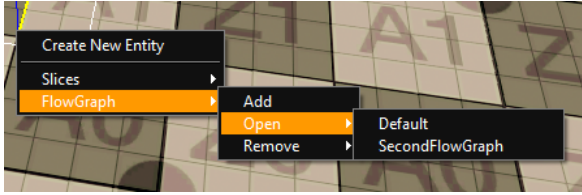
To add a flow graph to a component entity

1. In the viewport select an existing entity.
2. Right-click the entity, and click **FlowGraph, Add**.
3. Type a name for the flow graph.

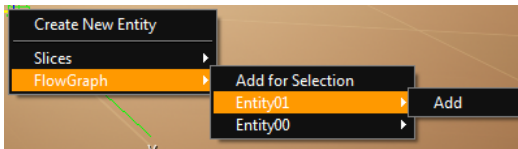
After you add a flow graph to the entity, you can access the flow graph from the **Flow Graph** editor in the **Graphs** pane under **FlowGraph Components**.



Component entities support multiple flow graphs. You can add, remove, or open a flow graph from the viewport using the context menu.



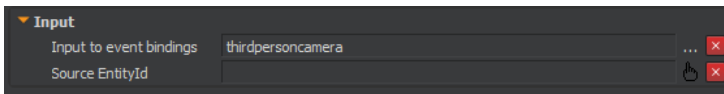
Flow graph context menus also work on multiple selected entities.



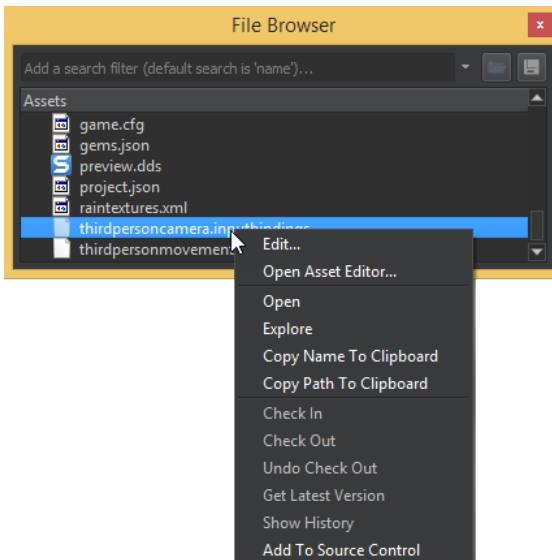
Input Configuration Component

component entity system is in preview release and is subject to change.

The **Input Configuration** component references an `.inputbindings` file. This file binds a set of inputs (such as from a mouse, game controller, and so on) to an event.



You can create or edit these reflected content files by right-clicking on the file in the [File Browser](#) (p. 324). Select an option from the context menu.



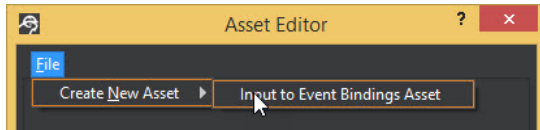
From the **File Browser**, you can also create a new asset.

Creating an Input to Event Binding Asset

Follow these steps to create a new input to event binding asset.

To create a new input to event binding asset

1. Open **File Browser** by clicking **View, Open View Pane, File Browser**.
2. Right-click on an asset, and then click **Open Asset Editor**.
3. Click **File, Create New Asset, Input to Event Binding Asset**.



Input Configuration Properties

The **Input Configuration** component has the following properties:

Input to event bindings

An asset reference to an `.inputbindings` file that defines bindings of raw input to events. Click the ellipsis (...) to select an `.inputbindings` file.

Source EntityId

Name of entity for input. Click the hand icon, and then select an entity in the viewport.

Creating a New Input Bindings File

Follow these steps to create a new `.inputbindings` file. After you create a new `.inputbindings` file, you can add [Input Event Groups \(p. 355\)](#) and [Event Generators \(p. 355\)](#).

To create a new `.inputbindings` file

1. Open the **File Browser** by clicking **View, Open View Pane, File Browser**.
2. Right-click anywhere and then click **Open Asset Editor**.
3. Click **File, Create New Asset, Input to Event Bindings Asset**.
4. Type a file name. Click **Save**.

Input Event Groups

An **Input Event Bindings** file can have zero (0) or more **Input Event Groups**.

To add an input event group

1. Click the plus sign (+) next to **Input Event Groups**.
2. In the **Event Name** box, type a name for your event.

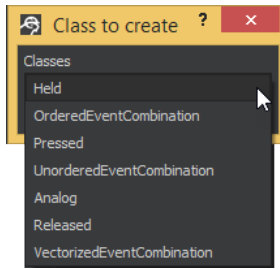
Event Generators

An **Event Generator** is a handler that generates the named event. For example, a pressed key, a held mouse button, or a series of actions on a controller results in the named event.

To add an event generator

1. Click the plus sign (+) next to **Event Generators**.

2. Select an event from the list.



These event generators (`InputSubComponents`) are categorized in the following manner:

Single Event to Action

Maps a single event to a single action. The following event generators are **Single Event to Action**:

- **Analog** – Analog input events such as a mouse or keyboard.
- **Held** – Event that completes when the **Input Device Type** is held for a specified duration.
- **Pressed** – Event that completes when the **Input Device Type** is pressed.
- **Released** – Event that completes when the **Input Device Type** is released.

GameplayNotificationBus handlers

These handlers aggregate one more `GameplayActionEvents` into a single output `GameplayActionEvent`. They do not listen for raw input like the `InputNotificationBus` handlers. The following events are `ActionNotificationBus` handlers:

- **Ordered Event Combination** – Combination input event handler that listens for a series of events and then treats them all as one. As long as the events occur in the specified order, the outgoing event will occur. For example, **Down** then **Right** then **Heavy Punch** results in the event `Heavy Special Attack`. When you add this event generator, it appears in the UI as the *first incoming event name* followed by an ellipsis (...).
- **Unordered Event Combination** – Combination input event handler that listens for a combination of events in no particular order as long as they all happen within a specified amount of time. When you add this event generator, it appears in the UI as **Unordered combo in *n***, where *n* is the value from the property **Max delay for all events**.
- **Vectorized Event Combination** – This class binds three incoming action values to an `AZ::Vector3` and sends out a new gameplay event containing that `AZ::Vector3`.

Bind to this action by inheriting `AZ::ActionNotificationBus<AZ::Vector3>::Handler` and connecting to the bus.

Event Generator Properties

Each event generator has a set of properties that you can use to customize the specifics of the event generator.

Single Event to Action Properties

The **Single Event To Action** event generators (**Held**, **Pressed**, **Analog**, **Released**) all have the following common properties:

- **Input Device Type** – The type of device that generates the input. Select from a list of available devices.

- **Input Name** – List of input options that depend on the selected input device type. For example, if you select `keyboard` for the **Input Device Type**, a list of possible keystrokes appear in this list.
- **Event value multiplier** – Multiplier by which to scale the input value.

The event generators **Held** and **Analog** also have the following unique properties:

Held

- **Duration to hold** – The number of seconds that the input must be held.
- **Invoke once per release** – If selected, event occurs only once for each held instance. If deselected, an event is generated for each duration continuously until released.

Analog

- **Send continuous updates** – If selected, updates are sent continuously. If deselected, sends a message only when the analog value has changed.
- **Dead zone** – A magnitude or absolute value. Values below this number are considered inactive, and no events are generated. Only magnitudes, or absolute values, above this number causes events to be generated.

Action Notification Bus Handlers Properties

The `ActionNotificationBus` handlers (**Ordered Event Combination**, **Unordered Event Combination**, and **Vectorized Event Combination**) aggregate one or more `GameplayActionEvents` into a single output `GameplayActionEvent`. They do not listen for raw input the way the `InputNotificationBus` handlers do.

The `ActionNotificationBus` handlers feature the following properties:

Ordered Event Combination and Unordered Event Combination

- **Incoming event names** – A resizable array of incoming event names, such as `jump` and `run`.
- **Max delay between events** – Delay in seconds between successful events which, if exceeded, causes a failure.

Vectorized Event Combination

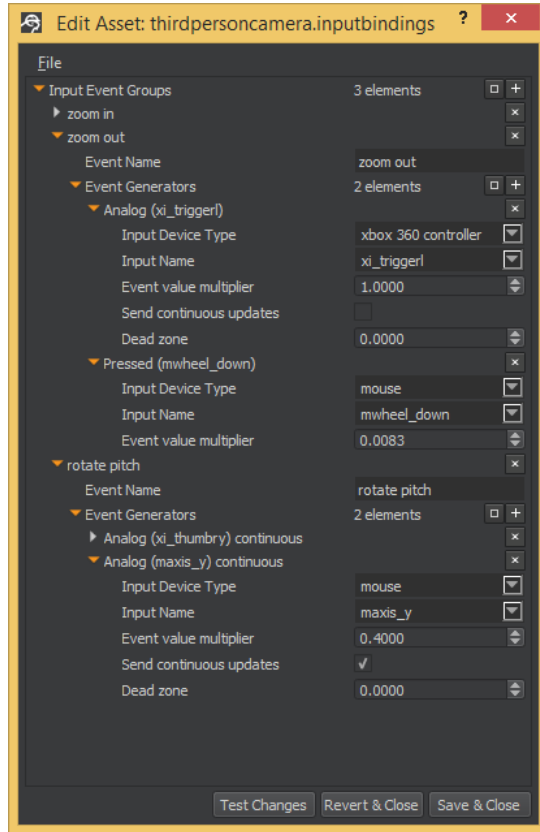
- **Dead zone length** – A threshold for vector length below which an event is not generated.
- **Incoming event names** – An array of three incoming event names, such as `x`, `y`, and `z`, mapped to a vector output.
- **Should normalize** – If selected, output event value is normalized.

Editing the Input Bindings File

Follow these steps to edit the `.inputbindings` file.

To edit the `.inputbindings` file

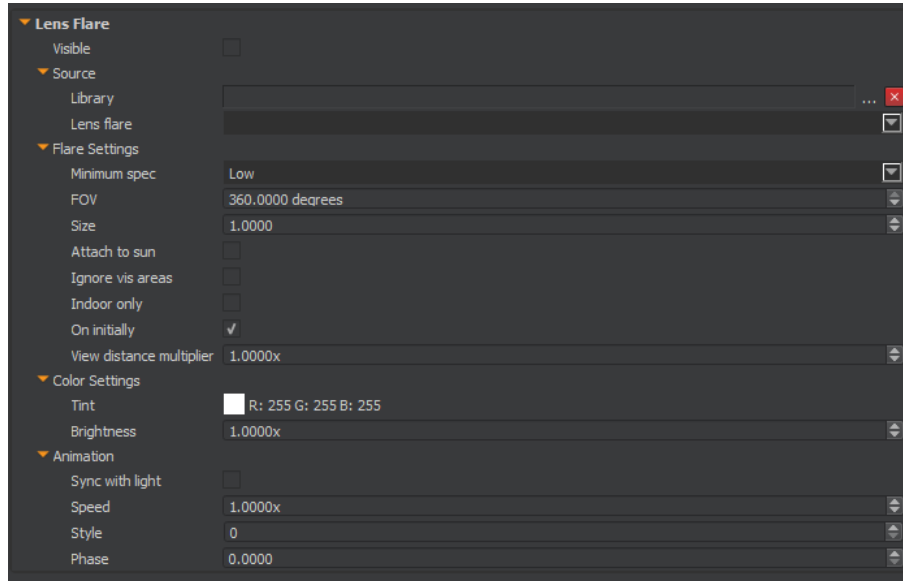
1. Open the **File Browser** by clicking **View, Open View Pane, File Browser**.
2. Right-click the `.inputbindings` file you want to edit. From the context menu, click **Edit**.



Lens Flare Component

component entity system is in preview release and is subject to change.

The Lens Flare component allows the placement of a lens flare on an entity.



Lens Flare Component Properties

The **Lens Flare** component has the following properties:

Visible

If selected, shows the lens flare.

Source

Library

Select a lens flare library that has been authored by the **Lens Flare** editor.

Lens flare

A lens flare selected from the available flares in the lens flare library.

Flare Settings

Minimum spec

The minimum spec at which this lens flare is enabled.

FOV

The field of view (FOV) for this lens flare in degrees.

Size

The size of the lens flare.

Attach to sun

If selected, attaches the lens flare to the sun.

Ignore vis areas

If selected, lens flare ignores vis areas.

Indoor only

If selected, lens flare is only rendered indoors.

On initially

If selected, the lens flare is on when created.

View distance multiplier

Adjust the maximum view distance. For example, `1.0` would use the default and `1.1` would be 10% further than the default.

Color Settings

Tint

Color of the lens flare.

Brightness

Brightness of the lens flare.

Animation

Sync with light

When selected, uses the animation settings of the provided light.

Speed

Multiple of the base animation rate.

Style

Light animation curve ID (style) as it corresponds to values in `Light.cfx`.

Phase

Animation start offset from 0 to 1. 0.1 would be 10% into the animation.

EBus Request Bus Interface

Use the following request function with the EBus interface to communicate with other components of your game.

For more information about using the Event Bus (EBus) interface, see [Event Bus \(EBus\)](#).

- **SetLensFlareState** (`On` or `Off`) – Turns the lens flare on or off.
- **TurnOnLensFlare** – Turns the lens flare on.
- **TurnOffLensFlare** – Turns the lens flare off.
- **ToggleLensFlare** – Toggles the lens flare state (`on` to `off`, or `off` to `on`).

EBus Notification Bus Interface

Use the following notification functions with the EBus interface to communicate with other components of your game.

For more information about using the Event Bus (EBus) interface, see [Event Bus \(EBus\)](#).

- **LensFlareTurnedOn** – Sends a signal when the lens flare is turned on.
- **LensFlareTurnedOff** – Sends a signal when the lens flare is turned off.

The following is an example of script using the **Request Bus Interface**.

```
function example:OnActivate()  
    self.lensFlareBusSender =  
    LensFlareComponentRequestBusSender(self.entityId);  
    self.lensFlareBusSender:SetLensFlareState("Off");  
    self.lensFlareBusSender:TurnLensFlareOn();  
    self.lensFlareBusSender:TurnLensFlareOff();  
    self.lensFlareBusSender:ToggleLensFlare();  
end
```

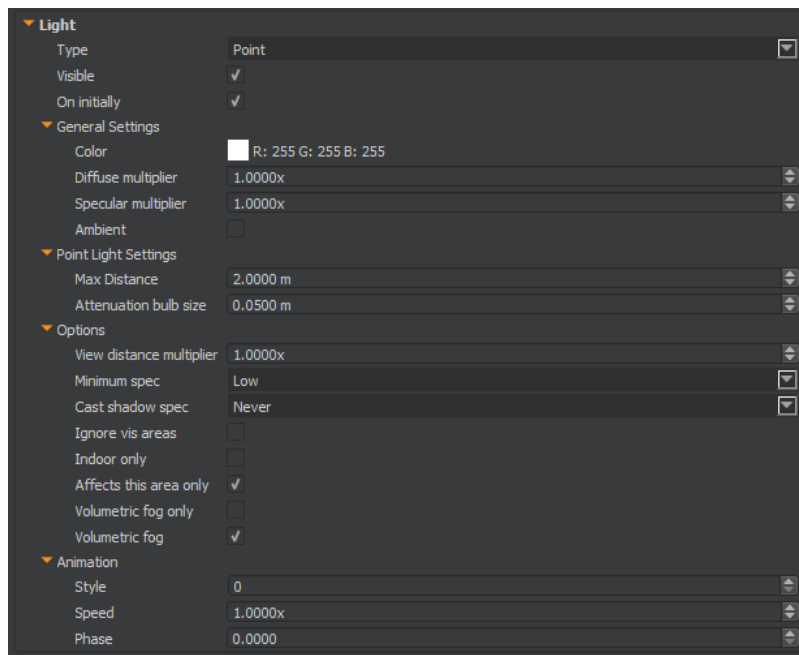
Light Component

component entity system is in preview release and is subject to change.

The **Light** component allows for the placement of a light on an entity.

There are four light types that share a set of common settings and then each have their own specific settings:

- Point
- Area
- Projector
- Environment Probe



Light Component Properties

The **Light** component has the following **common** properties:

Type

Selects the light type: **Point**, **Area**, **Projector**, or **Environment Probe**.

Visible

If selected, shows the light.

On initially

If selected, the light is on when created.

General Settings

Color

The color of this light.

Diffuse multiplier

Controls the strength of the diffuse color.

Specular multiplier

Controls the strength of the specular brightness.

Ambient

If selected, light acts as a multiplier for cubemap values.

Options**View distance multiplier**

Adjust the maximum view distance. For example, `1.0` would use the default and `1.1` would be 10% further than the default.

Minimum spec

The minimum spec at which this light is enabled.

Cast shadow spec

The minimum spec at which shadows are cast.

Ignore vis areas

If selected, light ignores vis areas.

Indoor only

If selected, light is only rendered indoors.

Affects this area only

If selected, light affects only the immediate area.

Volumetric fog only

If selected, affects only volumetric fog.

Volumetric fog

If selected, affects volumetric fog.

Animation**Style**

Light animation curve ID (style) as it corresponds to values in `Light.cfx`.

Speed

Multiple of the base animation rate.

Phase

Animation start offset from 0 to 1. `0.1` would be 10% into the animation.

Each of the light types—**Point**, **Area**, **Projector**, and **Environment probe**—have their own set of properties.

Point Light Settings**Max Distance**

Maximum distance at which this light can be seen.

Attenuation bulb size

Radius in meters before light falloff begins.

Area Settings**Area Width**

Width of the area light in meters.

Area Height

Height of the area light in meters.

Max Distance

Maximum distance in meters that the area light extends.

Projector Light Settings**Max Distance**

Maximum distance in meters that the projector light extends.

Attenuation bulb size

Radius in meters before light falloff begins.

FOV

Projector light's field of view (FOV) in degrees.

Near Plane

Distance of the near project plane to the entity position in meters.

Texture

Projector light's texture file.

Environment Probe Settings**Area X,Y,Z**

The XYZ dimensions of the environment probe's area of effect.

Sort priority

Priority number for probe rendering. The lower priority numbers (for example, 0 or 1) are rendered on top of the higher priority numbers (for example, 100).

Resolution

Cubemap resolution in pixel

Cubemap asset

File path for the cubemap asset.

EBus Request Bus Interface

Use the following request function with the EBus interface to communicate with other components of your game.

For more information about using the event bus (EBus) interface, see [Event Bus \(EBus\)](#).

SetLightState

Turns the light on or off.

Parameters

On or Off

Return

None

Scriptable

Yes

TurnOnLight

Turns the light on.

Parameters

None

Return

None

Scriptable

Yes

TurnOffLight

Turns the light off.

Parameters

None

Return

None

Scriptable

Yes

ToggleLight

Toggles the light state from on to off, or off to on.

Parameters

None

Return

None

Scriptable

Yes

EBus Notification Bus Interface

Use the following notification functions with the EBus interface to communicate with other components of your game.

For more information about using the event bus (EBus) interface, see [Event Bus \(EBus\)](#).

LightTurnedOn

Sends a signal when the light is turned on.

Parameters

None

Return

None

Scriptable

Yes

LightTurnedOff

Sends a signal when the light is turned off.

Parameters

None

Return

None

Scriptable
Yes

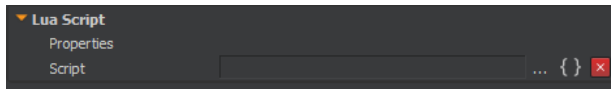
The following is an example of script using the **Request Bus Interface**.

```
function example:OnActivate()
    self.lightBusSender = LightComponentRequestBusSender(self.entityId);
    self.lightBusSender:SetLightState("Off");
    self.lightBusSender:TurnLightOn();
    self.lightBusSender:TurnLightOff();
    self.lightBusSender:ToggleLight();
end
```

Lua Script Component

component entity system is in preview release and is subject to change.

The **Lua Script** component attaches arbitrary Lua logic to an entity in the form of a Lua-based component.



- **Properties** - a
- **Script** - a .lua script file.

Lua Script Component Properties

The **Lua Script** component has the following properties:

Properties

Lua table of user-defined properties that will be reflected and available in the Entity Inspector.

Script

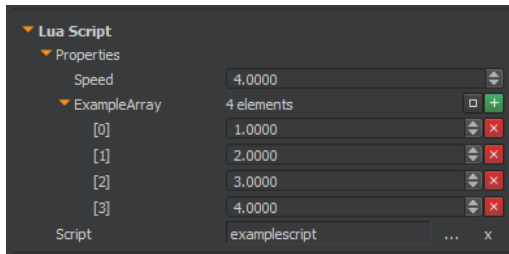
A .lua script file. To edit the script file, click **{ }** (brackets button).

Properties

Scripts support user-defined properties. These properties are reflected on the component. The following script sample shows properties defined in a .lua file.

```
MyScriptComponent = {
    Properties = {
        Speed = {
            default = 0,    -- Supports numbers, string or boolean
            min = 0,
            max = 100,
            step = 1,
            description = "Speed in m/s for the ...",
        }
        ExampleArray = {1,2,3,4,5}, -- default array type no attributes
    }
}
```

The user-defined properties are available to edit in the entity inspector.



Network Binding

You can accomplish networking binding for properties by adding the following table to the description of the variable inside of the **Properties** table.

```
MyScriptComponent = {
    Properties = {
        Speed = {
            default = 0,    -- Supports numbers, string or boolean
            min = 0,
            max = 100,
            step = 1,
            description = "Speed in m/s for the ...",

            -- If this table is missing, it is assumed the value is not
            networked.

            netSynched =
            {
                -- Optional fields
                OnNewValue = <function> -- OnNewValue will be called
                whenever the property has a new value.

                -- The following flags are mainly here for debugging/
                profiling niceness.
                Enabled = true           -- Will control whether or not
                the field is network enabled, if missing, will assume true.
                ForceIndex = [1..32]    -- Profiling helper tool to force
                a property to use a specific DataSet to make understanding what data is
                being used where easier.
            }
        }
    }
}
```

After you enter the table, you can use the the property as you normally would, but any changes made to it will be reflected across the network.

Exposing RPCs to scripts involves creating a new table inside of the component table, but outside of the properties table, as shown in the following.

```
MyScriptComponent = {
    Properties = {
        Speed = {
            default = 0,    -- Supports numbers, string or boolean
            min = 0,
            max = 100,
            step = 1,
```



```
        description = "Speed in m/s for the ...",
        netSynched =
        {
            OnNewValue = <function>
            ForceIndex = 1
        }
    }
}

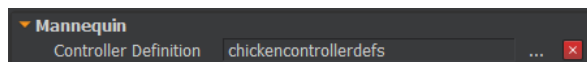
-- Table of RPCs the script wants to implement
NetRPCs =
{
    RPCNoParam = {
        OnMaster = <function> -- The function that will be called on
the Master Script. This should return a bool value indicating whether or not
Proxy components can execute the RPC on themselves. This must exist.
        OnProxy = <function> -- The function taht will be called on
the Proxy Script. This is optional and can be excluded if the master never
allows Proxy's to execute the function call.
    }
}
}
```

You can invoke the RPC just like any other function inside of the NetRPCs table. There is no need to specify the OnMaster/OnProxy from the calling script.

```
self.NetRPCs.RPCNoParam();
self.NetRPCs.RPCParam(1.0);
```

Mannequin Component

The **Mannequin** component animates a component entity using the [Mannequin System \(p. 247\)](#). This component works in conjunction with the [Mannequin Scope Context \(p. 373\)](#) component, which sets scope context. Using the mannequin scope context component is optional; as long as the appropriate scope context is set, the mannequin component functions as designed. The mannequin component simply acts as the programmer- and designer-facing interface for component entities with respect to mannequin.



Mannequin Component Properties

The **Mannequin** component has the following property:

Controller Definition

Path to the [controller definition file \(p. 247\)](#) to be used for animation.

EBus Request Bus Interface (Per Fragment)

Use the following request functions with the EBus interface to communicate with other components of your game.

For more information about using the Event Bus (EBus) interface, see [Event Bus \(EBus\)](#).

The following methods modify how a specific fragment on this component is played. Specific fragments are identified using a fragment ID (`RequestId`) that the `QueueFragment` method returns.

QueueFragment

Queues the indicated mannequin fragment.

Parameters

`priority` – Higher numbers indicate higher priority.

`fragmentName` – Name of the fragment to be played.

`fragTags` – Fragment tags to be applied (for multiple FragTags, use a + delimited list).

Return

`requestId` – ID used to uniquely identify and make modifications to this request.

Scriptable

Yes

QueueFragmentById

Queues the indicated mannequin fragment.

Parameters

`priority` – Higher numbers indicate higher priority.

`fragmentId` – ID of the fragment to be played.

`fragTags` – Fragment tags to be applied (For multiple FragTags, use a + delimited list).

Return

`requestId` – ID used to uniquely identify and make modifications to this request.

Scriptable

No

GetActionForRequestId

Allows users to retrieve the action associated with any given request ID.

Parameters

`requestID` – The request ID.

Return

`action` – ID associated with a fragment request.

Scriptable

No

StopRequest

Stops the actions associated with an indicated request.

Parameters

`requestID` – The request ID.

Return

`action` – ID associated with a fragment request.

Scriptable

Yes

GetRequestStatus

Indicates the status of a request.

Parameters

`requestID` – The request ID.

Return

Status (type `IAction::EStatus`) of the request.

Scriptable

Yes

ForceFinishRequest

Forces the actions associated with an indicated request to finish.

Parameters

`requestID` – The request ID.

Return

None

Scriptable

Yes

SetRequestSpeedBias

Sets speed bias for the actions associated with an indicated request.

Parameters

`requestID` – The request ID.

`speedBias` – The speed bias for this animation.

Return

None

Scriptable

Yes

GetRequestSpeedBias

Gets the speed bias for the actions associated with an indicated request.

Parameters

`requestID` – The request ID.

Return

Speed bias for the indicated request.

Scriptable

Yes

SetRequestAnimWeight

Sets the anim weight for the actions associated with an indicated request.

Parameters

`requestID` – The request ID.

`animWeight` – The weight for this animation.

Return

None

Scriptable

Yes

GetRequestAnimWeight

Gets the anim weight for the actions associated with an indicated request.

Parameters

`requestID` – The request ID.

Return

Anim weight for the indicated request.

Scriptable

Yes

EBus Request Bus Interface (Per Component)

The following methods modify how all fragments on this component are played.

PauseAll

Pauses all actions being managed by this mannequin component.

Parameters

None

Return

None

Scriptable

Yes

ResumeAll

Resumes all actions being managed by this mannequin component.

Parameters

A flag of type `IActionController::EResumeFlags` that indicates how the animations are to be resumed.

Return

None

Scriptable

Yes

SetTag

Sets indicated tag for this mannequin component.

Parameters

`tagName` – Name of the tag to be set.

Return

None

Scriptable

Yes

SetTagById

Sets indicated tag for this mannequin component.

Parameters

`tagId` – ID of the tag to be set.

Return

None

Scriptable

Yes

ClearTag

Clears indicated tag for this mannequin component.

Parameters

`tagName` – Name of the tag to be cleared.

Return

None

Scriptable

Yes

ClearTagById

Clears indicated tag for this mannequin component.

Parameters

`tagId` – ID of the tag to be cleared.

Return

None

Scriptable

Yes

SetGroupTag

Sets a tag in the indicated group.

Parameters

`groupName` – Name of the group.

`tagName` – Name of the tag to be set.

Return

None

Scriptable

Yes

SetGroupTagById

Sets a tag in the indicated group.

Parameters

`groupId` – Id of the group.

`tagId` – ID of the tag to be set.

Return

None

Scriptable

No

ClearGroup

Clears tags for the indicated group.

Parameters

`groupName` – Name of the group.

Return

None

Scriptable

Yes

ClearGroupById

Clears tags for the indicated group.

Parameters

`groupId` – Id of the group.

Return

None

Scriptable

No

SetScopeContext

Sets the scope context for this animation controller.

Parameters

`scopeContextName` – Name of the scope context that the `.adb` file is to be attached to.

`entityId` – Id of an entity whose character instance will be bound to this scope context.

`animationDatabase` – Path to the animation database file.

Return

None

Scriptable

Yes

SetScopeContextById

Sets the scope context for this animation controller.

Parameters

`scopeContextID` – ID of the scope context that the `.adb` file is to be attached to.

`entityId` – Id of an entity whose character instance will be bound to this scope context.

`animationDatabase` – Path to the animation database file.

Return

None

Scriptable

No

ClearScopeContext

Clears the indicated scope context.

Parameters

`scopeContextName` – Name of the scope context that is to be cleared.

Return

None

Scriptable

Yes

ClearScopeContextById

Clears the indicated scope context.

Parameters

`scopeContextId` – Id of the scope context that is to be cleared.

Return

None

Scriptable

No

GetActionController

Allows users to retrieve the action controller attached to this instance of the mannequin component.

Parameters

None

Return

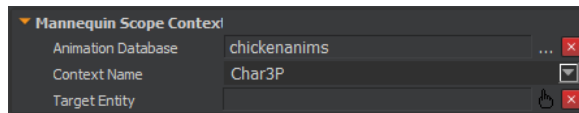
The action controller being used by this mannequin component.

Scriptable

No

Mannequin Scope Context Component

The **Mannequin Scope Context** component associates a runtime character instance with a given scope context and an `.adb` file. This component is used in conjunction with, and cannot function without the [Mannequin](#) (p. 367) component. The **Mannequin** component can, however, use other means to set scope contexts and is therefore able to function without the mannequin scope context component.



Mannequin Scope Context Component Properties

The **Mannequin Scope Context** component has the following properties:

Animation Database

Asset reference to an `.adb` file. Animation database files tie together most of the mannequin configuration.

Context Name

Name of the scope context that the `.adb` file is to be attached to.

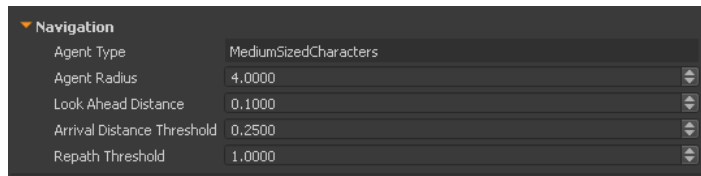
Target Entity

Reference to an entity whose character instance will be bound to this scope context.

Navigation Component

component entity system is in preview release and is subject to change.

The **Navigation** component provides basic pathfinding and pathfollowing services to an entity. It supports AI and other game logic by accepting navigation commands and dispatching per-frame movement requests to the physics component in order to follow the calculated path.



This works by scheduling asynchronous pathfinding requests to the navigation system for finding paths to target entities or positions. Once a valid path has been found, it informs all interested parties. It is up to the requester (or other interested parties) to then tell it to commit to that path and move its entity.

This component is not responsible for assessing the tactical viability of any pathfinding or pathfollowing request that it is given. Instead, it assumes that the requester has already made the requisite tactical decisions before issuing the movement request. The requester (or other interested parties) receives a notification when a path is found and includes is the potential for some additional validation before the path is actually traversed. This should be looked at as more of a screening opportunity than true tactical decision-making point. It mainly serves to ensure that the path is still fresh when the entity starts to move along it.

Navigation Component Properties

The **Navigation** component has the following properties:

AgentType

Type of the entity for navigation purposes. This type is used to select which [navigation mesh](#) (p. 1375) this entity follows in a scenario where there are different navmeshes for larger vehicles and smaller humanoid bots.

AgentRadius

Radius of this entity for navigation purposes. Independent of physics or any other collision concerns, this value is used by the pathfinder for moving around in an area with obstacles while cutting corners..

LookAheadDistance

Defines the distance between the points that an entity walks over while following a given path.

Arrival Distance Threshold

Entity's minimum distance from the end point before its movement is to be stopped and considered complete.

Repath Threshold

Entity's minimum distance from its previously known location before a new path is calculated.

EBus Request Bus Interface

Use the following request functions with the event bus (EBus) interface to communicate with other components of your game.

For more information about using the EBus interface, see [Event Bus \(EBus\)](#).

FindPath

Finds a requested path configuration.

Parameters

`request` – Allows the issuer of the request to override one, all, or none of the pathfinding configuration defaults for this entity.

Return

A unique identifier to this pathfinding request.

Scriptable

No

FindPathToEntity

Creates a path finding request to navigate towards the specified entity.

Parameters

`EntityId` of the entity toward which you want to navigate.

Return

A unique identifier to this pathfinding request.

Scriptable

Yes

Stop

Stops all pathfinding operations for the provided `requestId`. The ID is used to ensure that the request being canceled is the request that is currently being processed. If the `requestId` provided is different from the ID of the current request, then the stop command is ignored.

Parameters

`requestId` – ID of the request to be canceled.

Return

None

Scriptable

Yes

EBus Notification Bus Interface

Use the following notification functions with the event bus (EBus) interface to communicate with other components of your game.

For more information about using the EBus interface, see [Event Bus \(EBus\)](#).

OnSearchingForPath

Indicates that the pathfinding request has been submitted to the navigation system.

Parameters

`requestId` – ID of the path search request.

Return

None

Scriptable
Yes

OnPathFound

Indicates that a path has been found for the indicated request.

Parameters

`requestID` – ID of the found request for the path search

`currentPath` – The path that was calculated by the pathfinder.

Return

Flag indicating whether this path is to be traversed or not.

Scriptable
No

OnTraversalStarted

Indicates that traversal for the indicated request has started.

Parameters

`requestId` – ID of the request for which traversal has started.

Return

None

Scriptable
Yes

OnTraversalInProgress

Indicates that traversal for the indicated request is in progress.

Parameters

`requestId` – ID of the request for which traversal is in progress.

Return

None

Scriptable
Yes

OnTraversalComplete

Indicates that traversal for the indicated request has completed successfully.

Parameters

`requestId` – ID of the request for which traversal has finished.

Return

None

Scriptable
Yes

OnTraversalCancelled

Indicates that traversal for the indicated request was canceled before successful completion.

Parameters

`requestId` – ID of the request for which traversal was canceled.

Return

None

Scriptable

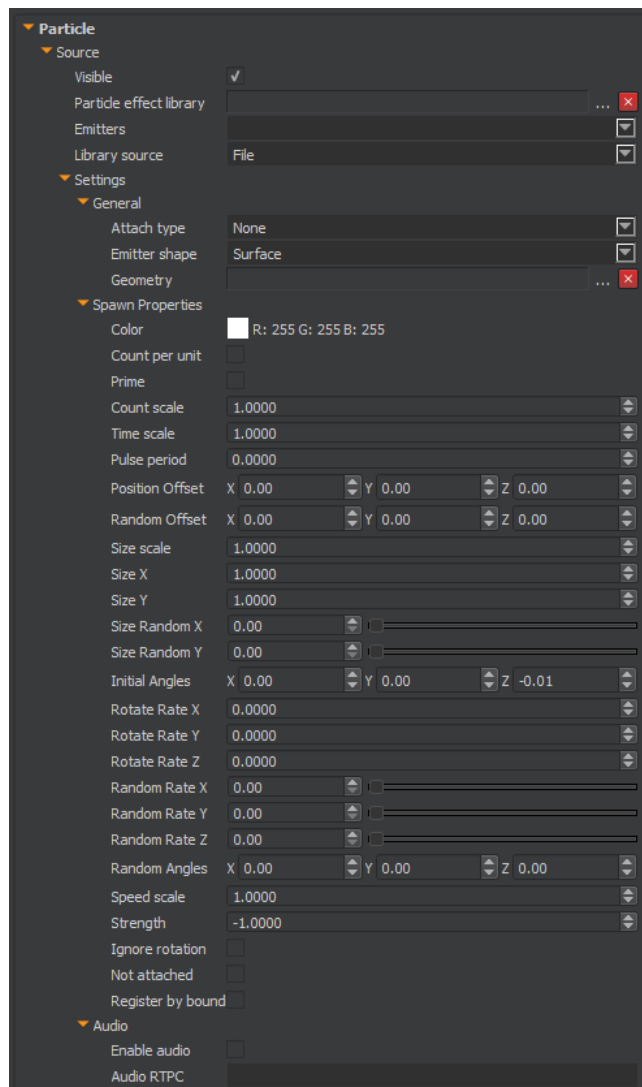
Yes

Particle Component

component entity system is in preview release and is subject to change.

The **Particle** component allows the placement of a single particle emitter on an entity. However, an entity can contain multiple particle components.

Particles can come from either a file library or the level library. If the **Library source** is set to **Level**, the **Particle effect library** field is removed, and any emitters in the level library will be available in the **Emitters** list. If no emitters are listed, you can open the [Particle Editor \(p. 929\)](#) to create some.



Particle Component Properties

The Particle component has the following properties:

Source

Visible

If selected, renders the emitter.

Particle effect library

Select the particle effect library.

Emitters

After specifying a particle effect library, select an emitter from the list.

Library source

File or Level.

General

Attach type

Select the type of object from which the particles are emitted (**None**, **Bounding box**, **Physics**, **Render**).

Attach form

Select the shape aspect from which particles are emitted (**Surface**, **Edges**, **Vertices**, **Volume**).

Spawn Properties

Count per unit

If selected, multiplies particle count also by geometry extent (length, area, volume).

Prime

If selected, sets emitter to behave as though it has been running indefinitely.

Count scale

Multiple for particle count (on top of **Count per unit**, if set).

Time scale

Multiple for emitter time evolution.

Pulse period

How often to restart emitter.

Size scale

Multiple for all effect sizes.

Speed scale

Multiple for particle emission speed.

Strength

Parameter strength curves.

Ignore rotation

If select, entity's rotation is ignored.

Not attached

If selected, the entity's position is ignored. Emitter does not follow its entity.

Register by bounding box

If selected, uses the bounding box instead of position to register in vis area.

Audio

Enable audio

If selected, enables audio.

Audio RTPC

Indicates what audio RTPC this particle effect instance drives.

EBus Request Bus Interface

Use the following request function with the EBus interface to communicate with other components of your game.

For more information about using the event bus (EBus) interface, see [Event Bus \(EBus\)](#).

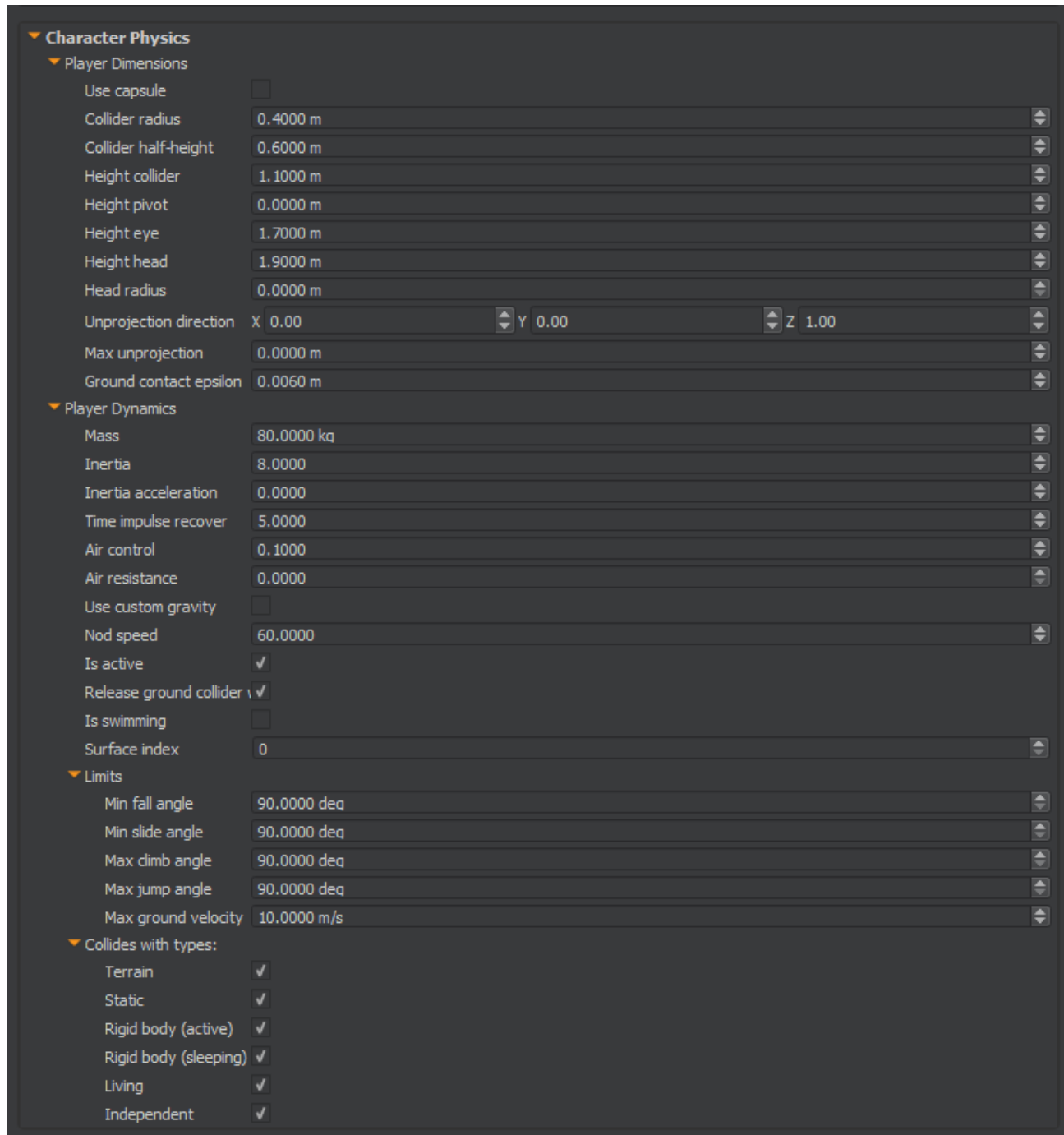
- **SetVisibility** (*true* or *false*) – Sets an explicit value for emitter visibility.
- **Show** – Shows the emitter.
- **Hide** – Hides the emitter.

The following is an example of script using the **Request Bus Interface**.

```
function example:OnActivate()  
    self.particleBusSender =  
    ParticleComponentRequestBusSender(self.entityId);  
    self.particleBusSender:Hide();  
    self.particleBusSender:Show();  
    self.particleBusSender:SetVisibility(false);  
end
```

Character Physics Component

The **Character Physics** component adds physical behavior to and configures simulation characteristics for character entities, such as players and enemies.



Character Physics Component Properties

Player Dimensions

Use capsule

When selected, uses capsule collider geometry. When not selected, uses cylinder collider geometry.

Collider radius

Radius of collision for the cylinder or capsule geometry.

Collider half-height

Half-height of straight section of collision for the cylinder or capsule geometry.

Height collider

Vertical offset of collision geometry center.

Height pivot

Offset from the central ground position that is considered entity center.

Height eye

Vertical offset of the camera.

Height head

Center of the head geometry.

Head radius

Radius of the head geometry that is used for the camera offset.

Unprojection direction

Unprojection direction to test in case the new position overlaps with the environment. For **Auto**, enter 0.

Max unprojection

Maximum allowed unprojection.

Ground contact epsilon

The amount that the living entity needs to move upwards before ground contact is lost.

Player Dynamics**Mass**

Mass in kg.

Inertia

Inertia coefficient. For no inertia, enter 0.

Inertia acceleration

Inertia felt on acceleration.

Time impulse recover

Duration after which inertia is forcefully turned on after receiving an impulse.

Air control

Air control coefficient. Values from 0.00 to 1.00.

Air resistance

Standard air resistance.

Use custom gravity

When selected, uses custom gravity. When not selected, uses world gravity.

Nod speed

Vertical camera shake speed after landings.

Is active

If not selected, disables all simulation for the character, except moving along the requested velocity.

Release ground collider

If selected, and the living entity is not active, the ground collider (if present) is explicitly released during the simulation step.

Is swimming

If selected, the entity can swim and is not bound to the ground plane.

Surface index

Surface identifier for collisions.

Limits

Min fall angle

Minimum angle of slope at which the entity starts falling.

Min slide angle

Minimum angle of slope at which entity starts sliding.

Max climb angle

Maximum angle of slope that the entity can climb.

Max jump angle

Maximum ground slope angle that entity can jump towards.

Max ground velocity

Maximum surface velocity on which entity can stand.

Collides with type**Terrain**

If selected, entity can collide with the terrain.

Static

If selected, entity can collide with static entities.

Rigid body (active)

If selected, entity can collide with active rigid bodies.

Rigid body (sleeping)

If selected, entity can collide with sleeping rigid entities.

Living

If selected, entity can collide with other living entities.

Independent

If selected, entity can collide with independent entities.

EBus Request Bus Interface – PhysicsComponentRequestBus

Use the following request functions with the EBus interface to communicate with other components of your game.

For more information about using the Event Bus (EBus) interface, see [Event Bus \(EBus\)](#).

EnablePhysics

Makes the entity a participant in the physics simulation.

Parameters

None

Return

None

Scriptable

Yes

EBus Request Bus Interface – CryPhysicsComponentRequestBus

Use the following request functions with the EBus interface to communicate with other components of your game.

For more information about using the Event Bus (EBus) interface, see [Event Bus \(EBus\)](#).

GetPhysicsParameters

Passes in any class that inherits from `pe_params` to retrieve them. For example, `pe_params_pos`.

Parameters

(output) `pe_params&`

Return

None

Scriptable

No

SetPhysicsParameters

Passes in any class that inherits from `pe_params` to set them. For example, `pe_params_pos`.

Parameters

`const pe_params&`

Return

None

Scriptable

No

GetPhysicsStatus

Passes in any class that inherits from `pe_status` to retrieve them. For example, `pe_status_pos`.

Parameters

(output) `pe_status&`

Return

None

Scriptable

No

ApplyPhysicsAction

Passes in any class that inherits from `pe_action` to set them. For example, `pe_action_impulse`.

Parameters

`const pe_action&`, `bool threadSafe`

Return

None

Scriptable

No

EBus Request Bus Interface – CryCharacterPhysicsRequestBus

Use the following request functions with the EBus interface to communicate with other components of your game.

For more information about using the Event Bus (EBus) interface, see [Event Bus \(EBus\)](#).

Move

Requests movement from living entity.

param `velocity` – Requests velocity (direction and magnitude).

param `jump` – Controls how velocity is applied within living entity. See `physinterface.h`, `\ref pe_action_move::iJump` for more details.

Parameters

`const AZ::Vector3& velocity, int jump`

Return

None

Scriptable

No

The following is an example of script using the **Request Bus Interface**.

```
self.physicsSender = CryCharacterPhysicsRequestBusSender(self.entityId);
self.transformBusSender = TransformBusSender(self.entityId);

local notJumping = 0;
local transform = self.transformBusSender:GetWorldTM();
local velocity = (transform :GetColumn(1) * deltaTime*
    self.Properties.MoveSpeed);
self.physicsSender:Move(velocity, notJumping);
```

Physics Component

The physics component provides a way to add physical behavior to an entity and configure simulation characteristics.

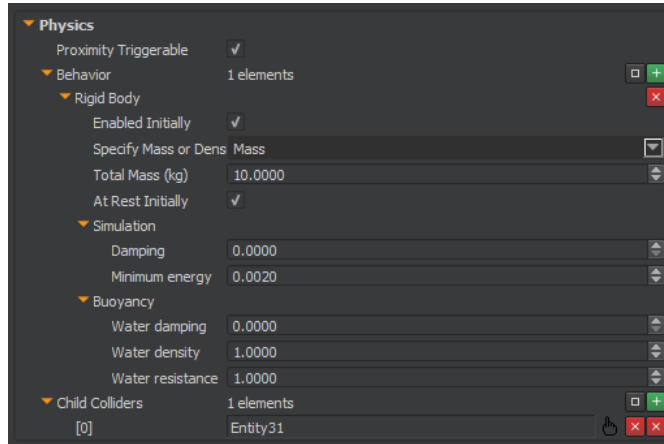
After adding the physics component to an entity, you will need to select a behavior for the physics component and then add a physics collider. Support behaviors include the following:

- **Rigid Body** - Rigid bodies are collidable objects that behave dynamically according to their simulation settings. An example of a rigid body is a ball.
- **Static Body** - Static bodies are collidable objects that do not move around in the world. An example of a static body is a wall.

Rigid Body

Rigid bodies can be moved as a result of physical interactions or through other means such as Flow Graph, Track View, or Lua script, for example.

A rigid body can behave like a static body if it has a mass of zero.



The **Physics Rigid Body** component has the following properties:

Proximity Triggerable

If selected, this entity can interact with proximity triggers.

Behavior

Enabled Initially

If selected, this entity is enabled when the physics simulation starts.

Specify Mass or Density

If **Mass** is selected, specify the total mass.

- **Total Mass (kg)** – Mass of the entity

If **Density** is selected, the density is calculated at spawn based on the density and volume of the entity.

- **Density (kg/cubic meter)** – Mass (kg) per cubic meter of the mesh's volume. Total mass of the entity is calculated at spawn. For example, water's density is 1000 kg/cubic meter.

At Rest Initially

If selected, the entity remains at rest after spawn until agitated. If unselected, the entity falls after spawn.

Simulation

Damping

Uniform damping value applied to the object's movement.

Minimum energy

The energy threshold under which the object goes to sleep.

Buoyancy

Water damping

Uniform damping value applied while in water.

Water density

Multiplier for water density.

Water resistance

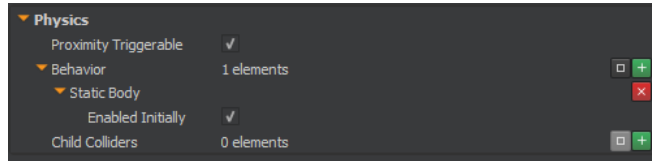
Multiplier for water resistance.

Child Colliders

Entity uses colliders from itself as well as the listed entities.

Static Body

Static bodies represent unmovable objects that other physical entities can collide with.



The **Physics Static Body** component has the following properties:

Proximity Triggerable

If selected, this entity can interact with proximity triggers.

Enabled Initially

If selected, this entity is enabled when the physics simulation starts.

Child Colliders

Entity uses colliders from itself as well as the listed entities.

EBus Request Bus Interface

Use the following request functions with the EBus interface to communicate with other components of your game.

For more information about using the event bus (EBus) interface, see [Event Bus \(EBus\)](#).

- **EnablePhysics** – Enables the physics simulation for this component.
- **DisablePhysics** – Disables the physics simulation for this component.

EBus Notification Bus Interface

Use the following notification functions with the EBus interface to communicate with other components of your game.

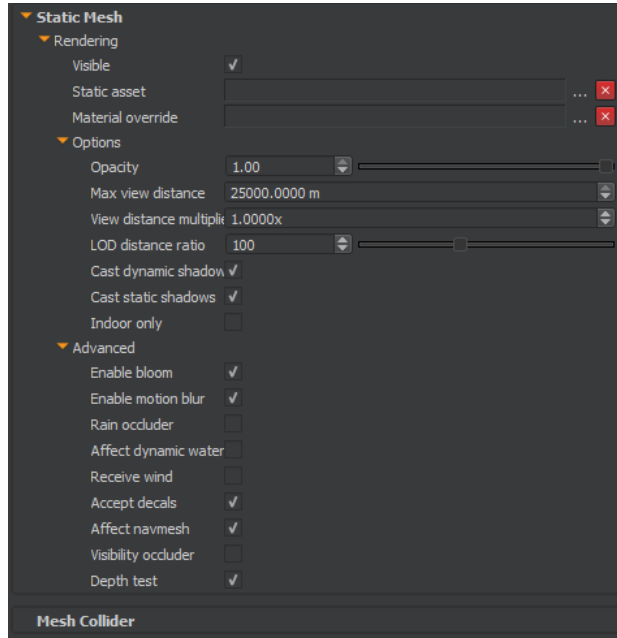
For more information about using the Event Bus (EBus) interface, see [Event Bus \(EBus\)](#).

- **OnPhysicsEnabled** – Event sent after the physics for this component have been enabled..
- **OnPhysicsDisabled** – Event sent after the physics for this component have been disabled.

Mesh Collider Component

component entity system is in preview release and is subject to change.

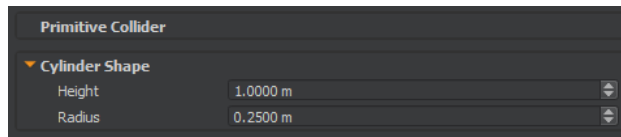
Physics colliders are used to define the shape around entities that collision detection and response takes place. The **Mesh Collider** component specifies that the collider geometry is provided by a mesh component. When you add a mesh collider, the [Static Mesh \(p. 402\)](#) component is also automatically added; specify the properties of the collider in the static mesh component. The mesh collider has no properties of its own.



Primitive Collider Component

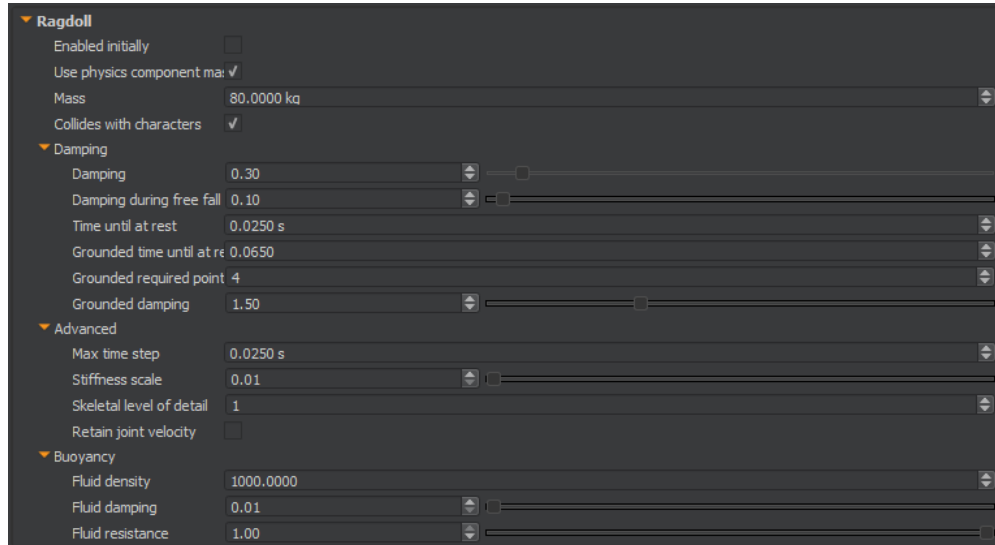
component entity system is in preview release and is subject to change.

Physics colliders are used to define the shape around entities that collision detection and response takes place. The **Primitive Collider** component specifies that the collider geometry is provided by a Cylinder Shape component. When you add a primitive collider, the cylinder shape component is also automatically added; specify the properties of the collider in the cylinder shape component. The primitive collider has no properties of its own.



Rag Doll Component

The **Rag Doll** component uses physics to drive characters. This component is the ideal alternative to animation for simulating environmental effects upon unconscious characters. To use the rag doll component, you need assets that were authored in external 3D modeling programs.



Rag Doll Component Properties

The **Rag Doll** component has the following properties:

Enabled initially

When selected, the entity starts as a rag doll.

Use physics component mass

When selected, the entity attempts first to use mass set by a physics component. If not selected, or no component is found, defaults to **Mass**.

Mass

Simulated mass for the entity. Its use is determined by the **Use physics component mass** setting.

Collides with characters

When selected, the entity collides with characters.

The following properties affect the damping of the entity. Damping is defined as a reduction in the amplitude of an oscillation or vibration.

Damping

Amount of physical force applied against the energy in the system to drive the entity to rest.

Damping during free fall

Amount of damping applied while in the air.

Time until at rest

Time without applied forces before physics is deactivated for this entity.

Grounded time until at rest

Amount of time the entity is on the ground before physics is deactivated for this entity.

Grounded required points of contact

The required number of contact points before the entity is considered grounded.

Grounded damping

Damping applied while grounded.

The following are **Advanced** properties for the rag doll component.

Max time step

Maximum time between steps for the physics simulation for this entity.

Stiffness scale

The amount of stiffness to apply to the joints.

Skeletal level of detail

Level of detail to apply to the entity. Default is 1, the lowest level of detail you can achieve.

Retain joint velocity

When selected, joint velocities are conserved at the instant of ragdolling.

The following properties control entity's **Buoyancy**—how it behaves in water or other liquids or fluids.

Fluid density

The density (kg per cubic meter) of the rag doll for fluid displacement.

The default is 1,000, which is the approximate density of water at 1 atmosphere. By contrast, platinum is approximately 22,000 kg per cubic meter.

Fluid damping

The amount of damping applied while the entity is in fluid.

Fluid resistance

The amount of resistance applied while the entity is in fluid.

EBus Request Bus Interface

Use the following request functions with the EBus interface to communicate with other components of your game.

For more information about using the Event Bus (EBus) interface, see [Event Bus \(EBus\)](#).

EnterRagdoll

Disables current physics and enables rag doll physics on an entity with a skinned mesh component.

Parameters

None

Return

None

Scriptable

Yes

ExitRagdoll

Disables rag doll physics and reenables the entity's physics component.

Parameters

None

Return

None

Scriptable

Yes

The following is an example of script using the **Request Bus Interface**.

```
self.ragdollSender = RagdollPhysicsRequestBusSender(self.entityId);  
  
self.ragdollSender:EnterRagdoll();  
self.ragdollSender:ExitRagdoll();
```

Shapes Components

The **Shape** components provide generic shape facilities for components that use shapes. The shape component helps to define the following:

- [Trigger \(p. 406\) Volumes](#)
Use shapes as volumes to specify triggering bounds.
- [Collision \(p. 386\) Volumes](#)
Use shapes as volumes to specify collider bounds.
- [Audio Area Ambiances \(p. 329\)](#)
Uses shapes as volumes in which a reverb is applied.
- [Audio Areas \(p. 332\)](#)
Uses shapes as volumes in which a sound plays.

Only one shape component can be attached to any particular entity. If you need more than one shape on a single entity, you can create child entities, then add shape and components to them.

Each shape component provides a generic 'ShapeService' that exposes functionality common to all shapes. Each shape also provides a more specific service, such as 'BoxShapeService' and 'SphereShapeService'.

The **Shapes** component includes the following shapes and its properties:

- **Cylinder Shape** – Define **Height** and **Radius** in meters.
- **Capsule Shape** – Define **Height** and **Radius** in meters.
- **Box Shape** – Define dimensions **X**, **Y**, and **Z** in meters.
- **Sphere Shape** – Define **Radius** in meters.

EBus Request Bus Interface

Use the following request functions with the EBus interface to communicate with other components of your game.

For more information about using the Event Bus (EBus) interface, see [Event Bus \(EBus\)](#).

GetShapeType

Allows users to fetch the type of shape that this component is using.

Parameters

None

Return

AZ::Crc32(shape_name)

For example: `AZ::Crc32("Box") | AZ::Crc32("Sphere") | AZ::Crc32("Capsule") | AZ::Crc32("Cylinder")`

Scriptable

Yes

GetEncompassingAabb

Returns an ABB that encompasses this entire shape.

Parameters

None

Return

AZ::Aabb that encompasses the shape

Scriptable

No

IsPointInside

Checks if a given point is inside a shape or outside it.

Parameters

point `Vector3` – The point to be checked

Return

bool indicating whether the point is inside or outside

Scriptable

Yes

ComponentRequestsBus

Each shape component has its own specific Ebus that can be used to access and use the services of that particular shape. All these buses are similar to each other and differ only in the types being serviced.

BoxShapeComponentRequestsBus

Fetches the configuration of the BoxShape.

Name

GetBoxConfiguration

Parameters

None

Return

BoxShapeConfiguration object which provides access to the box configuration

Scriptable

Yes

SphereShapeComponentRequestsBus

Fetches the configuration of the SphereShape.

Name

GetSphereConfiguration

Parameters

None

Return

`SphereShapeConfiguration` object which provides access to the sphere configuration

Scriptable

Yes

CapsuleShapeComponentRequestsBus

Fetches the configuration of the CapsuleShape.

Name

`GetCapsuleConfiguration`

Parameters

None

Return

`CapsuleShapeConfiguration` object which provides access to the capsule configuration

Scriptable

Yes

CylinderShapeComponentRequestsBus

Fetches the configuration of the CylinderShape.

Name

`GetCylinderConfiguration`

Parameters

None

Return

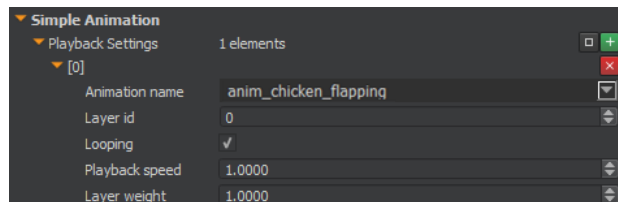
`CylinderShapeConfiguration` object which provides access to the cylinder configuration

Scriptable

Yes

Simple Animation Component

The **Simple Animation** component provides basic animation functionality for the entity. If the entity has a mesh component with a skinned mesh attached (`.chr` or `.cdf` file), the **Simple Animation** component provides a list of all valid animations as specified in the associated `.chrparams` file. The **Simple Animation** component does not provide interaction with the Mannequin system and should be used only for light-weight environment or background animation.



Ensure that the layer ID is set up correctly when assigning multiple animations to one component. Animations on higher layers override animations on lower layers.

Simple Animation Component Properties

The **Simple Animation** component has the following properties:

Animation Name

Name of the animation played by this component on this layer in the absence of an overriding animation.

Layer ID

Layer ID that this animation is to be played on. Animations can override each other if they are not properly authored.

Looping

If selected, animation continues to play in a loop until stopped.

Playback speed

Speed of the animation playback.

Layer weight

Weight of animations played on this layer.

EBus Request Bus Interface

Use the following request functions with the EBus interface to communicate with other components of your game.

For more information about using the Event Bus (EBus) interface, see [Event Bus \(EBus\)](#).

StartDefaultAnimations

Plays the default animations with default looping and speed parameters that were set up as a part of this component. The component allows for multiple layers to be set up with defaults; this method allows the playback of configured playback layers simultaneously.

Parameters

None

Return

Result indicating whether animations started successfully.

scriptable

Yes

StartAnimationByName

Plays the animation with the specified name on the specified layer.

Parameters

`name` – The name of the animation to play

`layerId` – The layer in which to play the animation

Return

Result indicating whether animations started successfully.

scriptable

Yes

StartAnimation

Plays the animation as configured by the `animatedLayer`.

Parameters

`animatedLayer` – A layer configured with the animation that is to be played on it.

Return

Result indicating whether animations started successfully.

scriptable

Yes

StartAnimationSet

Plays a set of animations as configured by each `AnimatedLayer` in the `animationSet`.

Parameters

`animationSet` – An `AnimatedLayer::AnimatedLayerSet` containing animations to be kicked off simultaneously.

Return

Result indicating whether animation set started successfully.

scriptable

No

StopAllAnimations

Stops all animations that are being played on all layers.

Parameters

None

Return

Result indicating whether animations stopped successfully.

scriptable

Yes

StopAnimationsOnLayer

Stops the animations currently playing on the indicated layer.

Parameters

`layerId` – ID for the layer that is to stop its animation
(`0, AnimatedLayer::s_maxActiveAnimatedLayers-1`).

Return

Result indicating whether animations stopped successfully.

scriptable

Yes

StopAnimationsOnLayers

Stops the animations currently playing on the indicated layers.

Parameters

`layerIds` – A bitset indicating layers to stop animating.

Return

Result indicating whether animations stopped successfully.

scriptable

No

EBus Response Bus Interface

Use the following response functions with the EBus interface to communicate with other components of your game.

For more information about using the Event Bus (EBus) interface, see [Event Bus \(EBus\)](#).

OnAnimationStarted

Informs all listeners about an animation being started on the indicated layer.

Parameters

`animatedLayer` – Animated layer indicating the animation and the parameters used to start the animation.

Return

None

scriptable

Yes

OnAnimationStopped

Informs all listeners about an animation being stopped on the indicated layer.

Parameters

`animatedLayer` – Animated layer indicating the animation and the parameters used on the animation that was stopped.

Return

None

scriptable

Yes

Script Examples

The following is an example of the `StartAnimation` function.

```
function chickenanimcontroller:OnActivate()

    -\\- For sending events on the SimpleAnimationComponent request bus.

    self.animBusSender =
    SimpleAnimationComponentRequestBusSenders(self.entityId);

    -\\- Start by playing the idle animation.

    -\\- Layer=0, looping = True, speed=1.0, blendtime= 0.0

    local animInfo = AnimatedLayer("anim_chicken_idle", 0, true, 1.0, 0.0);

    self.animBusSender:StartAnimation(animInfo);

end
```

The following is an example of script using the **Request Bus Interface**.

```
chickenanimcontroller =
```

```
{
  Properties =
  {
    FlapInterval = { default = 0.5, description = "How often the chicken
flaps.", suffix = " sec" },
    MoveSpeed = { default = 3.0, description = "How fast the chicken
moves.", suffix = " m/s" },
    IdlePlaybackSpeed = { default = 1.0, description = "Playback speed
for the idle animation." },
    FlapPlaybackSpeed = { default = 1.0, description = "Playback speed
for the flap/jump animation." },
    FlapBlendTime = { default = 0.2, description = "Blend time for the
flap animation." },
  },
}

function chickenanimcontroller:OnActivate()

  self.FlapCountdown = 0.0;

  -- For handling tick events.
  self.tickBusHandler = TickBusHandler(self, 0);

  -- For sending events on the SimpleAnimationComponent request bus.
  self.animBusSender =
SimpleAnimationComponentRequestBusSender(self.entityId);

  -- Start by playing the idle animation.
  -- Layer 0, looping, speed=1, no transition time.
  local animInfo = AnimatedLayer("anim_chicken_idle", 0, true,
self.Properties.IdlePlaybackSpeed, 0.0);
  self.animBusSender:StartAnimation(animInfo);
end

function chickenanimcontroller:OnTick(deltaTime, timePoint)

  -- Play the Flap animation FlapInterval seconds.
  self.FlapCountdown = self.FlapCountdown - deltaTime;
  if (self.FlapCountdown < 0.0) then
    -- Layer 0, non-looping, speed=1, 0.2 transition time.
    -- If the flap were partial body, we could use Layer 1.
    local animInfo = AnimatedLayer("anim_chicken_flapping", 0, false,
self.Properties.FlapPlaybackSpeed, self.Properties.FlapBlendTime, true);
    self.animBusSender:StartAnimation(animInfo);
    self.FlapCountdown = self.Properties.FlapInterval;
  end

  -- Get current transform
  local tm = self.transformBusHandler:GetWorldTM();

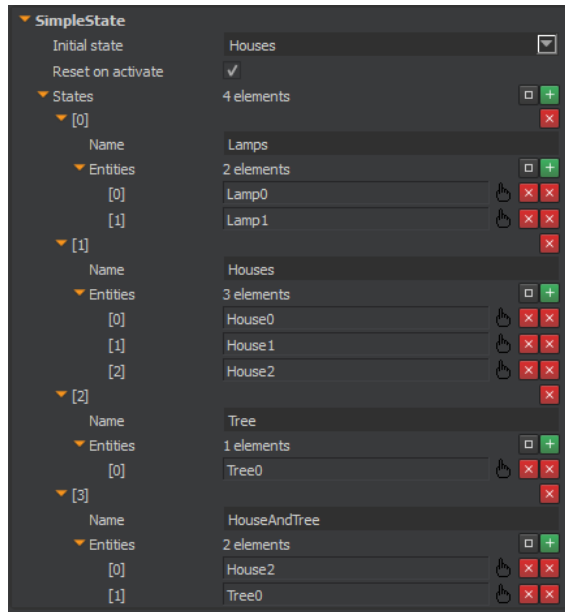
  -- Adjust translation forward at the configured movement speed.
  local forward = tm:GetColumn(1);
  local tx = tm:GetTranslation();
  tx = tx + forward * deltaTime * self.Properties.MoveSpeed;
  tm:SetTranslation(tx);

  -- Set our new transform.
  self.transformBusHandler:SetWorldTM(tm);
end
```

```
function chickenanimcontroller:OnDeactivate()  
    self.tickBusHandler:Disconnect();  
  
end
```

Simple State Component

The **Simple State** component provides a simple state machine. Each state is represented by a name and zero or more entities. The entities are activated upon entering the state and deactivated upon exiting it. A simple state component may be in NullState, which means no state is active.



Simple State Component Properties

The **Simple State** component has the following properties:

Initial state

The active state when the simple state component is first activated.

Reset on activate

If selected, simple state returns to the configured initial state when activated, and not the state held before deactivating.

States

The list of states on this simple state component.

State ([0], [1], [2], etc)

Includes a name for the state and a set of entities that are activated when the state is entered and deactivated when the state is exited.

Name

The name of this state. Indicates the state to which to transition on the SetState API.

Entities

List of the entities referenced by this state.

EBus Request Bus Interface

Use the following request functions with the EBus interface to communicate with other components of your game.

For more information about using the Event Bus (EBus) interface, see [Event Bus \(EBus\)](#).

SetState

Sets the active state to the named state.

Parameters

stateName

EBus Notification Bus Interface

Use the following notification functions with the EBus interface to communicate with other components of your game.

For more information about using the Event Bus (EBus) interface, see [Event Bus \(EBus\)](#).

OnStateChanged

Notifies that the state has changed from state `oldName` to state `newName`.

Parameters

oldName

newName

The following is an example of script using the **Request Bus Interface**.

```
simplestateexample =
{
    Properties =
    {
        TransitionInterval = 1.0,
        States = {"Houses", "Lamps", "Tree", "HouseAndTree"},
    },
}
function simplestateexample:OnActivate()
    self.TransitionCountDown = self.Properties.TransitionInterval;
    self.StateIdx = 0;
    self.busSender = SimpleStateComponentRequestBusSender(self.entityId);
    self.tickBusHandler = TickBusHandler(self, 0);
    self.stateChangedHandler =
SimpleStateComponentNotificationBusHandler(self, self.entityId);
    Debug.Log("SimpleStateComponent activated for entity: " ..
tostring(self.entityId));
end
function simplestateexample:OnTick(deltaTime, timePoint)
    self.TransitionCountDown = self.TransitionCountDown - deltaTime;
    if (self.TransitionCountDown < 0.0) then
        self.busSender:SetState(self.Properties.States[self.StateIdx]);
        self.StateIdx = (self.StateIdx + 1) %
table.getn(self.Properties.States);
        self.TransitionCountDown = self.Properties.TransitionInterval;
```

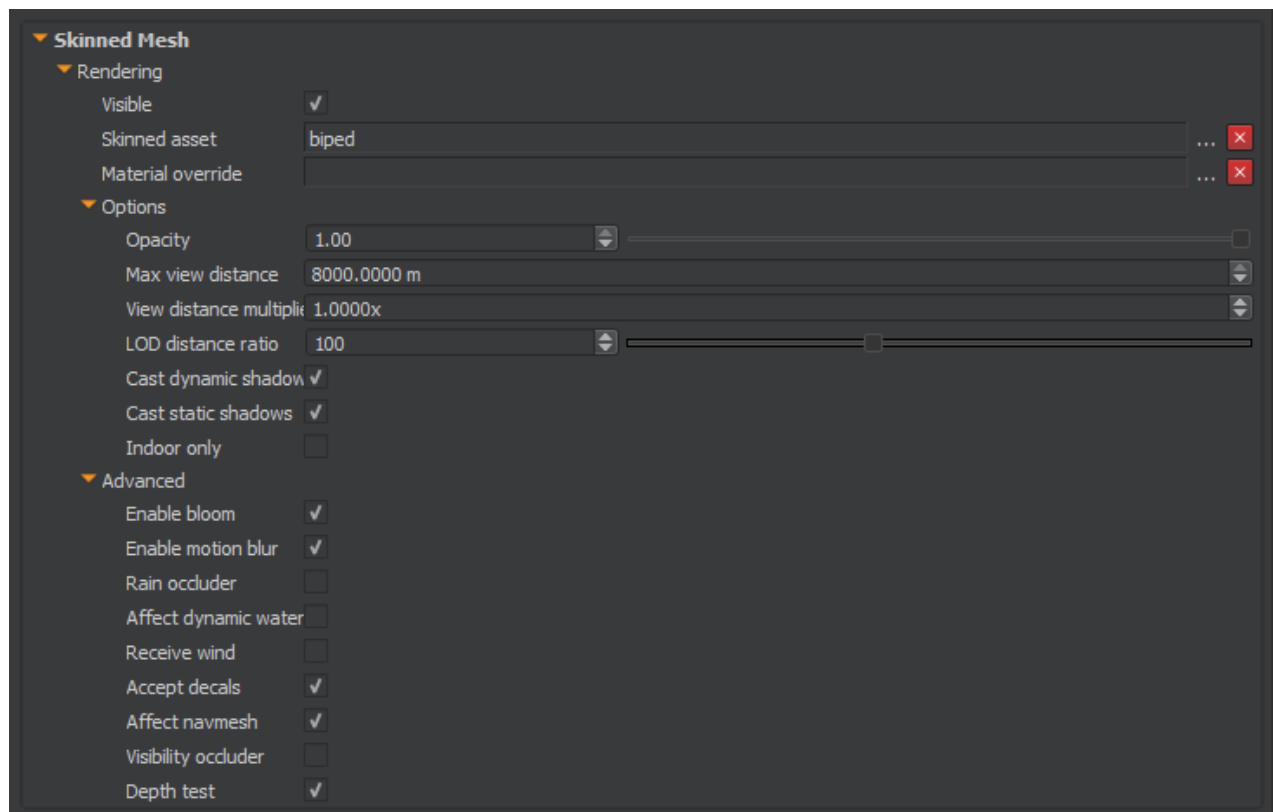


```
end
end
function simplestateexample:OnDeactivate()
    self.tickBusHandler:Disconnect();
    self.stateChangedHandler:Disconnect();
end

function simplestateexample:OnStateChanged(oldState, newState)
    Debug.Log("Old State: " .. oldState);
    Debug.Log("New State: " .. newState);
end
```

Skinned Mesh Component

The **Skinned Mesh** component is the primary way to add animated visual geometry to entities. This component also features key controls and options to use the engine's basic rendering features. Supported geometry types include skinned meshes (.chr) and character descriptors (.cdf).



Skinned Mesh Component Properties

The **Skinned Mesh** component has the following properties:

Visible

When selected, the entity is visible.

Skinned Asset

Asset file for the skinned mesh entity.

Material Override

Specifies an override material.

Options

Opacity

Scale of how opaque an entity is.

Max view distance

Maximum distance from which this entity can be viewed.

View distance multiplier

Adjusts the maximum view distance. If set to 1.0, then the default maximum view distance is used. 1.1, for example, extends the default by 10%.

LOD distance ratio

Sets the Level of Detail (LOD) ratio over distance.

Cast dynamic shadows

When selected, casts dynamic shadow maps.

Cast static shadows

When selected, casts static shadow maps.

Indoor only

When selected, renders the object only in indoor areas.

Advanced

Enable bloom

When selected, entity enables bloom post effect.

Enable motion blur

When selected, entity enables motion blur post effect.

Rain occluder

When selected, entity blocks or stops dynamic raindrops.

Affect dynamic water

When selected, entity generates ripples in dynamic water.

Receive wind

When selected, entity is affected by wind.

Accept decals

When selected, entity can receive decals.

Affect navmesh

When selected, entity affects navmesh generation.

Visibility occluder

When selected, entity can block visibility of other objects.

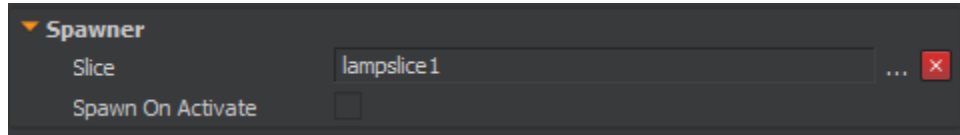
Depth test

When selected, entity requires depth testing.

Spawner Component

component entity system is in preview release and is subject to change.

Use the **Spawner** component to spawn a design-time or run-time dynamic slice (`*.dynamicslice`) at an entity's location with an optional offset.



Spawner Component Properties

The **Spawner** component has the following properties:

Slice

The slice to spawn.

Spawn on Activate

If selected, spawns the selected slice upon activation.

EBus Request Bus Interface

Use the following request functions with the Spawner Component Request Bus EBus interface to communicate with other components of your game.

For more information about using the Event Bus (EBus) interface, see [Event Bus \(EBus\)](#).

Spawn

Spawns the selected slice at the entity's location.

Parameters

None

Scriptable

Yes

SpawnRelative

Spawn the selected slice at the entity's location with the provided `relative offset`.

Parameters

`relative`

Scriptable

Yes

SpawnSlice

Spawns the `slice` at the entity's location.

Parameters

`slice`

Scriptable

No

SpawnSliceRelative

Spawns the `slice` at the entity's location with the `relative offset`.

Parameters

`slice`

relative

Scriptable

No

EBus Notification Bus Interface

Use the following notification functions with the Spawner Component Notification Bus EBus interface to communicate with other components of your game.

For more information about using the Event Bus (EBus) interface, see [Event Bus \(EBus\)](#).

OnSpawned

Notifies that a spawn has occurred.

Parameters

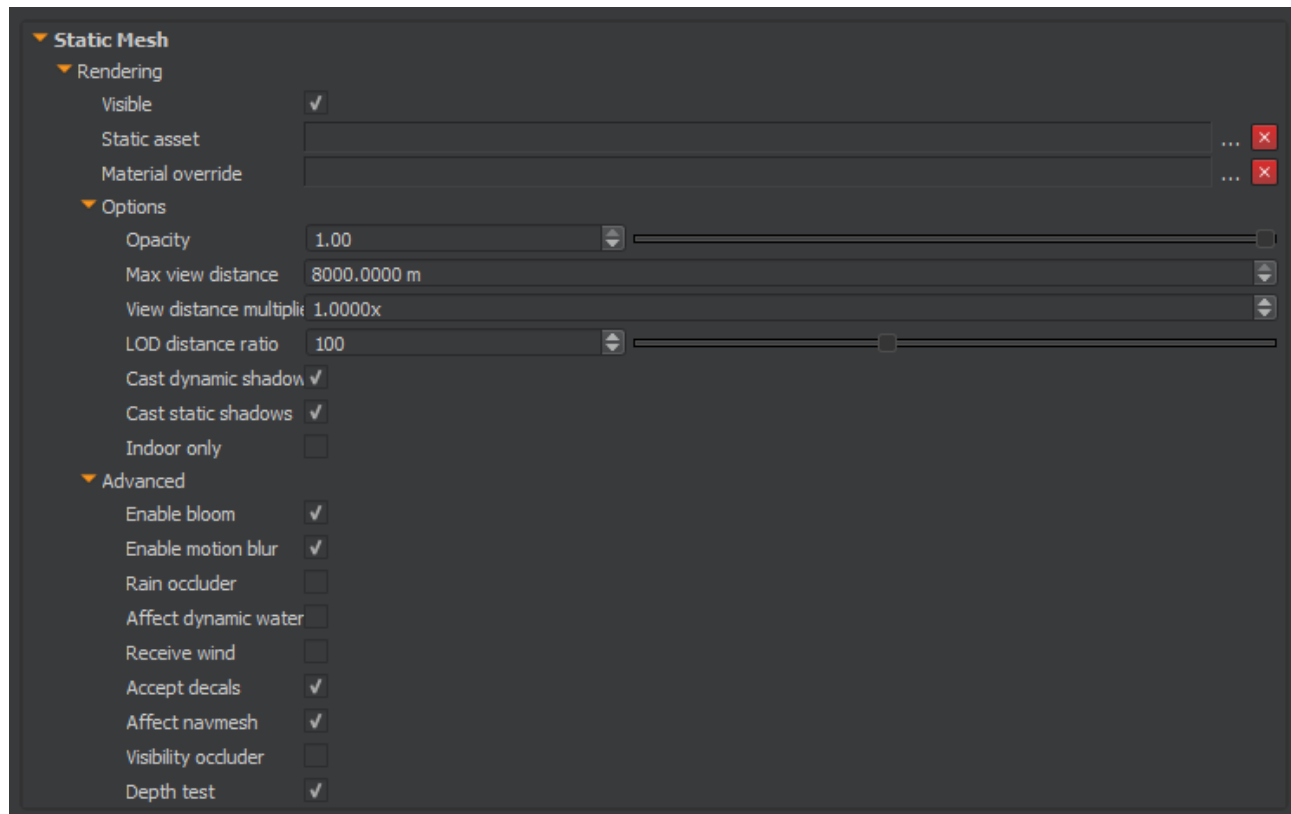
spawnedEntities

Scriptable

Yes

Static Mesh Component

The **Static Mesh** component is the primary method of adding static visual geometry to entities. This component also features key controls and options to use the engine's basic rendering features. The supported geometry type is static meshes (.cgf).



Static Mesh Component Properties

The **Static Mesh** component has the following properties:

Visible

When selected, the entity is visible.

Static Asset

Asset file for the static mesh entity.

Material Override

Specifies an override material.

Options

Opacity

Scale of how opaque an entity is.

Max view distance

Maximum distance from which this entity can be viewed.

View distance multiplier

Adjusts the maximum view distance. If set to 1.0, then the default maximum view distance is used. 1.1, for example, extends the default by 10%.

LOD distance ratio

Sets the Level of Detail (LOD) ratio over distance.

Cast dynamic shadows

When selected, casts dynamic shadow maps.

Cast static shadows

When selected, casts static shadow maps.

Indoor only

When selected, renders the object only in indoor areas.

Advanced

Enable bloom

When selected, entity enables bloom post effect.

Enable motion blur

When selected, entity enables motion blur post effect.

Rain occluder

When selected, entity blocks or stops dynamic raindrops.

Affect dynamic water

When selected, entity generates ripples in dynamic water.

Receive wind

When selected, entity is affected by wind.

Accept decals

When selected, entity can receive decals.

Affect navmesh

When selected, entity affects navmesh generation.

Visibility occluder

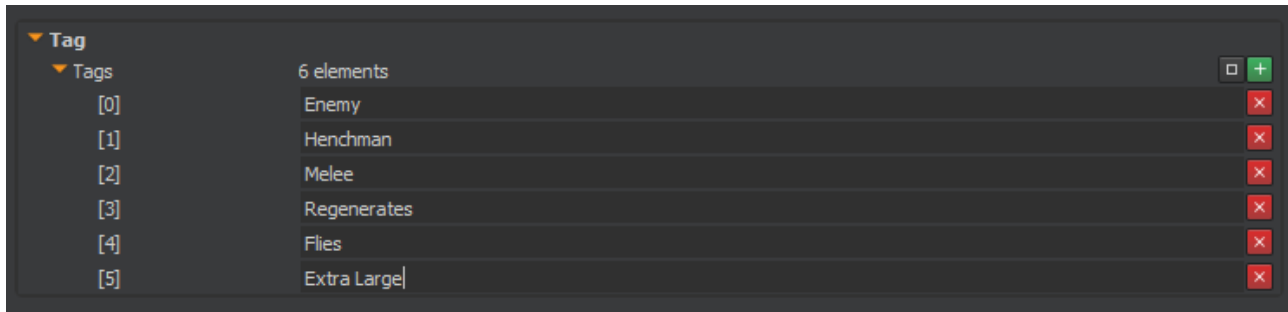
When selected, entity can block visibility of other objects.

Depth test

When selected, entity requires depth testing.

Tag Component

Use the **Tag** component to apply one or more labels, or tags, to an entity. You can use these tags to find or filter entities with particular traits. For example, you can set a weapon to inflict double damage to entities tagged as **burning**.



EBuses – Request Bus Interface: TagGlobalRequestBus

Use the following request function with the **TagGlobalRequestBus** EBus interface to communicate with other components of your game.

For more information about using the Event Bus (EBus) interface, see [Event Bus \(EBus\)](#).

RequestTaggedEntities

Handlers respond if they have the tag (listening on the tag's channel). Use `AZ::EbusAggregateResults` to handle more than the first responder.

Parameters

None

Return

`const AZ::EntityId`

Scriptable

Yes

EBuses – Request Bus Interface: TagRequestBus

Use the following request functions with the **TagRequestBus** EBus interface to communicate with other components of your game.

For more information about using the event bus (EBus) interface, see [Event Bus \(EBus\)](#).

HasTag

Returns true if the entity has the tag.

Parameters

`const Tag&`

Return

`bool`

Scriptable
Yes

AddTag

Adds the tag to the entity if it didn't already have it.

Parameters
const Tag&

Return
None

Scriptable
Yes

AddTags

Add a list of tags to the entity if it didn't already have them.

Parameters
const Tags&

Return
None

Scriptable
No

RemoveTag

Removes a tag from the entity if it had it.

Parameters
const Tag&

Return
None

Scriptable
Yes

RemoveTags

Removes a list of tags from the entity if it had them.

Parameters
const Tags&

Return
None

Scriptable
No

GetTags

Gets the list of tags on the entity.

Parameters
None

Return
const Tags&
Scriptable
No

The following is an example of script using the **Request Bus Interface**.

```
self.enemyFinder = TagGlobalRequestBusSender(Crc32("Enemy"));  
local enemies = self.enemyFinder:RequestTaggedEntities();  
  
local burning =  
    TagComponentRequestBusSender(self.entityId):HasTag(Crc32("Burning"));
```

EBus – Notification Bus Interface: TagComponentNotificationsBus

Use the following request functions with the **TagComponentNotificationsBus** notification bus interface to communicate with other components of your game.

For more information about using the event bus (EBus) interface, see [Event Bus \(EBus\)](#).

OnTagAdded

Notifies listeners about tags being added.

Parameters
const Tag& – Indicates the tag was added
Return
None
Scriptable
Yes

OnTagRemoved

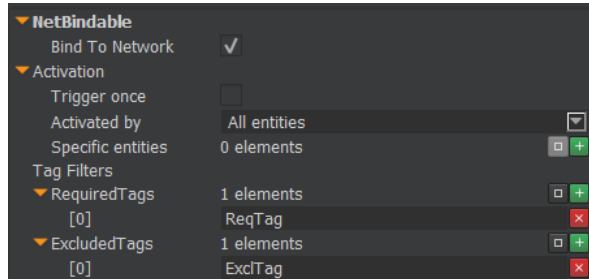
Notifies listeners about tags being removed.

Parameters
const Tag& – Indicates the tag was removed
Return
None
Scriptable
Yes

Trigger Area Component

component entity system is in preview release and is subject to change.

The **Trigger Area** component provides generic triggering services by using [Shape \(p. 390\)](#) components as its bounds. To use a trigger component, add one of the shape components to an entity and then add a trigger component to it. The shape then acts as the bounds for this trigger. You can set filters on the type of entities that can trigger an area and set the trigger to synchronize across the network.



Trigger Area Component Properties

The **Trigger Area** component has the following properties:

NetBindable

NetBindable components are synchronized over the network.

Bind To Network

When selected, synchronizes component across the network.

Activation

Trigger once

If selected, the trigger deactivates after the first trigger event.

Activated by

Select whether trigger is activated by **All entities**, which allows any entity to trigger the area, or by **Specific Entities**, which allows you to select specific entities.

Tag filters

RequiredTags

A list of tags that are required for an entity to trigger this area.

ExcludedTags

A list of tags that exclude an entity from triggering this area.

EBus Request Bus Interface

Use the following request functions with the EBus interface to communicate with other components of your game.

For more information about using the Event Bus (EBus) interface, see [Event Bus \(EBus\)](#).

AddRequiredTag

Adds a required tag to the activation filtering criteria of this component. Results in a reevaluation of the trigger. Entities inside that no longer satisfy tag criteria are ejected.

Parameters

const Tag&

requiredTag – Tag to be added

Return

None

Scriptable
Yes

RemoveRequiredTag

Removes a required tag from the activation filtering criteria of this component. Results in a reevaluation of the trigger. Entities inside that no longer satisfy tag criteria are ejected.

Parameters

`const Tag&`

`requiredTag` – Tag to be removed

Return

None

Scriptable

Yes

AddExcludedTag

Adds an excluded tag to the activation filtering criteria of this component. Results in a reevaluation of the trigger. Entities inside that no longer satisfy tag criteria are ejected.

Parameters

`const Tag&`

`excludedTag` – Tag to be added

Return

None

Scriptable

Yes

RemoveExcludedTag

Removes an excluded tag from the activation filtering criteria of this component. Results in a reevaluation of the trigger. Entities inside that no longer satisfy tag criteria are ejected.

Parameters

`const Tag&` **`excludedTag`** – Tag to be removed

Return

None

Scriptable

Yes

EBus Notification Bus Interface

Use the following request functions with the EBus interface to communicate with other components of your game.

For more information about using the Event Bus (EBus) interface, see [Event Bus \(EBus\)](#).

The **Trigger** component sends notifications to:

- Entities listening on the TriggerAreaNotificationBus for the entity with the trigger on it.

- Entities listening on the `TriggerAreaEntityNotificationBus` for the entity that enters or exits the trigger.

TriggerAreaNotificationBus

This bus allows the game to listen for events associated with a particular trigger. Notifies of all the entities that enter and exit this trigger.

OnTriggerAreaEntered

Notifies listeners when `enteringEntityId` enters this trigger.

Parameters

`enteringEntityId` – ID of entity that has entered this trigger

Return

None

Scriptable

Yes

OnTriggerAreaExited

Notifies listeners when `enteringEntityId` exits this trigger.

Parameters

`enteringEntityId` – ID of entity that has exited this trigger

Return

None

Scriptable

Yes

TriggerAreaEntityNotificationBus

This bus allows the game to listen for trigger-related events associated with a particular entity. Notifies of every time the player enters or exits any trigger.

OnEntityEnteredTriggerArea

Sent when the entity enters `triggerID`.

Parameters

`triggerId` – ID of entity that the trigger is on.

Return

None

Scriptable

Yes

OnEntityExitedTriggerArea

Sent when the entity exits `triggerID`.

Parameters

`triggerId` – ID of entity that the trigger is on.

Return

None

Scriptable

Yes

Working with Entities

component entity system is in preview release and is subject to change.

This section discusses how to create and manage entities and components.

The following topics are presented:

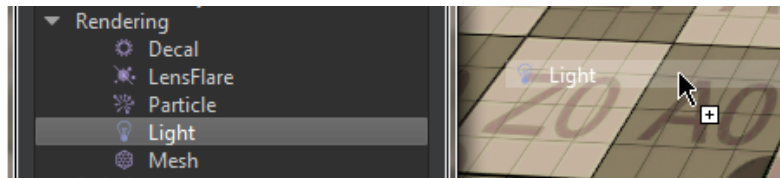
- Creating Entities
- Adding and Removing Components
- Finding an Entity
- Editing Component Properties

Creating an Entity

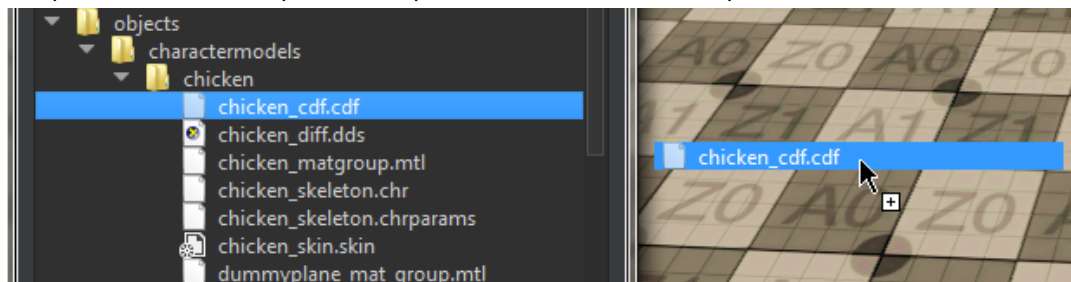
To create a new entity, the simplest method is to right-click in the Lumberyard Editor viewport and select **Create New Entity**. This will create a new entity at the cursor location with a basic transform component, which gives it a location in the 3D level.



Component Palette may also be used to create new entities pre-populated with a set of desired components. Using Component Palette, select one or more components and drag them into the Lumberyard Editor viewport. A single new entity will be created containing the selected components.



File Browser can be also used to create new entities by inferring the desired configuration given a particular asset. For example, dragging a .cgf mesh asset from File Browser into the Lumberyard Editor viewport creates a new entity, adds a mesh component, and assigns the asset to the mesh component. The same is possible for particles, slices, and Lua scripts.

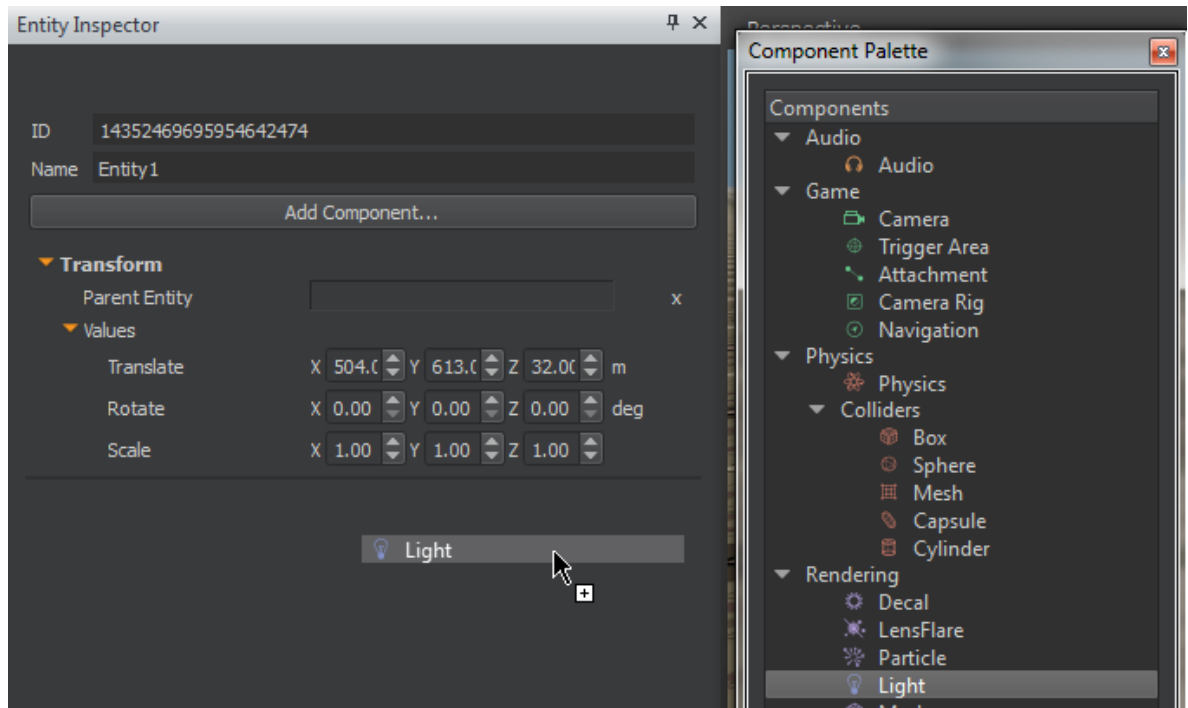


Adding and Removing Components

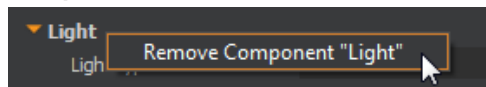
You can add new components to existing entities several different ways. While viewing an entity in Entity Inspector, you can add components in the following ways:

- Clicking the **Add Component** button in Entity Inspector and selecting the desired component from the categorized lists.

- Opening Component Palette and dragging any desired component from there to a blank area in Entity Inspector.

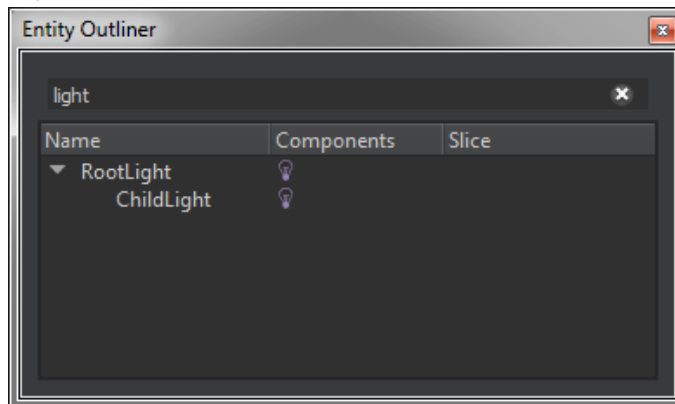


To delete a component, right-click the component name in Entity Inspector and choose **Remove Component**.

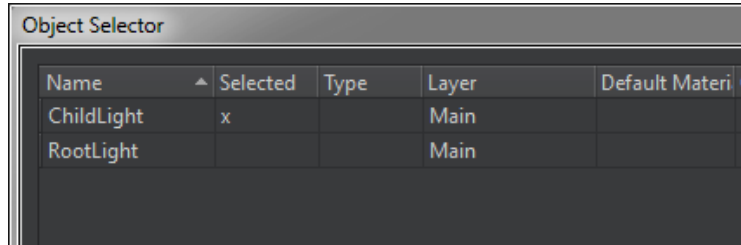


Finding an Entity

Entity Outliner provides a view of the entities in your level and any transform hierarchies you have in place, which will render as a tree. You may also use the search filter box at the top of Entity Outliner to find specific entities. Only entities whose name matching the filter's sub-string will be shown, along with any transform ancestors.



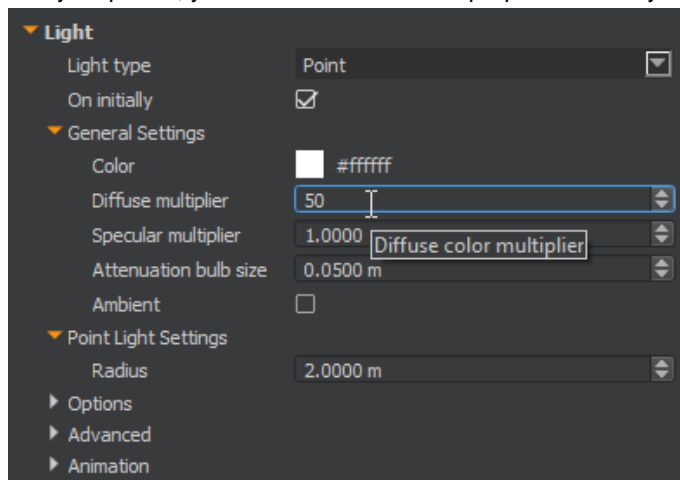
Clicking one or more entities in Entity Outliner will select the associated viewport objects, and vice versa.



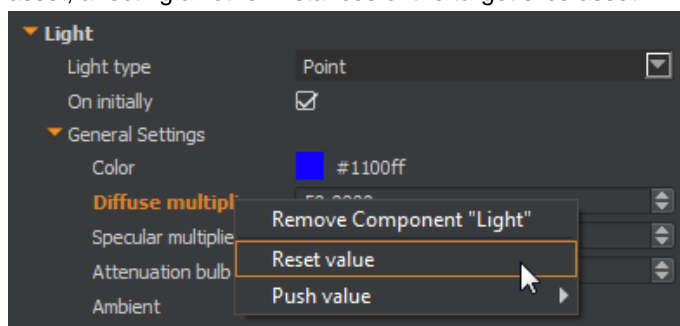
Editing Component Properties

To edit component properties, first select the entity you wish to edit. Then use Entity Inspector to edit component properties. In Lumberyard Editor choose **View, Open View Pane, Entity Inspector**.

Entity Inspector, you should see all visible properties for any components on your entity.

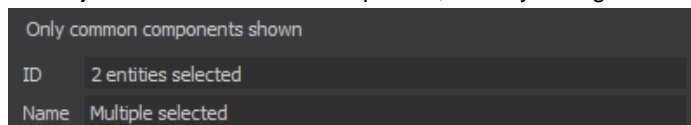


If your entity is part of a slice instance, any fields modified from the source slice asset will render in orange. Right-click the field name to reset to the original value, or to push your change to the slice asset, affecting all other instances of the target slice asset.



Any change to a property can be undone by pressing **Ctrl-Z**, and subsequently redone by pressing **Ctrl-Shift-Z**.

Multiple entities may be edited simultaneously. When multiple entities are selected, Entity Inspector first object's version of each component, but any changes will be propagated to all selected entities.



Working with Slices

component entity system is in preview release and is subject to change.

Slices are a cascaded data management system for entities. They are a superset to what are also known as prefabs, and represent the structure in which nearly all entity data is managed. For example, a level, a house, a car, and an entire world are all slices that simply depend on (cascade) from a number of other slices. A level is simply a root-level slice cascading from any slices instanced in the level, as well as any loosely-placed entities.

Slice changes can be pushed or pulled through any level of the hierarchy.

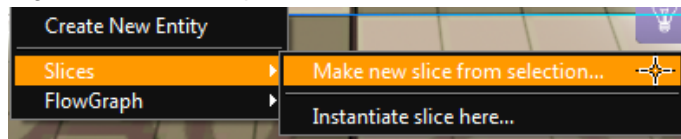
Note

Assets, including slices, cannot be moved from their original location without breaking references.

Creating a Slice

A slice can contain any number of entities, with no restriction on relationships between those entities. As such, there is no requirement to have a root entity for a slice.

1. In Lumberyard Editor, select one or more entities to include in the slice.
2. Right-click in the viewport and choose **Slices, Make slice from selection...**

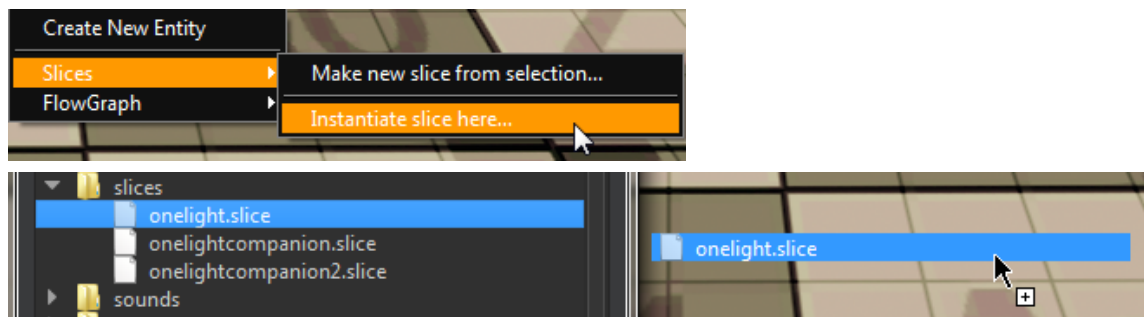


3. Save a slice to the desired location.

Instantiating a Slice

To create an instance of a slice in your level, there are a couple of options:

- In Lumberyard Editor, right-click in the viewport and choose **Slices, Instantiate slice here...**



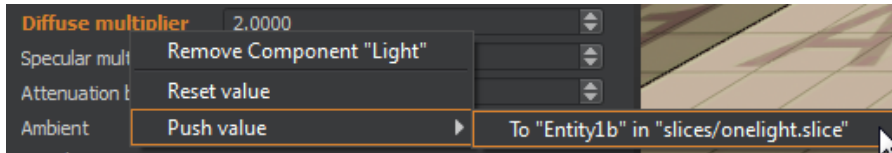
Modifying a Slice and Pushing Changes

If an entity comes from a hierarchy of cascaded slices, you have the option to push your property change to any level of the hierarchy.

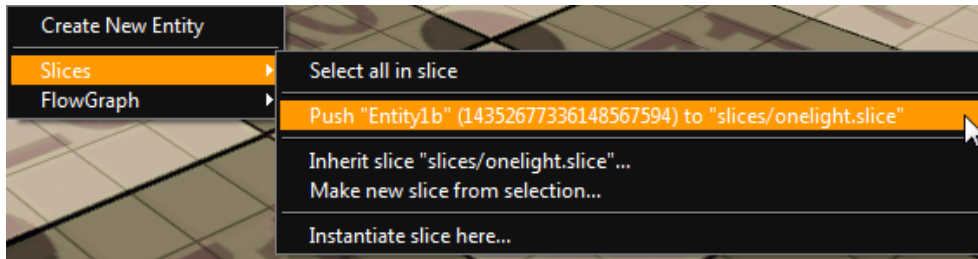
If all selected entities are from the same slice instance, they can be pushed together in a single operation. Otherwise you have the option for single-entity pushes.

To modify a slice

1. In Entity Inspector, modify any number of properties for components belonging to an entity that is part of a slice instance. The field should render in orange if it differs from the source slice asset.
2. Right-click on the property name, and choose **Push value**.



3. Right-click the entity in the Lumberyard Editor viewport and choose **Slices, Push entity**



Cloning a Slice

When cloning slices, the entire instance is cloned while maintaining the relationship to the source slice. Cloning a subset of the entities within a slice instance will clone them as loose entities.

To clone a slice, open Entity Outliner. Under the **Name** column, select all the entities within the slice using either **Ctrl+click**, **Shift+click**, or right-click **Select all in slice** and then select **Clone**.

A clone operation can be undo or redone by pressing **Ctrl-Z** and **Ctrl-Shift-Z** respectively.

Entities may also be cloned by selecting one or more entities and pressing **Ctrl+C**. Cloned entities are automatically selected to aid in initial placement.

Inheriting a Slice (Data Cascading)

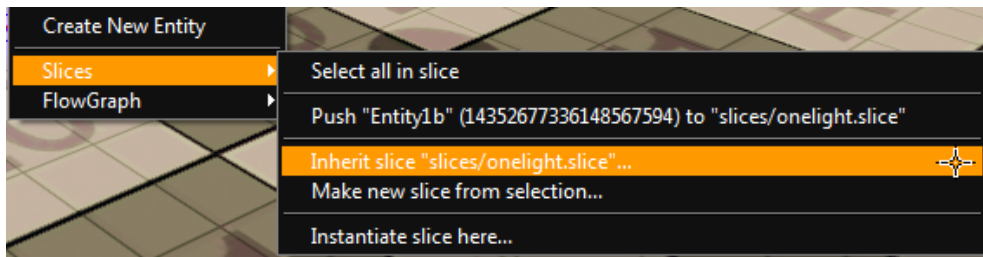
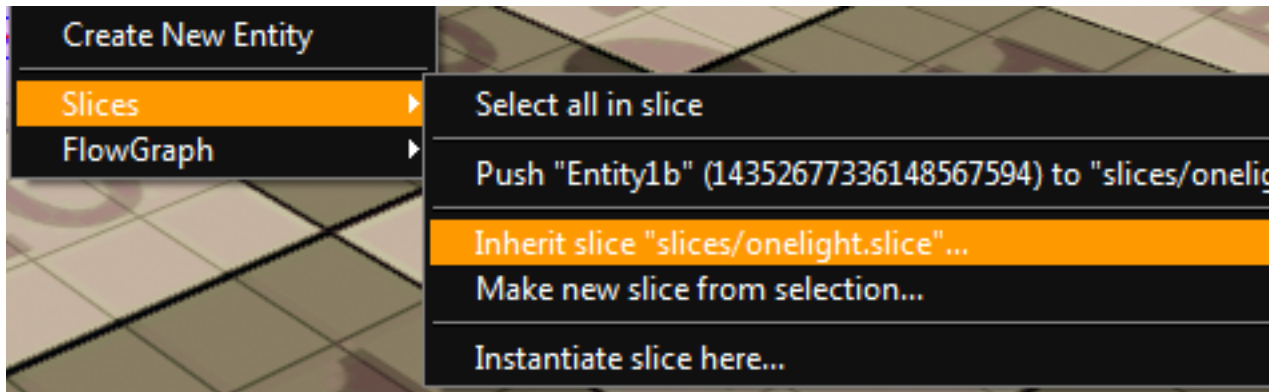
Slices have the ability to cascade (or inherit) slices from other slices.

All entities in a slice that originate from the inherited slice will retain their relationship to the hierarchy. Any changes made to the inherited prefabs will propagate down the hierarchy, unless the data has been overridden lower in the hierarchy.

Slices can reference any number of other slices, and can contain any number of slice instances.

To inherit a slice

1. Instantiate one or more slices.
2. Make desired additions or modifications to the instantiated entities. Create new entities, remove entities, add/remove components, or modify component fields.
3. Highlight the desired set of entities you'd like to be contained in your new inherited slice.
4. Right-click in the viewport and choose **Slices, Inherit slice**.



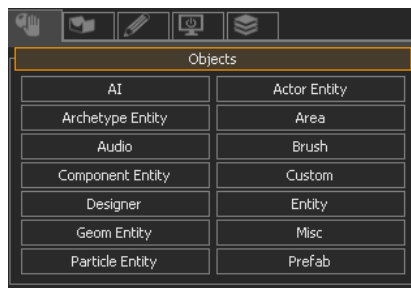
5. Save the slice to the desired location.

Slice Reloading

Slices support run-time reloading. If a slice has changed on disk for any reason, whether due to a data push operation, grabbing from source control, or through a hand-edit, Lumberyard Editor will reload the slice asset and re-calculate any slice instances affected by the change, such as instances of any slices that are dependent on the modified slice, through any number of chained dependencies.

Object and Entity System

Using the Object and Entity system, you can create and place objects, brushes, and entities in your level. Entities are objects with which the player interacts. Similar to brushes, they can be placed in a level, and are accessed from the **Objects** tab of **Rollup bar**.



Note

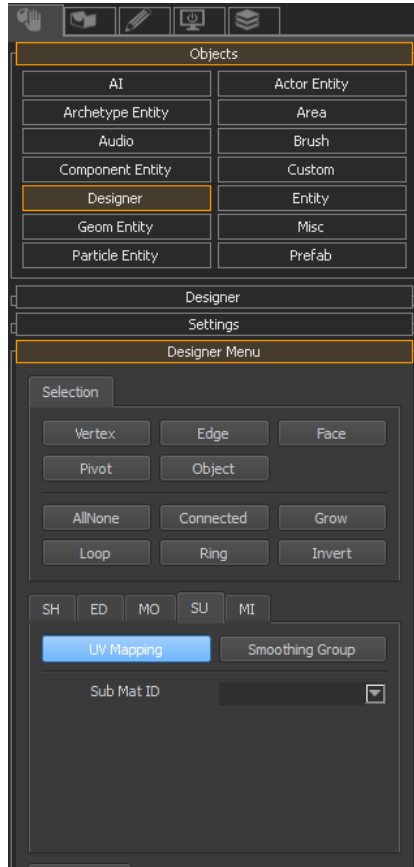
The [Component Entity System \(p. 320\)](#) replaces the existing Entity system in Lumberyard at a future date.

Topics

- [Using the Designer Tool \(p. 416\)](#)
- [Using the Measurement System Tool \(p. 425\)](#)
- [Using the Object Selector \(p. 426\)](#)
- [Brushes \(p. 429\)](#)
- [Prefabs \(p. 430\)](#)
- [Common Parameters and Properties \(p. 432\)](#)
- [Entity Reference \(p. 438\)](#)

Using the Designer Tool

The Designer Tool is an advanced object creation tool. You can easily create complex object meshes with powerful built-in functionality, without the need to use external DCC tools.



Topics

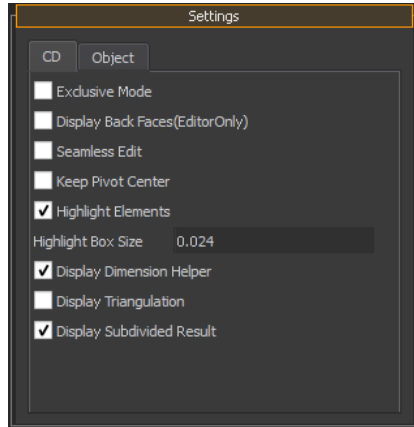
- [Designer Tool Settings \(p. 417\)](#)
- [Selection Tools \(p. 419\)](#)
- [Shape Tools \(p. 420\)](#)
- [Edit Tools \(p. 422\)](#)
- [Modify Tools \(p. 422\)](#)
- [Texture Tools \(p. 423\)](#)
- [Miscellaneous Tools \(p. 425\)](#)

Designer Tool Settings

The following parameter groups are available on the **Settings** panel.

CD Settings

The following parameters are available on the **CD** tab located on the **Settings** panel.

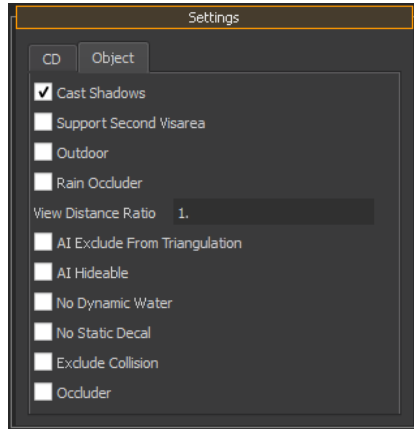


CD Parameters

Property	Description
Exclusive Mode	Use to make the view look like that of a DCC tool. In this mode, all objects except for the selected objects are hidden and the time of day and light settings are set only. When a level has a lot of objects and is complex, this mode makes the view's complexity decrease.
Display Back Faces (Editor Only)	Used to enable showing the backfaces of designer objects, such as when the camera is within an object.
Seamless Edit	Enables editing objects as the mouse cursor hovers over them.
Keep Pivot Center	Ensures that the pivot remains unaffected during editing.
Highlight Elements	Toggles visualization of the object's selected elements such as vertices, edges, and faces.
Highlight Box Size	When Highlight Elements is enabled, this controls the scale of the helpers used to highlight elements.
Display Dimension Helper	Enables visualization of the object's dimensions, width, height, and depth.
Display Triangulation	Overlays the object's triangulation.
Display Subdivided Result	Overlays the object's subdivisions.

Object Settings

The following parameters are available on the **Object** tab located on the **Settings** panel.

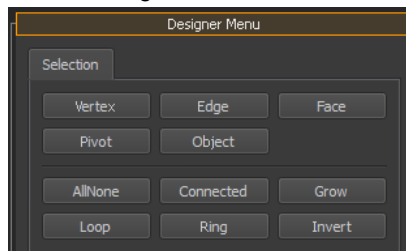


Object Parameters

Property	Description
Cast Shadows	Allows objects to cast shadows
Support Second Visarea	Normally, objects are considered to be in only one visarea. This option allows them to be added to multiple visareas if their bounding box overlaps them, at the cost of some performance. Without this option, some large objects may not be displayed when viewed through portals in certain situations.
Outdoor	When set, the object will not be rendered when inside a visarea.
Rain Occluder	Occludes dynamic raindrops
View Distance Ratio	Sets the distance from the current view at which the object renders.
AI Exclude From Triangulation	Deprecated
AI Hideable	When this option is set, AI will use this object as a hiding spot, using the specified hide point type.
No Static Decal	When this option is set, decals will not project onto the object.
Exclude Collision	Enable to exclude collisions.
Occluder	Used for the construction of a level occlusion mesh.

Selection Tools

The following function buttons are available from the **Selection** tab on the **Designer Menu** panel.



AllNone

Use to select or deselect all objects at once.

Connected

Use to select all faces connecting one another from the selected face.

Grow

Use to expand a selection based on the selected faces. Each time you press **Grow**, the selection range is enlarged based on the previous selected faces.

Invert

Use to invert the selection states of the faces. Selected faces will be unselected and unselected faces will be selected.

Loop

Use to select serial-linked edges or faces that form a loop from selected edges or faces.

Object

Use to select another object.

Pivot

Use to change the pivot position.

Ring

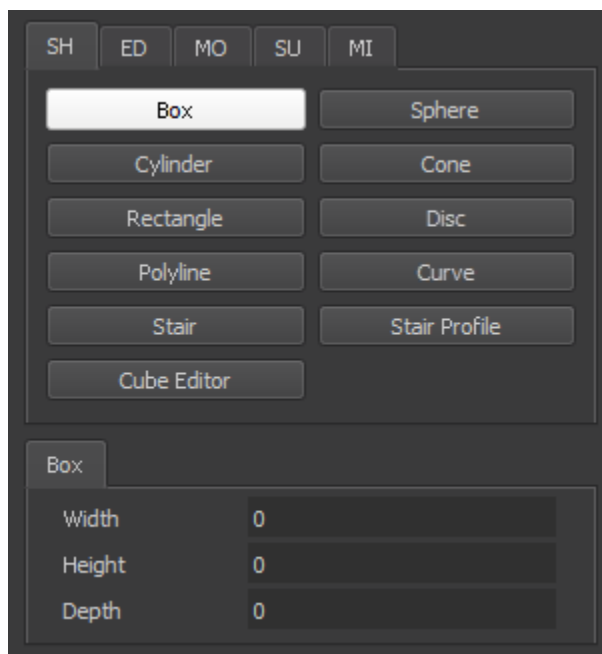
Use to select sequence edges that are not connected but on the opposite side to each other. You can also select serial-connected quad faces in a direction that is perpendicular to the direction that the selected two faces set.

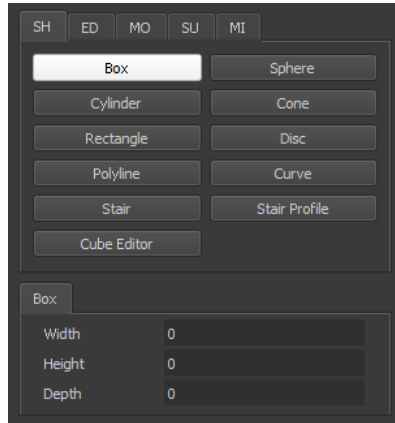
Vertex, Edge, Face

Use to select and move vertices, edges, and faces. You can select multiple buttons using the **Ctrl** key.

Shape Tools

The following buttons are available from the **SH** tab on the **Designer Menu** panel.





Box

Used to draw one or more boxes. You can adjust the **Width**, **Height**, and **Depth** values.

Cone

Used to draw a cone. You can adjust the **Subdivision Count**, **Height**, and **Radius** values.

Cube Editor

Used to create one or more cubes. You can add, remove, and paint cubes. The following functions are provided:

- **Add** - Add a cube on the brush with the specified Sub Material ID.
- **Remove** - Remove a cube under the brush.
- **Paint** - Paint selected cubes with the specified Sub Material ID.
- **Brush Size** - Select the cube brush size.
- **Sub Material ID** - Specifies the sub material ID. This ID will be recorded to faces affected.
- **Merge Sides** - When enabled, the added faces or remained faces after removing a cube will be merged with the adjoining faces.

Curve

Used to draw either a standard curve or a Bezier curve. You can adjust the **Subdivision Count** value.

Cylinder

Used to draw a cylinder. You can adjust the **Subdivision Count**, **Height**, and **Radius** values.

Disc

Used to draw a disc. You can adjust the **Subdivision Count** and **Radius** values.

Polyline

Used to draw a line or multiple line segments on a surface.

Rectangle

Used to draw a rectangle. You can adjust the **Width** and **Depth** values.

Stair

used to create a staircase. You can create stairs having uniform a step size even though the sizes of stairs are different by adjusting a tread size automatically so that a character can rise. The following values can be adjusted:

- **Step Rise** - The size of each step rise.
- **Mirror** - Mirrors a stair against an invisible plane centered.
- **Rotation by 90 Degrees** - Rotates a stair by 90 degrees maintaining the width, height and depth of a box.
- **Width** - The width of the stair.
- **Height** - The height of the stair.
- **Depth** - The depth of the stair.

Stair Profile

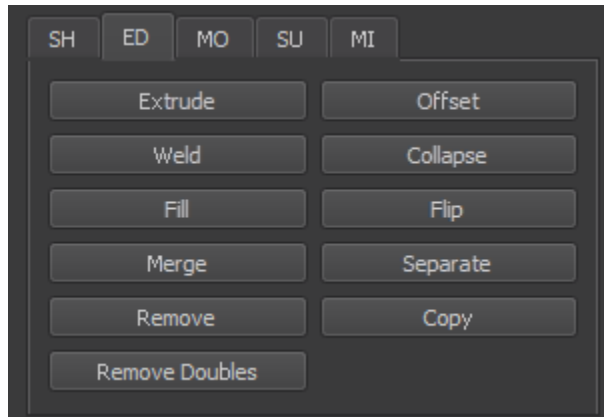
Used to draw a stair profile on a surface, which can be pulled using the **Extrude** function to be a stair. You can adjust the **Step Rise** value.

Sphere

You can adjust the **Subdivision Count** and **Radius** values.

Edit Tools

The following buttons are available from the **ED** tab on the **Designer Menu** panel.



Collapse

use to collapse all connected edges to the center position.

Copy

Use to copy an object face.

Extrude

Use to push or pull the selected face so you can expand a 2D surface to a 3D shape.

Fill

Use to fill a space based on selected edges or vertices.

Flip

use to flip an object face.

Merge

Used for merging multiple objects or connected faces to an object or a face.

Offset

Used to take a face and create an inset of the selected face.

Remove

Used to remove selected edges and faces.

Remove Doubles

Used to merge the selected vertices within the specified distance.

Separate

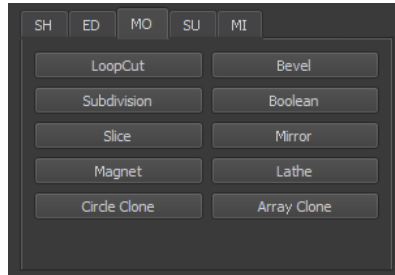
Used to separate two or more objects.

Weld

Used to merge the selected two vertices by moving the first vertex to the second vertex.

Modify Tools

The following buttons are available from the **MO** tab on the **Designer Menu** panel.



Bevel

Used to smooth edges of a shape. Most shapes have blunt edges, so applying the bevel to edges of a shape can add more realism.

Boolean

Select at least two objects, and chose either **Union**, **Difference**, or **Intersection**.

Array Clone

Places cloned objects evenly in a line.

Circle Clone

Places cloned objects in a circle

Lathe

Used to create a mesh by extruding each edge of a profile polygon along a path. You can make a complicated model using this method.

LoopCut

Used for cutting quad-shaped polygons by several loop edges. Set the direction and number of loops. The direction of the loops are set by the edge closest to the cursor and the number of loops are changed by moving the mouse wheel while pressing the CTRL key.

Magnet

Deprecated (merged with the Lathe function).

Mirror

Used to mirror a mesh along an arbitrary plane as well as its local X, Y, or Z axis plane. This tool has the following functions:

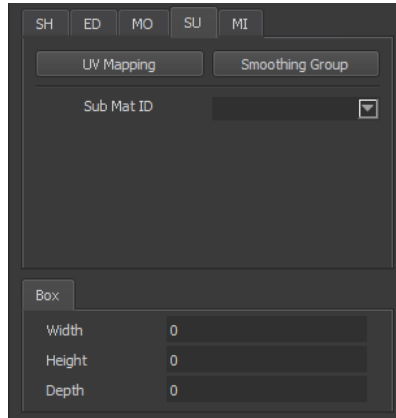
- **Apply** - Splits a mesh by a mirror plane and copies the half part to the other part and then starts the mirror editing.
- **Invert** - Invert a direction of the mirror plane.
- **Center Pivot** - Moves the pivot position to the center of the bounding box.
- **Align X**, **Align Y**, and **Align Z** - Aligns the mirror plane by X axis, Y axis, or Z axis.
- **Freeze** - Freezes the current geometry.

Subdivision

Used to create a smooth appearance of a mesh without complicated manipulations. A control mesh made this way doesn't need many vertices or faces to model complex smooth surfaces. You can also give each edge a semi-sharp crease, which defines how sharp each edge is.

Texture Tools

The following buttons are available from the **SU** tab on the **Designer Menu** panel.



Smoothing Group

Used for assigning numbers to faces. Faces with the same numbers and connected by an edge are rendered smoothly. A seam will be displayed between two faces with different smoothing group IDs. The following functions are available:

Parameters

Property	Description
Smoothing Groups	Used to assign a number to the selected faces.
Add Faces To SG	Used to select faces based on the selected number buttons.
Select Faces By SG	Used to select faces based on the selected number buttons.
Clear Empty SGs	Used to remove the assigned smoothing groups of the selected faces.
Auto Smooth with Threshold Angle	Sets the smoothing groups based on the angle between faces. Any two faces will be put in the same smoothing group if the angle between their normals is less than the threshold angle.
Threshold Angle	Used to set the angle in degrees

UV Mapping

Materials can be assigned to each face differently and you can manipulate the UV coordinates using this tool.

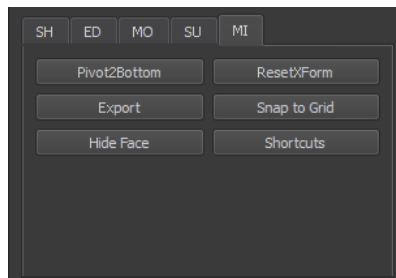
Parameters

Property	Description
Mapping	
UV offset	The parameters are set to solid directly.
Scale offset	The values are added to the existing parameters
Alignment	
Fit Texture	Fits the texture to the selected surfaces.

Property	Description
Reset	Resets the texture settings on selected surfaces.
Tiling	Changes texture tiling on selected surfaces in the X and Y directions.
Select	Selects all surfaces with the Material ID.
Assign	Assigns the Material ID to selected surfaces.

Miscellaneous Tools

The following buttons are available from the **MI** tab on the **Designer Menu** panel.



Export

Exports .obj, .cgf, or .grp files when these buttons are pressed.

Hide Face

Used to hide or unhide the selected faces.

ResetXForm

Resets the **Position**, **Rotation**, or **Scale** values when these checkboxes are selected.

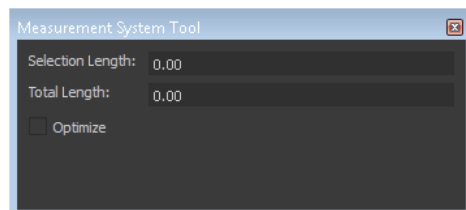
Shortcuts

Used to bind each function/subtool in the Designer tool to specific key combinations. The second column comprises CTRL, SHIFT and CTRL+SHIFT. The last column lists the available virtual keys.

Using the Measurement System Tool

The Measurement System Tool allows to measure the length of segmented objects like roads, rivers, and paths. Measuring of segments is done by following the shape of each segment. The measured path is shown in yellow color.

To read the length of some parts of a segmented object, a start point and an end point must be selected.



To measure a segmented object

1. Click to select the object in the viewport.
2. Click the **Edit** button. The object should turn yellow and be sunken.
3. Click **View, Open View Pane, Measurement System Tool**.
4. Click on the start of your desired first segment and the last segment of your choice to read its length. Double-clicking on any of the segment starting points selects the whole object for measuring or clears the start and end points.
5. Close the tool when done.

Using the Object Selector

Use the **Object Selector** to select and locate objects such as brushes, entities, tagpoints, volumes, and more. You can also hide and unhide objects, freeze and unfreeze objects, and delete objects. You can perform these actions on objects in [layers \(p. 872\)](#) that are selectable, visible, and not frozen.

To open Object Selector

Do one of the following:

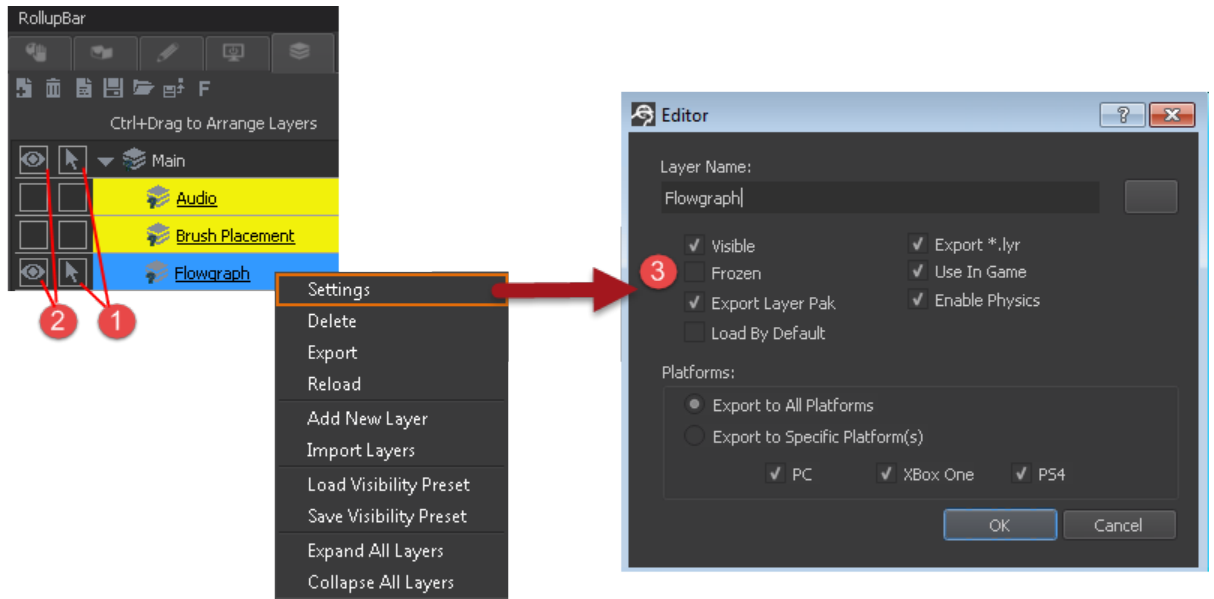
- On the main menu, click **View, Open View Pane, Object Selector**.
- Press **Ctrl+T**.
- On the top toolbar, click the **Object Selector** icon.



Finding an Object

You may sometimes find it difficult to select an object in your level, particularly when you have a large number of objects, or when other objects are surrounding or overlapping the object you want to select. The **Object Selector** provides several tools to help you find specific objects.

The Object Selector displays objects on [layers \(p. 872\)](#) that are selectable (1), visible (2), and not frozen (3).



To automatically select objects (in your **Perspective** viewport) when you click them in the list, enable the **Auto Select** option (bottom right).

To display objects with parent/child relationships, enable the **Display as Tree** option. When this option is enabled, each type of object is displayed with its icon, and grouped objects are shown as a tree in the list. If you have no grouped objects, you see only individual objects listed.

You can also use **Fast Select** to extend your search to include objects within prefabs and groups. To do this, enable **Search also inside Prefabs and Groups** (below **Fast Select**).

To find and select an object

1. [Open \(p. 426\) the Object Selector.](#)
2. Do one or both of the following:
 - If you know the object's name, type it into the **Fast Select** box at the bottom.
 - Select one or more of the **List Types** (on the right):
 - **Entities**
 - **Brushes**
 - **Prefabs**
 - **Tag Points**
 - **AI Points**
 - **Groups**
 - **Volumes**
 - **Shapes**
 - **Solids**
 - **Other**
3. Click the object(s) you want to select.
4. Click **Select** (on the right) to place an **X** for each selected object in the **Selected** column.

You can also use:

- **Select All** to select all currently listed objects
- **Select None** to deselect all objects.

- **Invert Selection** to deselect currently selected objects and select all the other listed objects.
5. Close the **Object Selector** to return to your **Perspective** viewport.
 6. Press **Z** on your keyboard to focus on the object(s) you selected.

Managing Objects

The **Object Selector** can also hide (and unhide), freeze (and unfreeze), and delete listed objects. You can perform these actions on objects that are contained in layers that are currently selectable, visible, and not frozen.

To hide or freeze objects

1. Find the object(s) you want to hide or freeze.
2. Click the object(s). To select multiple objects, use **Ctrl** or **Shift**.

Note

For this procedure, you need only click to select. There is no need to click the **Select** button on the right side of the **Object Selector**.

3. Click **Hide** or **Freeze**.

Clicking **Hide** hides your object(s) in the **Object Selector** list and in your **Perspective** viewport.

Clicking **Freeze** hides your object(s) in the **Object Selector** list and makes it unable to be interacted with in the **Perspective** viewport.

Other Actions

Task	Steps
To unhide hidden objects	Select the objects, and then click Unhide .
To view frozen objects	Click the Frozen option under Display List .
To unfreeze frozen objects	Select the objects, and then click Unfreeze .
To delete objects	Find the objects, click to select them, and then click Delete Selected . This deletes the objects from the Object Selector and from your level.

Object Selector Table

The objects in your level are listed in a table in the **Object Selector** window. To sort your displayed objects, click a column header. The results appear in alphabetic order.

Column Name	Description
Name	Name of the object.
Selected	X is displayed when object is selected in Perspective viewport.

Column Name	Description
Type	Scene element type of the object (entity, brush, prefab, tag point, AI point, group, volume, shape, solid, other).
Layer	Layer to which the object is assigned (objects on invisible or frozen layers are not displayed).
Default Material	Path to object's default material.
Custom Material	Path to object's customer material, if assigned.
Breakability	Type of breakability the object supports.
Track View	Traview that the object is used in.
FlowGraph	Flow graph that the object is used in.
Geometry	Path to the object's geometry, if applicable.
Instances In Level	Number of times the object is used in the level.
Number of LODs	Number of LODs (p. 1374) the object has.
Spec	The minimum specification that the object is set to display on.
AI GroupID	Group ID number associated with an AI character.

Brushes

Brushes are solid objects that cannot be modified or moved dynamically during gameplay, except if they have a break-point specified in the asset file, for example a breakable wooden shack.

Typically brushes are static objects placed in a level. They are one of the cheapest rendered objects as they don't have any of the entity or physics overhead of other objects. A large percentage of the visual objects in your levels will consist of brushes.

Brush Parameter Table

Parameter	Description
Geometry	This option specifies the geometry that needs to be used for the object.
CollisionFiltering	
Type	<ul style="list-style-type: none"> Ship Shield Asteroid
Ignore	<ul style="list-style-type: none"> Ship Shield Asteroid

Parameter	Description
OutdoorOnly	When set, the object will not be rendered when inside a visarea.
CastShadowMaps	When this option is set, the object will cast shadows onto other geometry/terrain/etc.
RainOccluder	Set the brush to occlude rain, this works in conjunction with Rain Entity. If your level does contain rain, you should set this wisely, as there is a limit of 512 objects that can occlude at any given time.
SupportSecondVisarea	Normally, objects are considered to be in only one visarea. This option allows them to be added to multiple visareas if their bounding box overlaps them, at the cost of some performance. Without this option, some large objects may not be displayed when viewed through portals in certain situations.
Hideable	When this option is set, AI will use this object as a hiding spot, using the specified hide point type.
LodRatio	Defines how far from the current camera position, the different Level Of Detail models for the object are used.
ViewDistanceMultiplier	Sets the distance from the current view at which the object renders.
NotTriangulate	Deprecated
AIRadius	Deprecated
NoStaticDecals	When this option is set, decals will not project onto the object.
NoAmbShadowCaster	When this option is set, no ambient shadows will be cast.
RecvWind	When this option is set, the object will be affected by the level wind.
Occluder	Used for the construction of a level occlusion mesh.
DrawLast	This function is exposed to give per-object control over alpha-sorting issues. An example can be seen below.

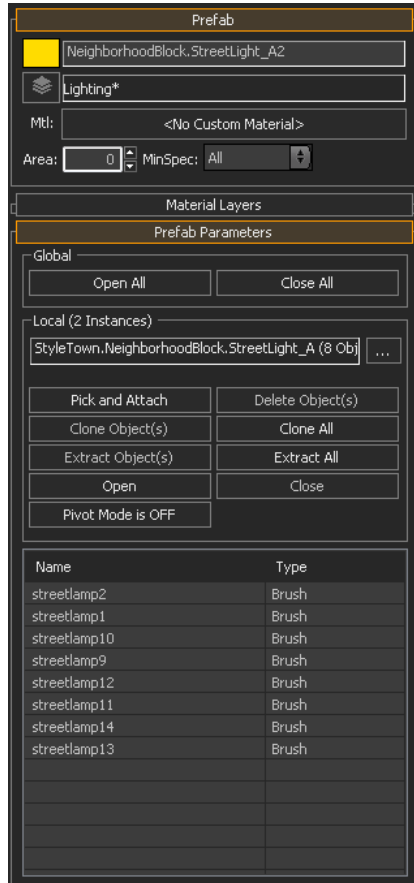
DrawLast

The **DrawLast** effects in front of glass objects. By enabling **DrawLast**, Lumberyard knows that any alpha based objects rendered between the player and itself should take ordering priority.

Prefabs

Prefabs are groups of objects that can be placed in the level as instances. An instance is an object that is an exact copy of every other object of the same type. Altering one prefab universally applies the changes to each instance of the prefab object. Any alterations need to be saved to the Prefab Library to ensure they are correctly propagated across the entire game.

The Prefabs Library is a tab in the Database View editor, and lists all the prefab objects that are available for a specific level.



Prefab Parameter Table

Parameter	Description
Global	
Open All	Open all instances of this prefab inside the level.
Close All	Close all instances of this prefab inside the level.
Local	
Pick and Attach	Allows you to add a new object to the selected prefab, by clicking on it.
Delete Object(s)	Allows you to delete one or more objects from the selected prefab.
Clone Object(s)	Allows you to clone one or more objects from the selected prefab.
Clone All	Clones all instances of this prefab inside the level.
Extract Object(s)	Extracts a clone of a single object from the prefab, without altering or removing anything from the prefab object itself.
Extract All	Extracts all the objects from the prefab, without altering the Prefab Library.
Open	Opens the prefab group, allowing you to edit and manipulate objects within it.

Parameter	Description
Close	Closes the prefab so that internal objects cannot be individually edited.

Common Parameters and Properties

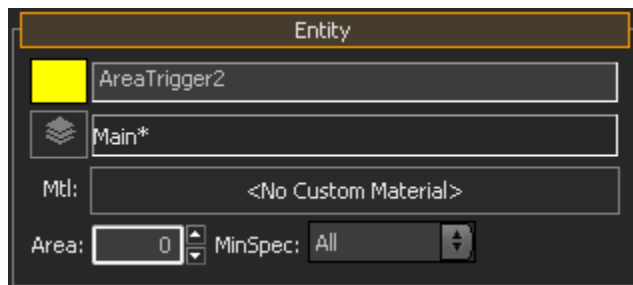
Many entities share common parameters and properties, as follows.

Entity Properties

The **Entity** pane is where you basic entity properties like the name of your object or the currently selected layer. The text box at the top of the pane allows you to enter a new name for your object.

Some entities have color schemes applied by default, depending on their type. The colored box next to the text box opens the color editor window.

Clicking the layers button opens the layer window, allowing you to place your object in the appropriate layer. The text to the right of the layer button tells you which layer is currently selected.

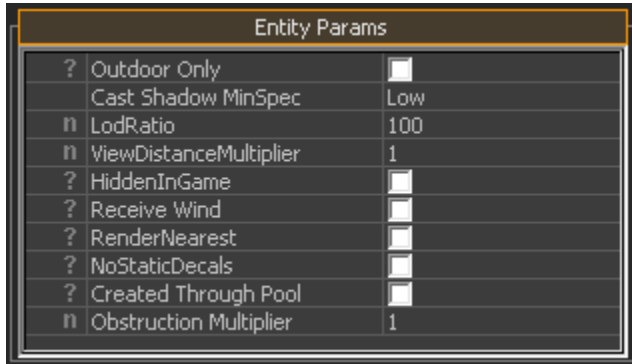


Standard Parameters

Property	Description
Area	Changing the value of Area increases or decreases the radius of the on screen object placement helper.
Mtl button	When you select an object that uses a material, clicking the Mtl (material) button opens the material window and allows you to pick your desired material. When you have assigned a custom material to be applied to the object, its path will be displayed in the Mtl button.
MinSpec	When set, the selected object only appears in game detail settings of the desired value and above.

Entity Parameters

The **Entity Params** panel lists all common entity parameters. Modifying these parameters enables effects such as wind and shadow to be added to an object and also toggles options such as hiding the object in-game.

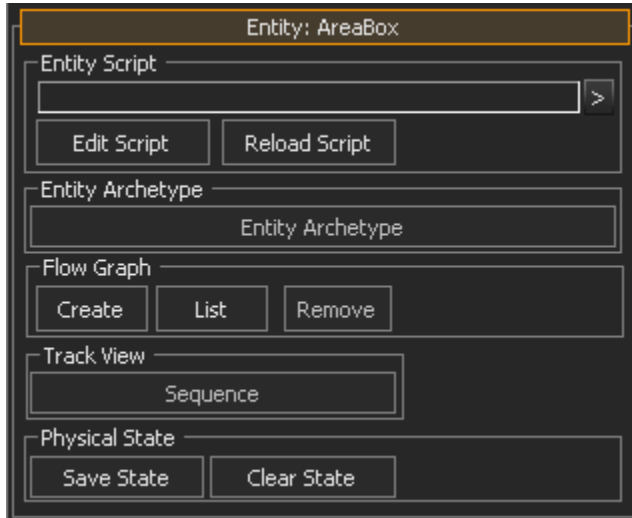


Entity Parameters

Property	Description
Outdoor Only	When set, the object will not be render when inside a VisArea.
Cast Shadow MinSpec	When set, this object will cast a shadow on the selected quality setting and above.
LodRatio	Defines how far from the current camera position that different Level Of Detail models for the object are used.
ViewDistRatio	Defines how far from the current camera position that the object will be rendered.
HiddenInGame	When set, this object will not be shown in pure game mode. Useful for debugging or prototyping.
Receive Wind	When set, this object will be influenced by any wind setup in your level.
RenderNearest	Used to eliminate z-buffer artifacts when rendering in first person view.
NoStaticDecals	If this is set to true, decals will not be rendered on this object.
Created Through Pool	This is mostly used on AI entities for memory optimization.

Scripting and Flow Graph Entity Parameters

This pane contains parameters related to entity scripting and Flow Graph.



Scripting and Flow Graph Parameters

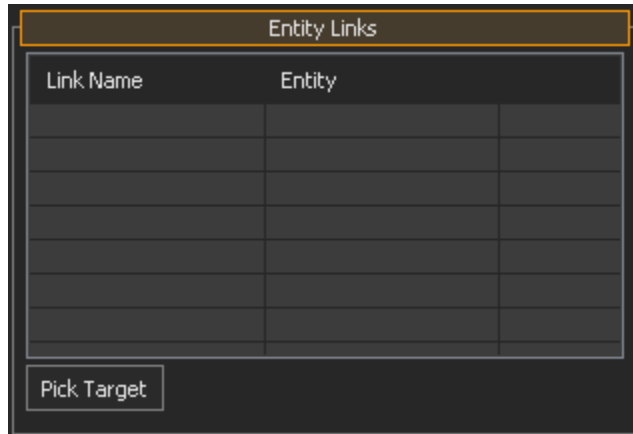
Property	Description
Edit Script	Opens the script file in your associated program and allows you to modify the script for the selected entity. The script file location is shown above this and the Reload Script button. Clicking the '>' button will give you more options related to this file.
Reload Script	Used to implement any changes made to the script. This is particularly useful for reviewing particle effects as reloading it activates it again.
Entity Archetype	If the entity is an Archetype entity, the name of entity will appear on the button and clicking will open the archetype in the Database View tool.
Create	Creates a new flow graph.
List	Lists the flow graphs that the selected entity is associated with.
Remove	If a flow graph was created for this entity, you can remove.
Sequence	If the entity is being used in a Track View sequence, the name of the sequence will be displayed here. Also open up the sequence in the Track View Window.
Save State	Saves the physical state of a selected entity (when AI/Physics is turned on). This can be useful for placing physical props realistically around your level without having to manually rotate and align their positions.
Clear State	Clears any saved physical state.

Entity Links

This pane displays entities linked to the main entity. Each entity can link to multiple entities. Creating an entity link is essentially making a dynamic link that can be referenced in LUA script.

To pick a target, click the **Pick Target** button and then select the desired entity to create a link. You can select multiple entities one at a time while the button is still active.

Double-click a linked entity in the list to select it. Right-click opens a menu with additional commands.



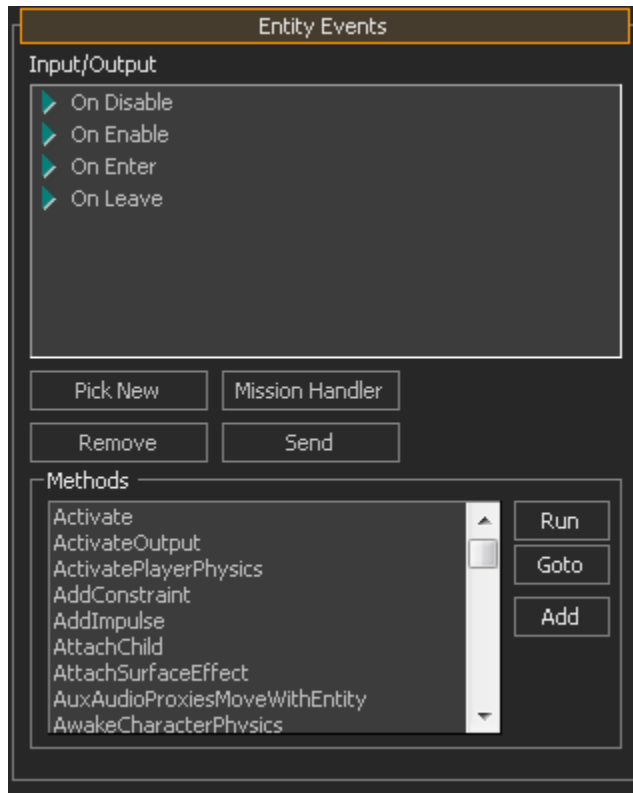
Entity Links Parameters

Property	Description
Change Target Entity	This will change the Entity associated with a link.
Rename Link	Renames the selected link.
Delete Link	Deletes the selected link.
Pick New Target	Same functionality as the Pick Target button.

Entity Events

This pane visually represents the script behind objects and allows you to edit and run the script. When **AI/Physics** is enabled you can test the effect of any changes you have made to the entity script.

AI/Physics should be enabled to test events.



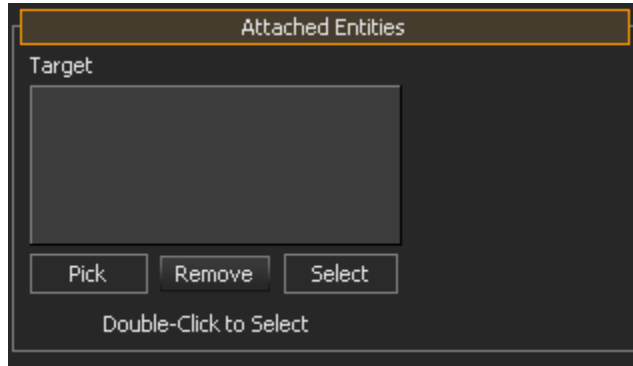
Entity Event Parameters

Property	Description
Input/Output	Displays a list of executable script commands.
Pick New	Deprecated
Mission Handler	Deprecated
Remove	Deprecated
Send	Once you have chosen an Input/Output event click Send to test and see the effect. For example, an Input event called OnKill might kill an entity and OnSpawn might spawn them back to life.
Methods	Displays a list of executable methods.
Run	Displays a list of executable methods.
Goto	Deprecated
Add	Deprecated

Attached Entities

This pane enables you to create links to other objects in the perspective viewport.

This pane is only visible for certain entities.



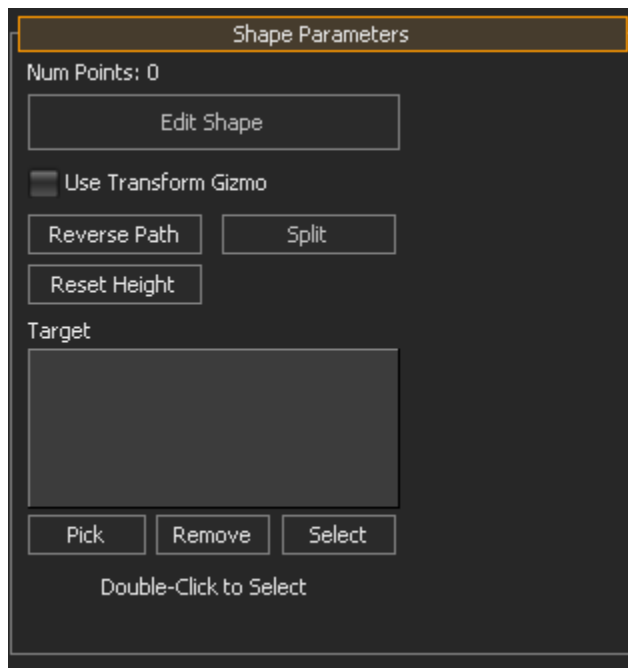
Attached Entity Parameters

Property	Description
Pick	Links two selected objects. You will visually see the link in the viewport and see the object name in the target window.
Remove	Removes a link between two objects.
Select	Selects an object from the target window. Double-clicking the object name in the target window will also select the object.

Shape Parameters

This pane allows you to edit the area of effect for a shape and create links to other objects in the viewport.

This pane is only visible for certain entities.



Shape Parameters

Property	Description
Num Points	Relates to the number of points the shape contains in the perspective viewpoint.
Edit Shape	Allows you to edit the selected shape.
Use Transform Gizmo	Enables the Transform Gizmo helper.
Reverse Path	Used with objects like AIPath and when clicked will reverse the AI path. The arrow on screen will point in the opposite direction to show the new path direction.
Split	Click two parts of your shape to split your shape and create a new independent shape.
Reset Height	Use to flatten the shape and all other points to the height of the selected point.
Pick	Links a shape to an object. You will visually see the link in the viewport and see the object name in the target window.
Remove	Removes a link between the selected shape and an object.
Select	Selects an object from the target window. Double-clicking the object name in the target Window will also select the object.

Entity Reference

The following is a complete list of the various entities that comprise the Entity system.

Topics

- [Actor Entity \(p. 439\)](#)
- [AI Control Objects \(p. 439\)](#)
- [Anim Entities \(p. 444\)](#)
- [Archetype Entity \(p. 444\)](#)
- [Area Entities \(p. 445\)](#)
- [Audio Entities \(p. 451\)](#)
- [Boid Entity \(p. 457\)](#)
- [Camera Entities \(p. 460\)](#)
- [Geom Entities \(p. 460\)](#)
- [Light Entities \(p. 461\)](#)
- [Lightning Arc Entity \(p. 465\)](#)
- [Miscellaneous Entities \(p. 467\)](#)
- [Particle Entities \(p. 469\)](#)
- [Physics Entities \(p. 470\)](#)
- [Rain Entity \(p. 478\)](#)
- [Render Entities \(p. 479\)](#)
- [River Entity \(p. 480\)](#)
- [Road Entity \(p. 481\)](#)
- [Rope Entity \(p. 482\)](#)

- [Snow Entity \(p. 484\)](#)
- [Tornado Entity \(p. 484\)](#)
- [Trigger Entities \(p. 485\)](#)

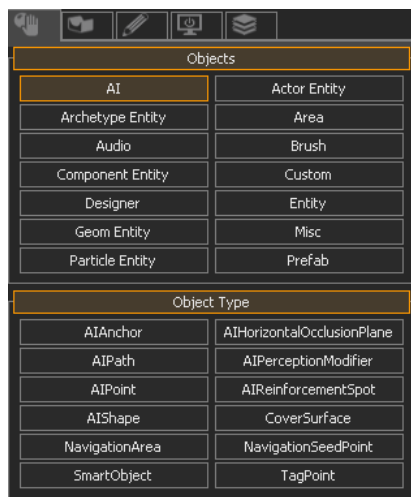
Actor Entity

This is a specialized entity that is the basis for characters in a game.

AI Control Objects

The following AI entities are provided:

- AIAnchor
- AI Horizontal Occlusion Plane
- AI Path
- AI Perception Modifier
- AI Point
- AI Reinforcement Spot
- AI Shape
- Cover Surface
- Navigation Area
- Navigation Seed Point
- Smart Object
- Tag Point



AIAnchor

An AIAnchor is a positional point object that can be used to define specific behaviors for an AI with reference to the location and/or direction of the anchor.

AIHorizontalOcclusion Plane

AI agents above and below an AI Horizontal Occlusion Plane will not be able to see through it. It can be used, for example, to restrict an AI on a high ledge from being able to see below the ledge.

Parameters

Parameter	Description
Width	Specifies how wide the entity is.
Height	Specifies how high the shape area should be (0 means infinite height).
Areald	Sets up the ID of the area, so areas with another ID can overlap.
Groupld	Sets up the Group ID of the area, so areas with another group ID can overlap.
Priority	Defines the Priority so areas with a higher priority will be processed first.
Closed	Sets if the area should be closed or if it should be just a line.
DisplayFilled	Just for visibility in the editor this option defines if the area should be rendered as filled or not.
DisplaySoundInfo	Enable to expand Sound Obstruction options.
Agent_height	When Render_voxel_grid is enabled this determines the height - along the y axis - of the grid cells rendered.
Agent_width	When Render_voxel_grid is enabled this determines the height - along the x axis - of the grid cells rendered.
Render_voxel_grid	If true, voxel grid will be rendered when helpers are enabled.
voxel_offset_x	Offset voxel grid on the X axis.
voxel_offset_y	Offset voxel grid on the Y axis.

AI Path

An AI path is an object which can be used to guide your AI agent along a specific route from point to point in your level.

Parameters

Parameter	Description
Road	Defines if the path is to be used by vehicles as a preferred path.
PathNavType	Sets the AI navigation type of the path. Types of paths available: <ul style="list-style-type: none"> • Flight • Free 2D • Road • Smart Object • Triangular • Unset • Volume • Waypoint 3D Surface • Waypoint Human

Parameter	Description
AnchorType	Sets an AI behavior for any AI using the path.
ValidatePath	Used for 3D Volume paths only, checks and displays path validity in the editor.
Width	Specifies how wide the entity is.
Height	Specifies how high the shape area should be (0 means infinite height).
Areald	Sets up the ID of the area, so areas with another ID can overlap.
Groupld	Sets up the Group ID of the area, so areas with another group ID can overlap.
Priority	Defines the Priority so areas with a higher priority will be processed first.
Closed	Sets if the area should be closed or if it should be just a line.
DisplayFilled	Just for visibility in the editor this option defines if the area should be rendered as filled or not.
DisplaySoundInfo	Enable to expand Sound Obstruction options.
Agent_height	When Render_voxel_grid is enabled this determines the height - along the y axis - of the grid cells rendered.
Agent_width	When Render_voxel_grid is enabled this determines the height - along the x axis - of the grid cells rendered.
Render_voxel_grid	If true, voxel grid will be rendered when helpers are enabled.
voxel_offset_x	Offset voxel grid on the X axis.
voxel_offset_y	Offset voxel grid on the Y axis.

AI Perception Modifier

Parameters

Parameter	Description
Height	Specifies how high the shape area should be (0 means infinite height).
Closed	Sets if the area should be closed or if it should be just a line.
DisplayFilled	Just for visibility in the editor this option defines if the area should be rendered as filled or not.

AI Point

An AI Point is an object that represents a named AI waypoint in your level.

Parameters

Parameter	Description
Type	<ul style="list-style-type: none"> Waypoint Hide Sec Hide Entry/Exit Exit-only
Nav Type	<ul style="list-style-type: none"> Human 3D Surface
Removable	Allows AI points to be removed, may be useful for adding entrances for easier traversing.
Regen Links	Prompts a regeneration of all links in the same navigation region as this one.
Linked Waypoints	Displays the list of waypoints that are connected to this point.
Pick	Allows the user to pick a second waypoint to create a permanent AI link.
Pick impass	Allows the user to pick a second waypoint to create a permanent non-passable link.
Select	Selects the currently highlighted link in the linked waypoints box.
Remove	Removes the currently highlighted waypoint links.
Remove all	Removes all waypoint links from the AI Point.
Remove all in area	Removes all waypoint links in the nav area.

AI Reinforcement Spot

Defines a point which any relevant AI can use to trigger their reinforcement behavior.

AI Shape

An AI shape is an object which can be used to define an area which AI will use for combat and will search for anchors within.

Parameters

Parameter	Description
AnchorType	Affects AI behaviors in the same way as the anchors do. The main usage is to check if a point (AI position, target position, etc) is inside a shape of a given AnchorType, in the same way as checking the proximity to an anchor of a given type.
LightLevel	Affects AI's ability to see (including sight range and speed of detection).
Width	Specifies how wide the entity is.

Parameter	Description
Height	Specifies how high the shape area should be (0 means infinite height).
Areald	Sets up the ID of the area, so areas with another ID can overlap.
Groupld	Sets up the Group ID of the area, so areas with another group ID can overlap.
Priority	Defines the Priority so areas with a higher priority will be processed first.
Closed	Sets if the area should be closed or if it should be just a line.
DisplayFilled	Just for visibility in the editor this option defines if the area should be rendered as filled or not.
DisplaySoundInfo	Enable to expand Sound Obstruction options.
Agent_height	When Render_voxel_grid is enabled this determines the height - along the y axis - of the grid cells rendered.
Agent_width	When Render_voxel_grid is enabled this determines the height - along the x axis - of the grid cells rendered.
Render_voxel_grid	If true, voxel grid will be rendered when helpers are enabled.
voxel_offset_x	Offset voxel grid on the X axis.
voxel_offset_y	Offset voxel grid on the Y axis.

Cover Surface

Cover surfaces can be used to allow the AI agent to take cover in combat situations.

Parameters

Parameter	Description
Limit Left	The generated cover path to the left side of the cover surface object will be limited to this length.
Limit Right	The generated cover path to the right side of the cover surface object will be limited to this length.
Limit Height	The resulting height of all cover surfaces will be limited to this value.

Navigation Area

For more information, see [Creating Navigation Areas \(p. 71\)](#).

Parameters

Parameter	Description
Height	The height of the navigation area.

Parameter	Description
DisplayFilled	Just for visibility in the editor this option defines if the area should be rendered as filled or not.

Navigation Seed Point

For more information, see [Adding Navigation Seed Points \(p. 73\)](#).

Smart Object

An AI Anchor is a point or collection of points which can be used by AI to perform a specific action or event, such as an animation or behavior. Certain smart objects can have special geometry assigned to them, to assist with object placement.

Tag Point

An AI Tagpoint is an object used to define a location.

Anim Entities

MannequinObject Entity

Entity Properties

Property	Description
ActionController	The root object controlling mannequin for a character. It is configured using a controller definition (defining the fragmentIDs, scopes, and scope contexts). It schedules actions onto scopes and holds the global tagstate.

Archetype Entity

An Archetype entity is based on a regular entity and specifies individual parameter values for that entity. If the value of an Archetype parameter is changed, all instances of that Archetype in the level are updated automatically.

As such, you can predefine variations of entity classes as Archetype Entities that can be used throughout the game. For global changes affecting all instances, the Archetype Entity just needs to be changed once.

EntityArchetype Parameters

Parameter	Description
Outdoor Only	When set, the object will not be rendered when inside a visarea.
Cast Shadow MinSpec	When set, the object will cast a shadow.
LodRatio	Defines how far from the current camera position, the different Level Of Detail models for the object are used.

Parameter	Description
ViewDistanceMultiplier	Defines how far from the current camera position, the the object can be seen.
HiddenInGame	When set, this object is not shown in the pure game mode.
Receive Wind	When set, this object will be influenced by any wind setup in the level.

Area Entities

Area entities are used to create three dimensional zones in the level that can be used to trigger events.

The following area entities can be accessed from the **Area** button on the Objects tab of the Rollup Bar.

- AreaBox
- AreaSolid
- AreaSphere
- ClipVolume
- OccluderArea
- OccluderPlane
- Portal
- Shape
- VisArea
- WaterVolume

AreaBox

This entity lets you create a box to which you can link triggers and other entities that should be enabled when the player enters or leaves the box.

Parameter Table

Parameter	Description
Areald	Sets up the ID of the area, so areas with another ID can overlap.
FadeInZone	Specifies in meters how big the zone around the box is that is used to fade in the effect attached to the box. Only when the player is inside the box the effect is rendered at 100%, at the beginning of the FadeInZone its rendered at 0%.
Width	Specifies how wide the box is.
Length	Defines how long the box is.
Height	Specifies how high the shape area should be (0 means infinite height).
GroupId	Sets up the Group ID of the area, so areas with another group ID can overlap.
Priority	Defines the Priority so areas with a higher priority will be processed first.

Parameter	Description
DisplayFilled	Just for visibility in the editor this option defines if the area should be rendered as filled or not.
DisplaySoundInfo	Enable to expand Sound Obstruction options.

AreaSolid

The AreaSolid is for defining complex range of sound obstructions with the Designer tool that is used for geometry editing.

AreaSphere

The AreaSphere object is used to link triggers and other entities that should be enabled when the player enters or leaves the sphere.

Parameter Table

Parameter	Description
Areald	Sets up the ID of the area, so areas with another ID can overlap.
FadeInZone	Specifies in meters how big the zone around the box is that is used to fade in the effect attached to the box. Only when the player is inside the box the effect is rendered at 100%, at the beginning of the fadeinzone its rendered at 0%.
Radius	Specifies how big the sphere should be.
GroupId	Sets up the Group ID of the area, so areas with another group ID can overlap.
Priority	Defines the Priority so areas with a higher priority will be processed first.
Filled	Just for visibility in the editor this option defines if the area should be rendered as filled or not.

Clip Volume

ClipVolumes define geometric shapes that can restrict the influence of lights and cubemaps in a level.

Lights can be associated with ClipVolumes by either placing the light directly inside the object or by creating an entity link from the light to the ClipVolume. Once an association has been established, the **AffectsThisAreaOnly** property on the light source will clip the light's influence to the geometry inside the ClipVolume.

Here are some restrictions on the use of ClipVolume objects:

- The Clip Volume mesh needs to be watertight.
- Clip Volume mesh complexity has an impact on performance.
- ClipVolumes must not overlap.
- Due to performance reasons, forward rendered objects perform the inside test based on their pivot only.
- Each light can be linked to a maximum of two ClipVolumes.

OccluderArea

The OccluderArea object prevents Lumberyard from rendering everything that is behind it. It is used for performance optimization in areas where automatic occlusion from brushes and terrain don't work very well. This object allows you to create an occlusion plane out of a custom shape with multiple edges, unlike an OccluderPlane object which can only be a square shape.

Parameter Table

Parameter	Description
DisplayFilled	Just for visibility in the editor this option defines if the area should be rendered as filled or not.
CullDistRatio	Specifies at what distance the culling effect should stop occurring.
UseIndoors	Specifies if the occluder area should be working inside an indoor visarea.

OccluderPlane

The OccluderPlane object is used to occlude objects behind the plane. Like with the OccluderArea object, this typically isn't required because occlusion is done automatically. This object can be used as a fallback method.

Parameter Table

Parameter	Description
Height	Specifies how high the occluder plane is.
DisplayFilled	Just for visibility in the editor this option defines if the plane should be rendered as filled or not.
CullDistRatio	Specifies at what distance the culling effect should stop occurring.
UseIndoors	Specifies if the occluder plane should work inside a visarea.
DoubleSide	Specifies if the occluder plane should work from both sides.

Portal

With Portals you can cut holes inside a VisArea to create an entrance into a VisArea. Portals have to be smaller than the VisArea Shape but thick enough to protrude both the inside and outside of the VisArea, like a door.

You can enable and disable Portals using Flow Graph and you can have multiple Portals in one VisArea.

Parameter Table

Parameter	Description
Height	Specifies how high the portal is.
DisplayFilled	Just for visibility in the editor this option defines if the area should be rendered as filled or not.

Parameter	Description
AffectedBySun	Specifies if shadows from the world outside the visarea can travel inside.
IgnoreSkyColor	If this option is turned off the ambient color (sky color in time of day window) is not used indoors.
IgnoreGI	If true, Global Illumination won't be used inside this object.
ViewDistRatio	Specifies how far the visarea is rendered.
SkyOnly	Lets you choose to see only the skybox when you look outside the visarea. If you don't render terrain and outside brushes the performance can be faster so use this option when it is appropriate.
OceanIsVisible	Specifies if the ocean rendering should be visible inside the visarea.
UseDeepness	Specifies if the portal should be working from both sides.
DoubleSide	Specifies if the portal should be working from both sides.

Shape

The Shape object lets you create a shape to which you can link triggers and other entities that should be enabled when the player enters or leaves the area shape.

Parameter Table

Parameter	Description
Width	Specifies how wide the entity is.
Height	Specifies how high the shape area should be (0 means infinite height).
Areald	Sets up the ID of the area, so areas with another ID can overlap.
GroupId	Sets up the Group ID of the area, so areas with another group ID can overlap.
Priority	Defines the Priority so areas with a higher priority will be processed first.
Closed	Sets if the area should be closed or if it should be just a line.
DisplayFilled	Just for visibility in the editor this option defines if the area should be rendered as filled or not.
DisplaySoundInfo	Enable to expand Sound Obstruction options.
Agent_height	When Render_voxel_grid is enabled this determines the height - along the y axis - of the grid cells rendered.
Agent_width	When Render_voxel_grid is enabled this determines the height - along the x axis - of the grid cells rendered.
Render_voxel_grid	If true, voxel grid will be rendered when helpers are enabled.

Parameter	Description
voxel_offset_x	Offset voxel grid on the X axis.
voxel_offset_y	Offset voxel grid on the Y axis.

VisArea

The VisArea object is used to define indoor areas for culling and optimization purposes, as well as lighting. Objects inside a VisArea won't be rendered from outside and vice versa, this can help with performance immensely.

VisAreas also can be setup to occlude certain lighting elements such as the sun, which gives flexibility in setting up lighting for your indoor areas.

1. In Rollup Bar, on the Objects tab, select **Area, VisArea**.
2. Place the Visarea object around the desired area in your level and set the **Height** parameter value. Keep the shape of the VisArea as simple as possible.
3. Ensure everything related is inside the VisArea.
4. Enable **Snap To Grid**.

Parameter Table

Parameter	Description
Height	Specifies how high the visarea is.
DisplayFilled	Just for visibility in the editor this option defines if the area should be rendered as filled or not.
AffectedBySun	Specifies if shadows from the world outside the visarea can travel inside.
IgnoreSkyColor	If this option is turned off the ambient color (sky color in time of day window) is not used indoors.
IgnoreGI	If true, Global Illumination won't be used inside this object.
ViewDistRatio	Specifies how far the visarea is rendered.
SkyOnly	Lets you choose to see only the skybox when you look outside the visarea. If you don't render terrain and outside brushes the performance can be faster so use this option when it is appropriate.
OceanIsVisible	Specifies if the ocean rendering should be visible inside the visarea.

WaterVolume

The WaterVolumes object is used for rivers, lakes, pools, puddles, and oceans. For more information about WaterVolumes, see [WaterVolume Shader \(p. 1032\)](#).

Parameter Table

Parameter	Description
Width	Specifies how wide the entity is.
Height	Specifies how high the shape area should be (0 means infinite height).
Areald	Sets up the ID of the area, so areas with another ID can overlap.
Groupld	Sets up the Group ID of the area, so areas with another group ID can overlap.
Priority	Defines the Priority so areas with a higher priority will be processed first.
Closed	Sets if the area should be closed or if it should be just a line.
DisplayFilled	Just for visibility in the editor this option defines if the area should be rendered as filled or not.
DisplaySoundInfo	Enable to expand Sound Obstruction options.
Agent_height	When Render_voxel_grid is enabled this determines the height - along the y axis - of the grid cells rendered.
Agent_width	When Render_voxel_grid is enabled this determines the height - along the x axis - of the grid cells rendered.
Render_voxel_grid	If true, voxel grid will be rendered when helpers are enabled.
voxel_offset_x	Offset voxel grid on the X axis.
voxel_offset_y	Offset voxel grid on the Y axis.
Depth	Sets the depth of the river.
Speed	Defines how fast physicalized objects move along the river. Use negative values to move in the opposite direction.
UScale	Sets the texture tiling on the U axis.
VScale	Sets the texture tiling on the V axis.
View Distance Multiplier	Sets the distance from the current view at which the object renders.
Caustics	Enables optical caustics effects.
CausticIntensity	Scales the intensity of the caustics for the water surface normals.
CausticTiling	Scales the caustic tiling applied to the water surface normals. It allows the scaling of caustics independently from the surface material.
CausticHeight	Sets the height above the water surface at which caustics become visible. Use this to make caustics appear on overhanging landforms or vegetation and other nearby objects.

Parameter	Description
FixedVolume	Traces a ray down to find a 'vessel' entity and 'spill' the requested amount of water into it. For static entities, it attempts to boolean-merge any surrounding static that intersects with the first vessel (use the No Dynamic Water flag on brushes that do not need that).
VolumeAccuracy	Water level is calculated until the resulting volume is within this (relative) difference from the target volume (if set to 0 it runs up to a hardcoded iteration limit).
ExtrudeBorder	Extrudes the border by this distance. This is particularly useful if wave simulation is enabled as waves can raise the surface and reveal open edges if they lie exactly on the vessel geometry.
ConvexBorder	Takes convex hull of the border. This is useful if the border would otherwise have multiple contours, which areas do not support.
ObjectSizeLimit	Only objects with a volume larger than this number takes part in water displacement (set in fractions of FixedVolume).
WaveSimCell	Size of cell for wave simulation (0 means no waves). Can be enabled regardless of whether FixedVolume is used.
WaveSpeed	Sets how "fast" the water appears.
WaveDamping	Standard damping.
WaveTimestep	This setting may need to be decreased to maintain stability if more aggressive values for speed are used.
MinWaveVel	Sleep threshold for the simulation.
DepthCells	Sets the depth of the moving layer of water (in WaveSimCell units). Larger values make waves more dramatic.
HeightLimit	Sets a hard limit on wave height (in WaveSimCell units).
Resistance	Sets how strongly moving objects transfer velocity to the water.
SimAreaGrowth	If changing water level is expected to make the area expand, the wave simulation grid should take it into account from the beginning. This sets the projected growth in fractions of the original size. If wave simulation is not used, this setting has no effect.

Audio Entities

There are four Audio entities, as follows:

- Audio Trigger Spot Entity
- Audio Area Entity
- Audio Area Ambience Entity
- Audio Area Random Entity

Audio Trigger Spot

The **AudioTriggerSpot** triggers an event on a specific position. This position can be automatically randomized on each axis or with time delays.



Audio Trigger Spot Properties

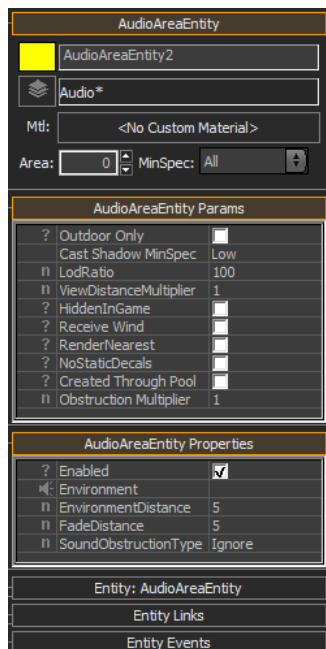
Properties	Description
Enabled	Defines whether the Entity is enabled (playing) or disabled (not playing).
MaxDelay	The maximum delay in seconds that it takes to trigger the sound when PlayRandom is enabled.
MinDelay	The minimum delay in seconds that it takes to trigger the sound when PlayRandom is enabled.
PlayOnX	Defines whether the sound gets positioned randomly on the X-axis when PlayRandom is enabled.
PlayOnY	Defines whether the sound gets positioned randomly on the Y-axis when PlayRandom is enabled.
PlayOnZ	Defines whether the sound gets positioned randomly on the Z-axis when PlayRandom is enabled.
PlayRandom	When the check box is enabled: The sound is triggered at random intervals between the MinDelay and MaxDelay settings used and on the PlayOnX , PlayOnY , or PlayOnZ axis that has been selected. When the check box is not enabled, the sound is played immediately on the entity.
PlayTriggerName	Name of the play event.

Properties	Description
RadiusRandom	The radius in meters in which the sound gets positioned randomly when PlayRandom is enabled.
SerializePlayState	Defines whether the play state of the entity gets saved and loaded at checkpoints.
SoundObstructionType	<p>Sets the number of ray casts that are used to calculate the obstruction. More ray casts used equals a greater performance requirement, but creates a more accurate result.</p> <ul style="list-style-type: none"> • Ignore – No raycasts are applied and the sound is unaffected by other objects in the game. • Single Ray – Used to calculate the treatment the sound receives depending on the position and physical properties of the objects found between the source and the listener. • Multiple Rays – Used to calculate the treatment the sound receives depending on the position and physical properties of the objects found between the source and the listener.
StopTriggerName	Name of the stop event.

Audio Area Entity

Audio Area Entities are used to play ambient sounds in an area, and are linked to Area Shapes, Area Boxes, and Area Spheres.

These entities are an advanced method of setting up ambient sounds in levels and require Flow Graph logic to play and control the sounds. This opens up many possibilities and gives advanced control over the ambience. When setting up a basic ambient sound, use the **Audio Area Ambience** entity instead, which does not require any Flow Graph logic.

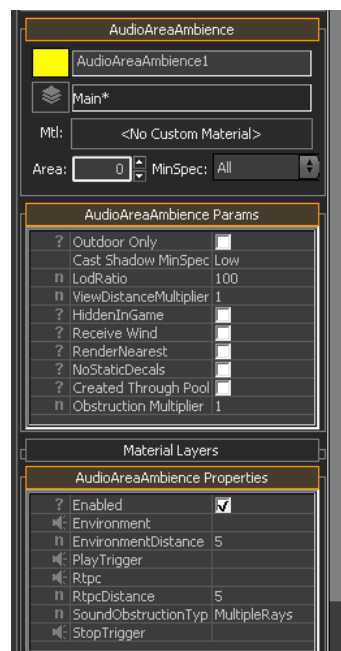


Audio Area Entity Properties

Properties	Description
Enabled	Defines whether the Entity is enabled (playing) or disabled (not playing).
Environment	Defines the name of the ATL environment used inside the connected shape.
EnvironmentDistance	The distance in meters from the edge of the assigned shape where the fading of the Environment begins.
FadeDistance	The distance in meters from the edge of the assigned shape where the Flowgraph Node is starting to output values.
SoundObstructionType	<p>Sets the number of ray casts that are used to calculate the obstruction. More ray casts used equals a greater performance requirement, but creates a more accurate result.</p> <ul style="list-style-type: none"> • Ignore – No raycasts are applied and the sound is unaffected by other objects in the game. • Single Ray – Used to calculate the treatment the sound receives depending on the position and physical properties of the objects found between the source and the listener. • Multiple Rays – Used to calculate the treatment the sound receives depending on the position and physical properties of the objects found between the source and the listener.

Audio Area Ambience

Audio Area Ambience entities are used to set up ambiances without having to define their functionality in Flow Graph. They are used when setting up basic ambient shapes in levels that do not require a more complex functionality.

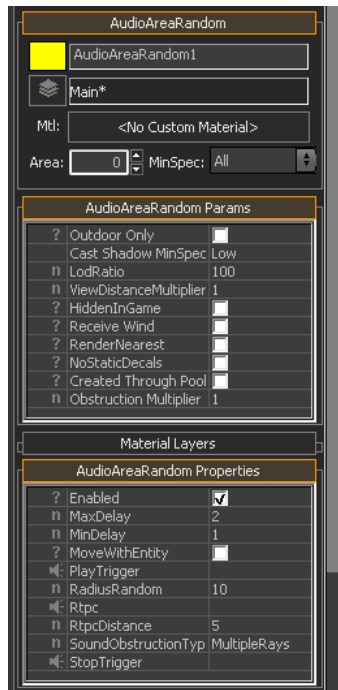


Audio Area Ambience Properties

Properties	Description
Enabled	Defines whether the Entity is enabled (playing) or disabled (not playing).
Environment	Defines the name of the ATL environment used inside the connected shape.
EnvironmentDistance	The distance in meters from the edge of the assigned shape where the fading of the environment begins.
PlayTrigger	Name of the play event.
Rtpc	Sets the RTPC that is controlling the playing of the sound object.
RtpcDistance	The distance in meters from the edge of the assigned shape where the connected RTPC is starting to receive values. The values sent to the RTPC are always from 0 to 1.
SoundObstructionType	<p>Sets the number of ray casts that are used to calculate the obstruction. More ray casts used equals a greater performance requirement, but creates a more accurate result.</p> <ul style="list-style-type: none">• Ignore – No raycasts are applied and the sound is unaffected by other objects in the game.• Single Ray – Used to calculate the treatment the sound receives depending on the position and physical properties of the objects found between the source and the listener.• Multiple Rays – Used to calculate the treatment the sound receives depending on the position and physical properties of the objects found between the source and the listener.
StopTrigger	Name of the stop event.

Audio Area Random

Audio Area Random entities trigger randomized shots in a confined area. The Entity needs to be linked to Area Shapes, Area Boxes, or Area Spheres. The sound is randomly triggered and positioned in a radius around the listener, providing they are inside the connected area.



Audio Area Random Properties

Properties	Description
Enabled	Defines whether the entity is enabled (playing) or disabled (not playing).
MaxDelay	The maximum delay in seconds it takes to trigger the sound.
MinDelay	The minimum delay in seconds it takes to trigger the sound.
MoveWithEntity	When enabled, the sound moves in relation to the listener after it has spawned; otherwise, it stays at its initial position.
PlayTrigger	Name of the play event.
RadiusRandom	Defines the size of the radius in which sounds spawn around the listener.
Rtpc	Sets the RTPC that is controlling the playing of the sound.
RtpcDistance	The distance in meters from the edge of the assigned shape where the connected RTPC is starting to receive values. The values sent to the RTPC range from 0 to 1.
SoundObstructionType	<p>Sets the number of ray casts that are used to calculate the obstruction. More ray casts used equals a greater performance requirement, but creates a more accurate result.</p> <ul style="list-style-type: none"> • Ignore – No raycasts are applied and the sound is unaffected by other objects in the game. • Single Ray – Used to calculate the treatment the sound receives depending on the position and physical properties of the objects found between the source and the listener.

Properties	Description
	<ul style="list-style-type: none"> • Multiple Rays – Used to calculate the treatment the sound receives depending on the position and physical properties of the objects found between the source and the listener.
StopTrigger	Name of the stop event.

Note

For each audio object, it is a good practice to see which **SoundObstructionType** works best and to select **Ignore** when there is no advantage gained from having obstruction and occlusion values calculated. Select **MultipleRays** only if the accuracy of the single raycast is not sufficient, or if you want the entity to be able to calculate both the occlusion and obstruction values separately.

Raycasts are skipped for entities that do not have an active playing trigger, even when the **SoundObstructionType** is set to **SingleRay** or **MultipleRays**.

Boid Entity

Boid entities simulate animals exhibiting group behavior, obstacle avoidance, animations, and sound. Their complex behavior arises from the interaction of an individual agent boid with other boids and the environment in which they move.

Not all parameters are available for all boid classes. For example, Behavior classes are needed only for the Bugs boid class and do not appear in other Boid properties.

Boid Properties

Properties	Description
Model1-5	Additional geometry for the boid; this can be a character (.CHR) or static geometry (.CGF). If you specify more than one option, the geometry is selected at random.
Model	Geometry for the boid; this can be a character (.CHR) or static geometry (.CGF).
Mass	Mass of each individual boid.
Invulnerable	Specifies whether the boid can be killed or not.
gravity_at_death	Gravity acceleration that affects the body of the killed boid.
Count	Specifies how many individual objects are spawned.
Behavior	Movement behavior for the boid entity: <ul style="list-style-type: none"> • 0 = Generic ground bugs, such as beetles • 1 = Flying insects, such as dragonflies • 2 = Leaping insects, such as grasshoppers

Flocking Properties

Properties	Description
AttractDistMax	Maximum distance at which one boid can see another boid. Boids that are too far away are not interacted with.

Properties	Description
AttractDistMin	Minimum distance that boids are comfortable with to stay close to each other before the separation force starts to affect them.
EnableFlocking	When enabled, the rules of the emergent flocking behavior is calculated on the whole flock of boids.
FactorAlign	Steer towards the average heading of local flock-mates.
FactorCohesion	Steer to move toward the average position of local flock-mates.
FactorSeparation	Steer to avoid crowding local flock-mates, only when closer then AttractDistMin.
FieldOfViewAngle	Field of vision of the boid to consider other boids as flock-mates.

Note

The following Ground properties apply only when boids are walking on the ground. Boids are able to land only in game mode and not while editing.

Ground Properties

Properties	Description
WalkToIdleDuration	Time it takes for boids to transition from walking to idle state.
WalkSpeed	Walk speed when boids land.
OnGroundWalkDurationMin	Minimum time that boids can spend in walk state.
OnGroundWalkDurationMax	Maximum time that boids can spend in walk state.
OnGroundIdleDurationMin	Minimum time that boids can spend in idle state.
OnGroundIdleDurationMax	Maximum time that boids can spend in idle state.
HeightOffset	Vertical offset of boids from the ground.
FactorSeparation	Tries to ensure that boids avoid one another.
FactorOrigin	Controls how much boids are attracted to their point of origin.
FactorCohesion	Tries to ensure that boids group together.
FactorAlign	Tries to ensure that all boids move in roughly the same direction.

Movement Properties

Properties	Description
FactorAvoidLand	Force coefficient to divert boid from the land or water.
FactorHeight	Controls the force that is applied to keep boids at the original height for the flock.
FactorOrigin	Controls the force that attract boids to the origin point of the flock.
FactorTakeOff	Vertical movement speed scale during take-off.

Properties	Description
FlightTime	Approximate flight time before attempting to land.
HeightMax	Maximal height boids can fly to (height above land).
HeightMin	Minimal height boid can fly at (height above land).
LandDecelerationHeight	Height at which boids start to decelerate when landing.
MaxAnimSpeed	If the boid had animations, then use this variable to control the speed of the animation.
SpeedMax	Maximum speed for boid movement.
SpeedMin	Minimum speed for boid movement.

Options Properties

Properties	Description
Activate	When checked, active boids are visible and move from the start of the level; alternatively, boids can be activated at a later stage with the activate event.
AnimationDist	Maximum distance from camera at which animations update.
FollowPlayer	When checked, boids wrap around only current player position, and the flock origin point becomes the player position. If the boid flies too far away from the player, it reappears on the opposite side.
NoLanding	Turns landing for birds flocks on and off.
ObstacleAvoidance	Boids sense the physical environment and can be diverted from the physical obstacles. This option adds heavier physical checks on the boids and should be used carefully (only when really needed).
Radius	Maximum radius that the boid can move from the flock origin point.
SpawnFromPoint	If true, all the boids spawn at the boid entity position.
StartOnGround	If true, boids spawn on the ground; otherwise, they spawn in the air.
VisibilityDist	Maximum distance from which the whole flock can be visible. If player camera is further away from the flock origin point than VisibilityDist, boids are not simulated and rendered.

ParticleEffect Properties

Properties	Description
EffectScale	Scale of the particle effect to be played.
waterJumpSplash	Particle effect to be played when the boid splashes into the water.

Camera Entities

There are two camera entities.

Camera

Parameters

Parameter	Description
FOV	The field of view of the camera.
NearZ	The cut off point closest to the camera.
FarZ	The max cut off point of the camera.
Shake Parameters	
Amplitude A	Vec3, the strength of the effect on each axis.
Amplitude A Multiplier	Multiplier for the Amplitude.
Frequency A	Vec3, how off the effect will play in each axis.
Frequency A Multiplier	Multiplier for the Frequency.
Noise A Amplitude Multiplier	Add some noise to the amplitude value.
Noise A Frequency Multiplier	Add some noise to the frequency value.
Time Offset A	A some time offset.
Amplitude B	Vec3, the strength of the effect on each axis.
Amplitude B Multiplier	Multiplier for the Amplitude.
Frequency B	Vec3, how off the effect will play in each axis.
Frequency B Multiplier	Multiplier for the Frequency.
Noise B Amplitude Multiplier	Add some noise to the amplitude value.
Noise B Frequency Multiplier	Add some noise to the frequency value.
Time Offset B	A some time offset.
Random Seed	Apply some random variation to the noise.

CameraSource Entity

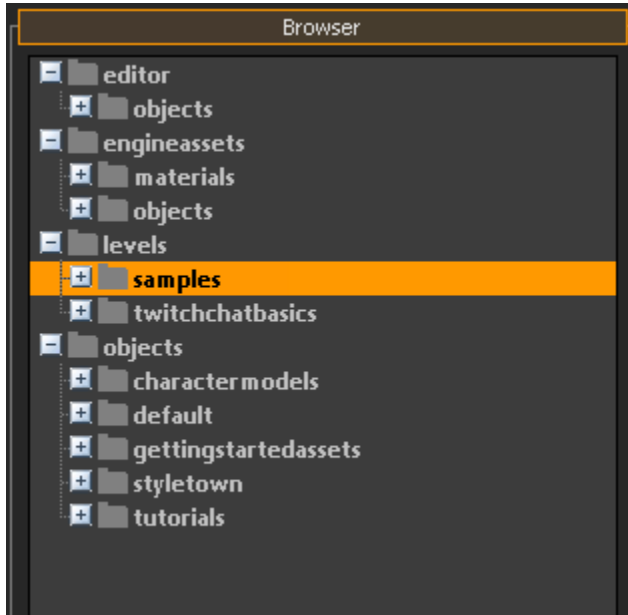
A Camera Source entity is the reference position for a scripted camera view to look from. The point at which the camera points is defined in the **Entity Links** panel. Click **Pick Target**, then on the object you wish to target to create a link.

Geom Entities

A Geom Entity is a very simple entity that takes its physical parameters from its assigned geometry. They are interactive entities with physical values, so they behave like real life objects. It is similar to a

Basic Entity, but simpler, more efficient, and has fewer configurable parameters. Geom Entities that have physical properties set in the asset will get pushed away or break up in explosions, for example.

Navigate through the object library browser and drag the desired object to your level.



Light Entities

Light Entity

Entity Properties

Property	Description
Active	Turns the light on/off.
AttenuationBulbSize	See Attenuation and Falloff for more information. When using AmbientLights, setting this value to '0' reverts to the older, non-physical attenuation model.
Color	
Diffuse	Specify the RGB diffuse color of the light.
DiffuseMultiplier	Control the strength of the diffuse color.
SpecularMultiplier	Control the strength of the specular brightness.
Options	
AffectThisAreaOnly	Set this parameter to false to make light cast in multiple visareas.
AffectVolumetricFogOnly	Enables the light to only affect volumetric fog and not meshes.
AmbientLight	Makes the light behave like an ambient light source, with no point of origin.

Property	Description
FakeLight	Disables light projection, useful for lights which you only want to have Flare effects from.
FogRadialLobe	Adjusts the blend ratio of the main radial lobe (parallel to the eye ray) and side radial lobe (perpendicular to the eye ray). The direction of the main radial lobe depends on the Anisotropic parameter value used in the Time of Day Editor.
ForceDisableCheapLight	Forces the engine to de-classify the light as a "CheapLight", which is a memory optimization done on export for Pure Game mode. Lights are automatically de-classified as needed, based on whether they're used in a FG, trackview, etc, so this option should never need to be used, but provided as a fail-safe.
IgnoresVisAreas	Controls whether the light should respond to visareas.
VolumetricFog	Enables the light to affect volumetric fog.
Projector	
ProjectorFov	Specifies the Angle on which the light texture is projected.
ProjectorNearPlane	Set the near plane for the projector, any surfaces closer to the light source than this value will not be projected on.
Texture	Here a texture can be specified that will be projected in the direction of the Y axis of the light entity. A light projector texture must use the LightProjector CryTif preset, be 512*512px resolution, and contain no alpha channel.
Shadows	
CastShadows	Makes the light cast a shadow based on the minimum selected config spec. "High" won't work on Low/Medium, for example. To ensure shadows are always cast, set the this to 'Low Spec'. This setting is often confused as a 'Quality' setting for the shadows, it is not a quality setting. It's a method to control what system spec the shadows should be cast on. With tiled shading, the amount of shadow-casting lights on screen is limited by default to '12'. This is because each 4 lights requires an additional 8MB of video memory for shadow texture mapping. The limit can be controlled with the r_ShadowCastingLightsMaxCount CVar.
ShadowBias	Moves the shadow cascade toward or away from the shadow-casting object.
ShadowMinResPercent	Specify, per-light, the percentage of the shadow pool the light should use for its shadows. Unless otherwise needed, "default" should be used for best performance vs quality.
ShadowSlopeBias	Allows you to adjust the gradient (slope-based) bias used to compute the shadow bias.

Property	Description
ShadowUpdateMinRadius	Define the minimum radius from the light source to the player camera that the ShadowUpdateRatio setting will be ignored. i.e; If set to 10 and the camera is less than 10m from the light source, the shadow will update normally. If further than 10m, the shadow will update as per ShadowUpdateRatio setting. This will not work in Very High spec as Shadow Caching is disabled.
ShadowUpdateRatio	Define the update ratio for shadow maps cast from this light. The lower the value (example 0.01), the less frequent the updates will be and the more "stuttering" the shadow will appear. This setting is enabled or disabled, depending on the ShadowUpdateMinRadius value and how far the player camera is from the light source. This will not work in Very High spec as Shadow Caching is disabled.
Shape	
PlanarLight	Used to turn the selected light entity into an Area Light. Was previously called "AreaLight". To use Area/Planar Lights, ensure r_DeferredShadingAreaLights is set to '1'.
Style	
AnimationPhase	This will start the light animation, specified with the light style property, at a different point along the sequence. This is typically used when you have multiply lights using the same animation in the same scene, using this property will make the animations play asynchronously.
AnimationSpeed	Specifies the speed at which the light animation should play.
AttachToSun	When enabled, sets the Sun to use the Flare properties for this light.
Flare	Specify the path to the Flare Library item.
FlareEnable	Used by the Flare Editor system.
FlareFOV	Control the FOV for the flare. This control needs to be enabled in the properties for the flare itself.
LightAnimation	Trackview sequence used to animate the light.
LightStyle	Specifies the a preset animation for the light to play. Styles are defined through Light.cfx shader. Valid values are 0-48. 40-48 are Testing/Debug styles.

Environment Probe Entity

With environment probes, also called light probes, you have the ability to place cubemaps throughout a level just as you would a light. It is very useful especially with reflective materials because it will automatically assign the cubemap to anything within its radius.

Properties

Property	Description
Active	Enables and disables the probe.

Property	Description
BoxSizeX, BoxSizeY, BoxSizeZ	Specifies the XYZ dimensions of the probe's area of effect. Probes are projected as cubes in the level. For a global probe, set values large enough to span the entire level.
Diffuse	Sets the diffuse color of the light. Set to 255,255,255.
DiffuseMultiplier	Makes the light brighter. Set to 1.
SpecularMultiplier	Multiplies the specular color brightness. Set to 1.
AffectsThisAreaOnly	Set parameter to False to make lights cover other VisAreas.
AttenuationFalloffMax	Controls the falloff amount (0–1) to create smoother transitions or hard edges. A value of 0.8 means that falloff begins at 80% at the boundaries of the box. Set value to 0 for a global probe (no falloff).
IgnoresVisAreas	Controls whether the light should respond to VisAreas. Set value to True for a global probe.
SortPriority	Gives control over which probe has more visual interest and therefore a higher priority. Set the value to 0 for a global probe, then increase the value for local probes, where higher values indicate more localized probes.
deferred_cubemap	Specifies the file location of the cubemap texture.
BoxHeight	Adjusts the height of cubemap box.
BoxLength	Adjusts the length of cubemap box.
BoxProject	When enabled, Lumberyard factors in the size of the cubemap box.
BoxWidth	Adjusts the width of cubemap box.

Parameters

Parameter	Description
cubemap_resolution	The size of the cubemap.
preview_cubemap	Set to see the cubemap in your level.
Outdoor Only	When set, object will not be rendered when inside a VisArea.
Cast Shadow MinSpec	When set, object casts a shadow on the selected quality setting and above.
LodRatio	Defines how far from the current camera position that different Level Of Detail (LOD) models for the object are used.
ViewDistanceMultiplier	Defines how far from the current camera position that the object is rendered.
HiddenInGame	When set, object is not shown in game mode.
Receive Wind	When set, object is influenced by wind parameters in the level.

Parameter	Description
RenderNearest	Used to eliminate Z-buffer artifacts when rendering in first-person view.
NoStaticDecals	If set to true, decals are not rendered for the selected object.
Created Through Pool	Used primarily for AI entities for memory optimization.

Lightning Arc Entity

You can use the Lightning Arc entity to create realistic electric arcing and sparking effects in your Track View cinematics and levels.

Material Setup

It is recommended to use a diffuse texture, transparency = 99, additive mode, with a slight glow, and using the Illum shader. The arc warps around the U coordinate and uses the V coordinate as a multi-frame animation.

Lightning Arc Parameters

Property	Description
Active	If set to true, it automatically starts sparking after jumping into the game.
ArcPreset	A valid preset must be given. This defines how the lightning arc looks.
Delay	Delay in seconds between sparks.
Delay Variation	Time randomization in seconds.

To set up the ArcPreset visual effect, open the `Libs\LightningArc\LightningArcEffects.xml` file and make desired changes. When finished, reload the `g_reloadGameFx` console variable.

ArcPreset Parameters

Property	Description
lightningDeviation	The smooth snaky effect given to the lightning in meters.
lightningFuzzyness	The noisy effect given to the lightning in meters.
lightningVelocity	After a spark is triggered, it starts to shift from its original position upwards.
branchMaxLevel	Should be kept at either 0 or 1, but either value can be used. However, it also allows child branches to strike out of the main beam and child sparks to branch out of other child beams if this value is 2 or higher.
branchProbability	Probability that a child sparks from another beam segment. If set to 0, no branch is generated, 0.5 is a 50% probability of sparking a branch, 2.0 is a probability of sparking 2 per beam, and so on.
maxNumStrikes	Hard limit on the number of beam segments that can be generated regardless of previous parameters.

Property	Description
strikeTimeMin	Minimum time a spark is kept alive.
strikeTimeMax	Maximum time a spark is kept alive.
strikeFadeOut	When the spark dies, it takes this time to fade out into oblivion. It decreases beamSize to 0 instead of actually fading via transparency.
strikeNumSegments	Number of snaky segments generated.
strikeNumPoints	The number of actual segments generated is defined by $\text{strikeNumSegments} * \text{strikeNumPoint}$. When the code generates the geometry, it creates a camera/beam-aligned quad with exactly 2 triangles. This means that the number of triangles per strike is going to be $\text{strikeNumSegments} * \text{strikeNumPoint} * 2$. Because <code>maxNumStrikes</code> is the hard limit of potential number of sparks active at any time, the potential number poly count of a given lightning effect is going to be $\text{strikeNumSegments} * \text{strikeNumPoint} * 2 * \text{maxNumStrike}$. However, remember that every time the <code>LightningArc</code> entity strikes, a new lightning effect is going to be triggered and therefore the total poly count of a given effect can go higher. The game has an internal hard limit for the total amount of lightning effects, lightning strikes, and poly count that cannot be surpassed; otherwise, geometry starts to disappear.
beamSize	Width of the beam being generated. Child beams have half the width.
beamTexTiling	texture tiling depends on the world size of the actual beam being mapped. A value of 2.0 means the texture wraps around twice every meter. A value of 0.25 means the texture warps around every 4 meters. Only the U coordinate of the texture map is affected by this parameter.
beamTexShift	The U coordinate moves in a given direction at this value's rate. While <code>beamTexTiling</code> only affects the U coordinate, the V coordinate is automatically calculated to select one of the texture's frames.
beamTexFrames	Number of frames in the animation.
beamTexFPS	Frames per second of the multi-frame animation.

Using Flow Graph

The entity:`LightningArc` node is used for creating special arcing effects.

entity:`LightningArc` node I/O ports

Port	Description
Enable/Disable	Allows to dynamically enable or disable the internal timer.
Strike	Allows to manually trigger a spark even when the entity is disabled. This allows synchronization of the spark effect with other level events.

Port	Description
EntityId	The entity that was last struck.
StrikeTime	The time the last spark takes to fade out.

Parameter Reload

Since the lightning effect is implemented using the Game Effects gem, it is possible to reload all parameters during runtime using the `g_reloadGameFx` console command.

Miscellaneous Entities

Miscellaneous entities are commonly used in level design.

The following area objects and entities can be accessed from the **Misc** button on the Objects tab of the Rollup Bar.

- CharAttachHelper
- Comment
- GravityVolume
- ReferencePicture
- SplineDistributor

CharAttachHelper

The CharAttachHelper object can be used to attach any arbitrary object to any bone of a character. The CharAttachHelper object must be linked to the target character, as well as the object to the CharAttachHelper. Use the **Link Object** button located in the toolbar to link objects.

Comment

The comment object allows the adding of comments anywhere inside a level. Comments can be used as a communication device if multiple people work on the same level.

To show comments in game, go to the Console window and type `cl_comment 1`.

Parameters

Parameter	Description
CharsPerLine	Maximum number of characters per line of text
Diffuse	Set the color of the text
Fixed	When using comments to indicate problems/bugs/issues in the level, this field can be used to mark them as "fixed". The text and icon color changes to green
Hidden	Hides the text.
MaxDist	Maximum distance where the comment is shown. If camera is further than this, the comment is hidden
Size	Size of the text
Text	Text to display

GravityVolume

The GravityVolume entity can be used to create tunnels through which the player is getting pushed by an invisible force. It does so by modifying the global gravity variable so that the player stays afloat while maintaining momentum.

Place a GravityVolume entity in the level and in a similar way to placing out a road or river, draw the gravity volume out. Once you have your shape finished double-click the left mouse to finalize the shape.

Parameters

Parameter	Description
Radius	Defines the radius how wide the tube is.
Gravity	Defines how fast objects are getting pushed through the tube.
Falloff	Sets up how the gravity should be decreased at the edge of the tube.
Damping	Specifies the damping amount.
StepSize	Defines how fine the subdivision of the tube geometry segments should be.
DontDisableInvisible	Active this property so that invisible ones don't get disabled.
Enabled	Turns the gravity effect on/off.

ReferencePicture

The ReferencePicture object is used with the ReferenceImage shader and does not receive light or other shader information from within the level. It keeps the image at its pure source.

Parameters

Parameter	Description
File	The image file used as the reference picture.

SplineDistributor

Parameters

Parameter	Description
Geometry	This option specifies the geometry that needs to be used for the object.
Step Size	Sets the distance between each point along the spline. Smaller values increase the polygon count of the surface but also smooths out corners.
OutdoorOnly	When set, the object will not be rendered when inside a visarea.

Parameter	Description
RainOccluder	Set the brush to occlude rain, this works in conjunction with Rain Entity. If your level does contain rain, you should set this wisely, as there is a limit of 512 objects that can occlude at any given time.
SupportSecondVisArea	Normally, objects are considered to be in only one visarea. This option allows them to be added to multiple visareas if their bounding box overlaps them, at the cost of some performance. Without this option, some large objects may not be displayed when viewed through portals in certain situations.
Hideable	When this option is set, AI will use this object as a hiding spot, using the specified hide point type.
LodRatio	Defines how far from the current camera position, the different Level Of Detail models for the object are used.
ViewDistanceMultiplier	Sets the distance from the current view at which the object renders.
NotTriangulate	Deprecated
NoStaticDecals	When this option is set, decals will not project onto the object.
RecvWind	When this option is set, the object will be affected by the level wind.
Occluder	Used for the construction of a level occlusion mesh.

Particle Entities

Particle effect entities act as a container for particle effects and can be attached to any object using the link feature. Particle entity properties become available after dragging a particle effect into a level or by selecting it.

Entity Properties

Property	Description
Active	Sets the initially active or inactive. Can be toggled in the editor for testing.
AttachForm	If AttachType is not empty, this property determines where particles emit from the attached geometry. Set to Vertices, Edges, Surface, or Volume.
AttachType	If this entity is attached to a parent entity, this field can be used to cause particles to emit from the entity's geometry. Set to BoundingBox, Physics, or Render to emit from the applicable geometry.
CountPerUnit	If AttachType is not empty, this multiplies the particle count by the "extent" of the attached geometry. Depending on AttachForm, the extent is either total vertex count, edge length, surface area, or volume.
CountScale	Multiplies the particle counts of the entire emitter.

Property	Description
ParticleEffect	Use to generate the following effects:
Prime	If true, and the assigned ParticleEffect is immortal, causes the emitter to start "primed" to its equilibrium state, rather than starting up from scratch. Very useful for placed effects such as fires or waterfalls, which are supposed to be already running when the level starts. Applies only to immortal, not mortal effects.
PulsePeriod	If not 0, restarts the emitter repeatedly at this time interval. Should be used to create emitters that pulse on and off at somewhat large intervals, a second or so. Do not set a low value such as 0.1 to try to make an instant effect into a continuous one. Make sure the actual library effect is set Continuous and has an appropriate Count.
RegisterByBBox	Uses the emitter's (automatically computed) bounding box to determine which VisAreas it is visible in. If this is disabled (the default), the emitter's origin alone determines VisArea membership, as the bounding box is hard to exactly control by the designer.
Scale	Multiplies the overall size and velocity of the entire emitter.
SpeedScale	Multiplies the particle emission speed of the entire emitter.
Strength	Used by effect parameters to modify their value. If a parameter has an Emitter Strength curve, and the emitter entity's Strength property is not negative, then Strength will be used as input to this curve.
TimeScale	Multiplies the elapsed time used to simulate the emitter. Less than 1 achieves a show-motion effect.
EnableAudio	Toggles sound emission on any sub-effects with an Audio parameter set.

Physics Entities

Physics entities are used to simulate physical events such as explosions, gravity fields, or wind, or to physicalize objects such as cloth, breakable entities, or ropes. Physical entities that are related to a body instead of an event are connected to an object.

The following entities can be accessed by clicking **Entity**, then expanding **Physics** on the Objects tab of Rollup Bar **Entity**.

- AnimObject
- BasicEntity
- Constraint
- DeadBody
- GravityBox
- GravitySphere
- GravityValve
- LivingEntity
- ParticlePhysics

- RigidBodyEx
- Wind
- WindArea

AnimObject

An AnimObject extends the functionality of a BasicEntity by the ability of playing pre-baked animations and physicalizing parts of the object afterwards.

AnimObject Properties

Property	Description
ActivatePhysicsDist	Used for objects with pre-baked physical animations (requires Articulated to be on and ActivatePhysicsThreshold to be greater than 0). Specifies the distance from the pivot after which parts automatically detach themselves from the animation and become fully physicalized. 0 disables distance-based detachment.
ActivatePhysicsThreshold	Greater than 0 values are used for objects with pre-baked physical animations (requires Articulated to be on). Specifies the amount of force (in fractions of gravity) that needs to be exerted on a part for it to become detached and fully controlled by the physics.
CanTriggerAreas	Triggers when this entity enters/exits. Only applicable to AreaTriggers; ProximityTriggers triggers regardless.
DmgFactorWhenCollidingAI	Multiplier applied when dealing damage to AI.
Faction	Entity faction.
InteractLargeObject	Players can trigger large object interactions (such as grab and kick) with the entity.
MissionCritical	Entity is not be hidden by explosions.
Model	Defines the CGA model to be used.
Pickable	Defines whether or not the object can be picked up.
SmartObjectClass	Specifies the smart object type of the object.
Usable	Defines whether or not the object can be used.
UseMessage	The message displayed when the object is in the crosshairs for use.
Animation	
Animation	Defines the animation to be played.
Loop	Defines whether the animation is looped.
PhysicalizeAfterAnimation	Defines whether the object is physicalized after the animation has reached its end.
playerAnimationState	If set, the animation plays immediately.
Playing	If set, the animation plays immediately.
Speed	Playback speed of the animation sequence.

Property	Description
Health	
Invulnerable	Object does not receive damage, but registers "Hit" output when applicable.
MaxHealth	Health of the entity, how much damage can it take before being considered "Dead" and triggering the output.
OnlyEnemyFire	Takes damage from enemy (Faction-based) fire, only if a faction is set.
MultiplayerOptions	
Networked	Physics is simulated on the server and serialized over the network; otherwise, simulated on the client.
Physics	
Articulated	Physicalizes the character as an articulated physical entity (i.e., with bendable joints).
Density	Can be used instead of Mass (if mass is -1) to set the density of each node.
Mass	The overall mass for the entire model.
Physicalize	Selects whether or not the model can become physicalized.
PushableByPlayers	Allows the object to be pushed by players.
RigidBody	If deselected, the object is static. Pre-baked physics objects must have it selected.

BasicEntity

A BasicEntity provides the simplest way of controlling objects physically. Once a model has been set, several properties can be set, defining its physical behavior. It is possible to specify either density or mass of the object. If one is specified, the other one must be set to a negative value (-1, or -0.01). Mass and density affect the way objects interact with other objects and float in the water (they sink if their density is more than that of the water). A zero-mass rigid body (with both mass and density 0) is a special case which means an "animated" rigid body (moved from outside the physics system).

The difference from a static entity is that the physics is aware that this object is actually dynamic, although it cannot simulate it directly. Note that both values describe the same physical property. When you specify mass, density is computed automatically, and vice versa. The relationship $\text{mass} = \text{density} \times \text{volume}$ is used. These computations imply that the object is solid. If a box is used to model an empty crate, one can assume that its density is a weighted average between wood density and inside air density.

BasicEntity Properties

Property	Description
CanTriggerAreas	Areas trigger when this entity enters/exits them. Only applicable to AreaTriggers; ProximityTriggers trigger regardless.
DmgFactorWhenCollidingAI	Multiplier applied when dealing damage to AI.

Property	Description
Faction	Entity faction.
InteractLargeObject	Players can trigger large object interactions (such as grab and kick) with the entity.
MissionCritical	Entity is not be hidden by explosions. The threshold for hiding/ removal is defined via the CVar <code>g_ec_removeThreshold</code> which is set to 20 by default. If an explosion occurs and more than 20 entities are hit by it, it keeps 20 and hides the rest for better performance. See <code>GameRulesClientServer.cpp</code> for more information.
Model	Defines the model to be used.
Pickable	Players can grab or pick up the object.
SmartObjectClass	Can be used to define AI interaction capabilities on code-side.
Usable	Entity is usable by players.
UseMessage	If useable is true, this message is displayed when players are in range. Can be a localized string such as <code>@use_object</code> .
Health	
Invulnerable	Object does not receive damage, but registers "Hit" output when applicable.
MaxHealth	Health of the entity, how much damage can it take before being considered "Dead" and triggering the output.
OnlyEnemyFire	Takes damage from enemy (faction-based) fire, only if a faction is set.
MultiplayerOptions	
Networked	Physics is simulated on the server and serialized over the network; otherwise, simulates on the client.
Physics	
Density	(= Mass / Volume) Density affects the way objects interact with other objects and float in the water (they sink if their density is more than that of the water). Note that both density and mass can be overridden in the asset file.
Mass	(= Density * Volume) Mass is the weight of the object (the density of the object multiplied by its volume).
Physicalize	If false, the object is not taken into account by physics.
PushableByPlayers	If true, the player pushes the object by walking/running into it.

Property	Description
RigidBody	False means a static entity, true - a simulated rigid body. Note that a rigid body can still behave like a static entity if it has mass 0 (set either explicitly or by unchecking RigidBodyActive). The main difference between these rigid bodies and pure statics is that the physics system knows that they can be moved by some other means (such as the trackview) and expects them to do so. This means that objects that are supposed to be externally animated should be mass-0 rigid bodies in order to interact properly with pure physicalized entities.

Constraint

A constraint entity can create a physical constraint between two objects. The objects are selected automatically during the first update, by sampling the environment in a sphere around the constraint object's world position with a specified radius. The "first" object (the one that will own the constraint information internally) is the lightest among the found objects, and the second is the second lightest (static objects are assumed to have infinite mass, so a static object is always heavier than a rigid body).

Constraints operate in a special "constraint frame." It can be set to be either the frame of the first constraint object (if UseEntityFrame is checked), or the frame of the constraint entity itself. In that frame, the constraint can operate either as a hinge around the x axis, or as a ball-in-a-socket around y and z axes (that is, with the x axis as the socket's normal). If x limits are set to a valid range (max>min) and the yz limits are identical (such as both ends are 0), it is the former and, if the yz limits are set and not x limits, it's the latter. If all limits are identical (remain 0, for instance), the constraint operates in a 3 degrees of freedom mode (does not constrain any rotational axes). If all limits are set, no axes are locked initially, but there are rotational limits for them.

Constraint Properties

Property	Description
damping	Sets the strength of the damping on an object's movement. Most objects can work with 0 damping; if an object has trouble coming to rest, try values like 0.2-0.3. Values of 0.5 and higher appear visually as overdamping. Note that when several objects are in contact, the highest damping is used for the entire group.
max_bend_torque	The maximum bending torque (Currently it's only checked against for hinge constraints that have reached one of the x limits).
max_pull_force	Specifies the maximum stretching force the constraint can withstand.
NoSelfCollisions	Disables collision checks between the constrained objects (To be used if the constraint is enough to prevent inter-penetrations).
radius	Defines spherical area to search for attachable objects.
UseEntityFrame	Defines whether to use the first found object or the constraint itself as a constraint frame.
Limits	
x_max	If set greater than x_min, the constraint only rotates the object along its x-axis within the defined angle.

Property	Description
x_min	See x_max.
yz_max	If set greater than yz_min, the constraint only rotates the object along its yz-axis within the defined angle.
yz_min	See yz_max.

DeadBody

A DeadBody entity can ragdollize characters assigned to it. As soon as a character is intended not to act any more, but to only react passively on external impacts, as if it were dead, this physical entity provides the necessary model.

A typical usage is to create the entity as non-resting, simulate it in the editor, and then save the settled physics state. Note that the entity does not react to collisions with the player, bullets, or explosions.

DeadBody Properties

Property	Description
CollidesWithPlayers	Defines whether the ragdoll of the entity may collide with the player (does not override the non-interactive ragdoll legal restriction)
ExtraStiff	Uses the main solver to apply stiffness instead of joint springs. It can handle a lot higher stiffness values, but the downside is that the same stiffness is applied to all joint axes, including locked and limited ones.
lying_damping	(0..1..10) Defines damping in the "lying" mode (which is when the ragdoll has enough contacts with the ground). Note that this is an overall damping, and there also exist per-joint dampings, set based on the asset.
mass	The mass of the object.
MaxTimeStep	As with other entities, decreasing it makes the simulation more stable, but makes this entity and all all entities it contacts with more expensive to simulate. Can be especially useful when higher stiffness is needed.
Model	Character model to be physicalized.
NoFriendlyFire	If set, the entity does not react on bullet impacts from friendly units.
PoseAnim	Allows to use the first frame of the specified animation as an initial pose
PushableByPlayers	If set, the entity does not react on bullet impacts from friendly units.
PushableByPlayers	See BasicEntity (does not override the non-interactive ragdoll legal restriction)
Resting	If set, object do not spawn in a physically 'awake' state. Instead it waits until physically interacted with first.
SmartObjectClass	Specifies the smart object type of the object.

Property	Description
Stiffness	Stiffness with which the ragdoll tries to maintain the original pose (set either in the model or from PoseAnim). For SDK character values around 2000 are practical. Higher values can lead to stability issues, which can be overcome by either decreasing MaxTimeStep (which makes it more expensive to simulate), or using ExtraStiff mode.
Bouyancy	
water_damping	<p>A cheaper alternative/addition to water resistance (applies uniform damping when in water).</p> <p>Sets the strength of the damping on an object's movement as soon as it is situated underwater. Most objects can work with 0 damping; if an object has trouble coming to rest, try values like 0.2-0.3.</p> <p>Values of 0.5 and higher appear visually as overdamping. Note that when several objects are in contact, the highest damping is used for the entire group.</p>
water_density	<p>Can be used to override the default water density (1000). Lower values assume that the body is floating in the water that's less dense than it actually is, and thus it sinks easier.</p> <p>(100..1000) This parameter could be used to specify that the object's physical geometry can leak. For instance, ground vehicles usually have quite large geometry volumes, but they are not waterproof, thus Archimedean force acting on them is less than submerged_volume 1000 (with 1000 being the actual water density).</p> <p>Decreasing per-object effective water density allows such objects to sink while still having large-volume physical geometry.</p> <p>Important note: If you are changing the default value (1000), it is highly recommended that you also change water_resistance in the same way (a rule of thumb might be to always keep them equal).</p>
water_resistance	<p>Can be used to override the default water resistance (1000). Sets how strongly the water affects the body (this applies to both water flow and neutral state).</p> <p>(0..2000) Water resistance coefficient. If non-0, precise water resistance is calculated. Otherwise only water_damping (proportional to the submerged volume) is used to uniformly damp the movement. The former is somewhat slower, but not prohibitively, so it is advised to always set the water resistance.</p> <p>Although water resistance is not too visible on a general object, setting it to a suitable value prevents very light objects from jumping in the water, and water flow affects things more realistically.</p> <p>Note that water damping is used regardless of whether water resistance is 0, so it is better to set damping to 0 when resistance is turned on.</p>

GravitySphere

A GravitySphere is a spherical area, which replaces the gravitational parameters of the environment. Objects reaching this area moved along the entities' Gravity vector and their own physical impact can be damped by a certain factor.

GravitySphere Properties

Property	Description
Active	Defines whether the entity affects its environment.
Damping	Damps physical impact of entities inside the sphere.
Radius	Size of the sphere.
Gravity	x,y, z vector of the gravity applied to objects within the sphere.

GravityValve

A GravityValve entity performs an additional gravity into an upwards showing direction, relative to the entity.

GravityValve Properties

Property	Description
Active	Defines whether the entity affects its environment.
Radius	Size of the affected area.
Strength	Gravitational force.

Wind

A wind entity is used to simulate wind in a local position. This should not be used to create the global wind in your level.

Wind Properties

Property	Description
FadeTime	The time the wind entity uses to fade between disabled and enabled states.
vVelocity	x,y,z vector sets the direction and strength of the wind.

WindArea

A WindArea simulates air moving with an arbitrary speed in a specific direction. It affects the flow direction of all objects and aero-form substances within the defined area, as well as vegetation bending depending on density and resistance values. If no direction is set, the wind-source moves omnidirectionally from the center of the WindArea.

WindArea Properties

Property	Description
Active	Defines whether wind is blowing or not.
AirDensity	Causes physicalized objects moving through the air to slow down, if > 0.
AirResistance	Causes very light physicalized objects to experience a buoyancy force, if > 0.
Ellipsoidal	Forces an ellipsoidal falloff.
FalloffInner	Distance after which the distance-based falloff begins.
Speed	Wind-speed in units per second.
Dir XYZ	x,y,z vector of normalized wind direction.
Size XYZ	x,y,z vector of affected area.

Useful Console Variables

The following console variables are useful for debugging physics entity issues:

p_draw_helpers

```
Same as p_draw_helpers_num, but encoded in letters
Usage [Entity_Types]_[Helper_Types] - [t|s|r|R|l|i|g|a|y|e]_[g|c|b|l|t|#]
Entity Types:
t - show terrain
s - show static entities
r - show sleeping rigid bodies
R - show active rigid bodies
l - show living entities
i - show independent entities
g - show triggers
a - show areas
y - show rays in RayWorldIntersection
e - show explosion occlusion maps
Helper Types
g - show geometry
c - show contact points
b - show bounding boxes
l - show tetrahedra lattices for breakable objects
j - show structural joints (forces translucency on the main geometry)
t(#) - show bounding volume trees up to the level #
f(#) - only show geometries with this bit flag set (multiple f's stack)
Example: p_draw_helpers larRis_g - show geometry for static, sleeping,
active, independent entities and areas
```

p_debug_joints

If set, breakable objects log tensions at the weakest spots.

Rain Entity

You can use the Rain entity to add realistic rain effects to your level.

Entity Properties

Property	Description
Amount	Sets the amount of rain and rain effects in a level. AttenAmount is multiplied by the amount, and is used to set the current amount.
DiffuseDarkening	Modifies the albedo of the rain effect, such as for horizontal water puddles.
DisableOcclusion	Blocks rain for selected objects in your level. Don't select for objects that are protected (under cover) from rain.
Enabled	Enables or disables the rain effects.
FakeGlossiness	Sets the amount of glossiness for wet surfaces.
FakeReflectionsAmount	Sets the amount of reflection from wet surfaces.
IgnoreVisAreas	Renders rain even when player is inside a VisArea.
PuddlesAmount	Sets the depth and brightness of water puddles generated by the rain.
PuddlesMaskAmount	Sets the strength of the water puddle mask to balance different puddle results.
PuddlesRipplesAmount	Sets the strength and frequency of ripples in water puddles.
Radius	Sets the coverage area of rain around the entity.
RainDropsAmount	Sets the amount of rain drops that can be seen in the air.
RainDropsLighting	Sets the brightness or backlighting of the rain drops.
RainDropsSpeed	Sets the speed at which rain drops travel.
SplashesAmount	Modifies the strength of the splash effect.

Render Entities

You can use the following Render entities in your level.

FogVolume Entity

Entity Properties

Property	Description
Active	If true, fog volume will be enabled.
Color	Specifies the RGB diffuse color of the fog volume
DensityOffset	Used in conjunction with the GlobalDensity parameter to offset the density.
FallOffDirLati	Controls the latitude falloff direction of the fog. A value of 90° means the falloff direction is upwards.

Property	Description
FallOffDirLong	Controls the longitude falloff direction of the fog, where 0° represents east. Rotation is counterclockwise.
FallOffScale	Scales the density distribution along the falloff direction. Higher values make the fog fall off more rapidly and generate thicker fog layers along the negative falloff direction.
FallOffShift	Controls how much to shift the fog density distribution along the falloff direction in world units (m). Positive values move thicker fog layers along the falloff direction into the fog volume.
GlobalDensity	Controls the density of the fog. The higher the value the more dense the fog.
HDRDynamic	Specifies how much brighter than the default white (RGB 255,255,255) the fog is.
NearCutoff	Stops rendering the object depending on camera distance to object.
SoftEdges	Factor used to soften the edges of the fog volume when viewed from outside. A value of 0.0 produces hard edges. Increasing this value up to 1.0 gradually softens the edges. This property currently has no effect on box type fog volumes as specified in the VolumeType parameter.
UseGlobalFogColor	If selected, ignores the Color parameter and uses the global (Time Of Day) fog color instead.
VolumeType	Produces a box volume for values above 1.0 or a spherical volume for lower values.
Size x, y, z	Specifies the height, width, and depth of the fog volume in meters.

River Entity

You can customize your rivers with a number of different parameters. Many of the settings are the same as those of the [WaterVolume Shader \(p. 1032\)](#).

Note

The Speed parameter listed below specifies the speed at which objects float down the river. The speed of the river itself is specified using the **Flow speed** parameter for the [WaterVolume Shader \(p. 1032\)](#).

Parameters

Parameter	Description
Width	Sets the width of the river. This is set much wider than the actual river (water) width, as the complete river is defined by the river bed and surrounding terrain.
BorderWidth	Used in conjunction with Align Height Map , creates a smooth edge for the river bed geometry if this value is greater than the Width value.

Parameter	Description
StepSize	Sets the distance between each point along the river spline. Smaller values increase the polygon count of the river surface but also smooths out corners.
ViewDistanceMultiplier	Sets the distance from the current view at which the river renders.
TileLength	Length of the river texture. Use in conjunction with StepSize to avoid stretching textures.
Depth	Sets the depth of the river.
Speed	Defines how fast physicalized objects move along the river. Use negative values to move in the opposite direction.
UScale	Sets the texture tiling on the U axis.
VScale	Sets the texture tiling on the V axis.
Caustics	Enables optical caustics effects.
CausticIntensity	Scales the intensity of the caustics for the water surface normals.
CausticTiling	Scales the caustic tiling applied to the water surface normals. It allows the scaling of caustics independently from the surface material.
CausticHeight	Sets the height above the water surface at which caustics become visible. Use this to make caustics appear on overhanging landforms or vegetation and other nearby objects.

Road Entity

You can modify any of several road parameters to customize your road.

Parameters

Parameter	Description
Width	Width of the road.
BorderWidth	Used in conjunction with Align Height Map , creates a smooth edge for the road if this value is greater than the Width value.
StepSize	Sets the distance between each point along the road spline. Smaller values increase the polygon count for the road surface but also smooths out corners.
ViewDistanceMultiplier	Specifies the distance at which the road renders.
TileLength	Length of the road texture. Used in conjunction with StepSize to avoid stretching textures.
SortPriority	Determines the rendering order. Higher values are rendered above lower values.
IgnoreTerrainHoles	If enabled, renders the road texture over holes created with the terrain Holes brush.

Rope Entity

The Rope entity is used to create realistic ropes in your level.

Parameters

Parameter	Description
Radius	The radius, or thickness, of the rope.
Smooth	Defines if the rope will be smoothed out or not.
Num Segments	The number of segments of geometry used in the rope along its length.
Num Sides	The number of sides around the circumference of the rope. 4 sides would make it a diamond shaped tube, 8 sides would make it much smoother, etc.
Texture U Tiling	Texture tiling in the U direction.
Texture V Tiling	Texture tiling in the V direction.
CastShadows	Enable shadow casting from the rope.
Bind Ends Radius	Specifies whether the ends will be automatically attached.
Bind Radius	The environment around the ends of the rope will be tested using a box of this radius to find places for the rope to attached to. Note that if bind radius is greater than 0.05 the ends are snapped to the colliding surface.
Physics Params	
Subdivide	Maximum number of subdivided vertices per segment.
Max Subdiv Verts	Maximum number of subdivided vertices per segment.
Physical Segments	Number of rope segments in physics (can be different from the number of segments used for rendering). For colliding ropes, make sure that there are enough physical segments so that segment length is at least two times smaller than the dimensions of the objects the rope collides with.
Tension	Specifies tension in the original state. A positive value will cause the rope ends to pull together, negative will add slack to the rope (-0.02 is a good starting point for experiments).
Friction	The friction effective in a non-strained mode. In a strained mode with dynamic tessellation, this that prevents the rope from slipping until it tilts too much.
Wind	
Wind Variation	How much the wind varies. Basically a randomization multiplier on top of the base Wind XYZ values.
Air Resistance	Must be set in order for global environment wind to take effect. Not necessary for simulated Wind XYZ values.

Parameter	Description
Water Resistance	How the rope interacts with water effectively damping when under water.
Check Collisions	Ignore collisions from other objects.
Ignore Attachment Collisions	Ignore collisions with the object it is attached to.
Ignore Player Collisions	Ignore collisions with players.
Non-shootable	Rope cannot be broken by shooting. Rope will still react to physical impulses from bullets.
Disabled	Simulation is completely disabled.
StaticAttachStart	Attach start point to the level.
StaticAttachEnd	Attach end point to the level.
Advanced	
Mass	This affects how strongly the rope will react to bullet hits. When interacting with solid physicalized objects, it is always treated as weightless.
Friction Pull	The friction effective in a non-strained mode. In a strained mode with dynamic tessellation, this that prevents the rope from slipping until it tilts too much.
Max Force	The rope will detach itself when this strain limit is breached.
Solver Iterations	Ropes with very large segment counts (40+) might need this increased (values up to 10k are still viable).
Max Timestamp	Sets the maximum time step the entity is allowed to make (defaults to 0.01). Smaller time steps increase stability (can be required for long and thin objects, for instance), but are more expensive. Each time the physical world is requested to make a step, the objects that have their maxsteps smaller than the requested one slice the big step into smaller chunks and perform several substeps. If several objects are in contact, the smallest max_time_step is used.
Stiffness	Rope's stiffness against stretching. Might need tweaking for longer ropes. Note the in most cases ropes will use exact length enforcement (meaning 'infinite' stiffness), but internally stiffness will still be used to compute the dynamics.
ContactHardness	Hardness of contacts and length enforcement in subdivision mode, when strained and potentially touching other objects in the middle. Higher values make it potentially less stable.
Damping	Sets the strength of the damping on an object's movement. Most objects can work with 0 damping; if an object has trouble coming to rest, try values like 0.2 - 0.3. Values of 0.5 and higher appear visually as overdamping. Note that when several objects are in contact, the highest damping is used for all associated contacts.

Parameter	Description
Sleep Speed	If the object's kinetic energy falls below some limit over several frames, the object is considered "sleeping". This limit is proportional to the square of the sleep speed value. A sleep speed of 0.01 loosely corresponds to the object's center moving at a velocity of the order of 1 cm/s.
Sound Data	
Name	Name of the sound to be attached.
Segment	Number of rope segments in physics (can be different from the number of segments used for rendering). For colliding ropes, make sure that there are enough physical segments so that segment length is at least two times smaller than the dimensions of the objects the rope collides with.
PosOffset	The position offset indicates how far a sound is moved away from its original attachment point. The number (.0-1) moves the sound along the length of the segment to which the sound is attached.

Snow Entity

You can use the Snow entity to add realistic snow effects to your level.

Entity Properties

Property	Description
Enabled	Select to enable snow.
Radius	Sets the coverage area of snow on the ground. Has no effect on the distance that snow in the air spawns at.
Brightness	The brightness of snowflakes in the air.
GravityScale	Controls how fast snow falls.
SnowFlakeCount	Sets the number of snowflakes in the air.
SnowFlakeSize	Sets the size of snowflakes in the air.
TurbulenceFreq	Frequency of air turbulence on falling snowflakes.
TurbulenceStrength	Strength of air turbulence on falling snowflakes.
WindScale	How strongly wind in a level effects falling snowflakes.
FrostAmount	Amount of frost that appears on a surface.
SnowAmount	Amount of snow that appears on a surface.
SurfaceFreezing	Strength of the visual freezing effect on a surface.

Tornado Entity

You can create realistic-looking tornadoes in your level.

Entity Properties

Property	Description
AttractorImpulse	The gravitational pull of the tornado on nearby objects.
CloudHeight	The height of the cloud above the tornado.
FunnelEffect	Specifies the particular particle effect.
Radius	Radius of the tornado's influence.
SpinImpulse	The rotational speed of the tornado.
UpImpulse	The upward speed of the tornado.
WanderSpeed	The speed that the tornado is moving along the ground.

Trigger Entities

There are two Trigger entities you can use in your level.

AreaTrigger Entity

Entity Properties

Property	Description
trigger-proximity	Turns the entity on or off.
InVehicleOnly	Sets up that the trigger can only be activated when player is inside vehicle.
OnlyLocalPlayer	Sets the trigger to be only triggerable by the local player entity.
OnlyPlayers	Sets the trigger to be only triggerable by players entities.
PlaySequence	Plays the Trackview sequence with the name specified in here.
ScriptCommand	Executes a script command when the trigger has been activated.
TriggerOnce	Disables the trigger after it has been triggered once.
MultipayerOptions	
Networked	If true physics will be simulated on the server and serialized over the network, otherwise they will be simulated on the client.

ProximityTrigger Entity

Entity Properties

Property	Description
ActivateWithUseButton	Specifies if the trigger is activated by pressing use.
DimX	Specifies how big the trigger is (x-axis).

Property	Description
DimY	Specifies how big the trigger is (y-axis).
DimZ	Specifies how big the trigger is (z-axis).
Enabled	Specifies if the trigger can be activated or not.
EnterDelay	Sets up a delay (in seconds) before the enter node of the trigger is activated.
ExitDelay	Sets up a delay (in seconds) before the exit node of the trigger is activated.
InVehicleOnly	Sets up that the trigger can only be activated when player is inside vehicle.
OnlyAI	Sets the trigger to be only triggerable by AI entities.
OnlyMyPlayer	Sets the trigger to be only triggerable by the local player.
OnlyOneEntity	Sets the trigger to be only triggerable by one entity. First one who triggers it has to leave it in order to be triggerable again.
OnlyPlayer	Sets the trigger to be only triggerable by player entities.
OnlySelectedEntity	Sets the trigger to be only triggerable by the entity with the name specified in this field. Wildcard matches can be used such as RigidbodyEx*, will allow all entities with that name, regardless of number suffix, etc.
OnlySpecialAI	Sets the trigger to be only triggerable by the special AI entities.
PlaySequence	Plays the Trackview sequence with the name specified in here.
RemoveOnTrigger	Similar to the deprecated "KillOnTrigger" param, if true, any entities (except player) which trigger this will be removed.
ScriptCommand	Executes a script command when the trigger has been activated
TriggerOnce	Disables the trigger after it has been triggered once.
MultiplayerOptions	
Networked	If true physics will be simulated on the server and serialized over the network, otherwise they will be simulated on the client.

Flow Graph System

Flow Graph is a visual scripting system that allows you to implement complex game logic without having to touch any code. Complex logic can be created with only a few clicks and an extensive library of nodes provides everything needed to fully control entities and AI agents in a level.

Flow Graph can also be used to prototype gameplay, effects, and sound design, with a level containing multiple flow graphs performing different tasks at the same time.

Flow graphs consist of nodes and links. Nodes can represent level entities (entity node) or actions (component node) that may perform a specific action on a target entity. Links are used to connect nodes, and are represented as lines that connect the inputs and outputs between nodes.

Flow Graph logic is stored in XML files and can be exported for use in other levels. As a flow graph is associated with a specific entity, the graph is always exported along with the entity. Layers are supported.

Topics

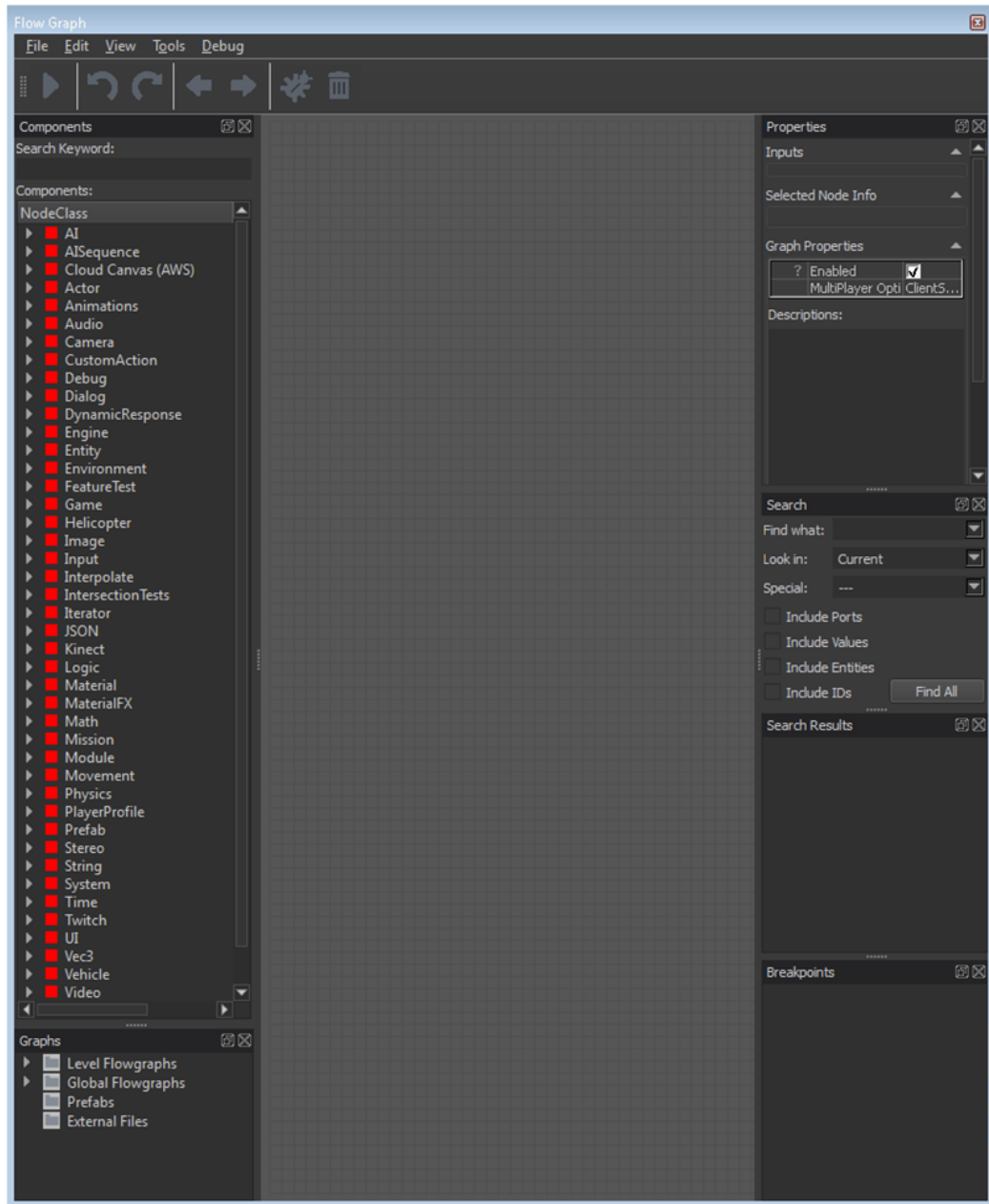
- [Using Flow Graph Editor \(p. 487\)](#)
- [Flow Graph Scripts \(p. 488\)](#)
- [Managing Flow Graphs \(p. 489\)](#)
- [Using Flow Graph Nodes \(p. 491\)](#)
- [Creating Flow Graph Nodes \(p. 494\)](#)
- [Flow Graph Node Reference \(p. 500\)](#)
- [Using Flow Graph Links \(p. 772\)](#)
- [Using Flow Graph Tokens \(p. 773\)](#)
- [Managing Flow Graph Modules \(p. 773\)](#)
- [Debugging Flow Graph \(p. 775\)](#)
- [Placing Cached Shadows \(p. 776\)](#)

Using Flow Graph Editor

Flow Graph editor uses drag-and-drop modules and connection links to various inputs and outputs to perform visual scripting. The following shows the components of the Flow Graph editor:

- **Node graph** - main window grid for displaying flow graph nodes and connections
- **Components** - browser tree pane for nodes

- **Graphs** - browser tree pane for graphs and entities
- **Properties** - pane for showing node input and output properties
- **Search** - pane for searching graphs and nodes
- **SearchResults** - pane for displaying search results
- **Breakpoints** - pane for displaying breakpoints



Flow Graph Scripts

Flow Graph scripts are organized into four different categories, and contained in the **Graphs** folder tree in the Flow Graph Editor.

Level Flowgraphs

This directory contains script files that are specific to the level that is currently open, and is organized as follows:

- **Entities** – Entity files are the flow graphs created and associated with an entity that has been placed in the level.
- **Components** – Component files are the flow graphs created and associated with a component that has been placed in the level.
- **Modules** – Modules that are specific to the level that is currently open.

Global Flowgraphs

- **UI Actions** - Used to encapsulate UI logic for easy debugging and maintenance.

Flow Graph Prefabs

Using Flow Graph, you can communicate directly to and from a prefab instance just like an entity by using prefab events. Simply create an event inside a prefab, give it a name, and then reference the prefab instance as you normally do for an entity.

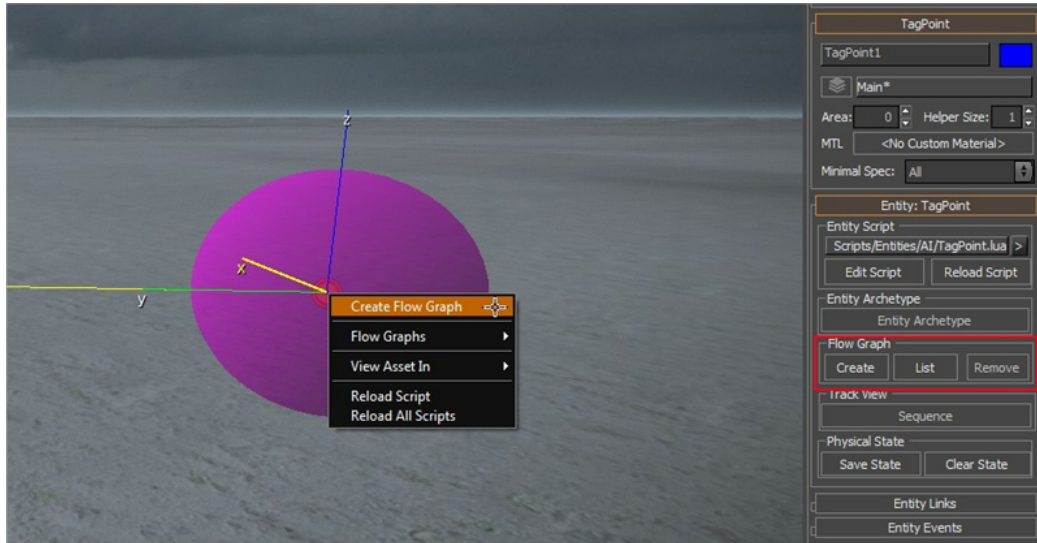
External Files

These are Flow Graph scripts that have been imported.

Managing Flow Graphs

Each flow graph is associated with a specific entity and is stored as a property of the entity. The name of the flow graph is the name of the entity for which it has been created. When the entity name is changed, the name in the flow graph is also automatically changed. When the entity is saved or exported, the flow graph belonging to it is also automatically saved.

There are two types of flow graphs: global flow graphs, which are used in multiple levels, and level flow graphs, which are associated with a single level.



To create a flow graph for an entity

1. In Rollup Bar, on the **Objects** tab, for an entity previously created, under **Flow Graph**, click **Create**.
2. Alternatively, right-click the entity in the viewport, then click **Create Flow Graph**. If this is the first flow graph in a level, you need to select a group to place the flow graph with, or click **New** to create a new group name for the flow graph. The **Flow Graphs** window displays the new flow graph in the tree.

To manage flow graphs

- In Flow Graph Editor, right-click the applicable flow graph in the **Flow Graphs** tree, then do the following as needed:
 - To delete a flow graph, click **Delete Graph**.
- Note**
When an entity is deleted from a level, the associated flow graph is also deleted.
- To enable or disable a flow graph, toggle **Enable** or **Disable**.
 - To enable or disable all flow graphs in a group, right-click the parent folder, then click **Enable All** or **Disable All** as needed. A disabled flow graph is displayed as crossed out, which means that all nodes in the flow graph are ignored when the game is running.
 - To move a flow graph to another group, right-click the parent folder, click **RenameFolder/MoveGraphs**, then select a group from the list or click **New** to move it to its own new group and name it.

When a level is exported with some flow graphs disabled, their disabled state is also exported to the game.

Saving Flow Graphs

The method of saving flow graphs differ depending on whether it is a global flow graph or a level flow graph.

Global flow graphs, which are listed under **Graphs, Global**, are saved by selecting the flow graph and then clicking **File, Save**.

Level flow graphs, which are listed under **Graphs, Level**, are saved automatically when either the level they are in is saved or the layer that they are on is saved. A layer gets saved whenever the corresponding level is saved.

Grouping Flow Graphs

To create a flow graph group

1. In the graph pane, select two or more flow graph nodes by CTRL+ click on each one.
2. Right-click the graph pane, and click **Group**. A box appears around the nodes.
3. Type a name for the group.

You can rename, move, add to, and remove a group.

To manage flow graph groups

1. To rename a group, double-click the group's name and type a new name.
2. To collapse a group to save space, click the down-arrow icon for the group. To expand the group back, click on the icon again.
3. To move a node within a group, click on the node's title bar and drag it to the desired location.
4. To move a group, click on an empty space in the group and drag it to the desired location.
5. To add a node to a group, click to select the group, Ctrl+click on the applicable node, right-click the graph pane, then click **Add group**. The group's box now encloses the new node.
6. To remove a node from a group, click to select it, right-click on an empty space in the group, then click **Ungroup**. The nodes selected are removed from the group. If the group as a whole is selected, the group is removed entirely.
7. To remove a group entirely, right-click the group's name and click in the **Ungroup**.

Importing and Exporting Flow Graphs

Flow graphs are saved as XML files and can be exported and imported.

To export a flow graph

- Select the nodes for export by Ctrl+Click each node, then right-click the final node, click **Selection, Export Selected Nodes**, then enter a file name for it.

You can import a previously exported flow graph's nodes into another flow graph as follows:

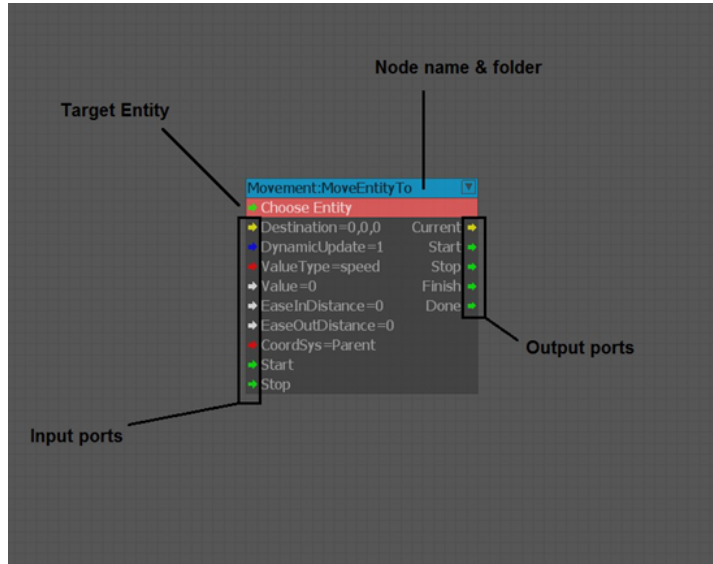
To import a flow graph

- Open the target flow graph you want to import to add the exported flow graph nodes to, right-click anywhere in the graph pane, click **Import**, then enter the name of file you want to import..

The imported flow graph is positioned relative to the old flow graph.

Using Flow Graph Nodes

Nodes can represent level entities (entity node) or actions (component node) that may perform a specific action on a target entity. A node is represented in Flow Graph as a box with inputs and outputs.



Node Input/Output Ports

A node consists of input ports on the left side for receiving information and output ports on the right side for transmitting information. Output ports are activated depending on the function of the node. Ports can have the following different data types.

Node Port Data Types

Data Type	Color	Description
Any	n/a	Unspecified, any data type can be received
Boolean	Blue	True or false value
EntityID	Green/Red	Value that uniquely identifies any entity in a level
Float	White	Floating-point 32-bit value
Integer	Red	Positive or negative 32-bit number
UInt64	n/a	Positive or negative 64-bit number
String	Turquoise	Array of characters used for storing text
Vec3	Yellow	3D vector consisting of three floating-point values. Used for storing positions, angles, or color values
Void	Green	Used for ports that do not accept any value but are instead triggered to pass the flow of control through a flow graph.

Differing colors for node backgrounds and links indicate the following:

- Nodes with a red background and a yellow title bar are debugging nodes and are not functional in release builds.
- Links that connect debugging nodes are yellow.
- Dotted links indicate they are disabled (by right-clicking them)

Values whose data type don't match the input port data type are automatically converted to match the type of the port connected to, if possible. Any output port can be connected to any input port, no matter what data type. An integer with the value 1 can be fed in a Boolean input port and converted to a `TRUE` value to match the data type of the port. For some component nodes, there is an input port at the top of the entity that is used for setting the target entity of the node.

Note

Mixing node port types or data types can result in unexpected behavior. For example while a **Math:SetColor** node input port is a `Vec3` data type, it treats input from a **Vec3:SetVec3** node differently than from a **Math:SetColor** node, both of which output a `Vec3` data type. While the port types for both nodes are vector, the **Vec3:SetVec3** are a group of three floating-point values whereas the **Math:SetColor** data type are a group of colors that range from 0-255.

Adding Entity Nodes

Entity nodes require that a level entity first be selected. To add an entity node, select an entity and open the graph where you want to add the entity. Next, open the graph context menu by right-clicking the main editing pane.

To add an Entity node

1. In the left-side **Flow Graphs** tree, expand **Entities\fg** and select the applicable entity.
2. Right-click anywhere in the graph pane and click **Add Selected Entity**.
3. Or, right-click anywhere in the graph pane and click **Add Graph Default Entity**, which always adds the entity to the flow graph to which it is attached.

Adding Component Nodes

Component nodes can be added from within the graph and don't require any selected entity. There are three ways to add these nodes, the context menu, the component node list window and the `QuickSearchNode` (Shortcut: Q).

To add a new component node, open the context menu by right-clicking the main editing pane, and then select **Add Node**. A long list of sub-folders are displayed, and a node can be selected from any directory. Select **Entity** to open the folder with the entity-related component nodes. Select **EntityPos** to complete the procedure.

To add a Component node

- Right-click anywhere in the graph pane, click **Add Node**, and select a node from the list.

Managing Nodes

You can easily move, copy, edit, and delete Flow Graph nodes as follows. All links between selected nodes are also moved when the nodes are moved and automatically rearrange themselves.

To move a node

1. Click and drag the node on the graph pane. Multiple nodes can be moved by holding down the `Ctrl` key and clicking the applicable nodes.
2. Or, use the mouse to draw a box around all the applicable nodes that need to be moved.

To copy a node

1. Right-click the node, click **Copy**, then click **Paste** at the desired location in the graph pane. Click **Paste With Links** to also copy all connected links.
2. Or, click the node, press `Ctrl+C`, then press `Ctrl+V` at the desired location.

To edit a node

There are two ways to edit a node's properties.

1. Double-click the applicable node input and change the property.
2. Or, change the property as listed under **Inputs** in the right-side panel of Flow Graph Editor.

To delete a node

There are two ways to delete a node. Once a node has been deleted, all the connected links are also automatically removed.

1. Right-click the node and click **Delete**.
2. Or, click the node and press the keyboard `Delete` key.

Creating Flow Graph Nodes

You can use a `.cpp` file to create new flow graph nodes. For multiple flow graph nodes that will belong to the same group, use a single `.cpp` file. Headers aren't needed except for some specialized nodes.

Use the following code template for your `.cpp` file and save the file to the `dev\Code\CryEngine\CryAction\FlowSystem\Nodes` directory.

In the template you can choose between an `eNCT_Instanced` node and a `eNCT_Singleton`. A singleton node creates one instance with a small memory footprint, although you can still use multiple nodes in your flow graph. Use singleton whenever you are not saving state data such as member variables.

```
#include "StdAfx.h"
#include "FlowBaseNode.h"

class CFlowNode_YourFlowNodeName : public CFlowBaseNode<eNCT_Instanced>
{
public:
    CFlowNode_YourFlowNodeName(SActivationInfo* pActInfo)
    {
    };

    virtual IFlowNodePtr Clone(SActivationInfo *pActInfo)
    {
        return new CFlowNode_YourFlowNodeName(pActInfo);
    };

    virtual void GetMemoryUsage(ICrySizer* s) const
    {
        s->Add(*this);
    }
};
```



```
virtual void GetConfiguration(SFlowNodeConfig& config)
{
    static const SInputPortConfig in_config[] = {
        {0}
    };
    static const SOutputPortConfig out_config[] = {
        {0}
    };
    config.sDescription = _HELP( "your_flow_node_tooltip_description" );
    config.pInputPorts = in_config;
    config.pOutputPorts = out_config;
    config.SetCategory(EFLN_APPROVED);
}

virtual void ProcessEvent(EFlowEvent event, SActivationInfo* pActInfo)
{
    switch (event)
    {
        {
        };
    }
};

REGISTER_FLOW_NODE("your_flow_node_group:your_flow_node_name",
    CFlowNode_your_flow_node_name);
```

For your flow node group, create a corresponding subfolder in the Flow Graph editor node selector where this node will be placed in the hierarchy.

Output Ports

You can add an output port by modifying the `GetConfiguration` function as shown in the following example:

```
class CFlowNode_your_flow_node_name : public CFlowBaseNode<eNCT_Instanced>
{
public:
    // ...

    virtual void GetConfiguration( SFlowNodeConfig& config )
    {
        static const SInputPortConfig in_config[] = {
            {0}
        };
        static const SOutputPortConfig out_config[] = {
            OutputPortConfig<int>("your_output", _HELP("your_help_text")),
            {0}
        };
        config.sDescription = _HELP( "your_flow_node_tooltip_description" );
        config.pInputPorts = in_config;
        config.pOutputPorts = out_config;
        config.nFlags = 0;
    }

    // ...
};
```

`OutputPortConfig` is a helper function that is useful for filling a small structure with appropriate data.

Available data types for this function include SFlowSystemVoid, Int, Float, EntityId, Vec3, String, and Bool. SFlowSystemVoid is a special data type that represents "no value".

OutputPortConfig takes the following parameters:

- Port name that is used internally and for saving the flow graph. Do not change this parameter later as doing so will break script compatibility for all flow graphs that use this node.

Note

Do not use the underscore "_" character as this was used in previous versions to specify a specialized editor for the port.

- Description used to display tooltip help text on mouse hover in the Flow Graph editor.
- Human-readable name used to display the name of the port in the Flow Graph editor. This is used to visually override a port name without breaking script compatibility.

To emit a value from the output port, use the function

CFlowBaseNode::ActivateOutput(pActInfo, nPort, value). This function takes a pActInfo, which is typically passed to ProcessEvent(), the nPort port identifier (count starts at zero from the top of out_config), and a value of the same type as the port.

Input Ports

You can add an input port by modifying the GetConfiguration function as shown in the following example:

```
class CFlowNode_YourFlowNodeName : public CFlowBaseNode<ENCT_Instanced>
{
public:
    // ...

    virtual void GetConfiguration( SFlowNodeConfig& config )
    {
        static const SInputPortConfig in_config[] = {
            InputPortConfig<int>( "YourInput", _HELP( "YourHelpText" ) ),
            {0}
        };
        static const SOutputPortConfig out_config[] = {
            {0}
        };
        config.sDescription = _HELP( "YourFlowNodeTooltipDescription" );
        config.pInputPorts = in_config;
        config.pOutputPorts = out_config;
        config.nFlags = 0;
    }

    // ...
};
```

InputPortConfig is a helper function that is useful for filling a small structure with appropriate data.

Available data types for this function include SFlowSystemVoid, Int, Float, EntityId, Vec3, String, and Bool. SFlowSystemVoid is a special data type that represents "no value".

InputPortConfig takes the following parameters:

- Port name used internally and for saving the flow graph. Do not change this parameter later as doing so will break script compatibility for all flow graphs that use this node.

Note

Do not use the underscore "_" character as this was used in previous versions to specify a specialized editor for the port.

- Default value of the port when a new node is created.
- Description used to display tooltip help text on mouse hover in the Flow Graph editor.
- Human-readable name used to display the name of the port in the Flow Graph editor. Use to visually override a port name without breaking script compatibility.
- Formatted string that specifies how the UI should function when setting the port value. You can choose a specialized widget or modify the allowed value range of the input.

Input Port UI Configuration

You can define the interface for setting the input port value by passing a series of options in the form of a string with key-value pairs in `InputPortConfig`.

Setting the input value range

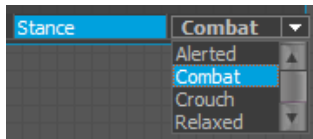
This will limit the widget's arrows and ramp and clamp manually-inserted values as shown in the figure:



```
_UICONFIG("v_min=0, v_max=10")
```

Setting the Dropdown List

There are several types of enums that you can use to display a dropdown list of readable strings. Each string maps to a value that is used by the node and that persists when the flow graph is saved. Enums can be of type `int` or `float` as shown in the following figure and code example.



```
_UICONFIG("enum_int:Relaxed=0,Alert=1,Combat=2,Crouch=3")
```

An enum can also be of type `string` with or without mapping to another value:

```
_UICONFIG("enum_string:a,b,c")  
_UICONFIG("enum_string:DisplayA=a,DisplayB=b,DisplayC=c")
```

Enums can also refer to the global and dynamic UI enums defined in `InitUIEnums`.

Optionally, the enum can depend on another port to affect the available selection:

```
_UICONFIG("enum_global:ENUM_NAME")  
_UICONFIG("enum_global:vehicleLightTypes")  
_UICONFIG("enum_global_def:ENUM_NAME")  
_UICONFIG("enum_global_ref:ENUM_NAME_FORMAT_STRING:REF_PORT")
```

Setting a Specialized Property Editor

You can indicate a dedicated property editor with the `dt` keyword followed by parameters optionally needed by the editor as shown in the following code example:

```
_UICONFIG("dt=editorName")_UICONFIG("dt=entityProperties,  
ref_entity=entityId")_UICONFIG("dt=matparamslot,  
slot_ref=Slot, sub_ref=SubMtlId, param=float")
```

There is a set of available editors that can be referenced in the following table:

Editor Name	Editor Type	
snd	IVariable::DT_SOUND	
sound	IVariable::DT_SOUND	
clr	IVariable::DT_COLOR	
color	IVariable::DT_COLOR	
tex	IVariable::DT_TEXTURE	
texture	IVariable::DT_TEXTURE	
obj	IVariable::DT_OBJECT	
object	IVariable::DT_OBJECT	
file	IVariable::DT_FILE	
text	IVariable::DT_LOCAL_STRING	
equip	IVariable::DT_EQUIP	
reverbpreset	IVariable::DT_REVERBPRESET	
aiaanchor	IVariable::DT_AI_ANCHOR	
aibehavior	IVariable::DT_AI_BEHAVIOR	
aiacharacter	IVariable::DT_AI_CHARACTER	
aipfpropertieslist	IVariable::DT_AI_PFPROPERTIESLIST	
aiaentityclasses	IVariable::DT_AIENTITYCLASSES	
soclass	IVariable::DT_SOCLASS	
soclasses	IVariable::DT_SOCLASSES	
sostate	IVariable::DT_SOSTATE	
sostates	IVariable::DT_SOSTATES	
sopattern	IVariable::DT_SOSTATEPATTERN	
soaction	IVariable::DT_SOACTION	
sohelper	IVariable::DT_SOHELPER	
sonavhelper	IVariable::DT_SONAVHELPER	
soanimhelper	IVariable::DT_SOANIMHELPER	

Editor Name	Editor Type	
soevent	IVariable::DT_SOEVENT	
customaction	IVariable::DT_CUSTOMACTION	
gametoken	IVariable::DT_GAMETOKEN	
mat	IVariable::DT_MATERIAL	
seq	IVariable::DT_SEQUENCE	
mission	IVariable::DT_MISSIONOBJ	
anim	IVariable::DT_USERITEMCB	
animstate	IVariable::DT_USERITEMCB	
animstateEx	IVariable::DT_USERITEMCB	
bone	IVariable::DT_USERITEMCB	
attachment	IVariable::DT_USERITEMCB	
dialog	IVariable::DT_USERITEMCB	
matparamslot	IVariable::DT_USERITEMCB	
matparamname	IVariable::DT_USERITEMCB	
matparamcharatt	IVariable::DT_USERITEMCB	
seqid	IVariable::DT_SEQUENCE_ID	
lightanimation	IVariable::DT_LIGHT_ANIMATION	
formation	IVariable::DT_USERITEMCB	
communicationVariable	IVariable::DT_USERITEMCB	
uiElements	IVariable::DT_USERITEMCB	
uiActions	IVariable::DT_USERITEMCB	
uiVariables	IVariable::DT_USERITEMCB	
uiArrays	IVariable::DT_USERITEMCB	
uiMovieclips	IVariable::DT_USERITEMCB	
uiVariablesTmpl	IVariable::DT_USERITEMCB	
uiArraysTmpl	IVariable::DT_USERITEMCB	
uiMovieclipsTmpl	IVariable::DT_USERITEMCB	
uiTemplates	IVariable::DT_USERITEMCB	
vehicleParts	IVariable::DT_USERITEMCB	
vehicleSeatViews	IVariable::DT_USERITEMCB	
entityProperties	IVariable::DT_USERITEMCB	

Editor Name	Editor Type	
actionFilter	IVariable::DT_USERITEMCB	
actionMaps	IVariable::DT_USERITEMCB	
actionMapActions	IVariable::DT_USERITEMCB	
geomcache	IVariable::DT_GEOM_CACHE	
audioTrigger	IVariable::DT_AUDIO_TRIGGER	
audioSwitch	IVariable::DT_AUDIO_SWITCH	
audioSwitchState	IVariable::DT_AUDIO_SWITCH_STATE	
audioRTPC	IVariable::DT_AUDIO_RTPC	
audioEnvironment	IVariable::DT_AUDIO_ENVIRONMENT	
audioPreloadRequest	IVariable::DT_AUDIO_PRELOAD_REQUEST	
dynamicResponseSignal	IVariable::DT_DYNAMIC_RESPONSE_SIGNAL	

Trigger Ports

It can be useful to have a trigger signal as an input or output port. You can implement these ports using the `Input/OutputPortConfig_Void` or `Input/OutputPortConfig_AnyType` data types. Do not use the `Boolean` data type.

Update Event

If you want an update loop for your node instead of having it react on ports, you can use the following code to add your node to the list of regularly updated nodes. You can also choose to enable the update event temporarily.

The following code adds your node to the list of regularly updated nodes:

```
pActInfo->pGraph->SetRegularlyUpdated( pActInfo->myID, true);
```

You will get a single `ProcessEvent(eFE_Updated)` call per game update call.

To remove it from this list, call the same function with `false` as the second parameter.

Flow Graph Node Reference

This section provides a listing of the flow graph nodes, including the various types of nodes, input and output ports, and their uses. The most commonly-used and important nodes include Entity (and ComponentEntity), Interpolate, Logic, Math, Mission, Time, Vec3, and Debug nodes.

For a list of UI, VR and Cloud Canvas flow graph nodes, see [UI Flow Graph Nodes \(p. 1185\)](#), [Setting Up Virtual Reality with Flow Graph \(p. 1307\)](#), and [Cloud Canvas Flow Graph Node Reference](#).

Note

Node input/output port descriptions are also available as tool tip text when you mouseover a port in the node graph or in the **Properties** pane in the Flow Graph editor.

Topics

- [Actor Nodes \(p. 502\)](#)
- [AI Nodes \(p. 506\)](#)
- [AISequence Nodes \(p. 525\)](#)
- [Animations Nodes \(p. 532\)](#)
- [Audio Nodes \(p. 541\)](#)
- [Camera Nodes \(p. 545\)](#)
- [ComponentEntity Nodes \(p. 547\)](#)
- [CustomAction Nodes \(p. 557\)](#)
- [Debug Nodes \(p. 560\)](#)
- [Dialog Nodes \(p. 572\)](#)
- [Dynamic Response Nodes \(p. 574\)](#)
- [Engine Nodes \(p. 577\)](#)
- [Entity Nodes \(p. 579\)](#)
- [Environment Nodes \(p. 593\)](#)
- [FeatureTest Nodes \(p. 598\)](#)
- [Game Nodes \(p. 600\)](#)
- [Helicopter Nodes \(p. 606\)](#)
- [Image Nodes \(p. 608\)](#)
- [Input Nodes \(p. 619\)](#)
- [Interpolate Nodes \(p. 638\)](#)
- [Intersection Tests Nodes \(p. 644\)](#)
- [Iterator Nodes \(p. 645\)](#)
- [JSON Nodes \(p. 648\)](#)
- [Kinect Nodes \(p. 650\)](#)
- [Logic Nodes \(p. 652\)](#)
- [Material Nodes \(p. 662\)](#)
- [MaterialFX Nodes \(p. 665\)](#)
- [Math Nodes \(p. 666\)](#)
- [Mission Nodes \(p. 685\)](#)
- [Module Nodes \(p. 689\)](#)
- [Movement Nodes \(p. 693\)](#)
- [Physics Nodes \(p. 696\)](#)
- [Prefab Nodes \(p. 702\)](#)
- [ProceduralMaterial Nodes \(p. 703\)](#)
- [Stereo Nodes \(p. 714\)](#)
- [String Nodes \(p. 715\)](#)
- [System Nodes \(p. 719\)](#)
- [Time Nodes \(p. 720\)](#)
- [Twitch Nodes \(p. 730\)](#)
- [Vec3 Nodes \(p. 738\)](#)
- [Vehicle Nodes \(p. 745\)](#)
- [Video Nodes \(p. 756\)](#)

- [Weapon Nodes](#) (p. 757)
- [XML Nodes](#) (p. 760)

Actor Nodes

You can use the following flow graph nodes to configure various actor behaviors and settings.

Note

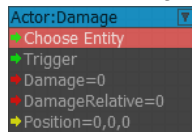
These nodes will only work with the Legacy Game Sample (CryEngine GameSDK), which is available at [Lumberyard Downloads](#).

Topics

- [Damage node](#) (p. 502)
- [EnslaveCharacter node](#) (p. 502)
- [GrabObject node](#) (p. 503)
- [HealthCheck node](#) (p. 503)
- [HealthGet node](#) (p. 504)
- [HealthSet node](#) (p. 504)
- [LocalPlayer node](#) (p. 505)
- [PlayMannequinFragment node](#) (p. 505)
- [ProcClipEventListener node](#) (p. 506)

Damage node

Used to damage the chosen entity using the **Damage** input value when the **Trigger** input is activated.

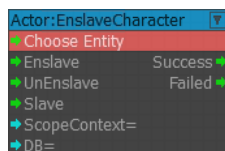


Inputs

Port	Type	Description
Trigger	Any	Activates the node
Damage	Integer	Type of damage to inflict
DamageRelative	Integer	Level of relative damage
Position	Vec3	Location the damage will occur

EnslaveCharacter node

Used to enslave one character to another character.



Inputs

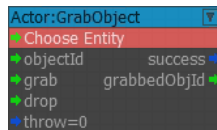
Port	Type	Description
Enslave	Any	Enslaves the character
Unenslave	Any	Unenslaves the character
Slave	Any	Character to enslave
ScopeContext	String	Context of the scope
DB	String	Optional database name

Outputs

Port	Type	Description
Success	Any	Triggers if enslaving succeeded
Failed	Any	Triggers if enslaving failed

GrabObject node

Used by the chosen entity to grab an object, then to drop or throw the object.



Inputs

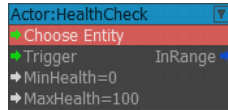
Port	Type	Description
objectId	Any	ID of the object to grab
grab	Any	Grabs the object
drop	Any	Drops the object
throw	Boolean	Throws the object

Outputs

Port	Type	Description
success	Boolean	True if the object was successfully dropped or thrown
grabbedObjId	Any	ID of the grabbed object

HealthCheck node

Used to check the health of the chosen actor entity. When the node is triggered the health of the entity is checked and if it is within the defined **MinHealth** and **MaxHealth** values, a `True` will be output on the **InRange** port.



Inputs

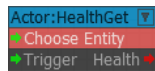
Port	Type	Description
Trigger	Any	Activates the port
MinHealth	Float	Lower limit of health range
MaxHealth	Float	Upper limit of health range

Outputs

Port	Type	Description
InRange	Boolean	True if health is between the MinHealth and MaxHealth values

HealthGet node

Used to get the health of an actor entity.



Inputs

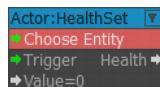
Port	Type	Description
Trigger	Any	Activate this port to get the current health of the chosen entity

Outputs

Port	Type	Description
Health	Integer	Current health of the chosen entity

HealthSet node

Used to set the health of the actor entity.



Inputs

Port	Type	Description
Trigger	Any	Activate this port to set the current health of the chosen entity

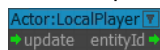
Port	Type	Description
Value	Float	Health value to the set for the chosen entity

Outputs

Port	Type	Description
Health	Integer	Current health of the chosen entity

LocalPlayer node

Used to update and output the ID of the local player entity.



Inputs

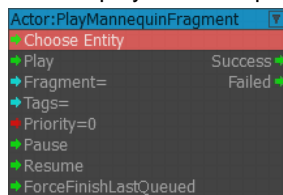
Port	Type	Description
update	Any	Updates the ID of the local player entity. Required for multiplayer games

Outputs

Port	Type	Description
entityId	Any	Outputs the ID of the local player entity

PlayMannequinFragment node

Used to play a Mannequin fragment for the chosen entity with the specified Mannequin tags.



Inputs

Port	Type	Description
Play	Any	Plays the fragment
Fragment	String	Name of the fragment
Tags	String	List of "+"-separated Mannequin tags
Priority	Integer	Priority number
Pause	Any	Pauses the entity actionController
Resume	Any	Resumes the entity actionController

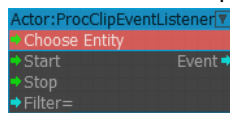
Port	Type	Description
ForceFinishLastQueued	Any	Finishes the last queued action

Outputs

Port	Type	Description
Success	Any	Triggers if the fragment command succeeded
Failed	Any	Triggers if the fragment command failed

ProcClipEventListener node

Used to listen for a procedural clip event.



Inputs

Port	Type	Description
Start	Any	Start listening for the procedural clip event
Stop	Any	Stop listening for the procedural clip event
Filter	String	Name of the filter used

Outputs

Port	Type	Description
Event	String	Outputs the procedural clip event

AI Nodes

You can use these flow graph nodes to configure AI agent behaviors and settings.

Note

These nodes will only work with the Legacy Game Sample (CryEngine GameSDK), which is available at [Lumberyard Downloads](#).

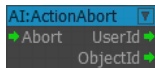
Topics

- [ActionAbort node](#) (p. 507)
- [ActiveCount node](#) (p. 508)
- [ActionEnd node](#) (p. 508)
- [ActionStart node](#) (p. 508)
- [ActiveCountInFaction node](#) (p. 509)
- [ActiveCountMonitor node](#) (p. 509)
- [AIGlobalPerceptionScaling node](#) (p. 510)
- [AlertMe node](#) (p. 510)

- [AttentionTarget node \(p. 511\)](#)
- [AutoDisable node \(p. 511\)](#)
- [Communication node \(p. 511\)](#)
- [EventListener node \(p. 512\)](#)
- [Execute node \(p. 513\)](#)
- [Faction node \(p. 514\)](#)
- [FactionReaction node \(p. 514\)](#)
- [GroupAlertness node \(p. 515\)](#)
- [GroupCount node \(p. 515\)](#)
- [GroupIDGet node \(p. 515\)](#)
- [GroupIDSet node \(p. 516\)](#)
- [IgnoreState node \(p. 516\)](#)
- [IsAliveCheck node \(p. 517\)](#)
- [LookAt node \(p. 517\)](#)
- [NavCostFactor node \(p. 518\)](#)
- [ObjectDrop node \(p. 518\)](#)
- [ObjectGrab node \(p. 519\)](#)
- [ObjectUse node \(p. 520\)](#)
- [PerceptionScale node \(p. 520\)](#)
- [RegenerateMNM node \(p. 521\)](#)
- [RequestReinforcementReadability node \(p. 521\)](#)
- [SetCommunicationVariable node \(p. 521\)](#)
- [SetFaction node \(p. 522\)](#)
- [SetState node \(p. 522\)](#)
- [ShapeState node \(p. 523\)](#)
- [Signal node \(p. 523\)](#)
- [SmartObjectEvent node \(p. 524\)](#)
- [SmartObjectHelper node \(p. 524\)](#)
- [Stance node \(p. 525\)](#)

ActionAbort node

Used to define a "clean-up" procedure that is run when an AI action is aborted.



Inputs

Port	Type	Description
Abort	Any	Aborts execution of AI action

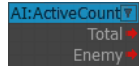
Outputs

Port	Type	Description
UserId	Any	ID of agent that is performing the action

Port	Type	Description
ObjectId	Any	ID of the object on which the agent is executing on

ActiveCount node

Used to count how many AI agents are active.

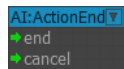


Outputs

Port	Type	Description
Total	Integer	How many total agents are active
Enemy	Integer	How many enemies there are

ActionEnd node

Used to end an AI action.

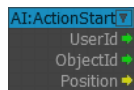


Inputs

Port	Type	Description
End	Any	Ends the AI action
Cancel	Any	Cancel the action

ActionStart node

Used to start an AI action.

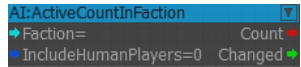


Outputs

Port	Type	Description
UserId	Any	ID of agent that is performing the action
ObjectId	Any	ID of the object on which the agent is executing on
Position	Vec3	Position of the object

ActiveCountInFaction node

Used to count how many AI factions are active.



Inputs

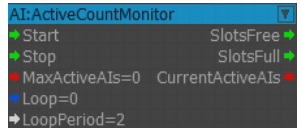
Port	Type	Description
Faction	String	Faction to be counted
IncludedHumanPlayers	Boolean	Include human players when counting active AI agents in the faction

Outputs

Port	Type	Description
Count	Integer	Number of active agents in the faction
Changed	Any	Triggers when the number of active agents in the faction changes

ActiveCountMonitor node

Used to monitor the active AI count against a limit and then periodically output the current state. When the condition is met, the monitor loop will stop automatically. This will then need to be restarted manually.



Inputs

Port	Type	Description
Start	Any	Starts monitoring
Stops	Any	Stops monitoring
MaxActiveAIs	Integer	Maximum number of active AIs limit
Loop	Boolean	Enables loop monitoring
LoopPeriod	Float	Period of time between checks if Loop is enabled

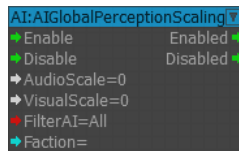
Outputs

Port	Type	Description
SlotsFree	Any	Triggers when the number of active agents drops below MaxActiveAIs

Port	Type	Description
SlotsFull	Any	Triggers when the number of active agents is equal to or above MaxActiveAIs
CurrentActiveAIs	Integer	Current number of active AI agents

AI:GlobalPerceptionScaling node

Used to specify a global scale for AI perception.



Inputs

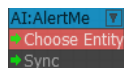
Port	Type	Description
Enable	Any	Enables perception scaling
Disable	Any	Disables perception scaling
AudioScale	Float	Auditory perception scaling factor
VisualScale	Float	Visual perception scaling factor
FilterAI	Integer	Filter which AI agents are used
Faction	String	Faction

Outputs

Port	Type	Description
Enabled	Any	Triggers when node is enabled
Disabled	Any	Triggers when node is disabled

AlertMe node

A generic AI signal.

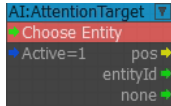


Inputs

Port	Type	Description
Sync	Any	Generic AI signal

AttentionTarget node

Used to output an AI agent's attention target.



Inputs

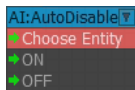
Port	Type	Description
Active	Boolean	Activates the node

Outputs

Port	Type	Description
Pos	Vec3	Position of the attention target
EntityId	Any	Entity ID of attention target
None	Any	Triggers when there is no attention target

AutoDisable node

Used to control auto-disabling.

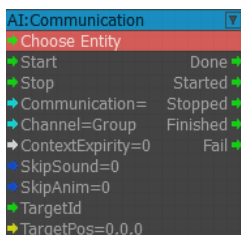


Inputs

Port	Type	Description
On	Any	Enables auto-disabling
Off	Any	Disables auto-disabling

Communication node

Used to specify the communication that an AI agent plays.



Inputs

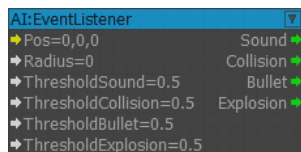
Port	Type	Description
Start	Any	Starts communication
Stop	Any	Stops communication
Communication	String	Name of communication to play
Channel	String	Name of channel to play the communications in
ContextExpiry	Float	How much time must elapse before communication can be played again
SkipSound	Boolean	Skips sound component
SkipAnim	Boolean	Skips animation component
TargetId	Any	(Optional) Target ID to play communication at
TargetPos	Vec3	(Optional) Target position to play communication at

Outputs

Port	Type	Description
Done	Any	Triggered when communication has finished playing
Started	Any	Triggers if communication is started
Stopped	Any	Triggers if communication is stopped
Finished	Any	Triggers if communication has finished playing
Fail	Any	Triggers if communication has failed

EventListener node

Used to listen for an event.



Inputs

Port	Type	Description
Pos	Vec3	Position of the listener
Radius	Float	Listening radius of the listener
ThresholdSound	Float	Sensitivity of the sound output
ThresholdCollision	Float	Sensitivity of the collision output

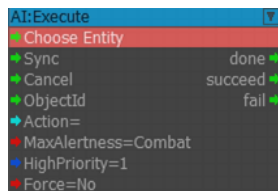
Port	Type	Description
ThresholdBullet	Float	Sensitivity of the bullet output
ThresholdExplosion	Float	Sensitivity of the explosion output

Outputs

Port	Type	Description
Sound	Any	Triggers for a sound event
Collision	Any	Triggers for a collision event
Bullet	Any	Triggers for a bullet event
Explosion	Any	Triggers for an explosion event

Execute node

Used to execute an AI action.



Inputs

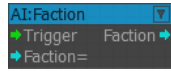
Port	Type	Description
Sync	Any	Activates the node
Cancel	Any	Cancels the operation
ObjectId	Any	ID of the entity that receives the action that is executed
Action	String	Action to be executed
MaxAlertness	Integer	Maximum alertness that allows execution
HighPriority	Boolean	Action priority level
Force	Integer	Force execution method

Outputs

Port	Type	Description
Done	Any	Triggers when the action has been executed
Succeed	Any	Triggers if the action is executed
Fail	Any	Triggers if the action is not executed

Faction node

Used to trigger an AI faction.



Inputs

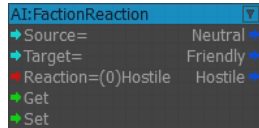
Port	Type	Description
Trigger	Any	Triggers the output
Faction	String	Name of faction to trigger

Outputs

Port	Type	Description
Faction	String	Outputs the faction that was triggered

FactionReaction node

Used to set or get AI faction reaction information.



Inputs

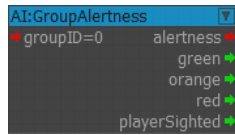
Port	Type	Description
Source	String	Source faction
Target	String	Target faction
Reaction	Integer	Source faction reaction to target faction
Get	Any	Gets the faction reaction and triggers output
Set	Any	Sets the faction reaction and triggers output

Outputs

Port	Type	Description
Neutral	Boolean	Triggers if source faction reaction to target faction is neutral
Friendly	Boolean	Triggers if source faction reaction to target faction is friendly
Hostile	Boolean	Triggers if source faction reaction to target faction is hostile

GroupAlertness node

Used to output the alertness level of any AI agent in a group.



Inputs

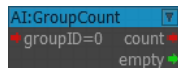
Port	Type	Description
GroupID	Integer	ID of group to set alertness level for

Outputs

Port	Type	Description
Alertness	Integer	Alertness level of the group
Green	Any	Triggers if alertness level is green
Orange	Any	Triggers if alertness level is orange
Red	Any	Triggers if alertness level is red
PlayerSighted	Any	Triggers if the player has been sighted

GroupCount node

Used to output the AI agent count in a group.



Inputs

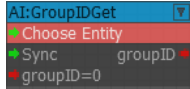
Port	Type	Description
GroupID	Integer	Agent group ID

Outputs

Port	Type	Description
Count	Integer	Number of agents in the group
Empty	Any	Triggers if no agents are in the group

GroupIDGet node

Used to output the group ID for an AI agent.



Inputs

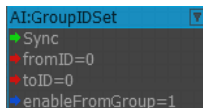
Port	Type	Description
Sync	Any	Triggers the output
GroupId	Integer	Group ID

Outputs

Port	Type	Description
GroupId	Integer	Outputs agent group ID

GroupIDSet node

Used to set the group ID for an AI agent.



Inputs

Port	Type	Description
Sync	Any	Triggers the output
FromId	Integer	The group to be merged
ToId	Integer	The group to merge to
EnabledFromGroup	Boolean	Enables members of the FromID group

IgnoreState node

Used to make an AI agent ignore enemies or to be ignored.

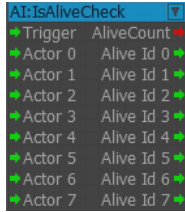


Inputs

Port	Type	Description
Hostile	Any	Activates the node
Ignore	Any	Agent will ignore enemies
ResetPerception	Any	Resets Ignore state

IsAliveCheck node

Used to check which AI actors of a group are active.



Inputs

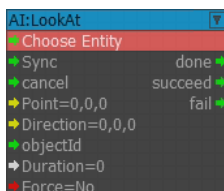
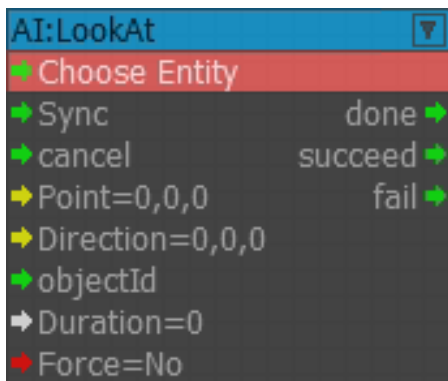
Port	Type	Description
Trigger	Any	Triggers the node
Actor 0 - 7	Any	Specific actors to check

Outputs

Port	Type	Description
AliveCount	Integer	Number of actors that are alive
AliveId0 - 7	Any	Triggers if specific actor is alive

LookAt node

Used to make an AI agent look at a specific location, an entity, or a direction.



Inputs

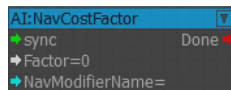
Port	Type	Description
Sync	Any	Activates the node
Cancel	Any	Cancels the operation
Point	Vec3	Point for agent to look at
Direction	Vec3	Direction for agent to look along
ObjectId	Any	ID of object for agent to look at
Duration	Float	Time in seconds for agent to look
Force	Integer	Force execution method

Outputs

Port	Type	Description
Done	Any	Triggers when agent is done looking
Succeed	Any	Triggers if agent is looking
Fail	Any	Triggers if agent is not looking

NavCostFactor node

Used to set the AI navigation cost factor for traveling through a region.



Inputs

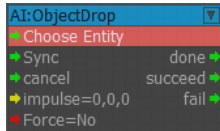
Port	Type	Description
Sync	Any	Activates the node
Factor	Float	Navigation cost factor
NavModifierName	String	Name of the cost factor navigation modifier

Outputs

Port	Type	Description
Done	Integer	Triggers when cost factor for travelling through a region has been set

ObjectDrop node

Used to have an AI agent drop a grabbed object.



Inputs

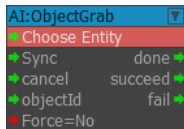
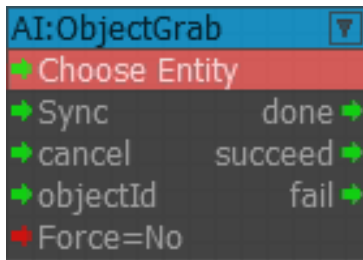
Port	Type	Description
Sync	Any	Activates the node
Cancel	Any	Cancels the operation
Impulse	Vec3	Impulse strength for dropping object
Force	Integer	Force execution method

Outputs

Port	Type	Description
Done	Any	Triggers when object has been dropped
Succeed	Any	Triggers if object is dropped
Fail	Any	Triggers if object is not dropped

ObjectGrab node

Used to make an AI agent grab an object.



Inputs

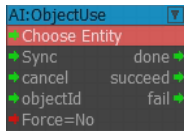
Port	Type	Description
Sync	Any	Activates the node
Cancel	Any	Cancels the operation
ObjectId	Any	Object to be grabbed
Force	Integer	Force execution method

Outputs

Port	Type	Description
Done	Any	Triggers when object has been grabbed
Succeed	Any	Triggers if object is grabbed
Fail	Any	Triggers if object is not grabbed

ObjectUse node

Used to make an AI agent use an object.



Inputs

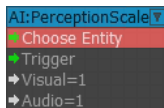
Port	Type	Description
Sync	Any	Activates the node
Cancel	Any	Cancels the operation
ObjectId	Any	Object to be used
Force	Integer	Force execution method

Outputs

Port	Type	Description
Done	Any	Triggers when object has been used
Succeed	Any	Triggers if object is used
Fail	Any	Triggers if object is not used

PerceptionScale node

Used to scale the perception for an AI agent.



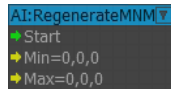
Inputs

Port	Type	Description
Trigger	Any	Triggers the node

Port	Type	Description
Visual	Float	Visual perception scale factor
Audio	Float	Auditory perception scale factor

RegenerateMNM node

Used to regenerate the AI multi-navation mesh.

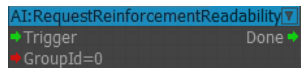


Inputs

Port	Type	Description
Start	Any	Triggers recalculation of MNM data for the bounding box
Min	Vec3	Minimum limit of bounding box
Max	Vec3	Maximum limit of bounding box

RequestReinforcementReadability node

Used to make an AI agent request reinforcements. There is no guarantee that the action will be performed however.



Inputs

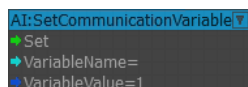
Port	Type	Description
Trigger	Any	Triggers the node
GroupId	Integer	ID of the group that is notified

Outputs

Port	Type	Description
Done	Any	Triggers when group has been notified

SetCommunicationVariable node

Used to set the communication variable that an AI agent uses to communicate their intentions.

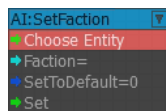


Inputs

Port	Type	Description
Set	Any	Sets the variable
VariableName	String	Variable to be set
VariableValue	Boolean	Value of variable

SetFaction node

Used to set the faction that an AI agent belongs to.

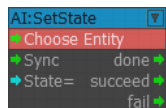


Inputs

Port	Type	Description
Faction	String	Faction to be set
SetToDefault	Boolean	Set to default faction
Set	Any	Sets the faction for the agent

SetState node

Used to set the Smart Object state for an AI agent.



Inputs

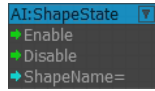
Port	Type	Description
Sync	Any	Activates the node
State	String	Smart object state to be set

Outputs

Port	Type	Description
Done	Any	Triggers when state has been set
Succeed	Any	Triggers if state has been set
Fail	Any	Triggers if state has not been set

ShapeState node

Use to enable or disable an AI shape.

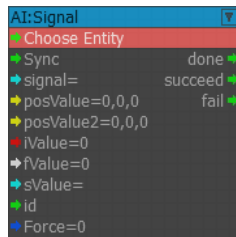


Inputs

Port	Type	Description
Enable	Any	Enables the AI shape
Disable	Any	Disables the AI shape
ShapeName	String	Name of the AI shape

Signal node

Sends an AI agent a signal.



Inputs

Port	Type	Description
Sync	Any	Activates the node
Signal	String	Name of the signal to be sent
PosValue	Vec3	Position value 1 of the signal
PosValue2	Vec3	Position value 2 of the signal
IValue	Integer	Integer value of the signal
FValue	Float	Floating point value of the signal
SValue	String	String value of the signal
Id	Any	ID of the signal
Force	Boolean	Force execution method

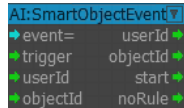
Outputs

Port	Type	Description
Done	Any	Triggers when the signal has been sent

Port	Type	Description
Succeed	Any	Triggers if the signal is sent
Fail	Any	Triggers if the signal is not sent

SmartObjectEvent node

Used to trigger a smart object event.



Inputs

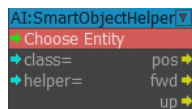
Port	Type	Description
Event	String	Smart object event to be triggered
Trigger	Any	Triggers the event
UserId	Any	Limits event to specific user ID
ObjectId	Any	Limits event to specific object

Outputs

Port	Type	Description
UserId	Any	ID of the user that receives the event
ObjectId	Any	ID of the object that receives the event
Start	Any	Triggers if matching rule is found
NoRule	Any	Triggers if no matching rule is found

SmartObjectHelper node

Used to output an AI agent's attention target parameter.



Inputs

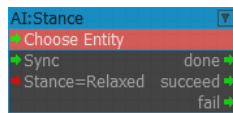
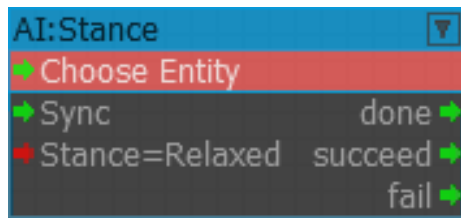
Port	Type	Description
Class	String	Class of smart object helper
Helper	String	Name of smart object helper

Outputs

Port	Type	Description
Pos	Vec3	Position of smart object helper
Fwd	Vec3	Forward direction of smart object helper
Up	Vec3	Up direction of smart object helper

Stance node

Used to control an AI agent's body stance.



Inputs

Port	Type	Description
Sync	Any	Activates the node
Stance	Integer	Body stance of the agent

Outputs

Port	Type	Description
Done	Any	Triggers when body stance has been completed
Succeed	Any	Triggers if stance has changed
Fail	Any	Triggers if stance has not changed

AISequence Nodes

You can use these flow graph nodes to configure AI sequence behaviors and settings. All AI sequence nodes must be executed with the **AISequence:Start** node.

Note

These nodes will only work with the Legacy Game Sample (CryEngine GameSDK), which is available at [Lumberyard Downloads](#).

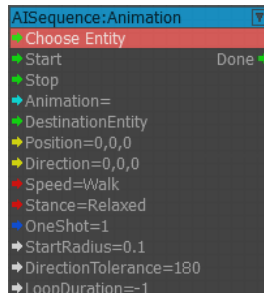
Topics

- [Animation node \(p. 526\)](#)

- [ApproachAndEnterVehicle](#) node (p. 527)
- [Bookmark](#) node (p. 527)
- [End](#) node (p. 528)
- [HoldFormation](#) node (p. 528)
- [JoinFormation](#) node (p. 528)
- [Move](#) node (p. 529)
- [MoveAlongPath](#) node (p. 529)
- [Shoot](#) node (p. 530)
- [Stance](#) node (p. 530)
- [Start](#) node (p. 531)
- [VehicleRotateTurret](#) node (p. 531)
- [Wait](#) node (p. 532)

Animation node

Used to make an AI agent move to a specific location and play an animation.



Inputs

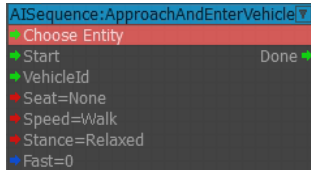
Port	Type	Description
Start	Any	Starts the animation
Stop	Any	Stops the animation
Animation	String	Name of the animation
DestinationEntity	Any	Destination to move to
Position	Vec3	Position to move to
Direction	Vec3	Direction to move along
Speed	Integer	Speed of movement
Stance	Integer	Stance while moving
OneShot	Boolean	True for a one-shot animation, false for a looping animation
StartRadius	Float	Start radius
DirectionTolerance	Float	Direction tolerance
LoopDuration	Float	Duration of the looping animation; ignored for a one-shot animation

Outputs

Port	Type	Description
Done	Any	Outputs when the animation has completed

ApproachAndEnterVehicle node

Used to make an AI agent approach and then enter a vehicle.



Inputs

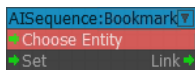
Port	Type	Description
Start	Any	Allows the AI agent to move to, and enter the specified vehicle
VehicleID	Any	Vehicle to be entered
Seat	Integer	Seat to be entered
Speed	Integer	Speed the AI agent approaches the vehicle at
Stance	Integer	Stance while approaching the vehicle
Fast	Boolean	Skip the approaching animation

Outputs

Port	Type	Description
Done	Any	Outputs when the AI agent has completed entering the vehicle

Bookmark node

Used to define a bookmark in a sequence of AI actions from which the sequence will resume after being interrupted.



Input

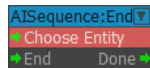
Port	Type	Description
Set	Any	Sets a bookmark for the AI sequence from which to resume from

Outputs

Port	Type	Description
Link	Any	Link to other nodes

End node

Used to define the end of a sequence of AI actions. This frees the AI agent to resume typical behaviors.



Inputs

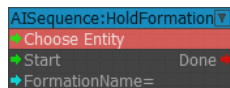
Port	Type	Description
End	Any	Triggers to end the AI sequence

Outputs

Port	Type	Description
Done	Any	Outputs when the AI sequence has ended

HoldFormation node

Use to create a formation to have an AI agent hold to.



Inputs

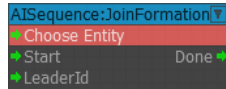
Port	Type	Description
Start	Any	Start formation hold
FormationName	String	Name of the formation

Outputs

Port	Type	Description
Done	Any	Triggered when the formation is complete

JoinFormation node

Use to have an AI agent join a formation.



Inputs

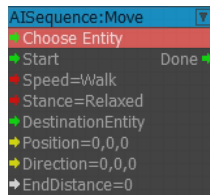
Port	Type	Description
Start	Any	Start formation join
LeaderId	Any	ID of the leader

Outputs

Port	Type	Description
Done	Any	Triggered when formation join is complete

Move node

Use to command an AI agent to move to a location.



Inputs

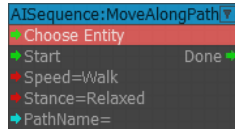
Port	Type	Description
Start	Any	Start movement
Speed	Integer	Movement speed
Stance	Integer	Stance while moving
DestinationEntity	Any	Destination entity to move to
Position	Vec3	Position to move to
Direction	Vec3	Direction to move along
EndDistance	Float	End distance to move to

Outputs

Port	Type	Description
Done	Any	Triggered when movement is complete

MoveAlongPath node

Use to have an AI agent move along a path.



Inputs

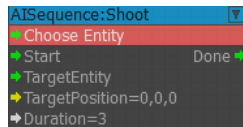
Port	Type	Description
Start	Any	Begins AI agent movement
Speed	Integer	Speed of the agent
Stance	Integer	Stance of the agent used while following the path
PathName	String	Name of the path the agent will follow

Outputs

Port	Type	Description
Done	Any	Triggered when the agent has completed the path

Shoot node

Use to make an AI agent shoot at an entity or a location for a specified length of time.



Inputs

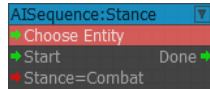
Port	Type	Description
Start	Any	Starts the AI agent shooting
TargetEntity	Any	Entity the agent will shoot at
TargetPosition	Vec3	Position the agent will shoot at
Duration	Float	Length of time to shoot for

Outputs

Port	Type	Description
Done	Any	Triggers when shooting is finished

Stance node

Use to control the stance of an AI agent.



Inputs

Port	Type	Description
Start	Any	Starts the stance
Stance	Integer	Name of the stance the AI agent will use

Outputs

Port	Type	Description
Done	Any	Triggers when the stance is complete

Start node

Use to define the start of an AI sequence of actions. All AI sequence nodes must be executed using this node.



Inputs

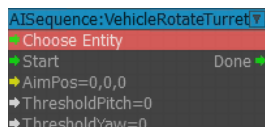
Port	Type	Description
Start	Any	Starts the AI sequence
Interruptible	Boolean	The agent will automatically stop when not in the Idle state.
ResumeAfterInterruption	Boolean	AI sequence will automatically resume from the start or from the bookmark ID the agent returns to from the Idle state

Outputs

Port	Typed	Description
Link	Any	Link to other nodes

VehicleRotateTurret node

Use to rotate a vehicle turret to an aiming position.



Inputs

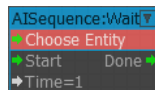
Port	Type	Description
Start	Any	Starts turret rotation
AimPos	Vec3	Position to aim at with the turret
ThresholdPitch	Float	Threshold of the pitch angle before triggering the output port. Must be used with the ThresholdYaw port
ThresholdYaw	Float	Threshold of the yaw angle before triggering the output port. Must be used with the ThresholdPitch port

Outputs

Port	Type	Description
Done	Any	Starts turret rotation

Wait node

Used to make the AI agent wait for a specified length of time.



Inputs

Port	Type	Description
Start	Any	Start waiting
Time	Float	Duration to wait for

Outputs

Port	Type	Description
Done	Any	Triggered when wait has completed

Animations Nodes

You can use these flow graph nodes to configure animation-related settings.

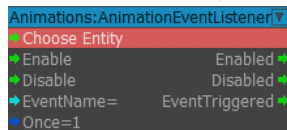
Topics

- [AnimationEventListener node \(p. 533\)](#)
- [AttachmentControl node \(p. 533\)](#)
- [BoneInfo node \(p. 534\)](#)
- [CheckAnimPlaying node \(p. 534\)](#)
- [CooperativeAnimation node \(p. 535\)](#)

- [LookAt node \(p. 536\)](#)
- [NoAiming node \(p. 537\)](#)
- [PlayAnimation node \(p. 537\)](#)
- [PlayCGA node \(p. 538\)](#)
- [PlaySequence node \(p. 538\)](#)
- [StopAnimation node \(p. 540\)](#)
- [SynchronizeTwoAnimations node \(p. 540\)](#)
- [TriggerOnKeyTime node \(p. 541\)](#)

AnimationEventListener node

Use to listen for a specific animation event and trigger the output.



Inputs

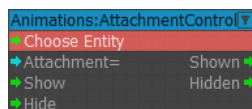
Port	Type	Description
Enable	Any	Start listening for animation events
Disable	Any	Stop listening for animation events
EventName	String	Name of the animation event to listen for
Once	Boolean	If set to True, the node will be disabled after the event has been received.

Outputs

Port	Type	Description
Enabled	Any	Triggered when listening has started
Disabled	Any	Triggered when listening has stopped
EventTriggered	Any	Triggered when the animation event is received

AttachmentControl node

Use to add and control an attachment for a character.



Inputs

Port	Type	Description
Attachment	String	Name of the attachment
Show	Any	Show the attachment

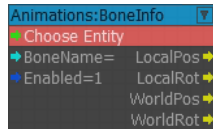
Port	Type	Description
Hide	Any	Hide the attachment

Outputs

Port	Type	Description
Shown	Any	Triggered when the attachment is shown
Hidden	Any	Triggered when the attachment is hidden

BoneInfo node

Use to specify and output character bones to create attachments or link objects.



Inputs

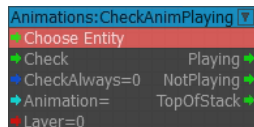
Port	Type	Description
BoneName	String	Name of the bone to get information for
Enabled	Boolean	Enables and disables the node

Outputs

Port	Type	Description
LocalPos	Vec3	Position of the bone in local space
LocalRot	Vec3	Rotation of the bone in local space
WorldPos	Vec3	Position of the bone in world space
WorldRot	Vec3	Rotation of the bone in world space

CheckAnimPlaying node

Use to check whether a defined animation is playing or not.



Inputs

Port	Type	Description
Check	Any	Checks once whether the animation is playing

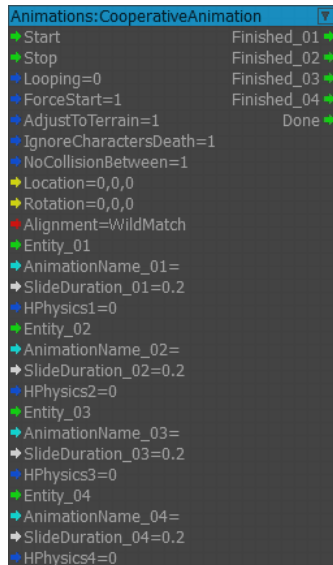
Port	Type	Description
CheckAlways	Boolean	Checks each frame whether the animation is playing
Animation	String	Name of the animation
Layer	Integer	Which layer the animation should be playing on

Outputs

Port	Type	Description
Playing	Any	Triggers when the animation is playing on the layer
NotPlaying	Any	Triggers when the animation is not playing on the layer
TopOfStack	Any	Triggers when the animation is at the top of the stack, meaning it is not currently blended out

CooperativeAnimation node

Use to allow the playing of a positioned and aligned animation for one or more characters.



Inputs

Port	Type	Description
Start	Any	Starts the animation
Stop	Any	Stops the animation
ForceStart		Force the animation to start
AdjustToTerrain		Makes sure the character is at terrain level

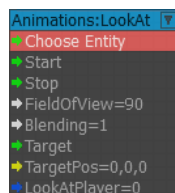
Port	Type	Description
IgnoreCharactersDeath		If false and any of the actors die, stops the animation for all the actors
NoCollisionBetween		If true, the first actor won't collide with the other actors
Location		Starts the animation at a specific location
Rotation		Starts the animation at a specific rotation
Alignment		Alignment type: <ul style="list-style-type: none"> • WildMatch: Move both characters the least amount • FirstActor: First actor can be rotated but not moved • FirstActorNoRot: First actor can neither be moved or rotated • FirstActorPosition: Slides the actor so that the first one is at the specified Location • Location: Moves both characters until the reference point of the animation is at the specified location
Entity_01 - Entity_04		Name of the specific entity
AnimationName_01 - AnimationName_4		Name of the specific animation
SlideDuration_01 - SlideDuration_04		Time in seconds to slide the entity into position
HPhysics1 - HPhysics4		Prohibits the character from being pushed through solid objects

Outputs

Port	Type	Description
Finished_01 - Finished_04	Any	Activates when the specific actor is done
Done	Any	Activates when all actors are done

LookAt node

Use to make a character look at a position.

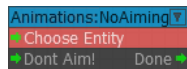


Inputs

Port	Type	Description
Start	Any	Character begins looking at a target
Stop	Any	Character stops looking at a target
FieldOfView	Float	Field of view for the character
Blending	Float	
Target	Any	The target for the character to look at
TargetPos	Vec3	The target look position
LookAtPlayer	Boolean	Character looks at player

NoAiming node

Use to suppress aiming for a character.



Inputs

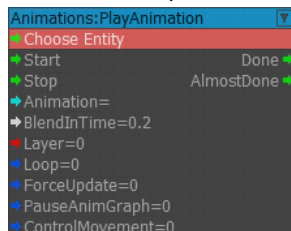
Port	Type	Description
Dont Aim!	Any	Suppresses aiming for a character

Outputs

Port	Type	Description
Done	Any	Triggered when aiming has ceased

PlayAnimation node

Use to play an animation on the character's skeleton, bypassing the AnimationGraph. The animation name can be specified directly as mapped in the .cal file.



Inputs

Port	Type	Description
Start	Any	Starts the animation

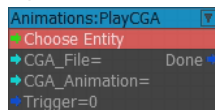
Port	Type	Description
Stop	Any	Stops the animation
Animation	String	Name of the animation to play
BlendInTime	Float	Blend-in time in seconds
Layer	Integer	Layer on which to play the animation
Loop	Boolean	Used to loop the animation indefinitely
ForceUpdate	Boolean	Animation plays even if not visible
PauseAnimGraph	Boolean	Deprecated
ControlMovement	Boolean	Controls movement of the entities

Outputs

Port	Type	Description
Done	Any	Triggers when the animation is done
AlmostDone	Any	Triggers when the animation is almost done

PlayCGA node

Use to play .cga files and their animation, as well as .anm files belonging to the .cga file. The Trigger input starts the animation.



Inputs

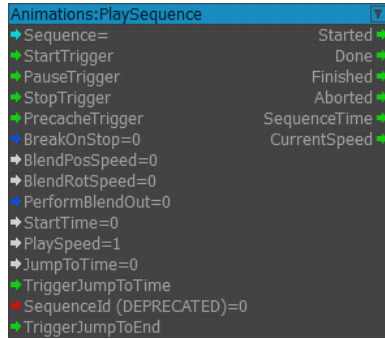
Port	Type	Description
CGA_File	String	File name of the animation
CGA_Animation	String	Name of the animation
Trigger	Boolean	Starts the animation

Outputs

Port	Type	Description
Done	Boolean	Triggers when the animation has finished

PlaySequence node

Use to play a Track View sequence. Use the PerformBlendOut input to make sure that the camera has a seamless blend into the game camera when the sequence is over. Make sure to beam the player to the right place when the sequence starts.



Inputs

Port	Type	Description
Sequence	String	Name of the sequence
StartTrigger	Any	Starts the sequence
PauseTrigger	Any	Pauses the sequence
StopTrigger	Any	Stops the sequence
PrecacheTrigger	Any	Precache keys that start in the first few seconds of the animation
BreakOnStop	Boolean	If set to true, stopping the sequence doesn't jump it to the end
BlendPosSpeed	Float	Speed at which the position gets blended into the animation
BlendRotSpeed	Float	Speed at which the rotation gets blended into the animation
PerformBlendOut	Boolean	If True, the cutscene will blend out after it has finished to the new view
StartTime	Float	Time at which the sequence will start playing
PlaySpeed	Float	Speed at which the sequence plays at
JumpToTime	Float	Jump to a specific time in the sequence
TriggerJumpToTime	Any	Trigger the animation to jump the animation to the sequence time specified
TriggerJumpToEnd	Any	Trigger the animation to jump the animation to the end of the sequence.

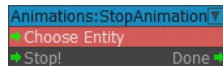
Outputs

Port	Type	Description
Started	Any	Triggered when the animation starts
Done	Any	Triggered when the animation has stopped or is aborted

Port	Type	Description
Finished	Any	Triggered when the animation has stopped
Aborted	Any	Triggered when the animation is aborted
SequenceTime	Float	Current time of the sequence
CurrentSpeed	Float	Speed at which the sequence is being played

StopAnimation node

Use to stop the animation.



Inputs

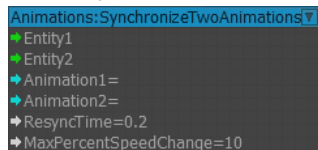
Port	Type	Description
Stop!	Any	Stops the animation

Outputs

Port	Type	Description
Done	Any	Triggers when the the animation has stopped

SynchronizeTwoAnimations node

Use to synchronize two animations for two entities.

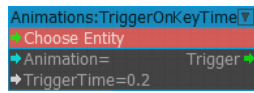


Inputs

Port	Type	Description
Entity1	Any	First entity to synchronize
Entity2	Any	Second entity to synchronize
Animation1	Sstring	First animation to synchronize
Animation2	String	Second animation to synchronize
ResyncTime	Float	Resync time
MaxPercentSpeedChange	Float	Maximum percentage speed change

TriggerOnKeyTime node

Use to play and output an animation at a specified time.



Inputs

Port	Type	Description
Animation	String	Animation to play
TriggerTime	Float	When to play the animation

Outputs

Port	Type	Description
Trigger	Any	Plays the animation

Audio Nodes

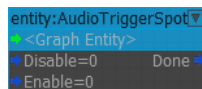
The following flow graph nodes are used to control various audio system functionality and settings. All references to Audiokinetic Wwise also applies to the LTX version.

Topics

- [entity:AudioTriggerSpot node \(p. 541\)](#)
- [entity:AudioAreaEntity node \(p. 542\)](#)
- [entity:AudioAreaAmbience node \(p. 542\)](#)
- [entity:AudioAreaRandom node \(p. 543\)](#)
- [PreloadData node \(p. 543\)](#)
- [Rtpc node \(p. 544\)](#)
- [Switch node \(p. 544\)](#)
- [Trigger node \(p. 544\)](#)

entity:AudioTriggerSpot node

Used to enable and disable the associated entity.



Inputs

Port	Type	Description
Disable	Boolean	Stops the sound. If available, triggers the event set in the StopTriggerName property.

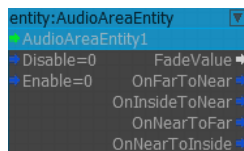
Port	Type	Description
Enable	Boolean	Starts the sound. Triggers the event set in the PlayTriggerName property.

Outputs

Port	Type	Description
Done	Boolean	Outputs when the event started by the play trigger has completed playing.

entity:AudioAreaEntity node

Used to enable and disable the associated entity, as well as control what happens when the player enters and leaves the shape.



Inputs

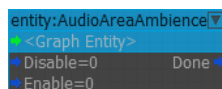
Port	Type	Description
Disable	Boolean	Stops the sound. If available, triggers the event set in the StopTriggerName property.
Enable	Boolean	Starts the sound. Triggers the event set in the PlayTriggerName property.

Outputs

Port	Type	Description
FadeValue	Float	Normalized value from 0 to 1 of the FadeDistance when the player approaches the shape.
OnFarToNear	Boolean	Player enters the fade distance
OnInsideToNear	Boolean	Player leaves the shape
OnNearToFar	Boolean	Player leaves the fade distance
OnNearToInside	Boolean	Player enters the shape

entity:AudioAreaAmbience node

Used to enable and disable the associated entity.



Inputs

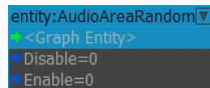
Port	Type	Description
Disable	Boolean	Disables the audio entity
Enable	Boolean	Enables the audio entity

Outputs

Port	Type	Description
Done	Boolean	Outputs when the audio entity is enabled.

entity:AudioAreaRandom node

Used to enable and disable the associated entity.

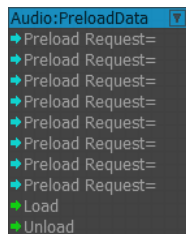


Inputs

Port	Type	Description
Disable	Boolean	Disables the entity
Enable	Boolean	Enables the entity

PreloadData node

Used to load and unload preload requests to optimize memory consumption. This node lists only preloads that are not set to **Autoload** in the Audio Controls Editor.



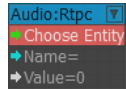
Inputs

Port	Type	Description
Preload Request	String	Defines the preload requests that should be loaded or unloaded
Load	Any	Loads the preload requests
Unload	Any	Loads the preload requests

Rtpc node

Use to change RTPC values. If you have an entity assigned to this node, the RTPC assigned to the **Name** input controls parameters only on the assigned entity. If no entity is assigned, the parameter change is applied to all entities.

For Wwise, any RTPC that is not assigned to an entity sets connected game parameters on all game objects. An RTPC that is assigned to an entity sets the connected game parameters only on the game object corresponding to the assigned entity in Wwise. You can monitor the RTPC changes for an entity in the game object profiler layout.



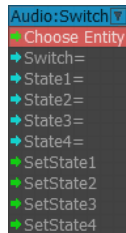
Inputs

Port	Type	Description
Name	String	Name of the RTPC
Value	Float	Sets the RTPC value

Switch node

Used to set the state of a switch. Multiple states can be selected in the node to reduce the complexity of Flow Graph logic when more than one state change should happen.

For Wwise, a connected switch state sets the Wwise switch only on a game object corresponding to the assigned entity. A switch state connected to a Wwise switch without an assigned entity is set on the Dummy Game object in Wwise. A switch state connected to a Wwise state always sets the state globally, regardless of the assigned entity.



Inputs

Port	Type	Description
Switch	String	Switch name
State1 - State4	String	Name of the state
SetState1 - SetState4	Any	Sets the state

Trigger node

Used to trigger events.

For Wwise, a trigger without an entity assigned is executed on the dummy game object in Wwise. A trigger with an entity assigned is executed on the game object corresponding to the assigned entity.



Inputs

Port	Type	Description
PlayTrigger	String	The name of the event. Any event can be triggered with this node.
StopTrigger	String	The name of the event. Any event can be triggered with this node. If no event is defined and a sound is started on the corresponding PlayTrigger, it stops at once when the stop input is triggered.
Play	Any	Triggers the event defined in the PlayTrigger input.
Stop	Any	Triggers the event defined in the StopTrigger input.

Outputs

Port	Type	Description
Done	Any	Event has completed

Camera Nodes

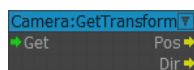
You can use the following flow graph nodes to configure player camera settings.

Topics

- [GetTransform node \(p. 545\)](#)
- [View node \(p. 546\)](#)
- [ViewShakeEx node \(p. 546\)](#)

GetTransform node

Used to get and output the position and direction of the player camera.



Inputs

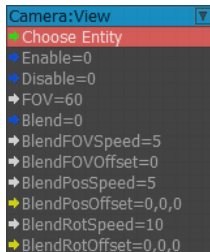
Port	Type	Description
Get	Any	Triggers the retrieval of the currently active camera position and direction

Outputs

Port	Type	Description
Pos	Vec3	Output camera position
Dir	Vec3	Output camera direction

View node

Used to create a custom view linked to the chosen entity.

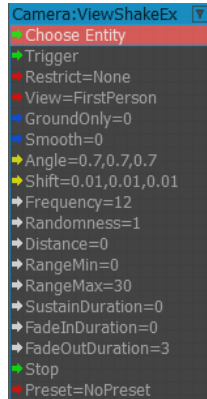


Inputs

Port	Type	Description
Enable	Boolean	Activates the node
Disable	Boolean	Deactivates the node
FOV	Float	Camera field of view
Blend	Boolean	Whether to blend the camera or not
BlendFOVSpeed	Float	Blended field of view speed
BlendFOVOffset	Float	Blended field of view offset
BlendPosSpeed	Float	Blended position speed
BlendPosOffset	Vec3	Blended position offset
BlendRotSpeed	Float	Blended rotation speed
BlendRotOffset	Vec3	Blended rotation offset

ViewShakeEx node

Used to enable camera shake on the player's view. You can specify the fade in and out durations and stop the effect.



Inputs

Port	Type	Description
Trigger	Any	Triggers the node
Restrict	Integer	Restricts the view
View	Integer	Which camera view to use
GroundOnly	Boolean	Apply shake only when the player is standing on the ground
Smooth	Boolean	AAAnys sudden direction changes
Angle	Vec3	Shake angle
Shift	Vec3	Shake shift
Frequency	Float	Shake frequency
Randomness	Float	Randomness of shake
Distance	Float	Distance to effect source
RangeMin	Float	Minimum strength effect range
RangeMax	Float	Maximum strength effect range
SustainDuration	Float	Duration of the non-fading part of the shake
FadeInDuration	Float	Fade in time
FadeOutDuration	Float	Fade out time
Stop	Any	Stops the shake
Preset	Integer	Preset input values

ComponentEntity Nodes

You can use the following flow graph nodes to get and set various component entity system settings. These nodes only work with the component entity system.

In addition, with the following exceptions, any flow graph node that has an **Entity ID** input port will not work with the component entity system nodes by default:

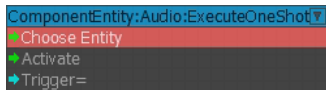
- **Physics:ActionImpulse** node
- **Physics:Dynamics** node
- **Movement:RotateEntity** node
- **Entity:EntityID** node

Topics

- [Audio:ExecuteOneShot](#) node (p. 548)
- [Audio:StopOneShot](#) node (p. 548)
- [EventActionHandler:AZVector3](#) node (p. 549)
- [EventActionHandler:EntityID](#) node (p. 549)
- [EventActionHandler:Float](#) node (p. 550)
- [EventActionSender:AZVector3](#) node (p. 550)
- [EventActionSender:EntityID](#) node (p. 551)
- [EventActionSender:Float](#) node (p. 551)
- [GameplayEventHandler:AZVector3](#) node (p. 551)
- [GameplayEventHandler:EntityID](#) node (p. 552)
- [GameplayEventHandler:Float](#) node (p. 552)
- [GameplayEventSender:AZVector3](#) node (p. 553)
- [GameplayEventSender:EntityID](#) node (p. 553)
- [GameplayEventSender:Float](#) node (p. 554)
- [Light:Switch](#) node (p. 554)
- [Particles:Switch](#) node (p. 554)
- [TransformComponent:GetEntityPosition](#) node (p. 555)
- [TransformComponent:GetEntityRotation](#) node (p. 555)
- [TransformComponent:SetEntityPosition](#) node (p. 556)
- [TransformComponent:SetEntityRotation](#) node (p. 556)
- [TriggerComponent:EnterTrigger](#) node (p. 557)

Audio:ExecuteOneShot node

Used to execute the audio trigger as a one-shot on the entity.



Inputs

Port		Description
Activate	Any	Activates the node
Trigger	String	Audio trigger

Audio:StopOneShot node

Used to stop the specified audio one shot trigger.

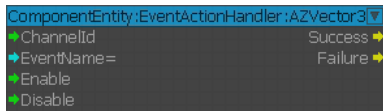


Inputs

Port		Description
Activate	Any	Activates the node
Trigger	String	Audio trigger

EventActionHandler:AZVector3 node

Used for the entity event action handler.



Inputs

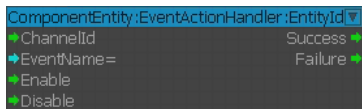
Port		Description
ChannelId	Any	Entity channel ID
EventName	String	Name of the event action handler
Enable	Any	Enables the event action handler
Disable	Any	Disables the event action handler

Outputs

Port		Description
Success	Vec3	Vector value on event action handler success
Failure	Vec3	Vector value on event action handler failure

EventActionHandler:EntityID node

Used for the entity event action handler.



Inputs

Port		Description
ChannelId	Any	Entity channel ID
EventName	String	Name of the event action handler

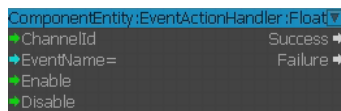
Port		Description
Enable	Any	Enables the event action handler
Disable	Any	Disables the event action handler

Outputs

Port		Description
Success	Any	Value on event action handler success
Failure	Any	Value on event action handler failure

EventActionHandler:Float node

Used for the entity event action handler.



Inputs

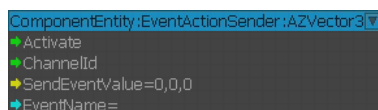
Port		Description
ChannelId	Any	Entity channel ID
EventName	String	Name of the event action handler
Enable	Any	Enables the event action handler
Disable	Any	Disables the event action handler

Outputs

Port		Description
Success	Float	Float value on event action handler success
Failure	Float	Float value on event action handler failure

EventActionSender:AZVector3 node

Used for the entity event action sender.



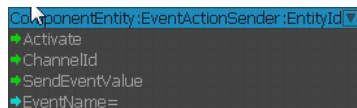
Inputs

Port		Description
Activate	Any	Activates the node

Port		Description
ChannelID	Any	ID of the channel for the event action sender
SendEventValue	Vec3	Vector value for the event action sender
Eventname	String	Name of the event action sender

EventActionSender:EntityID node

Used for the entity event action sender.

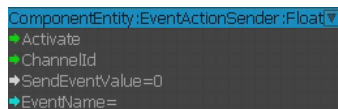


Inputs

Port		Description
Activate	Any	Activates the node
ChannelID	Any	ID of the channel for the event action sender
SendEventValue	Any	Value for the event action sender
Eventname	String	Name of the event action sender

EventActionSender:Float node

Used for the entity event action sender.

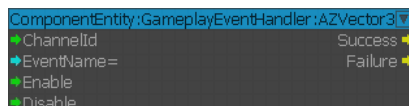


Inputs

Port		Description
Activate	Any	Activates the node
ChannelID	Any	ID of the channel for the event action sender
SendEventValue	Float	Float value for the event action sender
Eventname	String	Name of the event action sender

GameplayEventHandler:AZVector3 node

Used for the gameplay event handler.



Inputs

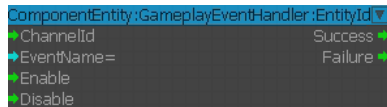
Port		Description
ChannelId	Any	Entity channel ID
EventName	String	Name of the gameplay event handler
Enable	Any	Enables the gameplay event handler
Disable	Any	Disables the gameplay event handler

Outputs

Port		Description
Success	Vec3	Vector value on event gameplay event handler success
Failure	Vec3	Vector value on gameplay event handler failure

GameplayEventHandler:EntityID node

Used for the gameplay event handler.



Inputs

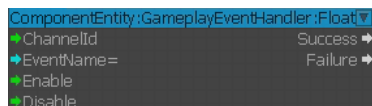
Port		Description
ChannelId	Any	Entity channel ID
EventName	String	Name of the gameplay event handler
Enable	Any	Enables the gameplay event handler
Disable	Any	Disables the gameplay event handler

Outputs

Port		Description
Success	Any	Value on gameplay event handler success
Failure	Any	Value on event gameplay event handler failure

GameplayEventHandler:Float node

Used for the gameplay event handler.



Inputs

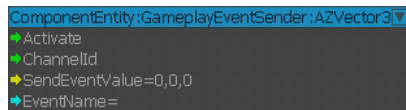
Port		Description
ChannelId	Any	Entity channel ID
EventName	String	Name of the gameplay event handler
Enable	Any	Enables the gameplay event handler
Disable	Any	Disables the gameplay event handler

Outputs

Port		Description
Success	Float	Float value on event action handler success
Failure	Float	Float value on event action handler failure

GameplayEventSender:AZVector3 node

Used for the gameplay event sender.

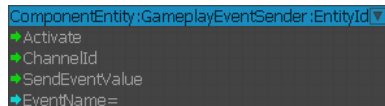


Inputs

Port		Description
Activate	Any	Activates the node
ChannelID	Any	ID of the channel for the gameplay event sender
SendEventValue	Vec3	Vector value for the gameplay event sender
Eventname	String	Name of the gameplay event sender

GameplayEventSender:EntityID node

Used for the gameplay event sender.



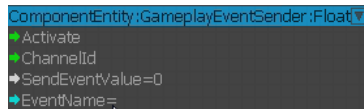
Inputs

Port		Description
Activate	Any	Activates the node

Port		Description
ChannelID	Any	ID of the channel for the gameplay event sender
SendEventValue	Any	Value for the gameplay event sender
Eventname	String	Name of the gameplay event sender

GameplayEventSender:Float node

Used for the gameplay event sender.



Inputs

Port		Description
Activate	Any	Activates the node
ChannelID	Any	ID of the channel for the gameplay event sender
SendEventValue	Float	Float value for the gameplay event sender
Eventname	String	Name of the gameplay event sender

Light:Switch node

Used to turn the light entity on or off.



Inputs

Port		Description
On	Any	Turns the light on
Off	Any	Turns the light off

Particles:Switch node

Used to show or hide the particle entity.

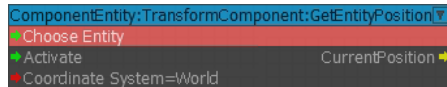


Inputs

Port		Description
Show	Any	Displays the particle
Hide	Any	Hides the particle

TransformComponent:GetEntityPosition node

Used to get the entity position.



Inputs

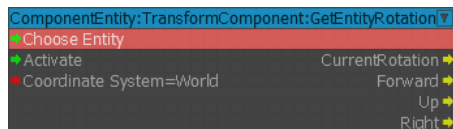
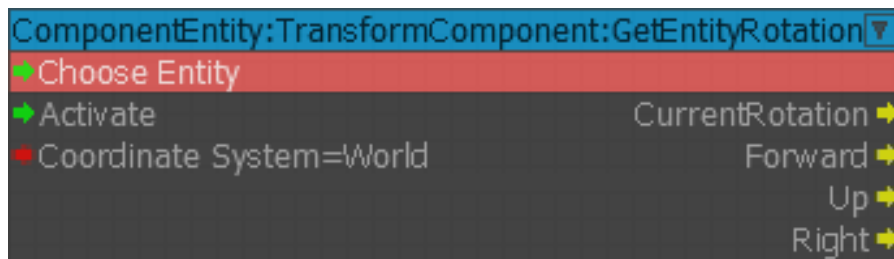
Port		Description
Activate	Any	Activates the node
Coordinate System	Integer	Coordinate system used

Outputs

Port		Description
CurrentPosition	Vec3	Outputs the current entity position

TransformComponent:GetEntityRotation node

Used to get the entity rotation.



Inputs

Port		Description
Activate	Any	Activates the node

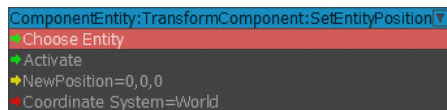
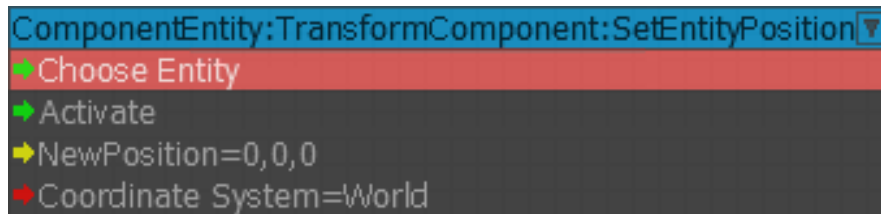
Port		Description
Coordinate System	Integer	Coordinate system used

Outputs

Port		Description
CurrentRotation	Vec3	Current entity rotation
Forward	Vec3	Entity forward position
Up	Vec3	Entity up position
Right	Vec3	Entity right position

TransformComponent:SetEntityPosition node

Used to

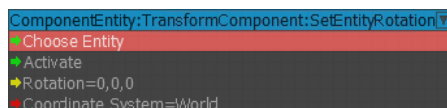


Inputs

Port		Description
Activate	Any	Activates the node
NewPosition	Vec3	Position to be set
Coordinate System	Integer	Coordinate system used

TransformComponent:SetEntityRotation node

Used to set entity rotation.



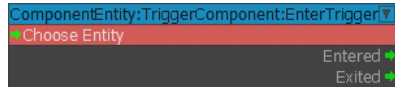
Inputs

Port		Description
Activate	Any	Activates the node

Port		Description
Rotation	Vec3	Rotation to be set
Coordinate System	Integer	Coordinate system used

TriggerComponent:EnterTrigger node

Used to trigger when the entity enters or leaves the trigger area.



Outputs

Port		Description
Entered	Any	Triggers when entity enters the trigger area
Exited	Any	Triggers when entity leaves the trigger area

CustomAction Nodes

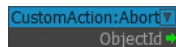
You can use the following flow graph nodes to control custom actions that entities take.

Topics

- [Abort node \(p. 557\)](#)
- [Control node \(p. 558\)](#)
- [End node \(p. 559\)](#)
- [Start node \(p. 559\)](#)
- [Succeed node \(p. 559\)](#)
- [SucceedWait node \(p. 560\)](#)
- [SucceedWaitComplete node \(p. 560\)](#)

Abort node

Used to start the abort path of a custom action.

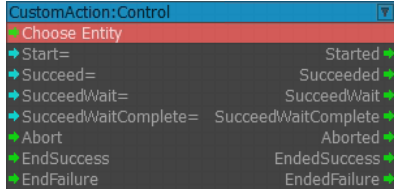


Outputs

Port	Type	Description
Objectid	Any	Entity ID of the object on which the custom action is executing on

Control node

Used to control a custom action instance.



Inputs

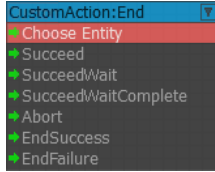
Port	Type	Description
Start	String	Entity is entering the start path
Succeed	String	Entity is entering the succeed path
SucceedWait	String	Entity is entering the succeed wait path
SucceedWaitComplete	String	Entity is entering the succeed wait complete path
Abort	Any	Entity is entering the abort path
EndSuccess	Any	Entity is entering the end succeed path
EndFailure	Any	Entity is entering the end failure path

Outputs

Port	Type	Description
Started	Any	Entity has entered the start path
Succeeded	Any	Entity has entered the succeed path
SucceedWait	Any	Entity has entered the succeed wait path
SucceedWaitComplete	Any	Entity has entered the succeed wait completed
Aborted	Any	Entity has entered the abort path
EndedSuccess	Any	Entity has entered the end success path
EndedFailure	Any	Entity has entered the end failure path

End node

Used to end a custom action.

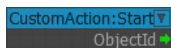


Inputs

Port	Type	Description
Succeed	Any	Entity has entered the succeed path
SucceedWait	Any	Entity has entered the succeed wait path
SucceedWaitComplete	Any	Entity has entered the succeed wait complete path
Abort	Any	Entity has entered the abort path
EndSuccess	Any	Entity has entered the end succeed path
EndFailure	Any	Entity has entered the end failure path

Start node

Used to start a custom action.

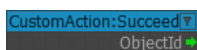


Outputs

Port	Type	Description
ObjectID	Any	Entity ID of the object on which the custom action is executing on

Succeed node

Used to indicate a custom action succeeded.

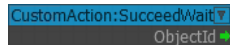


Outputs

Port	Type	Description
ObjectId	Any	Entity ID of the object on which the custom action is executing on

SucceedWait node

Used to indicate that a custom action wait succeeded.

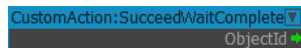


Outputs

Port	Type	Description
ObjectId	Any	Entity ID of the object on which the custom action is executing on

SucceedWaitComplete node

Used to indicate that a custom action wait succeeded and completed.



Outputs

Port	Type	Description
ObjectId	Any	Entity ID of the object on which the custom action is executing on

Debug Nodes

You can use the following flow graph nodes to configure various settings used for debugging purposes.

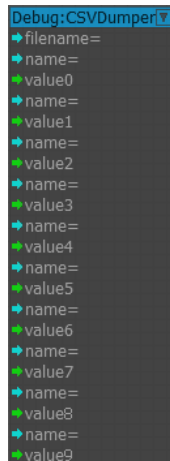
Topics

- [CSVDumper node \(p. 561\)](#)
- [ConsoleVariable node \(p. 561\)](#)
- [DisplayMessage node \(p. 562\)](#)
- [Draw2d nodes \(p. 562\)](#)
- [Draw nodes \(p. 565\)](#)
- [ExecuteString node \(p. 569\)](#)
- [FloatToString node \(p. 570\)](#)

- [Frame node \(p. 570\)](#)
- [FrameExtended node \(p. 571\)](#)
- [InputKey node \(p. 571\)](#)
- [Log node \(p. 572\)](#)
- [Memory node \(p. 572\)](#)

CSVDumper node

Used to store the cell values of the specified `.csv` file.

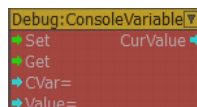


Inputs

Port	Type	Description
filename	String	CSV file to use
name	String	column/row name
value0 - value9	Any	cell values

ConsoleVariable node

Used to set and get the value of a console variable.



Inputs

Port	Type	Description
Set	Any	Set console variable value
Get	Any	Get console variable value
CVar	String	Name of console variable

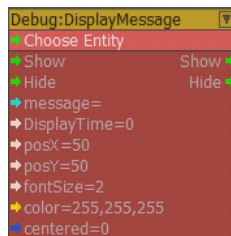
Port	Type	Description
Value	String	Value of console variable to set

Outputs

Port	Type	Description
CurValue		Current value of the console variable

DisplayMessage node

If an entity is not provided, the local player will be used instead.



Inputs

Port	Type	Description
Show	Any	Show message
Hide	Any	Hide message
message	String	Message to display on the HUD
DisplayTime	FloatFloat	Duration that the message will be visible for
posx	Float	Input x text position
posy	Float	Input y text position
fontSize	Float	Input font size
color	Vec3	Color of the message text
centered	Boolean	Centers the text around the coordinates

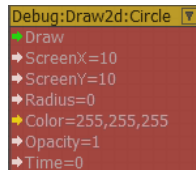
Outputs

Port	Type	Description
Show	Any	Displays the message
Hide	Any	Hides the message

Draw2d nodes

Draw2d:Circle node

Used to draw a circle.



Inputs

Port	Type	Description
Draw	Any	Draws a 2D circle
ScreenX	Float	X-axis position of the center of the circle
ScreenY	Float	Y-axis position of the center of the circle
Radius	Float	Radius of the circle
Color	Vec3	Color of the circle
Opacity	Float	Transparency of the circle
Time	Float	Number of seconds the circle will be visible for

Draw2d:Line node

Used to draw a line.



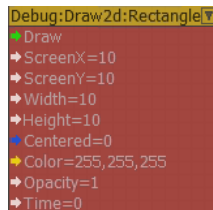
Inputs

Port	Type	Description
Draw	Any	Draws a line
StartX	Float	X-axis starting point of the line
StartY	Float	Y-axis starting point of the line
EndX	Float	X-axis ending point of the line
EndY	Float	Y-axis ending point of the line
Color	Vec3	Color of the line
Opacity	Float	Transparency of the line

Port	Type	Description
Time	Float	Number of seconds the line will be visible for

Draw2d:Rectangle node

Used to draw a rectangle.

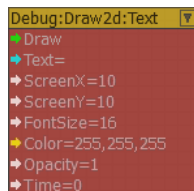


Inputs

Port	Type	Description
Draw	Any	Draws a rectangle
ScreenX	Float	X-axis position of the center of the rectangle
ScreenY	Float	X-axis position of the center of the rectangle
Width	Float	Width of the rectangle
Height	Float	Height of the rectangle
Centered	Boolean	Rectangle centered at ScreenX and ScreenY
Color	Vec3	Color of the rectangle
Opacity	Float	Transparency of the rectangle
Time	Float	Number of seconds the rectangle will be visible for

Draw2d:Text node

Used to output a text message.



Inputs

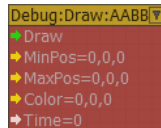
Port	Type	Description
Draw	Any	Displays text
Text	String	Text to display

Port	Type	Description
ScreenX	Float	X-axis position of the text
ScreenY	Float	Y-axis position of the text
FontSize	Float	Text message font size
Color	Vec3	Color of the text
Opacity	Float	Transparency of the text
Time	Float	Number of seconds the text will be visible for

Draw nodes

Draw:AABB node

Used to draw an AABB bounding box.

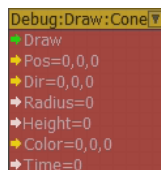


Inputs

Port	Type	Description
Draw	Any	Draws an AABB bounding box
MinPos	Vec3	Minimum position of the bounding box
MaxPos	Vec3	Maximum position of the bounding box
Color	Vec3	Color of the bounding box
Time	Float	Number of seconds the bounding box will be visible for

Draw:Cone node

Used to draw a cone.



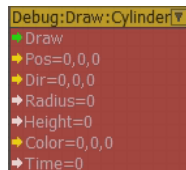
Inputs

Port	Type	Description
Draw	Any	Draws a cone

Port	Type	Description
Pos	Vec3	Position of the cone
Dir	Vec3	Direction of the cone axis
Radius	Float	Radius of the cone base
Height	Float	Height of the cone
Color	Vec3	Color of the cone
Time	Float	Number of seconds the cone will be visible for

Draw:Cylinder node

Used to draw a cylinder.

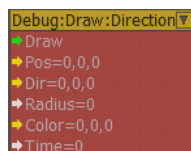


Inputs

Port	Type	Description
Draw	Any	Draws a cylinder
Pos	Vec3	Position of the cylinder
Dir	Vec3	Direction of the cylinder axis
Radius	Float	Radius of the cylinder
Height	Float	Height of the cylinder
Color	Vec3	Color of the cylinder
Time	Float	Number of seconds the cylinder will be visible for

Draw:Direction node

Used to draw an arrow.



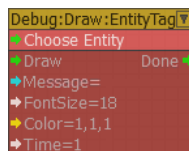
Inputs

Port	Type	Description
Draw	Any	Draws an arrow

Port	Type	Description
Pos	Vec3	Position of the arrow
Dir	Vec3	Direction the arrow is pointing
Radius	Float	Radius of the arrow head
Color	Vec3	Color of the arrow
Time	Float	Number of seconds the arrow will be visible for

Draw:EntityTag node

Used to draw a text message above an entity.



Inputs

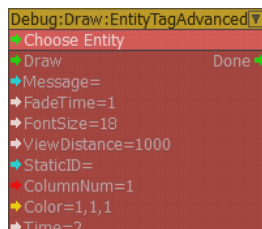
Port	Type	Description
Draw	Any	Displays a text message above an entity
Message	String	Text message
FontSize	Float	Text message font size
Color	Vec3	Text message color
Time	Float	Number of seconds the message will be visible for

Outputs

Port	Type	Description
Done		Triggers when the text message is no longer visible

Draw:EntityTagAdvanced node

Used to draw a text message above an entity.



Inputs

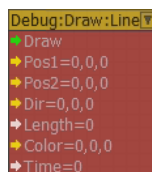
Port	Type	Description
Draw	Any	Displays a text message above an entity
Message	String	Message to be displayed
FadeTime	Float	Number of seconds for text message to fade out
FontSize	Float	Font size of the text message
ViewDistance	Float	Distance from camera the entity must be within for message to be displayed
StaticID	String	Static tag ID
ColumnNum	Integer	Which column above an entity the message will be displayed in
Color	Vec3	Color of the text message
Time	Float	Number of seconds the text message will be visible for

Outputs

Port	Type	Description
Done	Any	Triggers when the text message is no longer visible

Draw:Line node

Used to draw a line.



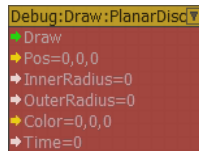
Inputs

Port	Type	Description
Draw	Any	Draws a line in 3D space
Pos1	Vec3	Starting point of the line
Pos2	Vec3	Ending point of the line
Dir	Vec3	Direction of the line
Length	Float	Length of the line
Color	Vec3	Color of the line

Port	Type	Description
Time	Float	Number of seconds the circle will be visible for

Draw:PlanarDisc node

Used to draw a disc.

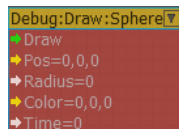


Inputs

Port	Type	Description
Draw	Any	Draws a disc
Pos	Vec3	Position of the disc center
InnerRadius	Float	Inner radius of the disc
OuterRadius	Float	Outer radius of the disc
Color	Vec3	Color of the disc
Time	Float	Number of seconds the circle will be visible for

Draw:Sphere node

Used to draw a sphere.

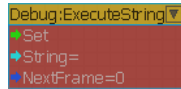


Inputs

Port	Type	Description
Draw	Any	Draws a sphere
Pos	Vec3	Position of the sphere center
Radius	Float	Radius of the sphere
Color	Vec3	Color of the sphere
Time	Float	Number of seconds the circle will be visible for

ExecuteString node

Used to execute a string when using the console.

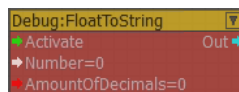


Inputs

Port	Type	Description
Set	Any	Executes the string
String	String	String to be executed
NextFrame	Boolean	String will be executed next frame

FloatToString node

Used to output a float value in string format with a limited number of decimals.



Inputs

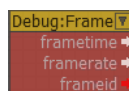
Port	Type	Description
Activate	Any	Activates the node
Number	Float	Floating point number to convert
AmountOfDecimals	Integer	Number of decimal places for the floating point

Outputs

Port	Type	Description
Out	String	Outputs a string representation of the floating point input

Frame node

Used to output the current frame rate data.

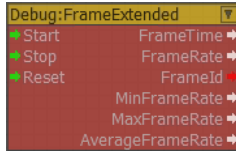


Outputs

Port	Type	Description
frametime	Float	Current frame time
framerate	Float	Current frame rate
frameid	Integer	Frame ID

FrameExtended node

Used to output extended current frame rate data.



Inputs

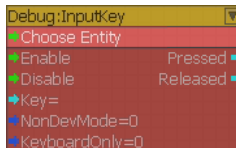
Port	Type	Description
Start	Any	Start collecting frame rate data
Stop	Any	Stop collecting frame rate data
Reset	Any	Resets the data

Outputs

Port	Type	Description
FrameTime	Float	Current frame time
FrameRate	Float	Current frame rate
FrameId	Integer	Frame ID
MinFrameRate	Float	Minimum frame rate
MaxFrameRate	Float	Maximum frame rate
AverageFrameRate	Float	Average frame rate

InputKey node

Used to catch key inputs. The Entity input is required for multiplayer games.



Inputs

Port	Type	Description
Enable	Any	Activates the node
Disable	Any	Deactivates the node
Key	String	Key name
NonDevMode	String	Can be used in non-dev mode if set to true

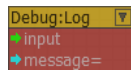
Port	Type	Description
Keyboard only	Boolean	Ignores non-keyword data if set to true

Outputs

Port	Type	Description
Pressed	String	Triggers when a key is pressed
Released	String	Triggers when a key is released

Log node

Used to log string input messages to the console.

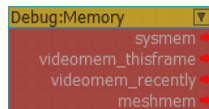


Inputs

Port	Type	Description
input	Any	Activates the node
message	String	Message to be logged

Memory node

Used to display video memory data.



Inputs

Port	Type	Description
system	Integer	Outputs system video memory data
videomem_thisframe	Integer	Outputs video memory used for current frame
videomem_recently	Integer	Outputs video memory recently used
meshmem	Integer	Outputs memory used for the mesh object

Dialog Nodes

You can use the following flow graph nodes to configure and control actor dialogs.

Note

These nodes will only work with the Legacy Game Sample (CryEngine GameSDK), which is available at [Lumberyard Downloads](#).

Topics

- [PlayDialog node \(p. 573\)](#)

PlayDialog node

Used to play a dialog.



Inputs

Port	Type	Description
Play	Any	Plays the dialog
Stop	Any	Stops the dialog
Dialog	String	Name of the dialog to play
StartLine	Integer	Line to start the dialog from
AllInterrupt	Integer	AI interrupt behavior; values are Never, Alert, and Combat
AwareDistance	Float	Distance that player is considered as listening at
AwareAngle	Float	View angle that player is considered as listening at
AwareTimeout	Float	Time out until non-aware player aborts dialog
Flags	Integer	Dialog playback flags
Buffer	String	Stores the dialog. Only one dialog can be played at any time in each buffer
BufferDisplay	Float	How many more seconds the dialog will wait until the previous dialog in its dialog has finished
Actor 1-8	Any	Actor entity IDs

Outputs

Port	Type	Description
Started	Any	Triggered when the dialog has started
Done	Any	Triggered when the dialog has finished or aborted
Finished	Any	Triggered when the dialog has finished
Aborted	Any	Triggered when the dialog has aborted
PlayerAbort	Integer	Triggered when the dialog has aborted because the player is out of range or out of view
AIAbort	Any	Triggered when the dialog has aborted because the AI got alerted
ActorDied	Any	Triggered when the dialog has aborted because the Actor died
LastLine	Integer	Last line played when the dialog was aborted
CurLine	Integer	Current line; triggered whenever a line starts

Dynamic Response Nodes

You can use the following flow graph nodes to configure settings for the Dynamic Response system.

Note

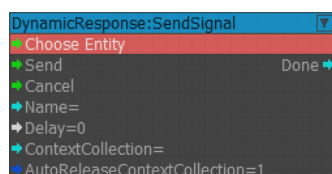
These nodes will only work with the Legacy Game Sample (CryEngine GameSDK), which is available at [Lumberyard Downloads](#).

Topics

- [SendSignal node \(p. 574\)](#)
- [SetFloatVariable node \(p. 575\)](#)
- [SetIntegerVariable node \(p. 576\)](#)
- [SetStringVariable node \(p. 577\)](#)

SendSignal node

Used to send a signal to the Dynamic Response system.



Inputs

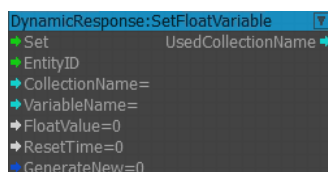
Port	Type	Description
Send	Any	Sends the dynamic response signal
Cancel	Any	Cancels the dynamic response signal
Name	String	Name of the dynamic response signal
Delay	Float	Delays the sending of the dynamic response signal
ContextCollection	String	Name of the variable collection sent along with the signal as a context
AutoReleaseContextCollection	Boolean	Controls whether the variable collection is released after processing the signal.

Outputs

Port	Type	Description
Done	String	Triggered when the signal is sent or is cancelled.

SetFloatVariable node

Used to set a float variable in the Dynamic Response system.



Inputs

Port	Type	Description
Set	Any	Set the given value to the specified variable
EntityID	Any	The ID of the entity to fetch the collection from
CollectionName	String	The name of the collection
VariableName	String	The name of the variable to set
FloatValue	Float	The value of the variable

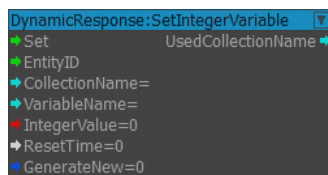
Port	Type	Description
ResetTime	Float	The time after which the variable is reset to its previous value
GenerateNew	Boolean	Determines whether a new variable collection should be generated

Outputs

Port	Type	Description
UsedCollectionName	String	Outputs the name of the variable collection created or used

SetIntegerVariable node

Used to set a float variable in the Dynamic Response system.



Inputs

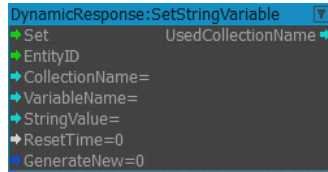
Port	Type	Description
Set	Any	Set the given value to the specified variable
EntityID	Voif	The ID of the entity to fetch the collection from
CollectionName	String	The name of the collection
VariableName	String	The name of the variable to set
FloatValue	Float	The value of the variable
ResetTime	Float	The time after which the variable is reset to its previous value
GenerateNew	Boolean	Determines whether a new variable collection should be generated

Outputs

Port	Type	Description
UsedCollectionName	String	Outputs the name of the variable collection created or used

SetStringVariable node

Used to set a string variable in the Dynamic Response system



Inputs

Port	Type	Description
Set	Any	Set the given value to the specified variable
EntityID	Any	The ID of the entity to fetch the collection from
CollectionName	String	The name of the collection
VariableName	String	The name of the variable to set
FloatValue	String	The value of the variable
ResetTime	Float	The time after which the variable is reset to its previous value
GenerateNew	Boolean	Determines whether a new variable collection should be generated

Outputs

Port	Type	Description
UsedCollectionName	String	Outputs the name of the variable collection created or used

Engine Nodes

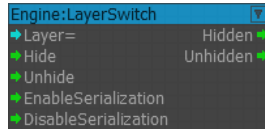
You can use the following flow graph nodes to configure various Lumberyard engine settings.

Topics

- [LayerSwitch node \(p. 577\)](#)
- [PortalSwitch node \(p. 578\)](#)
- [PrecacheArea node \(p. 578\)](#)
- [Viewport node \(p. 579\)](#)

LayerSwitch node

Used to activate and deactivate objects in a layer, as well as streaming in data to a layer.



Inputs

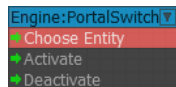
Port	Type	Description
Layer	String	Name of the layer
Hide	Any	Hides objects in the layer
Unhide	Any	Shows objects in the layer
EnableSerialization	Any	Enables objects in the layer
DisableSerialization	Any	Disables objects in the layer

Outputs

Port	Type	Description
Hidden	Any	Triggered if hidden
Unhidden	Any	Triggered if visible

PortalSwitch node

Used to switch the portal on or off.

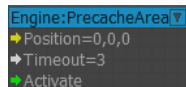


Inputs

Port	Type	Description
Activate	Any	Activates the portal switch
Deactivate	Any	Deactivates the portal switch

PrecacheArea node

Used to precache an area at a specified location



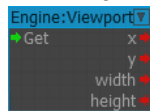
Inputs

Port	Type	Description
Position	Vec3	Location of the area to be precached

Port	Type	Description
Timeout	Float	Timeout interval in seconds
Activate	Any	Activates the node

Viewport node

Used to get current viewport information.



Inputs

Port	Type	Description
Get	Any	Gets the current viewport information

Outputs

Port	Type	Description
x	Integer	Outputs the top left X position of the viewport
y	Integer	Outputs the top left Y position of the viewport
width	Integer	Outputs the width of the viewport
height	Integer	Outputs the height of the viewport

Entity Nodes

You can use the following flow graph nodes to control entity behavior and configure related settings.

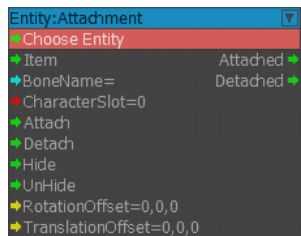
Topics

- [Attachment node \(p. 580\)](#)
- [BeamEntity node \(p. 581\)](#)
- [BroadcastEvent node \(p. 581\)](#)
- [CallScriptFunction node \(p. 582\)](#)
- [CharAttachmentMaterialParam node \(p. 582\)](#)
- [CheckDistance node \(p. 583\)](#)
- [ChildAttach node \(p. 584\)](#)
- [ChildDetach node \(p. 584\)](#)
- [Damage node \(p. 585\)](#)
- [EntitiesInRange node \(p. 585\)](#)
- [EntityId node \(p. 586\)](#)

- [EntityInfo node \(p. 586\)](#)
- [EntityPool node \(p. 587\)](#)
- [EntityPos node \(p. 587\)](#)
- [FindEntityByName node \(p. 588\)](#)
- [GetBounds node \(p. 588\)](#)
- [GetEntityExistence node \(p. 589\)](#)
- [GetPos node \(p. 589\)](#)
- [ParentId node \(p. 590\)](#)
- [PropertyGet node \(p. 590\)](#)
- [PropertySet node \(p. 591\)](#)
- [RemoveEntity node \(p. 591\)](#)
- [RenderParams node \(p. 592\)](#)
- [Spawn node \(p. 592\)](#)
- [SpawnArchetype node \(p. 593\)](#)

Attachment node

Used to attach and detach attachments to an entity.



Inputs

Port	Type	Description
Item	Any	Entity to be linked
BoneName	String	Attachment bone
CharacterSlot	Integer	Host character slot
Attach	Any	Attach entity attached to the bone
Detach	Any	Detach entity attached to bone
Hide	Any	Hide attachment
Unhide	Any	Show attachment
RotationOffset	Vec3	Rotation offset
TranslationOffset	Vec3	Translation offset

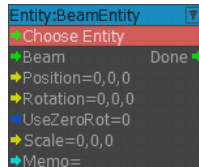
Outputs

Port	Type	Description
Attached	Any	Triggers when entity is attached

Port	Type	Description
Detached	Any	Triggers when entity is detached

BeamEntity node

Used to beam or teleport objects instantly to any position in the level. When the **Beam** port is triggered the target entity is moved to the position input on the **Position** port.



Inputs

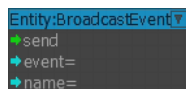
Port	Type	Description
Beam	Any	Trigger to beam the entity
Position	Vec3	Destination location to beam to
Rotation	Vec3	Rotation to apply to entity
UseZeroRot	Boolean	Applies rotation even if it is 0
Scale	Vec3	Vector scale value
Memo	String	Memo to log when position is 0

Outputs

Port	Type	Description
Done	Any	Triggers when entity has beamed to another location

BroadcastEvent node

Used to send an event to one or more entities. The entities that will receive this event are specified by inputting a string to the **name** port. Each entity that has the string that is input there as a part of their name will receive the event set in the **event** port.



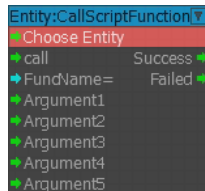
Inputs

Port	Type	Description
send	Any	Trigger to send an event

Port	Type	Description
event	String	Event to be sent
name	String	Entity to receive the event

CallScriptFunction node

Used to call a script function for the entity.



Inputs

Port	Type	Description
call	Any	Calls the function
FunctionName	String	Script function name
Argument1 - Argument5	Any	Function arguments

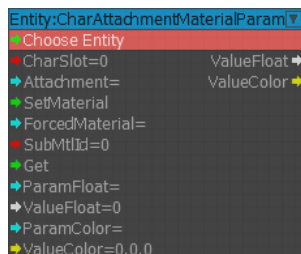
Outputs

Port	Type	Description
Success	Any	Script function was found and called
Failed	Any	Script function was not found or called

CharAttachmentMaterialParam node

Used to change a material on an attachment in a .cdf file. For example, you can change the material of a character's trousers.

SetMaterial is the trigger, **ForcedMaterial** is the full file path to the material (for example: `materials/references/basecolors/grey.mtl`) and **SubMtlId** is the number of the sub-material.



Inputs

Port	Type	Description
CharSlot	Integer	Character slot within the entity
Attachment	String	Attachment
SetMaterial	Any	Sets the material
ForcedMaterial	String	Forcefully set the material
SubMtlId	Integer	Submaterial ID
Get	Any	Trigger to get current value
ParamFloat	String	Float parameter to get or be set
ValueFloat	Float	Trigger to set value
ParamColor	String	Color parameter to get or be set
ValueColor	Vec3	Sets value color

Outputs

Port	Type	Description
ValueFloat	Float	Current floating point value
ValueColor	Vec3	Current color value

CheckDistance node

Used to check the distance between the node entity and the entities defined in the input ports.



Inputs

Port	Type	Description
Check	Any	Trigger to check distance

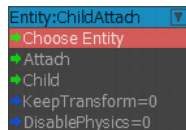
Port	Type	Description
MinDistance	Float	An entity that is nearer this distance will be ignored
MaxDistance	Float	An entity that is further than this distance will be ignored
Entity1 - Entity16	Any	Entity ID values

Outputs

Port	Type	Description
NearEntity	Any	Nearest entity
NearEntityDist	Float	Distance of nearest entity
FarEntity	Any	Furthest entity
FarEntityDist	Float	Distance of furthest entity
NoEntInRange	Any	Triggers when no entities are between MinDistance and MaxDistance

ChildAttach node

Used to attach another entity to its target entity. The child entity will be linked to the target entity until the link is removed. The entity defined in the Child input port is attached to the target entity.



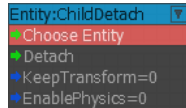
Inputs

Port	Type	Description
Attach	Any	Triggers entity attachment
Child	Any	Child entity to be attached
KeepTransform	Boolean	Child entity will be kept at the same transformation in world space
DisablePhysics	Boolean	Disable physics for child entity when attached

ChildDetach node

Used to detach entities from its parent entity. Usually the **ChildAttach** node has been used before to link the target entity to another entity.

When **KeepTransform** is set, the entity will keep its transformation in world space when detached. When **EnablePhysics** is set, physics will be re-enabled again when the entity is detached.

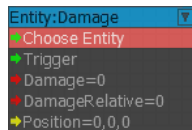


Inputs

Port	Type	Description
Detach	Any	Triggers entity detachment
KeepTransform	Boolean	Child entity will be kept at the same transformation in world space
EnablePhysics	Boolean	Enable physics for child entity when detached

Damage node

Used to damage the specified entity when the trigger is activated.

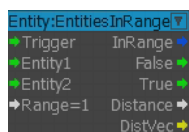


Inputs

Port	Type	Description
Trigger	Any	Triggers the node
Damage	Integer	Amount of damage to inflict
DamageRelative	Integer	Damage inflicted is relative to the health of the entity
Position	Vec3	Location damage occurs at

EntitiesInRange node

Used to take the positions of two entities and check if they are in a certain range to each other. Depending on the result of the check the output ports are triggered.



Inputs

Port	Type	Description
Trigger	Any	Triggers the node
Entity1	Any	Entity 1

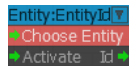
Port	Type	Description
Entity2	Any	Entity 2
Range	Float	Distance range to check

Outputs

Port	Type	Description
InRange	Boolean	True if entities are in range of each other
False	Any	Triggers if entites are not in range
True	Any	Triggers if entities are in range
Distance	Float	Floating point distance between the two entities
DistVec	Vec3	Vector distance between the two entities

EntityId node

Used to output the entity ID number of the specified entity. The node does not need to be triggered as the entity ID never changes.



Inputs

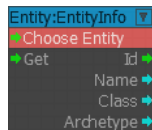
Port	Type	Description
Activate	Any	Entity ID

Outputs

Port	Type	Description
Id	Any	Outputs the entity ID

EntityInfo node

Used to output the ID, name, class, and archetype of the target entity.



Inputs

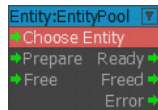
Port	Type	Description
Get	Any	Gets entity information

Outputs

Port	Type	Description
Id	Any	Entity ID
Name	String	Entity name
Class	String	Entity class
Archetype	String	Entity archetype

EntityPool node

Used to prepare an entity from the pool or free it back to the pool.



Inputs

Port	Type	Description
Prepare	Any	Brings entity into existence from the pool
Free	Any	Frees the entity back to the pool

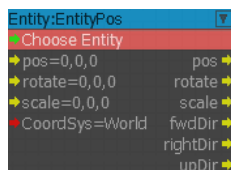
Outputs

Port	Type	Description
Ready	Any	Triggers when the entity is prepared and ready
Freed	Any	Triggers when the entity is freed and returns to the pool
Error	Any	Triggers when an error occurs

EntityPos node

Handles all position related manipulations of the owner entity. All position information of the specified entity can be read from the output ports.

Unlike the **GetPos** node, the output ports of this node are triggered whenever one of the target entity properties changes.



Inputs

Port	Type	Description
Pos	Vec3	Entity position
Rotate	Vec3	Entity rotation angle in degrees
Scale	Vec3	Entity scale
CoordSys	Integer	Coordinate system used

Outputs

Port	Type	Description
Pos	Vec3	Current entity position
Rotate	Vec3	Current entity rotation angle in degrees
Scale	Vec3	Current entity scale
FwdDir	Vec3	Current entity Y-axis position
RightDir	Vec3	Current entity X-axis position
UpDir	Vec3	Current entity Z-axis position

FindEntityByName node

Used to find an entity by name and output the entity ID.



Inputs

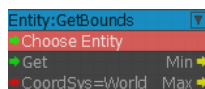
Port	Type	Description
Set	Any	Start searching for entity
Name	String	Name of entity to look for

Outputs

Port	Type	Description
EntityId	Any	Outputs the entity ID if found

GetBounds node

Used to get and output the bounds.



Inputs

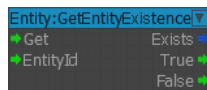
Port	Type	Description
Get	Any	Gets the AABB bounding box
CoordSys	Integer	Coordinate system used

Outputs

Port	Type	Description
Min	Vec3	Minimum position of the AABB
Max	Vec3	Maximum position of the AABB

GetEntityExistence node

Used to get an entity's existence.



Inputs

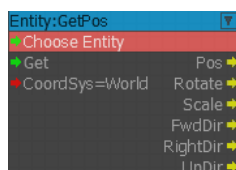
Port	Type	Description
Get	Any	Gets entity existence status
EntityId	Any	Entity ID

Outputs

Port	Type	Description
Exists	Boolean	True if the entity exists
True	Any	Triggers if the entity exists
False	Any	Triggers if the entity exists

GetPos node

Used to output position information only when the trigger is activated. Similar to the **EntityPos** node, which triggers the output ports continuously whenever any position information changes.



Inputs

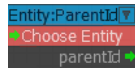
Port	Type	Description
Get	Any	Gets entity position
CoordSys	Integer	Coordinate system used

Outputs

Port	Type	Description
Pos	Vec3	Entity position
Rotate	Vec3	Entity rotation
Scale	Vec3	Entity scale
FwdDir	Vec3	Entity Y-axis position
RightDir	Vec3	Entity X-axis position
UpDir	Vec3	Entity Z-axis position

ParentId node

Used to obtain the parentID number of the specified entity.

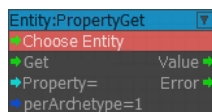


Outputs

Port	Type	Description
Parent Id	Any	Parent entity ID

PropertyGet node

Used to retrieve an entity property value.



Inputs

Port	Type	Description
Get	Any	Trigger to get entity property value
Property	String	Name of property to get

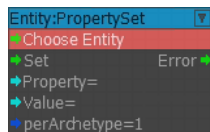
Port	Type	Description
PerArchetype	Boolean	True if a per archetype property; false if a per instance property

Outputs

Port	Type	Description
Value	Any	Outputs property value
Error	Any	Retrieves property value

PropertySet node

Used to change the entity property value. Will not work with SaveLoad however.



Inputs

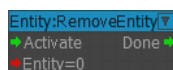
Port	Type	Description
Set	Any	Sets property value
Property	String	Name of property to set
Value	String	Property value to be set
PerArchetype	Boolean	True if a per archetype property; false if a per instance property

Outputs

Port	Type	Description
Error	Any	Any

RemoveEntity node

Used to remove an entity.



Inputs

Port	Type	Description
Activate	Any	Triggers the node

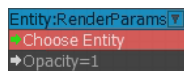
Port	Type	Description
Entity	Integer	Entity to remove

Outputs

Port	Type	Description
Done	Any	Triggers when entity has been removed

RenderParams node

Used to set rendering parameters.

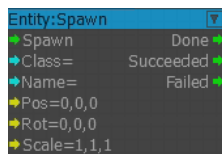


Inputs

Port	Type	Description
Opacity	Float	Sets entity transparency value

Spawn node

Used to spawn an entity with the specified properties.



Inputs

Port	Type	Description
Spawn	Any	Spawns an entity
Class	String	Entity class
Name	String	Entity class
Pos	Vec3	Entity position
Rot	Vec3	Entity rotation
Scale	Vec3	Entity scale

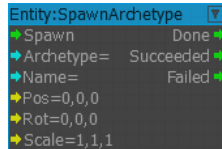
Outputs

Port	Type	Description
Done	Any	Triggers when entity has completed spawning

Port	Type	Description
Succeeded	Any	Triggers when entity is spawned
Failed	Any	Triggers if entity was not spawned

SpawnArchetype node

Used to spawn an archetype entity with the specified properties.



Inputs

Port	Type	Description
Spawn	Any	Spawns an entity
Archetype	String	Archetype entity name
Name	String	Entity name
Pos	Vec3	Entity position
Rot	Vec3	Entity rotation angle
Scale	Vec3	Entity scale

Outputs

Port	Type	Description
Done	Any	Triggers when entity has completed spawning
Succeeded	Any	Triggers when entity is spawned
Failed	Any	Triggers if entity was not spawned

Environment Nodes

You can use the following flow graph nodes to configure environment settings.

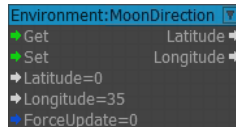
Topics

- [MoonDirection node \(p. 594\)](#)
- [OceanSwitch node \(p. 594\)](#)
- [PerEntityShadows node \(p. 594\)](#)
- [RainProperties node \(p. 595\)](#)
- [RecomputeStaticShadows node \(p. 595\)](#)
- [SetOceanMaterial node \(p. 596\)](#)

- [SkyMaterialSwitch node \(p. 596\)](#)
- [SkyboxSwitch node \(p. 597\)](#)
- [Sun node \(p. 597\)](#)
- [TornadoWander \(p. 598\)](#)
- [Wind node \(p. 598\)](#)

MoonDirection node

Used to set the moon's position in the sky.



Inputs

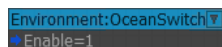
Port	Type	Description
Get	Any	Get current latitude and longitude
Set	Any	Set latitude and longitude
Latitude	Float	Latitude to be set
Longitude	Float	Longitude to be set
ForceUpdate	Boolean	Force immediate update of the sky

Outputs

Port	Type	Description
Latitude	Float	Output current latitude
Longitude	Float	Output current longitude

OceanSwitch node

Used to enable ocean rendering.

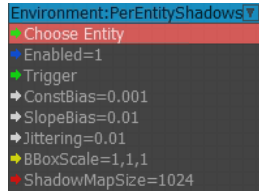


Inputs

Port	Type	Description
Enable	Boolean	Enable ocean rendering

PerEntityShadows node

Used to enable and specify per entity shadows.

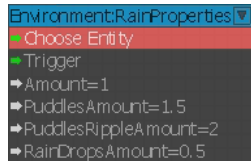


Inputs

Port	Type	Description
Enabled	Boolean	Activates the node
Trigger	Any	Triggers the parameters
ConstBias	Float	Reduces aAny self-shadowing artifacts
SlopeBias	Float	Reduces aAny self-shadowing artifacts
Jittering	Float	Filters kernel size, which directly affects shadow softness
BBoxScale	Vec3	Scale factor for the bounding box of the selected entity. Can be useful in case the bounding box is too small or too large
ShadowMapSize	Integer	Size of the custom shadow map, which is automatically rounded to the next power of two

RainProperties node

Used to get and output rain properties.

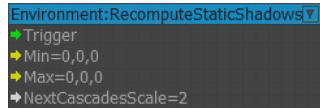


Inputs

Port	Type	Description
Trigger	Any	Activates the node
Amount	Float	Amount of rain
PuddlesAmount	Float	Amount of puddles
PuddlesRippleAmount	Float	Amount of puddle ripples
RainDropsAmount	Float	Amount of raindrops

RecomputeStaticShadows node

Cached shadow cascades are centered around the rendering camera by default, and automatically recenter and update once the camera gets close to the cascade border. Use this node to override this automated placement.

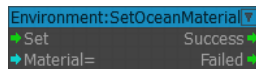


Input

Port	Type	Description
Trigger	Any	Activates the node
Min	Vec3	Minimum bounding box position
Max	Vec3	Maximum bounding box position
NextCascadesScale	Float	Input multiplier value

SetOceanMaterial node

Used to set the ocean material.



Inputs

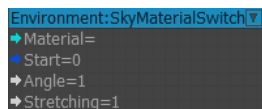
Port	Type	Description
Set	Any	Set material on for the ocean
Material	String	Material to be set for the ocean

Outputs

Port	Type	Description
Success	Any	Triggered when material set
Failed	Any	Triggered if an error occurred

SkyMaterialSwitch node

Used to enable sky material switching.



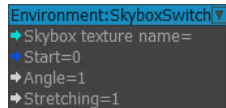
Inputs

Port	Type	Description
Material	String	Material to use for the sky
Start	Boolean	Start material switch

Port	Type	Description
Angle	Float	Starting angle
Stretching	Float	If stretching is performed or not

SkyboxSwitch node

Used to enable asynchronous sky box switching.

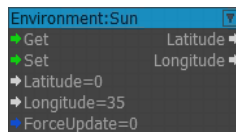


Inputs

Port	Type	Description
Skybox texture name	String	Name of texture file to use
Start	Boolean	Start asynchronous switching
Angle	Float	Starting angle
Stretching	Float	If stretching is performed or not

Sun node

Used to get and set the sun's position in the sky.



Inputs

Port	Type	Description
Get	Any	Get the current latitude and longitude
Set	Any	Set the latitude and longitude for the sun
Latitude	Float	Latitude to be set
Longitude	Float	Longitude to be set
ForceUpdate	Boolean	Forces an immediate update of the sky

Outputs

Port	Type	Description
Latitude	Float	Outputs current latitude

Port	Type	Description
Longitude	Float	Outputs current longitude

TornadoWander

Used to move a tornado entity in the direction of the target.



Inputs

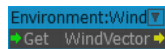
Port	Type	Description
Activate	Any	Activates the node
Target	Any	Location the tornado moves towards

Outputs

Port	Type	Description
Done	Any	Triggered when the tornado reaches the target

Wind node

Used to get and output the wind direction vector.



Inputs

Port	Type	Description
Get	Any	Get the current environment wind vector

Outputs

Port	Type	Description
WindVector	Vec3	Outputs current environment wind vector

FeatureTest Nodes

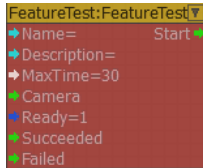
You can use the following flow graph nodes to configure feature test settings.

Topics

- [FeatureTest node \(p. 599\)](#)
- [Screenshot node \(p. 599\)](#)
- [ScreenshotCompare node \(p. 600\)](#)

FeatureTest node

Used to control automated feature tests.



Inputs

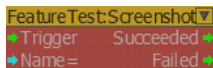
Port	Type	Description
Name	String	Name of the feature test
Description	String	Description of the feature test
MaxTime	Float	How long in game time the test is allowed to run before it fails
Camera	Any	(Optional) Camera entity used for the test
Ready	Boolean	Indicates whether all dependencies have been met and the test is ready to run
Succeeded	Any	Trigger to indicate the feature test has passed
Failed	Any	Trigger to indicate the feature test has failed

Outputs

Port	Type	Description
Start	Any	Trigger to start running the feature test

Screenshot node

Used to take a screenshot.



Inputs

Port	Type	Description
Trigger	Any	Trigger to capture a screenshot

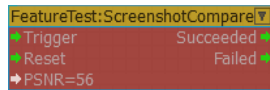
Port	Type	Description
Name	String	Name of the screenshot

Outputs

Port	Type	Description
Succeeded	Any	Triggers when the image is captured
Failed	Any	Triggers if the image is not captured

ScreenshotCompare node

Used to take a screenshot and compare it with a reference image.



Inputs

Port	Type	Description
Trigger	Any	Trigger to capture a screenshot
Reset	Any	Resets the current screenshot number back to 0
PSNR	Float	Picture signal to noise ratio used during comparison with the reference image to determine success of failure

Outputs

Port	Type	Description
Succeeded	Any	Triggers when the captured image matches the reference image
Failed	Any	Triggers when the captured image does not match the reference image

Game Nodes

You can use the following flow graph nodes to check and to configure various game settings.

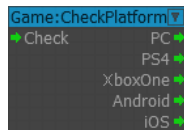
Topics

- [CheckPlatform node \(p. 601\)](#)
- [ForceFeedback node \(p. 601\)](#)
- [ForceFeedbackSetDeviceIndex node \(p. 602\)](#)
- [ForceFeedbackTriggerTweaker node \(p. 602\)](#)
- [ForceFeedbackTweaker node \(p. 603\)](#)
- [GetClientActorId node \(p. 603\)](#)

- [GetEntityState](#) node (p. 603)
- [GetGameRulesEntityId](#) node (p. 604)
- [GetSupportedGameRulesForMap](#) node (p. 604)
- [GetUsername](#) node (p. 604)
- [IsLevelOfType](#) node (p. 605)
- [ObjectEvent](#) node (p. 605)
- [Start](#) node (p. 606)

CheckPlatform node

Used to change game events depending on what platform you are running on.



Inputs

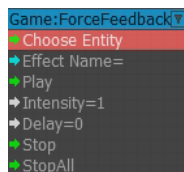
Port	Type	Description
Check	Any	Triggers a check of the current platform

Outputs

Port	Type	Description
PC	Any	Triggers if the platform is PC
PS4	Any	Triggers if the platform is PS4
XboxOne	Any	Triggers if the platform is XboxOne
Android	Any	Triggers if the platform is Android
iOS	Any	Triggers if the platform is iOS

ForceFeedback node

Used to start and stop force feedback effects.



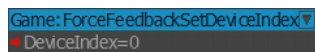
Inputs

Port	Type	Description
Effect Name	String	Name of the force feedback effect

Port	Type	Description
Play	Any	Plays the effect
Intensity	Float	Intensity level of effect
Delay	Float	Delays effect start by specified seconds
Stop	Any	Stops the effect
StopAll	Any	Stops all effects

ForceFeedbackSetDeviceIndex node

Used to set the receiving device ID for force feedback effects.

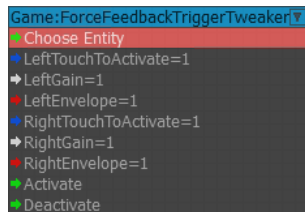


Inputs

Port	Type	Description
DeviceIndex	Integer	Sets the receiving device ID for force feedback effects

ForceFeedbackTriggerTweaker node

Used to control individual left and right trigger force feedback effects.

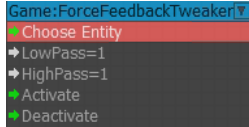


Inputs

Port	Type	Description
LeftTouchActivate	Boolean	Activates the left touch trigger
LeftGain	Float	Left trigger gain
LeftEnvelope	Integer	Left trigger envelope
RightTouchToActivate	Boolean	Activates the right touch trigger
RightGain	Float	Right trigger gain
RightEnvelope	Integer	Right trigger envelope
Activate	Any	Activates both triggers
Deactivate	Any	Deactivates both triggers

ForceFeedbackTweaker node

Used to control individual low and high frequency force feedback effects.

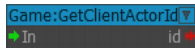


Inputs

Port	Type	Description
LowPass	Float	Low-frequency force feedback signal
HighPass	Float	High-frequency force feedback signal
Activate	Any	Activates force feedback effect
Deactivate	Any	Deactivates force feedback effect

GetClientActorId node

Used to output the client actor ID.



Inputs

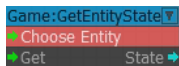
Port	Type	Description
In	Any	Gets client actor ID

Outputs

Port	Type	Description
id	Any	Outputs client actor ID

GetEntityState node

Used to output the current state of an entity.



Inputs

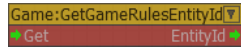
Port	Type	Description
Get	Any	Gets the entity state

Outputs

Port	Type	Description
State	String	Outputs the entity state

GetGameRulesEntityId node

Used to get the game rules entity ID.



Inputs

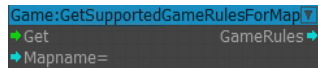
Port	Type	Description
Get	Any	Gets the entity ID of the rules script

Outputs

Port	Type	Description
EntityId	Any	The entity ID of the rules script

GetSupportedGameRulesForMap node

Used to get and output the supported game rules for a map.



Inputs

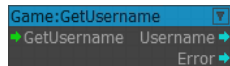
Port	Type	Description
Get	Any	Gets the game rules
Mapname	String	Map name

Outputs

Port	Type	Description
GameRules	String	Outputs the game rules

GetUsername node

Used to get the user name.



Inputs

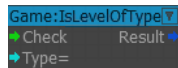
Port	Type	Description
GetUsername	Any	Gets the user name

Outputs

Port	Type	Description
Username	String	Outputs the user name
Error	String	Triggers if an error occurs

IsLevelOfType node

Used to check if a level is of a given type.



Inputs

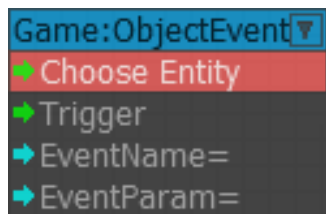
Port	Type	Description
Check	Any	Checks if a level is of a given type
Type	String	Level type to check against

Outputs

Port	Type	Description
Result	Boolean	The result of the check

ObjectEvent node

Used to broadcast a game object event or send it to a specific entity.



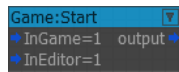
Inputs

Port	Type	Description
Trigger	Any	Triggers the node
EventName	String	Game object event name

Port	Type	Description
EventParam	String	Game object event parameter

Start node

Fires on the start of a game and used to trigger flow graphs upon level start.



Inputs

Port	Type	Description
InGame	Boolean	Triggers game mode to start
InEditor	Boolean	Triggers editor game mode to start

Outputs

Port	Type	Description
output	Boolean	Outputs the game mode

Helicopter Nodes

You can use the following flow graph nodes to configure flying vehicle and flight AI-related settings.

Note

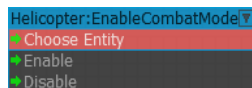
These nodes will only work with the Legacy Game Sample (CryEngine GameSDK), which is available at [Lumberyard Downloads](#).

Topics

- [EnableCombatMode node \(p. 606\)](#)
- [EnableFiring node \(p. 607\)](#)
- [FollowPath node \(p. 607\)](#)
- [ForceFire node \(p. 608\)](#)

EnableCombatMode node

Used to alter the path the flight AI should follow so as to find the best position from which to engage the target.



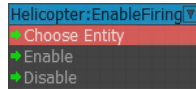
Inputs

Port	Type	Description
Enable	Any	Enables combat mode

Port	Type	Description
Disable	Any	Disables combat mode

EnableFiring node

Used to enable the flight AI to fire at a target when used in combination with the **EnableCombatMode** node.

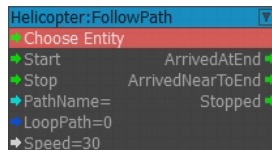


Inputs

Port	Type	Description
Enable	Any	Enables firing mode
Disable	Any	Disables firing mode

FollowPath node

Used to set the path that the flight AI should follow.



Inputs

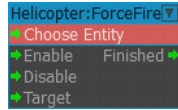
Port	Type	Description
Start	Any	Start following the path
Stop	Any	Stop following the path
PathName	String	Name of the path to follow
LoopPath	Boolean	How many times to loop around the path
Speed	Float	Speed of the flight AI

Outputs

Port	Type	Description
ArrivedAtEnd	Any	Triggers when flight AI is at the end of the path
ArrivedNearToEnd	Any	Triggers when flight AI is near the end of the path
Stopped	Any	Triggers when flight AI has stopped

ForceFire node

Used to force the attention target of the flight AI to a specific entity.



Inputs

Port	Type	Description
Enable	Any	Enables force firing
Disable	Any	Disables force firing
Target	Any	Attention target

Outputs

Port	Type	Description
Finished	Any	Triggers when finished

Image Nodes

You can use the following flow graph nodes to configure various visual effects and image settings.

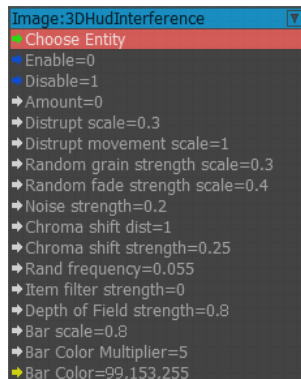
Topics

- [3DHudInterference node \(p. 609\)](#)
- [ColorCorrection node \(p. 610\)](#)
- [EffectAlienInterference node \(p. 610\)](#)
- [EffectBloodSplats node \(p. 611\)](#)
- [EffectDepthOfField node \(p. 611\)](#)
- [EffectFrost node \(p. 612\)](#)
- [EffectGhosting node \(p. 612\)](#)
- [EffectGroup node \(p. 612\)](#)
- [EffectRainDrops node \(p. 613\)](#)
- [EffectVolumetricScattering node \(p. 614\)](#)
- [EffectWaterDroplets node \(p. 614\)](#)
- [EffectWaterFlow node \(p. 614\)](#)
- [FilterBlur node \(p. 615\)](#)
- [FilterChromaShift node \(p. 615\)](#)
- [FilterDirectionalBlur node \(p. 615\)](#)
- [FilterGrain node \(p. 616\)](#)
- [FilterRadialBlur node \(p. 616\)](#)
- [FilterSharpen node \(p. 617\)](#)
- [FilterVisualArtifacts node \(p. 617\)](#)

- [ScreenCapture node \(p. 618\)](#)
- [ScreenFader node \(p. 618\)](#)
- [SetShadowMode node \(p. 619\)](#)

3DHudInterference node

Used to add distortion effects to the HUD.



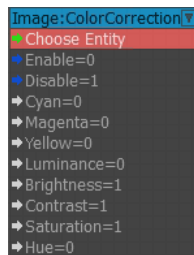
Inputs

Port	Type	Description
Enable	Boolean	Enables the effect
Disable	Boolean	Disables the effect
Amount	Float	Interference amount
Disrupt scale	Float	Disruption scale
Disrupt movement scale	Float	Disruption movement scale
Random grain strength scale	Float	Random grain strength scale
Random fade strength scale	Float	Random fade strength scale
Noise strength	Float	Noise strength
Chroma shift dist	Float	Chroma shift distance
Chroma shift strength	Float	Chroma shift strength
Rand frequency	Float	Random number generation frequency
Item filter strength	Float	Item filter strength. Uses the vertex color red channel to control item interference strength.
Depth of field strength	Float	Strength of the depth of field
Bar scale	Float	Bar scale
Bar color multiplier	Float	Bar color multiplier

Port	Type	Description
Bar color	Vec3	Bar color

ColorCorrection node

Used to control basic image settings such as saturation, contrast, brightness, and color.

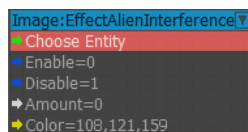


Inputs

Port	Type	Description
Enable	Boolean	Enables color correction
Disable	Boolean	Disables color correction
Cyan	Any	Cyan increase or decrease
Magenta	Any	Magenta increase or decrease
Yellow	Any	Yellow increase or decrease
Luminance	Any	Luminance increase or decrease
Brightness	Any	Brightness increase or decrease
Contrast	Any	Contrast increase or decrease
Saturation	Any	Saturation increase or decrease
Hue	Any	Hue increase or decrease

EffectAlienInterference node

Used to add distortion effects to the players view, but doesn't affect the HUD.



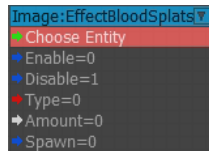
Inputs

Port	Type	Description
Enable	Boolean	Enables the effect
Disable	Boolean	Disables the effect

Port	Type	Description
Amount	Float	Intensity level of the effect
Color	Vec3	Color of the effect

EffectBloodSplats node

Used to place blood splats on the screen when used. Type=0 is human and Type =1 is alien. The Spawn input generates new blood splats.

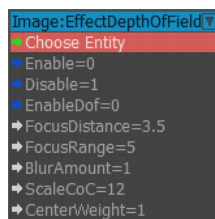


Inputs

Port	Type	Description
Enable	Boolean	Enables the effect
Disable	Boolean	Disables the effect
Type	Integer	Type of effect
Amount	Float	Intensity level of the effect
Spawn	Boolean	Where the effect spawns at

EffectDepthOfField node

Used to add a depth of field effect, giving control over distance, range, and amount.



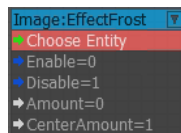
Inputs

Port	Type	Description
Enable	Boolean	Enables the node
Disable	Boolean	Disables the node
EnableDof	Boolean	Enables the depth of field effect
FocusDistance	Float	Sets the focus distance
FocusRange	Float	Sets the focus range

Port	Type	Description
BlurAmount	Float	Sets the amount of blurring
ScaleCoC	Float	Sets the circle of confusion scale, which is the optical spot caused by a light rays cone from a lens not coming to a perfect focus when imaging a point source. Also known as the blur circle of blur spot.
CenterWeight	Float	Sets the central samples weight

EffectFrost node

Used to simulate a frozen HUD.

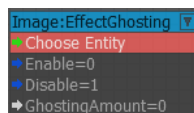


Inputs

Port	Type	Description
Enable	Boolean	Enables the effect
Disable	Boolean	Disables the effect
Amount	Any	Intensity level of the effect
CenterAmount	Any	Center of the effect

EffectGhosting node

Used to add a ghosting effect to the screen that overlaps and blurs previous frames together.

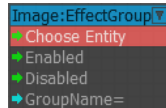


Inputs

Port	Type	Description
Enable	Boolean	Enables the effect
Disable	Boolean	Disables the effect
GhostingAmount	Float	Intensity level of effect

EffectGroup node

Used to enable the specified effect group.

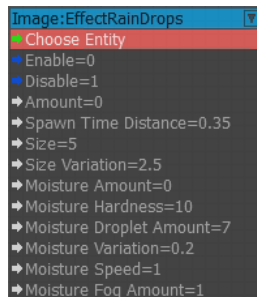


Inputs

Port	Type	Description
Enabled	Any	Enables the effect group
Disabled	Any	Disables the effect group
GroupName	String	Name of effect group

EffectRainDrops node

Used to add on-screen rain drops that travel down the player's HUD.

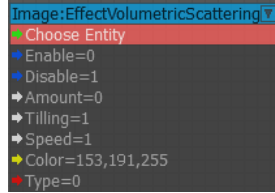


Inputs

Port	Type	Description
Enable	Boolean	Enables the node
Disable	Boolean	Disables the node
Amount	Float	Sets raindrop visibility
Spawn Time Distance	Float	Sets raindrop spawn time distance
Size	Float	Size of rain drops
Size Variation	Float	Amount of variation in size of rain drops
Moisture Amount	Float	Sets moisture visibility area size
Moisture Hardness	Float	Sets noise texture blending factor
Moisture Droplet Amount	Float	Sets droplet texture blending factor
Moisture Variation	Float	Sets moisture variation
Moisture Speed	Float	Sets moisture animation speed
Moisture Fog Amount	Float	Sets amount of fog in moisture

EffectVolumetricScattering node

Used to add a volumetric effect useful for simulating snowy environments. With the ability to control color, speed, and amount, you can simulate various environments, such as lava.

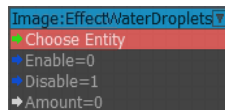


Inputs

Port	Type	Description
Enable	Boolean	Enables the node
Disable	Boolean	Disables the node
Amount	Float	Sets the amount of volumetric scattering
Tiling	Float	Sets the volumetric scattering tiling
Speed	Float	Sets the volumetric scattering animation speed
Color	Vec3	Sets the volumetric scattering color
Type	Integer	Defines the type of volumetric scattering

EffectWaterDroplets node

Used to add a water effect that appears from various sources on the screen. Unlike the RainDroplets node, this simulates more of a splash-type effect of water being thrown on the screen in various places.

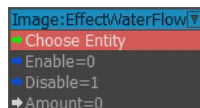


Inputs

Port	Type	Description
Enable	Boolean	Enables the effect
Disable	Boolean	Disables the effect
Amount	Float	Intensity level of effect

EffectWaterFlow node

Used to simulate dense water running down the screen, such as standing under a waterfall.

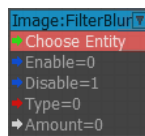


Inputs

Port	Type	Description
Enable	Boolean	Enables the effect
Disable	Boolean	Disables the effect
Amount	Float	Intensity level of filter

FilterBlur node

Used to Gaussian blur the entire screen, useful for simulating dense smoke affecting the player's eyes.

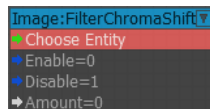


Inputs

Port	Type	Description
Enable	Boolean	Enables the filter
Disable	Boolean	Disables the filter
Type	Integer	Type of effect
Amount	Float	Intensity level of effect

FilterChromaShift node

Used to shift the chrominance information of the image. Best used in small amounts to create subtle film effects.

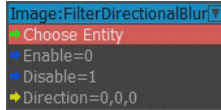


Inputs

Port	Type	Description
Enable	Boolean	Enables the filter
Disable	Boolean	Disables the filter
Amount	Float	Intensity level of filter

FilterDirectionalBlur node

Used to apply a blur in a specified direction based on movement.

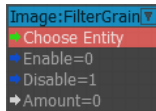


Inputs

Port	Type	Description
Enable	Boolean	Enables the filter
Disable	Boolean	Disables the filter
Direction	Vec3	Direction of blurring effect

FilterGrain node

Used to set a grain filter.

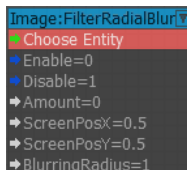


Inputs

Port	Type	Description
Enable	Boolean	Enables the filter
Disable	Boolean	Disables the filter
Amount	Float	Intensity level of filter

FilterRadialBlur node

Used to blur the screen around a defined 2D position on the screen.



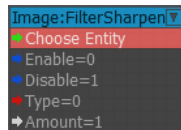
Inputs

Port	Type	Description
Enable	Boolean	Enables the filter
Disable	Boolean	Disables the filter
Amount	Float	Intensity level of filter
ScreenPosX	Float	X-axis center of blurring effect
ScreenPosY	Float	Y-axis center of blurring effect

Port	Type	Description
BlurringRadius	Float	Radius of blurring effect

FilterSharpen node

Used to add sharpening to the image. You can use negative values to blur the screen also.

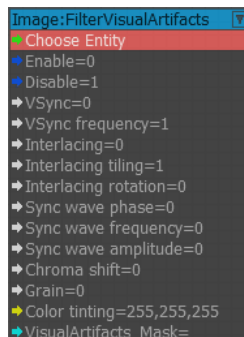


Inputs

Port	Type	Description
Enable	Boolean	Enables the filter
Disable	Boolean	Disables the filter
Type	Integer	Type of filter
Amount	Float	Intensity level of filter

FilterVisualArtifacts node

Used to apply numerous effects typically associate with old television sets, such as grain, vsync, interlacing, and pixelation. You can mask the effect using a texture, or apply it to the whole screen.



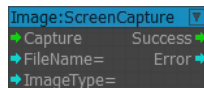
Inputs

Port	Type	Description
Enable	Boolean	Enables the node
Disable	Boolean	Disables the node
VSync	Float	Amount of visible vsync
VSync frequency	Float	Vsync frequency
Interlacing	Float	Amount of visible interlacing
Interlacing tiling	Float	Interlacing tiling

Port	Type	Description
Interlacing rotation	Float	Interlacing rotation
Sync wave phase	Float	Sync wave phase
Sync wave frequency	Float	Sync wave frequency
Sync wave amplitude	Float	Sync wave amplitude
Chroma shift	Float	Chromatic shift
Grain	Float	Amount of image grain
Color tinting	Vec3	Amount of color tinting
VisualArtifacts	String	Name of texture used

ScreenCapture node

Used to capture a screenshot.



Inputs

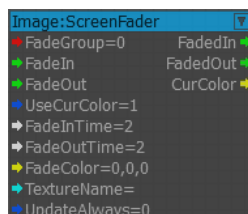
Port	Type	Description
Capture	Any	Trigger to capture the screenshot
FileName	Any	File to write the screenshot capture to
ImageType	Any	File type to use

Outputs

Port	Type	Description
Success	Any	Screenshot capture successful
Error	String	Screenshot capture failed

ScreenFader node

Used to perform customizable fade-in and fade-out effects, including the ability to fade from textures. The UseCurColor input uses the previously set color as the fading color if set to True, else it uses the FadeColor value.



Inputs

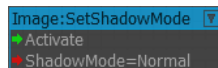
Port	Type	Description
FadeGroup	Any	Fade group
FadeIn	Any	Fade back from the specified color to a normal screen
FadeOut	Any	Fades the screen to the specified color
UseCurColor	Boolean	Uses the current color as the source color
FadeInTime	Float	Duration of fade in
FadeOutTime	Float	Duration of fade out
FadeColor	Vec3	Target color to fade to
TextureName	String	Name of the texture
UpdateAlways	Boolean	Use to always update the fader

Outputs

Port	Type	Description
FadedIn	Any	Triggered when the screen completed faded in
FadedOut	Any	Triggered when the screen completed faded out
CurColor	Any	Current faded color

SetShadowMode node

Used to set the shadow mode to Normal or HighQuality mode. Intended to be used for very specific lighting setups and will likely result in self-shadowing artifacts under typical use.



Inputs

Port	Type	Description
Activate	Any	Activates the node
ShadowMode	Integer	Shadow mode type to use

Input Nodes

You can use the following flow graph nodes to capture input events and configure input settings.

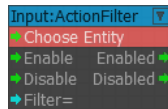
Topics

- [ActionFilter node \(p. 620\)](#)
- [ActionHandler node \(p. 620\)](#)

- [ActionListener](#) node (p. 621)
- [ActionMapManager](#) node (p. 622)
- [Gestures](#) nodes (p. 622)
- [MotionSensor](#) nodes (p. 628)
- [MouseButtonInfo](#) node (p. 632)
- [MouseCoords](#) node (p. 633)
- [MouseCursor](#) node (p. 634)
- [MouseEntitiesInBox](#) node (p. 634)
- [MouseRayCast](#) node (p. 634)
- [MouseSetPos](#) node (p. 635)
- [Touch:MultiTouchEvent](#) node (p. 636)
- [Touch:TouchEvent](#) node (p. 636)
- [Touch:MultiTouchCoords](#) node (p. 637)
- [Touch:TouchRaycast](#) node (p. 637)
- [Touch:VirtualThumbstick](#) node (p. 638)

ActionFilter node

Used to catch key inputs. Should only be used for debugging purposes however.



Inputs

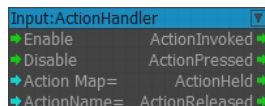
Port	Type	Description
Enable	Any	Enables the node
Disable	Any	Disables the node
Filter	String	Name of the action filter

Outputs

Port	Type	Description
Enabled	Any	Triggers when enabled
Disabled	Any	Triggers when disabled

ActionHandler node

Used to respond to actions listed in the **Action Map** input.



Inputs

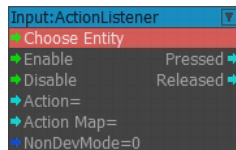
Port	Type	Description
Enable	Any	Enables listening to the action map
Disable	Any	Disables listening to the action map
Action Map	String	Name of the action map
ActionName	String	Name of the action to listen for

Outputs

Port	Type	Description
ActionInvoked	Any	Triggers when the action is invoked
ActionPressed	Any	Triggers when the action is pressed
ActionHeld	Any	Triggers when the action is sustained
ActionReleased	Any	Triggers when the action is released

ActionListener node

Used to listen for action events listed in the **Action Map**.



Inputs

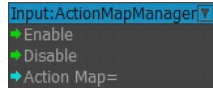
Port	Type	Description
Enable	Any	Enables the node
Disable	Any	Disables the node
Action	String	Action to trigger
Action Map	String	Action map to use
NonDevMode	Boolean	When set to true, can be used in non dev mode as well

Outputs

Port	Type	Description
Pressed	String	Triggers when the action is pressed
Released	String	Triggers when the action is released

ActionMapManager node

Used to enable or disable the **Action Map** input.



Inputs

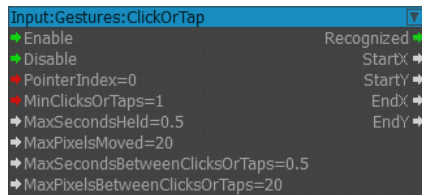
Port	Type	Description
Enable	Any	Enables the node
Disable	Any	Disables the node
Action Map	String	Name of the action map to use

Gestures nodes

This group of nodes is used to handle finger taps, swipes, and other gestures as input.

Gestures:ClickOrTap node

Used to recognize one or more mouse clicks or finger taps.



Inputs

Port	Type	Description
Enable	Any	Enables the node
Disable	Any	Disables the node
PointerIndex	Integer	Pointer (button or finger) index to track
MinClicksOrTaps	Integer	Minimum number of clicks or taps required for the gesture to be recognized
MaxSecondsHeld	Float	Maximum number of seconds allowed while held before the gesture stops being recognized
MaxPixelsMoved	Float	Maximum distance in pixels allowed to move while held before the gesture stops being recognized
MaxSecondsBetweenClicksOrTaps	Float	Maximum number of seconds allowed between clicks or taps

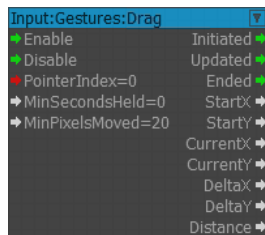
Port	Type	Description
MaxPixelsBetweenClicksOrTaps	Float	Maximum distance in pixels allowed between clicks or taps

Outputs

Port	Type	Description
Recognized	Any	Triggers when a discrete number of clicks or taps is recognized
StartX	Float	X-axis screen position of the click or tap start
StartY	Float	Y-axis screen position of the click or tap start
EndX	Float	X-axis screen position of the click or tap end
EndY	Float	Y-axis screen position of the click or tap end

Gestures:Drag node

Used to recognize finger drag gestures.



Inputs

Port	Type	Description
Enable	Any	Enables the node
Disable	Any	Disables the node
PointerIndex	Integer	Pointer (button or finger) index to track
MinSecondsHeld	Float	Mimimum number of seconds after the initial press before a drag is recognized
MinPixelsMoved	Float	Mimimum distance in pixels before a drag is recognized

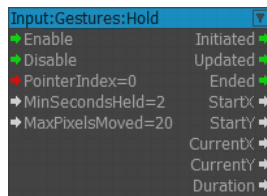
Outputs

Port	Type	Description
Initiated	Any	Activated when a continuous drag gesture is initiated
Updated	Any	Activated when a continuous drag gesture is updated

Port	Type	Description
Ended	Any	Activated when a continuous drag gesture has ended
StartX	Float	X-axis screen position of the drag start
StartY	Float	Y-axis screen position of the drag start
CurrentX	Float	Current X-axis screen position of the drag
CurrentY	Float	Current Y-axis screen position of the drag
DeltaX	Float	Number of pixels dragged on the X-axis screen
DeltaY	Float	Number of pixels dragged on the Y-axis screen
Distance	Float	Number of pixels dragged on screen

Gestures:Hold node

Used to recognize finger hold gestures.



Inputs

Port	Type	Description
Enable	Any	Enables the node
Disable	Any	Disables the node
PointerIndex	Integer	The button or finger index to track
MinSecondsHeld	Float	Minimum number of seconds before a hold is recognized
MaxPixelsMoved	Float	Minimum distance in pixels before a hold is recognized

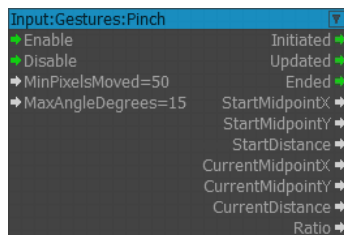
Outputs

Port	Type	Description
Initiated	Any	Activated when a continuous hold gesture is initiated
Updated	Any	Activated when a continuous hold gesture is updated
Ended	Any	Activated when a continuous hold gesture has ended

Port	Type	Description
StartX	Float	X-axis screen position of the hold start
StartY	Float	Y-axis screen position of the hold start
CurrentX	Float	Current X-axis screen position of the hold
CurrentY	Float	Current Y-axis screen position of the hold
Duration	Float	Duration of the hold in seconds

Gestures:Pinch node

Used to recognize finger pinch (away from or toward) gestures.



Inputs

Port	Type	Description
Enable	Any	Enables the node
Disable	Any	Disables the node
MinPixelsMoved	Float	Minimum distance in pixels before a pinch is recognized
MaxAngleDegrees	Float	Maximum angle in degrees that pinch can deviate before it is recognized

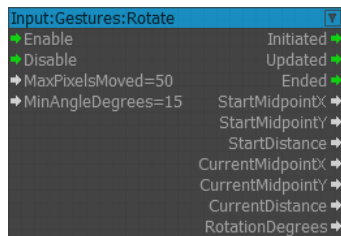
Outputs

Port	Type	Description
Initiated	Any	Activated when a continuous pinch gesture is initiated
Updated	Any	Activated when a continuous pinch gesture is updated
StartMidpointX	Any	Midpoint X-axis position of the pinch
StartMidpointY	Float	Midpoint Y-axis position of the pinch
StartDistance	Float	Pixel distance between the two touch positions when the pinch is started
CurrentMidpointX	Float	Current X-axis position of the pinch

Port	Type	Description
CurrentMidpointY	Float	Current Y-axis position of the pinch
CurrentDistance	Float	Current distance in pixels between the two touch positions
Ratio	Float	Ratio of the pinch (CurrentDistance/ StartDistance)

Gestures:Rotate node

Used to recognize finger rotation (movement in a circle around each other) gestures.



Inputs

Port	Type	Description
Enable	Any	Enables the node
Disable	Any	Disables the node
MaxPixelsMoved	Float	Maximum distance in pixels before a rotation is recognized
MinAngleDegrees	Float	Minimum angle in degrees before a rotation is recognized

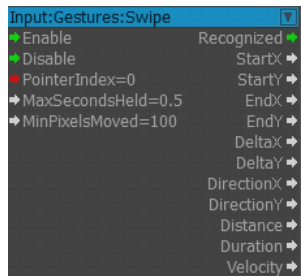
Outputs

Port	Type	Description
Initiated	Any	Activated when a continuous rotation gesture is initiated
Updated	Any	Activated when a continuous rotation gesture is updated
Ended	Any	Activated when a continuous rotation gesture has ended
StartMidpointX	Float	X-axis screen position where the rotation started
StartMidpointY	Float	Y-axis screen position where the rotation started
StartDistance	Float	Pixel distance between the two touch positions when the rotation started
CurrentMidpointX	Float	Current X-axis screen position of the rotation

Port	Type	Description
CurrentMidpointY	Float	Current Y-axis screen position of the rotation
CurrentDistance	Float	Current pixel distance between the two touch positions
RotationDegrees	Float	Current rotation in degrees

Gestures:Swipe node

Used to recognize finger swipe gestures.



Inputs

Port	Type	Description
Enable	Any	Enables the node
Disable	Any	Disables the node
PointerIndex	Integer	The button or finger index to track
MaxSecondsHeld	Float	Maximum number of seconds for a swipe to be recognized
MinPixelsMoved	Float	Minimum distance in pixels before a swipe is recognized

Outputs

Port	Type	Description
Recognized	Any	Activated when a continuous swipe gesture is recognized
StartX	Float	X-axis screen position where the swipe started
StartY	Float	Y-axis screen position where the swipe started
EndX	Float	X-axis screen position where the swipe ended
EndY	Float	Y-axis screen position where the swipe ended
DeltaX	Float	X-axis pixels swiped
DeltaY	Float	Y-axis pixels swiped

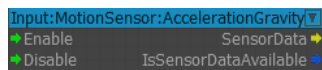
Port	Type	Description
DirectionX	Float	X-axis direction of the swipe
DirectionY	Float	Y-axis direction of the swipe
Distance	Float	Distance of the swipe in pixels
Duration	Float	Duration of the swipe in seconds
Velocity	Float	Velocity of the swipe in pixels per second

MotionSensor nodes

This group of nodes are used with a motion sensor or accelerometer input.

MotionSensor:AccelerationGravity node

Used to output gravity-generated acceleration.



Inputs

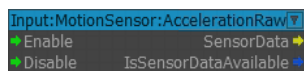
Port	Type	Description
Enable	Any	Enables the node
Disable	Any	Disables the node

Outputs

Port	Type	Description
SensorData	Vec3	Outputs raw gravity acceleration in g-forces
IsSensorDataAvailable	Boolean	Outputs true or false when the node is activated

MotionSensor:AccelerationRaw node

Used to output raw acceleration.



Inputs

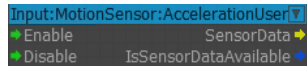
Port	Type	Description
Enable	Any	Enables the node
Disable	Any	Disables the node

Outputs

Port	Type	Description
SensorData	Vec3	Outputs raw sensor acceleration in g-forces
IsSensorDataAvailable	Boolean	Outputs true or false when the node is activated

MotionSensor:AccelerationUser node

Used to output user-generated acceleration.



Inputs

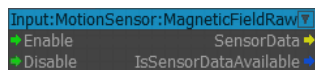
Port	Type	Description
Enable	Any	Enables the node
Disable	Any	Disables the node

Outputs

Port	Type	Description
SensorData	Vec3	Outputs user-generated acceleration in g-forces
IsSensorDataAvailable	Boolean	Outputs true or false when the node is activated

MotionSensor:MagneticFieldRaw node

Used to output raw magnetic field data as measured by a magnetometer. Includes device bias.



Inputs

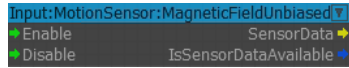
Port	Type	Description
Enable	Any	Enables the node
Disable	Any	Disables the node

Outputs

Port	Type	Description
SensorData	Vec3	Outputs raw magnetic field in microteslas
IsSensorDataAvailable	Boolean	Outputs true or false when the node is activated

MotionSensor:MagneticFieldUnbiased node

Used to output magnetic field data as measured by a magnetometer. Processed to remove device bias.



Inputs

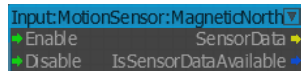
Port	Type	Description
Enable	Any	Enables the node
Disable	Any	Disables the node

Outputs

Port	Type	Description
SensorData	Vec3	Outputs unbiased magnetic field data in microteslas
IsSensorDataAvailable	Boolean	Outputs true or false when the node is activated

MotionSensor:MagneticNorth node

Used to output a vector pointing to magnetic north.



Inputs

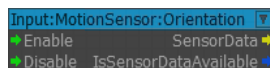
Port	Type	Description
Enable	Any	Enables the node
Disable	Any	Disables the node

Outputs

Port	Type	Description
SensorData	Vec3	Outputs a vector pointing to magnetic north
IsSensorDataAvailable	Boolean	Outputs true or false when the node is activated

MotionSensor:Orientation node

Used to measure the orientation or attitude of the device from an arbitrary but constant frame of reference.



Inputs

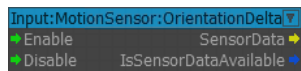
Port	Type	Description
Enable	Any	Enables the node
Disable	Any	Disables the node

Outputs

Port	Type	Description
SensorData	Vec3	Outputs an orientation or attitude angle in degrees
IsSensorDataAvailable	Boolean	Outputs true or false when the node is activated

MotionSensor:OrientationDelta node

Used to measure the change in orientation or attitude of the device since the last measurement.



Inputs

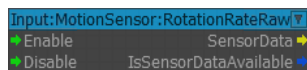
Port	Type	Description
Enable	Any	Enables the node
Disable	Any	Disables the node

Outputs

Port	Type	Description
SensorData	Vec3	Outputs an orientation or attitude angle in degrees
IsSensorDataAvailable	Boolean	Outputs true or false when the node is activated

MotionSensor:RotationRateRaw node

Used to output the raw rotation rate as measured by the gyroscope.



Inputs

Port	Type	Description
Enable	Any	Enables the node

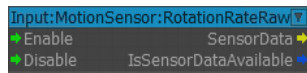
Port	Type	Description
Disable	Any	Disables the node

Outputs

Port	Type	Description
SensorData	Vec3	Outputs the raw gyroscope rotation rate in degrees per second
IsSensorDataAvailable	Boolean	Outputs true or false when the node is activated

MotionSensor:RotationRateUnbiased node

Used to output the rotation rate as measured by the gyroscope and processed to remove device bias.



Inputs

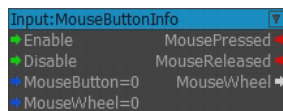
Port	Type	Description
Enable	Any	Enables the node
Disable	Any	Disables the node

Outputs

Port	Type	Description
SensorData	Vec3	Outputs an unbiased rotation rate in degrees per second
IsSensorDataAvailable	Boolean	Outputs true or false when the node is activated

MouseButtonInfo node

Used to output mouse button state information.



Inputs

Port	Type	Description
Enable	Any	Enables the node
Disable	Any	Disables the node

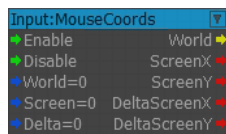
Port	Type	Description
MouseButton	Boolean	Mouse button state information
MouseWheel	Boolean	Mouse wheel state information

Outputs

Port	Type	Description
MousePressed	Integer	Outputs the mouse button that was pressed
MouseReleased	Integer	Outputs the mouse button that was released
MouseWheel	Float	Outputs a positive value when the mouse wheel is moved up and a negative value when moved down

MouseCoords node

Used to output mouse coordinates.



Inputs

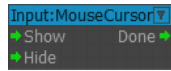
Port	Type	Description
Enable	Any	Enables the node
Disable	Any	Disables the node
World	Boolean	World coordinates used
Screen	Boolean	Screen coordinates of the mouse cursor
Delta	Boolean	Shows the number of screen pixels the mouse cursor has moved

Outputs

Port	Type	Description
World	Vec3	World coordinates of the mouse cursor
ScreenX	Integer	X-axis coordinate of mouse cursor
ScreenY	Integer	Y-axis coordinate of mouse cursor
DeltaScreenX	Integer	X-axis delta coordinate of mouse cursor
DeltaScreenY	Integer	Y-axis delta coordinate of mouse cursor

MouseCursor node

Used to show or hide the mouse cursor.



Inputs

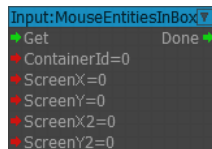
Port	Type	Description
Show	Any	Shows the mouse cursor
Hide	Any	Hides the mouse cursor

Outputs

Port	Type	Description
Done	Any	Triggers when the action is complete

MouseEntitiesInBox node

Used to show or hide the mouse coordinates.



Inputs

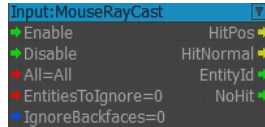
Port	Type	Description
Get	Any	Get the mouse cursor
ContainerId	Integer	ID of the container that stores the entities
ScreenX	Integer	X-axis screen position of the mouse cursor
ScreenY	Integer	Y-axis screen position of the mouse cursor
ScreenX2	Integer	X-axis screen position 2 of the mouse cursor
ScreenY2	Integer	Y-axis screen position 2 of the mouse cursor

Outputs

Port	Type	Description
Done	Any	Triggers when completed

MouseRayCast node

Used to output the mouse raycast information.



Inputs

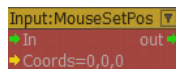
Port	Type	Description
Enable	Any	Enables the node
Disable	Any	Disables the node
All	Integer	Raycast filter type
EntitiesToIgnore	Integer	Entities to ignore during raycast
IgnoreBackFaces	Boolean	Ignore backfaces of geometry during raycast

Outputs

Port	Type	Description
HitPos	Vec3	Coordinates of the first position that was hit with the raycast
HitNormal	Vec3	Normal of the first position that was hit with the raycast
EntityId	Any	ID of the entity that was hit by the raycast
NoHit	Any	Activated each frame when enabled and no item was hit by the raycast

MouseSetPos node

Used to position the mouse at the specified location when activated.



Inputs

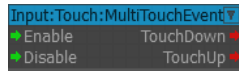
Port	Type	Description
In	Any	Activates the node
Coords	Vec3	Coordinates to set the mouse at

Outputs

Port	Type	Description
Out	Any	Triggers when the new mouse position is set

Touch:MultiTouchEvent node

Used to output finger touch location.



Inputs

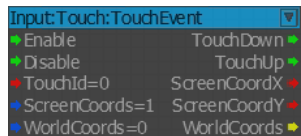
Port	Type	Description
Enable	Any	Enables the node
Disable	Any	Disables the node

Outputs

Port	Type	Description
TouchDown	Integer	Finger (touch) ID that was pressed
TouchUp	Integer	Finger (touch) ID that was released

Touch:TouchEvent node

Used to output finger touch location.



Inputs

Port	Type	Description
Enable	Any	Enables the node
Disable	Any	Disables the node
TouchId	Integer	Touch (finger) ID for which events will be sent from
ScreenCoords	Boolean	Output screen coordinates
WorldCoords	Boolean	Output world coordinates

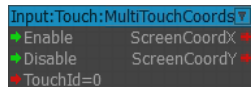
Outputs

Port	Type	Description
TouchDown	Any	Finger (touch) ID that was pressed
TouchUp	Any	Finger (touch) ID that was released

Port	Type	Description
ScreenCoordX	Integer	Screen X-axis coordinate of the touch
ScreenCoordY	Integer	Screen Y-axis coordinate of the touch
WorldCoords	Vec3	Touch position in world coordinates

Touch:MultiTouchCoords node

Used to output the finger touch location from the specified ID.



Inputs

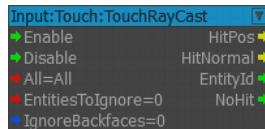
Port	Type	Description
Enable	Any	Enables the node
Disable	Any	Disables the node
TouchId	Integer	Finger (touch) ID for which the coordinates will be obtained

Outputs

Port	Type	Description
ScreenCoordX	Integer	X-axis location of the finger touch
ScreenCoordY	Integer	Y-axis location of the finger touch

Touch:TouchRaycast node

Used to generate a raycast for each finger frame ID.



Inputs

Port	Type	Description
Enable	Any	Enables the node
Disable	Any	Disables the node
All	Integer	Raycast filter type
EntitiesToIgnore	Integer	Entities to ignore during raycast

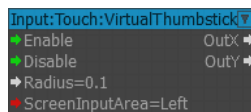
Port	Type	Description
IgnoreBackFaces	Boolean	Ignore backfaces of geometry during raycast

Outputs

Port	Type	Description
HitPos	Vec3	Coordinates of the first position that was hit with the raycast
HitNormal	Vec3	Normal of the first position that was hit with the raycast
EntityId	Any	ID of the entity that was hit by the raycast
NoHit	Any	Activated each frame when enabled and no item was hit by the raycast

Touch:VirtualThumbstick node

Used to implement a virtual thumbstick.



Inputs

Port	Type	Description
Enable	Any	Enables the node
Disable	Any	Disables the node
Radius	Float	Radius of thumbstick pad as a percentage of screen width
ScreenInputArea	Integer	What side of the screen the thumbstick should accept input from

Outputs

Port	Type	Description
OutX	Float	X-axis value of the thumbstick
OutY	Float	Y-axis value of the thumbstick

Interpolate Nodes

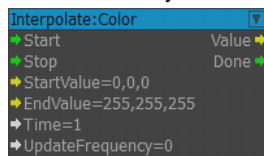
You can use these flow graph nodes to configure interpolate-related settings.

Topics

- [Color node \(p. 639\)](#)
- [Float node \(p. 639\)](#)
- [Int node \(p. 640\)](#)
- [SmoothAngleVec3 \(p. 641\)](#)
- [SmoothColor node \(p. 641\)](#)
- [SmoothFloat node \(p. 642\)](#)
- [SmoothInt node \(p. 642\)](#)
- [SmoothVec3 node \(p. 643\)](#)
- [Vec3 node \(p. 643\)](#)

Color node

Used to linearly calculate from an initial color value to an end color value within a given time frame.



Inputs

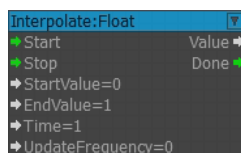
Port	Type	Description
Start	Any	Starts interpolation
Stop	Any	Stops interpolation
StartValue	Vec3	Starting value for color
EndValue	Vec3	Ending value for color
Time	Float	Interpolation duration in seconds
UpdateFrequency	Float	Interpolation update frequency in seconds. 0 = every frame

Outputs

Port	Type	Description
Value	Vec3	Current value
Done	Any	Triggered when finished

Float node

Used to linearly calculate from an initial floating point value to an end floating point value within a given time frame.



Inputs

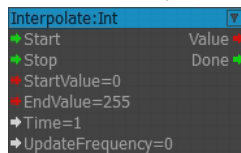
Port	Type	Description
Start	Any	Starts interpolation
Stop	Any	Stops interpolation
StartValue	Float	Starting value for floating point
EndValue	Float	Ending value for floating point
Time	Float	Interpolation duration in seconds
UpdateFrequency	Float	Interpolation update frequency in seconds. 0 = every frame

Outputs

Port	Type	Description
Value	Float	Current value
Done	Any	Triggered when finished

Int node

Used to linearly calculate from an initial integer value to an end integer value within a given time frame.



Input

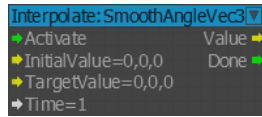
Port	Type	Description
Start	Any	Starts interpolation
Stop	Any	Stops interpolation
StartValue	Integer	Starting value for integer
EndValue	Integer	Ending value for integer
Time	Float	Interpolation duration in seconds
UpdateFrequency	Float	Interpolation update frequency in seconds. 0 = every frame

Outputs

Port	Type	Description
Value	Integer	Current value
Done	Any	Triggered when finished

SmoothAngleVec3

Used to non-linearly (damped spring system) calculate from an initial vector angle to an end vector angle within a given time frame. Calculation will slow down as it reaches the end value.



Inputs

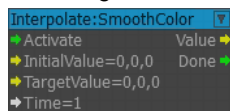
Port	Type	Description
Activate	Any	Triggers the node
InitialValue	Vec3	Initial interpolation value for vector angle
TargetValue	Vec3	Target interpolation value for vector angle
Time	Float	Interpolation duration in seconds

Outputs

Port	Type	Description
Value	Vec3	Current value
Done	Any	Triggered when finished

SmoothColor node

Used to non-linearly (damped spring system) calculate from an initial color value to an end color value within a given time frame. Calculation will slow down as it reaches the end value.



Inputs

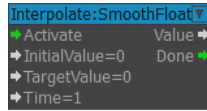
Port	Type	Description
Activate	Any	Triggers the node
InitialValue	Vec3	Initial interpolation value for color
TargetValue	Vec3	Target interpolation value for color
Time	Float	Interpolation duration in seconds

Outputs

Port	Type	Description
Value	Vec3	Current value
Done	Any	Triggered when finished

SmoothFloat node

Used to non-linearly (damped spring system) calculate from an initial floating point value to an end floating point value within a given time frame. Calculation will slow down as it reaches the end value.



Inputs

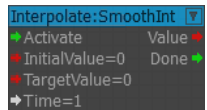
Port	Type	Description
Activate	Any	Triggers the node
InitialValue	Float	Initial interpolation value for floating point
TargetValue	Float	Target interpolation value for floating point
Time	Float	Interpolation duration in seconds

Outputs

Port	Type	Description
Value	Float	Current value
Done	Any	Triggered when finished

SmoothInt node

Used to non-linearly (damped spring system) calculate from an initial integer value to an end integer value within a given time frame. Calculation will slow down as it reaches the end value.



Inputs

Port	Type	Description
Activate	Any	Triggers the node
InitialValue	Integer	Initial interpolation value for integer
TargetValue	Integer	Target interpolation value for integer
Time	Float	Interpolation duration in seconds

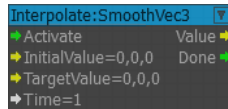
Outputs

Port	Type	Description
Value	Integer	Current value

Port	Type	Description
Done	Any	Triggered when finished

SmoothVec3 node

Used to non-linearly (damped spring system) calculate from an initial Vec3 value to an end Vec3 value within a given time frame. Calculation will slow down as it reaches the end value.



Inputs

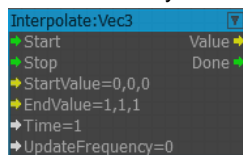
Port	Type	Description
Activate	Any	Triggers the node
InitialValue	Vec3	Initial interpolation value for Vec3
TargetValue	Vec3	Target interpolation value for Vec3
Time	Float	Interpolation duration in seconds

Outputs

Port	Type	Description
Value	Vec3	Current value
Done	Any	Triggered when finished

Vec3 node

Used to linearly calculate from an initial Vec3 value to an end Vec3 value within a given time frame.



Inputs

Port	Type	Description
Start	Any	Starts interpolation
Stop	Any	Stops interpolation
StartValue	Vec3	Starting value for Vec3
EndValue	Vec3	Ending value for Vec3
Time	Float	Interpolation duration in seconds

Port	Type	Description
UpdateFrequency	Float	Interpolation update frequency in seconds. 0 = every frame

Outputs

Port	Type	Description
Value	Vec3	Current value
Done	Any	Triggered when finished

Intersection Tests Nodes

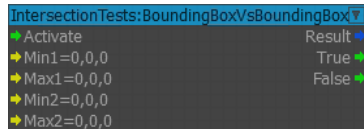
You can use the following flow graph nodes to configure intersection tests.

Topics

- [BoundingBoxVsBoundingBox node \(p. 644\)](#)
- [BoundingBoxVsSphere node \(p. 645\)](#)

BoundingBoxVsBoundingBox node

Used to test two bounding boxes to see if they intersect.



Inputs

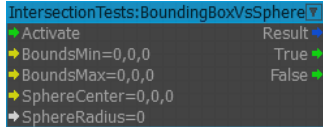
Port	Type	Description
Activate	Any	Triggers the node
Min1	Vec3	Minimum point for the first bounding box
Max1	Vec3	Maximum point for the first bounding box
Min2	Vec3	Minimum point for the second bounding box
Max2	Vec3	Maximum point for the second bounding box

Outputs

Port	Type	Description
Result	Boolean	Outputs true if an intersection occurred
True	Any	Triggers if an intersection occurred
False	Any	Triggers if an intersection did not occur

BoundingBoxVsSphere node

Used to test a bounding box and a sphere to see if they intersect.



Inputs

Port	Type	Description
Activate	Any	Triggers the node
BoundsMin	Vec3	Minimum point of the bounding box
BoundsMax	Vec3	Maximum point of the bounding box
SphereCenter	Vec3	Center of the sphere
SphereRadius	Float	Radius of the sphere

Outputs

Port	Type	Description
Result	Boolean	Outputs true if an intersection occurred
True	Any	Triggers if an intersection occurred
False	Any	Triggers if an intersection did not occur

Iterator Nodes

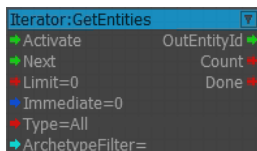
You can use the following flow graph nodes to configure iterator-related settings.

Topics

- [GetEntities node \(p. 645\)](#)
- [GetEntitiesInArea node \(p. 646\)](#)
- [GetEntitiesInBox node \(p. 647\)](#)
- [GetEntitiesInSphere node \(p. 647\)](#)

GetEntities node

Used to find and return all entities in the world.



Inputs

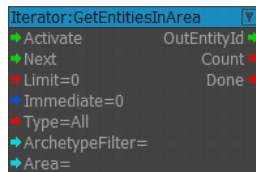
Port	Type	Description
Activate	Any	Triggers the node
Next	Any	Gets the next entity found
Limit	Integer	Limits how many entities are returned
Immediate	Boolean	Iterates immediately through the results
Type	Integer	Type of entity to iterate
ArchetypeFilter	Any	Returns archetype entities

Outputs

Port	Type	Description
OutEntityId	Any	Outputs the entity and entity ID
Count	Integer	Outputs the current of entities
Done	Integer	Triggered when all entities have been found, with the total count returned

GetEntitiesInArea node

Used to find and return all entities within the specified area shape.



Inputs

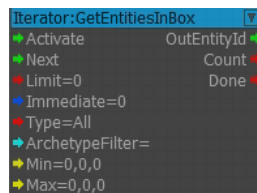
Port	Type	Description
Activate	Any	Triggers the node
Next	Any	Gets the next entity found
Limit	Integer	Limits how many entities are returned
Immediate	Boolean	Iterates immediately through the results
Type	Integer	Type of entity to iterate
ArchetypeFilter	String	Returns archetype entities
Area	String	Name of area shape to test against

Outputs

Port	Type	Description
OutEntityId	Any	Outputs the entity and entity ID
Count	Integer	Outputs the current of entities
Done	Integer	Triggered when all entities have been found, with the total count returned

GetEntitiesInBox node

Used to find and return all entities within the defined AABB box.



Inputs

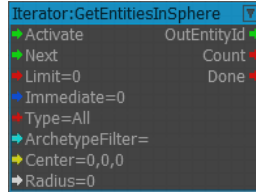
Port	Type	Description
Activate	Any	Triggers the node
Next	Any	Gets the next entity found
Limit	Integer	Limits how many entities are returned
Immediate	Any	Iterates immediately through the results
Type	Integer	Type of entity to iterate
ArchetypeFilter	String	Returns archetype entities
Min	Vec3	Minimum vector extents of the AABB bounding box to check for entities
Max	Vec3	Maximum vector extents of the AABB bounding box to check for entities

Outputs

Port	Type	Description
OutEntityId	Any	Outputs the entity and entity ID
Count	Integer	Outputs the current of entities
Done	Integer	Triggered when all entities have been found, with the total count returned

GetEntitiesInSphere node

Used to find and return all entities within the defined sphere volume.



Inputs

Port	Type	Description
Activate	Any	Triggers the node
Next	Any	Gets the next entity found
Limit	Integer	Limits how many entities are returned
Immediate	Boolean	Iterates immediately through the results
Type	Integer	Type of entity to iterate
ArchetypeFilter	String	Returns archetype entities
Center	Vec3	Center of the sphere
Radius	Float	Distance from the center of the sphere to check for entities

Outputs

Port	Type	Description
OutEntityId	Any	Outputs the entity and entity ID
Count	Integer	Outputs the current of entities
Done	Integer	Triggered when all entities have been found, with the total count returned

JSON Nodes

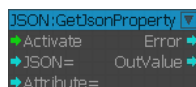
You can use these flow graph nodes to configure JSON settings.

Topics

- [GetJsonProperty node \(p. 648\)](#)
- [IsValueInJsonArray node \(p. 649\)](#)
- [IterateJsonArrayProperty node \(p. 649\)](#)
- [SetJsonProperty node \(p. 650\)](#)

GetJsonProperty node

Used to get the JSON attribute value.



Inputs

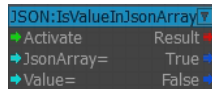
Port	Type	Description
Activate	Any	Triggers the node
JSON	String	The JSON code to parse
Attribute	String	The attribute to get the value of

Outputs

Port	Type	Description
Error	String	Triggers if the JSON could not be parsed or the attribute could not be found
OutValue	String	Outputs the attribute value

IsValueInJsonArray node

Used to look through a JSON array for the specified value.



Inputs

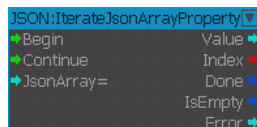
Port	Type	Description
Activate	Any	Triggers the node
JsonArray	String	The JSON array to search on
Value	String	The JSON value to search for

Outputs

Port	Type	Description
Result	Integer	Outputs the number of occurrences found
True	Boolean	Triggers if the value was found
False	Boolean	Triggers if the value was not found

IterateJsonArrayProperty node

Used to iterate through a JSON array, returning one element at a time.



Inputs

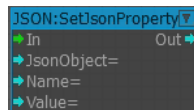
Port	Type	Description
Begin	Any	Starts iterating over the supplied JSON array
Continue	Any	Continues iterating over the supplied JSON array
JsonArray	String	The JSON array to iterated over

Outputs

Port	Type	Description
Value	Float	Value of the current array element
Index	Integer	Index of the current array element
Done	Any	Triggers when there are no more elements in the array
IsEmpty	Boolean	Triggers if the array is empty
Error	String	Triggers if an error occurs

SetJsonProperty node

Used to set a property on a JSON object.



Inputs

Port	Type	Description
In	Any	Activates the node
JsonObject	String	The JSON object to set the property on
Name	String	Name of the JSON property
Value	String	Value of the JSON property

Outputs

Port	Type	Description
Out	String	Outputs the JSON object

Kinect Nodes

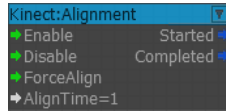
You can use these flow graph nodes to configure Kinect settings.

Topics

- [Alignment node \(p. 651\)](#)
- [Skeleton node \(p. 651\)](#)

Alignment node

Used to get the default Kinect skeleton joint lengths when a new closest tracked skeleton is detected.



Inputs

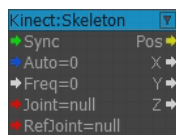
Port	Type	Description
Enable	Any	Enables the skeleton alignment watcher
Disable	Any	Disables the skeleton alignment watcher
ForceAlign	Any	Forces the beginning of the new skeleton alignment phase
AlignTime	Float	The time spent each time the skeleton alignment is started

Outputs

Port	Type	Description
Started	Boolean	Triggers when a new skeleton alignment has started
Completed	Boolean	Triggers when a skeleton alignment completes

Skeleton node

Used to get the status of the joints for a Kinect skeleton.



Inputs

Port	Type	Description
Sync	Any	Activates the node
Auto	Boolean	Forces an auto update
Freq	Float	Auto update frequency. Use 0 to update every frame.
Joint	Integer	The skeleton joint
RefJoint	Integer	The skeleton reference joint

Outputs

Port	Type	Description
Pos	Vec3	Outputs the skeleton vector position
X	Float	Outputs the skeleton X-axis position
Y	Float	Outputs the skeleton Y-axis position
Z	Float	Outputs the skeleton Z-axis position

Logic Nodes

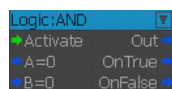
You can use the following flow graph nodes to define logic operations.

Topics

- [AND node \(p. 652\)](#)
- [All node \(p. 653\)](#)
- [Any node \(p. 653\)](#)
- [Blocker node \(p. 654\)](#)
- [CountBlocker node \(p. 654\)](#)
- [DeMultiplexer node \(p. 655\)](#)
- [Gate node \(p. 655\)](#)
- [IfCondition node \(p. 656\)](#)
- [Indexer node \(p. 656\)](#)
- [Multiplexer node \(p. 657\)](#)
- [NOT node \(p. 657\)](#)
- [OR node \(p. 658\)](#)
- [OnChange node \(p. 658\)](#)
- [Once node \(p. 659\)](#)
- [OnceNoSerialize node \(p. 659\)](#)
- [RandomSelect node \(p. 659\)](#)
- [RandomTrigger node \(p. 660\)](#)
- [SelectCondition node \(p. 661\)](#)
- [Sequencer node \(p. 661\)](#)
- [XOR node \(p. 662\)](#)

AND node

Used to perform a logical AND operation on the input ports. Output is true if both inputs are true.



Inputs

Port	Type	Description
Activate	Any	Triggers the node

Port	Type	Description
A	Boolean	First input
B	Boolean	Second input

Outputs

Port	Type	Description
Out	Boolean	Output value
OnTrue	Boolean	Triggers if Out is true
OnFalse	Boolean	Triggers if Out is false

All node

Used to trigger the output when all connected inputs are triggered.



Inputs

Port	Type	Description
In0 - In7	Any	Input values
Reset	Any	Resets the input values to 0

Outputs

Port	Type	Description
Out	Any	Triggered when all inputs are triggered

Any node

Used to trigger the output when any of the connected inputs are triggered.



Inputs

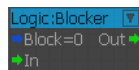
Port	Type	Description
In0 - In9	Any	Input values

Outputs

Port	Type	Description
Out	Any	Triggered when any inputs are triggered

Blocker node

Used to block or pass signals depending on the the state of the Block input.



Inputs

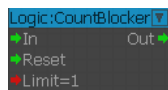
Port	Type	Description
Block	Boolean	If true, blocks In signal
In	Any	Input signal

Outputs

Port	Type	Description
Out	Any	If Block is false, outputs In signal. If Block is true, In signal is blocked.

CountBlocker node

Used to output a signal a number of times as defined by the Limit input.



Inputs

Port	Type	Description
In	Any	Input value
Reset	Any	Resets In to 0
Limit	Integer	Number of times In is sent to Out

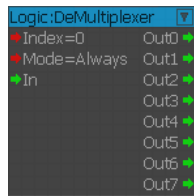
Outputs

Port	Type	Description
Out	Any	Passes In a limited number of times as defined by Limit

DeMultiplexer node

Used to send the In input to the selected Out output, based on the value of the Mode input:

- **Always:** Both the In and Index inputs activate the output.
- **IndexOnly:** Only the Index input activates the output.
- **InputOnly:** Only the In port activates the output.



Inputs

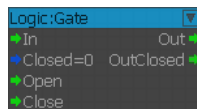
Port	Type	Description
Index	Integer	Determines which output receives the input (In)
Mode	Integer	Determines when the outputs are activated
In	Any	Input value

Outputs

Port	Type	Description
Out0 - Out7	Any	Outputs that can be triggered

Gate node

Used to block or pass a signal depending on the state of the Closed input.



Inputs

Port	Type	Description
In	Any	Input value
Closed	Boolean	If true, blocks the input from passing to the output

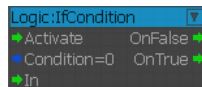
Port	Type	Description
Open	Any	Sets Closed to false
Close	Any	Sets Closed to true

Outputs

Port	Type	Description
Out	Any	Output value
OutClosed	Any	Output if Closed is true

IfCondition node

Used to output signals based on whether the Condition input is enabled.



Inputs

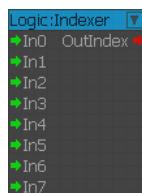
Port	Type	Description
Activate	Any	Triggers the node
Condition	Boolean	Condition value
In	Any	Input value

Outputs

Port	Type	Description
OnFalse	Any	Triggers if Condition is false
OnTrue	Any	Triggers if Condition is true

Indexer node

Used to return the index of an active input. Does not account for multiple activations on different inputs.



Inputs

Port	Type	Description
In0 - In7	Any	Input values

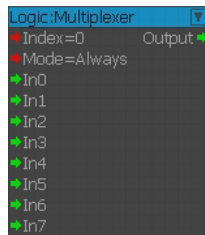
Outputs

Port	Type	Description
OutIndex	Integer	Outputs the index (number) of the active input

Multiplexer node

Used to select an input and send it to the output, based on the value of the Mode input:

- **Always:** Both the In and Index inputs activate the output.
- **IndexOnly:** Only the Index input activates the output.
- **InputOnly:** Only the In port activates the output.



Inputs

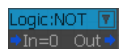
Port	Type	Description
Index	Integer	Determines which input is passed to the output
Mode	Integer	Determines which inputs activate the output
In0 - In7	Any	Input values

Outputs

Port	Type	Description
Output	Any	Output value

NOT node

Used to perform a logical NOT operation on the input ports. If the input is true, the output will be false and vice versa.



Inputs

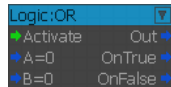
Port	Type	Description
In	Boolean	Input value

Outputs

Port	Type	Description
Out	Boolean	If the input is true, the output will be false and vice versa

OR node

Used to perform a logical OR operation on the input ports. The output is true if either of the two inputs is true.



Inputs

Port	Type	Description
Activate	Any	Triggers the node
A	Boolean	First input
B	Boolean	Second input

Outputs

Port	Type	Description
Out	Boolean	Output is true if either of the two inputs is true
OnTrue	Boolean	Triggers if Out is true
OnFalse	Boolean	Triggers if Out is false

OnChange node

Used to send the input value to the output when it is different from the previous value.



Inputs

Port	Type	Description
In	Boolean	Input value

Outputs

Port	Type	Description
Out	Boolean	Receives the input value when the input has changed from it's previous value

Once node

Used to pass the activated input to the output only once.



Inputs

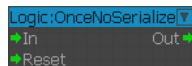
Port	Type	Description
In0 - In7	Any	Input values
Reset	Any	Resets the inputs and allows new activation to occur

Outputs

Port	Type	Description
Out	Any	Receives the active input only once

OnceNoSerialize node

Use to pass the activated input value to the output only once. The triggered flag is not serialized on a saved game. This means that even if a previous savegame is loaded after the node has been triggered, the node won't be triggered again.



Inputs

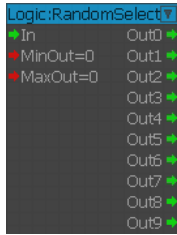
Port	Type	Description
In	Any	Input value
Reset	Any	Resets the input and allows new activation to occur

Outputs

Port	Type	Description
Out	Any	Receives the active input only once

RandomSelect node

Used to pass the activated input value to a random number of outputs.



Inputs

Port	Type	Description
In	Any	Input value
MinOut	Integer	Minimum number of outputs to trigger
MaxOut	Integer	Maximum number of outputs to trigger

Outputs

Port	Type	Description
Out0 - Out9	Any	Receives active input values

RandomTrigger node

Used to trigger one of the outputs in random order.



Inputs

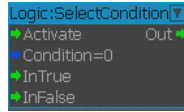
Port	Type	Description
In	Any	Input value
Reset	Any	Resets the activations to 0

Outputs

Port	Type	Description
Out0 - Out9	Any	Output value
Done	Any	Triggered when all outputs have been triggered

SelectCondition node

Used to trigger the output based on the state of the Condition node.



Inputs

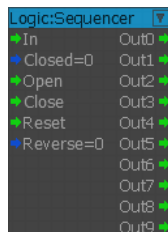
Port	Type	Description
Activate	Any	Triggers the node
Condition	Boolean	Condition value
InTrue	Any	Value sent to Out when Condition is true
InFalse	Any	Value sent to Out when Condition is false

Outputs

Port	Type	Description
Out	Any	Output value

Sequencer node

Used to trigger one of the outputs in sequential order for each input activation.



Inputs

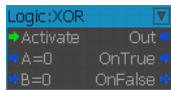
Port	Type	Description
In	Any	Input value
Closed	Boolean	If true, blocks all inputs
Open	Any	Sets Closed to false
Close	Any	Sets Closed to true
Reset	Any	Forces active output to Out0
Reverse	Boolean	If true, the order of output activation is reversed

Outputs

Port	Type	Description
Out0 - Out9	Any	Outputs are triggered in sequential order for each input activation

XOR node

Used to perform a logical XOR operation on the input ports. If one of the inputs is true, the output is true. If both inputs are true or are false, the output is false.



Inputs

Port	Type	Description
Activate	Any	Triggers the node
A	Boolean	First input
B	Boolean	Second input

Outputs

Port	Type	Description
Out	Boolean	If one of the inputs is true, the output is true. If both inputs are true or are false, the output is false
OnTrue	Boolean	Triggers if Out is true
OnFalse	Boolean	Triggers if Out is false

Material Nodes

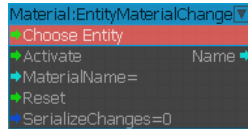
You can use the following flow graph nodes to define material settings.

Topics

- [EntityMaterialChange node \(p. 662\)](#)
- [EntityMaterialParams node \(p. 663\)](#)
- [MaterialClone node \(p. 664\)](#)
- [MaterialParams node \(p. 664\)](#)
- [SetObjectMaterial node \(p. 665\)](#)

EntityMaterialChange node

Used to apply the specified material to an entity.



Inputs

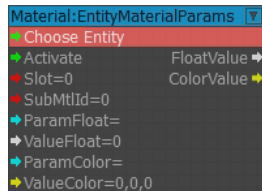
Port	Type	Description
Activate	Any	Activates the node
MaterialName	String	Name of material to apply
Reset	Any	Reset to the original material
SerializeChanges	Boolean	Serialize the change

Outputs

Port	Type	Description
Name	String	Outputs the name of the material

EntityMaterialParams node

Used to get the entity's material parameters.



Inputs

Port	Type	Description
Activate	Any	Activates the node
Slot	Integer	Material slot
SubMtlId	Integer	Submaterial ID
ParamFloat	String	Float parameter to be set
ValueFloat	Float	Sets float parameter value
ParamColor	String	Color parameter to be set
ValueColor	Vec3	Color value to be set

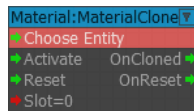
Outputs

Port	Type	Description
FloatValue	Float	Current float value

Port	Type	Description
ColorValue	Vec3	Current color value

MaterialClone node

Used to clone an entity's material or reset it back to the original.



Inputs

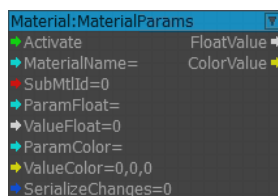
Port	Type	Description
Activate	Any	Activates the node
Reset	Any	Resets to the original material
Slot	Integer	Material slot

Outputs

Port	Type	Description
onCloned	Any	Activated when material is cloned
OnReset	Any	Activated when material is reset

MaterialParams node

Used to get the specified material's parameters.



Inputs

Port	Type	Description
Activate	Any	Activates the node
MaterialName	String	Material name
SubMtlId	Integer	Submaterial name
ParamFloat	String	Float parameter to be set

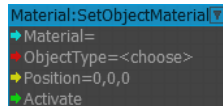
Port	Type	Description
ValueFloat	Float	Value of the float parameter
ParamColor	String	Color parameter to be set
ValueColor	Vec3	Value of the color parameter
SerializeChanges	Boolean	Serialize the change

Outputs

Port	Type	Description
FloatValue	Float	Current float value
ColorValue	Vec3	Current color value

SetObjectMaterial node

Used to set an object's (render node) material to the specified position.



Inputs

Port	Type	Description
Material	String	Set object material
ObjectType	Integer	Object type
Position	Vec3	Position to set material at
Activate	Any	Activates the node

MaterialFX Nodes

You can use the following flow graph nodes to define material FX settings.

Note

These nodes will only work with the Legacy Game Sample (CryEngine GameSDK), which is available at [Lumberyard Downloads](#).

Topics

- [HUDEndFX node \(p. 665\)](#)
- [HUDStartFX node \(p. 666\)](#)

HUDEndFX node

The MaterialFX end node for an HUD.

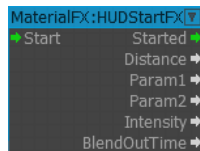


Inputs

Port	Type	Description
Trigger	Any	MaterialFX end node

HUDStartFX node

The MaterialFX start node for an HUD.



Inputs

Port	Type	Description
Start	Any	Triggered automatically by the material effect

Outputs

Port	Type	Description
Started	Any	Triggered when the material effect has started
Distance	Float	Outputs the distance to the player
Param1	Float	Custom parameter 1
Param2	Float	Custom parameter 2
Intensity	Float	Dynamic value set by game code
BlendOutTime	Float	Outputs the material effect blend out time in seconds

Math Nodes

You can use these flow graph nodes to define math operations.

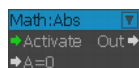
Topics

- [Abs node \(p. 667\)](#)
- [Add node \(p. 668\)](#)
- [AnglesToDir node \(p. 668\)](#)
- [ArcCos node \(p. 669\)](#)
- [ArcSin node \(p. 669\)](#)
- [ArcTan node \(p. 669\)](#)
- [ArcTan2 node \(p. 670\)](#)

- [BooleanFrom node \(p. 670\)](#)
- [BooleanTo node \(p. 671\)](#)
- [Calculate node \(p. 671\)](#)
- [Ceil node \(p. 672\)](#)
- [Clamp node \(p. 672\)](#)
- [Cosine node \(p. 672\)](#)
- [Counter node \(p. 673\)](#)
- [DirToAngles node \(p. 673\)](#)
- [Div node \(p. 674\)](#)
- [Equal node \(p. 674\)](#)
- [EvenOrOdd node \(p. 674\)](#)
- [Floor node \(p. 675\)](#)
- [InRange node \(p. 675\)](#)
- [Less node \(p. 676\)](#)
- [Mod node \(p. 676\)](#)
- [Mul node \(p. 677\)](#)
- [Noise1D node \(p. 677\)](#)
- [Noise3D node \(p. 678\)](#)
- [PortCounter node \(p. 678\)](#)
- [Power node \(p. 679\)](#)
- [Random node \(p. 679\)](#)
- [Reciprocal node \(p. 680\)](#)
- [Remainder node \(p. 680\)](#)
- [Round node \(p. 681\)](#)
- [SetColor node \(p. 681\)](#)
- [SetNumber node \(p. 681\)](#)
- [SinCos node \(p. 682\)](#)
- [Sine node \(p. 682\)](#)
- [Sqrt node \(p. 683\)](#)
- [Sub node \(p. 683\)](#)
- [Tangent node \(p. 684\)](#)
- [UpDownCounter node \(p. 684\)](#)

Abs node

Used to calculate the absolute value of the input.



Inputs

Port	Type	Description
Activate	Any	Triggers the node

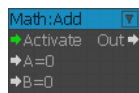
Port	Type	Description
A	Float	Input

Outputs

Port	Type	Description
Out	Float	Absolute value of the input

Add node

Used to add the two input values.



Inputs

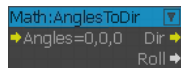
Port	Type	Description
Activate	Any	Triggers the node
A	Float	First operand
B	Float	Second operand

Outputs

Port	Type	Description
Out	Float	Absolute value of the input

AnglesToDir node

Used to convert the input angle to a unit vector direction.



Inputs

Port	Type	Description
Angles	Vec3	Input angle

Outputs

Port	Type	Description
Dir	Vec3	Direction unit vector

Port	Type	Description
Roll	Float	Roll output

ArcCos node

Used to calculate the inverse cosine of the input.



Inputs

Port	Type	Description
Activate	Any	Triggers the node
In	Float	Input angle

Outputs

Port	Type	Description
Activate	Any	Triggers the node
In	Float	Input angle

ArcSin node

Used to calculate the inverse sine of the input.



Inputs

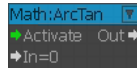
Port	Type	Description
Activate	Any	Triggers the node
In	Float	Input angle

Outputs

Port	Type	Description
Out	Float	Inverse sine (Arcsin) of the input

ArcTan node

Used to calculate the inverse tangent of the input.



Inputs

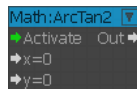
Port	Type	Description
Activate	Any	Triggers the node
In	Float	Input angle

Outputs

Port	Type	Description
Out	Float	Inverse tangent (Arctan) of the input

ArcTan2 node

Used to calculate the inverse tangent of the two inputs.



Inputs

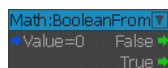
Port	Type	Description
Activate	Any	Triggers the node
X	Float	X input value
Y	Float	Y input value

Outputs

Port	Type	Description
Out	Float	Inverse tangent (Arctan) of the Y and X inputs

BooleanFrom node

Used to convert the Boolean input value (0 or 1) to true or false.



Inputs

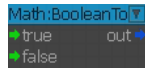
Port	Type	Description
Value	Boolean	Boolean input (0 or 1)

Outputs

Port	Type	Description
False	Float	Triggers if input is false (0)
True	Float	Triggers if input is true (1)

BooleanTo node

Used to convert the inputs to a Boolean 0 or 1 value.



Inputs

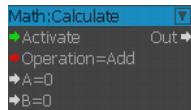
Port	Type	Description
True	Float	Will output true if event is received on this input
False	Float	Will output false if event is received on this input

Outputs

Port	Type	Description
Out	Boolean	Outputs true (1) or false (0) depending on input state

Calculate node

Used to calculate the output value based on the operation performed on the two inputs.



Inputs

Port	Type	Description
Activate	Any	Activates the node
Operation	Integer	The mathematical operation to be performed
A	Float	First operand
B	Float	Second operand

Outputs

Port	Type	Description
Out	Float	Result of operation on A and B

Ceil node

Used to output the ceiling value of the input.



Inputs

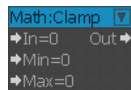
Port	Type	Description
In	Float	Input

Outputs

Port	Type	Description
Out	Float	Ceiling input value

Clamp node

Used to clamp the output value to the Min and Max range.



Inputs

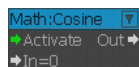
Port	Type	Description
In	Float	Input value
Min	Float	Minimum clamp value
Max	Float	Maximum clamp value

Outputs

Port	Type	Description
Out	Float	Triggered if the input is clamped within the range

Cosine node

Used to output the cosine of the input.



Inputs

Port	Type	Description
Activate	Any	Triggers the node

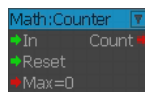
Port	Type	Description
In	Float	Input in degrees

Outputs

Port	Type	Description
Out	Float	Cosine of the input

Counter node

Used to output the number of times the input has been activated.



Inputs

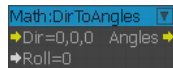
Port	Type	Description
In	Any	Input
Reset	Any	Resets the counter
Max	Integer	Maximum value of the counter before it is reset

Outputs

Port	Type	Description
Count	Integer	Number of times that the input was activated

DirToAngles node

Used to convert the input vector direction to an angle.



Inputs

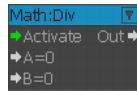
Port	Type	Description
Dir	Vec3	Vector direction
Roll	Float	Roll input

Outputs

Port	Type	Description
Angles	Vec3	Converts the direction to an angle in degrees

Div node

Used to divide input A by input B.



Inputs

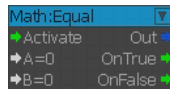
Port	Type	Description
Activate	Any	Triggers the node
A	Float	Dividend input
B	Float	Divisor input

Outputs

Port	Type	Description
Out	Float	Division of A by B

Equal node

Used to check if the two inputs are equal in value.



Inputs

Port	Type	Description
Activate	Any	Triggers the node
A	Float	First operand
B	Float	Second operand

Outputs

Port	Type	Description
Out	Boolean	True if the two inputs are equal in value
OnTrue	Any	Triggered if the inputs are equal in value
OnFalse	Any	Triggered if the inputs are not equal in value

EvenOrOdd node

Used to check if the input is an even or odd value.



Inputs

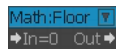
Port	Type	Description
In	Integer	Input

Outputs

Port	Type	Description
Odd	Any	Triggered if the input is an odd value
Even	Any	Triggered if the input is an even value

Floor node

Used to output the floor of the input.



Inputs

Port	Type	Description
In	Float	Input

Outputs

Port	Type	Description
Out	Float	Floored input

InRange node

Used to check if the input is within the Min and Max value range.



Inputs

Port	Type	Description
Activate	Any	Triggers the node
In	Float	Input
Min	Float	Minimum value of the range

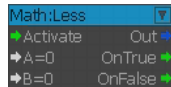
Port	Type	Description
Max	Float	maximum value of the range

Outputs

Port	Type	Description
Out	Boolean	True if the input is within the range
OnTrue	Any	Triggered if the input is within the range
OnFalse	Any	Triggered if the input is outside of the range

Less node

Used to check whether the A input is less than the B input.



Inputs

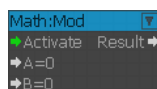
Port	Type	Description
Out	Boolean	True if A is less than B
OnTrue	Any	Triggered is A is less then B
OnFalse	Any	Triggered if A is greater than B

Outputs

Port	Type	Description
Out	Boolean	True if A is less than B
OnTrue	Any	Triggered is A is less then B
OnFalse	Any	Triggered if A is greater than B

Mod node

Used to calculate the modulus of the two inputs.



Inputs

Port	Type	Description
Activate	Any	Triggers the node

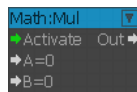
Port	Type	Description
A	Float	First operand
B	Float	Second operand

Outputs

Port	Type	Description
Result	Float	Modulus of the two inputs

Mul node

Used to multiply the two inputs.



Inputs

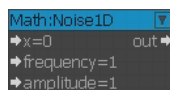
Port	Type	Description
Activate	Any	Triggers the node
A	Float	First operand
B	Float	Second operand

Outputs

Port	Type	Description
Out	Float	Multiplication of the two inputs

Noise1D node

Used to multiply the scalar input by the frequency and amplitude.



Inputs

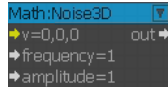
Port	Type	Description
X	Float	Scalar Input value to sample noise at
Frequency	Float	Frequency
Amplitude	Float	Amplitude

Outputs

Port	Type	Description
Out	Float	Multiplication of X by Frequency and Amplitude values

Noise3D node

Used to multiple the vector input by the frequency and amplitude.



Inputs

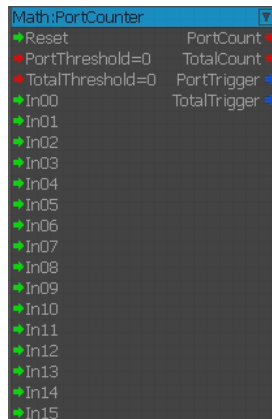
Port	Type	Description
V	Vec3	Vector input value to sample noise at
Frequency	Float	Frequency
Amplitude	Float	Amplitude

Outputs

Port	Type	Description
Out	Float	Multiplication of V by Frequency and Amplitude values

PortCounter node

Used to count the number of activated inputs.



Inputs

Port	Type	Description
Reset	Any	Resets PortCount and TotalCount

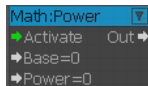
Port	Type	Description
PortThreshold	Integer	PortCount threshold value
TotalThreshold	Integer	TotalCount threshold value
In00 - In15	Any	Inputs

Outputs

Port	Type	Description
PortCount	Integer	Number of ports that have been set
TotalCount	Integer	Sum of all times any of the input ports have been set
PortTrigger	Boolean	Triggered when PortCount reaches PortThreshold
TotalTrigger	Boolean	Triggered when TotalCount reaches TotalThreshold

Power node

Used to calculate the Base input raised to the Power exponent.



Inputs

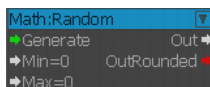
Port	Type	Description
Activate	Any	Triggers the node
Base	Float	Base input
Power	Float	Exponent input

Outputs

Port	Type	Description
Out	Float	Base input value raised to the Power exponent

Random node

Used to generate a random number between the Min and Max values, both as an integer and as a floating point number.



Inputs

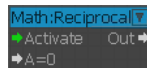
Port	Type	Description
Generate	Any	Generates a random number
Min	Float	Minimum value of the random number
Max	Float	Maximum value of the random number

Outputs

Port	Type	Description
Out	Float	Output as floating-point number
OutRounded	Integer	Output rounded to next integer value

Reciprocal node

Used to output the reciprocal value of the input.



Inputs

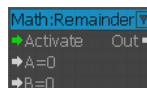
Port	Type	Description
Activate	Any	Triggers the node
A	Float	Input

Outputs

Port	Type	Description
Out	Float	Reciprocal of the input

Remainder node

Used to output the remainder value of A divided by B.



Inputs

Port	Type	Description
Activate	Any	Triggers the node
A	Float	Dividend input
B	Float	Divisor input

Outputs

Port	Type	Description
Out	Float	Remainder of the inputs

Round node

Used to round the input floating point value to an integer output.



Inputs

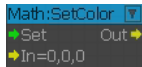
Port	Type	Description
Activate	Any	Triggers the node
In	Float	Floating-point Input

Outputs

Port	Type	Description
OutRounded	Integer	Rounded integer value of the input

SetColor node

Used to output the input vector color when the Set input is activated.



Inputs

Port	Type	Description
Set	Any	Triggers input to output
In	Vec3	Vector input value

Outputs

Port	Type	Description
Out	Vec3	Input value when Set is triggered

SetNumber node

Used to output the input scalar number when the Set input is activated.



Inputs

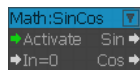
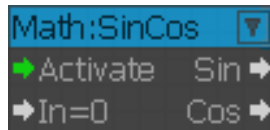
Port	Type	Description
Activate	Any	Triggers the node
In	Float	Input

Outputs

Port	Type	Description
Out	Float	Outputs the input

SinCos node

Used to calculate the sine and cosine of the input.



Inputs

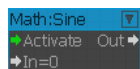
Port	Type	Description
Activate	Any	Triggers the node
In	Float	Input angle in degrees

Outputs

Port	Type	Description
Sin	Float	Sine of the input
Cos	Float	Cosine of the input

Sine node

Used to calculate the sine of the input.



Inputs

Port	Type	Description
Activate	Any	Triggers the node

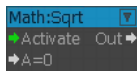
Port	Type	Description
In	Float	Input angle in degrees

Outputs

Port	Type	Description
Out	Float	Sine of the input

Sqrt node

Used to calculate the square root of the input.



Inputs

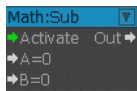
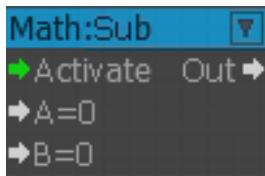
Port	Type	Description
Activate	Any	Triggers the node
A	Float	Input

Outputs

Port	Type	Description
Out	Float	Square root of the input

Sub node

Used to subtract the two inputs.



Inputs

Port	Type	Description
Activate	Any	Triggers the node
A	Float	First operand

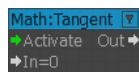
Port	Type	Description
B	Float	Second operand

Outputs

Port	Type	Description
Out	Float	Subtraction of the two inputs

Tangent node

Used to calculate the tangent of the input.



Inputs

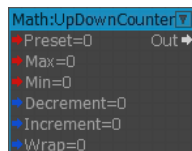
Port	Type	Description
Activate	Any	Triggers the node
In	Float	Input angle in degrees

Outputs

Port	Type	Description
Out	Float	Tangent of the inpt

UpDownCounter node

Used to output an up or down counter.



Inputs

Port	Type	Description
Preset	Integer	Preset input value
Max	Integer	Maximum counter limit
Min	Integer	Minimum counter limit
Decrement	Boolean	Decrements the count
Increment	Boolean	Increments the count
Wrap	Boolean	If true, the counter will wrap

Outputs

Port	Type	Description
Out	Float	Current count

Mission Nodes

You can use these flow graph nodes to configure mission-related settings. Game tokens are useful as variables used for passing data between flow graphs or within a flow graph, or for storing data between levels.

Topics

- [GameToken node \(p. 685\)](#)
- [GameTokenCheck node \(p. 686\)](#)
- [GameTokenCheckMulti node \(p. 686\)](#)
- [GameTokenGet node \(p. 687\)](#)
- [GameTokenModify node \(p. 687\)](#)
- [GameTokenSet node \(p. 688\)](#)
- [GameTokensLevelToLevelRestore node \(p. 688\)](#)
- [GameTokensLevelToLevelStore node \(p. 688\)](#)
- [LoadNextLevel node \(p. 689\)](#)

GameToken node

Used to get or set a game token. This is the most important and useful of all the mission nodes as it acts like a listener for any changes on the input.



Inputs

Port	Type	Description
Token	String	Game token to compare. Any change in this value will trigger the TokenValue output.
CompareValue	String	Value to compare the token value against

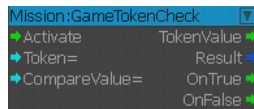
Outputs

Port	Type	Description
TokenValue	Any	Value of the game token. Triggers whenever the Token input changes value.
OnTrue	Boolean	Triggered if the token value is equal to CompareValue

Port	Type	Description
OnFalse	Boolean	Triggered if the token value is not equal to CompareValue

GameTokenCheck node

Used to check if the value of a game token equals a value.



Inputs

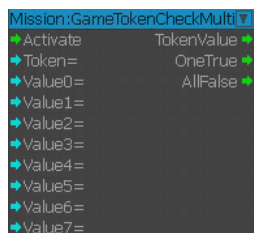
Port	Type	Description
Activate	Any	Activates the node
Token	String	Game token to check
CompareValue	String	Value to compare the token value against

Outputs

Port	Type	Description
TokenValue	Any	Value of the token
Result	Boolean	True if the token value is equal to CompareValue
OnTrue	Any	Triggered if the token value is equal to CompareValue
OnFalse	Any	Triggered if the token value is not equal to CompareValue

GameTokenCheckMulti node

Used to check if a game token is equal to any value in a list.



Inputs

Port	Type	Description
Activate	Any	Activates the node

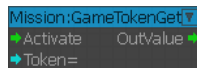
Port	Type	Description
Token	String	Game token to check
Value0 - Value7	String	Values to compare the token value with

Outputs

Port	Type	Description
TokenValue	Any	Value of the token
OneTrue	Any	Triggered if the token value is equal to at least one of the input port values
AllFalse	Any	Triggered if the token value is not equal to any of the input port values

GameTokenGet node

Used to get the value of the game token.



Inputs

Port	Type	Description
Activate	Any	Activates the node
Token	String	Game token to get

Outputs

Port	Type	Description
OutValue	Any	Displays value of the game token

GameTokenModify node

Used to modify the value of a game token.



Inputs

Port	Type	Description
Activate	Any	Activates the node
Token	String	Game token to set

Port	Type	Description
Operation	Integer	Operation to perform on the token
TokenType	Integer	Token type
OtherValue	String	Value to perform operation with

Outputs

Port	Type	Description
Result	Any	Result of the operation

GameTokenSet node

Used to set the value of a game token.



Inputs

Port	Type	Description
Activate	Any	Activates the node
Token	String	Game token to set
TokenValue	String	Value of token

Outputs

Port	Type	Description
OutValue	Any	Outputs token value

GameTokensLevelToLevelRestore node

Used to restore the values of all game tokens in a level that were stored in the previous level using the **GameTokensLevelToLevelStore** node.



Inputs

Port	Type	Description
Activate	Any	Activates the node

GameTokensLevelToLevelStore node

Used to store the values of all game tokens in a level.

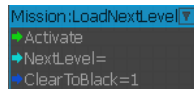


Inputs

Port	Type	Description
Activate	Any	Activates the node
Token0 - Token7	String	Stores token values

LoadNextLevel node

Used to load the next level.



Inputs

Port	Type	Description
Activate	Any	Activates the node
NextLevel	String	Ends the current level and loads the next level
ClearToBlack	Boolean	

Module Nodes

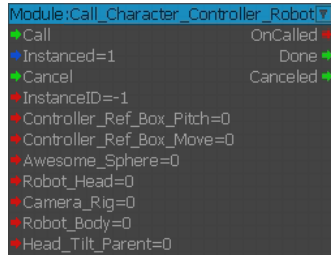
Module nodes are project-specific, user-created nodes. The nodes listed here are used in the Samples projects, which are located at *lumberyard_root_folder*\dev\SamplesProject\Levels\Samples.

Topics

- [Call_Character_Controller_Robot node \(p. 689\)](#)
- [Call_Character_Controller_Robot_Completed node \(p. 690\)](#)
- [Call_Free_Cam_Controller node \(p. 691\)](#)
- [Call_VR_Character_Controller_Robot node \(p. 692\)](#)
- [Utils:UserIDToModuleID node \(p. 693\)](#)

Call_Character_Controller_Robot node

Used to call a character's controller robot.



Inputs

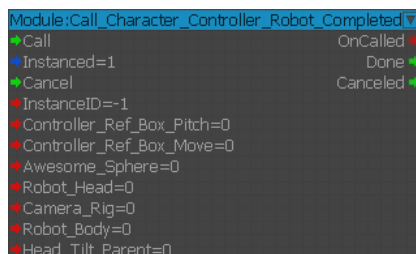
Port	Type	Description
Call	Any	Calls the module
Instanced	Boolean	Whether the module is instanced or not.
Cancel	Any	Cancels the module
InstanceID	Integer	Instance ID
Controller_Ref_Box_Pitch	Integer	Integer
Controller_Ref_Box_Move	Integer	Integer
Awesome_Sphere	Integer	Integer
Robot_Head	Integer	Integer
Camera_Rig	Integer	Integer
Robot_Body	Integer	Integer
Head_Tilt_Parent	Integer	Integer

Outputs

Port	Type	Description
OnCalled	Integer	Triggers when module is started
Done	Any	Successful status
Cancelled	Any	Failed status

Call_Character_Controller_Robot_Completed node

Used to call a character's controller robot.



Inputs

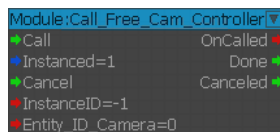
Port	Type	Description
Call	Any	Calls the module
Instanced	Boolean	Whether the module is instanced or not.
Cancel	Any	Cancels the module
InstanceID	Integer	Instance ID
Controller_Ref_Box_Pitch	Integer	Integer
Controller_Ref_Box_Move	Integer	Integer
Awesome_Sphere	Integer	Integer
Robot_Head	Integer	Integer
Camera_Rig	Integer	Integer
Robot_Body	Integer	Integer
Head_Tilt_Parent	Integer	Integer

Outputs

Port	Type	Description
OnCalled	Integer	Triggers when module is started
Done	Any	Successful status
Cancelled	Any	Failed status

Call_Free_Cam_Controller node

Used to call a camera controller.



Inputs

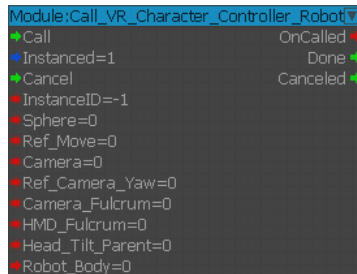
Port	Type	Description
Call	Any	Calls the module
Instanced	Boolean	Whether the module is instanced or not.
Cancel	Any	Cancels the module
InstanceID	Integer	Instance ID
Entity_ID_Camera	Integer	Integer

Outputs

Port	Type	Description
OnCalled	Integer	Triggers when module is started
Done	Any	Successful status
Cancelled	Any	Failed status

Call_VR_Character_Controller_Robot node

Used to call a VR character's controller robot.



Inputs

Port	Type	Description
Call	Any	Calls the module
Instanced	Boolean	Whether the module is instanced or not.
Cancel	Any	Cancel the module
InstanceID	Integer	Instance ID
Sphere	Integer	Integer
Ref_Move	Integer	Integer
Camera	Integer	Integer
Ref_Camera_Yaw	Integer	Integer
Camera_Fulcrum	Integer	Integer
HMD_Fulcrum	Integer	Integer
Head_Tilt_Parent	Integer	Integer
Robot_Body	Integer	Integer

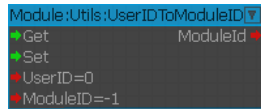
Outputs

Port	Type	Description
OnCalled	Integer	Triggers when module is started
Done	Any	Successful status

Port	Type	Description
Cancelled	Any	Failed status

Utils:UserIDToModuleID node

Used to map a user ID to a module instance ID.



Inputs

Port	Type	Description
Get	Any	Gets the module instance ID for the user ID
Set	Any	Gets the module instance ID for the user ID
UserID	Integer	User ID
ModuleID	Integer	Module instance ID

Outputs

Port	Type	Description
ModuleID	Integer	Module instance ID for the user ID

Movement Nodes

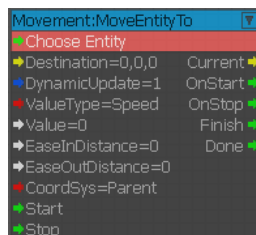
You can use the following flow graph nodes to specify entity movements.

Topics

- [MoveEntityTo node \(p. 693\)](#)
- [RotateEntity node \(p. 694\)](#)
- [RotateEntityTo node \(p. 695\)](#)

MoveEntityTo node

Used to move an entity to a destination position at a defined speed or in a defined interval of time.



Inputs

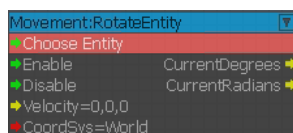
Port	Type	Description
Destination	Vec3	Position of the destination.
DynamicUpdate	Boolean	Indicates if destination position is to be followed if it changes.
ValueType	Integer	Type of input: Speed, Time,
Value	Float	Speed (m/sec) or Time (sec) value
EaseInDistance	Float	Distance from destination at which the entity starts slowing down
EaseOutDistance	Float	Distance from destination at which the entity starts speeding up
CoordSys	Integer	Coordinate system of the destination: Parent, World, or Local.
Start	Any	Starts movement
Stop	Any	Stops movement

Outputs

Port	Type	Description
Current	Vec3	Current position
OnStart	Any	Activated when Start is triggered
OnStop	Any	Activated when Stop is triggered
Finish	Any	Activated when destination is reached
Done	Any	Activated when destination is reached or Stop is triggered.

RotateEntity node

Used to rotate an entity at a defined speed.



Inputs

Port	Type	Description
Enable	Any	Enables updates
Disable	Any	Disables updates

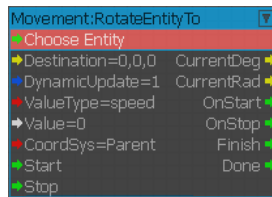
Port	Type	Description
Velocity	Vec3	Angular velocity (degrees/sec)
CoordSys	Integer	Coordinate system for rotation: World, Local

Outputs

Port	Type	Description
CurrentDegrees	Vec3	Current rotation in degrees
CurrentRadians	Vec3	Current rotation in radians

RotateEntityTo node

Used to rotate an entity at a defined speed or in a defined interval of time.



Inputs

Port	Type	Description
Destination	Vec3	Destination position (in degrees)
DynamicUpdate	Boolean	If dynamic updates are enabled or not
ValueType	Integer	Type of input value: Speed (m/sec) or Time (sec)
Value	Float	Value of Speed or Time
CoordSys	Integer	Coordinate system of the destination: Parent, World, Local
Start	Any	Starts movement
Stop	Any	Stops movement

Outputs

Port	Type	Description
CurrentDeg	Vec3	Current rotation in degrees
CurrentRad	Vec3	Current rotation in radians
OnStart	Any	Activated when Start input is triggered
OnStop	Any	Activated when Stop input is triggered
Finish	Any	Activated when destination rotation is reached

Port	Type	Description
Done	Any	Activated when destination rotation is reached or Stop is triggered

Physics Nodes

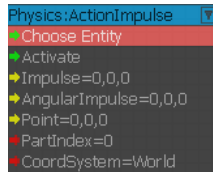
You can use the following flow graph nodes to configure physics.

Topics

- [ActionImpulse node \(p. 696\)](#)
- [CameraProxy node \(p. 696\)](#)
- [CollisionListener node \(p. 697\)](#)
- [Constraint node \(p. 698\)](#)
- [Dynamics node \(p. 699\)](#)
- [PhysicsEnable node \(p. 699\)](#)
- [PhysicsSleepQuery node \(p. 700\)](#)
- [RayCast node \(p. 700\)](#)
- [RaycastCamera node \(p. 701\)](#)

ActionImpulse node

Used to apply an impulse to an entity.

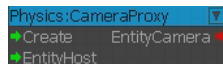


Inputs

Port	Type	Description
Activate	Any	Activates the node
Impulse	Vec3	Impulse vector
AngularImpulse	Vec3	Angular impulse vector
Point	Vec3	Location impulse is applied at
PartIndex	Integer	Part index
CoordSystem	Integer	Coordinate system used

CameraProxy node

Used to create a entity camera proxy.



Inputs

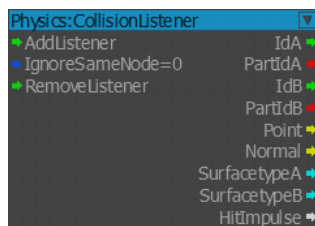
Port	Type	Description
Create	Any	Creates a physicalized camera proxy if one does not exist
EntityHost	Any	Syncs proxy rotation with the current view camera

Outputs

Port	Type	Description
EntityCamera	Integer	Retrieves the camera proxy

CollisionListener node

Used to setup physics collision listeners.



Inputs

Port	Type	Description
AddListener	Any	Adds collision listener
IgnoreSameNode	Boolean	Suppresses events if both colliders are registered via the same node
RemoveListener	Any	Removes collision listener

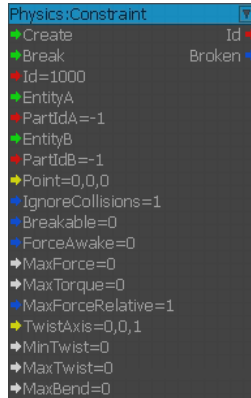
Outputs

Port	Type	Description
IdA	Any	ID of the first colliding entity
PartIdA	Integer	Part ID inside the first colliding entity
IdB	Any	ID of the second colliding entity
PartIdB	Integer	Part ID inside the second colliding entity
Point	Vec3	Location of collision point
Normal	Vec3	Collision normal
SurfaceTypeA	String	Surface type of the first colliding entity

Port	Type	Description
SurfacetypB	String	Surface type of the second colliding entity
HitImpulse	Float	Collision impulse along the normal

Constraint node

Used to create a physics constraint.



Inputs

Port	Type	Description
Create	Any	Creates the constraint
Break	Any	Breaks the constraint
Id	Integer	Constraint ID
EntityA	Any	Constraint owner entity
PartIdA	Integer	Part ID to attach to
EntityB	Any	Constraint buddy entity
PartIdB	Integer	Part ID to attach to
Point	Vec3	Connection point in worldspace
IgnoreCollisions	Boolean	Disables collisions between constrained entities
Breakable	Boolean	Break if force limit is reached
ForceAwake	Boolean	Make entity B always awake; restores previous sleep parameters
MaxForce	Float	Force limit
MaxTorque	Float	Rotational force (torque) force limit
MaxForceRelative	Any	Make limits relative to entity B's mass
TwistAxis	Boolean	Main rotation axis in worldspace

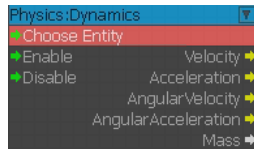
Port	Type	Description
MinTwist	Float	Lower rotation limit around TwistAxis
MaxTwist	Float	Upper rotation limit around TwistAxis
MaxBend	Float	Maximum bend of the TwistAxis

Outputs

Port	Type	Description
Id	Integer	Constraint ID
Broken	Boolean	Triggered when the constraint breaks

Dynamics node

Used to output the dynamic state of an entity.



Inputs

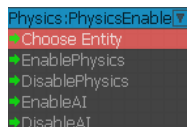
Port	Type	Description
Enable	Any	Enables updates
Disable	Any	Disables updates

Outputs

Port	Type	Description
Velocity	Vec3	Velocity of entity
Acceleration	Vec3	Acceleration of entity
AngularVelocity	Vec3	Angular velocity of entity
AngularAcceleration	Vec3	Angular acceleration of entity
Mass	Float	Mass of entity

PhysicsEnable node

Used to enable and disable physics.

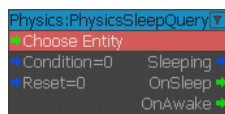


Inputs

Port	Type	Description
EnablePhysics	Any	Enables physics for entity
DisablePhysics	Any	Disables physics for entity
EnableAI	Any	Enables AI for entity
DisableAI	Any	Disables AI for entity

PhysicsSleepQuery node

Used to return the sleeping state of the physics of a given entity.



Inputs

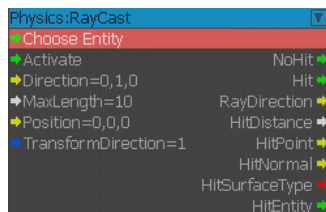
Port	Type	Description
Condition	Boolean	Sleeping state of the entity
Reset	Boolean	Resets the node

Outputs

Port	Type	Description
Sleeping	Boolean	Sleeping state of the entity
OnSleep	Any	Triggered when the entity physics switches to sleep
OneAwake	Any	Triggered when the entity physics switches to awake

RayCast node

Used to perform a raycast relative to an entity.



Inputs

Port	Type	Description
Activate	Any	Activates the node

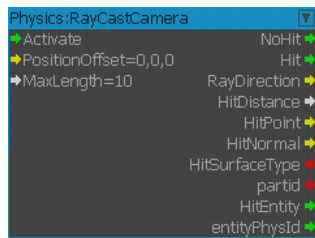
Port	Type	Description
Direction	Vec3	Direction of the raycast
MaxLength	Float	Maximum length of the raycast
Position	Vec3	Ray start position relative to the entity
TransformDirection	Boolean	Transforms direction by entity orientation

Outputs

Port	Type	Description
NoHit	Any	Triggered if no object was hit by the raycast
Hit	Any	Triggered if an object was hit by the raycast
RayDirection	Vec3	Direction of the cast ray
HitDistance	Float	Distance to the hit object
HitPoint	Vec3	Position of the hit
HitNormal	Vec3	Normal of the surface at the HitPoint
HitSurfaceType	Integer	Surface type index of the surface hit
HitEntity	Any	ID of the entity that was hit

RaycastCamera node

Used to perform a raycast relative to a camera.



Inputs

Port	Type	Description
Activate	Any	Activates the node
PositionOffset	Vec3	Ray start position relative to the camera
MaxLength	Float	Maximum length of the raycast

Outputs

Port	Type	Description
NoHit	Any	Triggered if no object was hit by the raycast

Port	Type	Description
Hit	Any	Triggered if an object was hit by the raycast
RayDirection	Vec3	Direction of the cast ray
HitDistance	Float	Distance to the hit object
HitPoint	Vec3	Position of the hit
HitNormal	Any	Normal of the surface at the HitPoint
HitSurfaceType	Integer	Surface type index of the surface hit
partid	Integer	Hit part ID
HitEntity	Any	ID of the entity that was hit
entityPhysId	Any	ID of the physical entity that was hit

Prefab Nodes

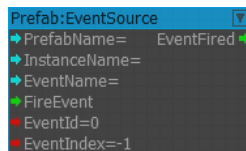
You can use the following flow graph nodes to configure prefab settings.

Topics

- [EventSource node \(p. 702\)](#)

EventSource node

Used to add an event source inside of a prefab for it to be handled like an instance.



Inputs

Port	Type	Description
PrefabName	String	Name of the prefab
InstanceName	String	Name of the prefab instance
EventName	String	Name of the event associated with the prefab
FireEvent	Any	Fires the associated event
EventId	Integer	ID of the event
EventIndex	Integer	Position of the event in the index

Outputs

Port	Type	Description
EventFired	Any	Triggered when the event has fired

ProceduralMaterial Nodes

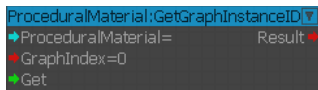
You can use the following flow graph nodes to configure procedural material settings.

Topics

- [GetGraphInstanceID node \(p. 703\)](#)
- [GetInputFloat node \(p. 704\)](#)
- [GetInputFloat2 node \(p. 704\)](#)
- [GetInputFloat3 node \(p. 704\)](#)
- [GetInputFloat4 node \(p. 705\)](#)
- [GetInput node \(p. 706\)](#)
- [GetInput2 node \(p. 706\)](#)
- [GetInput3 node \(p. 707\)](#)
- [GetInput4 node \(p. 707\)](#)
- [QueueGraphInstance node \(p. 708\)](#)
- [RenderASync node \(p. 708\)](#)
- [RenderSync node \(p. 709\)](#)
- [SetInputFloat node \(p. 709\)](#)
- [SetInputFloat2 node \(p. 709\)](#)
- [SetInputFloat3 node \(p. 710\)](#)
- [SetInputFloat4 node \(p. 711\)](#)
- [SetInputImage node \(p. 711\)](#)
- [SetInputInt node \(p. 712\)](#)
- [SetInputInt2 node \(p. 712\)](#)
- [SetInputInt3 node \(p. 713\)](#)
- [SetInputInt4 node \(p. 713\)](#)

GetGraphInstanceID node

Used to get the graph instance ID.



Inputs

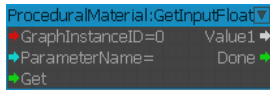
Port	Type	Description
ProceduralMaterial	String	Name of the procedural material
GraphicIndex	Integer	Graph index
Get	Any	Get the graph index

Outputs

Port	Type	Description
Result	Integer	Outputs the graph index

GetInputFloat node

Used to get the Substance input floating point value.



Inputs

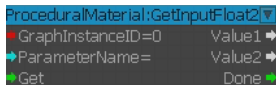
Port	Type	Description
GraphInstanceID	Integer	Graph instance ID
ParameterName	String	Parameter name
Get	Any	Get parameter value

Outputs

Port	Type	Description
Value1	Float	Outputs parameter value
Done	Any	Triggered when input completes

GetInputFloat2 node

Used to get the Substance input floating point values.



Inputs

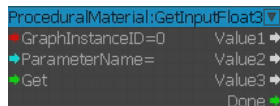
Port	Type	Description
GraphInstanceID	Integer	Graph instance ID
ParameterName	String	Parameter name
Get	Any	Get parameter value

Outputs

Port	Type	Description
Value1	Integer	Outputs parameter value 1
Value2	Integer	Outputs parameter value 2
Done	Any	Triggered when input completes

GetInputFloat3 node

Used to get the Substance input floating point values.



Inputs

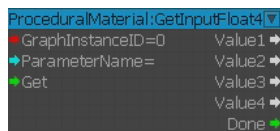
Port	Type	Description
GraphInstanceID	Integer	Graph instance ID
ParameterName	String	Parameter name
Get	Any	Get parameter value

Outputs

Port	Type	Description
Value1	Float	Outputs parameter value 1
Value2	Float	Outputs parameter value 2
Value3	Float	Outputs parameter value 3
Done	Any	Triggered when input completes

GetInputFloat4 node

Used to get the Substance input floating point values.



Inputs

Port	Type	Description
GraphInstanceID	Integer	Graph instance ID
ParameterName	String	Parameter name
Get	Any	Get parameter value

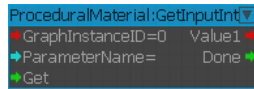
Outputs

Port	Type	Description
Value1	Float	Outputs parameter value 1
Value2	Float	Outputs parameter value 2
Value3	Float	Outputs parameter value 3
Value4	Float	Outputs parameter value 4

Port	Type	Description
Done	Any	Triggered when input completes

GetInput node

Used to get the Substance input value.



Inputs

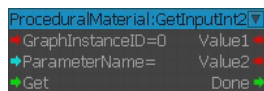
Port	Type	Description
GraphInstanceID	Integer	Graph instance ID
ParameterName	String	Parameter name
Get	Any	Get parameter value

Outputs

Port	Type	Description
Value1	Float	Outputs parameter value
Done	Any	Triggered when input completes

GetInput2 node

Used to get the Substance input value.



Inputs

Port	Type	Description
GraphInstanceID	Integer	Graph instance ID
ParameterName	String	Parameter name
Get	Any	Get parameter value

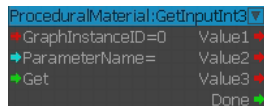
Outputs

Port	Type	Description
Value1	Float	Outputs parameter value 1

Port	Type	Description
Value2	Float	Outputs parameter value 2
Done	Any	Triggered when input completes

GetInput3 node

Used to get the Substance input value.



Inputs

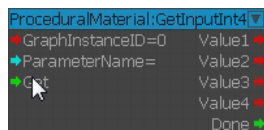
Port	Type	Description
GraphInstanceID	Integer	Graph instance ID
ParameterName	String	Parameter name
Get	Any	Get parameter value

Outputs

Port	Type	Description
Value1	Float	Outputs parameter value 1
Value2	Float	Outputs parameter value 2
Value3	Float	Outputs parameter value 3
Done	Any	Triggered when input completes

GetInput4 node

Used to get the Substance input value.



Inputs

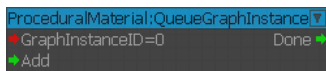
Port	Type	Description
GraphInstanceID	Integer	Graph instance ID
ParameterName	String	Parameter name
Get	Any	Get parameter value

Outputs

Port	Type	Description
Value1	Float	Outputs parameter value 1
Value2	Float	Outputs parameter value 2
Value3	Float	Outputs parameter value 3
Value4	Float	Outputs parameter value 4
Done	Any	Triggered when input completes

QueueGraphInstance node

Used to queue to graph instance.



Inputs

Port	Type	Description
GraphInstanceID	Integer	Graph instance ID
Add	Any	Add graph instance ID to the queue

Outputs

Port	Type	Description
Done	Any	Triggered when input completes

RenderASync node

Used to render queued graphs asynchronously.



Inputs

Port	Type	Description
Render	Any	Begin rendering graph instance asynchronously

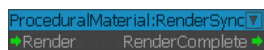
Outputs

Port	Type	Description
RenderBegin	Any	Triggered when rendering has started

Port	Type	Description
RenderComplete	Any	Triggered when rendering has completed

RenderSync node

Used to render queued graphs synchronously.



Inputs

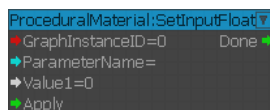
Port	Type	Description
Render	Any	Begin rendering graph instance synchronously

Outputs

Port	Type	Description
RenderComplete	Any	Triggered when rendering has completed

SetInputFloat node

Used to set the Substance input floating point value.



Inputs

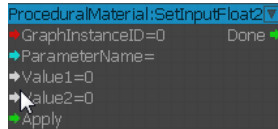
Port	Type	Description
GraphInstanceID	Integer	Graph instance ID
ParameterName	String	Parameter name
Value1	Float	Floating point parameter value to set
Apply	Any	Set parameter value

Outputs

Port	Type	Description
Done	Any	Triggered when input completes

SetInputFloat2 node

Used to set the Substance input floating point values.



Inputs

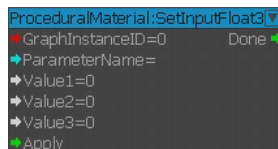
Port	Type	Description
GraphInstanceID	Integer	Graph instance ID
ParameterName	String	Parameter name
Value1	Float	Floating point parameter value 1 to set
Value2	Float	Floating point parameter value 2 to set
Apply	Any	Set parameter value

Outputs

Port	Type	Description
Done	Any	Triggered when input completes

SetInputFloat3 node

Used to set the Substance input floating point values.



Inputs

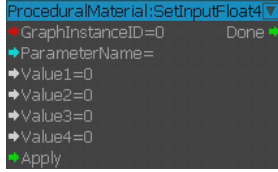
Port	Type	Description
GraphInstanceID	Integer	Graph instance ID
ParameterName	String	Parameter name
Value1	Float	Floating point parameter value 1 to set
Value2	Float	Floating point parameter value 2 to set
Value3	Float	Floating point parameter value 3 to set
Apply	Any	Set parameter value

Outputs

Port	Type	Description
Done	Any	Triggered when input completes

SetInputFloat4 node

Used to set the Substance input floating point values.



Inputs

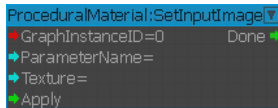
Port	Type	Description
GraphInstanceID	Integer	Graph instance ID
ParameterName	String	Parameter name
Value1	Float	Floating point parameter value 1 to set
Value2	Float	Floating point parameter value 2 to set
Value3	Float	Floating point parameter value 3 to set
Value4	Float	Floating point parameter value 4 to set
Apply	Any	Set parameter value

Outputs

Port	Type	Description
Done	Any	Triggered when input completes

SetInputImage node

Used to set the Substance input image.



Inputs

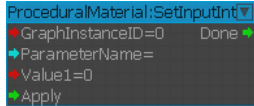
Port	Type	Description
GraphInstanceID	Integer	Graph instance ID
ParameterName	String	Parameter name
Texture	String	Image to be set
Apply	Any	Set input image

Outputs

Port	Type	Description
Done	Any	Triggered when input completes

SetInputInt node

Used to set the Substance input value.



Inputs

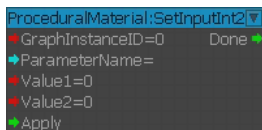
Port	Type	Description
GraphInstanceID	Integer	Graph instance ID
ParameterName	String	Parameter name
Value1	Float	Parameter value to set
Apply	Any	Set parameter value

Outputs

Port	Type	Description
Done	Any	Triggered when input completes

SetInputInt2 node

Used to set the Substance input values.



Inputs

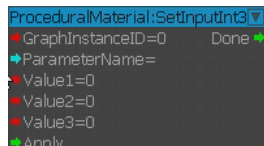
Port	Type	Description
GraphInstanceID	Integer	Graph instance ID
ParameterName	String	Parameter name
Value1	Float	Parameter value 1 to set
Value2	Float	Parameter value 2 to set
Apply	Any	Set parameter value

Outputs

Port	Type	Description
Done	Any	Triggered when input completes

SetInputInt3 node

Used to set the Substance input values.



Inputs

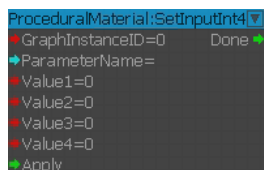
Port	Type	Description
GraphInstanceID	Integer	Graph instance ID
ParameterName	String	Parameter name
Value1	Float	Parameter value 1 to set
Value2	Float	Parameter value 2 to set
Value3	Float	Parameter value 3 to set
Apply	Any	Set parameter value

Outputs

Port	Type	Description
Done	Any	Triggered when input completes

SetInputInt4 node

Used to set the Substance input values.



Inputs

Port	Type	Description
GraphInstanceID	Integer	Graph instance ID

Port	Type	Description
ParameterName	String	Parameter name
Value1	Float	Parameter value 1 to set
Value2	Float	Parameter value 2 to set
Value3	Float	Parameter value 3 to set
Value4	Float	Parameter value 4 to set
Apply	Any	Set parameter value

Outputs

Port	Type	Description
Done	Any	Triggered when input completes

Stereo Nodes

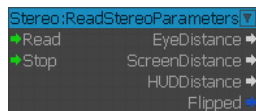
You can use the following flow graph nodes to configure stereographic settings.

Topics

- [ReadStereoParameters node \(p. 714\)](#)
- [StereoParameters node \(p. 715\)](#)

ReadStereoParameters node

Used to read the HUD stereo display parameters.



Inputs

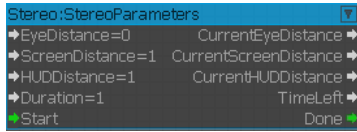
Port	Type	Description
Read	Any	Start reading stereo values
Stop	Any	Stop reading stereo values

Outputs

Port	Type	Description
EyeDistance	Float	Outputs eye distance
ScreenDistance	Float	Outputs screen distance
HUDDistance	Float	Outputs HUD distance
Flipped	Boolean	Output if stereo is flipped

StereoParameters node

Used to output the HUD stereo display parameters.



Inputs

Port	Type	Description
EyeDistance	Float	Sets the stereo eye distance
ScreenDistance	Float	Sets the stereo screen distance
HUDDistance	Float	Sets the stereo HUD distance
Duration	Float	Duration of the interpolation in seconds
Start	Any	Starts the interpolation

Outputs

Port	Type	Description
CurrentEyeDistance	Float	Outputs the current eye distance
CurrentScreenDistance	Float	Outputs the current screen distance
CurrentHUDDistance	Float	Outputs the current HUD distance
TimeLeft	Float	Time left to the end of the interpolation
Done	Any	Outputs when interpolation has completed

String Nodes

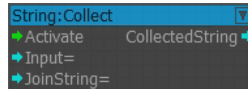
You can use the following flow graph nodes to configure strings.

Topics

- [Collect node \(p. 715\)](#)
- [Compare node \(p. 716\)](#)
- [Concat node \(p. 716\)](#)
- [ReplaceString node \(p. 717\)](#)
- [SetString node \(p. 717\)](#)
- [Split node \(p. 718\)](#)
- [URLDecode node \(p. 718\)](#)

Collect node

Used to collect a string.



Inputs

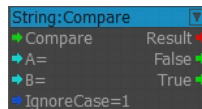
Port	Type	Description
Activate	Any	Collects the strings and triggers the output
Input	String	Each string that will be joined
JoinString	String	String to use between all collected strings

Outputs

Port	Type	Description
CollectedString	String	Outputs the collected string set

Compare node

Used to compare two strings.



Inputs

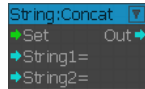
Port	Type	Description
Compare	Any	Triggers string comparison
A	String	First string to compare
B	String	Second string to compare
IgnoreCase	Boolean	Ignores casing

Outputs

Port	Type	Description
Result	Integer	Outputs -1 if string A less than string B, 0 if String A equals string B, 1 if string A is greater than string B
False	Any	Triggers if string A does not equal string B
True	Any	Triggers if string A equals string B

Concat node

Used to concatenate two strings.



Inputs

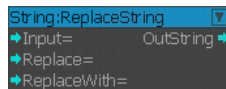
Port	Type	Description
Set	Any	Triggers string concatenation
String1	String	First string to concatenate
String2	String	Second string to concatenate

Outputs

Port	Type	Description
Out	String	Outputs the concatenated string

ReplaceString node

Used to replace a string.



Inputs

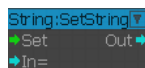
Port	Type	Description
Input	String	Triggers string replacement
Replace	String	The string to replace
ReplaceWith	String	The new string to replace with

Outputs

Port	Type	Description
OutString	String	Outputs the replaced string

SetString node

Used to set a string value.



Inputs

Port	Type	Description
Set	Any	Sends the string to the output

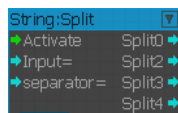
Port	Type	Description
In	String	String to set on

Outputs

Port	Type	Description
Out	String	Outputs the string value

Split node

Used to split a string.



Inputs

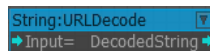
Port	Type	Description
Activate	Any	Triggers the string split
Input	String	The string to split
Separator	String	Character to separate the string on. If you pass a string, only the first character will be used

Outputs

Port	Type	Description
Split0 - Split4	String	Outputs the specific string split

URLDecode node

Used to decode the URL of a string.



Input

Port	Type	Description
Input	String	String to URL decode

Outputs

Port	Type	Description
DecodedString	String	Outputs the URL-decoded string

System Nodes

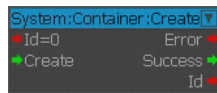
You can use the following flow graph nodes to configure system settings.

Topics

- [Container:Create node \(p. 719\)](#)
- [Container>Edit node \(p. 719\)](#)
- [Container:Iterate node \(p. 720\)](#)

Container:Create node

Used to create a container.



Inputs

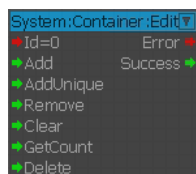
Port	Type	Description
Id	Integer	Container ID
Create	Any	Creates a container

Outputs

Port	Type	Description
Error	Integer	Triggers when an error occurs
Success	Any	Triggers when a container is created
Id	Integer	Outputs the container ID

Container>Edit node

Used to edit a container.



Inputs

Port	Type	Description
Id	Integer	Container ID
Add	Any	Adds the passed item to the container
AddUnique	Any	Adds the passed item if it didn't exist
Remove	Any	Removes all occurrences of the current item

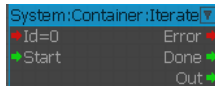
Port	Type	Description
Clear	Any	Empties the container
GetCount	Any	Gets the number of items in the container
Delete	Any	Deletes the container

Outputs

Port	Type	Description
Error	Integer	Triggers when an error occurs
Success	Any	Triggers when the operation successfully completed

Container:Iterate node

Used to iterate over a container.



Inputs

Port	Type	Description
Id	Integer	Container ID
Start	Any	Starts iterating the container

Outputs

Port	Type	Description
Error	Integer	Triggers when an error occurs
Done	Any	Triggers when the operation successfully completed
Out	Any	Outputs the container ID

Time Nodes

You can use these flow graph nodes to define time settings.

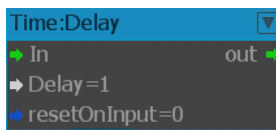
Topics

- [Delay](#) (p. 721)
- [FrameDelay](#) (p. 721)
- [MeasureTime](#) (p. 722)
- [RandomDelay](#) (p. 722)
- [RealTime](#) (p. 723)

- [ServerTime](#) (p. 724)
- [Time](#) (p. 724)
- [TimeOfDay](#) (p. 725)
- [TimeOfDayLoadDefinitionFile](#) (p. 726)
- [TimeOfDayTransitionTrigger](#) (p. 726)
- [TimeOfDayTrigger](#) (p. 728)
- [TimedCounter](#) (p. 728)
- [Timer](#) (p. 729)

Delay

Delays passing the signal from [In] to [Out] for the specified length of time (seconds).



Inputs

Port	Type	Description
In	Any	Value to pass after the specified delay time
Delay	Float	Delay time in seconds Default value: 1 Valid values: 0 – 100
resetOnInput	Boolean	When set to <code>true</code> , resets the node with each input, setting the delay counter to 0 and erasing previous inputs Default value: 0 Valid values: 0=false 1=true

Outputs

Port	Type	Description
out	Any	Value that is passed after the specified frame delay

FrameDelay

Delays passing the signal from [In] to [Out] for the specified number of frames.



Inputs

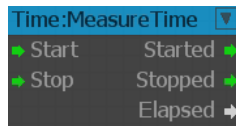
Port	Type	Description
In	Any	Value to pass after the specified delay time
NFrames	Integer	Number of frames to delay passing the signal from [In] to [Out] Default value: 1 Valid values: 0 – 100

Outputs

Port	Type	Description
out	Any	Value that is passed after the specified frame delay

MeasureTime

Measures the elapsed time.



Inputs

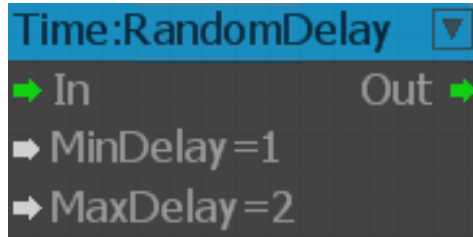
Port	Type	Description
Start	Any	Trigger to begin measuring time as it passes
Stop	Any	Trigger to stop measuring the elapsed time

Outputs

Port	Type	Description
Started	Any	Triggered on start
Stopped	Any	Triggered on stop
Elapsed	Any	Elapsed time in seconds

RandomDelay

Delays passing the signal from [In] to [Out] for a random amount of time (seconds) within the [MinDelay, MaxDelay] interval.



Inputs

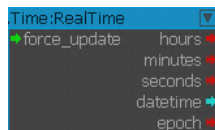
Port	Type	Description
In	Any	Value to pass after the specified delay time
MinDelay	Float	Minimum random delay time in seconds Default value: 1 Valid values: 0 – 100
MaxDelay	Float	Maximum random delay time in seconds Default value: 2 Valid values: 0 – 100

Outputs

Port	Type	Description
Out	Any	Value that is passed after the specified delay time

RealTime

Reads your system time. RealTime can be used to display time on screen (such as a player's watch) or synchronize the time of day with real world time.



Inputs

Port	Type	Description
force_update	Any	Forces an update of the system time

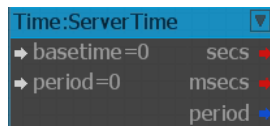
Outputs

Port	Type	Description
Hours	Integer	Current hour as reported by your system time
Minutes	Integer	Current minutes as reported by your system time

Port	Type	Description
Seconds	Integer	Current seconds as reported by your system time
Datetime	String	Outputs your system date and time
Epoch	Integer	Current epoch as reported by your system time

ServerTime

Reads the server time and reports the current time (seconds or milliseconds) for the specified period.



Inputs

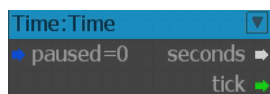
Port	Type	Description
Basetime	Float	Base time in seconds. The server time output is relative to the base time Default value: 0 Valid values: 0 – 100
Period	Float	Number of seconds that should pass before the timer resets to 0 Default value: 0 Valid values: 0 – 100

Outputs

Port	Type	Description
Secs	Integer	Current time in seconds, relative to the base time
Msecs	Integer	Current time in milliseconds, relative to the base time
Period	Boolean	Triggers the Period output once for each period of time, as specified by the Period input Valid values: 0=false 1=true

Time

Outputs the total number of seconds from the start of the game, ticking once per frame.



Inputs

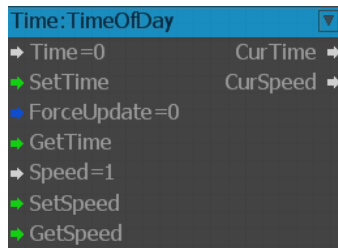
Port	Type	Description
Paused	Boolean	Pauses the time output when set to <code>true</code> . Default value: 0 Valid values: 0=false 1=true

Outputs

Port	Type	Description
seconds	Float	Current time in seconds
tick	Any	Triggers a tick once per frame

TimeOfDay

Changes the speed at which the time of day progresses and reads the current TimeOfDay setting.



Inputs

Port	Type	Description
Time	Float	Time of day in hours Default value: 0 Valid values: 0 – 24
SetTime	Any	Trigger to change the time of day to the value specified for the Time parameter
ForceUpdate	Boolean	Immediately updates the sky when set to <code>true</code> . Default value: 0 Valid values: 0=false 1=true
GetTime	Any	Retrigger the CurTime output without updating the value of the output
Speed	Float	Sets the speed at which the time of day changes. Default value: 1 Valid values: 0 – 100

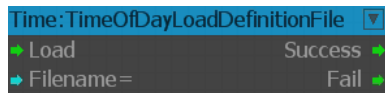
Port	Type	Description
SetSpeed	Any	Trigger to change the time of day speed to the value specified for the Speed parameter
GetSpeed	Any	Retrigger the CurTime output without updating the value of the output

Outputs

Port	Type	Description
CurTime	float	Current time of day based on when the Set input was last triggered. Use the Get input to retrigger this output and keep the current value for the output
CurSpeed	float	Speed for the current time of day based on when the SetSpeed input was last triggered. Use the GetSpeed input to retrigger this output and keep the current value for the output

TimeOfDayLoadDefinitionFile

Loads a Time of Day (TOD) definition file.



Inputs

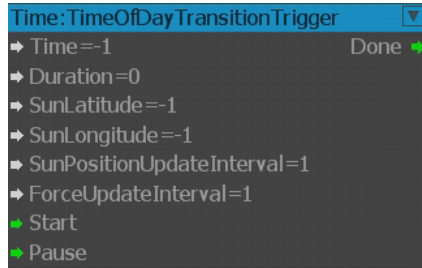
Port	Type	Description
Load	Any	Trigger to load and read the Time of Day definition file
Filename	String	Name of the XML file to load and read. The file must be in the level directory

Outputs

Port	Type	Description
Success	Any	Triggered when the Time of Day definition file has successfully loaded
Fail	Any	Triggered if the Time of Day definition file was not successfully loaded

TimeOfDayTransitionTrigger

Triggers sun position transitions when a specific time of day is reached.



Inputs

Port	Type	Description
Time	Float	Total length of time to blend the level's current time to the specified time. Set this value to -1 to disable time of day blending Default value: 1 Valid values: 0 – 24
Duration	Float	Blend duration in seconds. Default value: 0 Valid values: 0 – 100
SunLatitude	Float	Blends the level's current sun latitude value to the specified latitude in degrees. Set this value to -1 to disable latitude blending. Default value: -1 Valid values: 0 – 100
SunLongitude	Float	Blends the level's current sun latitude value to the specified latitude in degrees. Set this value to -1 to disable latitude blending Default value: -1 Valid values: 0 – 100
SunPositionUpdateInterval	Float	Amount of time in seconds between updates to reposition the sun. Set this value to 0 seconds to constantly update the sun position during the transition Default value: 1 Valid values: 0 – 100
ForceUpdateInterval	Float	Amount of time in seconds between updates to the time of day. Set this value to 0 seconds to constantly update the time of day during the transition Default value: 1 Valid values: 0 – 100

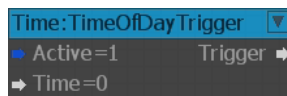
Port	Type	Description
Start	Any	Starts the transition.
Pause	Any	Pauses or resumes the transition

Outputs

Port	Type	Description
Done	Any	Triggered when the transition is finished

TimeOfDayTrigger

Triggers an action when a specific time of day is reached.



Inputs

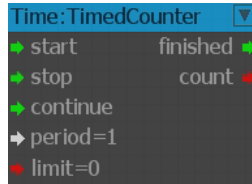
Port	Type	Description
Active	Boolean	Set this value to <code>true</code> to enable the trigger Default value: 1 Valid values: 0=false 1=true
Time	Float	Triggers the action at the specified time of day Default value: 0 Valid values: 0 – 100

Outputs

Port	Type	Description
Trigger	Float	Displays the current value for TimeOfDay. Triggered when the specified time of day has been reached

TimedCounter

Counts the number of ticks. Starting from 0, the counter increments by 1 every time the amount of time specified for the Period input has passed. When the counter reaches the value specified for the Limit input, the Finished output is triggered.



Inputs

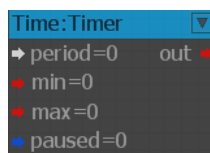
Port	Type	Description
Start	Any	Trigger to start the counter. If it is already running, this resets the counter.
Stop	Any	Stops the counter
Continue	Any	Resumes the counter
Period	Float	Tick period in seconds Default value: 1 Valid values: 0 – 100
Limit	Integer	Default value: 0 Valid values: 0 – 100

Outputs

Port	Type	Description
Finished	Any	Trigger indicating the counter is finished. The value that was provided as the Start input is the same as the Finished value.
Count	Integer	Value for the tick counter

Timer

Outputs the count from minimum to maximum, ticking for the specified period.



Inputs

Port	Type	Description
period	Float	Tick period in seconds Default value: 0 Valid values: 0 – 100

Port	Type	Description
min	Integer	Minimum value for the timer Default value: 0 Valid values: 0 – 100
max	Integer	Maximum value for the timer Default value: 0 Valid values: 0 – 100
paused	Boolean	Pauses the timer when set to <code>true</code> . Default value: 0 Valid values: 0=false 1=true

Outputs

Port	Type	Description
Out	Integer	Total count for the specified period

Twitch Nodes

You can use these flow graph nodes to configure Twitch-related settings.

The **Choose Entity** input port that is included for a number of flow graph nodes is used to change the attached entity dynamically.

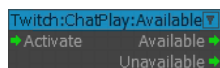
Topics

- [Twitch ChatPlay General Nodes \(p. 730\)](#)
- [Twitch ChatPlay Voting Nodes \(p. 734\)](#)
- [Twitch JoinIn Nodes \(p. 737\)](#)
- [TwitchAPI Nodes \(p. 737\)](#)

Twitch ChatPlay General Nodes

The following flow graph nodes are used to configure general Twitch ChatPlay-related settings.

Twitch:Chatplay:Available node



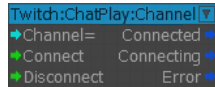
Inputs

Port	Type	Description
Activate	Void	Used to check the availability of Twitch ChatPlay.

Outputs

Port	Type	Description
Available	Void	Used to indicate that Twitch ChatPlay is available
Unavailable	Void	Used to indicate that Twitch ChatPlay is not available

Twitch:Chatplay:Channel node



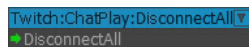
Inputs

Port	Type	Description
Channel	String	Twitch channel name
Connect	Void	Initiates connection. Idempotent if called while already connected or connecting. Resets the <code>Error</code> output state.
Disconnect	Void	Initiates disconnection. Idempotent if called while already disconnected or disconnecting.

Outputs

Port	Type	Description
Connected	Boolean	Current state of the connection to the channel
Connecting	Boolean	Indicates whether the node is currently attempting to connect
Error	Boolean	Indicates an error has occurred

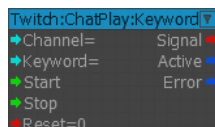
Twitch:Chatplay:DisconnectAll node



Inputs

Port	Type	Description
DisconnectAll	Void	Disconnects all Twitch ChatPlay channels

Twitch:Chatplay:Keyword node



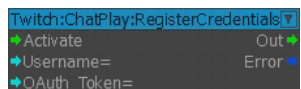
Inputs

Port	Type	Description
Channel	String	Twitch channel name
Keyword	String	Keyword to match
Start	Void	Starts scanning for keywords. Idempotent if called while already started.
Stop	Void	Stops scanning for a keywords. Idempotent if called while already stopped.
Reset	Integer	Controls the initial signal count. Changes to <code>Reset</code> are applied immediately to the current signal count.

Outputs

Port	Type	Description
Signal	Integer	Event that fires when the keyword is received on the specified channel. The value is incremented by +1 each time a keyword is received.
Active	Boolean	Indicates whether the node is currently active. True if signals can occur (set as soon as <code>Start</code> is triggered); otherwise, false (set as soon as <code>Stop</code> is triggered).
Error	Boolean	Used to indicate that an error has occurred

Twitch:Chatplay:RegisterCredentials node



Inputs

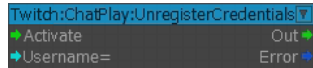
Port	Type	Description
Activate	Void	Registers the username and OAuth token credential pair
Username	String	Twitch username
OAuth_Token	String	OAuth tokens are generated with the Twitch Chat OAuth Password Generator .

Outputs

Port	Type	Description
Out	Void	Signalled when done registering credentials

Port	Type	Description
Error	Boolean	Used to indicate that an error has occurred

Twitch:Chatplay:UnregisterCredentials node



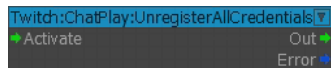
Inputs

Port	Type	Description
Activate	Void	Unregisters the username and associated OAuth token
Username	String	Twitch username

Outputs

Port	Type	Description
Out	Void	Used to indicate when the unregistering of the credential has completed
Error	Boolean	Used to indicate that an error has occurred

Twitch:Chatplay:UnregisterAllCredentials node



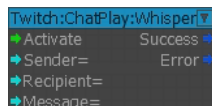
Inputs

Port	Type	Description
Activate	Void	Used to unregister all credentials at once

Outputs

Port	Type	Description
Out	Void	Used to indicate when the unregistering of all credential has completed
Error	Boolean	Used to indicate when an error occurs

Twitch:Chatplay:Whisper node



Inputs

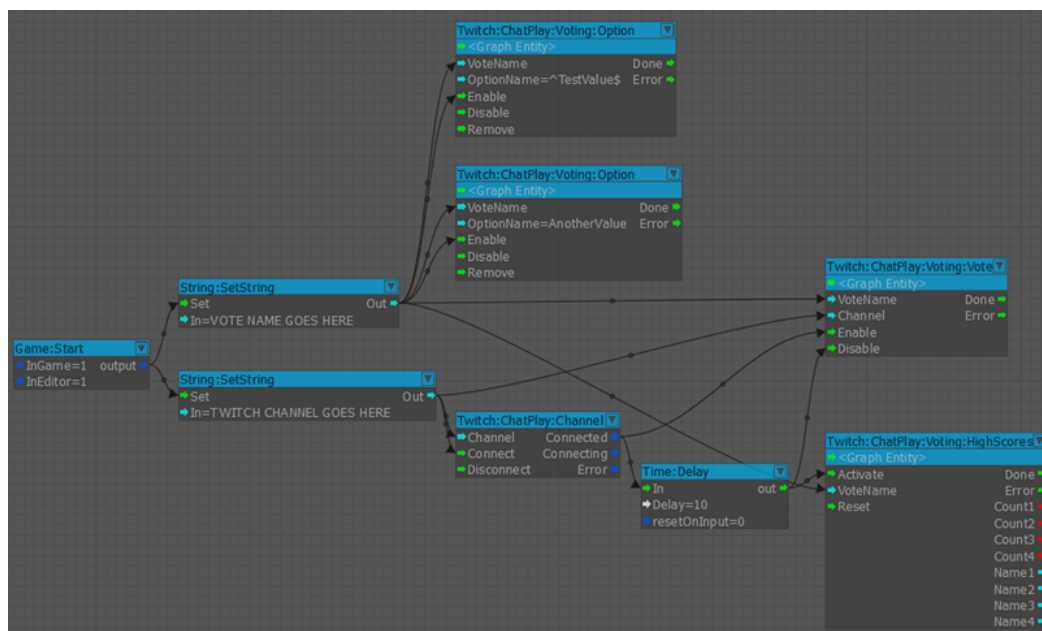
Port	Type	Description
Activate	Void	Sends the message as a whisper on behalf of the sender to the recipient.
Sender	String	Twitch username of sender. Must have credentials registered to successfully send a whisper (see Twitch:ChatPlay:RegisterCredentials node).
Recipient	String	Twitch username of recipient.
Message	String	Message to whisper to recipient.

Outputs

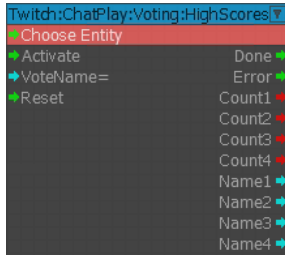
Port	Type	Description
Success	Boolean	True when message is sent successfully; otherwise, false.
Error	Boolean	Signalled as true when an error occurred, false otherwise.

Twitch ChatPlay Voting Nodes

Twitch ChatPlay voting functionality make it easier to set up polls, surveys, and votes. The following figure shows an example of how various Flow Graph voting nodes work together.



Twitch:Chatplay:Voting:HighScores node



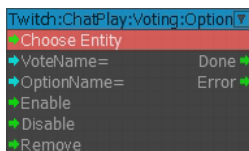
Inputs

Port	Type	Description
Activate	Void	Used to query the high scores.
VoteName	String	The name of the vote.
Reset	Void	Used to reset all counts to zero.

Outputs

Port	Type	Description
Done	Void	Used to indicate when the operation is complete
Error	Void	Used to indicate that an error occurred.
Count1 - Count4	Integer	Used to indicate the vote count for option 1, 2, 3, and 4.
Name1 - Name4	String	The names for options 1, 2, 3, and 4.

Twitch:Chatplay:Voting:Option node



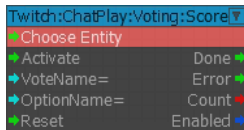
Inputs

Port	Type	Description
VoteName	String	The name of the vote.
OptionName	String	The name of the voting option.
Enable	Void	Used to enable the option and that it can be voted on.
Disable	Void	Used to disable the ability to vote on the option.
Remove	Void	Used to delete the option.

Outputs

Port	Type	Description
Done	Void	Used to indicate when the operation is complete.
Error	Void	Used to indicate that an error occurred.

Twitch:Chatplay:Voting:Score node



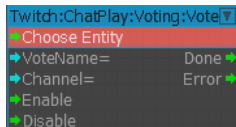
Inputs

Port	Type	Description
Activate	Void	Used to query the score for an option.
VoteName	String	The name of the vote.
OptionName	String	The name of the voting option.
Reset	Void	Used to reset the count to zero.

Outputs

Port	Type	Description
Done	Void	Used to indicate when the operation is complete.
Error	Void	Used to indicate that an error occurred.
Count	Integer	Used to indicate the current vote count
Enabled	Boolean	Used to indicate the current option state.

Twitch:Chatplay:Voting:Vote node



Inputs

Port	Type	Description
VoteName	String	The name of the vote.
Channel	String	The Twitch ChatPlay channel used to connect the vote to.
Enable	Void	Used to enable the vote and that it can be voted on.

Port	Type	Description
Disable	Void	Used to disable the ability to vote on the vote.

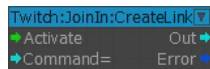
Outputs

Port	Type	Description
Done	Void	Used to indicate when the operation is complete.
Error	Void	Used to indicate that an error occurred.

Twitch JoinIn Nodes

Twitch JoinIn nodes are used to create a link that includes all the multiplayer session information necessary for other players to connect to the same session using the generated link.

Twitch:Joinin:CreateLink node



Inputs

Port	Type	Description
Activate	Void	Generates a <code>game :</code> protocol link that allows players to join the current game.
Command	String	The commands to pass when a game launches.

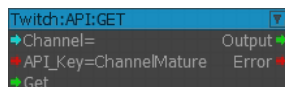
Outputs

Port	Type	Description
Out	String	Signalled with the generated link.
Error	Boolean	Used to indicate that an error occurred.

TwitchAPI Nodes

TwitchAPI nodes are used to make calls to Twitch's REST API from within Lumberyard.

Twitch:API:GET node



Inputs

Port	Type	Description
Channel	String	Twitch channel name

Port	Type	Description
API_Key	String enum	API call type and key. Call types based on channel ID: channel, chat, follows, streams, subscriptions, and user.
Get	Any	Caching has not been implemented, triggering the Get port will always start a new API call.

Outputs

Port	Type	Description
Output	Any	Returned value for the given API call type and key. Triggered whenever an API call completes.
Error	Integer	Indicates whether an error has occurred. It may be triggered with one of the following values: 1: the value for the requested API key was null 2: the value for the requested API key was of an unexpected type 3: the HTTP request failed

Vec3 Nodes

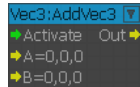
You can use the following flow graph nodes to define vector (Vec3) operations.

Topics

- [AddVec3 node \(p. 738\)](#)
- [Calculate node \(p. 739\)](#)
- [ClampVec3 node \(p. 739\)](#)
- [CrossVec3 node \(p. 740\)](#)
- [DotVec3 node \(p. 740\)](#)
- [EqualVec3 node \(p. 741\)](#)
- [FromVec3 node \(p. 741\)](#)
- [MagnitudeVec3 node \(p. 742\)](#)
- [MulVec3 node \(p. 742\)](#)
- [NormalizeVec3 node \(p. 742\)](#)
- [ReciprocalVec3 node \(p. 743\)](#)
- [RotateVec3onAxis node \(p. 743\)](#)
- [ScaleVec3 node \(p. 744\)](#)
- [SetVec3 node \(p. 744\)](#)
- [SubVec3 node \(p. 744\)](#)
- [ToVec3 node \(p. 745\)](#)

AddVec3 node

Used to output the sum of two vectors.



Inputs

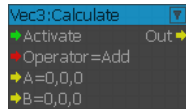
Port	Type	Description
Activate	Any	Triggers the node
A	Vec3	First operand
B	Vec3	Second operand

Outputs

Port	Type	Description
Out	Vec3	Addition of A and B

Calculate node

Used to output the specified calculation between two vectors.



Inputs

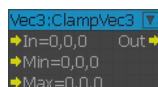
Port	Type	Description
Activate	Any	Triggers the node
Operator	Integer	Math operation to perform
A	Vec3	First operand
B	Vec3	Second operand

Outputs

Port	Type	Description
Out	Vec3	Calculated operation of A and B

ClampVec3 node

Used to clamp the output range of a vector between a minimum and a maximum.



Inputs

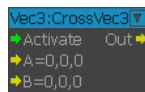
Port	Type	Description
In	Vec3	Input value
Min	Vec3	Minimum clamping value
Max	Vec3	Maximum clamping value

Outputs

Port	Type	Description
Out	Vec3	Triggers when the input value is between the minimum and maximum values

CrossVec3 node

Used to output the cross product of two vectors.



Inputs

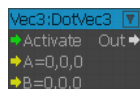
Port	Type	Description
Activate	Any	Triggers the node
A	Vec3	First operand
B	Vec3	Second operand

Outputs

Port	Type	Description
Out	Vec3	Outputs the cross product of the inputs

DotVec3 node

Used to output the dot product of the inputs.



Inputs

Port	Type	Description
Activate	Any	Triggers the node

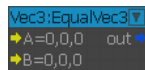
Port	Type	Description
A	Vec3	First operand
B	Vec3	Second operand

Outputs

Port	Type	Description
Out	Float	Outputs the dot product of the inputs

EqualVec3 node

Used to trigger an output when both vectors are equal in value.



Inputs

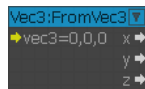
Port	Type	Description
A	Vec3	First operand
B	Vec3	Second operand

Outputs

Port	Type	Description
Out	Boolean	Triggers when A and B are equal in value

FromVec3 node

Used to output the x, y, and z values of the vector.



Inputs

Port	Type	Description
Vec3	Vec3	Input vector

Outputs

Port	Type	Description
X	Float	X-axis value of vector

Port	Type	Description
Y	Float	Y-axis value of vector
Z	Float	Z-axis value of vector

MagnitudeVec3 node

Used to output the magnitude (length) of the vector.



Inputs

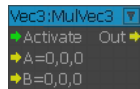
Port	Type	Description
Vector	Vec3	Input vector

Outputs

Port	Type	Description
Length	Any	Magnitude (length) of the input vector

MulVec3 node

Used to output the multiplication of two vectors.



Inputs

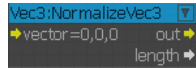
Port	Type	Description
Activate	Any	Triggers the node
A	Vec3	First operand
B	Vec3	Second operand

Outputs

Port	Type	Description
Out	Vec3	Multiplication of A and B

NormalizeVec3 node

Used to output the normalized value of the vector.



Inputs

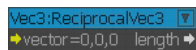
Port	Type	Description
Vector	Vec3	Vector input

Outputs

Port	Type	Description
Out	Vec3	Normalized vector input
Length	Float	Magnitude

ReciprocalVec3 node

Used to output the reciprocal of the vector.



Inputs

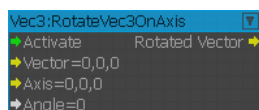
Port	Type	Description
Vector	Vec3	Input vector

Outputs

Port	Type	Description
Length	Float	Reciprocal value of input

RotateVec3OnAxis node

Used to output an axis-rotated value of the vector.



Inputs

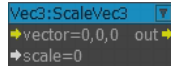
Port	Type	Description
Activate	Any	Triggers the node
Vector	Vec3	Input vector to rotate
Axis	Vec3	Axis to rotate input around
Angle	Float	Angle in degrees to rotate

Outputs

Port	Type	Description
Rotated Vector	Vec3	Result of the rotation

ScaleVec3 node

Used to output a scaled value of the vector.



Inputs

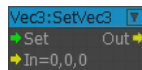
Port	Type	Description
Vector	Vec3	Input vector
Scale	Float	Scale factor to apply to the input

Outputs

Port	Type	Description
Out	Vec3	Result of the scaling

SetVec3 node

Used to output the input value when the Set input is activated.



Inputs

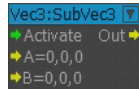
Port	Type	Description
Set	Any	Triggers the vector to the output
In	Vec3	Input value

Outputs

Port	Type	Description
Out	Vec3	Outputs the input value

SubVec3 node

Used to output the subtracted value of two vectors.



Inputs

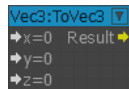
Port	Type	Description
Activate	Any	Triggers the node
A	Vec3	First operand
B	Vec3	Second operand

Outputs

Port	Type	Description
Out	Vec3	Subtraction of B from A

ToVec3 node

Used to output three floating point values to a vector.



Inputs

Port	Type	Description
X	Float	X-axis value
Y	Float	Y-axis value
Z	Float	Z-axis value

Outputs

Port	Type	Description
Result	Vec3	Vector output

Vehicle Nodes

You can use the following flow graph nodes to configure vehicle behavior and related settings.

Note

These nodes will only work with the Legacy Game Sample (CryEngine GameSDK), which is available at [Lumberyard Downloads](#).

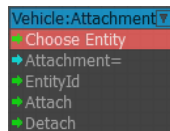
Topics

- [Attachment node \(p. 746\)](#)
- [ChangeSeat node \(p. 746\)](#)

- [ChaseTarget node \(p. 747\)](#)
- [Damage node \(p. 747\)](#)
- [Destroy node \(p. 748\)](#)
- [Enter node \(p. 748\)](#)
- [FollowPath node \(p. 749\)](#)
- [GetSeatHelper node \(p. 750\)](#)
- [Handbrake node \(p. 750\)](#)
- [Honk node \(p. 751\)](#)
- [Lights node \(p. 751\)](#)
- [Lock node \(p. 751\)](#)
- [MoveActionMult node \(p. 752\)](#)
- [Movement node \(p. 752\)](#)
- [MovementParams node \(p. 753\)](#)
- [Passenger node \(p. 753\)](#)
- [Seat node \(p. 753\)](#)
- [StickPath node \(p. 754\)](#)
- [Turret node \(p. 755\)](#)
- [Unload node \(p. 755\)](#)

Attachment node

Used to control vehicle entity attachments.

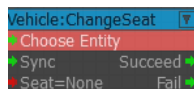


Inputs

Port	Type	Description
Attachment	String	Attachment to add
EntityId	Any	ID of the entity to use
Attach	Any	Attaches the item
Detach	Any	Detaches the item

ChangeSeat node

Used to move a character from one seat to another one. Only works if the character is already inside a vehicle.



Inputs

Port	Type	Description
Sync	Any	Triggers the seat change

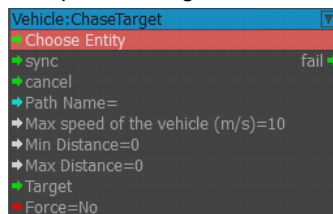
Port	Type	Description
Seat	Integer	Seat to change to

Outputs

Port	Type	Description
Succeed	Any	Seat change succeeded
Fail	Any	Seat change failed

ChaseTarget node

Used to follow or navigate along the specified path while attempting to establish line of sight or fire with the specified target.



Inputs

Port	Type	Description
Sync	Any	Triggers the chase
Cancel	Any	Cancels the chase
Path Name	String	Name of the path to follow
Max speed of the vehicle	Float	Maximum speed of the vehicle
Min Distance	Float	Minimum chase distance to the target
Max Distance	Float	Minimum chase distance to the target
Target	Any	ID of the target to chase
Force	Integer	Force execution method

Outputs

Port	Type	Description
Fail	Any	Chase failed

Damage node

Used to handle vehicle damage.



Inputs

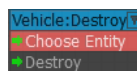
Port	Type	Description
HitTrigger	Any	Triggers that causes the vehicle to sustain damage
HitValue	Float	Amount of damage the vehicle will sustain
HitPosition	Vec3	Position at which the vehicle will sustain the hit
HitRadius	Float	Radius of the hit
Indestructible	Boolean	Value of true sets the vehicle to be indestructible
HitType	String	Type of damage
HitComponent	String	Vehicle component that will receive the hit

Outputs

Port	Type	Description
Damaged	Float	Amount of damage sustained by the vehicle
Destroyed	Boolean	True if vehicle was destroyed
Hit	Float	Hit value sustained by the vehicle

Destroy node

Used to destroy the vehicle.

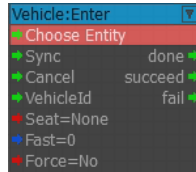


Inputs

Port	Type	Description
Destroy	Any	Trigger to destroy the vehicle

Enter node

Used to make an AI agent sit in a specified seat of a specified vehicle.



Inputs

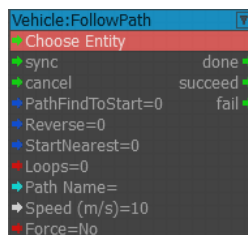
Port	Type	Description
Sync	Any	Activates the node
Cancel	Any	Cancels the operation
VehicleId	Any	ID of the vehicle
Seat	Integer	Seat to sit on
Fast	Boolean	Skip approach and enter vehicle
Force	Integer	Force execution method

Outputs

Port	Type	Description
Done	Any	Action completed
Succeed	Any	Action was successful
Fail	Any	Action failed

FollowPath node

Used to follow the path speed stance action.



Inputs

Port	Type	Description
Sync	Any	Activates the node
Cancel	Any	Cancels execution
PathFindToStart	Boolean	Whether to find the start of the path
Reverse	Boolean	Reverses the path direction
StartNearest	Boolean	Starts the path at the nearest point on path

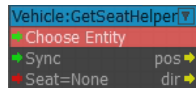
Port	Type	Description
Loops	Integer	Number of times to loop around the path
Path Name	String	Name of the path
Speed (m/s)	Float	Speed in meters/second
Force	Integer	Force execution method

Outputs

Port	Type	Description
Done	Any	Action completed
Succeed	Any	Action was successful
Fail	Any	Action failed

GetSeatHelper node

Used to gets the helper position of a seat for entering the vehicle.



Inputs

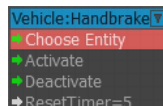
Port	Type	Description
Sync	Any	Get helper position
Seat	Integer	Seat to be entered

Outputs

Port	Type	Description
Pos	Vec3	Position of seat helper
Dir	Vec3	Direction of seat helper

Handbrake node

Used to toggle the vehicle handbrake. Currently only supported for the ArcadeWheeled movement type.

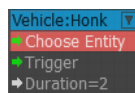


Inputs

Port	Type	Description
Activate	Any	Activates the vehicle handbrake
Deactivate	Any	Deactivates the vehicle handbrake
ResetTimer	Float	Resets the timer

Honk node

Use to control a vehicle's horn.

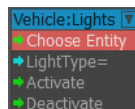


Inputs

Port	Type	Description
Trigger	Any	Activates the vehicle horn
Duration	Float	Duration in seconds of the horn

Lights node

Used to control a vehicle's lights.

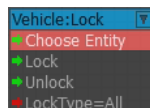


Inputs

Port	Type	Description
LightType	String	Type of vehicle light
Activate	Any	Activates vehicle lights
Deactivate	Any	Deactivates vehicle lights

Lock node

Used to lock or unlock all seats of a vehicle.

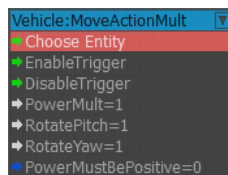


Inputs

Port	Type	Description
Lock	Any	Locks the vehicle
Unlock	Any	Unlocks the vehicle
LockType	Integer	Type of vehicle lock

MoveActionMult node

Used to add multipliers to a vehicle's movement actions.

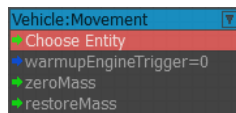


Inputs

Port	Type	Description
EnableTrigger	Any	Activates the node
DisableTrigger	Any	Deactivates the node
PowerMult	Float	Vehicle engine power multiplier
RotatePitch	Float	Vehicle pitch rotation multiplier
RotateYaw	Float	Vehicle yaw rotation multiplier
PowerMustBePositive	Boolean	True if power multiplication is positive (increase)

Movement node

Used to control vehicle movement.

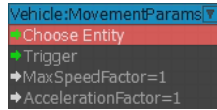


Inputs

Port	Type	Description
WarmUpEngineTrigger	Boolean	Warms up vehicle engine
ZeroMass	Any	Vehicle has zero mass
RestoreMass	Any	Restores vehicle mass

MovementParams node

Used to modify vehicle movement parameters.

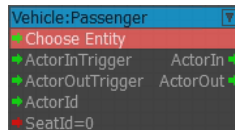


Inputs

Port	Type	Description
Trigger	Any	Activates the node
MaxSpeedFactor	Float	Maximum vehicle speed factor
AccelerationFactor	Float	Maximum vehicle acceleration factor

Passenger node

Used to manage vehicle passengers.



Inputs

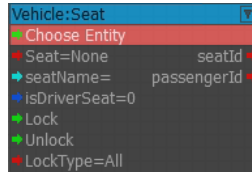
Port	Type	Description
ActorInTrigger	Any	Forces actor to get into vehicle if a seat is available
ActorOutTrigger	Any	Forces actor to get out of the vehicle
ActorId	Any	ID of the action
SeatId	Integer	ID of the seat

Outputs

Port	Type	Description
ActorIn	Any	Triggered if any actor got into vehicle
ActorOut	Any	Triggered if any actor got out of vehicle

Seat node

Used to manage vehicle seats.



Inputs

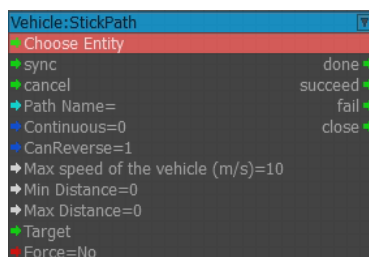
Port	Type	Description
Seat	Integer	ID of seat
SeatName	String	Name of seat
IsDriverSeat	Boolean	True is driver seat
Lock	Any	Locks the vehicle
Unlock	Any	Unlocks the vehicle
LockType	Integer	Type of vehicle lock

Outputs

Port	Type	Description
SeatId	Integer	ID of seat
PassengerId	Integer	ID of passenger

StickPath node

Used to follow the specified path to the end and sticking to the optional target, either continuously or as a one-off event.



Inputs

Port	Type	Description
Sync	Any	Activates the node
Cancel	Any	Cancel execution
Path Name	String	Name of path
Continuous	Boolean	Whether vehicle can continue to follow the path or stops once it reaches the target

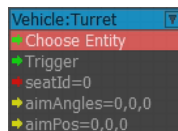
Port	Type	Description
CanReverse	Boolean	Whether vehicle is allowed to drive in reverse to follow target or path
Max speed of the vehicle	Float	Maximum speed of the vehicle
Min Distance	Float	Minimum stick distance to the target
Max Distance	Float	Maximum stick distance to the target
Target	Any	ID of target to stick to when following the path
Force	Integer	Force execution method

Outputs

Port	Type	Description
Done	Any	Action completed
Succeed	Any	Action was successful
Fail	Any	Action failed
Close	Any	Close to destination

Turret node

Use to control the vehicle turret.

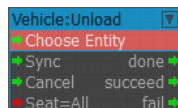


Inputs

Port	Type	Description
Trigger	Any	Activates vehicle turret
SeatId	Integer	ID of seat
AimAngles	Vec3	Turret aiming angle
AimPos	Vec3	Turret target location

Unload node

Use to unloads vehicle, ejecting specified passengers.



Inputs

Port	Type	Description
Sync	Any	Triggers the action
Cancel	Any	Cancels execution
Seat	Integer	Seat to eject passenger from

Outputs

Port	Type	Description
Done	Any	Action completed
Succeed	Any	Action succeeded
Fail	Any	Action failed

Video Nodes

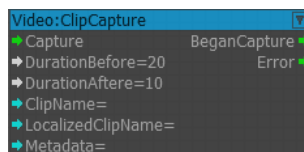
You can use the following flow graph nodes to configure video settings.

Topics

- [ClipCapture node \(p. 756\)](#)

ClipCapture node

Used to capture video clips while a game is running and (for Xbox One) save them locally or to the cloud.



Inputs

Port	Type	Description
Capture	Any	Begin capturing a video clip
DurationBefore	Float	Record the specified number seconds before the Capture input is triggered
DurationAfter	Float	Record the specified number seconds after the Capture input is triggered
ClipName	String	For Xbox One, the MagicMoment ID used to look up the description string entered through the Xbox Developer Portal.
LocalizedClipName	String	For Xbox One, the ClipName shown during the Toast popup

Port	Type	Description
Metadata	String	(Optional). Used to tag video clips

Outputs

Port	Type	Description
BeganCapture	Any	Triggered when video clip capture has begun
Error	Any	Triggered when a clip capture error has occurred

Weapon Nodes

You can use the following flow graph nodes to configure weapon settings.

Note

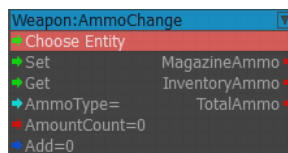
These nodes will only work with the Legacy Game Sample (CryEngine GameSDK), which is available at [Lumberyard Downloads](#).

Topics

- [AmmoChange node \(p. 757\)](#)
- [AutoSightWeapon node \(p. 758\)](#)
- [ChangeFireMode node \(p. 758\)](#)
- [FireWeapon node \(p. 758\)](#)
- [Listener node \(p. 759\)](#)

AmmoChange node

Used to give or take ammunition to for from the player. Weapon and ammo type must match.



Inputs

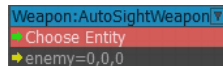
Port	Type	Description
Set	Any	Activates the node
Get	Any	Retrieves the amount of ammunition left
AmmoType	String	Type of ammunition to add
AmountCount	Integer	Gets the amount of ammunition left
Add	Boolean	Adds the specified amount of ammunition

Outputs

Port	Type	Description
MagazineAmmo	Integer	Ammunition left in the weapon magazine
InventoryAmmo	Integer	Ammunition left in inventory
TotalAmmo	Integer	Total ammunition available

AutoSightWeapon node

This node

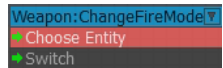


Inputs

Port	Type	Description
Enemy	Vec3	Aims the weapon at the enemy's position

ChangeFireMode node

Used to change the weapon fire mode.

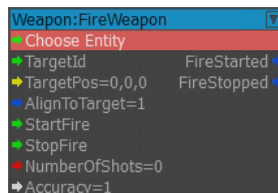


Inputs

Port	Type	Description
Switch	Any	Switches the weapon fire mode

FireWeapon node

Use to fire a weapon and set a target entity or a target position.



Inputs

Port	Type	Description
TargetId	Any	Target ID
TargetPos	Vec3	Target position

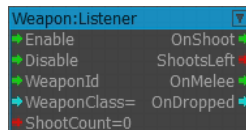
Port	Type	Description
AlignToTarget	Boolean	Aims the weapon at the target
StartFire	Any	Starts firing weapon
StopFire	Any	Stops firing weapon
NumberOfShots	Integer	Fires the specified number of shots
Accuracy	Float	Specifies firing accuracy from 0% to 100%

Outputs

Port	Type	Description
FireStarted	Boolean	Triggers when firing starts
FireStopped	Boolean	Triggers when firing stops

Listener node

Use to listen on WeaponId or player's WeaponClass, or as a fallback on the current player's weapon and to trigger OnShoot when shot.



Inputs

Port	Type	Description
Enable	Any	Enable listener
Disable	Any	Disables listener
WeaponId	Any	Weapon ID
WeaponClass	String	Weapon name
ShootCount	Integer	Number of times the listener can be triggered. 0 = infinite

Outputs

Port	Type	Description
OnShoot	Any	Triggered when shooting
ShootsLeft	Integer	Triggered when shooting left
OnMelee	Any	Triggered on melee attack
OnDropped	String	Triggered when weapon is dropped

XML Nodes

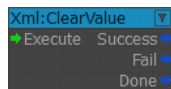
You can use the following flow graph nodes to specify XML elements.

Topics

- [ClearValue node \(p. 760\)](#)
- [DeleteAllAttributes node \(p. 761\)](#)
- [DeleteAllChildren node \(p. 761\)](#)
- [DeleteAttribute node \(p. 762\)](#)
- [DeleteChild node \(p. 762\)](#)
- [DeleteChildAt node \(p. 762\)](#)
- [GetAttribute node \(p. 763\)](#)
- [GetAttributeCount node \(p. 763\)](#)
- [GetChild node \(p. 764\)](#)
- [GetChildAt node \(p. 764\)](#)
- [GetChildCount node \(p. 765\)](#)
- [GetParent node \(p. 765\)](#)
- [GetRoot node \(p. 766\)](#)
- [GetValue node \(p. 766\)](#)
- [HasAttribute node \(p. 767\)](#)
- [IncAttribute node \(p. 767\)](#)
- [IncValue node \(p. 768\)](#)
- [NewChild node \(p. 768\)](#)
- [NewDocument node \(p. 769\)](#)
- [OpenDocument node \(p. 769\)](#)
- [SaveDocument node \(p. 770\)](#)
- [SetAttribute node \(p. 771\)](#)
- [SetValue node \(p. 771\)](#)

ClearValue node

Used to clear the value of the active element.



Inputs

Port	Type	Description
Execute	Any	Executes the instruction

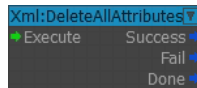
Outputs

Port	Type	Description
Success	Boolean	Called if the instruction executed successfully
Fail	Boolean	Called if the instruction failed

Port	Type	Description
Done	Boolean	Called when the instruction has completed carrying out

DeleteAllAttributes node

Used to delete all attributes from the active element.



Inputs

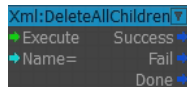
Port	Type	Description
Execute	Any	Executes the instruction

Outputs

Port	Type	Description
Success	Boolean	Called if the instruction executed successfully
Fail	Boolean	Called if the instruction failed
Done	Boolean	Called when the instruction has completed carrying out

DeleteAllChildren node

Used to delete all children of the active element.



Inputs

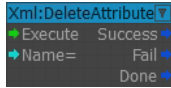
Port	Type	Description
Execute	Any	Executes the instruction
Name	String	Optional child name

Outputs

Port	Type	Description
Success	Boolean	Called if the instruction executed successfully
Fail	Boolean	Called if the instruction failed
Done	Boolean	Called when the instruction has completed carrying out

DeleteAttribute node

Used to delete an attribute from the active element.



Inputs

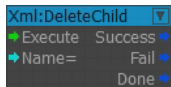
Port	Type	Description
Execute	Any	Executes the instruction
Name	String	Optional child name

Outputs

Port	Type	Description
Success	Boolean	Called if the instruction executed successfully
Fail	Boolean	Called if the instruction failed
Done	Boolean	Called when the instruction has completed carrying out

DeleteChild node

Used to delete the first child node with the given name.



Inputs

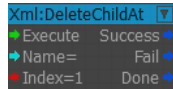
Port	Type	Description
Execute	Any	Executes the instruction
Name	String	Optional child name

Outputs

Port	Type	Description
Success	Boolean	Called if the instruction executed successfully
Fail	Boolean	Called if the instruction failed
Done	Boolean	Called when the instruction has completed carrying out

DeleteChildAt node

Used to delete the nth child node with the given name.



Inputs

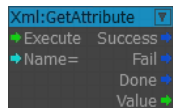
Port	Type	Description
Execute	Any	Executes the instruction
Name	String	Name of the attribute
Index	Integer	Location of the child node in the list

Outputs

Port	Type	Description
Success	Boolean	Called if the instruction executed successfully
Fail	Boolean	Called if the instruction failed
Done	Boolean	Called when the instruction has completed carrying out

GetAttribute node

Used to get the value of an attribute for the active element.



Inputs

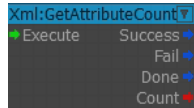
Port	Type	Description
Execute	Any	Executes the instruction
Name	String	Name of the attribute

Outputs

Port	Type	Description
Success	Boolean	Called if the instruction executed successfully
Fail	Boolean	Called if the instruction failed
Done	Boolean	Called when the instruction has completed carrying out

GetAttributeCount node

Used to get the number of attributes for the active element.



Inputs

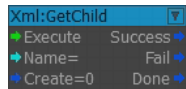
Port	Type	Description
Execute	Any	Executes the instruction

Outputs

Port	Type	Description
Success	Boolean	Called if the instruction executed successfully
Fail	Boolean	Called if the instruction failed
Done	Boolean	Called when the instruction has completed carrying out
Count	Integer	Outputs the count

GetChild node

Used to navigate to the first child node with the given name.



Inputs

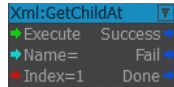
Port	Type	Description
Execute	Any	Executes the instruction
Name	String	Name of the attribute
Create	Boolean	Creates a child node if one does not exist

Outputs

Port	Type	Description
Success	Boolean	Called if the instruction executed successfully
Fail	Boolean	Called if the instruction failed
Done	Boolean	Called when the instruction has completed carrying out

GetChildAt node

Used to navigate to the nth child node with the given name.



Inputs

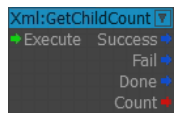
Port	Type	Description
Execute	Any	Executes the instruction
Name	String	Name of the child node
Index	Integer	Location of the child node in the list

Outputs

Port	Type	Description
Success	Boolean	Called if the instruction executed successfully
Fail	Boolean	Called if the instruction failed
Done	Boolean	Called when the instruction has completed carrying out

GetChildCount node

Used to return the number of children of the active element.



Inputs

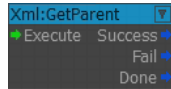
Port	Type	Description
Execute	Any	Executes the instruction

Inputs

Port	Type	Description
Success	Boolean	Called if the instruction executed successfully
Fail	Boolean	Called if the instruction failed
Done	Boolean	Called when the instruction has completed carrying out

GetParent node

Used to sets the active element to the current active element's parent (move one up).



Inputs

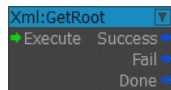
Port	Type	Description
Execute	Any	Executes the instruction

Outputs

Port	Type	Description
Success	Boolean	Called if the instruction executed successfully
Fail	Boolean	Called if the instruction failed
Done	Boolean	Called when the instruction has completed carrying out

GetRoot node

Used to set the active element to the root node (move to top).



Inputs

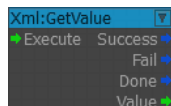
Port	Type	Description
Execute	Any	Executes the instruction

Outputs

Port	Type	Description
Success	Boolean	Called if the instruction executed successfully
Fail	Boolean	Called if the instruction failed
Done	Boolean	Called when the instruction has completed carrying out

GetValue node

Used to get the value of the active element.



Inputs

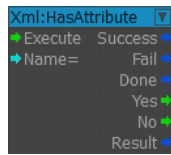
Port	Type	Description
Execute	Any	Executes the instruction

Outputs

Port	Type	Description
Success	Boolean	Called if the instruction executed successfully
Fail	Boolean	Called if the instruction failed
Done	Boolean	Called when the instruction has completed carrying out
Value	Any	Outputs the value of the element

HasAttribute node

Used to check if an attribute exists for the active element.



Inputs

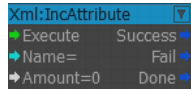
Port	Type	Description
Execute	Any	Executes the instruction
Name	String	Name of the attribute

Outputs

Port	Type	Description
Success	Boolean	Called if the instruction executed successfully
Fail	Boolean	Called if the instruction failed
Done	Boolean	Called when the instruction has completed carrying out
Yes	Any	Has the attribute
No	Any	Does not have the attribute
Result	Boolean	Boolean result

IncAttribute node

Used to increment an attribute by the given amount for the active element.



Inputs

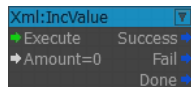
Port	Type	Description
Execute	Any	Executes the instruction
Name	String	Name of the attribute
Amount	Float	Amount to increment by

Outputs

Port	Type	Description
Success	Boolean	Called if the instruction executed successfully
Fail	Boolean	Called if the instruction failed
Done	Boolean	Called when the instruction has completed carrying out

IncValue node

Used to increment the value of the active element.



Inputs

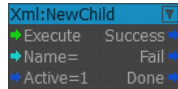
Port	Type	Description
Execute	Any	Executes the instruction
Amount	Float	Amount to increment by

Outputs

Port	Type	Description
Success	Boolean	Called if the instruction executed successfully
Fail	Boolean	Called if the instruction failed
Done	Boolean	Called when the instruction has completed carrying out

NewChild node

Used to create a new child node at end of parent's sibling list.



Inputs

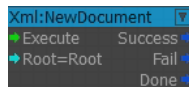
Port	Type	Description
Execute	Any	Executes the instruction
Name	String	Name of the child node
Active	Boolean	Makes the child node the active element

Outputs

Port	Type	Description
Success	Boolean	Called if the instruction executed successfully
Fail	Boolean	Called if the instruction failed
Done	Boolean	Called when the instruction has completed carrying out

NewDocument node

Used to create a blank document for writing new data into.



Inputs

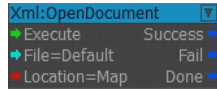
Port	Type	Description
Execute	Any	Executes the instruction
Root	String	Name of the XML root element

Outputs

Port	Type	Description
Success	Boolean	Called if the instruction executed successfully
Fail	Boolean	Called if the instruction failed
Done	Boolean	Called when the instruction has completed carrying out

OpenDocument node

Used to open an XML document from disk.



Inputs

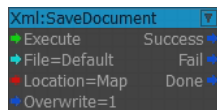
Port	Type	Description
Execute	Any	Executes the instruction
File	String	File name of the XML document
Location	Integer	File path of the XML document

Outputs

Port	Type	Description
Success	Boolean	Called if the instruction executed successfully
Fail	Boolean	Called if the instruction failed
Done	Boolean	Called when the instruction has completed carrying out

SaveDocument node

Used to save active XML data to disk.



Inputs

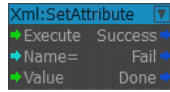
Port	Type	Description
Execute	Any	Executes the instruction
File	String	File name of the saved XML document
Location	Integer	File path of the XML document
Overwrite	Boolean	Determines where document should overwrite existing XML document

Outputs

Port	Type	Description
Success	Boolean	Called if the instruction executed successfully
Fail	Boolean	Called if the instruction failed
Done	Boolean	Called when the instruction has completed carrying out

SetAttribute node

Used to set an attribute for the active element.



Inputs

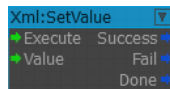
Port	Type	Description
Execute	Any	Executes the instruction
Name	String	Name of the attribute to set
Value	Any	Sets the value of the element

Outputs

Port	Type	Description
Success	Boolean	Called if the instruction executed successfully
Fail	Boolean	Called if the instruction failed
Done	Boolean	Called when the instruction has completed carrying out

SetValue node

Used to set the value of the active element.



Inputs

Port	Type	Description
Execute	Any	Executes the instruction
Value	Any	Sets the value of the attribute

Outputs

Port	Type	Description
Success	Boolean	Called if the instruction executed successfully
Fail	Boolean	Called if the instruction failed
Done	Boolean	Called when the instruction has completed carrying out

Using Flow Graph Links

Links are used to connect Flow Graph node inputs and outputs for transferring information between them. Information is transferred immediately, regardless of link length or shape. When a connected node is moved, the link automatically adjusts itself. Links are created by simply clicking and dragging your mouse from the output of one node to the input of another node.

An input port can have only one link connected to it. If you want to connect multiple links to one input port, helper nodes such as the **Logic:Any** node can be used. Output ports can have an unlimited number of links.

Node links can be deleted or disabled. If you merely want to disable a link but still have it show on the flow graph, click **Disable** instead.

To delete a node link

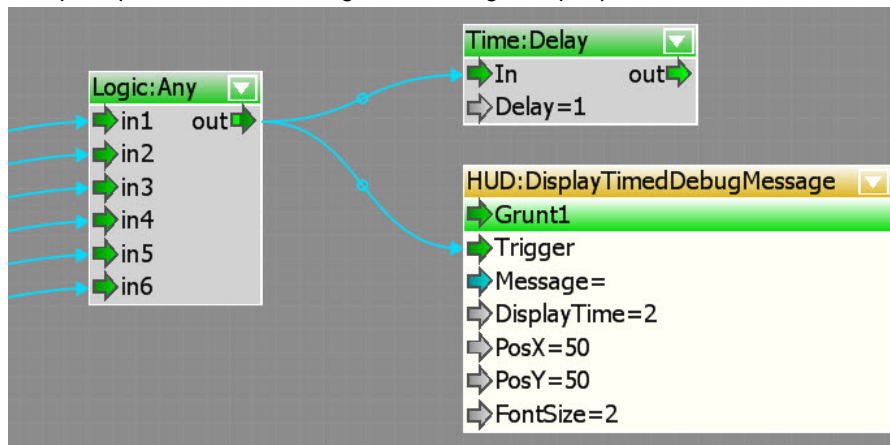
1. Click the link to select it, right-click the dot in the middle of the link, then click **Remove**.
2. Alternatively, click the input port the link is connected to and drag it away from the port. When the mouse is released, the link disappears.

By default, all information between nodes is relayed instantly. However, you can delay signal propagation between nodes.

To delay link propagation

1. Click the link to select it, right-click the dot in the middle of the link, then click **Delay**.
2. In the new **Time:Delay** node, double-click **Delay** and enter a value in seconds. The default value is 1 second if no value is entered.

Connecting multiple links to an input port is possible using the **Logic:Any** node. This node can take multiple inputs and route the signals to a single output port.



To add multiple links to an input port

1. Right-click anywhere in the graph pane and then click **Add node, Logic, Any**.
2. Drag from the various output port links to the **in1...in10** input ports of the **Logic:Any** node as needed.
3. Create links by dragging from the **out** output port to the input ports of the desired nodes.

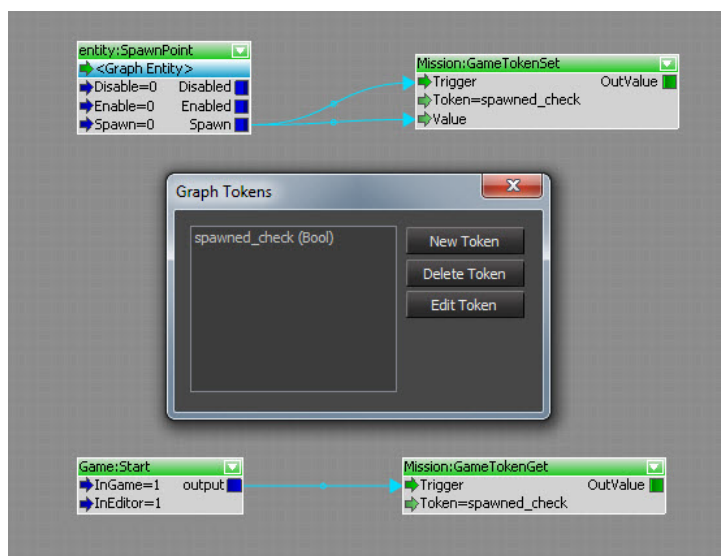
You can also highlight links to make debugging complex flow graphs easier.

- To highlight incoming links red, select an input node and press **F**.
- To highlight outgoing links blue, select an output node and press **G**.

Using Flow Graph Tokens

A flow graph token is a variable used for storing values for reuse in the same flow graph. Flow graph tokens can be used for performing simple logic and checks within a flow graph script. They are typically used to send different variables across a very large flow graph and to alleviate the need for extra node links.

Flow graph tokens share many similarities with game tokens. They can have the same types of variables set and even appear under the command `gt_show=1` along with the rest of the game tokens.



To create a Flow Graph token

1. In Flow Graph Editor, click **Tools, Edit Graph Tokens**.
2. In the **Graph Tokens** window, click **New Token**, then name the token.
3. Right-click anywhere in the flow graph, then click **Add Node, Mission, GameTokenSet**.
4. In the **Mission:GameTokenSet** node, double-click **Value** and enter a value.

Managing Flow Graph Modules

A module is simply an exported flow graph that can be loaded and called from another flow graph during gameplay.

Any flow graph can be converted to a module by first creating a new module using Flow Graph Editor and then copying the flow graph contents to the new module.

Modules used in multiple levels are called global modules, while modules used only in a specific level are called level modules.

The advantages of using modules include:

- Flow graphs can be used in multiple levels, but exist in a single location

- Modules can receive unique input values from their callers, allowing them to be robust
- Modules can return unique output values to their callers, allowing them to be used in different situations
- Modules can be instanced, so multiple copies of the same module can be active simultaneously, but running with different inputs

To create or delete a module

In Flow Graph Editor, under **Flow Graphs** do the following:

1. To create a module, right click **FG Modules**, then click **New Global FG Module** or **New Level FG Module** as applicable. The new module appears under the **Global** or **Local** folders respectively.
2. To delete a module, right-click the module and click **Delete Module**.

Module Node Ports

Flow Graph **Module** nodes have a variety of different of input and output node ports.

All inputs passed to the **Call** node activates the corresponding outputs on the **Start** node, and similarly inputs to the **End** node passes back to the **Call** node when **Success** or **Cancel** are activated.

Module Inputs

- **Call** - Call to load and start the module. If the module is already started it triggers the update port of the **Start** node with updated parameters if not instanced. It is named **Module:Call_YourModuleName**.
- **Instanced** - If set to 1 (default), creates a new independent instance of the module whenever you trigger the **Call** input port.
- **Cancel** - Cancels the module. This requires the correct InstanceID if instanced.
- **InstanceID** - Identifies a module instance. A value of -1 (default) creates a new instance; otherwise, it updates the given instance if instanced.

Module Outputs

- **OnCalled** - Called when module is started. Returns a value of -1 if the module is not instanced.
- **Done** - Called when the module returns with a success status.
- **Canceled** - Called when the module returns with a failed status.

You can also customize the inputs and outputs for each module to pass extra data back and forth.

To customize module ports

1. In Flow Graph Editor, select the module, then click **Tools, Edit Module**.
2. In the **Module Ports** dialog box, click **Edit Input** or **Edit Output** as needed, then make a **Type** selection as follows:
 - **Bool**
 - **EntityId**
 - **Int**
 - **Float**
 - **String**
 - **Vec3**

3. Click **OK** to update module nodes with the changes.

Debugging Flow Graph

Topics

- [Using Flow Graph Debugger \(p. 775\)](#)
- [Using Console Variables \(p. 775\)](#)

Using Flow Graph Debugger

Using the Flow Graph Debugger, you can add breakpoints to any input or output port of a node. Once a node port is triggered, the game is paused and the Flow Graph Editor displays the applicable node in the center of the graph pane.

To enable Flow Graph Debugger, click the bug (toggle visual flowgraph debugging) toolbar icon in Flow Graph Editor.

To resume the game once a breakpoint is triggered, click the play (Start Flowgraph Update) toolbar icon, or press **F5**.

To manage Flow Graph breakpoints

- In Flow Graph Editor, right-click the applicable input or output node port, then do the following as needed:
 - To create a breakpoint, click **Add Breakpoint**. A red dot is displayed next to the node port.
 - To remove a breakpoint, click **Remove Breakpoint**.
 - To enable or disable a breakpoint, toggle the **Enabled** check box.
 - To remove all breakpoints on a node, or for all nodes on the entire flow graph, click **Remove Breakpoints for Node** or **Remove Breakpoints for Graph** respectively.

Every breakpoint can be converted to a tracepoint, which instead of pausing the game outputs the information about a triggered breakpoint to the console and to a log file. Simply right-click on the applicable breakpoint-enabled node port, then click **Tracepoint..** The red dot changes to a red diamond to indicate that the port has a tracepoint enabled on it.

Tracepoint data sent to the Console looks like this, as an example:

```
[TRACEPOINT HIT - FrameID: 71054] GRAPH: AnimObject1 (ID: 96) - NODE:  
Entity:MaterialParam (ID: 5) - PORT: ValueColor - VALUE:  
0.867136,0.005522,0.005522
```

Using Console Variables

The following Console variables can be used to troubleshoot Flow Graph issues.

- `fg_abortOnLoadError` — Aborts on a loading error of a flow graph, where 0=dialog, 1=log, 2=abort
- `fg_debugmodules` — 0=disabled, 1=show all modules, 2=show all modules and active modules
- `fg_debugmodules_filter` *filterstring* — Used to only show modules that match the *filterstring*

- `fg_iDebugNextStep` — - Step-by-step debugging
- `fg_iEnableFlowgraphNodeDebugging` — toggles flow graph debugging of nodes
- `fg_inspectorLog` — log inspector on Console
- `fg_noDebugText` — Don't display flow graph debugging text
- `fg_profile` — toggles flow graph profiling
- `fg_SystemEnable` — toggles Flow Graph system updates
- `gt_showFilter` — Filter string for flow graph tokens and game tokens
- `gt_showLines` — Specifies how many lines to display
- `gt_showPosX 0` — Shows the X-axis position
- `gt_showPosY` — Shows the Y-axis position
- `gt_show Value` — Shows game token and graph token state, where 1=screen and log , 2=screen only , 3=log only.

Placing Cached Shadows

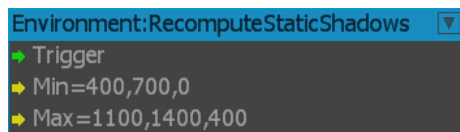
Cached shadows display shadow properties for an entire scene. It replaces the shadow cascades that appear farthest from the viewer and reduces the number of shadow draw calls per frame.

Note

To eliminate visible artifacts when time of day is updated or in scenes that have huge objects casting shadows in the distance, we recommend turning off cached shadows.

To specify placement of cached shadows, use the Flow Graph Editor. Before you trigger an update, compile all of your shaders to ensure that all objects are rendered into the cached shadows.

Use `Environment:RecomputeStaticShadows` for cached shadows. This node takes the minimum and maximum positions of the world space bounding area, and triggers the re-rendering of the cached shadows.



Recommended Settings

`r_ShadowsCache`

Default value: 4

Bounding area: 1000 x 1000 meters (recommended maximum, X/Y direction) and as small a range as possible (Z direction)

Related Console Variables

`r_ShadowsCache`

Replaces all sun cascades above the specified console variable (cvar) value with cached shadows.

Valid values: 0=no cached shadows | 1=replace first cascade and up | 2=replace second cascade and up | etc.

r_ShadowsStaticMapResolution

The resolution of the cached shadows. The cached shadows for mobile platforms has 16 bit precision and consumes 8 MB of video memory. The cached shadows for other platforms has 16 bit precision and consumes 128 MB of video memory.

Default value: 2048 (mobile platforms), 8192 (other platforms)

e_ShadowsStaticMapUpdate

Triggers update of the cached shadows.

Valid values: 0=no update | 1=one update | 2=continuous updates

e_ShadowsStaticObjectLod

The level of detail (LOD) used for rendering objects into the cached shadows.

Gems

Gems are packages that contain code and/or assets to augment your game projects. You can create and select gems to include in your project through the Lumberyard [Project Configurator \(p. 985\)](#). With the [Modular Gems System \(p. 779\)](#) you can choose the features and assets that you need for your game without including unnecessary components. The gems that you use are automatically detected and built through the integrated [Waf Build System \(p. 1318\)](#).

All Lumberyard gems are located in the `lumberyard_root_folder\dev\Gems` folder.

To enable gems, you use the **Project Configurator**, which you can launch from the [Lumberyard Setup Assistant \(p. 15\)](#).

To enable one or more gems

1. In the **Project Configurator**, select your active project and click **Set as default**.
2. Click **Enable Gems**.
3. Select the gems that you want to enable.
4. Click **Save**.

After enabling one or more gems, you must rebuild your project to make the gems function in the Lumberyard Editor.

To rebuild your project after enabling gems

1. Open a command line window and navigate to the `[Lumberyard root directory]\dev` directory.
2. Type `lmbwaf configure`. A success message at the end indicates a successful completion.
3. If the previous step was successful, at the same command line, type `lmbwaf build_win_x64_profile -p all`.

To view other build commands or variables to use for this step, see [Build Configuration \(p. 1335\)](#).

This step may take some time to complete. A success message at the end indicates a successful completion.

4. Start Lumberyard Editor. You are now ready to use your gems.

To create a new gem

1. Go to `lumberyard_root_folder\dev\Bin64\`, then open ProjectConfigurator.
2. Select the project and click **Enable Gems, Create a new Gem**.
3. Click in the **Name** box and type a name. Click **OK**. Only alphanumeric characters are allowed; no special characters or whitespaces are allowed in the name.
4. Close the **Project Configurator**.
5. Open the `lumberyard_root_folder\dev\Gems\gem_name\gem.json` file and specify the following gem metadata fields:
 - Version
 - DisplayName
 - Tags
 - IconPath

Topics

- [Modular Gems System \(p. 779\)](#)
- [Lumberyard Gems \(p. 782\)](#)

Modular Gems System

The Modular Gems system is a management infrastructure for sharing code and art assets between Lumberyard game projects. Modular Gems system consists of gems packages that are accessed and managed using the [Project Configurator \(p. 985\)](#). A gems package contains assets, code, `gem.json` file, and an icon file. For a list of available gems, see [Gems \(p. 778\)](#).

All Lumberyard gems are located in the `lumberyard_root_folder\dev\Gems` folder.

Gems can also be accessed through code, as in the following example:

```
#include <GemName/GemNameBus.h>
//...
EBUS_EVENT(GemName::GemNameRequestBus, MyFunction, withArgs);
```

Gem Assets

Assets function similarly to the way that they do in a normal game project. Each gem has an `Assets` folder containing models, textures, scripts, and animations that are accessed just as if they were in a game project. This is the root folder that Lumberyard uses to resolve the asset file path. For example, when Lumberyard is looking for `textures\rain\rainfall_ddn.tif`, it looks in `gem_root\Assets\textures\rain\rainfall_ddn.tif`.

Gem Code

Gems are loaded dynamically at runtime, are able to receive system events (init and shutdown, primarily), and communicate with other gems. The following items make up a gem's code component, located in the `<GemFolder>\Code` folder:

- `wscript`: This is the Waf build script. It is auto-generated by the template, and does not need to be changed by most authors. It contains all build configuration options, including target name, include paths, required libs, defines, and so on.

- `gemname.waf_files`: This is a JSON list of all files included in the project. The root object contains properties for each Uber File, and a special `NoUberFile` object. Each child object contains a named array of files, where the name is the filter that is used in generated projects. The gem template provides a default `.waf_files` list. All new files should be added to this.
- `gemname_tests.waf_files`: This is a JSON list of all test files for a gem, in the same format as `gemname.waf_files`.
- `Include/GemName` folder: This is where the gem's interface is. This folder can be included by other gems, and should contain no implementations or non-pure-virtual function definitions. The gem template provides a default `GemNameBus.h` that contains a `GemNameRequestBus` interface, which defines public functionality. For more information, see [Event Bus \(EBus\)](#).
- `Source` folder: This folder contains the following automatically generated files:
 - `StdAfx.h`: Includes frequently required files.
 - `StdAfx.cpp`: Includes `StdAfx.h`.
 - `GemNameGem.h`: Contains the definition of the actual `IGem` implementation class.
 - `GemNameSystemComponent.h`: Contains the definition of a System Component that handles calls to `GemNameRequestBus`.
 - `GemNameSystemComponent.cpp`: Contains the implementation of the `GemNameSystemComponent` class.
 - `GemNameModule.cpp`: Contains the `AZ::Module` class definition, which is used to register components and do additional component reflection.

Note

This class can be made to extend `CryHooksModule` (in `IGem.h`) instead of having `gEnv` attached automatically.

- `Tests` folder: This provides an example of how to build unit tests into your gems. All files in this folder should be added to `gemname_tests.waf_files`.

The `Tests` folder contains the `GemNameTest.cpp` file, which is ready for you to write gtests for your gem.

For more information about `waf` files and `wscript` files, see [Waf Build System \(p. 1318\)](#).

Gem JSON File

This file contains metadata for the gem. The following `gem.json` file is for the `LightningArc` Gem:

```
{
  "Dependencies": [                                // Optional,
  defaults to []
    {
      "Uuid": "d378b5a7b47747d0a7aa741945df58f3", // Required
      "VersionConstraints": [                       // Required
        "~>1.0"
      ],
      "_comment": "GameEffectSystem"              // Useful comment
    }
  ],
  "GemFormatVersion": 3,                          // Required
  "LinkType": "Dynamic",                          // Required
  "Name": "LightningArc",                         // Required
  "Summary": "Provides an entity that can be used to produce electricity effects.", // Optional, defaults to ""
  "Tags": ["Weather", "Weather Effects", "Sky"], // Optional,
  defaults to []
}
```

```
"DisplayName": "Lightning Arc", // Optional,  
defaults to [Name]  
  "Uuid": "4c28210b23544635aa15be668dbff15d", // Required  
  "Version": "1.0.0", // Required  
  "IconPath": "preview.png" // Optional,  
defaults to ""  
}
```

A `<GemFolder>\gem.json` file defines the following gem properties:

- **Dependencies:** The uuids and versions of other gems that this gem depends on. Acceptable version specifiers are made of an operator and a version number. Some examples:
 - `==1.2.3`: Minimum: 1.2.3 Maximum: 1.2.3
 - `>=1.2.3`: Minimum: 1.2.3 Maximum: None
 - `<=1.2.3`: Minimum: None Maximum: 1.2.3
 - `>2.0.0`: Minimum: 1.0.0 (exclusive) Maximum: None
 - `<2.0.0`: Minimum: None Maximum: 2.0.0 (exclusive)
 - `~>1.2.3`: Minimum: 1.2.3 Maximum: 1.3.0 (exclusive)
 - `~>1.2`: Minimum: 1.2.0 Maximum: 2.0.0 (exclusive)
 - `*`: Allow any version (not recommended, overwrites all other constraints)
- **GemFormatVersion:** The `GemFormatVersion` value is versioning for how a gem is built. Gems from Lumberyard 1.4 and earlier (legacy gems) all have a `GemFormatVersion` value of 2. Starting in Lumberyard 1.5, all the gems included with Lumberyard are AZ modules and have a `GemFormatVersion` value of 3. This tells Lumberyard that the gem is an AZ module and that it should be loaded accordingly.
- **LinkType:** How other gems and game projects should link against this gem:
 - `Dynamic`: Produces a .dll file and does no linking.
 - `DynamicStatic`: Produces a .dll file and links all dependent projects against the .dll file using an import library.
 - `NoCode`: Produces no .dll or .lib file. The gem has assets but no code.
- **Name:** The name of the gem.
- **Summary:** A short description of the gem.
- **Tags:** A list of tags describing the gem.
- **DisplayName:** The friendly name of the gem, used in the UI.
- **Uuid:** The unique ID of the gem. Used to identify the gem to the engine.
- **Version:** The API version of the gem. The version should follow the [Semantic Versioning](#) specification.
- **IconPath:** The path from the gem folder to the icon to display. It may be a .jpg, .png, or .gif, and should be 160x90px.

Gem List File

The `gems.json` list file, found at the root of each project directory, lists all of the gems used in the project. An example gem list follows.

```
{  
  "GemListFormatVersion": 2,  
  "Gems": [  
    {  
      "Path": "Gems/Rain", // Required
```

```
        "Uuid": "e5f049ad7f534847a89c27b7339cf6a6", // Required
        "Version": "0.1.0", // Required
        "_comment": "Rain" // Useful comment
    }
}
}
```

Lumberyard Gems

Lumberyard ships with the following gems that are ready to be enabled:

Topics

- [Boids Gem \(p. 782\)](#)
- [Camera Framework Gem \(p. 789\)](#)
- [Cloud Gem \(p. 789\)](#)
- [Cloud Canvas Gem \(p. 795\)](#)
- [GameEffect Gem \(p. 796\)](#)
- [GameLift Gem \(p. 796\)](#)
- [Gestures Gem \(p. 797\)](#)
- [Input Management Framework Gem \(p. 807\)](#)
- [Lightning Arc Gem \(p. 807\)](#)
- [Metastream Gem \(p. 814\)](#)
- [Multiplayer Gem \(p. 818\)](#)
- [Physics Entities Gem \(p. 822\)](#)
- [Process Life Management Gem \(p. 822\)](#)
- [Rain Gem \(p. 823\)](#)
- [Snow Gem \(p. 827\)](#)
- [Substance Gem \(p. 831\)](#)
- [Tornadoes Gem \(p. 831\)](#)
- [UiBasics Gem \(p. 836\)](#)
- [UiDemo Gem \(p. 836\)](#)
- [User Login Default Gem \(p. 836\)](#)
- [Woodland Asset Collection Gem \(p. 836\)](#)

Boids Gem

The Boids Gem provides entities that simulate animated animals that produce sound, exhibit group behavior, and avoid obstacles. Their complex behavior arises from the interaction of an individual agent boid with other boids and their environment.

A boids entity is a group of animals. You can control such aspects as the total number of boids, their mass, flocking behavior, speed, interaction with the player, and more. All boids entities exhibit by default a combination of basic motion, player avoidance, and flocking behavior.

The boids entity can also be affected by other entities; for example, a rain entity placed in the same scene with a turtles entity results in wet turtles.

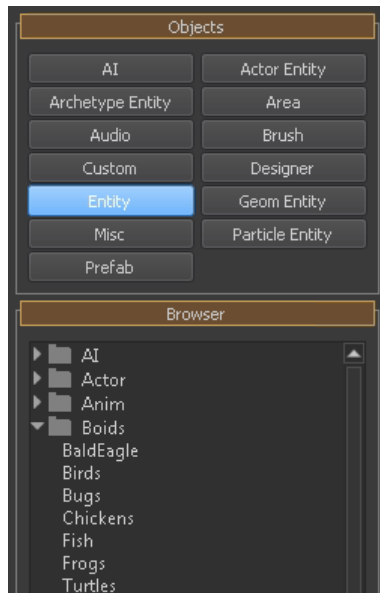


Topics

- [Configuring the Boids Gem \(p. 784\)](#)
- [Boids Entity Flow Graph Nodes \(p. 787\)](#)

To place Boids entities

1. In the **Rollup Bar**, on the Objects tab, click **Entity**.
2. Expand **Boids**.



3. Drag one of the boids entities into your level in the **Perspective** viewport.

Configuring the Boids Gem

You can configure the **Entity Properties** and **Entity Params** (p. 432) for the boids entity to specify such features as the number of boids to spawn per group, flocking behavior, the character model to use, and more.

The following table lists boids entity properties and their descriptions. As noted, certain properties appear for only specific boids.

Note

Boids spawn on terrain, not on objects placed on the terrain.

Boids Entity Properties

Property Name	Description
Boids	
Behavior	Sets movement behavior. <ul style="list-style-type: none">• 0 – Crawling bugs (for example, beetles)• 1 – Flying insects (for example, dragonflies)• 2 – Jumping bugs (for example, grasshoppers)
Count	Number of individuals to spawn per boid group.
Invulnerable	Sets invulnerability, where boids entities cannot be killed by anything.

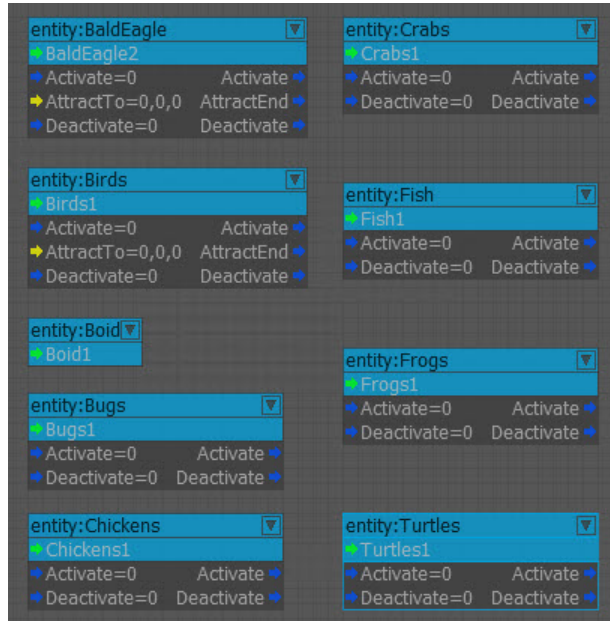
Property Name	Description
	<p>When invulnerability is not set, the following can kill boids:</p> <ul style="list-style-type: none"> • Collisions with other entities at speeds greater than 1. • Being thrown at speeds greater than 5 (applies to chickens, turtles, and frogs only). • Collision with a particle moving at a speed greater than 5 (applies to chickens, turtles, and frogs only). • Collision when <code>OnBoidHit</code> function is used.
Mass (kg)	Mass of each individual in the group. Used when physicalizing (p. 157) the boid entity.
Model	3D model file used for the boid representation. You can use geometry files (*.cge, *.chr, *.skin, *.cdf) for this property. To change the model, click in the property, and then click the folder icon. Navigate to and open the file you want to use. The bugs boid entity has 5 Model entries for specifying 5 different models.
Flocking	
AttractDistMax	Maximum separation distance in meters at which boids can interact with each other. Boids do not interact with each other at distances beyond this range.
AttractDistMin	Minimum separation distance in meters between boids before FactorSeparation force affects them.
EnableFlocking	When selected, enables flocking behavior within a group. This means that the boids congregate or mass together.
FactorAlign	Multiplier that determines how closely boids in a group maneuver in the same direction.
FactorCohesion	Multiplier that determines how closely boids in a group congregate.
FactorSeparation	Multiplier that determines how strongly boids in a group repel one another. Avoids crowding flock mates when closer than AttractDistMin .
FieldOfViewAngle	Viewing angle within which each boid can consider other boids flock mates.
Ground	
	The Ground group of properties appears only for boids entities with flight and applies only when boids are walking on the ground, which occurs only in game mode, and not in edit mode.
FactorAlign	Multiplier that determines how closely boids in a group maneuver in the same direction.
FactorCohesion	Multiplier that determines how closely boids in a group congregate.
FactorOrigin	Multiplier that determines how strongly boids in a group are attracted to their point of origin.
FactorSeparation	Multiplier that determines how strongly boids in a group repel one another.
HeightOffset	Boids vertical offset from the ground.
OnGroundIdleDurationMax	Maximum amount of time that boids idle on the ground.

Property Name	Description
OnGroundIdleDurationMin	Minimum amount of time that boids idle on the ground.
OnGroundWalkDurationMax	Maximum amount of time that boids walk on the ground.
OnGroundWalkDurationMin	Minimum amount of time that boids walk on the ground.
WalkSpeed	Speed at which boids walk on the ground when they land.
WalkToIdleDuration	Time it takes for boids to transition from walk to idle.
Movement	
FactorAvoidLand	Multiplier that determines how strongly boids in a group avoid land or water.
FactorHeight	Multiplier that determines how strongly boids in a group are kept at their original height.
FactorOrigin	Multiplier that determines how strongly boids in a group are attracted to their point of origin.
FactorTakeOff	Speed of vertical movement during takeoff. Appears only for boids entities with flight.
FlightTime	Duration of flight before attempting to land. Appears only for boids entities with flight.
FactorRandomAcceleration	Multiplier that determines that randomness of acceleration. Appears only for fish boids.
HeightMax	Maximum height above land to which boids can fly.
HeightMin	Minimum height above land at which boids can fly.
LandDecelerationHeight	Height at which boids begin to decelerate when landing. Appears only for boids entities with flight.
MaxAnimSpeed	Multiplier for maximum deviation allowed from original animation speed, for those boids with animations.
SpeedMax	Maximum speed (meters/second) at which boids can move.
SpeedMin	Minimum speed (meters/second) at which boids can move.
Options	
Activate	Activates the selected boid entity from the start of the level. Boids can also be activated at a later stage with the <code>activate</code> event.
AnimationDist	Maximum distance from the camera at which animations are updated. Appears only for boids entities with flight.
AvoidWater	Value that determines how strongly boids avoid bodies of water. Appears only for boids that move on land.
FollowPlayer	When selected, boids flock toward current player position, which is their point of origin. Boids stay within value set by Radius . If boids move too far from the player, they reappear on the other side of the radius area.
NoLanding	When selected, boids with flight do not land.

Property Name	Description
ObstacleAvoidance	When selected, boids are diverted from physical obstacles. This feature is resource-intensive, so use it cautiously.
PickableMessage	Message that appears if a boid is able to be picked up. Appears for all boids except fish.
PickableWhenAlive	When selected, boid can be picked up when alive. Appears for all boids except fish.
PickableWhenDead	When selected, boid can be picked up when dead. Appears for all boids except fish.
Radius	Maximum radius in meters that boids can move from their flock point of origin.
SpawnFromPoint	When selected, boids spawn at the boid entity position.
StartOnGround	When selected, boids spawn on the ground. When unselected, boids spawn in the air.
VisibilityDist	Maximum camera distance in meters from which the entire flock is visible. If the player camera's distance from the flock exceeds this value, boids are not rendered.
ParticleEffects	
EffectsScale	Scale of the particle effect to be displayed. Appears only for frogs.
waterJumpSplash	Name of the splash particle effect to be displayed when a boid jumps into the water. Appears only for frogs.

Boids Entity Flow Graph Nodes

To place a boids entity flow graph node into a [flow graph](#) (p. 487), select the entity in your **Perspective** viewport. Then right-click and select **Create Flow Graph**. If working with a level flowgraph, select the entity in your **Perspective** viewport. Then in your flow graph, right-click and click **Add Selected Entity**. A flow graph node appears with the title **entity:Entity name**.



entity:*Boid Entity Type*

Inputs

Entity Name

Selected entity's name or label. Displays **<Graph Entity>** if the flow graph is an [entity file \(p. 488\)](#).

Activate

Activates the entity.

Deactivate

Deactivates the entity.

AttractTo

Attracts the entity to a specific XYZ coordinate in the level.

Applies only to the birds and bald eagles entities.

Outputs

Activate

Triggers output when the entity is activated.

Deactivate

Triggers output when the entity is deactivated.

AttractEnd

Triggers output when the entity's distance is less than 5 meters from the attraction point (**AttractTo** input).

Applies only to the birds and bald eagles entities.

Lua Bindings for Boids

Individual boids have Lua-specific behavior. These scripts are available in `dev\Gems\Boids\Assets\Scripts\Entities\Boids`.

The following boids functions are bound from C++ to Lua:

- CreateFlock
- SetFlockParams
- EnableFlock
- SetFlockPercentEnabled
- OnBoidHit
- SetAttractionPoint
- CanPickup
- GetUsableMessage
- OnPickup

Console Variable for Boids

The console variable `boids_enable` is defined in `dev\Gems\Boids\Code\source\ScriptBind_Boids.cpp`.

The `count` value for boids can be modified by the CVar `e_ObjQuality`.

Camera Framework Gem

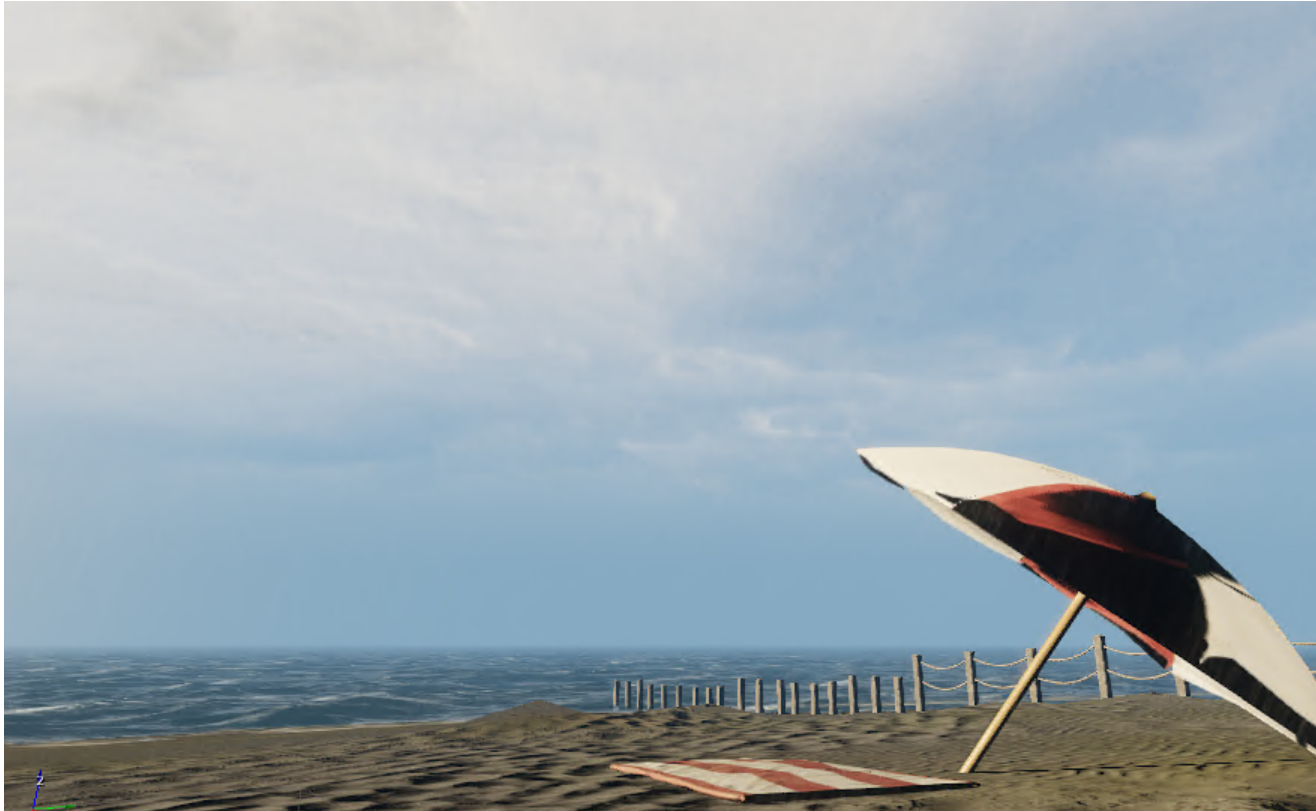
The Camera Framework Gem is a base upon which you can build more complex camera systems. This Gem contains the `CameraComponent` and the `CameraRigComponent`, which work together to define a basic camera and its control rig. You can customize the `CameraRigComponent` through three different behaviors:

- Target acquiring behavior
- Target transform modifying behaviors
- Final camera transform modifying behaviors

Cloud Gem

The Clouds Gem creates realistic and detailed cloud and weather effects in your game levels. You can create clouds with either simple, sprite-based shading, or more complex, voxelized 3D volume shading. To enable the Clouds Gem in your project, see [Gems \(p. 778\)](#).

For more information about working with clouds, including setting cloud shading parameters, adding 3D cloud shadows, and creating 3D cloud templates, see [Adding Clouds \(p. 870\)](#).



Topics

- [Placing Simple Clouds \(p. 790\)](#)
- [Placing Complex Clouds \(p. 792\)](#)

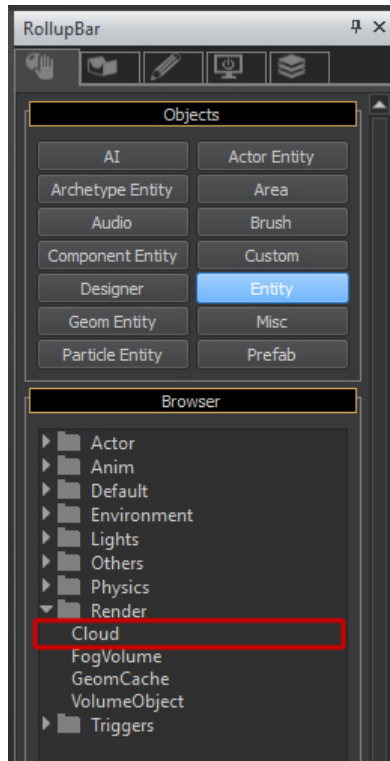
Placing Simple Clouds

You can place simple clouds with sprite-based shading and customize it for your level by choosing your cloud texture and modifying such properties as movement speed, size, movement from wind, and so on.



To add simple clouds to your level

1. In the **Rollup Bar's Objects** tab, click **Entity**.
2. Under **Browser**, expand **Render**.
3. Drag the **Cloud** entity into your scene.



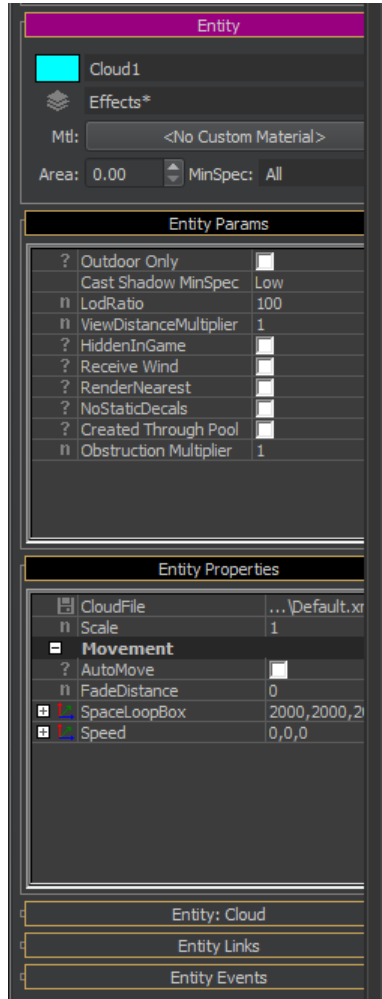
Files Associated with Simple Clouds

The following are files associated with simple clouds.

Filename	Location
CloudFile	Lib\Clouds\default.xml
Script	Scripts\Entities\Render\cloud.lua
Entity	Entities\cloud.ent

Configuring Simple Clouds

You can configure the properties for your simple clouds under **Entity Params** and **Entity Properties**.



Properties	Description
CloudFile	The .xm1 file containing the description of the cloud
Scale	Deprecated
Movement	
AutoMove	Enables cloud movement
FadeDistance	The distance in meters at which the cloud fades in when moving from one side of the space loop box to the other.
SpaceLoopBox	The size of the box in which the volume object moves from one end to the other
Speed	The rate of movement in the x, y, and z dimensions

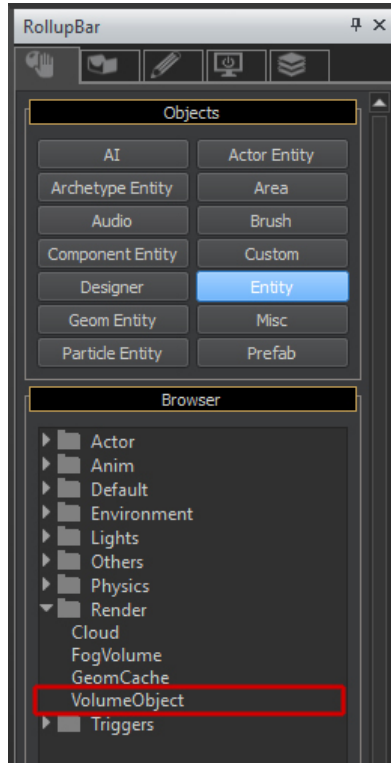
Placing Complex Clouds

You can place more complex clouds, also called volume objects, which feature complex voxelized three-dimensional volume shading.



To add complex clouds to your level

1. In the **Rollup Bar's Object's** tab, click **Entity**.
2. Under **Browser**, expand **Render**.
3. Drag the **VolumeObject** entity into your scene.



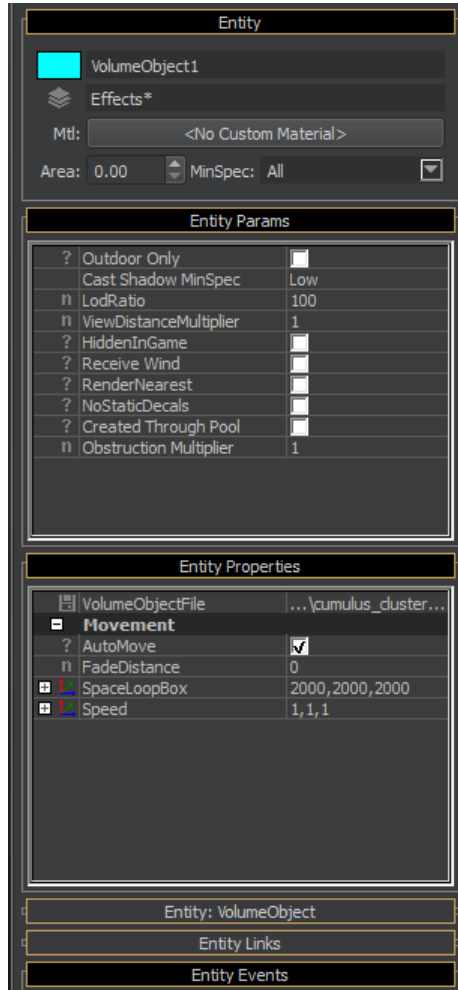
Files Associated with Complex Clouds

The following are files associated with volume objects, or complex clouds.

Filename	Location
CloudFile	Lib\CLOUDS\default.xml
Script	Scripts\Entities\Render\volumeobject.lua
Entity	Entities\volumeobject.ent

Configuring Complex Clouds

You can configure the properties for your complex clouds under **Entity Params** and **Entity Properties**.



Properties	Description
VolumeObjectFile	The .xml file containing the description of the cloud
Movement	
AutoMove	Enables volume object movement
FadeDistance	The distance in meters at which the cloud fades in when moving from one side of the space loop box to the other
SpaceLoopBox	The size of the box in which the volume object moves from one end to the other
Speed	The rate of movement in the x, y, and z dimensions

Cloud Canvas Gem

The Cloud Canvas Gem enables you to use Cloud Canvas visual scripting to AWS services. With Cloud Canvas you can build connected game features that use Amazon DynamoDB (DynamoDB), Amazon Lambda, Amazon Simple Storage Service (Amazon S3), Amazon Cognito, Amazon Simple Notification Service (Amazon SNS), and Amazon Simple Queue Service (Amazon SQS). You can also create cloud-hosted features such as daily gifts, in-game messages, leaderboards, notifications,

server-side combat resolution, and asynchronous multiplayer gameplay (e.g. card games, word games, ghost racers, etc.). Cloud Canvas eliminates the need for you to acquire, configure, or operate host servers yourself, and reduces or eliminates the need to write server code for your connected gameplay features.

AWS services accessed via Cloud Canvas may be subject to separate charges and additional terms. For more information, see [Cloud Canvas](#) in the *Lumberyard Developer Guide*.

GameEffect Gem

The Game Effect System Gem provides fundamentals for creating and managing the visual effects of the Lightning Arc Gem. If you install the [Lightning Arc Gem \(p. 807\)](#), you must also install the Game Effect System Gem. The Lightning Arc Gem is the only Lumberyard gem that is dependent on the Game Effect System Gem.

GameLift Gem

The GameLift Gem provides two flow graph nodes to support Amazon GameLift, which is an AWS service for deploying, operating, and scaling session-based multiplayer games. With Amazon GameLift, Amazon Lumberyard developers can quickly scale high-performance game servers up and down to meet player demand, without any additional engineering effort or upfront costs.

Topics

- [GameLift:Start node \(p. 796\)](#)
- [GameLift:CreateGameSession node \(p. 796\)](#)

GameLift:Start node

Used to start the GameLift session service.

Node Inputs

Activate

AWSAccessKey

AWSSecretKey

AWSRegion

Endpoint

FleetID

AliasID

PlayerID

Node Outputs

Success

Failed

GameLift:CreateGameSession node

Used to create a GameLift game session.

Node Inputs

Activate

ServerName

Map

MaxPlayers

Node Outputs

Success

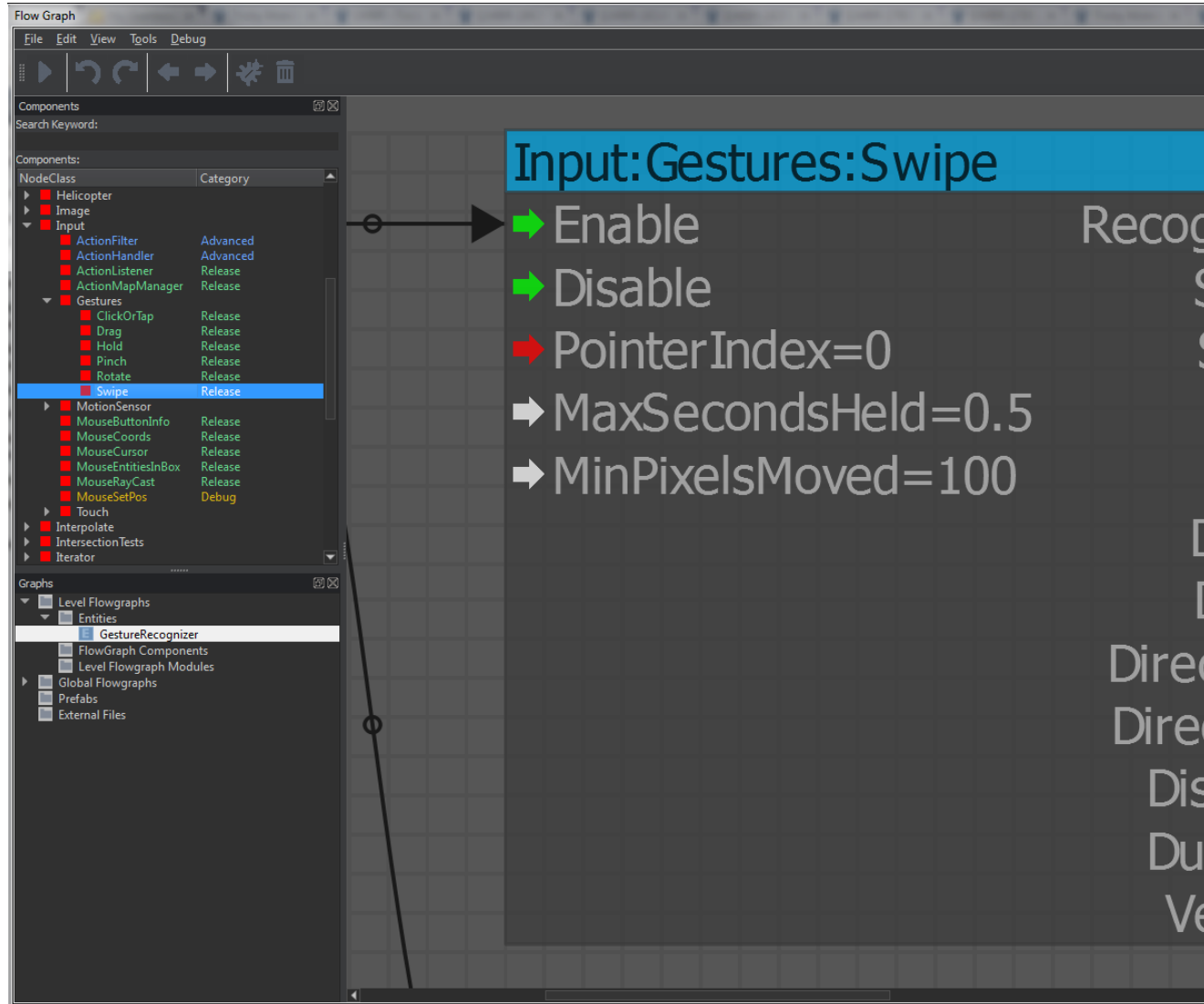
Failed

Gestures Gem

The Gestures Gem processes raw input to detect some of the most common gesture-based input actions, including the following:

- Tap or click – Single-touch, discrete gesture
- Drag or pan – Single-touch, continuous gesture
- Hold or press – Single-touch, continuous gesture
- Swipe – Single-touch, discrete gesture
- Pinch – Multiple-touch, continuous gesture
- Rotate – Multiple-touch, continuous gesture

You can configure and register gesture listeners using either C++ or flow graph nodes that are exposed through the Gestures Gem.



Multiple-touch gestures (such as pinch and rotate) can be recognized only through multiple simultaneous touches on a supported touch screen (currently, mobile devices running iOS or Android). On the other hand, single-touch gestures (such as tap, drag, hold, and swipe) function identically with both supported touch screens and mouse input on a PC. The underlying C++ gesture recognition framework can be easily extended to write your own custom gestures and expose them through the **Flow Graph** editor.

Gestures Flow Graph Nodes

The Gestures Gem's flow graph nodes are contained in the **Input, Gestures** filter in the **Flow Graph** editor. Each node contains a number of input ports that you can use to configure how the gesture is recognized. Data is sent through output nodes each time the gesture is recognized (for discrete gestures such as tap or swipe), or for each frame while the gesture is being recognized (for continuous gestures such as drag, hold, pinch, and rotate).

You can use these flow graph nodes to configure gestures-related settings.

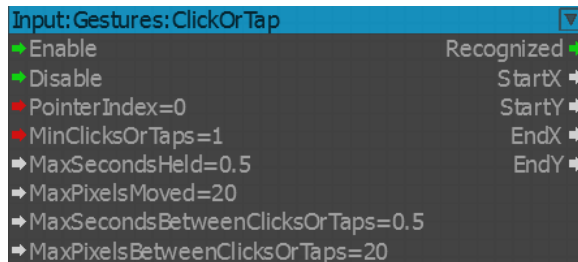
For more information on the Gestures Gem, see [Gestures Gem \(p. 797\)](#).

Topics

- [ClickorTap](#) (p. 799)
- [Drag](#) (p. 800)
- [Hold](#) (p. 801)
- [Pinch](#) (p. 803)
- [Rotate](#) (p. 804)
- [Swipe](#) (p. 805)

ClickorTap

Recognizes a discrete (or series of discrete) click (or tap) gestures.



Inputs

Enable

Enables gesture recognizer.

Disable

Disables gesture recognizer.

PointerIndex

The pointer (button or finger) index to track.

Default value: 0

Type: Integer

MinClicksOrTaps

The minimum number of clicks or taps required for the gesture to be recognized.

Default value: 1

Type: Integer

MaxSecondsHeld

The maximum time in seconds a gesture can be held before the gesture stops being recognized.

Default value: .5

Type: Float

MaxPixelsMoved

The maximum distance in pixels allowed to move while being held before the gesture stops being recognized.

Default value: 20

Type: Float

MaxSecondsBetweenClicksOrTaps

The maximum time in seconds allowed between clicks or taps (only used when `MinClicksOrTaps > 1`).

Default value: .5

Type: Float

MaxPixelsBetweenClicksOrTaps

The maximum distance in pixels allowed between clicks or taps (only used when `MinClicksOrTaps > 1`).

Default value: 20

Type: Float

Outputs

Recognized

Activated when a discrete (or series of discrete) click (or tap) gestures is recognized.

StartX

Starting X screen position of the click or tap in pixels.

Type: Float

StartY

Starting Y screen position of the click or tap in pixels.

Type: Float

EndX

Final X screen position of the click or tap in pixels.

Type: Float

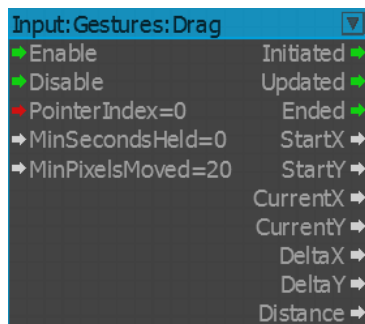
EndY

Final Y screen position of the click or tap in pixels.

Type: Float

Drag

Recognizes continuous drag gestures.



Inputs

Enable

Enables gesture recognizer.

Disable

Disables gesture recognizer.

PointerIndex

The pointer (button or finger) index to track.

Default value: 0

Type: Integer

MinSecondsHeld

The minimum time in seconds after the initial press before a drag is recognized.

Default value: 0

Type: Float

MinPixelsMoved

The minimum distance in pixels that must be dragged before a drag is recognized.

Default value: 20

Type: Float

Outputs

Recognized

Activated when a continuous drag gesture is initiated.

Updated

Activated when a continuous drag gesture is updated.

Ended

Activated when a continuous drag gesture is ended.

StartX

X pixel position where the drag started.

Type: Float

StartY

Y pixel position where the drag started.

Type: Float

CurrentX

Current X pixel position (or where the drag ended).

Type: Float

CurrentY

Current Y pixel position (or where the drag ended).

Type: Float

DeltaX

X pixels dragged ($\text{CurrentX} - \text{StartX}$).

Type: Float

DeltaY

Y pixels dragged ($\text{CurrentY} - \text{StartY}$).

Type: Float

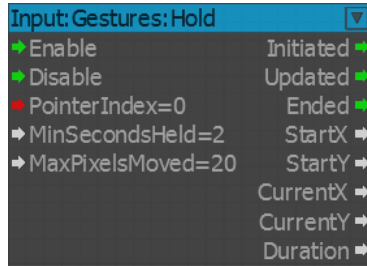
Distance

Pixel distance from the drag's start position to its current (or end) position.

Type: Float

Hold

Recognizes continuous hold gestures.



Inputs

Enable

Enables gesture recognizer.

Disable

Disables gesture recognizer.

PointerIndex

The pointer (button or finger) index to track.

Default value: 0

Type: Integer

MinSecondsHeld

The minimum time in seconds after the initial press before a hold is recognized.

Default value: 2

Type: Float

MaxPixelsMoved

The maximum distance in pixels that can be moved before a hold stops being recognized.

Default value: 20

Type: Float

Outputs

Initiated

Activated when a continuous hold gesture is initiated.

Updated

Activated when a continuous hold gesture is updated.

Ended

Activated when a continuous hold gesture is ended.

StartX

X pixel position where the hold started.

Type: Float

StartY

Y pixel position where the hold started.

Type: Float

CurrentX

X pixel position where the hold is currently (or where it ended).

Type: Float

CurrentY

Y pixel position where the hold is currently (or where it ended).

Type: Float

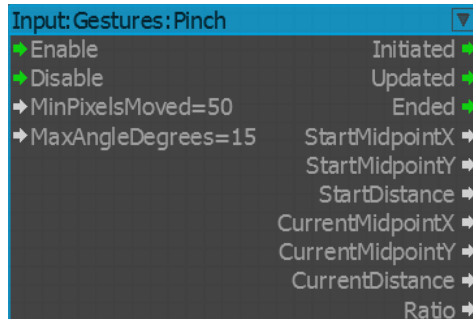
Duration

Duration of the hold in seconds.

Type: Float

Pinch

Recognizes continuous pinch gestures (the primary and secondary touches moving towards or away from each other).



Inputs

Enable

Enables gesture recognizer.

Disable

Disables gesture recognizer.

MinPixelsMoved

The minimum distance in pixels that must be pinched before a pinch is recognized.

Default value: 50

Type: Float

MaxAngleDegrees

The maximum angle in degrees that a pinch can deviate before it is recognized.

Default value: 15

Type: Float

Outputs

Initiated

Activated when a continuous pinch gesture is initiated.

Updated

Activated when a continuous pinch gesture is updated.

Ended

Activated when a continuous pinch gesture is ended.

StartMidpointX

X pixel position (midpoint) where the pinch started.

Type: Float

StartMidpointY

Y pixel position (midpoint) where the pinch started.

Type: Float

StartDistance

Pixel distance between the two touch positions when the pinch started.

Type: Float

CurrentMidpointX

Current X pixel position (midpoint) of the pinch (or where it ended).

Type: Float

CurrentMidpointY

Current Y pixel position (midpoint) of the pinch (or where it ended).

Type: Float

CurrentDistance

Current pixel distance between the two touch positions (or when the pinch ended).

Type: Float

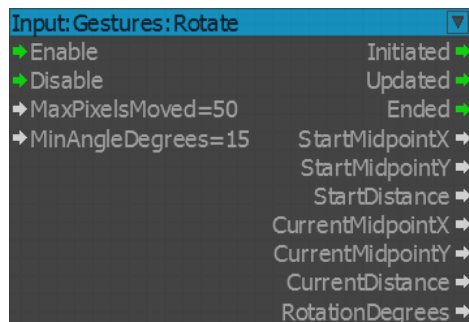
Ratio

The ratio of the pinch ($\text{CurrentDistance} / \text{StartDistance}$).

Type: Float

Rotate

Recognizes continuous rotate gestures (the primary and/or secondary touches moving in a circular motion around the other).



Inputs

Enable

Enables gesture recognizer.

Disable

Disables gesture recognizer.

MaxPixelsMoved

The maximum distance in pixels that the touches can move toward or away from each other before a rotate is recognized.

Default value: 50

Type: Float

MinAngleDegrees

The minimum angle in degrees that must be rotated before the gesture is recognized.

Default value: 15

Type: Float

Outputs

Initiated

Activated when a continuous rotate gesture is initiated.

Updated

Activated when a continuous rotate gesture is updated.

Ended

Activated when a continuous rotate gesture is ended.

StartMidpointX

X pixel position (midpoint) where the rotate started.

Type: Float

StartMidpointY

Y pixel position (midpoint) where the rotate started.

Type: Float

StartDistance

Pixel distance between the two touch positions when the rotate started.

Type: Float

CurrentMidpointX

Current X pixel position (midpoint) of the rotate (or where it ended).

Type: Float

CurrentMidpointY

Current Y pixel position (midpoint) of the rotate (or where it ended).

Type: Float

CurrentDistance

Pixel distance between the two touch positions currently (or when the rotate ended).

Type: Float

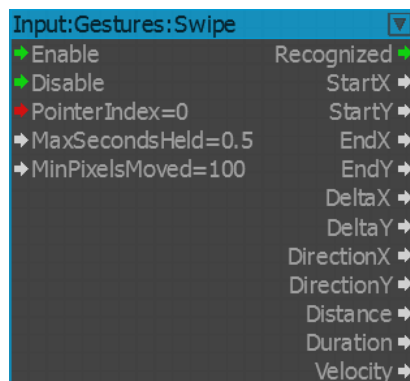
RotationDegrees

The current rotation in degrees in the range [-180, 180].

Type: Float

Swipe

Recognizes discrete swipe gestures.



Inputs

Enable

Enables gesture recognizer.

Disable

Disables gesture recognizer.

PointerIndex

The pointer (button or finger) index to track.

Default value: 0

Type: Integer

MaxSecondsHeld

The maximum time in seconds after the initial press for a swipe to be recognized.

Default value: .5

Type: Float

MinPixelsMoved

The minimum distance in pixels that must be moved before a swipe is recognized.

Default value: 100

Type: Float

Outputs

Recognized

Activated when a discrete swipe gesture is recognized.

StartX

X pixel position where the swipe started.

Type: Float

StartY

Y pixel position where the swipe started.

Type: Float

EndX

X pixel position where the swipe ended.

Type: Float

EndY

Y pixel position where the swipe ended.

Type: Float

DeltaX

X pixels swiped ($EndX - StartX$)

Type: Float

DeltaY

Y pixels swiped ($EndY - StartY$).

Type: Float

DirectionX

X direction of the swipe (normalized $DeltaX$, $DeltaY$).

Type: Float

DirectionY

Y direction of the swipe (normalized `DeltaX`, `DeltaY`).

Type: Float

Distance

Distance of the swipe in pixels.

Type: Float

Duration

Duration of the swipe in seconds.

Type: Float

Velocity

Velocity of the swipe in pixels per second.

Type: Float

C++

From C++, you can access the Gestures Gem interface using a convenience function such as the following:

```
#include <Gestures/IGesturesGem.h>
IGesturesGem* GetIGesturesGem()
{
    ISystem* system = GetISystem();
    IGemManager* gemManager = system ? system->GetGemManager() :
nullptr;
    return gemManager ? gemManager->GetGem<Gestures::IGesturesGem>() :
nullptr;
}
```

For examples of how to create and register your own gesture recognizers from C++, refer to the various `GestureRecognizer*FlowNode.cpp` files, which contain the code that drives the respective flow graph nodes.

Input Management Framework Gem

The Input Management Framework Gem is in preview release and is subject to change.

This Gem provides a framework for managing cross-platform game input such as keyboard, controller, and touch in Lumberyard using the component entity system.

Lightning Arc Gem

The Lightning Arc Gem creates realistic electric arcing and sparking effects between points in a level.

While active, the entity sparks a new electrical arc to the assigned target entities randomly. The entity is able to trigger new sparks in either game mode or in **AI/Physics** mode.



Using the LightningArc Sample

The LightningArc Sample uses the LightningArc gem to demonstrate the various prescribed arc types.

Topics

- [Enabling the Lightning Arc Gem \(p. 808\)](#)
- [Placing Lightning Arc \(p. 808\)](#)
- [Configuring the Lightning Arc \(p. 810\)](#)
- [Customizing a Lightning Arc Preset \(p. 811\)](#)

Enabling the Lightning Arc Gem

You enable the Lightning Arc Gem from **Project Configurator**. You must also enable the Game Effect Gem, as the Lightning Arc Gem is dependent on it. This and other dependencies are listed in **Project Configurator**. For more information, see [Project Configurator \(p. 985\)](#).

To enable the lightning arc Gem

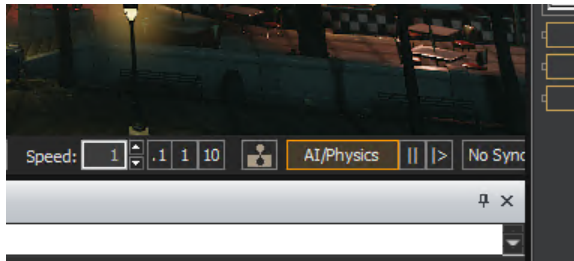
1. Start **Project Configurator** and click **Enable Packages**.
2. Select **Lightning Arc** and **Game Effect System**.
3. Click **Save**.
4. Use the procedure in [Gems \(p. 778\)](#) to rebuild your project.

Placing Lightning Arc

When you place a lightning arc entity, you must specify at least one target. The lightning arcs between the lightning arc entity and each target that is linked. The lightning arc appears in the Lumberyard Editor when you turn on **AI/Physics** or enter game mode (**Ctrl + G**).

To place a lightning arc

1. In the **Rollup Bar's Object** tab, click **Entity**. Under **Browser**, expand **Environment**, and then select **LightningArc**. Drag **LightningArc** into your scene.
2. Beneath **Entity Properties**, ensure that **Active** is selected.
3. Click **AI/Physics** in the bottom toolbar. This makes the lightning arc visible in Lumberyard Editor after you place and link the targets.

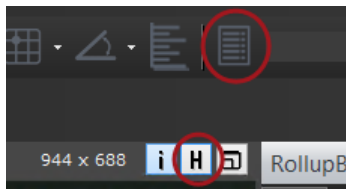


4. To place one or more targets, in the **Rollup Bar's Objects** tab, click **AI**. Under **Object Type**, click **Tagpoint**.
5. Move your mouse into the scene, and click to place the tag point where your lightning will arc.
6. To link your tag point, select your lightning arc entity in the scene.

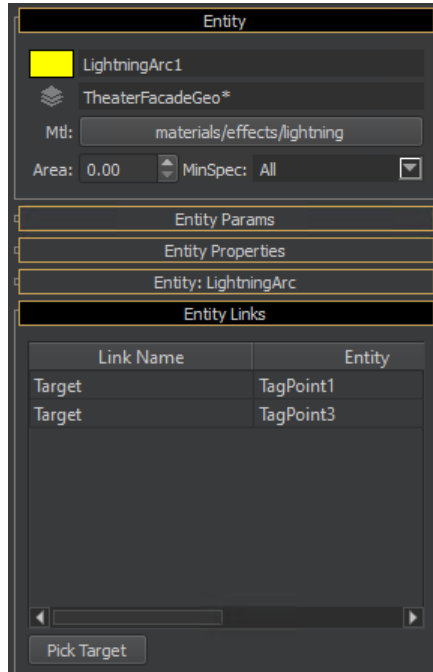
Note

If your entities are currently hidden, click the **H** icon at the top of the **Perspective** viewport to reveal them.

Alternatively, you can use the object selector to select an object by name.



7. If necessary, scroll down or collapse other headings in the **Rollup Bar** to find **Entity Links**. Click **Pick Target**. Select the tag point you placed. Once it appears in the **Link Name** list, double-click the link name and change it to **Target**.



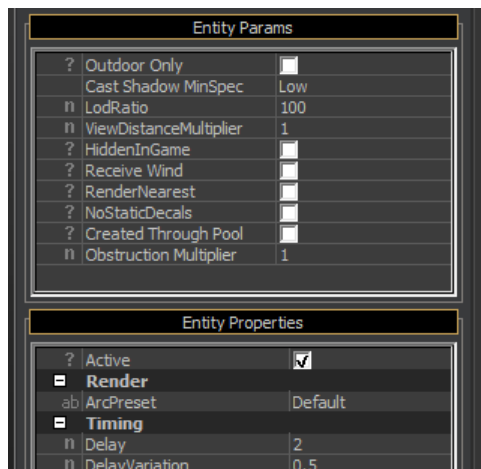
8. Assign a lightning material: Beneath **Entity**, click in the **Mtl** text box. The **Material Editor** appears.
9. Expand `materials\effects`. Right-click the desired lightning effect. Then click **Assign to Selected Objects**. Close the **Material Editor**.

Configuring the Lightning Arc

You can configure the properties for the lightning arc entity to make the lightning arc show outside only, toggle wind effects, add delays and variations between arcs, and more. You can also carefully customize your lightning arcs by selecting different presets for the type of arc generated.

To configure lightning arc entity parameters and properties

1. In the **Perspective** viewport, select the lightning arc entity you want to configure.
2. Beneath **Entity Params** (p. 432) and **Entity Properties**, select or clear check boxes for the preferred effects.



Lightning Arc Entity Properties

Properties	Description
Active	Activates the effect
Render	
ArcPreset	Sets the specified arc preset defined in the <code>lightningarceffects.xml</code> file as explained in Customizing a Lightning Arc Preset (p. 811) .
Timing	
Delay	Sets the delay time between arcs
DelayVariation	Sets the variation of the delay based on the delay time

Customizing a Lightning Arc Preset

You can customize your lightning arc entity using the presets in the `lightningarceffects.xml` file. You can also copy and modify existing presets to create your own customized lightning arc presets.

To use a lightning arc preset

1. In Lumberyard Editor, use the **Select** tool to select the lightning arc entity you want to customize.
2. In a text editor, open `\dev\Gems\LightningArc\Assets\libs\lightningarc\lightningarceffects.xml` in the Lumberyard root directory (`\lumberyard\dev`).
3. Choose one of the existing presets from the `lightningarceffects.xml` file (follows **Arc name** in the example) and, in Lumberyard Editor, under **Entity Properties**, type your chosen **Arc name** into the **ArcPreset** field .

For example, type **ExtendedArc** or **KickSparks**, which are existing names of presets as shown in the following `lightningarceffects.xml` file. This sample shows only the partial contents; open the file on your computer to view the full contents of the file.

```
<LightningArc>

<Arc name="Default">
  <param name="lightningDeviation" value="0.2" />
  <param name="lightningFuzzyness" value="0.1" />
  <param name="branchMaxLevel" value="1" />
  <param name="branchProbability" value="2.0" />
  <param name="lightningVelocity" value="0.6" />
  <param name="strikeTimeMin" value="0.35" />
  <param name="strikeTimeMax" value="0.35" />
  <param name="strikeFadeOut" value="0.6" />
  <param name="strikeNumSegments" value="6" /> <!-- int max is 7 -->
  <param name="strikeNumPoints" value="5" />
  <param name="maxNumStrikes" value="6" />
  <param name="beamSize" value="0.2" />
  <param name="beamTexTiling" value="0.25" />
  <param name="beamTexShift" value="0.05" />
  <param name="beamTexFrames" value="4.0" />
  <param name="beamTexFPS" value="15.0" />
</Arc>
```

```
<Arc name="ExtendedArc">
  <param name="lightningDeviation" value="0.1" />
  <param name="lightningFuzzyness" value="0.05" />
  <param name="branchMaxLevel" value="1" />
  <param name="branchProbability" value="10.0" />
  <param name="lightningVelocity" value="0.25" />
  <param name="strikeTimeMin" value="2.0" />
  <param name="strikeTimeMax" value="2.0" />
  <param name="strikeFadeOut" value="1.0" />
  <param name="strikeNumSegments" value="6" /> <!-- int max is 7 -->
  <param name="strikeNumPoints" value="6" />
  <param name="maxNumStrikes" value="5" />
  <param name="beamSize" value="0.18" />
  <param name="beamTexTiling" value="0.25" />
  <param name="beamTexShift" value="0.05" />
  <param name="beamTexFrames" value="4.0" />
  <param name="beamTexFPS" value="18.0" />
</Arc>

  <Arc name="KickSparks">
  <param name="lightningDeviation" value="0.2" />
  <param name="lightningFuzzyness" value="0.1" />
  <param name="branchMaxLevel" value="1" />
  <param name="branchProbability" value="3.0" />
  <param name="lightningVelocity" value="16.0" />
  <param name="strikeTimeMin" value="0.0" />
  <param name="strikeTimeMax" value="0.05" />
  <param name="strikeFadeOut" value="0.05" />
  <param name="strikeNumSegments" value="6" /> <!-- int max is 7 -->
  <param name="strikeNumPoints" value="5" />
  <param name="maxNumStrikes" value="6" />
  <param name="beamSize" value="0.1" />
  <param name="beamTexTiling" value="0.75" />
  <param name="beamTexShift" value="0.15" />
  <param name="beamTexFrames" value="4.0" />
  <param name="beamTexFPS" value="15.0" />
</Arc>
```

To create a new lightning arc preset

1. Open the `lightningarceffects.xml` file.
2. Copy the text (between and including `<Arc name="Name">` through `</Arc>`) for an existing preset.
3. Paste it at the end of the file before the `</LightningArc>` closing bracket.
4. Replace the **Arc name** with your own custom preset name, then modify the following parameters to fit your needs.

The following table lists definitions for the parameters in the `lightningarceffects.xml` file.

Lightning Arc Entity Properties

Parameter	Description
<code>lightningDeviation</code>	The smoothness of the effect in meters.
<code>lightningFuzzyness</code>	The noisiness of the effect in meters.

Parameter	Description
branchMaxLevel	Allows child branches to strike out of the main beam and child branches to strike out from other child beams if this value is 2 or higher. A setting of 0 or 1 is recommended.
branchProbability	Probability that child branch will strike out from another beam segment. Consider these examples: <ul style="list-style-type: none"> • 0 – No branch is generated • 0.5 – Creates one branch per beam half the time • 1.0 – Creates one branch per beam • 2.0 – Creates 2 branches per beam
lightningVelocity	Rate at which a branch shifts upward from its original position after being triggered.
strikeTimeMin	Minimum time a branch remains visible.
strikeTimeMax	Maximum time a branch remains visible.
strikeFadeOut	Time to fade out after a branch disappears. This setting decreases the branch beamSize to 0 instead of actually fading with transparency.
strikeNumSegments	Number of snaking segments generated.
strikeNumPoints	Number of points per segment generated to create the noisy effect. The number of actual segments generated is defined by <code>strikeNumSegments * strikeNumPoints</code> . When the code generates the geometry, it creates a camera-aligned beam with exactly two triangles. This means the number of triangles per strike is <code>strikeNumSegments*strikeNumPoint*2</code> . Since <code>maxNumStrikes</code> is the hard limit of potential number of sparks active at any time, the potential number polygons of a given lightning effect is <code>strikeNumSegments*strikeNumPoint*2*maxNumStrike</code> . Note that with the LightningArc entity, each lightning strike triggers a new lightning strike. Therefore the total poly count of a given effect can be much higher. The game has internal limits for the total amount of lightning effects, lightning strikes, and polygons that cannot be surpassed.
maxNumStrikes	Hard limit on the number of beam segments that can be generated.
beamSize	Width of the beam generated. Child beams have half the width.
beamTexTiling	Texture tiling depends on the world size. A value of 2.0 means the texture wraps around twice every meter. A value of 0.25 means the texture will wrap around every 4 meters.
beamTexShift	Rate at which the U coordinate moves in a given direction. While <code>beamTexTiling</code> affects only the U coordinate, the V coordinate is automatically calculated to select one of the texture's frames.
beamTexFrames	Number of frames in the animation.
beamTexFPS	Frames per second of the multiframe animation.

Metastream Gem

Twitch Metastream is a feature that allows broadcasters to customize game streams with overlays of statistics and events from a game session. Using any web authoring tool, broadcasters can create custom HTML5 pages to control the information, graphics, layout, and behavior of the overlays.

Examples of information displayed in an overlay include:

- Character art
- Character strengths and weaknesses
- Player standings
- Stats for two leaders in a match
- Gold collected
- Kills, deaths, and assists
- Damage dealt

Broadcasters can switch between different graphic overlays that are timed to game events. They can also use a picture-in-picture style to display complementary information such as a minimap and live team stats.

To enable broadcasters to use Twitch Metastream, you must do the following:

1. Add the Metastream Gem to your project.
2. Set the `metastream_enabled` console variable to enable the feature.
3. Add a single line of code for each event you want broadcasters to access.

Note

Twitch Metastream is supported on Windows only.

Adding the Metastream Gem

Add the Metastream Gem to your project to turn on the local HTTP Metastream server that is included with Lumberyard.

To add the Metastream Gem

1. Open the **Project Configurator** (located in the `\dev\Bin64` directory at the root of your Lumberyard installation).
2. In the **Project Configurator**, select your project and click **Set as default**.
3. Under your project name, click **Enable Gems**.
4. On the **Gems (extensions)** page, select the **Metastream Gem**.
5. Click **Save**.
6. Rebuild your project.

For information, see [Gems \(p. 778\)](#).

Setting the Metastream Console Variable

After you add the Metastream Gem to your project, you can enable the Twitch Metastream feature by setting the `metastream` console variable. This allows you to create in-game options to enable or disable the feature.

To set the Metastream console variable

1. Edit your project's `game.cfg` file (located in the `\dev\project name\` directory) using a text editor.
2. Set `metastream_enabled` to 1.
3. Save the `game.cfg` file.
4. Close and reopen your project for the change to take effect.

Setting Options for the HTTP Server

After you enable Metastream, an HTTP server is embedded into the game client and serves as the access point for exposed data. Use a text editor to edit your project's `game.cfg` file (located in the `\dev\project name\` directory) to set the following options for the HTTP server.

metastream_enabled

Enables or disables the Metastream feature if the gem is enabled for the project.

Input type: Integer

Default: 1

metastream_serverPort

Sets the TCP port for the embedded HTTP server.

Input type: Integer

Default: 8082

metastream_docroot

Sets the document root for the embedded HTTP server.

Input type: String

Default: "Gems/Metastream/Files"

Exposing Data through Metastream

Metastream exposes data through the C++ API or the `Metastream::CacheData` flow graph node.

Exposing Data through the C++ API

The Metastream Gem uses a simple API to expose in-game data using the EBus system:

```
void MetastreamRequests::AddToCache(const char* table, const char* key, const char* value)
```

The following example shows how to use the Metastream C++ API in a project:

```
EBUS_EVENT(Metastream::MetastreamRequestBus, AddToCache, table, key, value);
```

Note

Any value that is added to the cache should be JSON compliant. For information, see the [JSON RFC](#).

Exposing Data through Flow Graph

The `Metastream::CacheData` node exposes in-game data using the flow graph.

Node Inputs

Activate

Writes or updates the key-value pair in the specified table and exposes it through Metastream.

Type: Any

Table

Writes the key-value pair to the specified table.

Type: String

Key

Identifies the value.

Type: String

Value

Writes the value to Metastream and automatically converts the value to meet JSON compliance.

Type: Any

Node Outputs

Out

Signals when the data was successfully written to Metastream.

Type: Any

Error

Signals with true if an error occurred.

Type: Bool

Accessing Data through the HTTP API

You can access game data that has been exposed through Metastream by using the HTTP API Get requests. You can then use JavaScript to work with the data.

http://localhost:*port*/pathToFile

Serves a file from the document root. File types include HTML, JS, CSS, images, sounds, resources, or assets.

Note

The data path is reserved for Metastream data. Files that are saved to the *document_root*/data/ directory will not be accessible.

http://localhost:*port*/data

Returns a list of available Metastream tables that contain key-value pairs.

http://localhost:*port*/data?table=*table_name*

Returns a list of all Metastream keys in the specified table.

Note

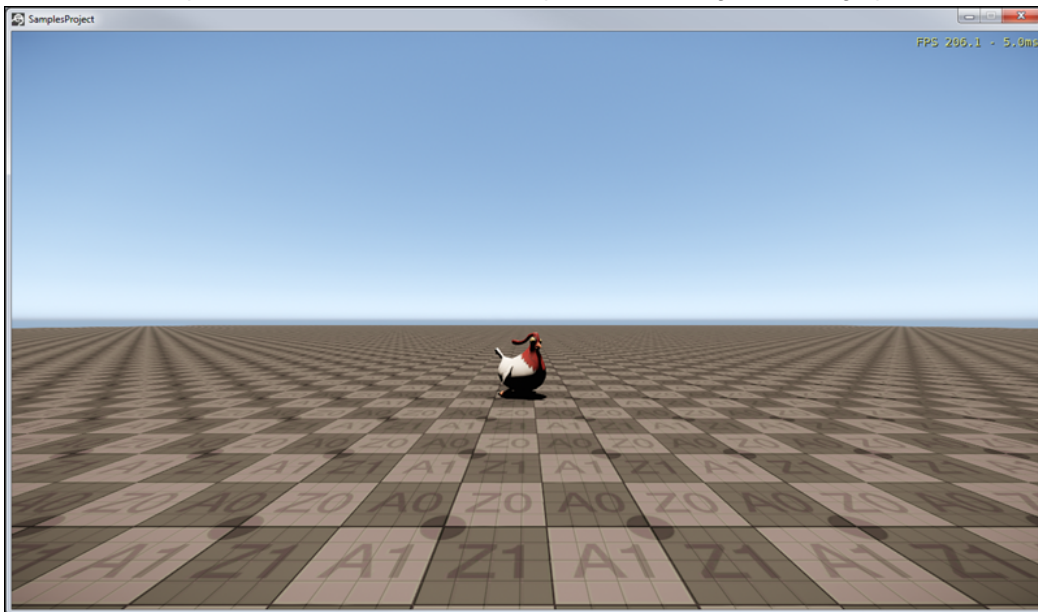
You can retrieve multiple key-value pairs in a single request by listing the keys in a comma-separated list. For example, `http://localhost:8082/data?table=sample&key=key1,key2,key3`

Data requests are returned in the following format:

Request	Return
<code>/data</code>	<pre>{ "tables": ["table1", "table2", ...] }</pre>
<code>/data?table=<i>table_name</i></code>	<pre>{ "keys": ["key1", "key2", ...] }</pre>
<code>/data?table=<i>table_name</i>&key=<i>key_name</i></code>	<pre>{ "key_name": value }</pre>
<code>/data?table=<i>table_name</i>&key=<i>keys_list</i></code>	<pre>{ "key1": value1, "key2": value2, ... }</pre>

Using the Metastream Sample

Located in the `\dev\SamplesProject\Levels\Samples\Metastream_Sample` directory, the Metastream sample level demonstrates how to expose data through the flow graph.



In conjunction with the sample level, the Metastream sample HTML file (located in the `\dev\Gems\Metastream\Files` directory) shows how to use the data to create a simple, dynamic overlay. These overlays can provide a more engaging experience for viewers without creating visual clutter on a broadcaster's game screen.

The following example from Amazon Game Studios' game Breakaway shows an overlay with stats from the two leaders in a match.



Multiplayer Gem

The Multiplayer Gem provides flow graph nodes to support multiplayer games using GridMate.

Topics

- [Multiplayer:IsClient node \(p. 818\)](#)
- [Multiplayer:IsServer node \(p. 818\)](#)
- [Multiplayer:Connect node \(p. 819\)](#)
- [Multiplayer:Disconnect node \(p. 819\)](#)
- [Multiplayer:Host node \(p. 819\)](#)
- [Multiplayer:ListServers node \(p. 820\)](#)
- [Multiplayer:ListServersResult node \(p. 820\)](#)
- [Multiplayer:SetOwner node \(p. 820\)](#)
- [Multiplayer:OnConnected node \(p. 820\)](#)
- [Multiplayer:OnDisconnected node \(p. 821\)](#)
- [Multiplayer:OnPlayerConnected node \(p. 821\)](#)
- [Multiplayer:OnPlayerDisconnected node \(p. 821\)](#)
- [Multiplayer:OnLocalPlayerReady node \(p. 821\)](#)
- [Multiplayer:OnPlayerReady node \(p. 821\)](#)

Multiplayer:IsClient node

Checks whether the current session is a client.

Node Inputs

Activate

Node Outputs

True

False

Multiplayer:IsServer node

Checks whether the current session is hosting.

Node Inputs

Activate

Node Outputs

True

False

Multiplayer:Connect node

Connect to a server.

Node Inputs

Activate

ServerAddress

Result

Node Outputs

Success

Failed

Multiplayer:Disconnect node

Disconnect from a server.

Node Inputs

Activate

Node Outputs

Success

Failed

Multiplayer:Host node

Host a server.

Node Inputs

Activate

ServerName

Map

MaxPlayers

Node Outputs

Success

Failed

Multiplayer:ListServers node

List the available servers.

Node Inputs

Activate

MaxResults

Node Outputs

Success

Failed

NumResults

Results

Multiplayer:ListServersResult node

Convert the ListServers list into fields.

Node Inputs

Results

Node Outputs

SessionId

ServerName

MapName

MaxPlayers

NumPlayers

Multiplayer:SetOwner node

Set the owner (network authority) for an entity.

Node Inputs

Activate

EntityId

MemberId

Node Outputs

Success

Failed

Multiplayer:OnConnected node

Indicate whether the multiplayer session is connected.

Node Outputs

True

False

Multiplayer:OnDisconnected node

Indicate whether the multiplayer session is disconnected.

Node Outputs

True

False

Multiplayer:OnPlayerConnected node

Activate when a new player connects.

Node Outputs

Name

MemberId

Multiplayer:OnPlayerDisconnected node

Activate when a player disconnects.

Node Outputs

MemberId

Multiplayer:OnLocalPlayerReady node

Activate when the local player has a valid actor entity.

Node Outputs

EntityId

MemberId

Multiplayer:OnPlayerReady node

Activate when a player has a valid actor entity.

Node Outputs

EntityId

MemberId

Physics Entities Gem

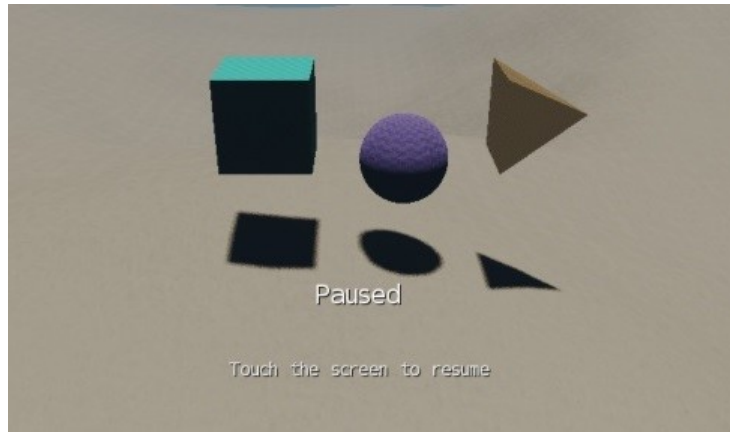
The PhysicsEntities Gem is a collection of physics entities used to simulate physical events such as explosions, gravity fields, or wind, or to physicalize objects such as cloth, breakable entities, or ropes. Physical entities that are related to a body instead of an event are connected to an object.

To access the Physics entities

1. On the **Objects** tab in the Rollup Bar, choose **Entity**.
2. Expand **Physics**.
3. Drag the entity into your level in the viewport.

Process Life Management Gem

The **ProcessLifeManagement** Gem demonstrates how you can respond to various application lifecycle events dispatched by the Lumberyard engine, in order to pause your game, display a modal splash screen, or anything else you may need to do when your application loses/regains focus.



Topics

- [Process Life Management Gem C++ \(p. 822\)](#)

Process Life Management Gem C++

You can access all system-specific events from C++ (even without enabling the Process Life Management Gem) by connecting to the appropriate EBus. Lumberyard also generates platform-agnostic events so that you can handle these events for all supported platforms.

Lumberyard Application Lifecycle Events

Lumberyard Application Lifecycle Events	iOS	Android
OnApplicationConstrained	applicationWillResignActive	onPause()
OnApplicationUnconstrained	applicationDidBecomeActive	onResume()
OnApplicationSuspended	applicationDidEnterBackground	onPause()
OnApplicationResumed	applicationWillEnterForeground	onResume()

Lumberyard Application Lifecycle Events	iOS	Android
OnMobileApplicationWillTerminate	applicationWillTerminate	onDestroy()
OnMobileApplicationLowMemoryWarning	ApplicationDidReceiveMemoryWarning	onLowMemory()

As demonstrated in `ProcessLifeManagementGem.h\ProcessLifeManagementGem.cpp`, use the following basic steps to receive process lifecycle events in your game.

To receive process lifecycle events in your game

1. Derive your class from `AzFramework::ApplicationLifecycleEvents::Bus::Handler` (or `AzFramework::[Ios|Android|Windows]LifecycleEvents::Bus::Handler` for platform specific events).
2. Override the functions corresponding to the events that you want to override:

```
void OnApplicationConstrained(Event /lastEvent/) override;  
    void OnApplicationUnconstrained(Event /lastEvent/) override;  
  
    void OnApplicationSuspended(Event /lastEvent/) override;  
    void OnApplicationResumed(Event /lastEvent/) override
```

3. Connect to the event bus when you want to start listening for events. In addition, be sure to disconnect when you no longer want to receive them. Use the following syntax:

```
ApplicationLifecycleEvents::Bus::Handler::BusConnect();  
...  
ApplicationLifecycleEvents::Bus::Handler::BusDisconnect();
```

Rain Gem

The Rain Gem creates realistic rain effects in your levels, including rain drops, puddles, mist, wet surfaces, and splashes. To enable the Rain Gem in your project, see [Gems \(p. 778\)](#).

This gem is a game object extension. On initialization, it preloads all textures listed in the `raintextures.xml` file.



Note

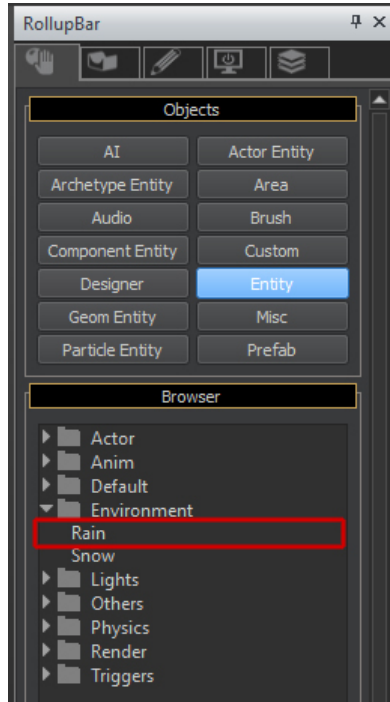
Place only a single Rain entity in your scene.

Placing Rain

You can place rain and customize it for your level by modifying properties for amount of puddles, strength and frequency of puddle ripples, quantity of rain, size and speed of the rain drops, and more.

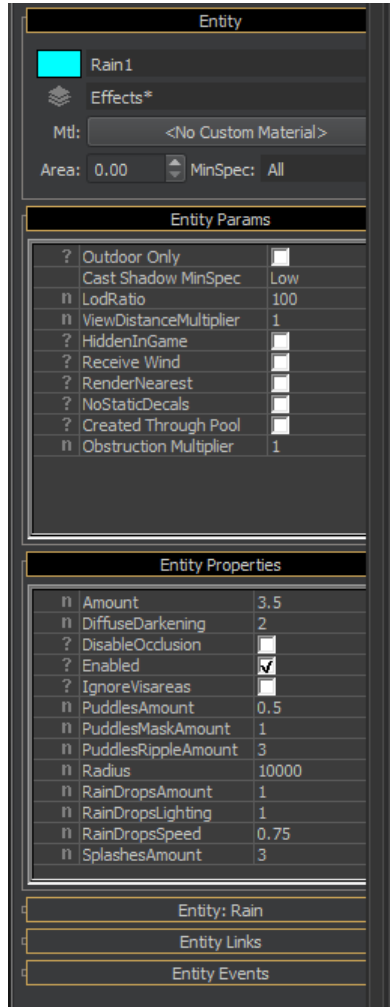
To add rain to your level

1. In the **Rollup Bar**, click **Entity**.
2. Under **Browser**, expand **Environment**.
3. Drag the **Rain** entity into your scene.



Configuring Rain

You can configure the rain's properties under [Entity Params](#) (p. 432) and **Entity Properties**.



Rain Entity Properties

Properties	Description
Amount	Sets overall amount of the rain entity's various effects
DiffuseDarkening	Sets the degree to which the rain darkens the surface diffuse
DisableOcclusion	Turns off checking whether an object is under cover and should be occluded from rain
Enabled	Toggles the rain effect
IgnoreVisareas	Continue to render rain when player is inside a visarea
PuddlesAmount	Sets the size and number of puddles that the rain creates
PuddlesMaskAmount	Sets the strength of the puddle mask to balance different puddle results
PuddlesRippleAmount	Sets the height and frequency of ripples in rain puddles
Radius	Sets the area on which rain falls
RainDropsAmount	Sets the number of rain drops

Properties	Description
RainDropsLighting	Sets the brightness of the rain drops
RainDropsSpeed	Sets the rate at which rain falls
SplashesAmount	Sets the degree of splashing on a surface

Using Console Variables for Rain

You can use the following console variables for the rain entity.

Rain Entity Console Variables

Variable	Description
r_Rain	Enables rain rendering
r_RainAmount	Sets rain amount
r_RainDistMultiplier	Multiplier for the rain layer's distance from the camera
r_RainDropsEffect	Enables rain drops effect
r_RainIgnoreNearest	Disables the layer showing the reflection of objects in rainy or wet areas nearest objects
r_RainMaxViewDist	Sets the maximum distance at which rain is visible
r_RainMaxViewDistDeferred	Sets the maximum distance (in meters) at which the deferred rain reflection layer is visible
r_RainOccluderSizeBlock	Blocks rain for objects bigger than this value

Using the Rain Sample

The Rain Sample uses the Rain, Clouds, and LightningArc gems to demonstrate how to use rain as an environment special effects (FX) in a level. The Lightning entity (from the LightningArc gem) shows how the lightning FX can enhance a rain storm with flashes of light and random strikes of lightning on the ground. The clouds are enabled to show how they can fill a scene.

Snow Gem

The Snow Gem creates realistic snow effects in your levels, including snowflake and surface effects, such as snow buildup. To enable the Snow Gem in your project, see [Gems \(p. 778\)](#).

Note

Place only a single **Snow** entity in your scene.

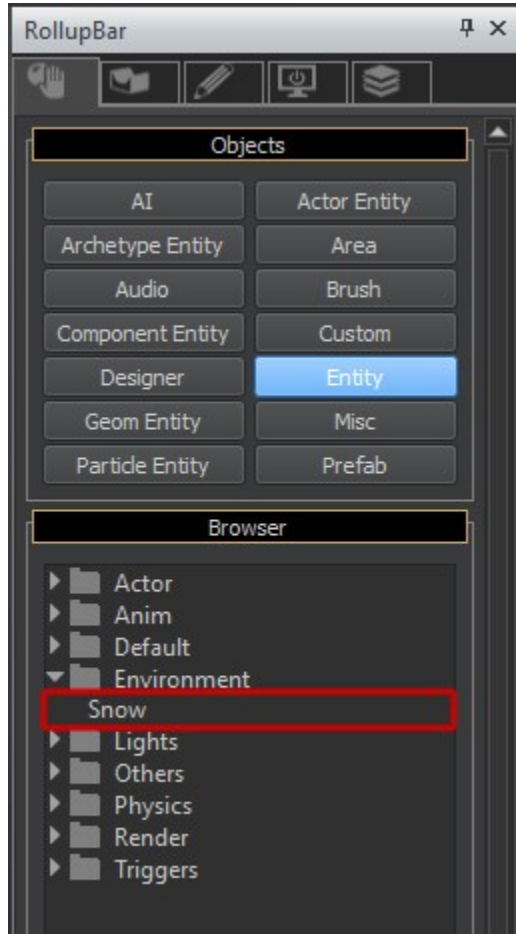


Placing Snow

You can place your snow and customize it to your level by modifying properties for brightness, gravity, size and quantity of snow flakes, how much snow and frost builds on a surface, and more.

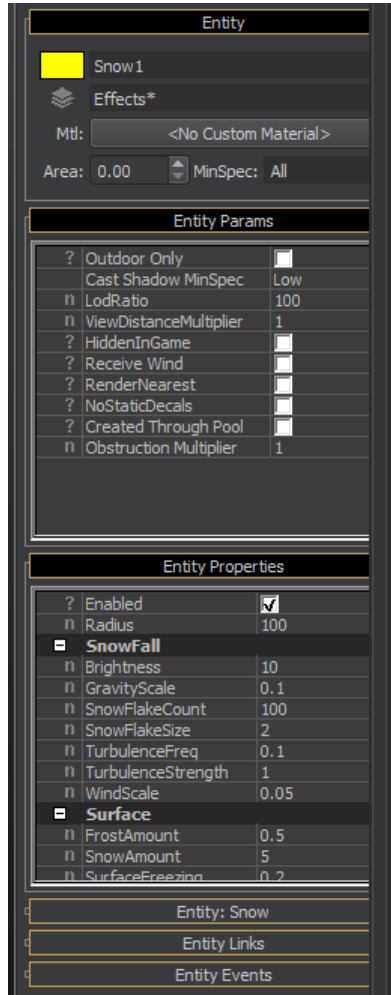
To add snow to your level

1. In the **Rollup Bar**, click **Entity**.
2. Under **Browser**, expand **Environment**.
3. Drag the **Snow** entity into your scene.



Configuring Snow

You can configure the snow's properties under [Entity Params](#) (p. 432) and **Entity Properties**.



Snow Entity Properties

Properties	Description
Enabled	Enables snow effect
Radius	Sets the area on which snow falls
SnowFall	
Brightness	Sets the brightness of the snow effect
GravityScale	Sets the gravity strength, which determines the rate at which snow falls
SnowFlakeCount	Sets the quantity of snowflakes
SnowFlakeSize	Sets size of individual snowflakes
TurbulenceFreq	Sets the frequency of the turbulence affecting the snow
TurbulenceStrength	Sets the strength of the turbulence affecting the snow
WindScale	Determines the impact of wind on the falling snow
Surface	

Properties	Description
FrostAmount	Sets the amount of frost on a surface
SnowAmount	Sets the amount of snow on a surface
SurfaceFreezing	Sets the degree to which surfaces appear frozen

Using Console Variables for Snow

You can use the following [console variables](#) (p. 56) for the snow entity.

Snow Entity Console Variables

Variable	Description
r_Snow	Enables snow rendering
r_SnowDisplacement	Enables displacement for snow accumulation
r_SnowFlakeClusters	Number of snow flake clusters
r_SnowHalfRes	When enabled, renders snow at half resolution to conserve fill rate

Using the Snow Sample

The Snow Sample uses the Snow and Clouds gems to demonstrate how to use the Snow entity as an environment special effects in a level. The Snow entity shows how snow falls and provides properties that you can set to randomly change the snow fall over time, creating a more dynamic weather experience.

Substance Gem

The Substance Gem is used in conjunction with the Substance Editor to manage substances.

Tornadoes Gem

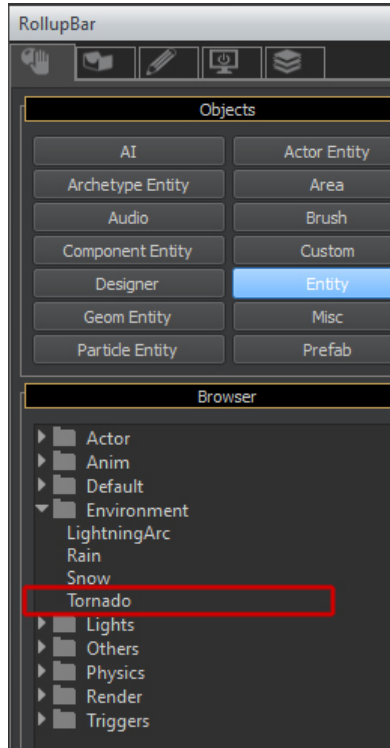
The Tornadoes Gem creates realistic tornado effects in your levels.

You can place your tornado and customize it to your level by modifying such properties as its height, funnel effect, radius, spin impulse, and so on. To enable the Tornadoes Gem in your project, see [Gems](#) (p. 778).



To add a tornado to your level

1. In the **Rollup Bar**, click **Entity**.
2. Under **Browser**, expand **Environment**.
3. Drag the **Tornado** entity into your scene.

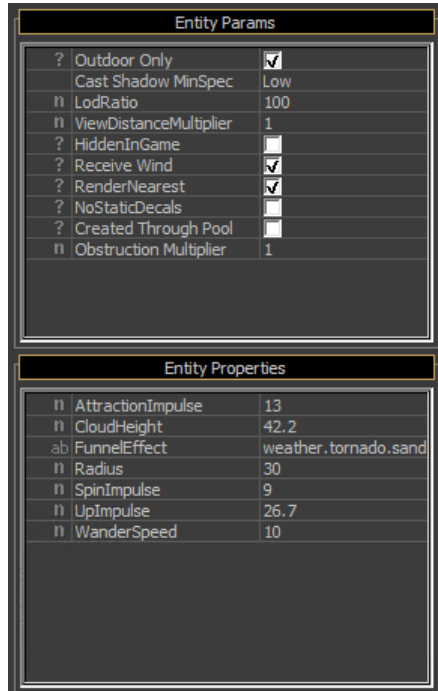


Configuring Tornadoes

You can configure the properties for the tornado entity to set properties for attraction impulse, spin speed affecting close objects, wander speed, and more. You can also set what type of material is inside the tornado.

To configure the tornado parameters and properties

1. Select the tornado entity you want to configure..
2. Under **Entity Params** (p. 432) and **Entity Properties**, select or clear check boxes for the preferred effects.



Tornado Entity Properties

Properties	Description
AttractionImpulse	Specifies how strongly the tornado attracts objects.
CloudHeight	Sets the height of the clouds above the tornado
FunnelEffect	Sets the specified particle effect defined in one of the tornado.xml files as explained in Customizing a Tornado Preset (p. 834) .
Radius	Sets the area that the tornado influences
SpinImpulse	Sets the spin speed that affects objects close to the tornado
UpImpulse	Sets the speed of upward pull that affects objects close to the tornado
WanderSpeed	Sets the speed at which the tornado moves

Customizing a Tornado Preset

You can customize your tornado entity using the presets in the `tornado.xml` file.

To use or customize a tornado preset

1. In a text editor, open `\dev\Gems\Tornadoes\Assets\libs\Particles\tornado.xml` in the Lumberyard root directory (`\lumberyard\dev`).
2. Choose one of the existing presets from the `tornado.xml` file (follows **Arc name** in the example) and, in Lumberyard Editor, under **Entity Properties**, type your chosen **Arc name** into the **ArcPreset** field .

For example, type **ExtendedArc** or **KickSparks**, which are existing names of presets as shown in the following `lightningarceffects.xml` file. This sample shows only the partial contents; open the file on your computer to view the full contents of the file.

Do one of the following:

- Copy one of the existing presets.
 - Create your own new preset in the file and copy it.
3. In the **Rollup Bar's Objects** tab, click **Entity**. Under **Entity Properties**, click **FunnelEffect** and paste the copied text.

The following is a sample of a preset contained in the `\dev\Gems\Tornadoes\Assets\libs\MaterialEffects\FXLibs\tornado.xml` file in the Lumberyard root directory (`\lumberyard\dev`).

```
<Particles Name="tornado.leaves">
  <Params Count="35" Continuous="true" ParticleLifeTime="4,Random=0.248"
FocusGravityDir="true" EmitAngle="Random=1" OrientToVelocity="true"
Texture="textures/sprites/smoke/smoke_b.tif" SoftParticle="true"
Alpha="0.267,ParticleAge=(;t=0.055,v=1;t=0.518;t=1)"
Color="(x=0.608,y=0.467,z=0.34)" DiffuseLighting="0.554"
DiffuseBacklighting="0.494"
Size="20,Random=0.812,ParticleAge=(v=1;t=0.51,v=0.25;t=1)"
Stretch="0.2" Speed="5" GravityScale="-2" TurbulenceSize="1.699147"
TurbulenceSpeed="335.1465" RandomAngles="y=359"/>
  <Childs>
    <Particles Name="base_dirt1">
      <Params Count="44" Continuous="true" ParticleLifeTime="0.35"
FocusGravityDir="true" EmitAngle="70" OrientToVelocity="true"
Texture="textures/sprites/dirt/dirt_c.tif" SoftParticle="true"
Alpha="0.3,ParticleAge=(;t=0.5,v=1;t=1)" Color="(x=0.733,y=0.725,z=0.616)"
DiffuseBacklighting="1" Size="120,Random=0.2307692" Speed="180"
Turbulence3DSpeed="50" TurbulenceSize="10" TurbulenceSpeed="-35.8"
Bounciness="-1" SortOffset="-0.02" VisibleUnderwater="If_False"
ConfigMin="Medium"/>
    </Particles>
    <Particles Name="base_smoke1">
      <Params Count="25" Continuous="true" ParticleLifeTime="0.9,Random=0.168"
RandomOffset="x=15,y=15" FocusGravityDir="true" EmitAngle="90"
OrientToVelocity="true" Texture="textures/sprites/smoke/smoke_tiled_c.tif"
TextureTiling="TilesX=2,TilesY=2,VariantCount=4" SoftParticle="true"
Alpha="ParticleAge=(;t=0.49,v=1;t=1)" Color="(x=0.73,y=0.62,z=0.52)"
DiffuseBacklighting="1" Size="40,Random=0.119,ParticleAge=(v=0.34;t=1,v=1)"
Speed="60,Random=0.238" RandomAngles="y=359" RandomRotationRate="y=180"
Bounciness="-1" SortOffset="-0.01" ConfigMin="Medium"/>
    </Particles>
    <Particles Name="debris">
      <Params Count="50" Continuous="true" ParticleLifeTime="30,Random=0.3"
RandomOffset="x=20,y=20" FocusGravityDir="true" EmitAngle="Random=1"
Facing="Free" Texture="textures/sprites/wood/wood_chip_tiled.tif"
TextureTiling="TilesX=2,TilesY=2,VariantCount=4" DiffuseBacklighting="1"
Size="5,Random=0.317,ParticleAge=(v=0.114;t=1,v=1,flags=4)"
Speed="10,Random=0.3,EmitterStrength=(v=0.5;t=1,v=1,flags=4)"
GravityScale="ParticleAge=(;t=1,v=1)"
TurbulenceSize="60,Random=0.2,ParticleAge=(v=0.09;t=0.663,v=0.23;t=1,v=1,flags=4)"
```

```
TurbulenceSpeed="-100,Random=0.5" RandomAngles="z=180"
RandomRotationRate="x=600,y=600,z=600" FillRateCost="0.2"/>
</Particles>
<Particles Name="leaves1">
  <Params Count="50" Continuous="true" ParticleLifeTime="30,Random=0.3"
FocusGravityDir="true" EmitAngle="Random=1" Facing="Free"
Texture="textures/sprites/leaves/leaf_tiled_a.tif"
TextureTiling="TilesX=2,TilesY=2,VariantCount=4" DiffuseBacklighting="1"
Size="2,Random=0.317,ParticleAge=(v=0.114;t=1,v=1,flags=4)"
Speed="10,Random=0.3,EmitterStrength=(v=0.5;t=1,v=1,flags=4)"
GravityScale="ParticleAge=(;t=1,v=1)"
TurbulenceSize="60,Random=0.2,ParticleAge=(v=0.09;t=0.663,v=0.23;t=1,v=1,flags=4)"
TurbulenceSpeed="-100,Random=0.5" RandomAngles="z=180"
RandomRotationRate="x=600,y=600,z=600" FillRateCost="0.2"/>
</Particles>
<Particles Name="base_smoke2">
  <Params Count="40" Continuous="true" ParticleLifeTime="6"
EmitAngle="3,Random=1" OrientToVelocity="true"
Texture="textures/sprites/smoke/smoke_tiled_d.tif"
TextureTiling="TilesX=2,TilesY=2,VariantCount=4" SoftParticle="true"
Alpha="0.5,Random=0.248,ParticleAge=(;t=0.055,v=1;t=0.996)"
Color="(x=0.353,y=0.318,z=0.28),ParticleAge=(v=(x=0.42,y=0.34,z=0.17);t=0.925,v=(x=0.867,y=0.867,z=0.17);t=0.925,v=(x=0.867,y=0.867,z=0.17))"
DiffuseBacklighting="0.3"
Size="130,Random=0.188,ParticleAge=(v=0.57,flags=32;t=0.255,v=0.5;t=0.996,v=1,flags=4)"
Stretch="0.2,ParticleAge=(t=0.514,v=0.5;t=1,v=1)" Speed="3"
GravityScale="-1.6" TurbulenceSize="1.7" TurbulenceSpeed="-120"
InitAngles="y=90" RandomAngles="y=180" RotationRate="y=100"
RandomRotationRate="y=20" SortOffset="-0.31" FillRateCost="0.7"/>
</Particles>
</Childs>
</Particles>
```

UiBasics Gem

The UiBasics Gem is a collection of assets to be used as defaults with the Lumberyard UI Editor, including basic UI prefabs (image, text, button, and text input) and the textures that those prefabs require. For more information, see [UI System \(p. 1137\)](#).

UiDemo Gem

The UiDemo Gem is a collection of assets that you can use to complete the **UI Creation** tutorial. For more information see, [Amazon Lumberyard Tutorials](#).

User Login Default Gem

The UserLoginDefault Gem provides a default user login implementation for all platforms, which is useful for testing and debugging.

Woodland Asset Collection Gem

The Woodland Asset Collection Gem is a collection of animations, materials, objects, and effects to create realistic and detailed forest levels. You can download the Woodland Assets separately from the [Amazon Lumberyard Downloads](#) page and install it as a gem using the Project Configurator. .



To access the Woodland Asset Collection Gem assets

1. On the **Objects** tab in the **Rollup Bar**, choose **Brush**.
2. Expand the appropriate folder as listed and drag the assets into your level in the viewport.
3. For certain woodland materials, use the following:
 - For clouds use the [Common.Cloud Shader \(p. 1003\)](#) and [DistanceClouds Shader \(p. 1004\)](#).
 - For skies use the [Sky Shader \(p. 1023\)](#) and [SkyHDR Shader \(p. 1023\)](#) shaders
 - For terrain use the [Terrain.Layer Shader \(p. 1025\)](#).
 - For water use the [Water Shader \(p. 1029\)](#), [Waterfall Shader \(p. 1030\)](#), and [WaterVolume Shader \(p. 1032\)](#).

Levels and Environment

A level, also known as world or map, represents the space or area available to the player during the course of completing a discrete game objective. Most games consist of multiple levels through which a player can advance to or move through, although usually only a single level is loaded at a time. Each level can be grouped into multiple layers, which you use to logically group types of objects.

The environment includes lighting, terrain, bodies of water, vegetation, sky, and weather effects.

Topics

- [Creating a New Level \(p. 838\)](#)
- [Creating Terrain \(p. 839\)](#)
- [Adding Sky Effects \(p. 860\)](#)
- [Adding Weather Effects \(p. 868\)](#)
- [Working with Layers \(p. 872\)](#)
- [Adding Vegetation \(p. 875\)](#)

Creating a New Level

The first step in creating a game world is to create a level.

To create a new level

1. In Lumberyard Editor, click **File, New**.
2. In the **New Level** window, type a file name and select a directory location for the file.
3. Select the desired **Heightmap Resolution** and **Meters per Texel** values. Click **OK**.
4. In the **Generate Terrain Texture** window, for **Texture Dimensions**, select texture dimensions to match your terrain heightmap dimensions.
5. For **Terrain Color Multiplier**, specify a value. Here are some guidelines:
 - If colors are distorted or have artifacts, increase the value of the **Terrain Color Multiplier** to compensate for the compression.
 - If only darker colors are used in the level, use this setting to make colors use more of the dynamic range.
 - For colors in the 0–63 range, enter a value of **4** for **Terrain Color Multiplier** to make them fill the entire 0–255 range. When rendering, the decompressed color values are divided by the multiplier in the shader to restore original brightness.

Note

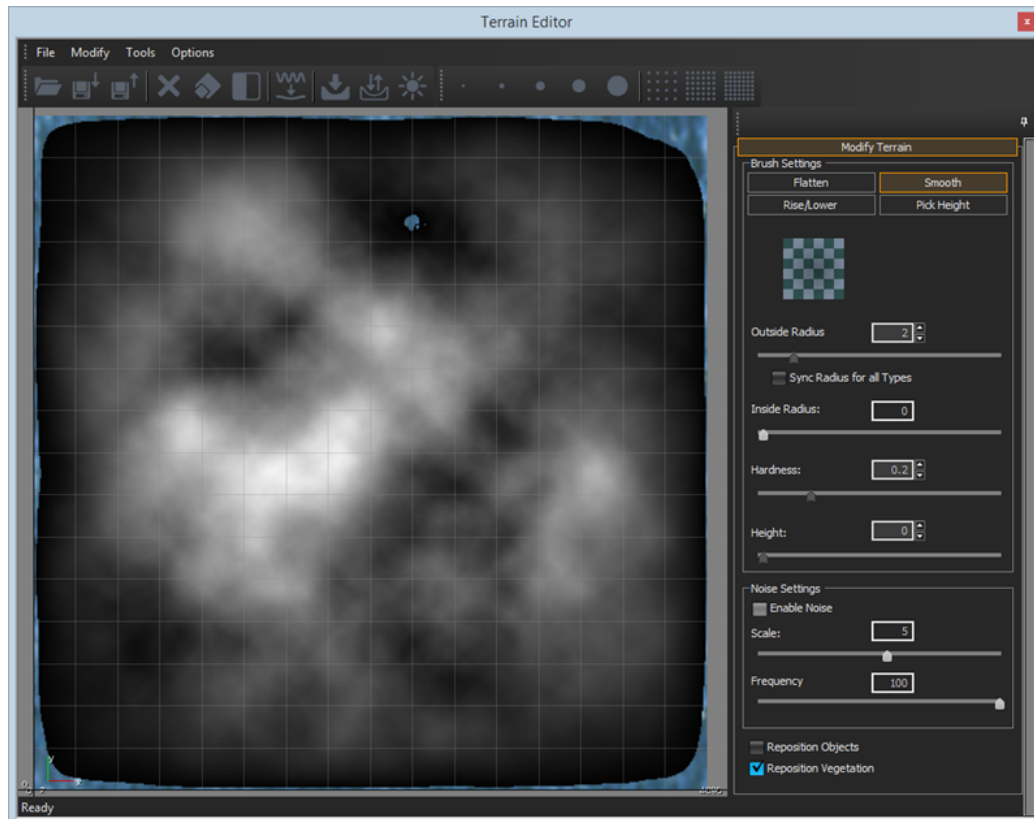
If the console variable `e_TerrainAo` is nonzero, it may darken the terrain and objects depending on nearby terrain and vegetation. Additionally, terrain normals (and hence lighting) will be more detailed at a distance.

6. For **Texture Generation Option**, selecting the **High Quality** setting takes two to three times longer but results in fewer compression artifacts and does not affect memory or CPU usage in game mode.

Creating Terrain

You can add realistic elements such as mountains, valleys, lakes, rivers, and roads to your terrain for your environment levels.

One of the primary tools used to first create a terrain is the Terrain Editor, as the following shows:



Topics

- [Using the Terrain Heightmap \(p. 840\)](#)
- [Using Terrain Texture Layers \(p. 843\)](#)
- [Creating Landforms and Topography \(p. 848\)](#)
- [Creating Bodies of water \(p. 851\)](#)
- [Copying and Moving Terrain Areas \(p. 857\)](#)
- [Importing and Exporting Terrain Blocks \(p. 858\)](#)
- [Importing Splat Maps \(p. 858\)](#)

Using the Terrain Heightmap

The heightmap is the base of the terrain in your level. You have three options for obtaining a terrain heightmap:

- Create a new heightmap using the Terrain Editor
- Create a new heightmap using a third-party terrain-building tool
- Importing an existing heightmap

Topics

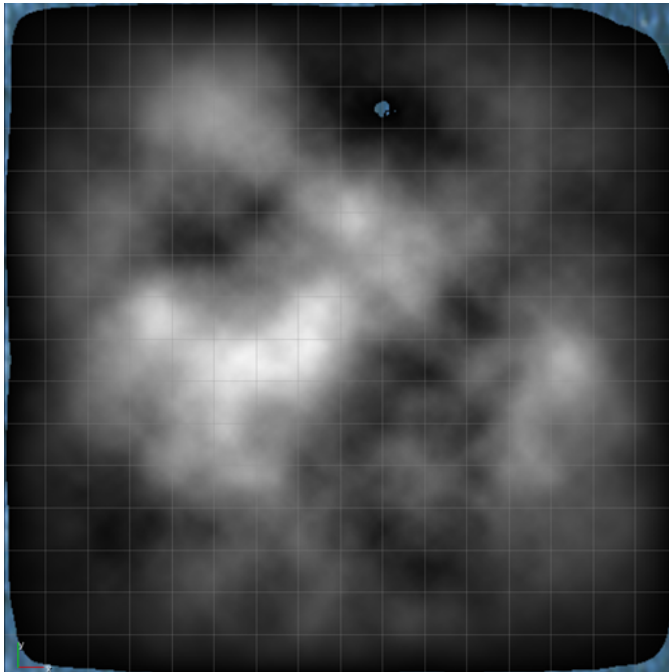
- [Creating a Terrain Heightmap \(p. 840\)](#)
- [Setting Heightmap Properties \(p. 841\)](#)
- [Importing a Terrain Heightmap \(p. 842\)](#)
- [Exporting a Terrain Heightmap \(p. 842\)](#)
- [Resizing a Terrain Heightmap \(p. 843\)](#)
- [Rotating a Terrain Heightmap \(p. 843\)](#)

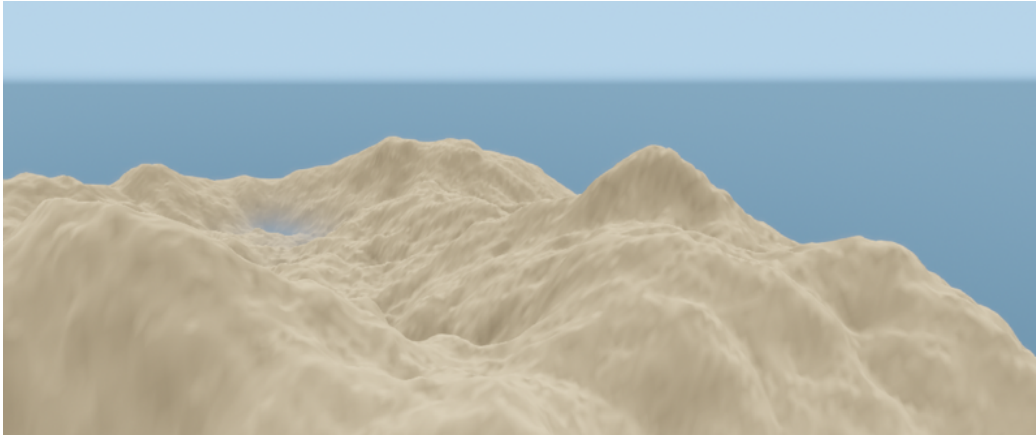
Creating a Terrain Heightmap

The first step in creating the heightmap using Lumberyard Editor is to specify the resolution and grid spacing, both of which define the terrain size. Terrain size is determined by multiplying heightmap resolution by meters per texel. This value should not exceed 4096 x 4096 kilometers.

Meters per texel is the distance in meters between two vertices on the grid. So a value of two means there is a grid point every two meters. You can use larger values to create a larger terrain, but it is less detailed for the same heightmap resolution.

The following images show a terrain heightmap and the corresponding generated terrain.





To create a new heightmap using Terrain Editor

1. In Lumberyard Editor, click **File, New**.
2. In **New Level**, enter a file name and directory location for the heightmap file.
3. Select the desired **Heightmap Resolution** and **Meters per Texel** values.
4. Click **Terrain, Edit Terrain**.
5. In Terrain Editor, click **Tools, Generate Terrain**.
6. In **Generation**, adjust the following parameter values as needed.

Feature Size

Determines the amount of land created.

Bumpiness/Noise (Fade)

Determines the degree of bumpiness or deformation of the surface.

Slope Detail (Passes)

Determines the number of times that effect is applied.

Seed (Random Base)

Determines the degree of random variation for the heightmap.

Slope Smoothing (Blur Passes)

Sets the number of times that smoothing is applied to the noise filter.

Sharpness (Exp. Base)

Determines the sharpness of the surface.

Sharpness (Freq. Step)

Determines the number of times that the sharpness filter is applied to the surface.

Setting Heightmap Properties

You can use the Terrain Editor to set various heightmap properties and parameters that affect the shape of the terrain profile.

To set heightmap properties

1. In Lumberyard Editor, click **Terrain, Edit Terrain**.
2. In **Terrain Editor**, click **Modify**, and then click and adjust the various following properties and parameters:

Make Isle

Sinks the heightmap so that it is surrounded by ocean.

Remove Ocean

Sets the ocean level to -100000 meters.

Set Ocean Height

Sets the ocean level in meters.

Set Terrain Max Height

Sets the maximum height for the tallest mountain. (Default is 1024 meters).

Set Unit Size

Sets the meters per texel size of the heightmap.

Flatten

Flattens terrain to either a higher or lower point.

Smooth

Removes all hard edges from the heightmap.

Smooth Slope

Removes hard edges from steep areas of the heightmap.

Smooth Beaches/Coast

Removes hard edges from flat areas of the heightmap.

Normalize

Ensures the entire greyscale spectrum is used between the **Max Height** value and zero.

Reduce Range (Light)

Makes heightmap mountains smaller.

Reduce Range (Heavy)

Makes heightmap mountains small.

Erase Terrain

Deletes all heightmap data.

Resize Terrain

Resizes the terrain heightmap.

Invert Terrain

Inverts all grayscale data, changing black to white and vice versa.

Importing a Terrain Heightmap

The following file formats are supported for importing a heightmap file:

- .tif
- .png
- .jpg
- .tga
- .bmp
- .pgm
- .raw
- .r16

To import a heightmap

1. In Lumberyard Editor, click **Terrain, Edit Terrain**.
2. In **Terrain Editor**, click **File, Import Heightmap**.

Exporting a Terrain Heightmap

You can export a heightmap file created using the Terrain Editor to the following file formats:

- .tif
- .png
- .jpg
- .tga
- .bmp
- .pgm
- .raw
- .r16

To export a heightmap

1. In Lumberyard Editor, click **Terrain, Edit Terrain**.
2. In **Terrain Editor**, click **File, Export Heightmap** and enter a file name and directory location.

Resizing a Terrain Heightmap

Resizing the terrain heightmap involves changing the resolution of your heightmap. Terrain size is determined by multiplying heightmap by meters per texel. When resizing, this value should not exceed 4096x4096 kilometers.

Meters per texel is the distance in meters between two vertices on the grid. So a value of two means there is a grid point every two meters. You can use larger values to create a larger terrain, but it is less detailed for the same heightmap resolution.

To resize a heightmap

1. In Lumberyard Editor, click **Terrain, Resize Terrain**.
2. For **Heightmap Resolution**, select the desired value.
3. For **Meters Per Texel**, select the desired value.

Rotating a Terrain Heightmap

Rotating a terrain heightmap involves just a few simple steps.

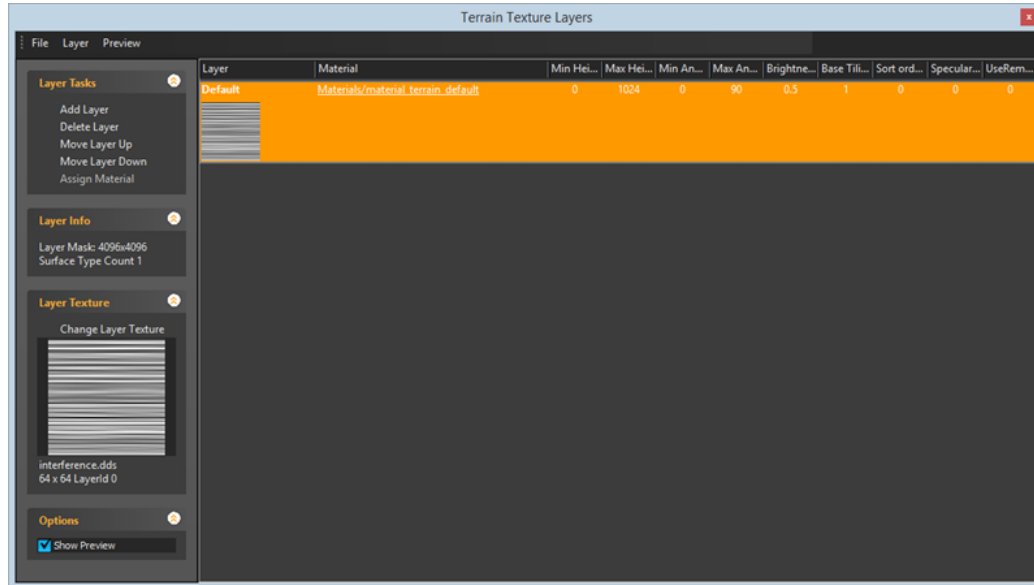
To rotate the heightmap

1. In **Rollup Bar**, on the **Terrain** tab, click **Move Area**.
2. Click **Select Source**.
3. At the bottom of Lumberyard Editor, type the **X** and **Y** coordinates for the heightmap center, then click **Lock Selection**.
4. Click **Select Target** and repeat Step 3.
5. In **Target Rotation**, select a value in degrees.

Using Terrain Texture Layers

You can create terrain texture layers and paint them to enhance your environment level.

The primary tool for creating and managing terrain texture layers in your level is the Terrain Texture Layers editor, as shown below:



Topics

- [Adding a Terrain Texture Layer \(p. 844\)](#)
- [Applying a Texture Layer Material \(p. 844\)](#)
- [Importing Terrain Texture Layers \(p. 845\)](#)
- [Exporting Terrain Texture Layers \(p. 845\)](#)
- [Painting Terrain Texture Layers \(p. 845\)](#)
- [Changing Terrain Tile Resolution \(p. 846\)](#)
- [Generating the Terrain Texture \(p. 847\)](#)

Adding a Terrain Texture Layer

Create a new texture layer and change the terrain texture and material as desired.

To add a terrain texture layer

1. In Lumberyard Editor, click **Terrain, Terrain Texture Layers**.
2. In **Terrain Texture Layers**, under **Layer Tasks**, click **Add Layer**.
3. Double-click the **NewLayer** text and assign a unique name to it.
4. Click **Materials/material_terrain_default** to open Material Editor.
5. In the tree, click **Materials\material_terrain_default** and adjust material and other settings as needed.

Applying a Texture Layer Material

All terrain texture layer materials use the [Terrain.Layer Shader \(p. 1025\)](#). All terrain materials should be "high-passed" in your DCC tool in order for them to work correctly with this shader.

To apply or edit a material for a texture layer

1. In **Terrain Texture Layers**, double-click the layer you want to apply or edit a material for.

2. In **Material Editor**, expand the tree and select your asset.
3. Change settings and shader parameters as needed.
4. In **Terrain Texture Layers**, click **Assign Material**.
5. Close Material Editor.

Importing Terrain Texture Layers

By importing a saved layer, all materials, textures, and shader settings can be quickly applied to your level.

To import a terrain texture layer

1. In Lumberyard Editor, click **Terrain, Terrain Texture Layers**.
2. Click **File, Import Layers**.
3. Select the layer (.lay) file for import, then click **Open**.

Exporting Terrain Texture Layers

By exporting your terrain texture layer, you can reuse it in multiple levels.

To export a terrain texture layer

1. In Lumberyard Editor, click **Terrain, Terrain Texture Layers**.
2. Click **File, Export Layers**.
3. Type a file name and select a directory path for the exported file, then click **Save**.

Painting Terrain Texture Layers

Lumberyard uses two components for painting terrain texture layers:

- The first is a low-resolution texture with color information. This texture is visible from a distance and provides underlying color information for the base terrain texture. This texture should be less than 512 x 512 pixels in size.
- The second is a high-resolution material. This material is visible at close distances and can have several texture maps like diffuse, bump, and specular. The diffuse map should be set to white (255).

The distance at which low-resolution textures are replaced with those of a higher resolution is defined by the **DetailLayersViewDistRatio** parameter. To access this parameter, open **Rollup Bar**, click **Terrain, Environment** and adjust the value as needed.

To paint a terrain texture layer

1. In **Rollup Bar**, on the **Terrain** tab, click **Terrain, Layer Painter**.
2. Adjust the following terrain brush settings as needed.

Radius

Specifies the size of the brush. Use the slider to adjust the size, or use the following shortcut keys: [to increase the brush radius size or] to decrease the brush radius size.

Hardness

Specifies the strength of the brush when applying the material. Provide a lower value for a softer translucent effect. Provide a value of 1 for a painted material that appears opaque.

Use the slider to adjust the size, or use the following shortcut keys: **Shift+[** to decrease the hardness shape of the fall-off curve between the inner and outer radius of the brush or **Shift+]** to increase the hardness shape.

Paint LayerID (DetailLayer)

When enabled, the painter only paints the detail texture layer of the terrain material.

Mask by Layer Altitude and Slope

Sets the material to only paint between the layer **Altitude** and **Slope** parameters defined below.

Mask by

Select a layer to prevent it from being painted over.

3. Adjust the following layer brush settings as needed.

Brightness

Modifies the brightness of the material base color. Click the **Color** box to open up the color selector and alter the base color of your material. Click **Save Layer** when done.

Altitude

Sets a minimum and maximum altitude mask for painting—the brush applies only within these boundaries.

Slope (degrees)

Sets a minimum and maximum slope mask for painting—the brush applies only within these boundaries.

Changing Terrain Tile Resolution

A terrain layer can be divided into multiple tiles, each of which can be painted with a resolution between 64x64 and 2048x2048 kilometers. The higher the resolution, the softer the transition between terrain texture layers.

If you know a player spends a lot of time in specific areas of the level and thus have more opportunity to view the terrain, you can save resources by increasing the resolution in just those areas. Follow this two-step process:

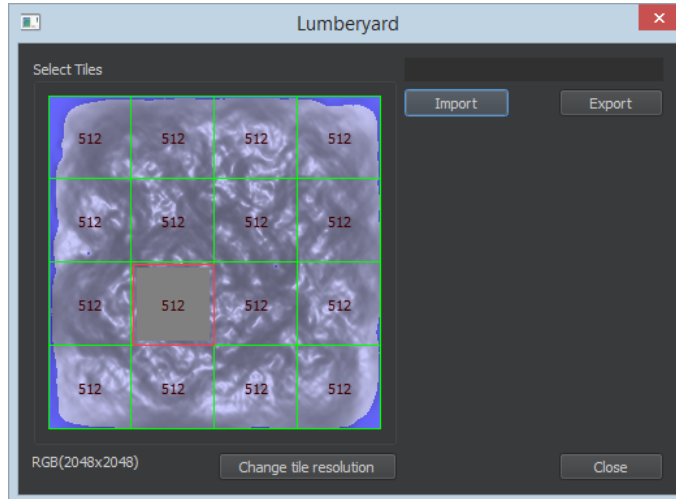
You first subdivide the texture layer, then change the individual tile resolution, as follows:

To subdivide the terrain texture layer

1. In Lumberyard Editor, click **Terrain, Terrain Texture Layers**.
2. Click **File, Refine Terrain Texture Tiles**. The layer is now split into 2x2 (4) tiles.
3. Repeat step 2. The layer is now divided into 4x4 (16) tiles.
4. Repeat only as needed as there is no way to go back and reduce the number of tiles.

To change terrain tile resolution

1. In Lumberyard Editor, click **Terrain, Export/Import Megaterrain Texture**.
2. In **Terrain Texture**, click a tile whose resolution you want to change. Then click **Change tile resolution**.



3. Select a new resolution, click **OK**. Then click **Close**.

Generating the Terrain Texture

When you are done creating and painting all terrain texture layers and assigning materials, the terrain texture is ready to be generated. When the generated terrain texture is compressed, which uses the DXT algorithm, colors can appear distorted. To improve color rendering, use the full dynamic range (RGB values 0–255).

To generate the terrain texture

1. In Lumberyard Editor, click **Terrain, Generate Terrain Texture**.
2. In **Generate Terrain Texture**, for **Texture Dimensions**, select texture dimensions to match your terrain heightmap dimensions.
3. Select a value for **Terrain Color Multiplier**, as follows:
 - If colors are distorted or have artifacts, increase the value of the **Terrain Color Multiplier** to compensate for the compression.
 - If only darker colors are used in the level, use this setting to allow colors to use more of the dynamic range.
 - For colors in the 0–63 range, enter a value of 4 for **Terrain Color Multiplier** to make them fill the entire 0–255 range. When rendering, the decompressed color values are divided by the multiplier in the shader to restore original brightness.

Note

If the console variable **e_TerrainAo** is nonzero, it may darken the terrain and objects depending on nearby terrain and vegetation. Additionally, terrain normals (and hence lighting) are more detailed at a distance.

4. Under **Texture Generation Option**, select the **High Quality** check box; This setting takes longer but results in fewer compression artifacts and does not affect memory or CPU usage in game mode.

Creating Landforms and Topography

You can add realistic mountains, hills, valleys, and other landforms to your terrain in your environment level. The primary method for creating interesting terrain features and landforms involves the following brushes:

- **Rise/Lower brush** – Increases and decreases the local terrain height to quickly create hills, valleys, and river beds, for example.
- **Flatten brush** – Flattens the terrain at a specified height and diameter. Use the **Pick Height** feature to select a height from which to begin flattening.
- **Smooth brush** – Smooths over sharp gradients in the terrain.
- **Holes brush** – Used to make holes in the terrain for creating areas beneath or inside the terrain such as caves.

You can also use the Terrain Editor to modify the terrain heightmap, although this method is not as accurate and does not give you the control you get from working directly in the viewport in Lumberyard Editor. For more information, see [Setting Heightmap Properties \(p. 841\)](#).

Topics

- [Using the Rise/Lower Brush \(p. 848\)](#)
- [Using the Smooth Brush \(p. 849\)](#)
- [Using the Flatten Brush \(p. 849\)](#)
- [Using the Holes Brush \(p. 849\)](#)
- [Terrain Brush Parameters \(p. 849\)](#)
- [Creating Roads \(p. 850\)](#)

Using the Rise/Lower Brush

The Rise/Lower brush is perhaps the most versatile of all the terrain brushes and is often the first used. With it you can create many macroterrain landforms and features such as mountains, hills, cliffs, valleys, and riverbeds, for example. After using this brush, see [Using the Flatten Brush \(p. 849\)](#) and [Using the Smooth Brush \(p. 849\)](#) to learn how to control the shape and overall visual look.

To use the Rise/Lower brush

1. In **Rollup Bar**, on the **Terrain** tab, click **Modify, Rise/Lower**.
2. Adjust the **Height** slider to the desired height:
 - Use positive values for landforms that rise above the base level.
 - Use negative values for valleys and other landforms that sink below the base level.
3. Adjust the **Outside Radius** and **Inner Radius** sliders (and the difference between the values) to control the steepness of the terrain.
4. In the level, drag the mouse around to achieve the desired effect.
5. Under **Modify Terrain**, adjust **Brush Settings** and **Noise Settings** parameters as needed. See [Terrain Brush Parameters \(p. 849\)](#) for more information.
6. When done, click **Terrain, Modify** or press **Esc**.

Using the Smooth Brush

The Smooth brush softens sharp gradients in the terrain, such as the sides of mountains, cliffs and lake beds for example. This brush averages out the height of the terrain based on nearby terrain areas to provide a smoother surface.

To use the smooth brush

1. In **Rollup Bar**, on the **Terrain** tab, click **Modify, Smooth**.
2. In the level, drag the mouse to create the smoothing effect.
3. Under **Modify Terrain**, adjust **Brush Settings** and **Noise Settings** parameters as needed. See [Terrain Brush Parameters \(p. 849\)](#) for more information.
4. When done, click **Terrain, Modify** or press **Esc**.

Using the Flatten Brush

The Flatten brush makes any piece of terrain completely flat at a height that you define. This is useful for creating a variety of features such as plateaus, mesas, and buttes as well as creating flat spots wherever needed.

To use the Flatten brush

1. In **Rollup Bar**, on the **Terrain** tab, click **Modify, Flatten**.
2. In the level, drag the mouse to create a flat spot. The terrain is flattened at the selected **Height** and **Diameter** brush settings.
3. Under **Modify Terrain**, adjust **Brush Settings** and **Noise Settings** parameters as needed. See [Terrain Brush Parameters \(p. 849\)](#) for more information.

Using the Holes Brush

The Holes brush makes a geometrical hole in both the terrain layer as well as the visual mesh. It is useful for creating craters, sinkholes, caves, and other areas beneath or inside the terrain. The resulting holes can be filled with various objects such as rocks or vegetation.

To use the Holes brush

1. In **Rollup Bar**, on the **Terrain** tab, click **Holes**.
2. Adjust the **Brush Radius** slider to adjust the size of the hole.
3. Click **Make Hole** to create a hole.
4. In the level, click to place the hole. By default you can see the ocean showing through.
5. To remove a hole, click **Remove Hole**. You are limited to removing one terrain unit adjacent to the existing terrain.

Terrain Brush Parameters

A number of settings apply to multiple terrain brushes. Use the following parameters to adjust the rise/lower, smooth, and flatten brushes.

Outside Radius

The outer edge of the area of the terrain brush effect.

Sync Radius for All Types

Select to set the same outer radius value across the flatten, smooth, and rise/lower brushes.

Inside Radius

The inner edge of the area of the terrain brush effect. Within this radius the effect of the brush is at its maximum.

Hardness

Controls the shape of the fall-off curve between the inner and outer radius of the brush.

Height

For the rise/lower and flatten brushes, the incremental amount the terrain is be raised/lowered or flattened with each click in the terrain level.

Enable Noise

Select to add random terrain variances to the brush.

Scale

Controls the strength of the noise effect.

Frequency

How often the noise effect is applied.

Reposition Objects

Select to realign objects with the modified terrain. Objects remain on top.

Reposition Vegetation

Select to realign vegetation with the modified terrain. Vegetation remains on top.

Creating Roads

You can add realistic roads to your terrain in your environment level.

For information on the road entity, see [Road Entity \(p. 481\)](#).

Topics

- [Creating the Road Entity \(p. 850\)](#)
- [Applying a Road Material \(p. 851\)](#)
- [Adjusting Road Spline Geometry \(p. 851\)](#)
- [Splitting and Merging Roads \(p. 851\)](#)

Creating the Road Entity

You can create and place roads using the Road entity as follows.

When performing this procedure, you may notice that parts of the road disappear into the terrain. The **Align Height Map** step resolves this by stretching the terrain height to match the path of the road based on its shape and on **BorderWidth** parameter. For information on **BorderWidth** and related settings, see [Road Entity \(p. 481\)](#).

To create and place the Road entity

1. In **Rollup Bar**, on the **Objects** tab, click **Misc, Road**.
2. In your level, start at the beginning of the road and click to place a series of points that define the road's path.
3. When complete, double-click where you want the road to end.
4. In **Rollup Bar**, under **Road Parameters**, click **Align Height Map** to adjust the terrain height to match the path of the road.

Applying a Road Material

After the Road entity has been placed, you can apply a material to the road.

To apply a material to a road

1. In **Rollup Bar**, on the **Objects** tab, click **Misc, Road**.
2. Click **<No Custom Material>** to open Material Editor.
3. In Material Editor, expand the tree and select your asset.
4. Modify material settings and shader parameters as needed.
5. When finished, click **Assign Item to Selected Objects**, and close Material Editor.

Adjusting Road Spline Geometry

You can make precise changes to the geometry of a road by adjusting the spline points and parameters.

To adjust road spline parameters

1. In **Rollup Bar**, on the **Objects** tab, click **Misc, Road**.
2. Under **Spline Parameters**, click **Edit**, and do any of the following for the road in your level:
 - To move a point, drag it.
 - To add a new point, hold down **Ctrl** while you click on the spline at the desired location.
 - To delete a point, double-click it.
 - To change the angle at a point, select it and adjust the **Angle** value.
 - To change the width at a point, select it, clear the **Default width** check box, and adjust the **Width** value.

Splitting and Merging Roads

You can split a road apart or merge two roads together.

To split or merge roads

1. In **Rollup Bar**, on the **Objects** tab, click **Misc, Road**.
2. Under **Spline Parameters**, click **Edit**, and do the following for the road in your level:
 - To split a road apart, select the desired point and click **Split**.
 - To merge two roads together, select the end point of one road and the start point of another road. Then click **Merge**.

Creating Bodies of water

You can create realistic-looking ocean, lakes, rivers, waterfalls, and pools with waves and ripples. Players and objects interacting with water surfaces also generate waves and ripples. Water gets its appearance from reflections on the surface and the interaction of light with particles suspended underneath the surface. You need both to achieve an authentic look.

Lumberyard offers three different shaders for rendering bodies of water:

- **Water Shader (p. 1029)** – For oceans only
- **WaterVolume Shader (p. 1032)** – For lakes, rivers, ponds and all other water volumes

- [Waterfall Shader \(p. 1030\)](#) – For waterfalls only

Lumberyard also supports caustics. Caustics are optical properties caused by light refracting through a volume of water, creating light and dark patterns at the bottom. Realistic caustic effects also include water ripples generated from players and other objects interacting with the water surface.

Note

To make caustics visible, you must place water volumes at a height of 1 or greater in your level.

Topics

- [Preparing the Terrain \(p. 852\)](#)
- [Setting Ocean Parameters \(p. 852\)](#)
- [Creating Rivers \(p. 853\)](#)
- [Adding Waterfalls \(p. 855\)](#)
- [Adding Water Puddles \(p. 855\)](#)
- [Adding Fog Above Water \(p. 856\)](#)
- [Advanced Water Volume Parameters \(p. 856\)](#)

Preparing the Terrain

For all water volumes such as lakes, ponds, and reservoirs, the terrain must first be lowered and sculpted to contain the body of water. To create the bottom and walls of your body of water, you need to consider the depth, shape, and edges of your landform geography.

For rivers, see [Preparing the River Terrain \(p. 853\)](#).

To prepare the terrain for bodies of water

1. In **Rollup Bar**, on the **Terrain** tab, click **Modify, Rise/Lower**.
2. Adjust the **Outside Radius** value as needed for the widest point of the water volume.
3. Adjust the **Height** value to a negative value for the depth of the water volume.
4. Adjust the other terrain brush settings as needed to fine-tune the look of the walls. See [Terrain Brush Parameters \(p. 849\)](#) for more information.
5. In your level, drag to define the shape. Release the mouse button and repeat as needed.

Setting Ocean Parameters

When you create a new level, Lumberyard creates an ocean by default, complete with waves and reflections. The ocean uses the [Water Shader \(p. 1029\)](#). You can change the ocean's various properties and effects.

To set ocean parameters

1. In **Rollup Bar**, on the **Terrain** tab, click **Environment**.
2. Under **Ocean**, adjust the following parameter values:
 - **Material** – Click the ... button to access Material Editor and select your asset.
 - **Caustic depth** – Set the depth to which caustic effects are visible.
 - **Caustic intensity** – Scale the intensity of the caustics for the water surface normals.
 - **Caustic tiling** – Scale the caustic tiling applied to the water surface normals. You can scale caustics independently of the surface material in cases of strong tiled normals or vice-versa.

Creating Rivers

You can add realistic rivers, complete with waterfalls, to your terrain in your environment level.

The following are best practices and guidelines to keep in mind when creating rivers.

- Rivers are 2D objects, which means rivers cannot be made to flow down steep inclines. However, to make a river flow down gentle inclines, you can rotate the river along the Z axis slightly (Z=0.5 to 1.0).
- To create rivers that appear to flow down steep inclines, create multiple rivers and connect them with waterfalls.
- The more points you place for the river geometry, the more control you have for direction and curvature.
- The wider the river, the further apart the points should be to avoid clipping at sharp corners.
- For more realism, paint the bottom of the river a different texture and add vegetation.
- For more realism, add particle effects.

For information on the river entity see [River Entity \(p. 480\)](#).

Topics

- [Preparing the River Terrain \(p. 853\)](#)
- [Creating the River Entity \(p. 853\)](#)
- [Applying a River Material \(p. 854\)](#)
- [Adjusting River Spline Geometry \(p. 854\)](#)
- [Splitting and Merging Rivers \(p. 855\)](#)

Preparing the River Terrain

Rivers need a riverbed and walls, which you implement as a deformation in the terrain. Use the rise/lower terrain brush for this effect.

To create a realistic-looking riverbed and walls, make sure that the walls of the river are above the starting (first) point of the river for the entire length of the river.

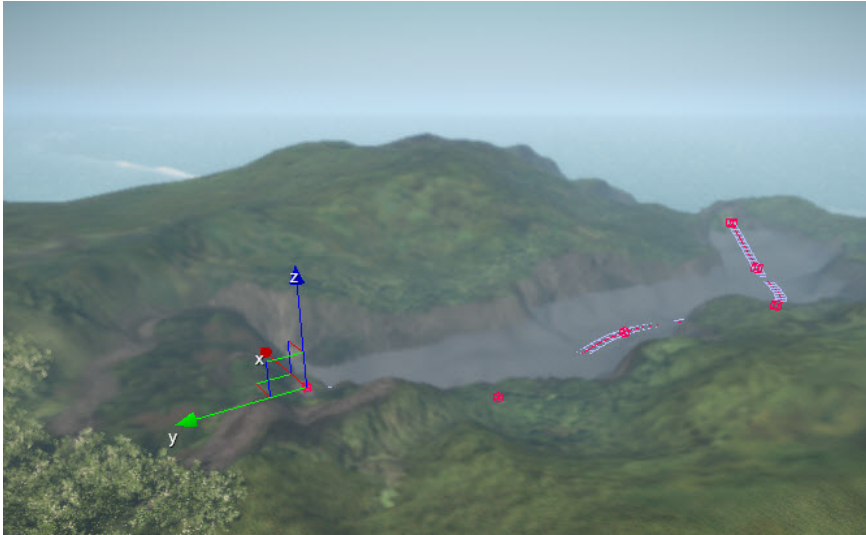
To create the riverbed and walls

1. In **Rollup Bar**, on the **Terrain** tab, click **Modify, Rise/Lower**.
2. Adjust the **Outside Radius** slider as needed for the width of the riverbed.
3. Adjust the **Height slider** to a negative value for the depth of the riverbed.
4. Adjust the other terrain brush settings as needed to fine-tune the look of the riverbed. See [Terrain Brush Parameters \(p. 849\)](#) for more information.
5. In your level, position the mouse at the start of river, and then drag to define the direction and course of the river. Release the mouse at the end of the river.

Creating the River Entity

After you have prepared the riverbed, you next create and place the River entity.

When performing this procedure, you may notice that parts of the river disappear into the terrain. The **Align Height Map** step resolves this by stretching the terrain height to match the path of the river based on its shape and on **BorderWidth** parameter. For information on **BorderWidth** and related settings, see [River Entity \(p. 480\)](#).



To create and place the River entity

1. In **Rollup Bar**, on the **Objects** tab, click **Misc, River**.
2. In your level, starting at the beginning of the river bed, click to place a series of points that define the river's path.
3. When complete, double-click at the end of the river bed.
4. In **Rollup Bar**, under **River Parameters**, click **Align Height Map** to adjust the terrain height to match the path of the river.

Applying a River Material

After you place the river entity, you can apply a material to the river. Rivers use the [WaterVolume Shader \(p. 1032\)](#).

To apply a material to a river

1. In **Rollup Bar**, on the **Objects** tab, click **Misc, River**.
2. Click **<No Custom Material>** to open Material Editor.
3. In Material Editor, expand the tree and select your asset.
4. Modify material settings and [WaterVolume Shader \(p. 1032\)](#) parameters as needed.
5. When finished, click **Assign Item to Selected Objects**, and close Material Editor.

Adjusting River Spline Geometry

You can make precise changes to the geometry of a river. You simply adjust the spline points and parameters.

To adjust river spline parameters

1. In **Rollup Bar**, on the **Objects** tab, click **Misc, River**.
2. Under **Spline Parameters**, click **Edit**, and do any of the following for the river in your level:
 - To move a point, drag it.
 - To add a new point, hold down **Ctrl** while clicking the spline at the desired location.
 - To delete a point, double-click it.

- To change the angle at a point, select it and adjust the **Angle** value.
- To change the width at a point, select it, clear the **Default width** check box, and adjust the **Width** value.

Tip

You can also change the positions of any spline point. Just select a point and use the **X**, **Y**, **Z**, and **XY** axis-lock buttons located at the top of Lumberyard Editor.

Splitting and Merging Rivers

You can split a river apart and merge two rivers together.

To split or merge rivers

1. In **Rollup Bar**, on the **Objects** tab, click **Misc, River**.
2. Under **Spline Parameters**, click **Edit**, and do either of the following in your level:
 - To split a river apart, select the desired point and click **Split**.
 - To merge two rivers together, select the end point of one river and the start point of another river. Then click **Merge**.

Adding Waterfalls

A waterfall is a natural feature to add to cliffs or when a river changes elevation or course. Waterfalls are placed as 2D decals in the terrain.

To add a waterfall

1. In Lumberyard Editor, click **View, Open View Pane, Material Editor**.
2. In Material Editor, select a suitable texture asset.
3. Under **Material Settings**, select the **Waterfall** shader.
4. Under **Texture Maps**, place the texture in the alpha channel of the **Diffuse** texture map.
5. Expand **DiffuseOscillator** and adjust parameter values to produce a realistic animation effect for the texture.
6. Under **Shader Params**, adjust **Foam** parameters as needed.
7. Adjust other material settings and shader parameters as needed.
8. Place the waterfall in your level, clicking to create a simple geometry that follows the terrain.
9. Apply water (rain) particle effects if desired.

Adding Water Puddles

To create realistic water puddles and water rifts, use non-tiling textures that can be placed as decals. While water puddles could be created as a water volume, using decals is less demanding on resources. For more information on decals, see [Working with Decals \(p. 1049\)](#).

For proper blending between the water puddle and the terrain, use an alpha channel with a smooth gradient so it fades into the terrain and the transition won't be noticeable.

To add a water puddle

1. In Lumberyard Editor, click **View, Open View Pane, Material Editor**.
2. In Material Editor, select a suitable material asset.

3. Under **Lighting Settings\Specular**, type **85, 85, 85**
4. In your level, click to place the puddle.
5. In **Rollup Bar**, on the **Objects** tab, click **Custom, GameVolume**.
6. Under **GameVolume Params**, click **VolumeClass** and select **WaterVolume**.
7. In your level, click boundary points around the puddle. Double-click the last point to complete the enclosure.

Adding Fog Above Water

You can add realistic-looking fog above water surfaces. For more information about Lumberyard's fog system, see [Fog Systems \(p. 1072\)](#).

To add fog above water

1. In your level, click to select the water volume entity above which you want to add fog.
2. In **Rollup Bar**, on the **Objects** tab, click **Area, WaterVolume, WaterVolume Params**, and modify the following parameters as needed.

FogDensity

Specifies how dense the fog appears.

FogColor

Sets the fog color.

FogColorMultiplier

Defines how bright the fog color is.

FogColorAffectedBySun

Enables the [Setting Sun Parameters \(p. 862\)](#) **Sun color** parameter value to affect fog color.

FogShadowing

Enables the surface of water to receive shadows. You can control the shadow darkness. Valid values are 0–1.

For this parameter to function, the console variable **r_FogShadowsWater** must be set to 1.

FogShadowing is only available when the **Config Spec** setting in Lumberyard Editor is set to **Very High**.

In addition, if the **VolFogShadows** property is enabled in the **Terrain\Environment** panel in **Rollup Bar**, shadow darkness is automatically set to full. However, the fog above the water will have volumetric shadowing.

CapFogAtVolumeDepth

If false, continues to render fog below the specified river depth.

Advanced Water Volume Parameters

The following advanced parameters apply to water volumes.

To set advanced Water Volume parameters

1. In Rollup Bar, on the **Objects** tab, click **Area**.
2. Under **Object Type** click **WaterVolume**.
3. Under **WaterVolume Params\Advanced**, adjust the following parameter values as needed:

FixedVolume

Traces a ray down to find a 'vessel' entity and 'spill' the requested amount of water into it. For static entities, it attempts to boolean-merge any surrounding static that intersects with the first vessel (use the **No Dynamic Water** flag on brushes that do not need that).

VolumeAccuracy

Water level is calculated until the resulting volume is within this (relative) difference from the target volume (if set to 0 it runs up to a hardcoded iteration limit).

ExtrudeBorder

Extrudes the border by this distance. This is particularly useful if wave simulation is enabled as waves can raise the surface and reveal open edges if they lie exactly on the vessel geometry.

ConvexBorder

Takes convex hull of the border. This is useful if the border would otherwise have multiple contours, which areas do not support.

ObjectSizeLimit

Only objects with a volume larger than this number takes part in water displacement (set in fractions of FixedVolume).

WaveSimCell

Size of cell for wave simulation (0 means no waves). Can be enabled regardless of whether FixedVolume is used.

WaveSpeed

Sets how "fast" the water appears.

WaveDamping

Standard damping.

WaveTimestep

This setting may need to be decreased to maintain stability if more aggressive values for speed are used.

MinWaveVel

Sleep threshold for the simulation.

DepthCells

Sets the depth of the moving layer of water (in WaveSimCell units). Larger values make waves more dramatic.

HeightLimit

Sets a hard limit on wave height (in WaveSimCell units).

Resistance

Sets how strongly moving objects transfer velocity to the water.

SimAreaGrowth

If changing water level is expected to make the area expand, the wave simulation grid should take it into account from the beginning. This sets the projected growth in fractions of the original size. If wave simulation is not used, this setting has no effect.

Copying and Moving Terrain Areas

You can copy and move areas or sections of terrain, including vegetation, water, and other objects in your level. You can also rotate sections of terrain.

To copy or move a section of terrain

1. In **Rollup Bar**, on the **Terrain** tab, click **Move Area**.
2. Click **Select Source** and then click in the level to define the volume that is copied or moved.
3. Click **Select Target** and click in the level to define the target volume location.
4. Adjust the values of the following parameters as needed.

Sync Height

Sets the Z position of the source and target volumes to the same value.

Target Rotation

Rotates the source volume counterclockwise by the selected amount when moved to the target location.

DymX, Y, Z

Changes the dimension of the source volume.

Only Vegetation

Moves or copies only vegetation and other objects and not the terrain itself.

Only Terrain

Moves or copies just the terrain and not vegetation or other objects.

5. Click **Copy** or **Move**.

Importing and Exporting Terrain Blocks

You can import and export terrain areas or blocks. When importing or exporting, you should also import or export the associated terrain texture layers.

To import a terrain block and texture layers

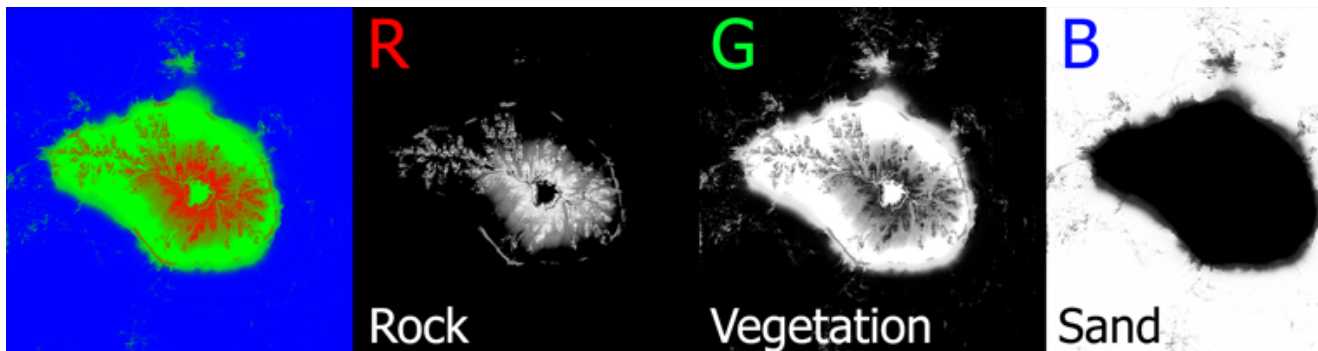
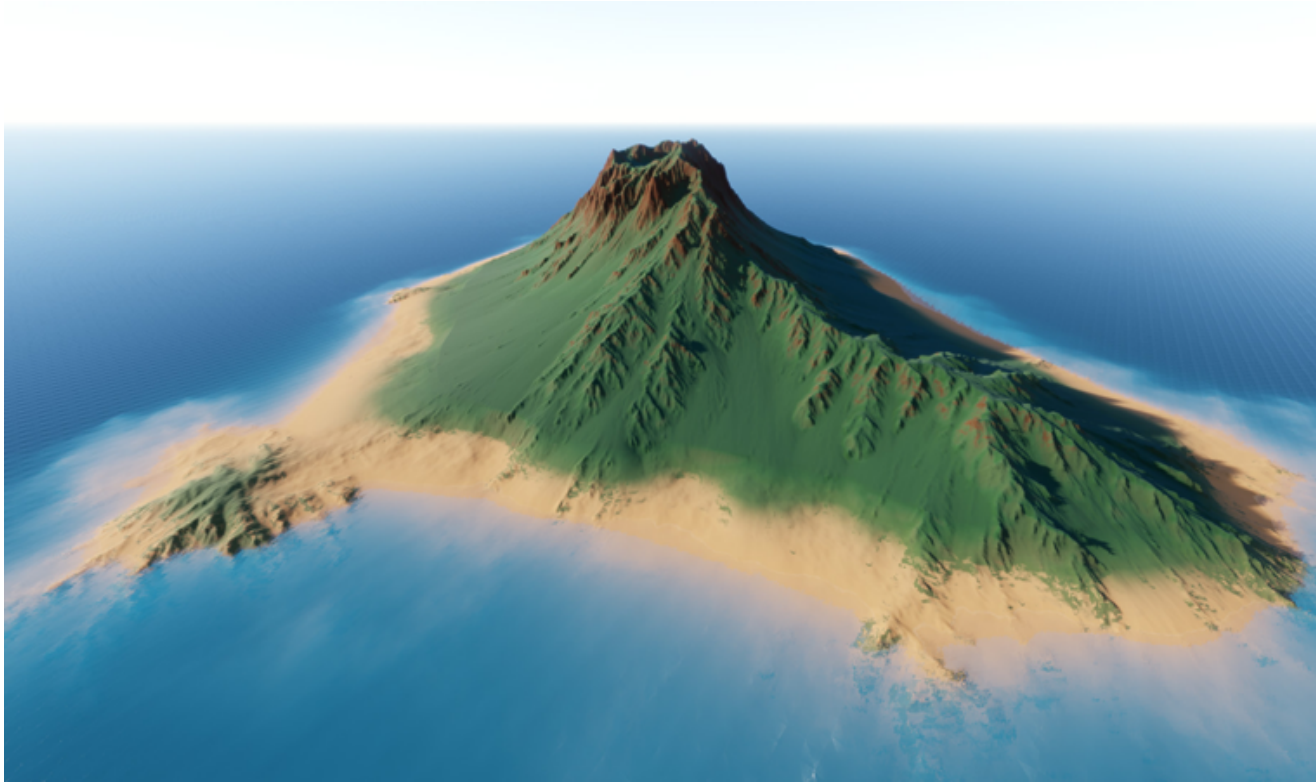
1. In Lumberyard Editor, click **Terrain, Import Terrain Block** and select a `.trb` file to import.
2. Click **Terrain, Terrain Texture Layers**.
3. Click **File, Import Layers** and select terrain texture files to import.

To export a terrain block and texture layers

1. In Lumberyard Editor, click **Terrain, Export Terrain Block**, and select a `.trb` file to export.
2. Click **Terrain, Terrain Texture Layers**.
3. Click **File, Export Layers**, and select terrain texture files to export.

Importing Splat Maps

Splat maps are 8-bit monochrome bitmap `.bmp` files that contain weight information for each vertex in a terrain map. Splat maps are generated using a DCC tool such as World Machine's Splat Converter.



All splat map operations in Lumberyard are done using the **Terrain Texture Layers** editor.

To import splat maps

1. In Lumberyard Editor, choose **Terrain, Terrain Texture Layers**.
2. In the **Terrain Texture Layers** editor, under **Layer Tasks**, assign each splat map to a texture layer by clicking a layer and then clicking **Assign Splat Map**.
3. When prompted, select a `.bmp` file to assign. You don't need to assign a splat map path to a layer, but you can't assign more than one path either.
4. Under **Layer Tasks**, click **Import Splat Map**. This clears the current weight map for the terrain and then rebuilds it using the selected splat maps.
5. In **Lumberyard Editor**, select **Terrain, Generate Terrain Texture**.

Note

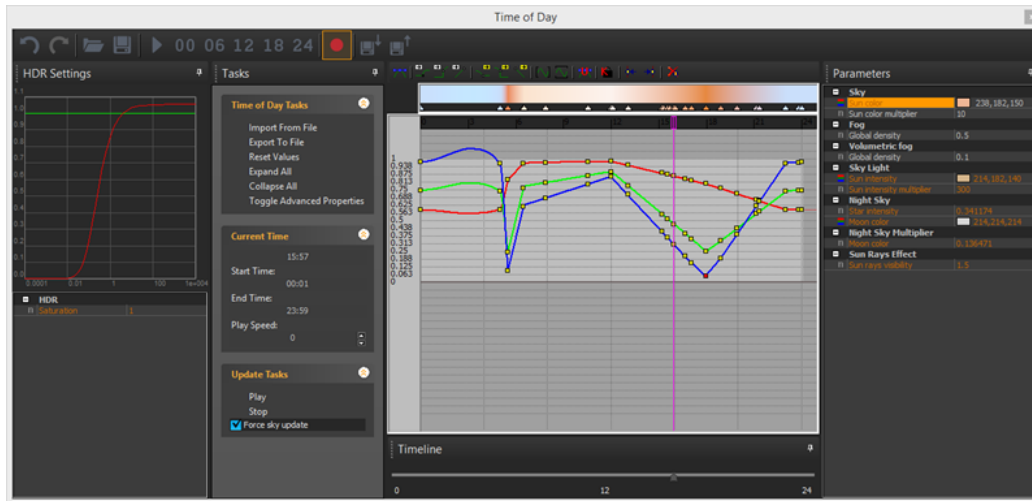
Once a splat map has been imported, it does not not apply any masking functionality during editing.

Adding Sky Effects

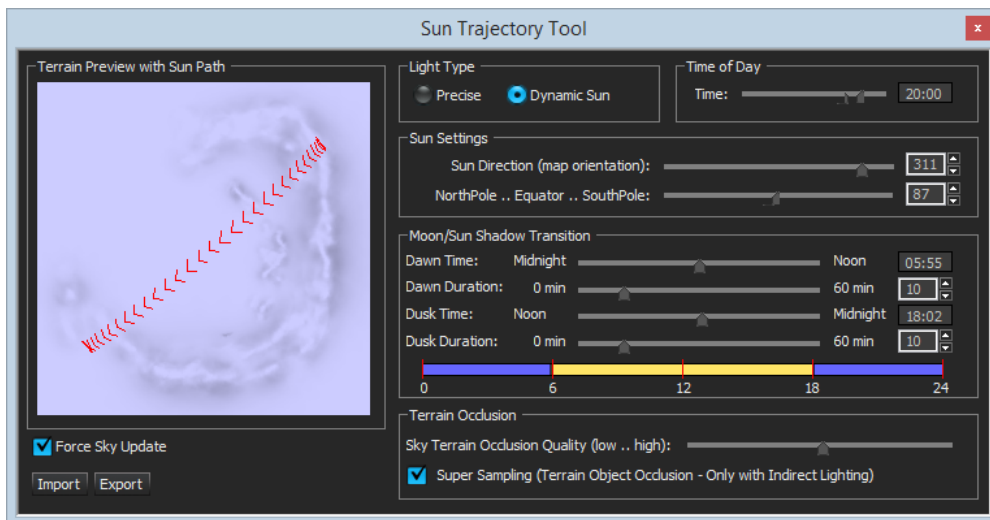
You can create realistic-looking skies by setting sun, moon, atmospheric, and time-of-day effects. You can create two types of skies: dynamic and static.

Dynamic skies use the [SkyHDR Shader \(p. 1023\)](#) to render realistic effects.

The primary tools used to create different sky effects for your level are the Time of Day Editor and the Sun Trajectory Tool, as shown below:



The Sun Trajectory Tool:



Topics

- [Creating a Dynamic Daytime Sky \(p. 861\)](#)
- [Creating a Dynamic Night Sky \(p. 863\)](#)
- [Creating Time of Day Sky Effects \(p. 864\)](#)
- [Creating a Static Sky \(SkyBox\) \(p. 867\)](#)

Creating a Dynamic Daytime Sky

To add a dynamic daytime sky, you adjust various sun parameters, atmospheric properties, sun ray effect, and sun shadows. Dynamic skies use the [SkyHDR Shader \(p. 1023\)](#).

All properties and parameters in the following topics are ignored when you use a static sky (SkyBox), which uses the [Sky Shader \(p. 1023\)](#).

Topics

- [Setting Daytime Atmospheric Effects \(p. 861\)](#)
- [Setting Sun Parameters \(p. 862\)](#)
- [Adding Sun Rays \(p. 862\)](#)
- [Setting Sun Shadow Settings \(p. 862\)](#)
- [Adding Cascaded Sun Shadows \(p. 863\)](#)

Setting Daytime Atmospheric Effects

To create dynamic daytime sky atmospheric effects, you modify sun and light-scattering setting that affect the appearance of distant objects, which shift in color due to atmospheric interference. These settings do not directly affect the rendering of objects or environment lighting colors and intensities.

To set daytime atmospheric effects

1. In Lumberyard Editor, choose **Terrain, Time Of Day**.
2. Under **Tasks, Time of Day Tasks**, choose **Toggle Advanced Properties** to view all settings.
3. Under **Parameters, Sky Light**, adjust the values of the following parameters:

Sun intensity

Uses an RGB sun color value to compute the atmosphere color. Used in conjunction with **Sun color** to provide desired scene luminance.

Sun intensity multiplier

Sets the brightness of the sun. Brightness is multiplied by sun intensity to yield the overall color. Used in conjunction with Sun color multiplier to provide desired scene luminance. Higher values result in brighter skies while low values can simulate an eclipse.

Mie scattering

Mie scattering is caused by pollen, dust, smoke, water droplets, and other particles in the lower portion of the atmosphere. It occurs when the particles causing the scattering are larger than the wavelengths of radiation in contact with them. Mie scattering is responsible for the white appearance of clouds.

Lower values result in a clear sky while larger values make the sky appear hazy. A good value is 4.8.

Rayleigh scattering

Rayleigh scattering consists of scattering from atmospheric gases. This occurs when the particles causing the scattering are smaller in size than the wavelengths of radiation in contact with them. As the wavelength decreases, the amount of scattering increases. Because of Rayleigh scattering, the sky appears blue.

The default value of around 2.0 produces blue sky during the day and red-yellow colors at sunset. Larger values create a red-yellow sky while lower values create a blue sky.

Sun anisotropy factor

Controls the sun's apparent size. As this value approaches -1.0, the sun's disk becomes sharper and smaller. Larger values produce a fuzzier and larger disk. A good value is -0.995.

Wavelength R, G, B

Sets the wavelengths (in nm) of the RGB values of sky colors. Adjusting the values shifts the colors of the resulting gradients and produces different kinds of atmospheres. Useful when used with Rayleigh scattering if you choose a sun intensity of pure, bright white.

Setting Sun Parameters

You can define how the sun appears in the daytime sky.

To set sun parameters

1. In Lumberyard Editor, choose **Terrain, Time Of Day**.
2. Under **Tasks, Time of Day Tasks**, choose **Toggle Advanced Properties** to view all parameters.
3. Under **Parameters, Sky**, adjust the values of the following parameters:
 - **Sun color** – Sets the RGB values of the sun disk color. Used in conjunction with **Sun intensity** to provide desired scene luminance.
 - **Sun color multiplier** – Sets the brightness of the sun. This value is multiplied by the sun color to yield the overall color. Used in conjunction with **Sun intensity multiplier** to provide desired scene luminance.
 - **Sun specular multiplier** – Controls the brightness and intensity of the sun on specular materials in your scene.

Adding Sun Rays

You can create a sun rays effect, which simulates the shafts of light that the sun produces under certain atmospheric conditions.

To add sun rays

1. In Lumberyard Editor, choose **Terrain, Time Of Day**.
2. Under **Time of Day Tasks, Time of Day Tasks**, choose **Toggle Advanced Properties** to view all parameters.
3. Under **Sun Rays Effect, Sun Rays Effect**, adjust the values of the following parameters:
 - **Sun shafts visibility** – (Deprecated) - This value controls the visibility of sun shafts. Higher values accentuate the shadow streaks that are caused by the sun light penetrating objects.
 - **Sun rays visibility** – Sets the brightness level of the sun rays.
 - **Sun rays attenuation** – Sets the length of the sun rays. Higher values produce shorter rays around the sun.
 - **Sun rays suncolor influence** – Sets the degree to which the color of the sun contributes to the color of the sun rays.
 - **Sun rays custom color** – Specifies a custom color the for the sun rays.

Setting Sun Shadow Settings

You can define how sun shadows appear in your level.

To set sun shadow settings

1. In **Rollup Bar**, under **Terrain**, choose **Environment**.
2. Under **EnvState**, adjust the values of the following:

- **Sun shadows min spec** – Specifies the minimum system specification for casting sun shadows.
- **Sun shadows additional cascade min spec** – Specifies the minimum system specification for rendering an additional sun shadow cascade at a larger viewing distance.

Adding Cascaded Sun Shadows

You can create multiple cascaded shadow maps for your level, which controls how sun shadows look at varying distances. The higher the cascade, the further it is away from the camera (cascade 0 is closest to the camera) and the lower the resolution of the shadows.

To create cascaded sun shadows

1. In Lumberyard Editor, choose **Terrain, Time Of Day**.
2. Under **Time of Day Tasks, Tasks**, click **Toggle Advanced Properties** to view all parameters.
3. Under **Parameters, Cascade Shadows**, adjust the parameter values for each shadow cascade as follows:
 - **Cascade *number* Bias** – Distance of the shadow connection from the shadow-casting object. Values between 0.01 and 0.05 produce the most realistic effect.
 - **Cascade *number* Slope Bias** – The slope gradient for the shadows. Higher values reduce shadows that are cast from an object with a high light angle. Values between 32 and 64 produce the most realistic effect. Slope bias has little to no impact on performance.
 - **Shadow Jittering** – The softness of all the cascaded sun shadows. Larger softness values impact performance.

Creating a Dynamic Night Sky

To add a dynamic nighttime sky, you adjust various horizon, moon, and stars settings. Dynamic skies use the [SkyHDR Shader \(p. 1023\)](#).

All properties and settings in the following topics are ignored when using a static sky (SkyBox).

Topics

- [Setting Nighttime Atmospheric Effects \(p. 863\)](#)
- [Setting Moon Parameters \(p. 864\)](#)

Setting Nighttime Atmospheric Effects

To add dynamic nighttime atmospheric effects, you set various horizon, moon, and star field parameters.

To set nighttime atmospheric parameters

1. In Lumberyard Editor, choose **Terrain, Time Of Day**.
2. Under **Tasks, Time of Day Tasks**, choose **Toggle Advanced Properties** to view all parameters.
3. Under **Parameters, Night Sky** and **Parameters, Night Sky Multiplier**, adjust the values of the following parameters:

Horizon color (and multiplier)

RGB value that is scaled by the multiplier and specifies the horizon color of the night sky gradient.

Zenith color (and multiplier)

RGB value that is scaled by the multiplier and specifies the zenith color of the night sky gradient.

Zenith shift

Shifts the night sky gradient. Small values shift it towards the bottom while larger values shift it towards the top.

Star intensity

Overall brightness of the stars. Star flickering is by design and cannot be controlled.

Moon color (and multiplier)

RGB value that is scaled by the multiplier specifies the moon's emissive color.

Moon inner corona color (and multiplier)

RGB value that is scaled by the multiplier specifies the color of the moon's inner corona.

Moon inner corona scale

Size and blurriness of the moon's inner corona. Smaller values produce a bigger, blurry corona while larger values produce a smaller, more focused corona.

Moon outer corona color (and multiplier)

RGB value that is scaled by the multiplier specifies the color of the moon's outer corona.

Moon outer corona scale

Size and blurriness of the moon's outer corona. Smaller values produce a bigger, blurry corona while larger values produce a smaller, more focused corona.

Setting Moon Parameters

You can define how the moon appears in the nighttime sky.

To set moon parameters

1. In **Rollup Bar**, on the **Terrain** tab, choose **Environment**.
2. Under **Moon**, adjust the values for the following parameters:
 - **Latitude** – Sets the latitude of the moon.
 - **Longitude** – Sets the longitude of the moon.
 - **Size** – Adjusts the size of the moon image.
 - **Texture** – Sets the asset for creating the texture. Choose the folder icon to access **Preview** and select a suitable asset.

Creating Time of Day Sky Effects

You can use time of day effects to create dynamic skies to simulate the changing lighting effects that are caused by the sun moving across the sky. You can also configure and store a complete day–night cycle of changing environment parameters to add realism to your level.

The **Time of Day** editor and **Sun Trajectory Tool** are used to achieve these effects.

Note

All properties and parameters in the following topics are ignored when using a static sky (SkyBox).

Topics

- [Setting Dawn and Dusk Effects \(p. 865\)](#)
- [Setting a Day-Night Cycle \(p. 865\)](#)

Setting Dawn and Dusk Effects

You can simulate the changing lighting effects that are caused by the sun moving across a dynamic sky. You can set sunrise time, duration of dawn, sunset time, duration of dusk, current time, and the path of the sun.

To set dawn and dusk effects

1. In Layer Editor, click **Terrain, Lighting**.
2. In **Sun Trajectory Tool**, set the following properties and parameter values

Time of Day

Sets the current time.

Sun Direction

Direction where the sun rises.

Dawn Time

Time of sunrise.

Dawn Duration

Duration of moon-to-sun lighting transition.

Dusk Time

Time of sunset.

Dusk Duration

Duration of sun-to-moon lighting transition.

Force sky update

If selected, updates the sky light calculations for each frame. If deselected, calculations are distributed over several frames.

Import

Imports settings from a saved lighting (.lgt) file.

Export

Exports current settings to a lighting (.lgt) file

Terrain Occlusion

Creates the effect of indirect lighting.

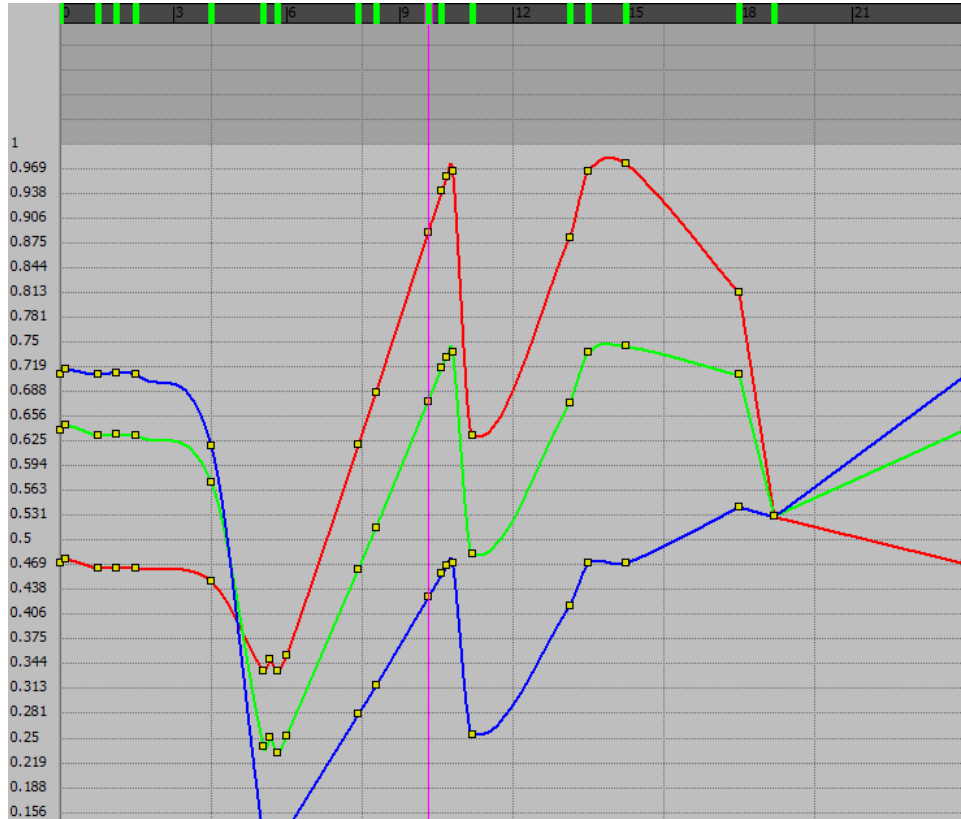
Super Sampling

Interpolates the pixels of indirect sampling data to eliminate hard transitions.

Setting a Day-Night Cycle

You can use the Time of Day Editor to configure changes to environment parameters over time to mimic a day-night lighting cycle. The Time of Day Editor uses a 24-hour timeline graph and a recording function to store changing environment parameter values in an XML file. To record, ensure the red button is selected.

Environment parameter values that you change in the **Parameters** panel of the Time of Day Editor are set for the currently selected time. The TOD graph shows the change of the selected parameter made over time. Each time a parameter value is changed, the graph curve is updated for the currently selected time. You can also directly change the curve by dragging it up or down between the key frame points. Key frame points are displayed as yellow dots. You can insert new key frame points by double-clicking the curve. To remove existing key frame points, double-click the key frames (yellow dots) themselves. Lumberyard interpolates parameter values for times that lie between key frame points.



To configure a day-night cycle

1. In Lumberyard Editor, click **Terrain, Time Of Day**.
2. Under **Time of Day Tasks**, click **Toggle Advanced Properties** to view all parameters.
3. Under **Parameters**, adjust the value of each parameter that you want to create a cycle for. Then do the following:
 - a. Click the red button to start recording.
 - b. Under **Tasks, Current Time**, set the time to apply the parameter value. The graph reflects the new value at the specified time.
 - c. Set a new parameter value and current time value pair. Repeat as many times as needed to get a realistic change over time for the parameter.
 - d. Click the red button to stop recording.
4. Under **Tasks**, complete the following tasks as needed to export, import, and play a time-of-day (day–night) cycle.

Import From File

Imports cycle settings from an .xml file.

Export To File

Exports cycle settings to an .xml file.

Reset Values

Resets all parameters to their default values.

Current Time

The current time in the Time of Day Editor.

Start Time

Time used when the game is started; not the same as the current time.

End Time

Time used when the game is ended. If the end time is set to 23.59, the time loops, starting the next cycle once the day is over.

Play Speed

Speed at which time advances in the cycle.

Play

Starts or resumes the playback of the cycle in the editor. If the current time value is not within the start and end times, playback begins at the specified start time.

Stop

Stops the playback of the cycle in the Time of Day Editor.

Force Sky Update

Updates the sky lighting calculations in each frame. If deselected, calculations are distributed over several frames. The effect may not be visible for some time.

Creating a Static Sky (SkyBox)

Static skies use the [Sky Shader \(p. 1023\)](#) and a SkyBox, which is a cube that uses textures on five of the sides (all except the bottom) to render a hemispheric dome to simulate the sky in your level. As such, static skies cannot take advantage of dynamic or animated Time of Day effects, HDR settings, and sun and moon parameters.

Topics

- [Setting SkyBox Parameters \(p. 867\)](#)
- [Asynchronous SkyBox Switching \(p. 868\)](#)

Setting SkyBox Parameters

Setting up a static sky involves using the [Sky Shader \(p. 1023\)](#), setting SkyBox parameters and placing the skybox in your level.

To set Skybox parameters

1. In Layer Editor, click **Terrain, Time of Day**.
2. Under **Tasks, Time of Day Tasks**, click **Toggle Advanced Properties** to view all parameters.
3. Under **Parameters**, under **Advanced**, set **Skybox multiplier** to 1.
4. In **Rollup Bar**, under **Terrain**, click **Environment**.
5. Under **Skybox**, click **Material**, and then select your asset.
6. In Material Editor, select a suitable asset from the browser tree pane, and then select a suitable material that uses the [Sky Shader \(p. 1023\)](#).
7. Under **Material Settings**, for **Shader**, make sure **Sky** is selected.
8. Adjust shader parameters as needed.
9. Click **Assign Item to Selected Objects** (left-most icon on toolbar). Close Material Editor.
10. Drag to place the previously-selected asset in your level.
11. In **Rollup Bar**, on the **Terrain** tab under **Environment, SkyBox**, adjust values for the following parameters:
 - **Material low spec**
 - **Angle**
 - **Stretching**

Asynchronous SkyBox Switching

Using Flow Graph, you can perform asynchronous SkyBox switching.

To perform asynchronous skybox switching

1. In **Rollup Bar**, select your asset.
2. Under **Flow Graph**, click **Create** and then type a name for the flow graph.
3. In Flow Graph, under **Flow Graphs**, select the entity.
4. Right-click in the flow graph grid and click **Add Node, Environment, Skybox Switch**.

Adding Weather Effects

Lumberyard offers a variety of realistic weather effects for your level environment.

Topics

- [Adding Wind Effects \(p. 868\)](#)
- [Adding Clouds \(p. 870\)](#)

Adding Wind Effects

You can create realistic wind effects in your level environment.

Topics

- [Adding Global Wind \(p. 868\)](#)
- [Adding Ocean Wind \(p. 869\)](#)
- [Creating Wind Areas \(p. 869\)](#)
- [Adding Localized Wind \(p. 869\)](#)

Adding Global Wind

Global wind and breezes affect everything in your level, such as all vegetation. Here's how to set them up:

To set global wind parameters

1. In **Rollup Bar**, under **Terrain**, click **Environment**.
2. Under the **EnvState** section, adjust values of the following parameters:
 - **Wind vector** – Speed and wind direction vector. Positive x values are east; positive y values are north.
 - **Breeze generation** – Enables breezes.
 - **Breeze strength** – Controls the intensity of the breeze.
 - **Breeze movement speed** – Controls the velocity of the breeze. Use it to produce short, rapid gusts of wind.
 - **Breeze variation** – Varies breeze speed, strength, and size.
 - **Breeze life time** – Sets the duration of each breeze, in seconds.
 - **Breeze count** – Sets the number of breezes generated per instance.
 - **Breeze spawn radius** – Radius of breeze travel.
 - **Breeze spread** – Determines the degree of variation in breeze direction.

- **Breeze radius** – Sets the radius of breeze influence.

Adding Ocean Wind

You can simulate realistic wind and wave effects for the ocean in your level.

To set ocean wind parameters

1. In **Rollup Bar**, on the **Terrain** tab, under **Terrain**, click **Environment**.
2. Under the **OceanAnimation** section, adjust the following parameters:
 - **Wind direction** – Sets the wind direction from 1 to 4 in 90 degree increments.
 - **Wind speed** – Sets the wind speed for surface waves.
 - **Wave frequency** – Sets the frequency of waves. Smaller values mean fewer, longer waves (deep ocean depth). Larger values mean more, shorter waves (shallow ocean depth).
 - **Wave height** – Sets wave height in meters by means of vertex displacement.

Creating Wind Areas

Wind areas define a location within which objects experience wind. If no direction is set, wind moves omnidirectionally from the center of the wind area.

To create a wind area

1. In **Rollup Bar**, under **Objects**, click **Entity**.
2. Under **Browser**, expand **Physics** and double-click **WindArea**.
3. Drag to place the entity in your level. A bounding box with direction areas appears.
4. Under **Entity Properties**, adjust values of the following parameters:
 - **Active** – Enables or disables wind inside the area.
 - **AirDensity** – If greater than 0, causes objects moving through the air to slow down.
 - **AirResistance** – If greater than 0, causes lightweight objects to experience buoyancy.
 - **Ellipsoidal** – Specifies an ellipsoidal drop off in air speed.
 - **FalloffInner** – Sets the distance at which distance-based air speed begins to drop off.
 - **Speed** – Sets the wind speed.
 - **Dir** – Sets the wind direction.
 - **Size** – Sets the size of the wind area.

Adding Localized Wind

Localized wind is used to simulate wind from a specific object, such as a fan or jet exhaust. You set up localized wind with the wind entity.

To set localized wind parameters

1. In **Rollup Bar**, under **Objects**, click **Entity**.
2. Under **Browser**, expand **Physics** and double-click **Wind**.
3. Drag to place the entity in your level at the desired location.
4. Under **Entity Properties**, adjust the following parameters:
 - **FadeTime** – Enables or disables fade time.

- **vVelocity** – Sets the wind strength and direction.

Adding Clouds

You can create realistic-looking clouds in your level that move, cast shadows, and that objects can fly through.

Topics

- [Setting Cloud Shading Parameters \(p. 870\)](#)
- [Adding 3D Cloud Shadows \(p. 870\)](#)
- [Creating 3D Cloud Templates \(p. 871\)](#)

Setting Cloud Shading Parameters

Cloud shading, unlike cloud shadows, effects the brightness and color of clouds in your level. The environment sky and sun color affect how clouds look.

To set cloud shading parameters

1. In Lumberyard Editor, click **Terrain, Time Of Day**.
2. In **Time of Day Editor**, under **Tasks**, click **Toggle Advanced Properties**.
3. Under **Parameters\Cloud Shading**, adjust the following parameters:
 - **Sun contribution** – Specifies how much the sun affects the cloud brightness.
 - **Sky contribution** – Specifies how much the sky light affects the cloud brightness.
 - **Sun custom color** – Sets the RGB sun color.
 - **Sun custom color multiplier** – Sets the brightness of the sun, which is multiplied by the sun custom color.
 - **Sun custom color influence** – Sets the degree to which the color of the sun contributes to the color of the clouds.

Adding 3D Cloud Shadows

3D clouds don't actually cast real-time shadows. Instead a moveable texture is imposed on the entire level, creating the illusion that the clouds cast shadows.

To add 3D cloud shadows

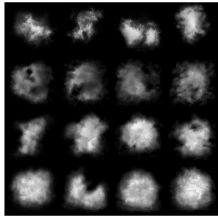
1. In **Rollup Bar**, under **Terrain**, click **Environment**.
2. Under **CloudShadows**, click **Cloud shadow texture** and the folder icon.
3. In **Preview**, select a suitable asset.
4. Drag the shadow to the desired location in your level.
5. Under **CloudShadows**, adjust the following parameters:
 - **Cloud shadow speed** – Sets the speed that shadows move across the terrain.
 - **Cloud shadow tiling** – Sets the tiling multiplier of the shadow texture.
 - **Cloud shadow brightness** – Sets the brightness level of the shadow.
 - **Cloud shadow invert** – Enables inverting of the cloud shadow texture.
6. In the **Console** window, click the ... button.
7. In **Console Variables**, set the variable **e_GsmCastFromTerrain** to 1.

Creating 3D Cloud Templates

You can use the Clouds tool in Lumberyard Editor to create new cloud template XML files. You can use those template files later to add and place clouds as described in previous procedures. For more information, see the topics listed in [Adding Clouds \(p. 870\)](#).

The basic process for creating a cloud template is to create an area box that defines the size of the cloud, assign a material, select the [Common.Cloud Shader \(p. 1003\)](#), and then export and save the template. See the following procedure for details.

All clouds use a texture map, which is made up of multiple sprites that are organized into columns and rows, as the following image shows. You create cloud texture maps using your DCC tool.



To create a new 3D cloud template

1. In **Rollup Bar**, under **Objects**, click **Area, AreaBox**.
2. Under **AreaBox**, click **<No Custom Material>**.
3. In **Material Editor**, select the cloud texture map you created in your DCC.
4. Under **Material Settings**, for **Shader**, select **Common.Cloud**.
5. Under **Shader Params**, adjust the parameters for the desired effect.
6. Click **Assign item to Selected Objects**. Close the Material Editor.
7. Click to place the area box in your level.
8. In Lumberyard Editor, click **Clouds, Create**, and type a name for the cloud template.
9. Under **Cloud Params**, adjust the following cloud texture map parameters for desired effect.

Number of Rows

Sets the number of sprite rows in the cloud texture. Leave at 4 when using the default cumulus_01.dds texture.

Number of Columns

Sets the number of sprite columns in the cloud texture. Leave at 4 when using default cumulus_01.dds texture.

Sprite Row

Designates a row in the cloud texture for rendering.

Number of Sprites

Sets the number of sprites to be generated in the cloud.

Size of Sprites

Sets the scale of the sprites in the cloud.

Size Variation

Defines the randomization in size of the sprites within the cloud.

Angle Variations

Defines limits of randomization in the rotation of the sprites within the cloud.

Minimal Distance between Sprites

Defines the minimum distance between the generated sprites within the cloud.

Every Box has Sprites

Density

Show Particles like Spheres

Turns on additional sphere rendering for each sprite generated.

Preview Cloud

Renders the generated cloud.

Auto Update

Updates the cloud rendering automatically with each parameter change.

10. Click **Generate Clouds**. The cloud should be visible inside the **AreaBox** in your level.
11. Click **Export**, then save the cloud template in a suitable directory.

Working with Layers

You use level layers to organize objects and content, as well as for streaming. Any object placed on a layer can be hidden or unhidden using layers or Flow Graph. Doing so keeps performance high and memory consumption low.

You can also divide layers into sub-layers and into action bubbles, which represent the logical steps the player progresses through on the level.

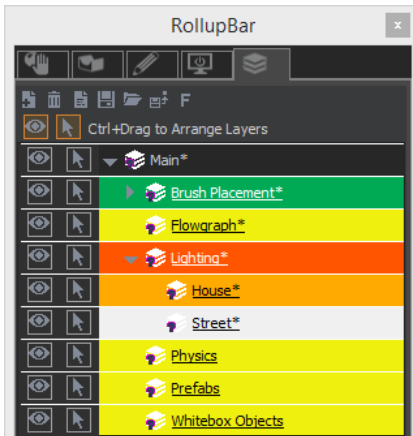
Layers are stored as `.LYR` files on disk.

Topics

- [Managing Level Layers \(p. 872\)](#)
- [Assigning Objects to Layers \(p. 873\)](#)
- [Streaming and Switching Layers \(p. 874\)](#)

Managing Level Layers

Layers are used in a level to group similar objects like brushes and entities together.



To create a new layer

1. In **Rollup Bar**, on the **Layers** tab, click the New Layer icon (left-most option).
2. Type a name for the layer, and then enable or disable the various layer settings as needed.
3. After you create a layer, you can manage it by right-clicking the layer name in **Rollup Bar** and doing the following:
 - To delete the layer, click **Delete**.
 - To export the layer, click **Export**.

- To reload the layer, click **Reload**.
- To import a layer that has been previously exported, click **Import Layers** and then select the applicable layer(s).
- To group or nest a layer under another layer, press **Ctrl** and the left mouse button while you drag the layer onto the layer that you want to group it under, then release.
- To ungroup a layer, press **Ctrl** and the left mouse button while you drag the layer to an empty space at the bottom of the layer list, then release.

Level Layer Settings

You can modify a layer's settings in the Layer dialog box.

To change a layer's settings

1. In **Rollup Bar**, click the **Layers** tab.
2. Right-click the layer and click **Settings**.
3. Modify the following settings as desired and then click **OK**:

Visible

When enabled, the layer is visible.

Frozen

When enabled, object interaction on a layer is disabled (frozen)

Export Layer Pak

TBD

Load By Default

When enabled, loads the layer when the level is loaded. Enable this when layer streaming is enabled and layer objects are visible in the level. When disabled, the layer is not loaded initially when the level is loaded. To enable layer streaming, see [Streaming and Switching Layers \(p. 874\)](#).

Export to *.lyr

When enabled, layer is saved in the `.lyr` file format on disk when you save the level. When disabled, the layer is stored as an external layer and is underlined to indicate such.

Use In Game

When enabled, the layer is exported to the game when the level is exported to Lumberyard (click **File, Export to Engine**).

Enable Physics

When enabled, objects on the layer respond to physics.

Export to All Platforms

Exports layer to all supported game platforms.

Export to Specific Platform(s)

Exports layer to selected game platform(s) only.

Assigning Objects to Layers

You can assign objects to a specific layer for a given level so you can control them as a group. By default, all objects are placed in the Main layer unless you assign it to another layer.

If a different layer has been selected, any objects you add to the level are automatically assigned to that layer.

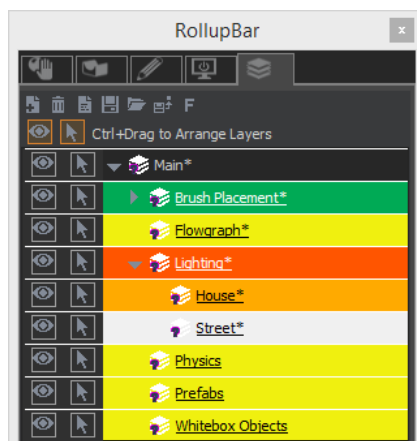
To assign an object to a layer

1. In Lumberyard Editor, select an object in the viewport.

2. In **Rollup Bar**, on the **Objects** tab, under **Entity**, click the layer icon, and then select the desired layer.

To control objects on a layer

1. In **Rollup Bar**, click the **Layers** tab.
2. Select a layer and do any of the following:
 - To control object visibility, toggle the eye icon to display or hide all objects on the layer.
 - To control object interaction, toggle the arrow icon to enable or disable (freeze) interaction with all visible objects on the layer.
 - To toggle visibility or interaction for all objects on all layers, toggle the eye or arrow icons above the layer list.



Streaming and Switching Layers

Layer streaming and layer switching control the visibility of all entities and geometry on a level in real time. Without the efficient use of layer switching, game performance can quickly degrade. It is important to find suitable locations within a level where layer switching should take place, such as between rooms, or between indoors and outdoors for example.

The following guidelines and limitations apply to layer streaming and switching:

- Make sure that the starting area in your level layer is visible (not hidden) at game start. When a level is loaded, many entity types are automatically hidden for memory optimization.
- Triggers inside a hidden layer do not function so do not switch layers that contain triggers.
- Physics proxies are not affected by layer switching.
- Use the `es_LayerDebugInfo 1` console variable to display all active layers for debugging purposes.

Layer Streaming

Layer streaming is disabled by default. To use it, you must first enable it.

To enable layer streaming

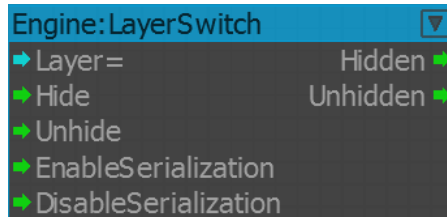
1. In **Rollup Bar**, on the **Terrain** tab, click **Environment**.
2. Under **EnvState**, select **Use layers activation**.

Layer Switching

You use Flow Graph to set up the switching of layer visibility. The logic is simple:

1. Start the game.
2. Hide layer B and show layer A when the player is in proximity.
3. Switch visibility when the player is going from location A to B.

The following figure shows the flow graph node used to switch layers.



To switch layer visibility

1. In **Rollup Bar**, on the **Objects** tab, click **Entity**.
2. Under **Browser**, expand **Default** and double-click **FlowgraphEntity**.
3. Scroll down to the **Flow Graph** section if needed and click **Create**.
4. Type a group name for the Flow Graph and click **OK**.
5. In the **Flow Graph** window, under **Flow Graphs**, select `FlowgraphEntity1`.
6. Right-click anywhere in the graph, and then click **Add Node, Engine, LayerSwitch**.
7. In the **Engine:LayerSwitch** node, double-click **Layer=** and choose a layer from the list to be switched. Then connect the following inputs and outputs as needed.
 - **Hide** – Input that hides the layer.
 - **Unhide** – Input that makes the layer visible.
 - **EnableSerialization** – Input that enables layer serialization.
 - **DisableSerialization** – Input that disables layer serialization.
 - **Hidden** – Output that signals that the layer is now hidden.
 - **Unhidden** – Output that signals that the layer is now visible.

For more information about flow graphs and how to connect inputs and output, see the [Flow Graph System \(p. 487\)](#)

Adding Vegetation

You can add realistic trees, bushes, grasses, and other vegetation to your Lumberyard terrain.

Topics

- [Vegetation Best Practices \(p. 876\)](#)
- [Vegetation Recommendations \(p. 876\)](#)
- [Vegetation Texture Mapping \(p. 876\)](#)
- [Adding Trees and Bushes \(p. 877\)](#)
- [Adding Grass \(p. 877\)](#)
- [Adding Vegetation Bending Effects \(p. 878\)](#)

- [Vegetation Parameters](#) (p. 880)
- [Vegetation Debugging](#) (p. 881)

Vegetation Best Practices

Keep in mind the following best practices, recommendations, and guidelines when you add vegetation to your terrain level.

- Manually place vegetation to get the most control and best results.
- To save memory, place grass manually.
- Keep the polygon count for grass blades as low as possible.
- Do not exceed a diameter of 8 meters for grass patches. This size provides a balance between performance and coverage.
- Grasses and small plants do not require specular or opacity texture maps. For more information, see [Working with Textures](#) (p. 1044).
- Set the **Opacity** texture at a much lower resolution than the other maps.
- Use a **Glossiness** value of 8 or above for realistic results.
- Use the automerged method to apply wind bending effects to grass.
- Use a maximum of 72 bones per tree for touch bending.

Vegetation Recommendations

The following settings are recommended when creating vegetation in your DCC tool.

Vegetatio	Polygon Range	Texture Size	Proxies	Material IDs
Grass	0-300	512x512	Bending	Grass, grass proxy
Bushes	300-600	1024x1024	Bending, collision	Leaf, leaf proxy
Small Trees	600-1000	(2) 1024x1024	Bending, collision **	Trunk, leaf, leaf proxy
Large Trees		(2) 1024x1024	Bending, collision ***	Trunk, leaf, leaf proxy, trunk proxy

** Smaller breakable tree trunks are physicalized.

*** Larger non-breakable tree trunks are not physicalized.

Vegetation Texture Mapping

Vegetation gets its appearance from texture mapping. Trees use two different sets of textures maps, one for leaves and branches and one for the trunk. Normal and specular maps can have a gloss map in the alpha channel.

The texture map you use depends on the type of vegetation:

Grass – Diffuse map only

Leaves and branches (trees or bushes) – Diffuse, specular, normal, and opacity maps

Tree trunks – Diffuse, specular and normal maps

Vegetation placement on a terrain texture layer is based on the pivot point of the vegetation object. Bigger vegetation objects might overlap with other terrain texture layers. This is most obvious if you have two different materials touching, like grass and mud.

Adding Trees and Bushes

You can add realistic trees and bushes to your terrain in your environment level. You must add trees and bushes manually.

To add trees or bushes

1. In **Rollup Bar**, on the **Objects** tab, click **Geom Entity**.
2. Under **Browser**, select the desired vegetation.
3. Drag to place the tree or bush in your level.

Adding Grass

You can add realistic grass to your terrain in your environment level. You can drag to place and quickly paint the entire terrain, or manually click clump-by-clump to provide the most control and best results.

Topics

- [Adding Grass Manually \(p. 877\)](#)
- [Painting to Add Grass \(p. 877\)](#)

Adding Grass Manually

Although you can paint in your terrain to add grass quickly, the manual approach saves memory and results in better control and a more realistic effect.

To manually add grass

1. In Lumberyard Editor, click **View, Open View Pane, Material Editor**.
2. Expand the left tree and select a suitable asset.
3. Under **Material Settings**, select the **Vegetation** shader.
4. Under **Shader Generation Params** select **Grass**.
5. Modify other settings and parameter values for the desired effect.
6. Click **Assign Object to Item**. Close Material Editor.
7. If necessary, depending on your terrain, in **Rollup Bar**, on the **Terrain** tab, select the **AlignToTerrain** check box.
8. Click to place grass in your level and repeat as needed.

Note

When you add or move grass, it may sporadically jump around. This happens if you move vegetation to a location that is too dense to accommodate it. When this occurs, the vegetation moves to its last position and is outlined in red. You can then move it elsewhere or delete it.

Painting to Add Grass

You can drag the mouse to quickly paint all terrain in your level with grass. This method of placing vegetation is controlled by the texture layer that the vegetation object is associated with. Painted

vegetation is visible wherever the texture layer appears. This is a quick way to automatically cover a huge area with grass.

Note

Painting a level with grass consumes 8 MB of memory cache.

To add grass by painting

1. In **Rollup Bar**, on the **Terrain** tab, click **Vegetation**.
2. Under **Vegetation**, click **Add Vegetation Object**.
3. In **Preview**, select a suitable asset.
4. Click to place the grass in your terrain.
5. Under the **Use on Terrain Layers** parameter, select the check box for your asset. The terrain should now be covered with the grass object.

Adding Vegetation Bending Effects

Lumberyard provides three methods for adding realistic bending motions to vegetation:

- **Touch (Collision) Bending** – bending effects for larger vegetation caused by players brushing against or colliding with branches
- **Detail (Wind) Bending** – physically accurate wind effects for larger vegetation defined by using vertex colors and environment wind parameters
- **Automerged (Wind) Bending** – physically accurate wind effects for grass defined by vegetation and environment wind parameters

You can use touch and detail bending effects together. For example, a player can brush against a branch that is also swaying in the breeze. Use automerged bending by itself for objects like grass.

From a performance standpoint, detail bending is the least expensive, touch bending is more expensive, and automerged bending is the most expensive.

Topics

- [Adding Touch \(Collision\) Bending Effects \(p. 878\)](#)
- [Adding Detail \(Wind\) Bending Effects \(p. 879\)](#)
- [Using AutoMerged Wind Bending Effects \(p. 879\)](#)

Adding Touch (Collision) Bending Effects

The touch bending technique simulates a player touching, brushing against, and interacting with vegetation. Use it for bushes, branches, and bigger leaves with stems. To implement touch bending, you use UV layout instancing.

All touch-bendable vegetation uses a collision proxy to define the volume of bending effect. By using a collision volume proxy, touch is detected inside the volume. This volume should be large enough to enclose all branches that are affected by touch bending. The proxy is physicalized using the **noCollide Physics** setting.

Using UV Layout Instancing

UV instancing for touch bending is a type of bone-and-rope technique. By sharing the same UV space, objects can inherit the joint setup from a "master leaf."

To create UV instances, you duplicate the master leaf of an element or cluster within the same object. You can rotate, translate, scale, and even change an instance's shape simply by moving individual vertices without changing vertex count.

To control where branches and leaves should bend, you place joints (also called helpers or locators) at various positions on a master leaf, including the tip. You must follow a specific naming convention for the joints, such as branch1_1 (first branch, first joint at the base)—Branch1_1 is the base and does not move.

Make sure the joints snap to the same location as the vertex nodes. Lumberyard interpolates between these joints using a rope setup, and weights all other joints automatically.

Adding Detail (Wind) Bending Effects

Detail bending refers to the procedural movement of larger foliage caused by wind or other similar effects. You control the motion by the use of vertex colors in your DCC tool.

When you use detail bending, make sure the distribution of polygons on foliage geometry is regular and properly tessellated. Otherwise you may see visual artifacts. Also make sure that leaves do not belong to a single node.

Defining Vegetation Vertex Colors

Vertex colors are used to specify detail wind bending effects for vegetation objects. All three RGB channels are used to control the movement of the geometry. Using your DCC tool, each channel should be edited and viewed separately.

Color	RGB Values	Bending Influence
Red	100/0/0	Irregular bending at the outsides – movement of smaller shapes
Green	0/100/0	Delays the start of the movement – used to create variations.
Blue	0/0/100	Bends the leaves up and down – movement of the big shapes.

Setting the Detail Bending Parameter

Once vertex colors are defined, set the **Bending** parameter for detail bending. This value controls the procedural bending deformation of vegetation. It ranges from 0 to 100 in value, 0 meaning no bending effect and 100 meaning the maximum effect, when receiving environment wind and breezes. For more information, see, [Adding Global Wind \(p. 868\)](#).

To set the vegetation detail bending parameter

1. In **Rollup Bar**, on the **Terrain** tab, click **Vegetation**.
2. Under **Objects**, select your asset.
3. Click **Bending**, and adjust the value from 0 - 100 as needed.

Using AutoMerged Wind Bending Effects

Automerged vegetation has physically accurate wind motions that are defined by wind environment properties and various vegetation properties. It is recommended for use with grass only.

Automerged vegetation reduces the number of draw calls while still allowing you to add any amount or size of grass patches to the terrain. It merges multiple meshes within several sectors as long as they

are using the same material and texture. You can paint single grass blade objects on the terrain as well as on brushes in different heights independently while they get merged into larger chunks in real time.

When AutoMerged is enabled, touch bending, vertex colors, and detail bending settings are all ignored, and vegetation movement is defined solely by the AutoMerged parameters. For more information, see [Adding Touch \(Collision\) Bending Effects \(p. 878\)](#) and [Adding Detail \(Wind\) Bending Effects \(p. 879\)](#).

To enable AutoMerged vegetation and set parameters

1. In **Rollup Bar**, on the **Terrain** tab, click **Vegetation, Add Vegetation Object**.
2. Expand **Objects** tree and select the grass object you want to modify.
3. Select the **AutoMerged** check box, and adjust the following parameter values:
 - **Stiffness** – Defines the stiffness of the vegetation
 - **Damping** – Specifies the amount of damping on the bending motion
 - **AirResistance**— Specifies the amount of bending similar to the **Bending** parameter used for [Setting the Detail Bending Parameter \(p. 879\)](#).

The four AutoMerged parameters together define the amount and type of bending motions the vegetation object displays in reaction to wind and breezes. For more information, see, [Adding Global Wind \(p. 868\)](#).

Vegetation Parameters

The following vegetation parameters can be accessed in the **Terrain, Vegetation** panel in **Rollup Bar** for a previously selected vegetation object. You can adjust parameters for one or more selected objects.

Unless otherwise noted, parameters apply to newly added and placed vegetation assets only.

Size

Changes the size of newly placed vegetation objects. Use this to uniformly scale the vegetation, where 1 represents 100%.

SizeVar

Changes the limit of size changes for a set of newly placed vegetation objects of a single kind. Keep Size at 1 and set SizeVar to 0.2 to get a nice variation in sizes.

RandomRotation

Randomly rotates objects while you paint new vegetation objects. To create a more natural look and distribution, you can set up a RandomRotation in the vegetation objects when you paint them. This feature works only with the Paint Object tool.

AlignToTerrainCoefficient

Points the vegetation object away from the ground. When this effect is applied, vegetation on cliffs points away from the surface instead of growing straight up.

UseTerrainColor

Makes the individual object receive the color of the underlying terrain for a better match. Use this option to blend the grass with the underlying terrain color. You can also use this setting on other objects, but it works best with grass. This effect is especially useful for making grass appear to fade in the distance.

AllowIndoor

Enables the vegetation to be rendered within vis areas.

Bending

This value controls the bending deformation of the vegetation objects. It ranges from 0 to 100, with 0 representing no bending effect and 100 the maximum effect. This effect is based on the amount of environment wind (WindVector) in the level.

GrowOnBrushes

Controls the placement of objects on brushes.

GrowOnTerrain

Controls the placement of objects on terrain. Useful if you want them placed only on brushes.

AutoMerged

Enables AutoMerged system on this object. For more information, see [Using AutoMerged Wind Bending Effects \(p. 879\)](#).

Stiffness

Controls the stiffness of selected vegetation and how much it reacts to physics for AutoMerged vegetation.

Damping

Determines how responsive the vegetation is to physics damping for AutoMerged vegetation.

AirResistance

Degree that vegetation resists air movement (wind). Similar to the Bending setting but specifically designed for AutoMerged vegetation.

Pickable

Allows the player to pick up the object.

Density

Adjusts the distance between individual objects that you create while painting new vegetation. The density setting ranges from 0 to 100. If your density setting is bigger than your brush radius, the vegetation will not be created, so always make sure you have a suitable brush radius.

ElevationMin

Limits the minimum height at which you can paint vegetation objects. For painting underwater vegetation, set this value to lower than the ocean; 0 is a safe option.

ElevationMax

Limits the maximum height at which you can paint vegetation objects.

SlopeMin

Limits the minimum angle of the terrain on which you can paint vegetation objects. 255 equals 90 degrees. When you specify a SlopeMin value higher than 0, you can no longer place objects on flat grounds.

SlopeMax

Limits the maximum angle of the terrain on which you can paint vegetation objects. 255 equals 90 degrees. When you specify a SlopeMax lower than 255, you can no longer place objects on very steep areas.

CastShadow

Makes the object cast a shadow based on the minimum selected **Config Spec** setting. For example, High won't work on Low or Medium specs.

Vegetation Debugging

Branches and tree trunks can be broken upon collision.

e_vegetation 1 | 0

Enables and disables rendering of the vegetation. 1 = on, 0 = off.

e_MergedMeshesDebug 1

Displays statistics on global memory consumption of vegetation objects placed in the level.

e_MergedMeshesDebug 2

Displays vegetation in the cells that form the merged meshes. They are color coded over distance. Red boxes should be displayed only around the player (the cell the player is standing in and the surrounding eight cells). Beyond this, all cells should be green.

Displayed above each cell is information about the current LOD step and memory consumption for the cell—this updates as you move closer and further away.

Mobile Support

Mobile support is in preview release and is subject to change.

You can use Lumberyard to build your games for Android devices such as the Nvidia Shield, Samsung Galaxy Note 5, and Motorola Nexus 6, and iOS devices that use the A8 GPUs, including iPhone 6s, iPhone 6s Plus, iPad Air 2, and iPad Pro. Lumberyard includes two Android-supported sample projects and four iOS-supported sample projects that you can use to learn how to build assets, build shaders using the remote shader compiler, and build the Lumberyard runtime (Android) or iOS applications using the Lumberyard build tools.

Topics

- [Android Support \(p. 882\)](#)
- [iOS Support \(p. 905\)](#)
- [Design Considerations for Creating Mobile Games Using Lumberyard \(p. 916\)](#)
- [Adding IP Addresses to Allow Access to the Asset Processor and Remote Console \(p. 919\)](#)

Android Support

Mobile support is in preview release and is subject to change.

You can use Lumberyard to build your games for Android devices such as the Nvidia Shield, Samsung Galaxy Note 5, and Motorola Nexus 6. Lumberyard includes two Android-supported sample projects that you can use to learn how to build assets for Android, build shaders using the remote shader compiler, and build the Lumberyard runtime using the build tools.

Prerequisites

To build games for Android, Lumberyard requires the following:

- Visual Studio 2015 with Update 1 or later for debugging (PC only)
- SDK-19 (Android 4.4.2) to SDK-23 (Android 6.0)
- Your device [set up for development](#) and connected to your computer using a USB cable

Note

You can build games for Android on a Mac; however, the Asset Processor and shader compiler require a PC.

Setting Up Your PC

After you download and extract Lumberyard on your PC, you must extract and run Lumberyard Setup Assistant to install the third-party software that is required to run the game and compile the game code, engine and asset pipeline, and for Android devices.

To install third-party software using Lumberyard Setup Assistant

1. Run Lumberyard Setup Assistant by double-clicking `SetupAssistant.bat`, which is located in the Lumberyard root directory (`\lumberyard\dev`).
2. In Lumberyard Setup Assistant, on the **Get started** page, select **Compile for Android devices** and click **Next**.
3. Follow the instructions on the screen to complete the installations for any third-party software or SDKs that you need. For more information about using Lumberyard Setup Assistant, see [Using Lumberyard Setup Assistant to Set Up Your Development Environment \(p. 14\)](#).
4. Modify your environment variables by doing the following:
 - a. In the Windows **Control Panel**, click **System, Advanced system settings**.
 - b. In the **System Properties** dialog box, click **Environment Variables**.
 - c. Under **User variables**, edit the **PATH** variable to add the directory where you installed the Android SDK and the platform-tools and tools subdirectories. For example: `C:\Android\android-sdk, C:\Android\android-sdk\platform-tools, C:\Android\android-sdk\tools`
 - d. Add the Java SDK and JRE to the **PATH** variable. For example: `C:\Program Files\Java\jdk1.7.0_79\bin` and `C:\Program Files\Java\jre7\bin`
5. Locate the directory where you installed the Android SDK. Run the **SDK Manager** and select the version of the SDK that you want to install. You must also install a version of the build tools. Note the version you installed.
6. Modify configuration files to tell Lumberyard which version of the SDK to use when building your game:
 - a. In the File Explorer, locate `_waf_\android` in the directory where you installed Lumberyard.
 - b. Edit the `android_settings.json` file to set `BUILD_TOOLS_VER` with the version of the build tools that you just installed and to set `SDK_VERSION` with the version of the SDK that you want to use.
 - c. Save the file.
7. In a command line window, change to the `\lumberyard\dev` directory.
8. To initialize the build system, run the following command:

```
lubr_waf.bat configure
```

Setting Up Your Mac

After you download and extract Lumberyard on your Mac, you must extract and run Lumberyard Setup Assistant to install the third-party software that is required to run the game and compile the game code, engine and asset pipeline, and that is required for Android devices.

To install third-party software from Lumberyard Setup Assistant

1. Unzip the `SetupAssistant.zip` file (located in the `\lumberyard\dev\Bin64` directory) and move the `.APP` into `Bin64`. Run Lumberyard Setup Assistant.
2. In Lumberyard Setup Assistant, on the **Get started** page, select **Run your game project**, **Compile the game code**, and **Compile for Android devices**. Click **Next**.
3. Follow the instructions on the screen to complete the installations for any third-party software or SDKs that you need. Be sure to install the Wwise audio library and JDK v7u79. For more information about using Lumberyard Setup Assistant, see [Using Lumberyard Setup Assistant to Set Up Your Development Environment \(p. 14\)](#).
4. In a command line window, change to the `\lumberyard\dev` directory.
5. To initialize the build system, run the following command:

```
sh lmr_waf.sh configure
```

6. In the Finder, open the `user_settings.options` file (located in the `\lumberyard\dev_WAF_` directory).
7. Edit the `bootstrap_tool_param` as follows:

```
bootstrap_tool_param = --none --enablecapability compilegame --  
enablecapability compileandroid --no-modify-environment
```

8. Modify your environment variables by doing the following:
 - a. If you are using Bash, edit the `.bash_profile` file to add the paths for `android-sdk/platform-tools` and `android-sdk/tools`.
 - b. In a command line window, change to the SDK directory and run the following command:
`tools/android update sdk --no-ui`
9. Locate the directory where you installed the Android SDK. Run the **android** executable file (located in the `tools` directory) and select the version of the SDK that you want to install. You must also install a version of the build tools. Note the version you installed.
10. Modify configuration files to tell Lumberyard which version of the SDK to use when building your game:
 - a. In the File Explorer, locate `_WAF_\android` in the directory where you installed Lumberyard.
 - b. Edit the `android_settings.json` file to set `BUILD_TOOLS_VER` with the version of the build tools that you just installed and to set `SDK_VERSION` with the version of the SDK that you want to use.
 - c. Save the file.

Important

You must save these files with the correct line endings. If you are not using the Vim text editor, please research the correct method to save your files. If you are using Vim, save the file by running the following command: `w ++ff=mac`

11. In a command line window, change to the `\lumberyard\dev` directory.
12. To initialize the build system, run the following command:

```
sh lmr_waf.sh configure
```

Topics

- [Configuring Your Game Project for Android \(p. 885\)](#)
- [Building Game Assets for Android Games \(p. 888\)](#)

- [Building Shaders for Android Games \(p. 890\)](#)
- [Building Android Games \(p. 892\)](#)
- [Android Debugging \(p. 893\)](#)
- [Deploying Android Games \(p. 894\)](#)
- [Running Android Games \(p. 896\)](#)
- [Using Virtual File System with Android \(p. 898\)](#)
- [Using a Samsung Device with Lumberyard \(p. 900\)](#)
- [Using Lumberyard with Android Studio \(p. 900\)](#)

Configuring Your Game Project for Android

Mobile support is in preview release and is subject to change.

Before you use Lumberyard to build your Android games, you must configure your game project to be built for Android. You can also customize the Android settings in your game project to allow for store deployment.

Prerequisites

To configure your game project for Android, you must have the following:

- Lumberyard and the Lumberyard SDK installed
- Your development environment set up
- Basic knowledge of the Lumberyard Waf build system and the JSON data format
- Lumberyard configured to build Android games

For information, see [Android Support \(p. 882\)](#).

- A game project

Setting Your Game Project to Build for Android

You can enable your game project to be built for Android by modifying certain settings in your game project's `project.json` file.

To modify your game project's `project.json` file

1. In a file browser, navigate to your game project's asset directory. For example, `\dev\SamplesProject` in the directory where you installed Lumberyard.
2. Use a text editor to open the `project.json` file.
3. Verify the following entry appears or add the entry if it does not exist: `"android_settings": {}`
4. Save the `project.json` file.
5. In a command line window, navigate to the root of the directory where you installed Lumberyard (`\lumberyard\dev`).
6. Run the `libr_waf configure` command:
 - On a PC, type: `libr_waf.bat configure`
 - On a Mac, type: `./libr_waf.sh configure`
7. Build and test your game project on Android. For information, see [Building Android Games \(p. 892\)](#).

Customizing Android Settings for Your Game Project

After you add the Android configuration entry for your game project, you can customize various settings to generate your project and prepare your Android game for store deployment.

You can customize the following Android settings:

Android package name

Tag name: "package_name"

Type: String in dot-separated format

Example: "com.mycompany.mygame"

Manifest code version number

Tag name: "version_number"

Type: Whole number value

Manifest version name

Tag name: "version_name"

Type: String

Example: "1.0.0"

Orientation

Tag name: "orientation"

Type: String

Valid values: See the [Android Developers](#) page for valid values.

Application icon overrides

Tag name: "icons"

Type: Mapping of strings for each resolution option. All entries require a path relative to `\Code\project\Resources` or an absolute resource path. Include the name of a `.png` image in the string.

Valid values: "mdpi", "hdpi", "xhdpi", "xxhdpi", "xxxhdpi", "default" (the image set for "default" is used if a specific DPI override is not specified)

Application splash screen overrides

Tag name: "splash_screen"

Type: Mapping of two maps

- Landscape tag name: "land"
- Portrait tag name: "port"

Both orientation maps allow the same options. All entries require a path relative to `\Code\project\Resources` or an absolute resource path. Include the name of a `.png` image in the string.

Valid values: "mdpi", "hdpi", "xhdpi", "xxhdpi", "default" (the image set for "default" is used if a specific DPI override is not specified)

Allow assets to pack into the APK

Tag name: "place_assets_in_apk"

Type: Whole number value

Valid values: 0 (No) or 1 (Yes)

To add an Android setting override

1. In a file browser, navigate to your game project's asset directory.
2. Use a text editor to open the `project.json` file.
3. Add any of the customizable settings above to the "android_settings" entry in the `project.json` file.

The following example includes all of the customizable Android settings:

```
"android_settings" :
{
  "package_name"   : "com.lumberyard.samples",
  "version_number" : 1,
  "version_name"   : "1.0.0.0",
  "orientation"   : "landscape",
  "icons"         :
  {
    "default"      : "AndroidLauncher/icon-xhdpi.png",

    "mdpi"         : "AndroidLauncher/icon-mdpi.png",
    "hdpi"         : "AndroidLauncher/icon-hdpi.png",
    "xhdpi"        : "AndroidLauncher/icon-xhdpi.png",
    "xxhdpi"       : "AndroidLauncher/icon-xxhdpi.png",
    "xxxhdpi"     : "AndroidLauncher/icon-xxxhdpi.png"
  },
  "splash_screen" :
  {
    "land" :
    {
      "default" : "AndroidLauncher/splash-xhdpi.png",

      "mdpi"    : "AndroidLauncher/icon-mdpi.png",
      "hdpi"    : "AndroidLauncher/icon-hdpi.png",
      "xhdpi"   : "AndroidLauncher/icon-xhdpi.png",
      "xxhdpi"  : "AndroidLauncher/icon-xxhdpi.png"
    },
    "port" :
    {
      "default" : "AndroidLauncher/icon-xhdpi.png",

      "mdpi"    : "AndroidLauncher/icon-mdpi.png",
      "hdpi"    : "AndroidLauncher/icon-hdpi.png",
      "xhdpi"   : "AndroidLauncher/icon-xhdpi.png",
      "xxhdpi"  : "AndroidLauncher/icon-xxhdpi.png"
    }
  }
  "place_assets_in_apk" : 0
},
```

4. Save the file.
5. In a command line window, navigate to the root of the directory where you installed Lumberyard (`\lumberyard\dev`).
6. Run the `lubr_waf` configure command:
 - On a PC, type: `lubr_waf.bat` configure

- On a Mac, type: `./lmb_r_waf.sh configure`

Building Game Assets for Android Games

Mobile support is in preview release and is subject to change.

When you build an Android game using Lumberyard, you must first build the assets that are included with the game. All built assets are located in the `cache` directory of your Lumberyard installation. For example, when you build the Samples Project, the assets are saved to the `\lumberyard\dev\cache\SamplesProject\es3` directory. The initial build of the Samples Project assets may take up to an hour to process, but incremental changes should process almost instantly.

To build Android game assets on your PC

1. Close all instances of Lumberyard Editor and the Asset Processor.
2. Edit the `bootstrap.cfg` file (located in the `\lumberyard\dev` directory) to set `sys_game_folder` to `SamplesProject` (or the project you want to build). Save the file.
3. Edit the `AssetProcessorPlatformConfig.ini` file (located in the `\lumberyard\dev` directory) to uncomment `es3=enabled`. Save the file.

Note

If the Asset Processor was running when you edited the `AssetProcessorPlatformConfig.ini` file, you must restart the Asset Processor.

4. Open Lumberyard Editor, which automatically launches the Asset Processor to process and build your game assets as you make changes to your game levels in Lumberyard Editor.

Note

You can also launch the Asset Processor (GUI or batch version) from the `\lumberyard\dev\Bin64` directory.

Using Assets in Your Game

You can use assets in your game by copying them to your device manually or by packing them into an `.apk` file. We recommend copying the assets to your device manually for a faster build time during development.

Manually Copying Assets

As part of the build process, Lumberyard can automatically copy assets built by the Asset Processor to your device, or you can manually copy assets from a command line window using Android Debug Bridge (ADB). Game assets should be copied to the `/storage/sdcard0/<Your Game Name>` directory.

For example, to manually copy the Samples Project assets, type the following in a command line window:

```
adb push cache/SamplesProject/es3 /storage/sdcard0/SamplesProject
```

Building Assets into an .Apk File

To build an `.apk` file that includes all of your assets, edit the `project.json` file for your game project and set `place_assets_in_apk` to `1`. This method requires a longer build time than manually copying your assets.

For example, to build an .apk file for the Samples Project assets, edit the `project.json` file (located in the `\lumberyard\dev\SamplesProject` directory) to set `place_assets_in_apk` to 1:

```
"android_settings": {  
  "package_name" : "com.lumberyard.samples",  
  "version_number": 1,  
  "version_name" : "1.0.0.0",  
  "orientation" : "landscape",  
  "place_assets_in_apk" : 1  
},
```

When you generate a build, your computer creates an .apk file that includes an executable and game data. Be sure to run the shader compiler when you run your game for the first time.

Note

If you receive an error indicating the `\dev\Solutions\android\SamplesProject\assets` directory does not exist, you can try running the command from a command line window with Administrator privileges.

Sharing Game Assets Between PCs and Macs

After you build the assets to include with your Android game, you can share the `cache` folder between your PC and Mac. This ensures that changes you make in Lumberyard Editor on your PC are automatically retrieved by OS X.

To set up asset sharing on your PC

1. Navigate to the `\dev` folder in the directory where you installed Lumberyard.
2. Right-click the `cache` folder and click **Properties**.
3. In the **cache Properties** dialog box, on the **Sharing** tab, click **Advanced Sharing**. You must have administrator privileges.
4. In the **Advanced Sharing** dialog box, select **Share this folder**. Click **OK**.
5. (Optional) Click **Permissions** to set permissions for specific users. This step is required if you want to modify the shared assets on your Mac.

To view shared assets on your Mac

1. In the Finder, click **Go, Connect to Server**.
2. For the **Server Address**, type `smb://IP address or DNS name of PC/Cache`
3. Click **Connect**.
4. (Optional) Configure your system preferences to automatically connect to the shared folder when OS X starts:
 - a. Open **System Preferences, Users & Groups, Login Items**.
 - b. In the **Login Items** dialog box, click **+** to add a new login.
 - c. In the **Shared** pane, locate and select your PC. In the right pane, select your shared `cache` folder and click **Add**.
5. In a Terminal window, navigate to the `\dev` folder in the directory where you installed Lumberyard.
6. To create a symbolic link to the shared `cache` folder, type: `sudo ln -s /Volumes/Cache Cache`

If prompted, type the password for your OS X login.

Building Shaders for Android Games

Mobile support is in preview release and is subject to change.

Lumberyard uses a versatile shader system to achieve high quality, realistic graphics. Because the shader compilation pipeline depends on the Windows-specific HLSL optimizer, you must connect to a shader compiler on your PC when running a game on an Android device during the development stage. This compiles the subset of shaders required by your game on demand.

Note

You must connect your PC and Android device to the same network and configure any firewalls to allow traffic through port 61453.

When a new shader is compiled, the game waits for the binary shader permutation to compile on your PC and be sent back to your device. Once this occurs, the shader is cached on your device until you delete the game. When you are ready to release your game, you must pack up and include all cached binary shaders.

Building the Shader Compiler

Building Lumberyard Editor will also build the shader compiler. Otherwise, you can build the shader compiler by changing to the `\lumberyard\dev` directory in a command line window and typing the following:

```
lmb_r_waf.bat build_win_x64_profile -p all --targets=CrySCompileServer
```

The shader compiler executable is created in the `\lumberyard\dev\Tools\CrySCompileServer\x64\profile` directory.

You must also set up the mobile device system CFG file (`system_android_es3.cfg`) to connect to the remote shader compiler on the PC.

Running the Shader Compiler

You can run the shader compiler on your PC.

To run the shader compiler on your PC

1. Edit the `system_android_es3.cfg` file (located in the `\lumberyard\dev` directory) to set the `localhost` for `r_ShaderCompilerServer` to the IP address of the PC on which you will run the shader compiler.
2. Run `CrySCompileServer.exe` (located in the `\lumberyard\dev\Tools\CrySCompileServer\x64\profile` directory).

Generating and Retrieving Shaders

You can generate and retrieve shaders for your Android game.

To generate and retrieve shaders

1. Build, deploy, and run your game on an Android device. For information, see [Building Android Games \(p. 892\)](#)
2. In your game, explore every area in every level to ensure that all shader permutations required for the game are generated. Exit the game when you are finished.

3. Manually copy the shaders off your Android device onto your PC. Shaders should be saved to the `/storage/sdcard0/<Your Game Name>/user/cache/shaders` directory.

For example, to manually copy the Samples Project shaders, type the following in a command line window: `adb pull /storage/sdcard0/SamplesProject/user <Lumberyard root directory>\cache\SamplesProject\es3\user`

Note

If you do not see shaders located in the `Cache\game project name\es3\user\cache\shaders` directory, check the `Cache\game project name\es3\user\shaders\cache` directory. Move the shaders in this directory to the `Cache\game project name\es3\user\cache\shaders` directory.

Building Shader .Pak Files

You can use a command line prompt and batch file to build a .pak file that includes your shaders.

To build a shader .pak file

1. In a command line window, navigate to the `dev` directory of your build and locate the `BuildShaderPak_ES3.bat` file.
2. To use the `BuildShaderPak_ES3.bat` file, provide command line arguments for the following:
 - Shader version (for example, `GLES3` or `GLES3_1`)
 - Shaderlist_<shader version>.txt file (located in the `Cache` folder of the remote shader compiler executable, for example `dev\Tools\CrySCompileServer\x64\profile\Cache\game project name\ShaderList_GLES3_1.txt` in the Lumberyard root directory)
 - Game project name for which to build the shaders

For example, to build the shader PAK file for the Samples Project, type the following in a command line window: `BuildShaderPak_ES3.bat GLES3 Tools\CrySCompileServer\x64\profile\Cache\SamplesProject\ShaderList_GLES3.txt SamplesProject`

You can derive the shader version from the name of the folder for the files retrieved from the device. For example, `GLES3_1` is the shader version for the following directory: `\lumberyard\dev\Cache\game project\es3\user\cache\shaders\cache\GLES3_1`.

Deploying Shader .Pak Files

When the batch file finishes building the shader PAK file for your game project, you will find the following in the `\Build\Platform\Game Project Name\Platform` directory at the root of your Lumberyard installation (`\lumberyard\dev`):

- `ShaderCache.pak` – Contains all compiled shaders that are used only when the shader cannot be found in the current level's shader cache.
- `ShaderCacheStartup.pak` – Contains a subset of compiled shaders that are required for accelerating the startup time of the engine.

To enable your game to run the shaders from the PAK files

1. Do one of the following:
 - a. Rename the `ShaderCache.pak` file to `shaders.pak`. Copy the PAK files to the `cache\game project name\es3\game project name` directory at the root of your Lumberyard

installation (\lumberyard\dev). When you run your game, it will attempt to load the shaders from the `Shaders.pak` file.

- b. Copy the `shaders*.pak` files to the `cache\game project name\es3\game project name` directory at the root of your Lumberyard installation (\lumberyard\dev). Update the `android.cfg` file to set `r_ShadersPreactivate` to 3. This preloads the game with the shaders from the `ShaderCache.pak` file, and the shaders remain in memory until you exit the game.
2. When the shader PAK files are in the correct cache location, you can deploy the assets to the device. The game will use the shaders and will only connect to the remote shader compiler if it cannot find a shader.

Building Android Games

Mobile support is in preview release and is subject to change.

Before you can deploy your game to Android devices, you must ensure the following:

- The shader compiler (located in the `\lumberyard\dev\Tools\CrySCompileServer\x64\profile\CrySCompileServer.exe` directory) is running on your PC. For information, see [Building Shaders for Android Games \(p. 890\)](#).
- You are using Android NDK r11 or above. For information, see [Android Support \(p. 882\)](#).

Building Android Games Using Clang

You can build your game for Android using Clang. Building for the Android (ARM) platform in Visual Studio will build using Clang.

To build your game for Android using Clang (recommended)

1. Ensure you are using Android SDK 21 or later and Android NDK r11 or later. For information, see [Android Support \(p. 882\)](#).
2. In a command line window, navigate to `\dev` in the directory where you installed Lumberyard.
3. Build various targets of your game:
 - To build debug
 - On a PC, type: `lmbr_waf.bat build_android_armv7_clang_debug -p all`
 - On a Mac, type: `sh lmbr_waf.sh build_android_armv7_clang_debug -p all`
 - To build profile
 - On a PC, type: `lmbr_waf.bat build_android_armv7_clang_profile -p all`
 - On a Mac, type: `sh lmbr_waf.sh build_android_armv7_clang_profile -p all`
 - To build release
 - On a PC, type: `lmbr_waf.bat build_android_armv7_clang_release -p all`
 - On a Mac, type: `sh lmbr_waf.sh build_android_armv7_clang_release -p all`
4. Debug your application. For information, see [Android Debugging \(p. 893\)](#).

Building Android Games Using GCC

You can build your game for Android using the GNU Compiler Collection (GCC). GCC may no longer be supported in a future release.

To build your game for Android using the GCC (legacy)

1. In a command line window, navigate to `\dev` in the directory where you installed Lumberyard.
2. Build various targets of your game:
 - To build debug
 - On a PC, type: `libr_waf.bat build_android_armv7_gcc_debug -p all`
 - On a Mac, type: `sh libr_waf.sh build_android_armv7_gcc_debug -p all`
 - To build profile
 - On a PC, type: `libr_waf.bat build_android_armv7_gcc_profile -p all`
 - On a Mac, type: `sh libr_waf.sh build_android_armv7_gcc_profile -p all`
 - To build release
 - On a PC, type: `libr_waf.bat build_android_armv7_gcc_release -p all`
 - On a Mac, type: `sh libr_waf.sh build_android_armv7_gcc_release -p all`
3. Debug your application. For information, see [Android Debugging \(p. 893\)](#).

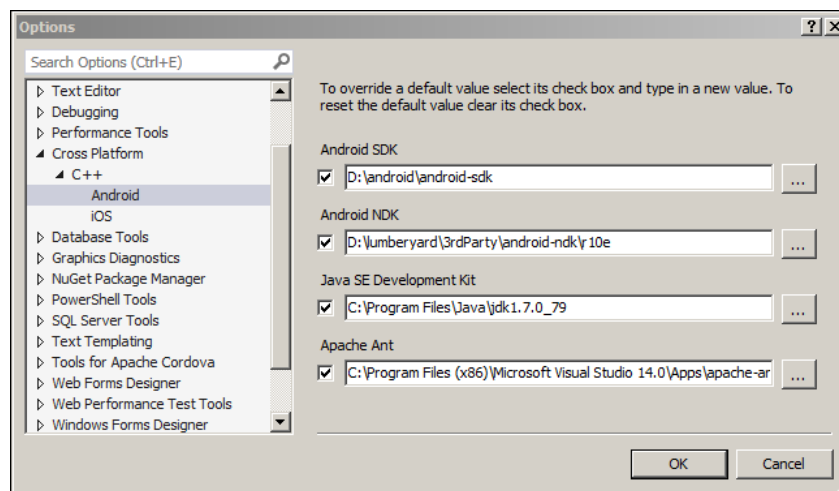
Android Debugging

Mobile support is in preview release and is subject to change.

You can debug your Android game using Visual Studio 2015.

To debug your Android game

1. In the Visual Studio 2015 installer, select **Cross platform tools for C++ development**. Follow the on-screen instructions to complete the installation.
2. In Visual Studio 2015, click **Tools, Options**.
3. In the left pane of the **Options** dialog box, expand **Cross Platform, C++**, and click **Android**.
4. Edit the paths to use the correct directories on your computer:

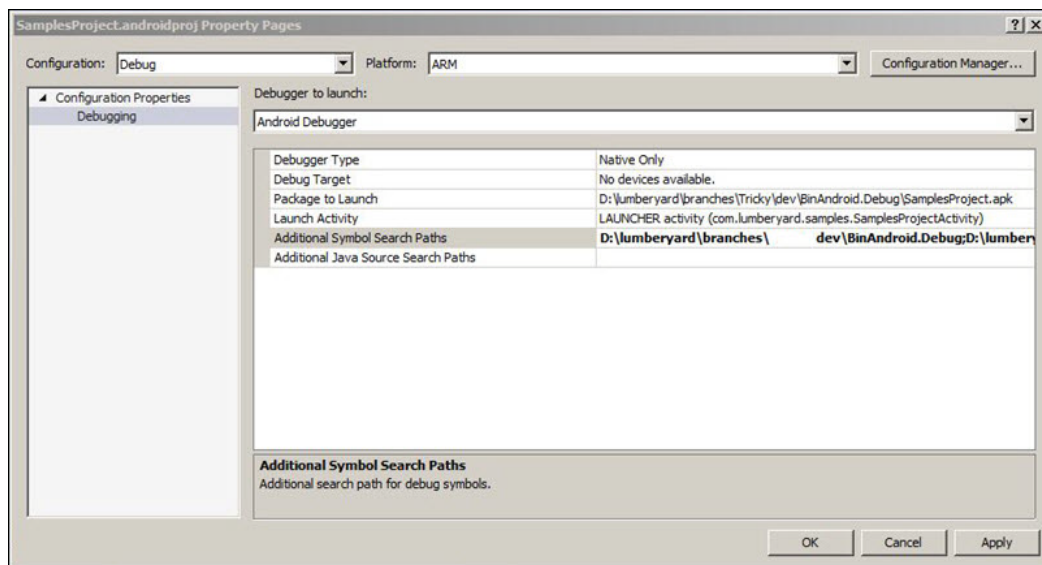


5. Click **OK** and close Visual Studio 2015.
6. Build a debug version of an `.apk` file and then relaunch Visual Studio 2015.

Note

Before you can use Visual Studio 2015 to run and debug the .apk, you must build your game assets for Android and then deploy them to the device or bundle them with the .apk. For information, see [Building Game Assets for Android Games \(p. 888\)](#).

7. In Visual Studio 2015, click **File, Open Project/Solution**. Locate and select your .apk file (BinAndroid.Debug\).
8. In the project window, right-click the project and click **Properties**.
9. Do one of the following:
 - If you are using Visual Studio 2015 without any updates, type **.CryEngineActivity** for **Launch Activity**.
 - If you are using Visual Studio 2015 with Update 1 or later, verify that the correct activity is already set for **Launch Activity**. For the Samples Project, you should see **LAUNCHER activity (com.lumberyard.samples.SamplesProjectActivity)**.
10. For **Additional Symbol Search Paths**, type **dev/Code** and **dev/BinAndroid.Debug**.



11. Open your code files by pressing **Ctrl+O** or clicking **File, Open**.
12. Set breakpoints, if necessary, and then press **F5** to run your game.

Deploying Android Games

Mobile support is in preview release and is subject to change.

Before you can deploy your game to Android devices, you must ensure the shader compiler (located in the `\lumberyard\dev\Tools\CrySCompileServer\x64\profile\CrySCompileServer.exe` directory) is running on your PC. For more information, see [Building Shaders for Android Games \(p. 890\)](#).

You can deploy your game to Android devices using the Resource Compiler.

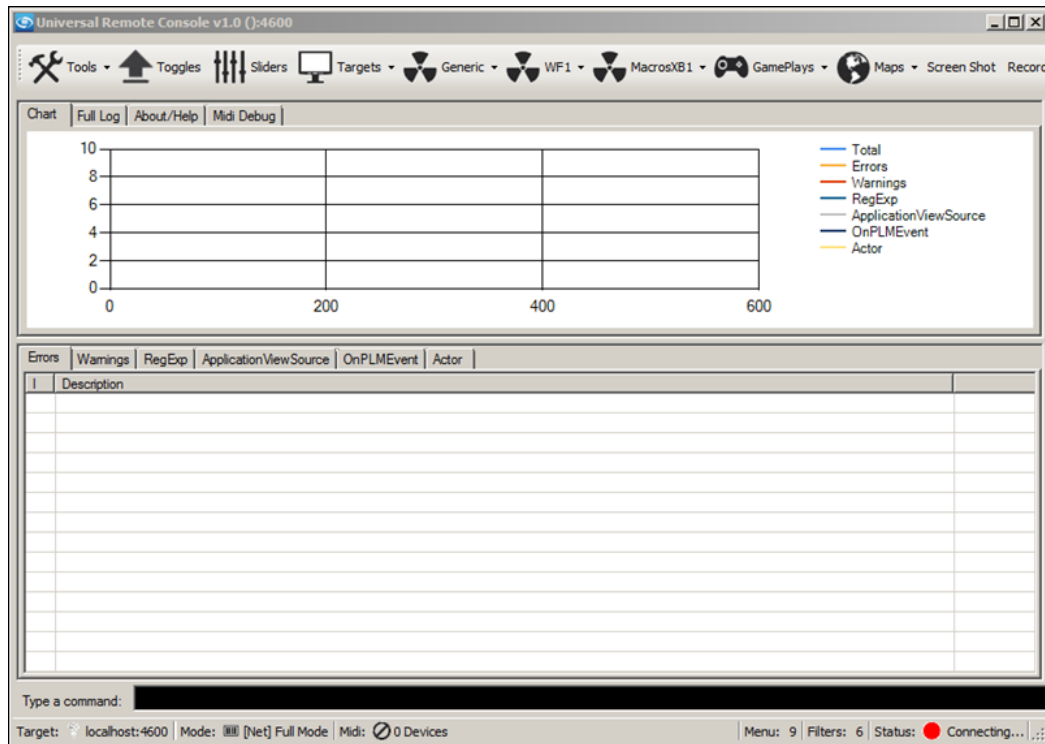
Once you have deployed your game, see [Running Android Games \(p. 896\)](#).

Using the Remote Console to Deploy Your Android Game

You can operate and configure the Lumberyard runtime application using a series of console commands on your PC. You must connect your PC and Android device to the same network and configure any firewalls to allow traffic through port 4600.

To deploy your game using the Remote Console

1. Launch the **Remote Console** application (located in the `\lumberyard\dev\Tools\RemoteConsole\` directory).

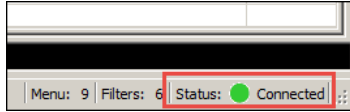


2. On the **Full Log** tab, view the output from the runtime engine's logging system.
3. Start running a Lumberyard application on your Android device.
4. In the Remote Console, click **Targets** and then type the IP address of the Android device for **Custom IP**.
5. (Optional) If your network allows you to assign IP addresses per device so that the IP address is always fixed to a MAC address, you can edit the `params.xml` file (located in the same directory as the application) to add your device to the list of targets:

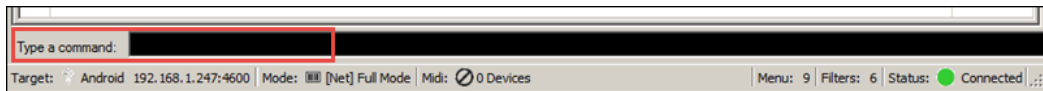
```
<Targets>
  <Target name="PC" ip="localhost" port="4600"/>
  <Target name="Android" ip="192.168.5.247" port="4600"/>
</Targets>
```

Adding your device to the targets allows you to select from a list of devices instead of entering the IP address each time.

6. Verify that you see a green connected status in the Remote Console, which indicates the Remote Console can successfully connect to your Lumberyard application.



7. Issue commands to your application by typing in the text window. The text window supports autocomplete, and commands like `map` will detect the available options. Useful commands include:
 - `cl_DisableHUDText` – Disables the heads-up display text.
 - `g_debug_stats` – Enables debugging for gameplay events.
 - `r_DisplayInfo` – Displays rendering information.
 - `r_ProfileShaders` – Displays profiling information for the shaders.



Running Android Games

Mobile support is in preview release and is subject to change.

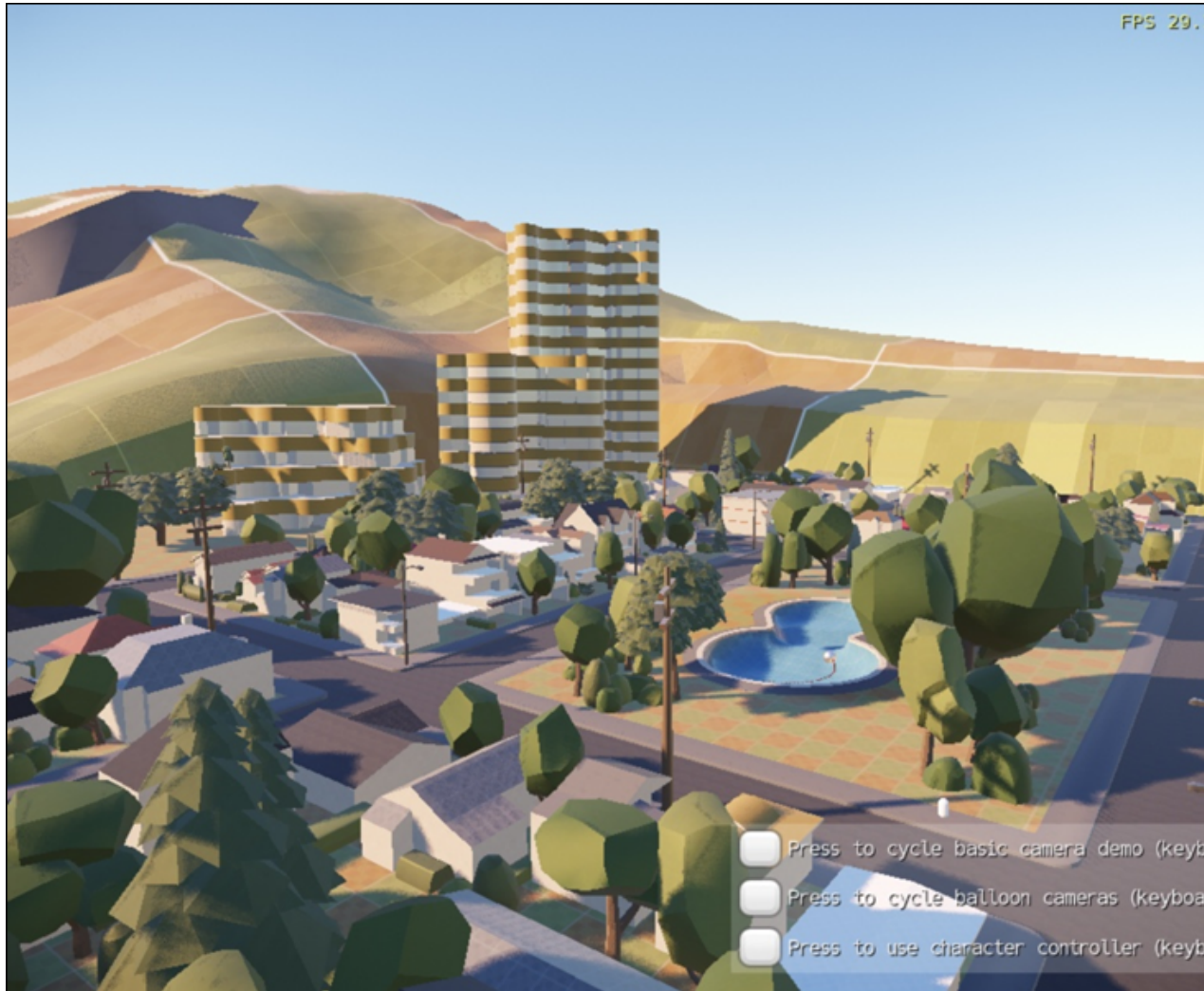
Before you can run your game on Android devices, you must deploy your game using the Remote Console. For information, see [Deploying Android Games \(p. 894\)](#).

To run your game on an Android device

1. Launch your game by tapping the icon on your device's home screen. You can also launch your game from the Visual Studio 2015 debugger.

Note

You can check the Asset Processor to verify a connection from **PC-GAME** with the **es3** platform. Serving files from your PC may impact load time, so it may take time for the game world to appear.



2. (Optional) Load different levels by editing the `SamplesProject\autoexec.cfg` file and running the game again. Android supports the following levels:
 - Animation_Basic_Sample
 - Camera_Sample (default)
3. Use the following controls to navigate around your game:
 - Switch between cameras by selecting the buttons in the lower right corner of the screen.
 - Move the robot in the Character Controller view by touching the left side of the screen.
 - Look around the Character Controller view by touching the right side of the screen.
 - Jump in the Character Controller view by double-tapping anywhere on the screen.



4. (Optional) In a command line window, type `adb logcat` to view logging information for your game.

Using Virtual File System with Android

Mobile support is in preview release and is subject to change.

The **Asset Processor** can use the virtual file system (VFS) to serve files to your Android devices over a USB connection. This method offers the following benefits:

- You can edit game content and data on a PC and view changes on the Android devices.
- You needn't rebuild the Android application package (.apk) file when editing nonvisual data.
- You can iterate much faster.

This topic demonstrates how to set up your PC and Android device to run the Samples Project using VFS.

Before you begin setting up VFS, identify the IP address of the PC running the Asset Processor. You must provide the IP address during setup.

To set up VFS

1. On your PC, edit the `bootstrap.cfg` file (located in the `\lumberyard\dev` directory) to set `remote_filesystem` to `1`. This notifies the runtime to turn on VFS.
2. Tell the runtime to create a connection over USB by setting the following:

```
remote_ip=127.0.0.1
android_connect_to_remote=1
connect_to_remote=1
wait_for_connect=0
```

3. Save your changes and then copy the file to your device using the **Android Debug Bridge** command line window: `adb push bootstrap.cfg /storage/sdcard0/SamplesProject/bootstrap.cfg`
4. (Optional) To send traffic to the shader compiler through VFS, edit the `system_android_es3.cfg` file (located in the `\lumberyard\dev` directory) to add `r_AssetProcessorShaderCompiler=1`.

To enable USB I/O connections to your device

1. Ensure you have built an `.apk` file so that you can run your game with VFS. For instructions, see [Building Game Assets for Android Games \(p. 888\)](#).
2. On your PC, edit the `AssetProcessorPlatformConfig.ini` file (located in the `\lumberyard\dev` directory) to add `es3=enabled` to the `[Platforms]` section. This enables the **Asset Processor** to create data for Android devices.
3. Start the **Asset Processor** (located in the `\lumberyard\dev\Bin64` directory).
4. Install the game on your device by typing the following in an ADB command line window: `adb install -r BinAndroid.Debug\SamplesProject.apk`
5. Tell your Android device to send traffic to the **Asset Processor** by typing the following in an ADB command line window: `adb reverse tcp:45643 tcp:45643`

To run the game

- Launch your game by tapping the icon on your device's home screen. You can also launch your game from the Visual Studio 2015 debugger.

Note

You can check the **Asset Processor** to verify a connection from **PC-GAME** with the **es3** platform. Serving files from your PC can affect load time, so it may take time for the game world to appear.



Using a Samsung Device with Lumberyard

Mobile support is in preview release and is subject to change.

Before you can use a Samsung device to test a Lumberyard game, you must perform additional setup steps for building and debugging:

- Install Visual Studio 2015 Update 1.
- Navigate to the **Property Pages** for your .apk file, and clear the **Deploy** check box in the **Configuration Manager** window.
- If you encounter an error that prevents Visual Studio from executing run-as, search the Internet for ways to address the error, specific to your device.

Using Lumberyard with Android Studio

Mobile support is in preview release and is subject to change.

Android Studio is the integrated development environment (IDE) provided by Google so you can build applications. The IDE includes editing, debugging, and performance tools, as well as a build and deploy system.

Prerequisites

To use Lumberyard with Android Studio, you must have the following:

- Lumberyard and the Lumberyard SDK installed
- Your development environment set up
- Basic knowledge of the Lumberyard Waf build system

- Lumberyard configured to build Android games

For information, see [Android Support \(p. 882\)](#).

- Android Studio 2.1.x installed

For information, see [Android Studio](#).

Note

We highly recommend using the Canary version of Android Studio 2.2 Preview 1+ to support the latest version of the Google Experimental Gradle plugin. For information, see [Android Studio Canary Channel](#).

Topics

- [Creating a Lumberyard Project for Android Studio \(p. 901\)](#)
- [Importing Your Lumberyard Project into Android Studio \(p. 901\)](#)
- [Building and Debugging Your Lumberyard Android Application in Android Studio \(p. 904\)](#)

Creating a Lumberyard Project for Android Studio

Mobile support is in preview release and is subject to change.

You can use a PC or Mac to create your Lumberyard Android Studio project.

To create an Android Studio project

1. In a command line window, navigate to the root of the directory where you installed Lumberyard (`\lumberyard\dev` on PC or `~/lumberyard/dev` on Mac).
2. Run the following command: `lmb_r_waf configure`

Note

The `configure` command automatically generates the Android Studio project. To disable this functionality and manually generate an Android Studio project, edit the `user_settings.options` file (located in the `\lumberyard\dev_WAF_` directory) to change the **generate_android_studio_projects_automatically** option from **True** to **False**. Then run `lmb_r_waf android_studio` or `lmb_r_waf configure android_studio` (if base projects were not created from a previous `configure` command).

3. Verify the Android Studio project was created successfully:

```
[WAF] Executing 'android_studio' in 'C:\Lumberyard\dev\BinTemp'  
[INFO] Created at C:\Lumberyard\dev\Solutions\LumberyardAndroidSDK  
[WAF] 'android_studio' finished successfully (1.261s)
```

Importing Your Lumberyard Project into Android Studio

Mobile support is in preview release and is subject to change.

After you create a Lumberyard project, you can import it into Android Studio.

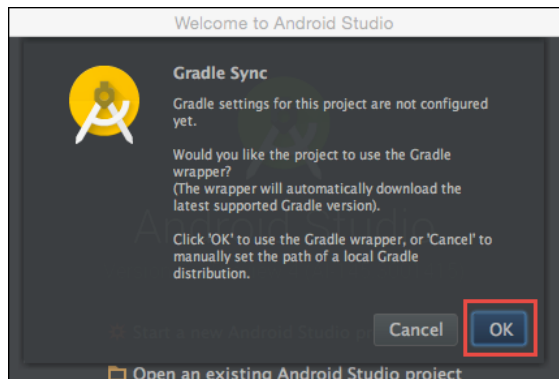
To import your Lumberyard project into Android Studio

1. Open Android Studio.

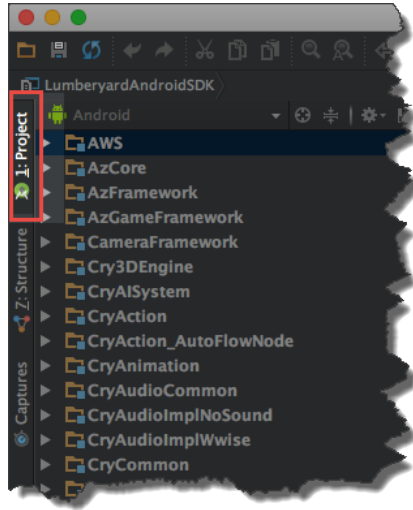
2. On the **Welcome to Android Studio** screen, click **Import project (Eclipse ADT, Gradle, etc.)**.



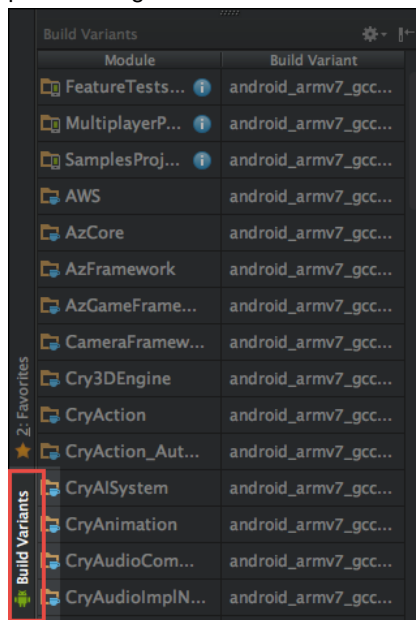
3. Locate the Android Studio project that was created when you ran the `lmbr_waf` configure command. The default location is `\lumberyard\dev\Solutions\LumberyardAndroidSDK`.
4. In the **Gradle Sync** dialog box indicating that Gradle settings are not configured for this project, click **OK**.



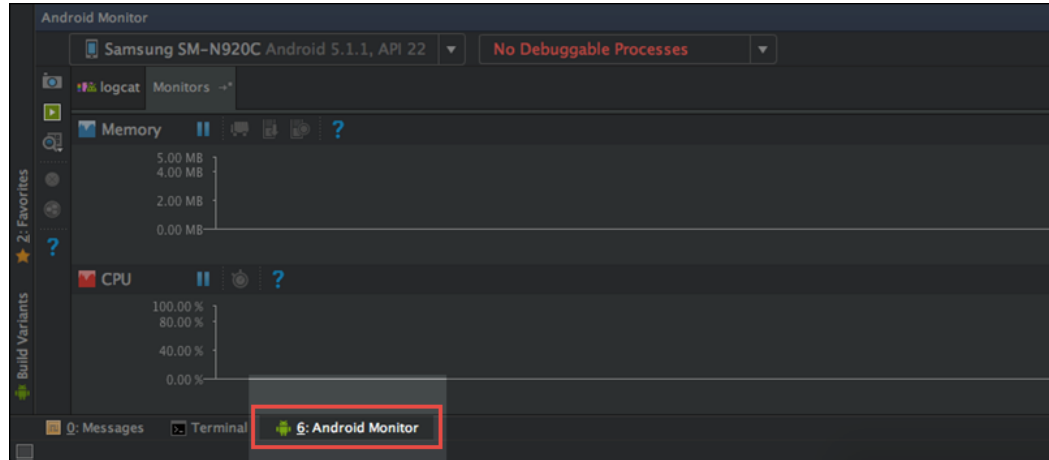
5. When the project has been imported and opens, do the following:
 - Click **Project** to view the project source view pane.



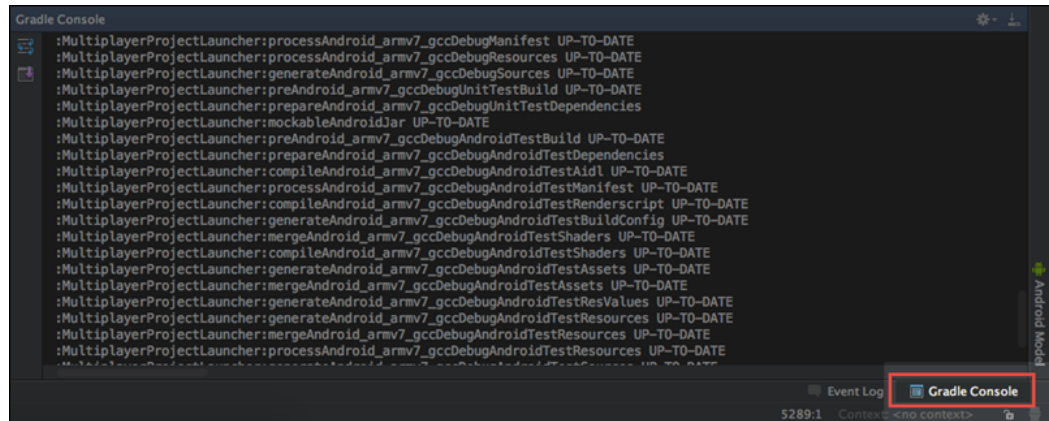
- Click **Build Variants** to view the build targets and change the build configuration for your target project. This impacts the FeatureTestsLauncher, MultiplayerProjectLauncher, and SamplesProjectLauncher only. The build configuration for all other targets during the build process is ignored.



- Click **Android Monitor** to view the logcat and system monitors as well as CPU/GPU, memory, and network usage.



- Click **Gradle Console** to view the build output.



Building and Debugging Your Lumberyard Android Application in Android Studio

Mobile support is in preview release and is subject to change.

After your Lumberyard project is imported, you can build and debug it using Android Studio.

To run and debug your application in Android Studio

1. In Android Studio, select a game project to debug from the target list in the menu bar.

The prepopulated list of targets vary depending on the Android Studio version.

The Stable version might display the following:

- SamplesProjectLauncher
- FeatureTestsLauncher
- MultiplayerProjectLauncher
- FeatureTestsLauncher-native
- MultiplayerProjectLauncher-native
- SamplesProjectLauncher-native

The Canary version might display the following:

- SamplesProjectLauncher
- FeatureTestsLauncher
- MultiplayerProjectLauncher

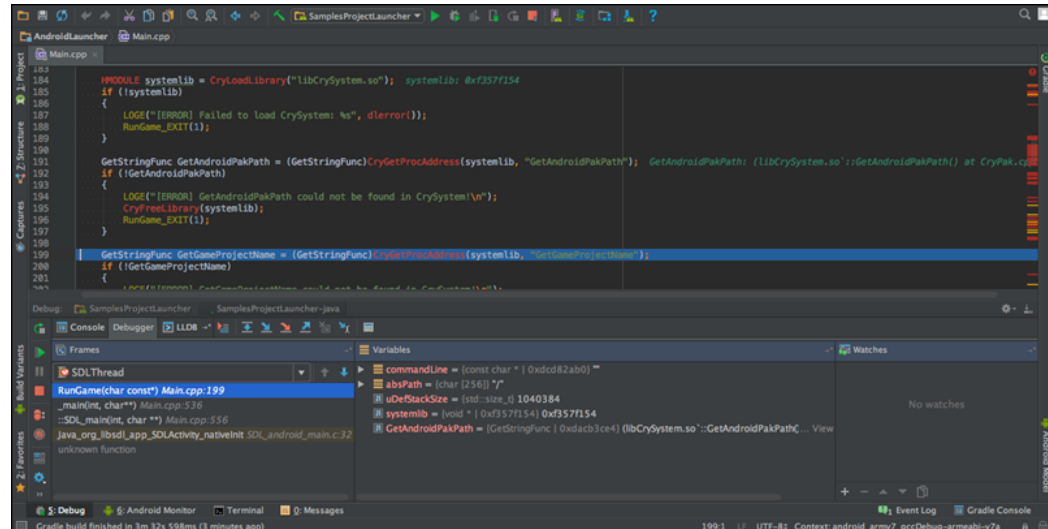
For Stable versions of Android Studio 2.1.x, select the target with the `-native` suffix in order to debug native code.

2. Set your native break points.
3. Connect your Android device to your computer.
4. In Android Studio, click **Debug target**.
5. In the **Select Deployment Target** dialog box, select your device and click **OK**.

Note

Lumberyard does not support Android emulators. If you do not see your physical device, click **Cancel** and run `adb kill-server` in the terminal pane in Android Studio. Then attempt another run/debug build.

6. Android Studio will build the project, launch the application, and connect to the debugger.



7. You can also do the following from the menu bar:
 - Click **Make project** to build all targets in the project, minus the APK packaging step.
 - Click **Select target** to run the selected game project (if multiple game projects are enabled).
 - Click **Run target** to run the selected game project with application-specific monitoring on the **Android Monitor** tab.
 - Click **Debug target** to run the selected game project with the Android Studio debugger attached.

iOS Support

Mobile support is in preview release and is subject to change.

You can use Lumberyard to build your games for iOS devices that use the A8 GPUs, including iPhone 6s, iPhone 6s Plus, iPad Air 2, and iPad Pro. In addition, GMEM and Metal support enables you to use

Lumberyard to create high fidelity visuals by talking directly to the hardware, using the latest rendering techniques, and pushing more data to the GPU.

Lumberyard includes four iOS-supported sample projects that you can use to learn how to build assets for iOS games using the Asset Processor, build shaders using the remote shader compiler, and build and deploy iOS applications using the Lumberyard build tools.

Prerequisites

To build games for iOS, Lumberyard requires the following on your Mac:

- [Xcode 7.1](#) or later
- iOS v9.0 SDK or later
- [Lumberyard Mac Support Files](#)

Note

Lumberyard Editor requires Windows 7 or later to edit levels and build game assets. You must have access to a PC with Lumberyard installed and be able to navigate and run commands from Terminal on your Mac.

Setting Up Your Mac

Download and extract Lumberyard on your Mac using the [Lumberyard OS X Support Files](#) download. This contains all the source code and tools you need to build your iOS game. Then run the Lumberyard Setup Assistant to install the third-party software that is required to run the game and compile the game and engine code for iOS devices.

To run Lumberyard Setup Assistant

1. In a terminal window, navigate to the `Bin64` folder in the directory where you installed Lumberyard.
2. Run Lumberyard Setup Assistant by double-clicking the app in the Finder or by running the `SetupAssistant.app` from the command line.
3. In Lumberyard Setup Assistant, on the **Get started** page, select **Compile for iOS devices**, **Compile the Game Code**, and **Compile the Engine and Asset Pipeline**. Click **Next**.
4. Follow the instructions onscreen to complete the installations for any third-party software or SDKs that you need. For more information about using Lumberyard Setup Assistant, see [Using Lumberyard Setup Assistant to Set Up Your Development Environment \(p. 14\)](#).
5. Open a command line window and navigate to your Lumberyard `dev` directory.
6. To initialize the build system, run the following command: `sh lmbr_waf.sh configure`

Topics

- [Building Game Assets for iOS Games \(p. 907\)](#)
- [Building Shaders for iOS Games \(p. 908\)](#)
- [Building and Deploying iOS Games \(p. 909\)](#)
- [iOS Debugging and Troubleshooting \(p. 912\)](#)
- [Creating iOS Games \(p. 913\)](#)
- [Preparing Lumberyard iOS Games for Distribution \(p. 914\)](#)
- [Using Virtual File System with iOS \(p. 914\)](#)

Building Game Assets for iOS Games

Mobile support is in preview release and is subject to change.

When you build an iOS game using Lumberyard, you must first build the assets that are included with the application. All built assets are located in the `cache` folder of your Lumberyard installation. For example, when you build the Samples Project, the assets are saved to the `\lumberyard\dev\cache\SamplesProject\ios` directory. The initial build of the Samples Project assets may take up to an hour to process, but incremental changes should process almost instantly.

Lumberyard Editor requires Windows 7 or later to edit levels and build game assets. You must have access to a PC with Lumberyard installed.

To build iOS game assets on your PC

1. On your PC, close all instances of Lumberyard Editor and the Asset Processor.
2. Edit the `bootstrap.cfg` file (located in the `\lumberyard\dev` directory) to set `sys_game_folder` to `SamplesProject` (or the project you want to build). Save the file.
3. Edit the `AssetProcessorPlatformConfig.ini` file (located in the `\lumberyard\dev` directory) to uncomment `ios=enabled`. Save the file.
4. Open Lumberyard Editor, which automatically launches the Asset Processor to process and build your game assets as you make changes to your game levels in Lumberyard Editor.

Note

You can also launch the Asset Processor (GUI or batch version) from the `\lumberyard\dev\Bin64` directory.

Sharing Game Assets Between PCs and Macs

After you build the assets to include with your iOS application, you can share the `cache` folder between your PC and Mac. This ensures that changes you make in Lumberyard Editor on your PC are automatically retrieved by OS X.

To set up asset sharing on your PC

1. Navigate to the `\dev` folder in the directory where you installed Lumberyard.
2. Right-click the `cache` folder and click **Properties**.
3. In the **cache Properties** dialog box, on the **Sharing** tab, click **Advanced Sharing**. You must have administrator privileges.
4. In the **Advanced Sharing** dialog box, select **Share this folder**. Click **OK**.
5. (Optional) Click **Permissions** to set permissions for specific users. This step is required if you want to modify the shared assets on your Mac.

To view shared assets on your Mac

1. In the Finder, click **Go, Connect to Server**.
2. For the **Server Address**, type `smb://IP address or DNS name of PC/Cache`
3. Click **Connect**.
4. (Optional) Configure your system preferences to automatically connect to the shared folder when OS X starts:
 - a. Open **System Preferences, Users & Groups, Login Items**.
 - b. In the **Login Items** dialog box, click **+** to add a new login.

- c. In the **Shared** pane, locate and select your PC. In the right pane, select your shared `cache` folder and click **Add**.
5. In a Terminal window, navigate to the `\dev` folder in the directory where you installed Lumberyard.
6. To create a symbolic link to the shared `cache` folder, type: `sudo ln -s /Volumes/Cache Cache`

If prompted, type the password for your OS X login.

Building Shaders for iOS Games

Mobile support is in preview release and is subject to change.

Lumberyard uses a versatile shader system to achieve high quality, realistic graphics. Because the shader compilation pipeline depends on the Windows-specific HLSL optimizer, you must connect to a shader compiler on your PC when running a game on iOS during development. This compiles the subset of shaders required by your game, on demand.

Note

You must connect your PC and iOS device to the same network and configure any firewalls to allow traffic through port 61453.

When a new shader is compiled, the game waits for the binary shader permutation to compile on your PC and be sent back to your device. Once this occurs, the shader is cached on your device until you delete the app. When you are ready to release your game, you must pack up and include all cached binary shaders.

To build the shader compiler (if not already done)

In a command line window, change to the `\lumberyard\dev` directory and type: `lmbr_waf.bat build_win_x64_profile -p all --targets=CrySCompileServer`

You must also set up the mobile device system CFG file (`system_ios_ios.cfg`) to connect to the remote shader compiler on the PC.

To run the shader compiler on your PC

1. Edit the `system_ios_ios.cfg` file (located in the `\lumberyard\dev` directory) to set the **localhost** for `r_ShaderCompilerServer` to the IP address of the PC on which you will run the shader compiler.
2. Launch the Asset Processor if it is not still running.
3. Verify that you are sharing the `cache` folder between your PC and Mac by checking the corresponding cache file (located in the `\lumberyard\dev\cache\SamplesProject\ios\system_ios_ios.cfg` directory).

To generate and retrieve shaders for your iOS game

1. Build, deploy, and run your game on an iOS device. For information, see [Building and Deploying iOS Games \(p. 909\)](#)
2. In your game, explore every area in every level to ensure that all shader permutations required for the game are generated. Exit the game when you are finished.
3. In Xcode, click **Window, Devices**.
4. In the **Devices** window, click the settings wheel and select **Download Container**.
5. In the Finder, locate and right-click the container package for your project (`.xcappdata` file). Select **Show Package Contents**.

6. Copy the `shaders` folder from the `\AppData\Documents` directory on your Mac to the `Cache\game project name\ios\user` directory at the root of your Lumberyard installation (`\lumberyard\dev`) on your PC.

Building Shader .Pak Files

You can use a command line prompt and batch file to build a `.pak` file that includes your shaders.

To build a shader .pak file

1. In a command line window, navigate to the `dev` directory of your build and locate the `BuildShaderPak_Metal.bat` file.
2. To use the `BuildShaderPak_Metal.bat` file, provide a command line argument for the shader type (for example, `gmem128` or `gmem256`) and the game project name for which to build the shaders: `BuildShaderPak_Metal.bat Shader Type Game Project Name`

For example, to build the shaders for the Samples Project, type the following in a command line window: `BuildShaderPak_METAL.bat gmem256 SamplesProject`

You can derive the shader type from the name of the folder for the files retrieved from the device. For example, `gmem256` is the shader type for the following directory: `\lumberyard\dev\Cache\game project\ios\user\cache\shaders\cache\metal\gmem256`.

Deploying Shader .Pak Files

When the batch file finishes building the shader PAK file for your game project, you will find the following in the `\Build\Platform\Game Project Name\Platform` directory at the root of your Lumberyard installation (`\lumberyard\dev`):

- `ShaderCache.pak` – Contains all compiled shaders that are used only when the shader cannot be found in the current level's shader cache.
- `ShaderCacheStartup.pak` – Contains a subset of compiled shaders that are required for accelerating the startup time of the engine.

To enable your game to run the shaders from the PAK files

1. Do one of the following:
 - a. Rename the `ShaderCache.pak` file to `shaders.pak`. Copy the PAK files to the `cache\game project name\ios\game project name` directory at the root of your Lumberyard installation (`\lumberyard\dev`). When you run your game, it will attempt to load the shaders from the `Shaders.pak` file.
 - b. Copy the `shaders*.pak` files to the `cache\game project name\ios\game project name` directory at the root of your Lumberyard installation (`\lumberyard\dev`). Update the `ios.cfg` file to set `r_ShadersPreactivate` to `3`. This preloads the game with the shaders from the `ShaderCache.pak` file, and the shaders remain in memory until you exit the game.
2. When the shader PAK files are in the correct cache location, you can deploy the assets to the device. The game will use the shaders and will only connect to the remote shader compiler if it cannot find a shader.

Building and Deploying iOS Games

Mobile support is in preview release and is subject to change.

Before you can deploy your games to iOS devices, you must ensure the shader compiler (located in the `\lumberyard\dev\Tools\CrySCompileServer\x64\profile\CrySCompileServer.exe` directory) is running on your PC. For more information, see [Building Shaders for iOS Games \(p. 908\)](#).

To build your game for iOS

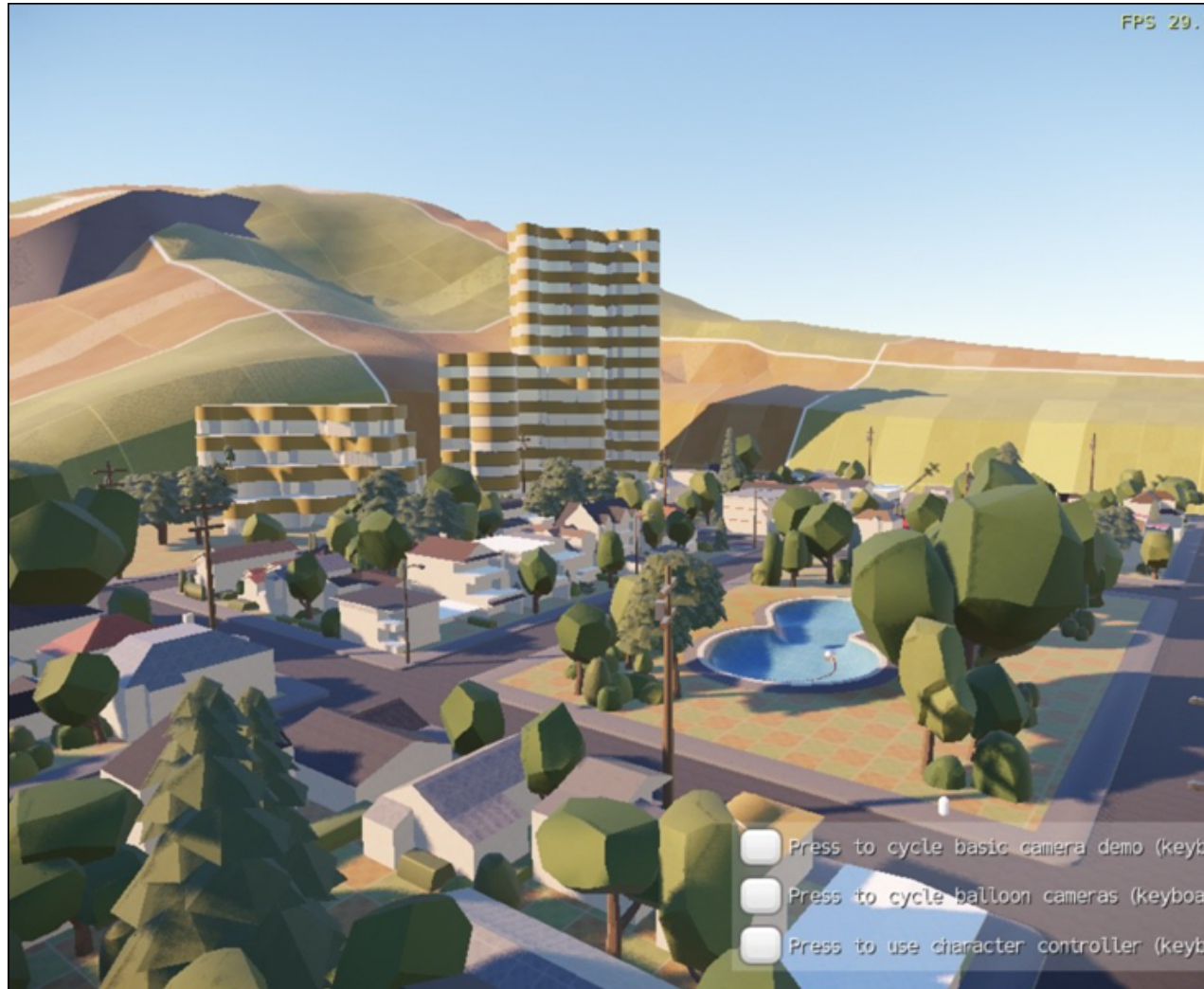
1. In a Terminal window, navigate to the `\dev` folder in the directory where you installed Lumberyard.
2. To generate an Xcode project and prepare the Lumberyard build system to build your iOS app, type: `sh lmbr_waf.sh configure`
3. Build various targets of your game:
 - To build debug, type: `sh lmbr_waf.sh build_ios_debug -p all`
 - To build profile, type: `sh lmbr_waf.sh build_ios_profile -p all`
 - To build release, type: `sh lmbr_waf.sh build_ios_release -p all`
4. Alternatively, build your game with Xcode by using the generated solution located in the `Solutions` folder in the directory where you installed Lumberyard.

To deploy your game to an iOS device

1. Open the Xcode solution that you generated (located in the `Solutions` folder in the directory where you installed Lumberyard).
2. (Recommended) Disable Metal API validation in the Lumberyard Xcode solution by doing the following:
 - a. Navigate to **Product, Scheme, Edit Scheme**.
 - b. On the **Options** tab, select **Disabled** from the drop-down list for **Metal API Validation**.

Unless specifically needed, we recommend disabling Metal API validation.

3. Build, run, and debug your application on an iOS device as you would any Xcode project. For information, see [Launching Your App on Devices](#).



Note

The simulator is not supported. In order to deploy, run, or debug your application, you must use a physical device running iOS 9 or later that is connected through USB to your Mac. You can build without a physical device connected.

4. (Optional) Load different levels by editing the `SamplesProject\autoexec.cfg` file and running the game from Xcode again. iOS supports the following levels:
 - Animation_Basic_Sample
 - Camera_Sample (default)
 - Movers_Sample
 - Trigger_Sample
5. Use the following controls to navigate around your game:
 - Switch between cameras by selecting the buttons in the lower right corner of the screen.
 - Move the robot in the Character Controller view by touching the left side of the screen.
 - Look around the Character Controller view by touching the right side of the screen.
 - Jump in the Character Controller view by double-tapping anywhere on the screen.



iOS Debugging and Troubleshooting

Mobile support is in preview release and is subject to change.

Lumberyard provides full access to the source code, which allows you to debug your iOS application using Xcode without additional Lumberyard-specific steps to follow. For information about debugging and profiling your iOS application, see [Debugging](#) in the official Apple developer documentation.

Unable to see activity in the shader compiler window

You must connect to the shader compiler on your PC in order to compile the subset of shaders required by your game, on demand. To verify that your app has connected correctly and obtained all shaders, you can view the output in the shader compiler window. If you still do not see any activity in the window, please check your setup by following the instructions on the [Building Shaders for iOS Games \(p. 908\)](#) page.

Assets appear out of date on iOS devices

When you make and save changes to your project in Lumberyard Editor, these changes are automatically reflected on your iOS device the next time you deploy. Ensure you have set up your `cache` folder to share between your PC and Mac. If you encounter Xcode errors when deploying to your iOS devices or your assets appear out of date on the iOS devices, you can try cleaning your product from Xcode (click **Product, Clean**), which clears the `.app` package built to `BinIos` or `BinIos.Debug` (debug builds) in the directory where you installed Lumberyard.

Switching projects and enabling iOS assets results in errors

If Lumberyard Editor and/or the Asset Processor are running, you may encounter errors when switching projects by modifying `sys_game_folder` in the `bootstrap.cfg` file or when enabling iOS assets to build by modifying the `AssetProcessorPlatformConfig.ini` file. We recommend that you close all running instances of Lumberyard Editor and the Asset Processor before switching projects or enabling iOS assets using these methods. The Asset Processor continues to run in the background, even after closing, so you can right-click **AssetProcessor_tmp.exe** in Windows Task Manager and click **End Process Tree**.

Cleaning the project does not create a full rebuild of the iOS application

Lumberyard uses a custom build step to generate the final executable and temporary C++ object files, which output to the `\BinTemp\ios_debug` or `\BinTemp\ios_profile` directory where you installed Lumberyard. Unlike a regular Xcode project, in order to create a full rebuild of the iOS application, you must manually delete the contents of the output folder or run one of the following Waf commands from a Terminal window:

- To build debug, type: `libr_waf.sh clean_ios_debug`
- To build profile, type: `libr_waf.sh clean_ios_profile`
- To build release, type: `libr_waf.sh clean_ios_release`

Observed frame rate varies greatly

While running your iOS application, the observable frame rate can vary depending on the build (debug or profile) you are running, whether you are connected to the Xcode debugger, and whether Metal API validation is enabled. To display the frame rate in the upper right corner of the screen, set the `r_DisplayInfo` configuration variable to 1 or higher. When your Xcode project is generated, the default build scheme is set up for debugging. If you want to test or profile your application's speed, we recommend that you [edit your active scheme](#) to run a profile build. Deselect **Debug executable** and disable **Metal API Validation**. Additionally, set the target resolution using the `r_WidthAndHeightAsFractionOfScreenSize` console variable or the `r_width` and `r_height` console variables in the `system_ios_ios.cfg` file. The default value is 1; however, you can lower the target render resolution to help improve performance. If the target render resolution is lower than the default (native device resolution), Lumberyard uses an anti-aliasing algorithm to help maintain the same level of visual quality as the native resolution.

Creating iOS Games

Mobile support is in preview release and is subject to change.

The topics in [iOS Support \(p. 905\)](#) demonstrate how to use the Samples Project that is included with Lumberyard to build game assets, shaders, and iOS applications. You can follow the same instructions to create your own game for iOS devices.

Note

Ensure you have the prerequisites (see [iOS Support \(p. 905\)](#)) and your Mac is properly set up to compile for iOS devices.

To create your iOS game

1. On your PC, use the Project Configurator to create a new project. For information, see [Project Configurator \(p. 985\)](#).
2. Submit the new project into your revision control system and then check out the project onto your Mac.
3. Edit the `user_settings.options` file (located in the `\lumberyard\dev_WAF_` directory) to set `enabled_game_projects` to the name of the project you created:

```
[Game Projects]
enabled_game_projects = MyProject
```

You can simultaneously build multiple projects by separating each project name with a comma:

```
[Game Projects]
enabled_game_projects = SamplesProject,MyProject,OtherProject
```

4. In a command line window, configure and build your project using the instructions on the [Building and Deploying iOS Games \(p. 909\)](#) page.

Note

If you enabled multiple projects, you can switch between multiple targets in your Xcode project.

Preparing Lumberyard iOS Games for Distribution

Mobile support is in preview release and is subject to change.

Once you have finished your Lumberyard iOS game, you can prepare it for store deployment by including the cached binary shaders (for information, see [Building Shaders for iOS Games \(p. 908\)](#)) and editing the `Info.plist` file (located in the `\lumberyard\dev\Code\project_name\Resources\IOSLauncher` directory) to use your project's settings:

- Display name
- App icon
- Splash screen
- Screen orientation
- Other related settings

Note

Ensure the `Info.plist` file is writeable before you make changes to your project settings.

Lumberyard provides default values in the `Info.plist` file as well as default app icons and splash screens in the `Images.xcassets` folder. For more information, see the Lumberyard [Logos and Branding Guidelines](#).

For information about setting these values in the Xcode solution, see [Configuring Your Xcode Project for Distribution](#).

Using Virtual File System with iOS

Mobile support is in preview release and is subject to change.

The **Asset Processor** can use the virtual file system (VFS) to serve files to your iOS devices over a USB connection. This method offers the following benefits:

- You can edit game content and data on a PC and view changes on the iOS devices.
- You needn't rebuild the game when editing nonvisual data.
- You can iterate much faster.

This topic demonstrates how to set up your PC and iOS device to run the Samples Project using VFS.

Prerequisites

Before you can use the VFS with iOS, you must do the following:

- Download the [usbmuxconnect package](#) and save to a location on your Mac.
- Familiarize yourself with command line instructions so you can build the **Asset Processor** application on your Mac.
- Share the cache folder for your game assets between your Mac and PC. For instructions, see [Sharing Game Assets Between PCs and Macs \(p. 907\)](#).

To build the Asset Processor on your Mac

- In a command line window, navigate to the `\lumberyard\dev` directory and type: `lmb_r_waf.sh -p all build_darwin_x64_profile -targets=AssetProcessor`

To set up VFS and connect your game to the Asset Processor

1. On your PC, do the following:
 - a. Edit the `AssetProcessorPlatformConfig.ini` file (located in the `\lumberyard\dev` directory) to enable asset processing for iOS:

```
[Platforms]
pc=enabled
;es3=enabled
ios=enabled
```

- b. Start the **Asset Processor** (located in the `\lumberyard\dev\Bin64` directory) and the shader compiler (located in the `\lumberyard\dev\Tools\CryShaderCompiler\x64\profile` directory).
- c. Edit the `bootstrap.cfg` file (located in the `\lumberyard\dev` directory) to set the following:

```
remote_filesystem=1
ios_connect_to_remote=0
ios_wait_for_connect=1
```

- d. Edit the `system_ios_ios.cfg` file (located in the `\lumberyard\dev` directory) to set `r_AssetProcessorShaderCompiler` to 1.
2. On your Mac, do the following:
 - a. Start the **Asset Processor** and type the IP address and port of the PC that is running the **Asset Processor**. Use the format `IP address:Port`. For example, if your IP address is 10.11.12.13 and you are using the default port, you would type `10.11.12.13:45643`.
 - b. On the **Connection** tab, click **Add Connection** and then select the **Auto Connect** check box.

To run the game

1. On your Mac, build and launch your game for iOS with Xcode. For instructions, see [Building and Deploying iOS Games \(p. 909\)](#). Allow the game to run for a few minutes.
2. In a command line window, navigate to the location where you saved the `usbmuxconnect` package and type: `itnl --iport 22229 --lport 22229`

Note

If the device cannot be reached, stop the game, disconnect and then reconnect the device, and start again from step 1.

3. Verify that you see a connection for the iOS platform in the **Asset Processor** on your PC.

Design Considerations for Creating Mobile Games Using Lumberyard

Mobile support is in preview release and is subject to change.

Lumberyard is a cross-platform game engine, which allows you to develop your game with less concern about the release platform(s). However, some mobile development considerations are discussed below, including game logic, input, and application lifecycle.

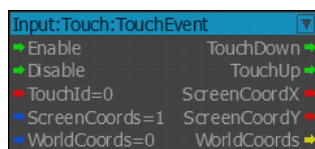
Input

You may need to consider the various physical input devices when you design your game. Lumberyard provides support for the following input devices for iOS and Android:

- Touch screens
- Motion sensors

Touch

You can use the **TouchEvent** node (located under **Input, Touch**) in the Flow Graph Editor to script touch-specific input.



You can also script touch input using more advanced flow nodes:

- MultiTouchCoords – Outputs touch events from the specified ID (finger)
- MultiTouchEvent – Returns touch location information.
- TouchRayCast – Generates a ray cast for every frame.
- VirtualThumbstick – Implements a virtual thumbstick.

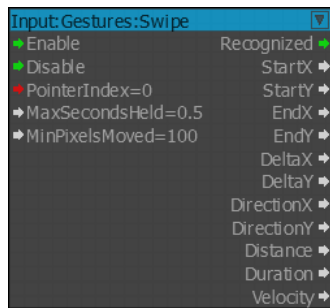
For more information about using flow graph nodes, see [Flow Graph System \(p. 487\)](#).

If you have created your game logic to use mouse-based input, Lumberyard provides a way to emulate mouse events using the primary touch on mobile devices. To enable the ability to emulate mouse events, set `s_SimulateMouseEventsWithPrimaryTouch` to `1`. To support multi-touch input logic and prevent emulated mouse events from being generated alongside touch events, set `s_SimulateMouseEventsWithPrimaryTouch` to `0`.

Gestures

Lumberyard provides a Gestures Gem (in the Project Configurator) that allows you to script input in the Flow Graph Editor using flow nodes (located under **Input, Gestures**) that detect common gesture-based input actions, including:

- Tap (or click, single-touch)
- Drag (or pan, single-touch)
- Hold (or press, single-touch)
- Swipe (single-touch)
- Pinch (multi-touch)
- Rotate (multi-touch)

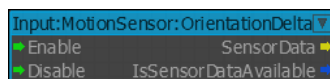


Gestures that require only a single touch to be recognized (Tap, Drag, Hold, and Swipe) function the same when using mouse input on PC. Multi-touch gestures (Pinch and Rotate) can only be recognized through multiple, simultaneous touches.

Motion Sensors

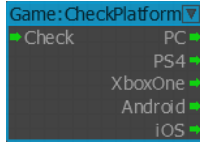
You can use a range of MotionSensor nodes in the Flow Graph Editor to return motion sensor data generated by mobile devices from the accelerometer, gyroscope, and magnetometer. Each flow node returns a vector (or quaternion for orientation) for the device's:

- Acceleration – Raw, user-generated, or gravity
- Rotation – Raw or unbiased
- Magnetic Field – Raw, unbiased, or magnetic north
- Orientation – Absolute or difference from the previous reading



Game Logic

You can use the `CheckPlatform` node in the Flow Graph Editor to modify your game logic by branching your logic based on the current platform.



You can also use the `AZ_PLATFORM_* #defines` in C++ to explicitly include or exclude code for compilation based on specific platforms. Or you can include entire files for compilation for a specific platform by listing the files in a separate `.waf_files` file.

For example, `Code\Framework\AzFramework\AzFramework\API\ApplicationAPI_ios.h` is only listed in `Code\Framework\AzFramework\AzFramework\azframework_ios.waf_files`, which is referenced exclusively for iOS in:

```
Code\Framework\AzFramework\AzFramework\wscript
ios_file_list      = ['azframework_ios.waf_files'],
```

Application Lifecycle

Lumberyard provides a Process Life Management Gem (in the Project Configurator) that shows how you can respond to various application lifecycle events in order to pause your game, display a modal splash screen, and any other actions that need to occur if your application loses focus. You can access system-specific events in C++ by connecting to the appropriate EBus; however, Lumberyard also generates platform-agnostic events that you can use for all supported platforms.

Lumberyard Application Lifecycle Events	iOS	Android
OnApplicationConstrained	<code>applicationWillResignActive</code>	<code>onPause()</code>
OnApplicationUnconstrained	<code>applicationDidBecomeActive</code>	<code>onResume()</code>
OnApplicationSuspended	<code>applicationDidEnterBackground</code>	<code>onPause()</code>
OnApplicationResumed	<code>applicationWillEnterForeground</code>	<code>onResume()</code>
OnMobileApplicationWillTerminate	<code>applicationWillTerminate</code>	<code>onDestroy()</code>
OnMobileApplicationLowMemoryWarning	<code>applicationDidReceiveMemoryWarning</code>	<code>onLowMemory()</code>

To receive process lifecycle events in your game

1. Derive your class from `AzFramework::ApplicationLifecycleEvents::Bus::Handler` (or `AzFramework::[Ios|Android|Windows]LifecycleEvents::Bus::Handler` for platform-specific events).
2. Override the functions corresponding to the events you wish to override:

```
void OnApplicationConstrained(Event /*lastEvent*/) override;
void OnApplicationUnconstrained(Event /*lastEvent*/) override;

void OnApplicationSuspended(Event /*lastEvent*/) override;
void OnApplicationResumed(Event /*lastEvent*/) override
```

3. Connect to the event bus when you want to start listening for events (be sure to also disconnect when you no longer wish to receive them):


```
ApplicationLifecycleEvents::Bus::Handler::BusConnect();  
...  
ApplicationLifecycleEvents::Bus::Handler::BusDisconnect();
```

For a complete example of how to subscribe and respond to application events, see the `Gems\ProcessLifeManagement\Code\Source\ProcessLifeManagementGem.h\` .cpp directory.

Adding IP Addresses to Allow Access to the Asset Processor and Remote Console

Mobile support is in preview release and is subject to change.

The Asset Processor is a networked application that Lumberyard uses to build source assets into game engine ready assets. To ensure your external device can connect to the Asset Processor, you must add the IP address of the external device (Android or iOS) to the **white_list** in the `bootstrap.cfg` file (located in the `\lumberyard\dev` directory).

The Universal Remote Console is a networked application that Lumberyard uses to send commands and view output from the running game engine. To ensure remote console access to a running game instance on your external device, you must add the IP address of the computer that will run the remote console to the **log_RemoteConsoleAllowedAddresses** list in the appropriate configuration file (located in the `\lumberyard\dev` directory):

- Android – `system_android_es3.cfg`
- iOS – `system_ios_ios.cfg`

You must update the configuration file to include the allowed IP addresses before you deploy your game to the external device.

OS X Support

OS X support is in preview release and is subject to change.

You can use Lumberyard to build OS X applications. Lumberyard includes four OS X-supported sample projects that you can use to learn how to build assets for OS X games using the Asset Processor, build shaders using the remote shader compiler, and build and deploy OS X applications using the Lumberyard build tools.

Prerequisites

To build games for iOS, Lumberyard requires the following on your Mac:

- [Xcode 7.1](#) or later
- OS X Yosemite or OS X El Capitan

Note

Lumberyard Editor requires Windows 7 or later to edit levels and build game assets. You must have access to a PC with Lumberyard installed and be able to navigate and run commands from Terminal on your Mac.

Setting Up Your Mac

After you download and extract Lumberyard on your Mac, you must run Lumberyard Setup Assistant to install the third-party software that is required to run the game and compile the game code, engine, and asset pipeline.

To run Lumberyard Setup Assistant

1. Open the directory where you extracted Lumberyard and navigate to the `Bin64` directory. Run `SetupAssistant.app`.
2. Verify that the engine root path is correct.
3. On the **Get started** page, select the following and then click **Next**:
 - **Run your game project**

- **Compile the engine and asset pipeline**

Note

The resource compiler and other asset pipeline tools will not compile because they are not currently supported on OS X.

- **Compile the game code**

4. Follow the instructions onscreen to complete the installations for any third-party software or SDKs that you need. For more information about using Lumberyard Setup Assistant, see [Using Lumberyard Setup Assistant to Set Up Your Development Environment \(p. 14\)](#).
5. Open a command line window and navigate to your Lumberyard `dev` directory.
6. To initialize the build system, run the following command: `sh lmbr_waf.sh configure`
7. In the Finder, open the `user_settings.options` file (located in the `\lumberyard\dev_WAF_\` directory).
8. Verify that **enabled_game_projects** is set to your game project. For example, you can set this option to `SamplesProject`. If **enabled_game_projects** is not set correctly, edit and save the `user_settings.options` file and then run the `configure` command (`sh lmbr_waf.sh configure`) again.

Topics

- [Building Game Assets for OS X Games \(p. 921\)](#)
- [Building Shaders for OS X Games \(p. 922\)](#)
- [Building and Deploying OS X Games \(p. 923\)](#)
- [OS X Debugging and Troubleshooting \(p. 925\)](#)
- [Creating OS X Games \(p. 926\)](#)

Building Game Assets for OS X Games

OS X support is in preview release and is subject to change.

When you build an OS X game using Lumberyard, you must first build the assets that are included with the application. All built assets are located in the `cache` folder of your Lumberyard installation. For example, when you build the Samples Project, the assets are saved to the `\lumberyard\dev\cache\SamplesProject\osx_gl` directory. The initial build of the Samples Project assets may take up to an hour to process, but incremental changes should process almost instantly.

Lumberyard Editor requires Windows 7 or later to edit levels and build game assets. You must have access to a PC with Lumberyard installed.

To build OS X game assets on your PC

1. On your PC, close all instances of Lumberyard Editor and the Asset Processor.
2. Edit the `bootstrap.cfg` file (located in the `\lumberyard\dev` directory) to set `sys_game_folder` to `SamplesProject` (or the project you want to build). Save the file.
3. Edit the `AssetProcessorPlatformConfig.ini` file (located in the `\lumberyard\dev` directory) to uncomment `osx_gl=enabled`. Save the file.
4. Open Lumberyard Editor, which automatically launches the Asset Processor to process and build your game assets as you make changes to your game levels in Lumberyard Editor.

Note

You can also launch the Asset Processor (GUI or batch version) from the `\lumberyard\dev\Bin64` directory.

Sharing Game Assets Between PCs and Macs

After you build the assets to include with your OS X application, you can share the `cache` folder between your PC and Mac. This ensures that changes you make in Lumberyard Editor on your PC are automatically retrieved by OS X.

To set up asset sharing on your PC

1. Navigate to the `\dev` folder in the directory where you installed Lumberyard.
2. Right-click the `cache` folder and click **Properties**.
3. In the **Cache Properties** dialog box, on the **Sharing** tab, click **Advanced Sharing**. You must have administrator privileges.
4. In the **Advanced Sharing** dialog box, select **Share this folder**. Click **OK**.
5. (Optional) Click **Permissions** to set permissions for specific users. This step is required if you want to modify the shared assets on your Mac.

To view shared assets on your Mac

1. In the Finder, click **Go, Connect to Server**.
2. For the **Server Address**, type `smb://IP address or DNS name of PC/Cache`
3. Click **Connect**.
4. (Optional) Configure your system preferences to automatically connect to the shared folder when OS X starts:
 - a. Open **System Preferences, Users & Groups, Login Items**.
 - b. In the **Login Items** dialog box, click **+** to add a new login.
 - c. In the **Shared** pane, locate and select your PC. In the right pane, select your shared `cache` folder and click **Add**.
5. In a Terminal window, navigate to the `\dev` folder in the directory where you installed Lumberyard.
6. To create a symbolic link to the shared `cache` folder, type: `ln -s /Volumes/Cache Cache`

If prompted, type the password for your OS X login.

Building Shaders for OS X Games

OS X support is in preview release and is subject to change.

Lumberyard uses a versatile shader system to achieve high quality, realistic graphics. Because the shader compilation pipeline depends on the Windows-specific HLSL optimizer, you must connect to a shader compiler on your PC when running a game on OS X during development. This compiles the subset of shaders required by your game, on demand.

Note

You must connect your PC and OS X computer to the same network and configure any firewalls to allow traffic through port 61453.

When a new shader is compiled, the game waits for the binary shader permutation to compile on your PC and be sent back to your OS X computer. Once this occurs, the shader is cached locally. When you are ready to release your game, you must pack up and include all cached binary shaders.

To build the shader compiler (if not already done)

On your PC, in a command line window, change to the `\lumberyard\dev` directory and type:
`libr_waf.bat build_win_x64_profile -p all --targets=CrySCompileServer`

To run the shader compiler on your PC

1. Edit the `system_osx_pc.cfg` file (located in the root directory of your Lumberyard installation, `\lumberyard\dev`) to set the **localhost** for `r_ShaderCompilerServer` to the IP address of the PC on which you will run the shader compiler.
2. Launch the Asset Processor if it is not still running.
3. Verify that you are sharing the `cache` folder between your PC and Mac by checking the corresponding cache file (located in the `\lumberyard\dev\cache\SamplesProject\ios\system_osx_pc.cfg` directory).

Building and Deploying OS X Games

OS X support is in preview release and is subject to change.

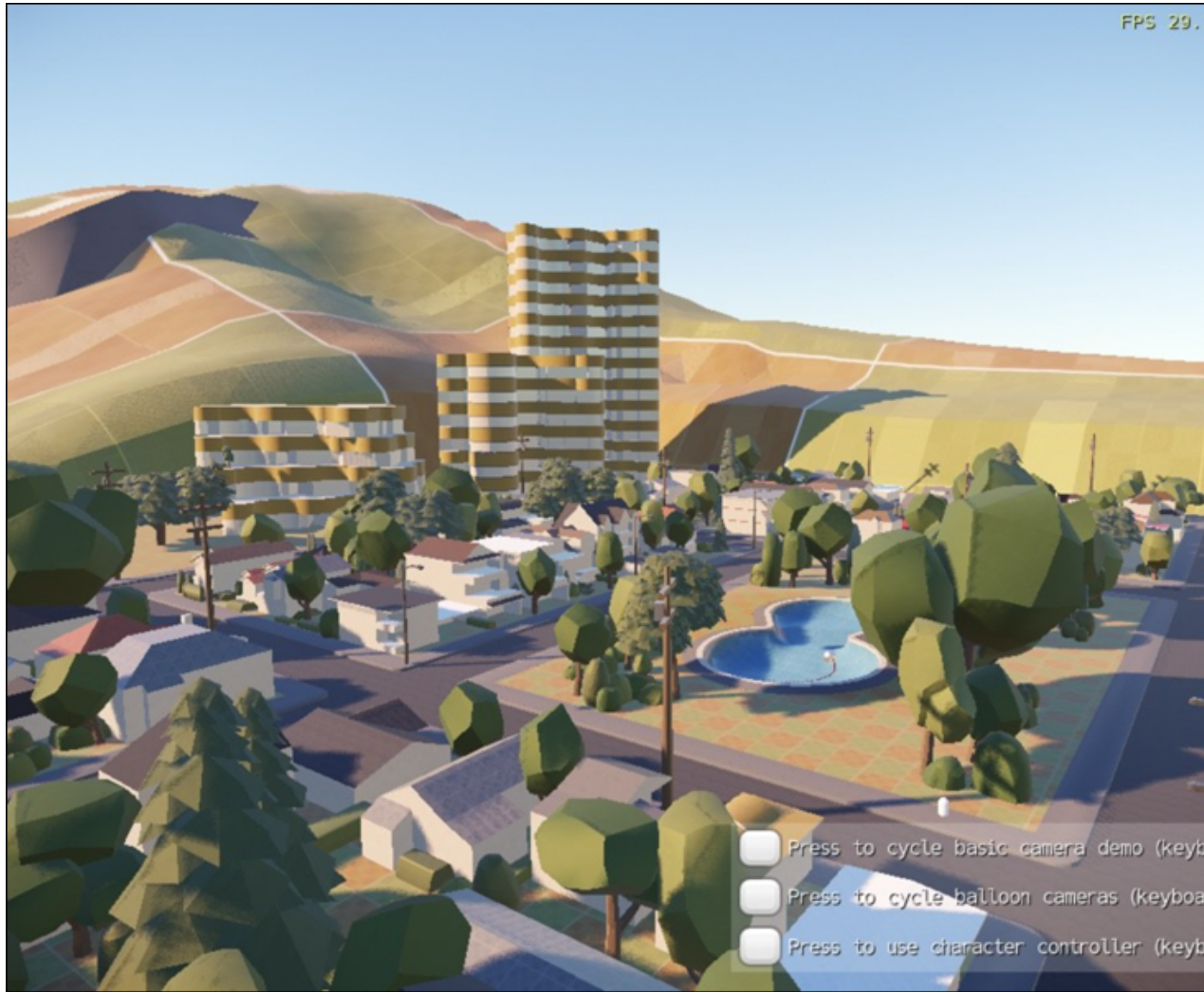
Before you can deploy your games to OS X computers, you must ensure the shader compiler (located in the `\lumberyard\dev\Tools\CrySCompileServer\x64\profile\CrySCompileServer.exe` directory) is running on your PC. For more information, see [Building Shaders for OS X Games](#) (p. 922).

To build your game for OS X

1. On your Mac, in a Terminal window, navigate to the root directory of your Lumberyard installation (`\lumberyard\dev`).
2. To generate an Xcode project and prepare the Lumberyard build system to build your app, type:
`sh libr_waf.sh configure xcode_mac`
3. Build various targets of your game:
 - To build debug, type: `sh libr_waf.sh build_darwin_x64_debug -p all`
 - To build profile, type: `sh libr_waf.sh build_darwin_x64_profile -p all`
 - To build release, type: `sh libr_waf.sh build_darwin_x64_release -p all`
4. Alternatively, build your game with Xcode by using the generated solution located in the `Solutions` folder in the directory where you installed Lumberyard.

To deploy your game to an OS X computer

1. Open the Xcode solution that you generated (located in the `Solutions` folder in the directory where you installed Lumberyard).
2. Build, run, and debug your application as you would any Xcode project. For information, see [Launching Your App on Devices](#).



3. (Optional) Load different levels by editing the `SamplesProject\autoexec.cfg` file and running the game from Xcode again. OS X supports the following levels:
 - Animation_Basic_Sample
 - Camera_Sample (default)
 - Movers_Sample
 - Trigger_Sample
4. Use the following controls to navigate around your game:
 - Switch between cameras by selecting the buttons in the lower right corner of the screen.
 - Move the robot in the Character Controller view by using the mouse or keyboard (**WASD**).
 - Jump in the Character Controller view by pressing the **Space** key.



OS X Debugging and Troubleshooting

OS X support is in preview release and is subject to change.

Lumberyard provides full access to the source code, which allows you to debug your OS X application using Xcode without additional Lumberyard-specific steps to follow. For information about debugging and profiling your OS X application, see [Debugging](#) in the official Apple developer documentation.

Unable to see activity in the shader compiler window

You must connect to the shader compiler on your PC in order to compile the subset of shaders required by your game, on demand. To verify that your app has connected correctly and obtained all shaders, you can view the output in the shader compiler window. If you still do not see any activity in the window, please check your setup by following the instructions on the [Building Shaders for OS X Games \(p. 922\)](#) page.

Switching projects and enabling OS X assets results in errors

If Lumberyard Editor and/or the Asset Processor are running, you may encounter errors when switching projects by modifying `sys_game_folder` in the `bootstrap.cfg` file or when enabling OS X assets to build by modifying the `AssetProcessorPlatformConfig.ini` file. We recommend that you close all running instances of Lumberyard Editor and the Asset Processor before switching projects or enabling OS X assets using these methods. The Asset Processor continues to run in the background, even after closing, so you can right-click **AssetProcessor_tmp.exe** in Windows Task Manager and click **End Process Tree**.

Cleaning the project does not create a full rebuild of the OS X application

Lumberyard uses a custom build step to generate the final executable and temporary C++ object files, which output to the `\BinTemp\darwin_x64_debug` or `\BinTemp\darwin_x64_profile` directory where you installed Lumberyard. Unlike a regular Xcode project, in order to create a full rebuild of the OS X application, you must manually delete the contents of the output folder or run one of the following Waf commands from a Terminal window:

- To build debug, type: `lubr_waf.sh clean_darwin_x64_debug`
- To build profile, type: `lubr_waf.sh clean_darwin_x64_profile`
- To build release, type: `lubr_waf.sh clean_darwin_x64_release`

Observed frame rate varies greatly

While running your application, the observable frame rate can vary depending on the build (debug or profile) you are running and whether you are connected to the Xcode debugger. To display the frame rate in the upper right corner of the screen, set the `r_DisplayInfo` configuration variable to 1 or higher. When your Xcode project is generated, the default build scheme is set up for debugging. If you want to test or profile your application's speed, we recommend that you [edit your active scheme](#) to run a profile build. Deselect **Debug executable**.

Creating OS X Games

OS X support is in preview release and is subject to change.

The topics in [OS X Support \(p. 920\)](#) demonstrate how to use the Samples Project that is included with Lumberyard to build game assets, shaders, and OS X applications. You can follow the same instructions to create your own game for OS X computers.

Note

Ensure you have the prerequisites (see [OS X Support \(p. 920\)](#)) and your Mac is properly set up to compile for OS X computers.

To create your OS X game

1. On your PC, use the Project Configurator to create a new project. For information, see [Project Configurator \(p. 985\)](#).
2. Submit the new project into your revision control system and then check out the project onto your Mac.
3. Edit the `user_settings.options` file (located in the `\lumberyard\dev_WAF_` directory) to set `enabled_game_projects` to the name of the project you created:

```
[Game Projects]
enabled_game_projects = MyProject
```

You can simultaneously build multiple projects by separating each project name with a comma:


```
[Game Projects]  
enabled_game_projects = SamplesProject,MyProject,OtherProject
```

4. In a command line window, configure and build your project using the instructions on the [Building and Deploying OS X Games \(p. 923\)](#) page.

Note

If you enabled multiple projects, you can switch between multiple targets in your Xcode project.

Particle Effects System

Particle Editor is in preview release and is subject to change.

Lumberyard includes an advanced particle effects system that you can use to simulate explosions, fire, smoke, sparks, water spray, fog, snow, rain and other effects. The Particle Editor is the main tool that you use to create and manage particles in your game.

You can place emitters in your level, link them to an object, setup a material to define a custom effect, and control effects using Flow Graph and Track View Editor.

Lumberyard uses two shaders for rendering particles:

- [Particles Shader \(p. 1019\)](#) - Use to render particle effects that are affected by light. These effects can cast shadows and cause reflections.
- [ParticleImposter Shader \(p. 1019\)](#) - Use to create particle effects that are not affected by light. These effects do not cast shadows or cause reflections.

You place particle effects emitters in a scene and then you link to an asset or control them using Flow Graph or Track View Editor.

Topics

- [Particles Best Practices \(p. 929\)](#)
- [Using the Particle Editor \(p. 929\)](#)
- [Using the Gradient Editor \(p. 932\)](#)
- [Using Particle Editor Shortcut Keys \(p. 933\)](#)
- [Managing Particle Libraries \(p. 934\)](#)
- [Creating Custom Attribute Panels \(p. 937\)](#)
- [Particle Trails \(p. 938\)](#)
- [GPU Particles \(p. 940\)](#)
- [Particle Level Of Detail \(LOD\) \(p. 947\)](#)
- [Managing Emitters \(p. 950\)](#)
- [Advanced Particle Techniques \(p. 952\)](#)
- [Particle Entity Parameters and Properties \(p. 954\)](#)

- [Particle Attributes and Parameters Reference \(p. 956\)](#)
- [Particle Debugging \(p. 980\)](#)

Particles Best Practices

Particle Editor is in preview release and is subject to change.

The total number of particles in a scene is actually not a critical factor when considering best practices for working with particles. Total fill-rate, physics, and to some extent spawn rate are more important. Following are some best practices for working with particles:

- Use soft particles only on sub-emitters that are near the ground, and have only small particles. Create similar sub-emitters higher up that emit particles that never intersect the ground, and don't need soft particles.
- Use low-resolution textures (if sharp details are not required) and use texture compression.
- Use an alpha texture with high average opacity rather than additive blending.
- Each second-generation effect causes an emitter to be created for each particle in the parent effect. This can be somewhat expensive, so use sparingly.
- Use physicalized particles sparingly because they are expensive. You can split an effect into subeffects, so that only a few large particles have physics enabled for appearance, and the rest just go through the ground or fade out quickly.
- Instead of multiple overlaid sprites for chaotic glow effects, use just two particles at a time. Carefully tune the lifetime, rotation rate, and set curves for Alpha, Color, Size, so that they combine in chaotic ways. Or, just increase the emissive lighting parameter.
- For large full-screen particles, use a **Fill Rate Cost** value of 1 or above.
- For small particles, such as sparks, set a maximum distance value to ensure they aren't rendered as very small, single pixel particles. Turn off small particles used in collisions for the lowest **Config spec** setting.

Using the Particle Editor

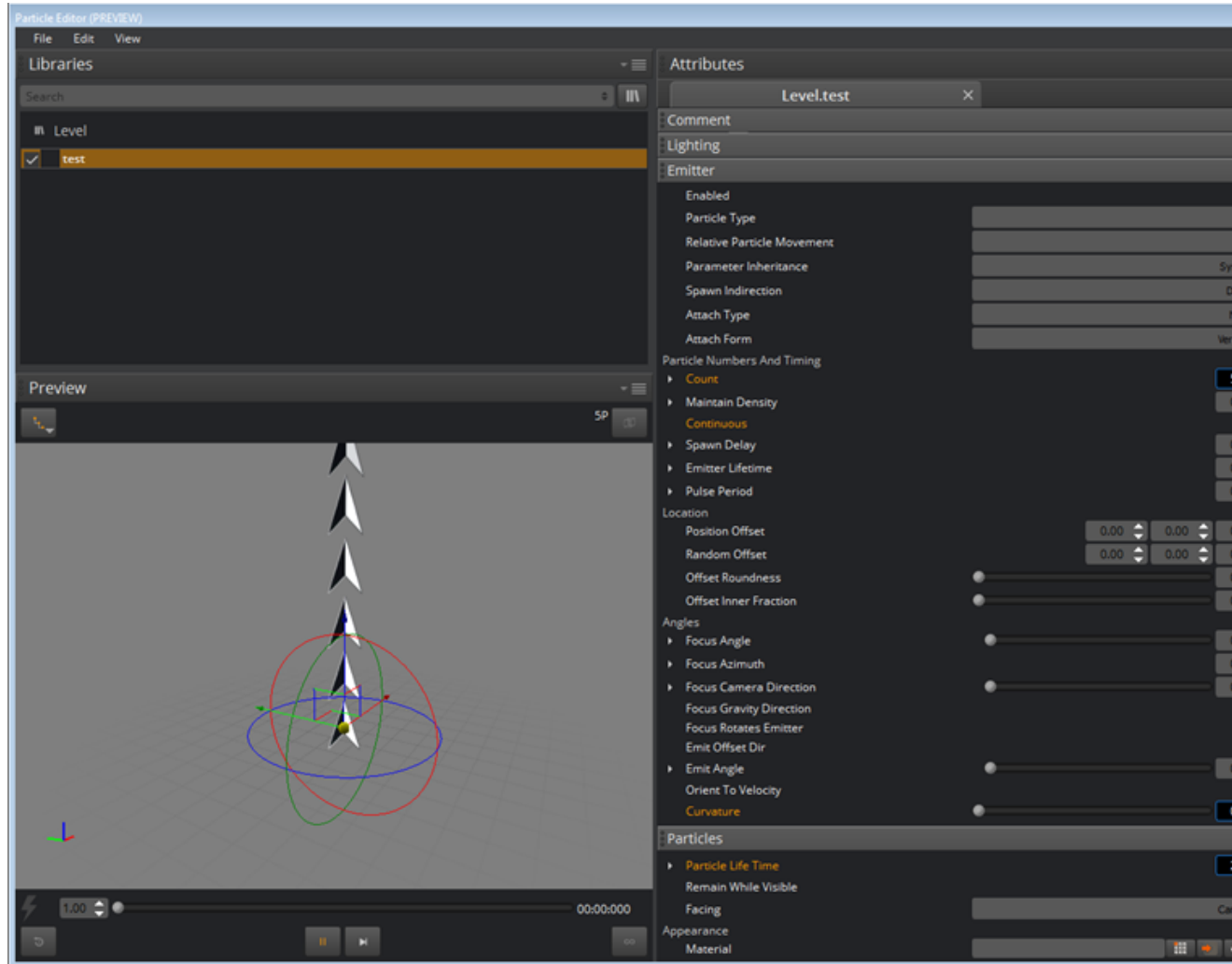
Particle Editor is in preview release and is subject to change.

You can use Particle Editor to create, edit, preview, manage, and save particle entities. The particle entity determines the position, angle, scale, and link information with other entities inside the level.

To add a particle entity to your level, simply drag a particle emitter from the **Library** and drop it into the viewport in Lumberyard Editor.

To access the Particle Editor from Lumberyard Editor, choose **View, Open View Pane, Particle Editor**. Or, choose the particle editor icon from the Lumberyard Editor toolbar. Particle Editor contains the following:

- **Library** panel – Lists particle art assets.
- **Preview** window – Displays the active selected particle effects. The camera is automatically positioned to capture the particle in its entirety. Click to pan the camera and use the mouse wheel to control the zoom level.
- **Attributes** panel – Lists the selected particle and its properties.



The following toolbar menu items and buttons are available in the Particle Editor main window and **Libraries** panel:

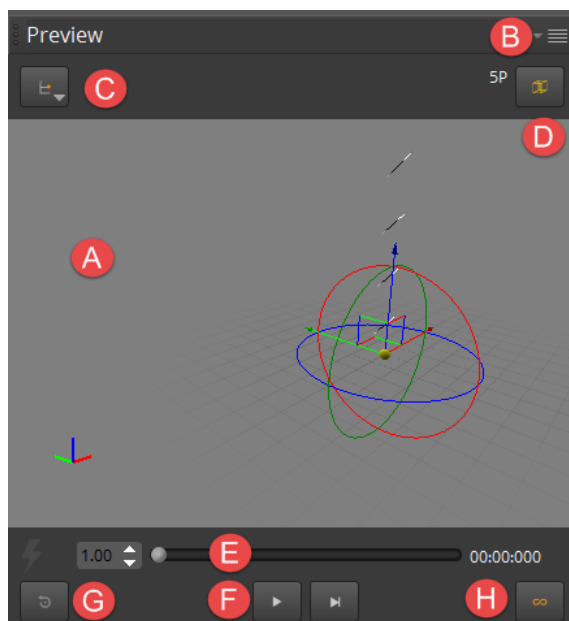
- **Import** – Opens the file browser to import the selected particle libraries.
- **Save** – Saves all modified particle libraries to disk.
- **Add library** – Adds a new particle library.
- **Remove** – Right-click the library to remove it from the window. Removes the currently selected library from memory (it is still available on disk).
- **Add Particle** – Adds a new particle effect, by default it is a child of the selected particle, folder, or library. The **New Particle Name** window opens, where you can set the particle name.
- **Add Folder** – Adds a folder within the library to organize your effects.
- **Duplicate** – Duplicates the currently selected particle effect.
- **Rename** – Renames the currently selected item.
- **Reload** – Right-click the library to reload saved particles.
- **Undo** – Undoes the last change.
- **Redo** – Removes the last undo.
- **Copy** – Copies all of the settings for the currently selected item to the clipboard.
- **Paste** – Writes data from the clip board to the currently selected item.

- **Reset to default** – Resets all properties and parameters for the currently selected item to the default values and states.
- **Edit Hotkeys** – Opens the **Hotkey** dialog for editing.

Using the Preview Window

The Preview pane has the following attributes:

- A – Viewport
- B – Main menu
- C – Camera list (choose what you want to see in the viewport)
- D – Toggles the wireframe view of the emitter
- E – Shows the playback timeline
- F – Play, pause, and step forward controls
- G – Resets emitter playback
- H – Loops playback



Customizing the UI

You can dock and float Particle Editor windows, or rearrange the panels to set up your workspace in different layouts.

All panels are moveable, which are indicated by the drag handle or page tear icon that appears in the left side of any panel or header bar. This indicates that the panel or window is customizable.

The Particle Editor is customizable in the following manner:

- **Floating panels** – You can click on the panel header bar and drag the editor to float it separately from the other panels in the editor. To add it back, drag the panel into the Particle Editor main window, and when the panel appears highlighted in blue, drop it into the Particle Editor main window.
- **Docking panels** – You can dock any of the panels along the inside edge of the editor window. Panels can also be docked along the top or any side of the other panels in the editor.

- **Tabbing panels** – To minimize the amount of panels that are seen at one time in the UI, you can dock a panel inside another one. This causes the two panels to display as tabs. You can toggle the tabs to display the panel and hide another.
- **Resetting Layout** – You can reset the layout back to default by choosing **View** and then choosing **Reset to default**. This resets the layout of the Particle Editor with the **Library** panel on the top left, the **Preview** window below, and the **Attributes** panel to the right.
- **Import Layout** – You can export and share layouts so that teams only have to customize the setup once. After you export a layout, you can choose **View** choose **Import layout**, and then choose **Browse** to find the layout file you want to use.
- **Export Layout** – After you create a layout, you can export a layout to share with your team by choosing **View**, choosing **Export layout**, and then choose **Browse** to find the location where you want to export the layout file.
- **Show/Hide Panels** – You can customize visible panels by showing or hiding them. To do this, choose **View** and then select a panel to show or hide it.

Using the Gradient Editor

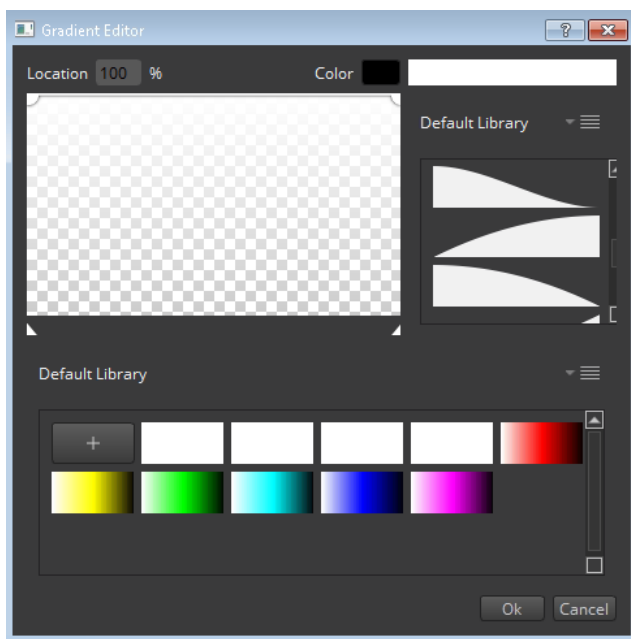
Particle Editor is in preview release and is subject to change.

You can use the Gradient Editor to apply color ranges to an emitter. With an emitter selected, in the Particles attribute, expand the color section to display the additional subparameters.

To access the Gradient editor

1. In the **Library**, choose an emitter.
2. In the **Attributes** panel, select **Particles**, and then expand **Color**.
3. Choose the **Emitter Strength** or **Particle Age** gradient box to open Gradient Editor.

Emitter strength of color provides the variance of the gradient and alpha applied.



The gradient Editor UI includes the following:

- **Location** – Set location value ranges from 0%-100%
- **Color** – Select the color thumbnail to open the Color Picker window
- **Gradient box** – The gradient and alpha combined that is applied
- **Gradient viewport:**
 - X-axis is the gradient generator of the color change over the full gradient
 - Y-axis is 0 - 100% alpha of the gradient color
- **Default alpha curve library** – Provides users with sets of alpha curves to start with
- **Default gradient library** – Provides users with sets of gradients to start with

Working with Color Gradients

When you select a gradient from **Default Library**, it displays in the Gradient Editor viewport along with the alpha curve. You can perform the following actions when selecting a gradient:

- To change a color, click the triangle key frames to engage the color picker to select a new color.
- To add a color to the gradient, double-click the x axis to generate another color key frame. This adds the selected color in the color thumbnail on top of the UI. Any adjustment to the gradient is displayed in the gradient output at the top of the UI.
- To display the RGBA values, hover over the color key frame.
- To delete a color key frame, select it, which highlights it orange, then press the **Delete** key.
- To adjust the alpha curve in the gradient viewport, click on the circle in the viewport, which is the alpha key frame and drag it up and down to adjust the percent of alpha (up is towards 100% and down is towards 0%). Dragging left and right adjusts the curve based on the curve endpoints.
- Alpha curve context menu – right-click the alpha curve key frame to display the following actions:
 - Delete selected keys
 - Create flat or linear curves
 - Adjust the in-and-out tangent of the curve to be linear or flat
 - Add a created curve to the library or preset list
 - Reset the curve to defaults
- To add an alpha key, double-click on the curve.
- To delete an alpha key, select the circle key and press **Delete**.
- To add the generated alpha curve to the preset list, click the **+** button.
- To add the generated gradient to the gradient presets list, click the **+** button.
- To delete a curve or gradient preset, right-click on the gradient or curve and select **Remove**.

Using Particle Editor Shortcut Keys

Particle Editor is in preview release and is subject to change.

Shortcut keys are available for most of the commands in Particle Editor menus. You can edit them by choosing **Edit**, choosing **Edit Hotkeys**, and then modifying them in the **Hotkey Configuration** dialog.

Hotkey Configuration Editor

Function	Description
Actions	<ul style="list-style-type: none">• Export: Exports the hotkey list to a file

Function	Description
	<ul style="list-style-type: none"> • Import: Imports a hotkey list from a file
Click to assign	Left-click on a shortcut to record a new shortcut; right-click to clear it. Click OK to save your changes.

Particle Editor supports the following keyboard shortcut keys.



Managing Particle Libraries

Particle Editor is in preview release and is subject to change.

This section discusses how to add and manage particle libraries using Particle Editor. The following functionality is supported:

- **Multi-library:** You can view multiple libraries and interact with them at the same time.
- **Multi-selection:** Pressing the `Ctrl` or `Shift` key allows you to select multiple emitters. The emitters do not need to be from the same library. The following functionality is supported:
 - **Copy:** Copies the selected items. When pasted, the copied items will become children of the item they are pasted on if there are multiple items.
 - **Delete:** Deletes all selected emitters.
 - **Group:** If all selected items share the same parent, you can group them together.

In addition, hot keys used with multiple items selected will apply to all selected items.

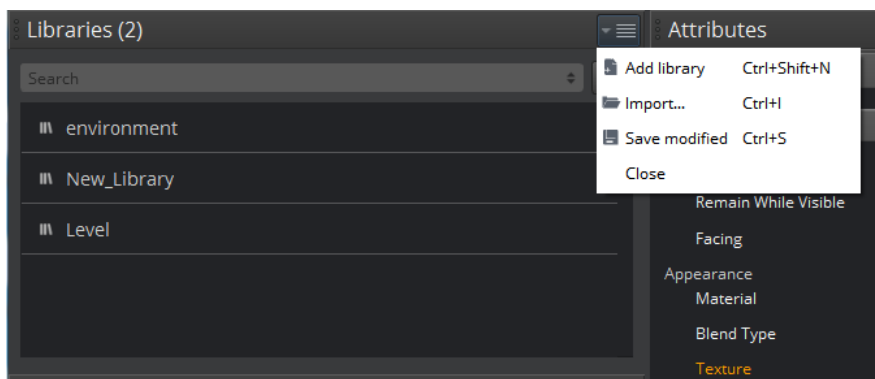
- **Drag and drop:** You can add emitters from any number of libraries into a specific library by dragging and dropping them onto the library's name. You can also drag emitters from within the same library to a new parent in the library.
- **Search:** You can type queries into the search field to view live results. Using the drop-down arrows will display previous search results.

Adding Particle Libraries

All particle effect data is stored in an XML-based library file. To create a new library, do the following:

To add a new particle library

1. In the Particle Editor, choose **File** and then choose **Add Library**. Alternatively, you can click the drop-down arrow to access the same menu or simply click the library icon button.
2. In the highlighted name field, type a name for the library.
3. Choose either **Add Particle** or **Add Folder** as applicable, then type a name and click **OK**.



Importing Particle Libraries

All particle effect data is stored in an XML-based library file. To import a new particle library, do the following:

To import a particle library

1. In the Particle Editor, choose **File**, choose **Import**. In the dialog that appears, select a preexisting library to load, then click **OK**. Alternatively, you can click the drop-down arrow to access the menu.
2. In the dialog box that appears, choose the library and then click **OK**.

Exporting Particle Libraries

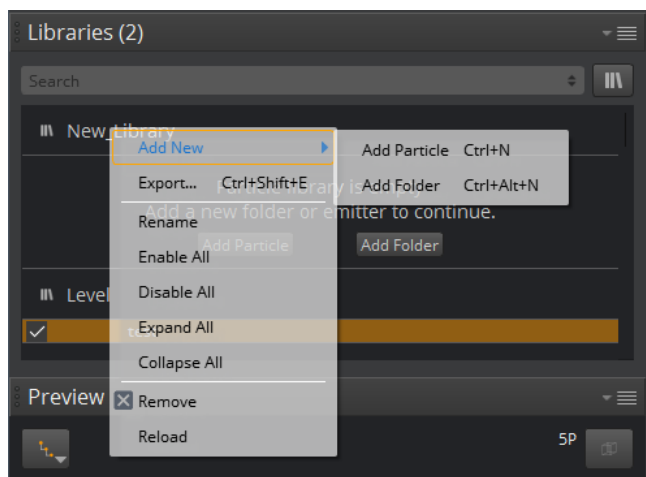
To export a new particle library, do the following:

To export a particle library

1. In the Particle Editor, right-click on the library name and choose **Export**.
2. In the dialog box that appears, select a location to save the library at and then click **Save**.

Using Particle Libraries

The following functionality is supported for managing particle libraries. To access this menu, right-click on the library name.



Function	Description
Add New	Add particle: Adds a new emitter to the library. Default hot key is <code>Ctrl+N</code> Add folder: Adds a new folder to the library. Default hot key is <code>Ctrl+Alt+N</code>
Export	Saves the library as the selected XML file.
Rename	Use to rename the library.
Disable/Enable All	Disables or enables all items in the library.
Expand/Collapse All	Expands or collapses all branches in the library.
Remove	Removes the library.
Reload	Reloads the library.

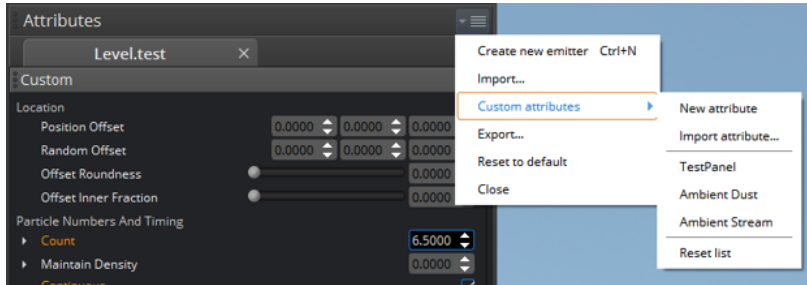
You can also left-click on a library name to collapse or expand the entire library. In doing so, the contents will not lose their collapse or expand state.

To save changes to a library, choose **File, Save**.

Creating Custom Attribute Panels

Particle Editor is in preview release and is subject to change.

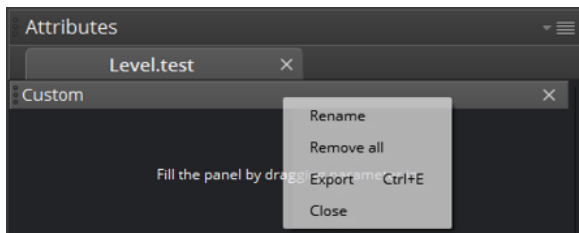
You can create your own custom particle attribute panel to allow you to quickly access those parameters you care about. Custom attribute panels can be created, imported, or selected by right-clicking the **Attributes** title bar and selecting **Custom attributes**.



Custom attribute submenu

Menu item	Description
New attribute	Adds a new empty custom panel to the Attribute view.
Import attribute	Use to load a pre-existing custom attribute panel. This also adds the panel to the panel preset list.
Panel preset list	List of custom panel presets
Reset List	Resets the preset list

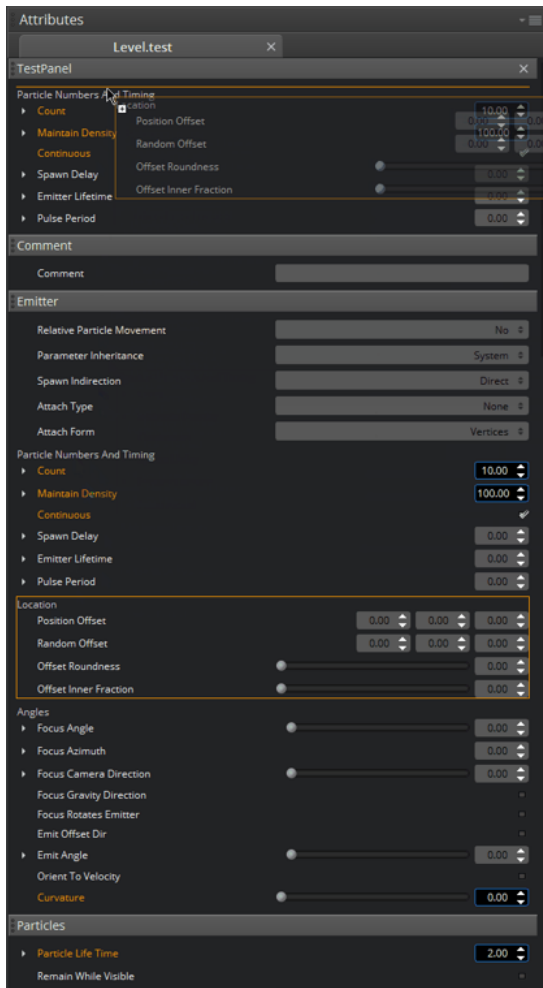
Once created, custom attribute panels can be renamed, emptied, and exported by right-clicking the custom panel's title bar.



Menu item	Description
Rename	Use to rename the custom panel.
Remove all	Use to remove all attributes in the panel.
Export	Exports the custom panel as the selected <code>.custom_attribute</code> file. This also will add the panel to the panel preset list.
Close	Use to close the custom panel.

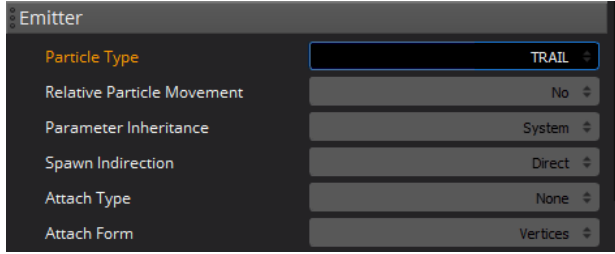
You can fill the custom attribute panel by dragging parameters into the panel. Dragging from an existing standard panel will copy the parameter onto the custom panel. In contrast, dragging a parameter from a custom panel will move the parameter to the new location. Select multiple parameters to quickly populate a custom panel by holding down the control key and clicking the desired parameters.

Preview the drop location with the drop indicator that appears when dragging to a valid location. If there is no drop indicator, the parameter will be inserted at the end of the panel. The following shows an example custom panel with drop indicators:



Particle Trails

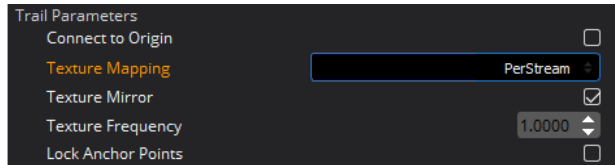
The particle Trail is an emitter type that connects different particles together. When selected, all particles in the particle emitter are automatically linked together forming a trail.



Particle Trail Parameters

The particle Trail is an emitter type that connects different particles together. When selected, all particles in the particle emitter are automatically linked together forming a trail.

The following table lists the various parameters associated with particle Trails:



Parameter Function	Description
Connect To Origin	Connects the newest particle to the emitter origin, with the parameters of a particle age = 0.
Texture Mapping	Specifies how the texture is repeated over the trail stream. PerParticle sets a default frequency of one texture per particle. PerStream sets a default frequency of one texture stretched over the whole stream.
Texture Mirror	Option which causes adjacent texture tiles to alternate direction. If false, they wrap at each repetition. Default value: true.
Texture Frequency	Multiplies the texture repeating frequency specified above. Can be less than 0 or greater than 0, which determines texture direction. Default value: false
Lock Anchor Points	Locks the UVs of the vertices of the trail. This will cause the texture to stay on the PerStream texture-mapped trail. It will not "catch up" as the UVs would do normally.

Particle Trail Visibility

You can make short trail segments not be visible. For a trail segment to be drawn, the distance between the start and the end of the segment needs to exceed the **Min visible distance** setting in the following table. The purpose for this feature is to automatically turn off drawing trails that are not moving, or are moving too slow. This is useful for particle trail effects that only need to be drawn when the emitter is moving.

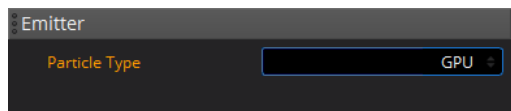


Parameter Function	Description
Min visible segment length	Enables and disables the feature.
Min visible distance	The minimal distance between the start and end of a trail segment. Segments smaller than this value will become transparent.

GPU Particles

Unlike CPU particles, GPU particles are processed and rendered entirely by the graphics card GPU. Since the GPU is handling the calculations, many more particles can be processed at once, allowing for much denser and more detailed particle behavior.

GPU particles have **Particle Type** of **GPU** as displayed in the **Emitter** panel of Particle Editor.



The following attributes and parameters are supported for GPU particles:

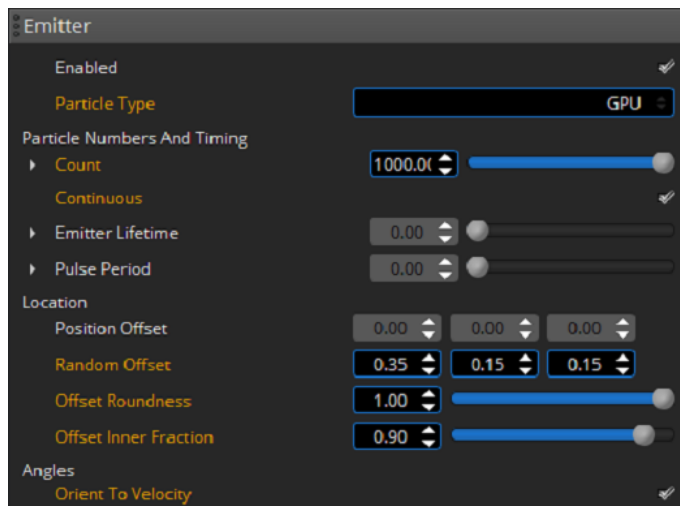
Attribute Comment

You can use the **Comment** area to save any comments about an attribute. Comments are editable.



GPU Emitter Attribute

Parameters in this attribute control the particle amount and spawning location of the particles.

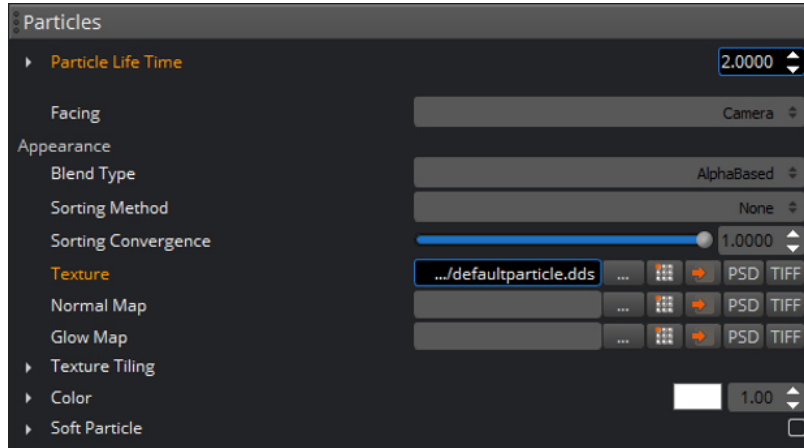


Attribute Parameters

Parameter Function	Description
Enabled	Enables or disables the particle emitter.
Particle Type	Select either CPU or GPU particles.
Count	The total number of particles at any one time that are active. Determines the emission rate (Count/Particle Lifetime). Value range: 0 - millions (performance based on hardware specs)
Continuous	If disabled, all particles are emitted at once, and the emitter then dies. If enabled, particles are emitted gradually over the emitter lifetime. If enabled, and Emitter Lifetime = 0, particles are emitted gradually, at a rate of Count/Particle Lifetime per second, indefinitely.
Emitter Lifetime	If Continuous is enabled, specifies the lifetime of the emitter. Emitter Lifetime does not apply to non-continuous effects, which always disappear as soon as they have emitted all of their particles.
Pulse Period	If greater than 0 and Continuous is disabled, the emitter spawns another burst of particles repeatedly at this interval.
Position Offset	X, Y, and Z values define the spawning position away from the emitter itself, in emitter space.
Random Offset	X, Y, and Z values define the range of a random spawning box in both directions away from the position offset.
Offset Roundness	Fraction of spawning shape volume corners to round. Value range: 0 (box shape) to 1 (ellipsoid shape). Values in-between correspond to a rounded box.
Offset Inner Fraction	Ratio of inner to outer spawning shape volume. Value range: 0 (spawn uniformly within the entire volume) to 1 (spawn only at the outer edge of the volume). Values in-between vary the thickness of the inner cutout volume.
Orient to Velocity	Rotate particles so that they are oriented towards the velocity of each individual particle.

GPU Particles Attribute

Parameters in this attribute control the basic appearance of the particle. This attribute should be set up first as it includes the **Texture** slot, which is used for most particles.



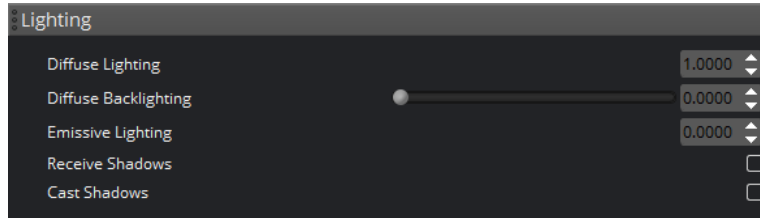
Attribute Parameters

Parameter Function	Description
Particle Life Time	The lifetime of individual particles in seconds. Even after the emitter's lifetime has expired, spawned particles still will live out their own lifetime.
Facing	Applies only to 2D particles. Determines how the sprite is oriented in space. Texture orientation is further modified by rotational parameters: <ul style="list-style-type: none"> • Camera (default): Particle faces the camera, texture X and Y aligned with screen X and Y. In this mode, particles are assumed to represent spherical objects, and are lit accordingly. Rotations become local to the camera. • Free: Rotates freely in 3D.
Blend Type	Applies only to 2D particles. Determines how the sprite blends with the background. <ul style="list-style-type: none"> • AlphaBased: Uses alpha channel for transparency. Linearly interpolates existing color with particle color, based on particle alpha. Final Color = Particle Color * Particle Alpha + Background Color * (1 - Particle Alpha). • Additive: Adds color values of particle to existing color. Final Color = Particle Color + Background Color. • Multiplicative: Multiplies the existing color with the particle color. Final Color = Particle Color * 2 * Background Color. • Opaque: No blending. Existing colors are replaced with particle color. Fully opaque particles with an alpha threshold discarding fully transparent pixels.
Sorting Method	Configure method of sorting used. This only applies when Blend Type is set to AlphaBlend , otherwise this function will have no effect. <ul style="list-style-type: none"> • Bitonic: Used when a full sort is required, but does not support partial convergence. • Odd-Even: Converges over time, allowing for tweaking performance utilization of the sort versus quality. Especially useful in conjunction with particle LODs as distant particles

Parameter Function	Description
	don't change sort order often, making them ideal for a low convergence odd-even sort.
Sorting Convergence	Odd-even sort convergence for each frame on a scale from zero to one. For example, a value of 0.5 would cause the sort to always fully converge in two frames, while a value of one would cause the sort to fully converge every frame.
Texture	Use to open the Asset Browser and assign a texture used for 2D sprite images. Displays a preview of the texture when the mouse cursor is over the input box.
Normal Map	2D normal map image used for individual particles.
Glow Map	2D glow map image used for individual particles. Lighting attribute values will affect the glow map visibility.
Texture Tiling	Controls 2D image tiling and animation parameters.
Color	<p>Used to select the color to apply to the particle:</p> <ul style="list-style-type: none"> • Random: How much a particle's initial color varies downward from the default. 0 = no variation, 1 = random black to default. Value range: 0-1 • Random Hue: Causes the Random color variation to occur separately in the 3 color channels. If false, variation is in luminance only. Default value: false • Emitter Strength: Define the color of the particle over the emitter's lifetime. Double-clicking opens the Gradient Editor. • Particle Age: Defines the color of the particle over the particle's lifetime. Double-clicking opens the Gradient Editor. <p>Particle color can also be randomly interpolated between two gradients by right-clicking the Emitter Strength or Particle Age fields and selecting Randomly between two gradients.</p>
Soft Particle	<p>Enables particle soft-shadowing. This will cause the particle to softly fade with depth proximity, causing a reduction in hard edges when interacting with other geometry. A value of one indicates maximum fading, a value of zero will nullify the effect.</p> <p>Note This option must be enabled before this variable will take effect. Also, instead of setting the value to 0, disabling this option is preferred.</p>

GPU Lighting Attribute

Parameters in this attribute control the lighting of the GPU particle.

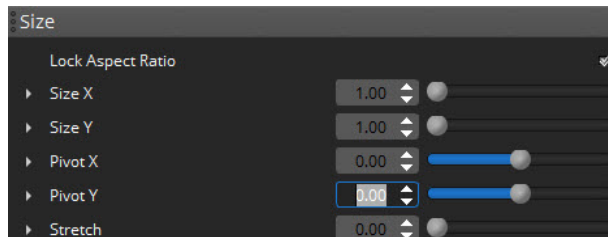


Attribute Parameters

Parameter Function	Description
Diffuse Lighting	The amount of diffuse lighting the particles receive. A value of one means that the particles should receive normal lighting comparable to any other geometry in the world.
Diffuse Backlighting	The amount of diffuse lighting for back-facing particles. Can be used to convey translucency.
Emissive Lighting	The quantity of self-emission. A value of one indicates that the particle is identical to the texture input.
Receive Shadows	Allows the particles to receive shadows from lights in the scene.
Cast Shadows	Allows the particles to cast shadows from the perspective of the lights in the scene.

GPU Size Attribute

Parameters in this attribute control the size and shape of the sprite.



Attribute Parameters

Parameter Function	Description
Lock Aspect Ratio	Maintain particle aspect ratio. Default value: false
Size X, Y	For 2D particles, the width and height of the particle in world-space units.
Pivot X, Y	Moves the horizontal and vertical offset of the pivot point of the particle. Positive values point to the right and down. Value range: -1 to +1 Default value: 0 (texture center)

Parameter Function	Description
Stretch	The amount of stretch applied to the particle in the direction of travel, in seconds (based on current velocity). Stretches in both directions by default.

GPU Rotation Attribute

Parameters in this attribute control the rotation of the particle.



Attribute Parameters

Parameter Function	Description
Init Angles	X, Y, and Z values define the initial angle applied to the particles upon spawning, in degrees.
Random Angles	X, Y, and Z values define the random variation (bidirectional) to Init Angles , in degrees.
Rotation Rate X, Y, Z	Constant particle rotation, in degrees/second. The axes are the same as for Init Angles .

GPU Movement Attribute

Parameters in this attribute control the movement of the sprite.



Attribute Parameters

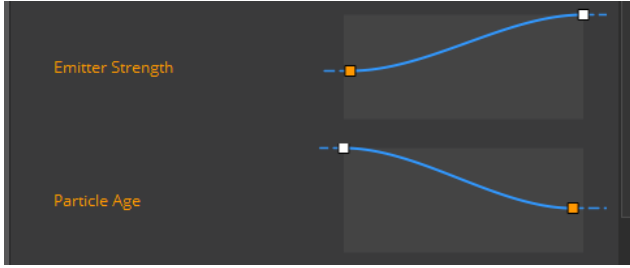
Parameter Function	Description
Speed	The initial speed of particles.

Parameter Function	Description
	Value range: any Default values: 5
Acceleration	X, Y, and Z values define the constant acceleration applied to particles in world space. Value range: any Default values: 0,0,0
Inherit Velocity	What fraction of initial velocity is inherited from the particle's parent. For indirect particles, the parent particle's velocity is inherited. For direct particles, the emitter's velocity is inherited. Value range: any Default value: 0
Air Resistance	Drag constant. Behaves as exponential decay of velocity, simulating air friction.
Gravity Scale	Multiple of world gravity to apply to particles. A value of 0.0 means no gravity. Most physicalized particles should be set to 1, which corresponds to -9.8 m/s Earth gravity. (use Air Resistance to provide drag). Set to a negative value for buoyant particles such as smoke. Value range: any Default values: 0,0,0
Turbulence3DSpeed	Adds a 3D random turbulent movement to the particle, with the specified average speed, in meters/second-squared. Value range: 0+ Default value: 0
Turbulence Size	Adds a spiral movement to the particles, with the specified radius. The axis of the spiral is set from the particle's velocity. Value range: 0+
Turbulence Speed	When Turbulence Size is greater than 0, the angular speed, in degrees/second, of the spiral motion. Value range: any Default value: 0

GPU Particle Parameter Modifiers

Modifiers allow parameter values to mutate over particle or emitter lifetime.





Attribute Parameters

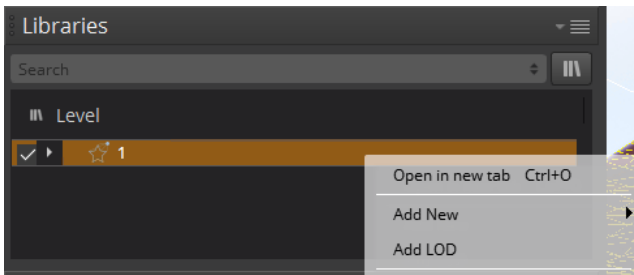
Parameter Function	Description
Random	Value can be randomized. Value range: 0.0 (no randomness) to 1.0 (complete randomness)
Emitter Strength	Value can be influenced by emitter strength. Serves as a user specified modifier per-emitter. Default value: -1
Particle Age	Value can be influenced by particle age. Allows to vary parameter during the lifetime of the particle.

Particle Level Of Detail (LOD)

The Level Of Detail (LOD) system blends multiple particle emitters depending on their distance to the camera. This allows for computationally-heavy particle emitters to be swapped out for emitters that require less computation and rendering time.

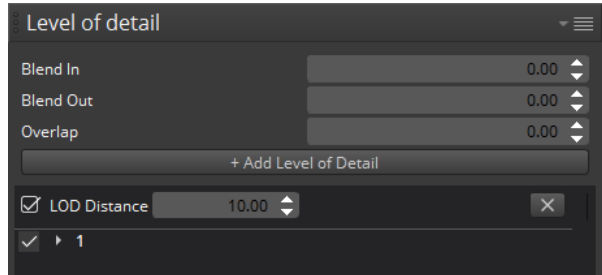
You can add an LOD for a particle emitter by right-clicking on the emitter in the **Libraries** panel and selecting **Add LOD**.

The LOD will be a copy of the base particle emitter and will have the same settings. The LOD will also be applied to any parent or child particle emitters the selected emitter has all the way up and down the hierarchy.



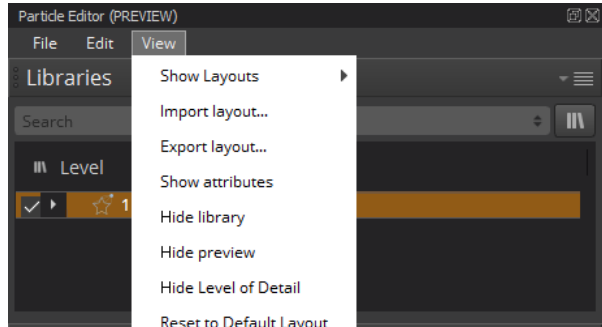
Level Of Detail Panel

The **level of detail** panel is displayed when an LOD has been added and show which level of detail is selected in the **View** options of the Particle Editor.



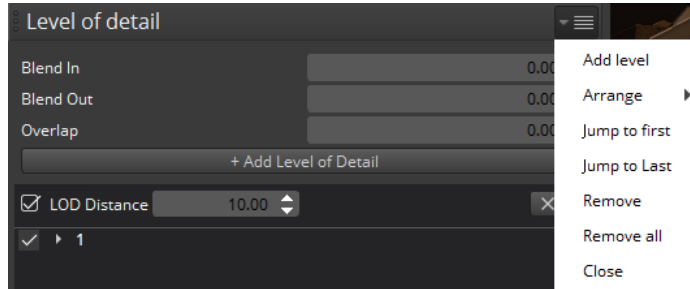
Parameter Function	Description
Blend In	Amount of time it takes for a LOD to blend in in seconds.
Blend Out	Amount of time it takes for a LOD to blend out in seconds
Overlap	Amount of time both LODs are shown before the old LOD blends out and the new LOD blends in in seconds.
+ Add Level of Detail	Adds another LOD. New LOD distance will be set to 10 additional units from the furthest LOD. Under + Add Level of Detail is the list of added LOD levels.

You can hide or display the **Level of Detail** panel by clicking the **View** tab and accessing the dropdown menu.



Parameter Function	Description
Hide Level of Detail	Hides the LOD panel when panel is visible.
Show Level of Detail	Displays the LOD panel when panel is hidden.

You can manage your LOD levels in the list by right-clicking the top-right menu button in the **Level of detail** panel.



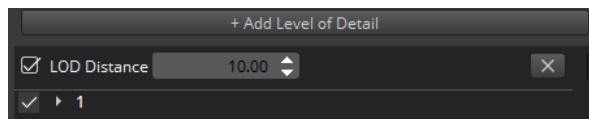
Menu Option	Description
Add level	Adds a new LOD level at the bottom of the list.
Arrange	Shows the Arrange submenu. <ul style="list-style-type: none"> • Move Up: Moves the selected LOD level up one position in the list. This also changes the Level LOD Distance to be 1.0 lower than the previous level. • Move Down: Moves the selected LOD level down one position. This also changes the Level LOD Distance to be 1.0 higher than the previous level. • Move to top: Moves the selected LOD level to the top of the list. This also changes the Level LOD Distance to be 1.0 lower than the previous top level. • Move to bottom: Moves the selected LOD Level to the end of the list. This also changes the Level LOD Distance to be 1.0 higher than the previous bottom level.
Jump to first	Selects the first LOD level in the list. This also selects the top particle emitter in that list and loads it in the Attribute view.
Jump to last	Selects the last LOD level in the list. This also selects the top particle emitter in that list and loads it in the Attribute view.
Remove	Removes the currently selected LOD level.
Remove All	Removes all LOD levels for all related particle emitters.
Close	Closes the Level of Detail panel.

LOD Level Panel

Each LOD level has its own panel in the LOD level list. These panels show all relevant information for each individual level.

Note

If the camera distance is lower than the top LOD level, the base particle emitter is shown. This makes the base particle; the starting LOD level.



UI Element	Description
Top-left checkbox	Turns the entire level on or off.
LOD distance value	The camera distance at which the LOD level will become active. At this level, the particle emitter will blend towards this LOD level and blend out the previous LOD level.
Top-right button	Deletes the corresponding LOD level.
Particle checkbox	Turns off the particle emitter at this level. When turned off nothing will be drawn. This allows users to turn off particle emitters based on LOD level.
Particle name	<p>When a particle name is left-clicked, the LOD level particle emitter for the selected particle emitter is loaded in the Attribute view, allowing for change to the LOD level particle emitter.</p> <p>When a particle name is right-clicked, the Remove option removes the particle from the LOD level. Any child particle emitters are also removed from the LOD level.</p>

Managing Emitters

Particle Editor is in preview release and is subject to change.

In this section, you learn how to create, edit, and manage emitters for particle effects. For more information about emitter attributes and parameters displayed in the **Attributes** panel, see [Particle Attributes and Parameters Reference \(p. 956\)](#).

Before you can create or edit emitters, you must first set up a particle library. For more information, see [Adding Particle Libraries \(p. 935\)](#).

Creating Emitters

To create an emitter, do the following:

To create new emitters

1. In the **Attributes** panel, click the down arrow and choose **Create new emitter**. Alternatively, right-click the library name, click **Add New**, then click **Add Particle**.
2. In the dialog box that appears, type a name for the emitter and then choose **OK**. Do not use special characters in the name.
3. In the **Attributes** panel, edit the attributes and parameters as needed.

Duplicating Emitters

To duplicate an emitter, do the following (this also duplicates any associated child emitters):

To duplicate emitters

1. In the **Library** panel, right-click the emitter that you want to duplicate and then choose **Duplicate**.
2. In the dialog box that appears, type a name for the emitter and then choose **OK**.

Creating Child Emitters

To create a child emitter, you must first set the parent effect. Then, you attach the child emitter to the parent particles. Multiple child emitters can be attached to the parent particle. A particle effect can have any number of child effects, also known as subeffects, which you can nest in a library by dragging-and-dropping them where needed.

To create child emitters

1. In the **Library** panel, right-click the emitter you want to create a child emitter for, choose **Add New**, and then choose **Add Particle**.
2. In the dialog box that appears, type a name for the child emitter and then choose **OK**.

You can also assign an existing emitter to be a child by choosing the emitter and then dragging it on top of another emitter. The selected emitter now lives underneath as a child.

To remove a child emitter from a parent, select the child emitter and drag the emitter outside of the directory structure to the library name in the **Library** pane. This emitter is now a peer emitter and is no longer a child emitter.

There are two kinds of child effects:

- **Regular child effects** – These effects behave like separate effects, except that they are spawned with and attached to their parent effect. Each child effect has its own independent parameters and lifetime, allowing for an overall effect that consists of several parts.
- **Second-generation child effects** – These are effects are attached to the individual particles of the parent effect. A separate emitter is spawned for each particle of the parent effect, and those emitters move with their parent particles. This allows you to create much more complex effects. Second-generation effects can be nested multiple times, creating third-generation (and greater) effects.

An example of a child effect is attaching an emitter to a parent particle and leaving trailing particles behind.

Editing Emitters

To edit emitter attributes and parameters, do the following:

To edit emitter attributes and parameters

1. In the **Library** panel, choose the emitter.
2. In the **Attributes** panel, adjust attribute and parameter settings and values in the different sections to achieve the desired effect.

Organizing Emitters in a Library

All particle emitters are listed in the **Library**. When you organize your emitters, you create relationships between them. You can have single emitters, emitters with child emitters, and emitters that have parent and child emitters of their own. You can also create folders within each library to help organize your particle effects. This relationship is displayed in a tree hierarchy in the **Library**.

There is a visual indicator that shows you where the emitter is being placed based on the position of the cursor. If an emitter is being placed on another emitter or a folder for grouping, the folder row appears highlighted with a blue stroke.

Reverting Changes to Emitter Attributes

Emitter attributes are a list of attributes or property types an emitter can have. Emitters have default parameters set to the attribute as a common starting place for that attribute.

The default attributes are indicated with a white text label. When you change the attribute parameter, the text label changes color to orange, which indicates that the attributes parameter has been changed from the default state.

To revert the last change to the emitter attribute by undoing the last action, right-click the attribute name and then choose **Undo**. To revert any changes made to the attributes parameter back to the default parameter, right-click the attribute name and then choose **reset to default**.

Attributes are categorized so that you can identify them easily. By default, categories are stacked, but you can reorder and rearrange them, including arranging joining categories as tabs.

- To reorder attribute categories, drag the category to the desired position, and then drop the category in the position you want when the blue highlight appears.
- To combine categories into tabs, drag a category onto another category header bar. These categories now are combined and tabbed. If the category is not expanded, the tabs collapse until you choose the attribute category header to expand them.
- To revert the changes made to the **Attributes** panel layout, click the hamburger icon at the upper-right side, and then choose **Reset layout** to revert the layout back to the default.

Advanced Particle Techniques

Particle Editor is in preview release and is subject to change.

Attaching Particle Effects to Basic Geometry Entities

To attach a particle effect to a geometric entity, choose the chain link (**Link Object**) toolbar icon in Lumberyard Editor and link the particle entity to the source object entity. Then set the **AttachType** and **AttachForm** parameters, which are located in under **ParticleEntity Properties** in **Rollup Bar** for the entity.

For second-generation particle effects, you can attach emitters to the parent particles as part of the parent particle effect. If the parent particle effect contains geometry, the second generation effect can optionally emit particles from that geometry, based on the **AttachType** and **AttachForm** parameters.

Attaching Particles to Breakable Objects

There are several ways to create breakable geometry objects, but all are based on a multi-part .cgf file, and all optionally allow secondary particle effects to be spawned on the broken pieces. Here are the different ways to create breakable objects that spawn particle effects:

- **Pure particle effect** – To create a particle effect that instantly creates an exploding object, use the following method:
 - Set the effect **Geometry** to a multi-part .cgf file.
 - Set **Geometry in Pieces** = True and **Count** = 1 (for one exploding object).
 - Set appropriate values for **Speed**, **Focus**, **Rotation Rate** to create a nice exploding effect.
 - Optionally set **Rigid Body** for physicalized pieces.

- **DestroyableObjects** – This is a special entity that can be "exploded" via an event, with all pieces breaking off at once. Particle emitters are optionally attached to each piece based on its material surface properties.
- **Physically breakable object** – This is a basic Entity set to use a multi-part `.cgf` file with physics parameters specifying how pieces break off. The physics system breaks pieces of this geometry off individually based on external forces. Particle emitters are optionally attached to each piece based on its surface properties.

Attaching Particles to Character Animations

If a particle effect is already playing on a character skeleton, you cannot trigger it again. In other words, it will not play if it's already playing.

To attach a particle to an animation

1. In Geppetto, in the **Assets** panel, expand **Characters, Characters** and select a suitable `.cdf` asset.
2. Expand **Animations** and select the applicable animation to which you want to attach a particle. The animation plays in the viewport.
3. In the **Playback** window, click **Pause** if the animation is playing, then double-click in the timeline at the precise point in time you want the particle event to start playing.
4. Right-click, then select **New Event** to create an event at this animation point. Name the event **effect**.
5. In **Particle Editor**, in the **Properties** panel, select the desired particle effect.
6. In **Geppetto**, click **Use Selected Effect**.
7. In the **Playback** window, click **Play** to view the particle effect in the animation.
8. Click **Save**. This creates an `.animevent` file that is stored with the `.cdf` file.

Precise positioning can be achieved by attaching the particle to a particular joint for the character.

Generating Particles from Surface Properties

An object's material surface properties define which event-driven effects can occur when something happens to the object. These events can be specified on a render material, and also on individual pieces or surfaces of a `.cgf` asset.

Many of these properties specify those particle effects that are spawned based on events such as "bullet" hit or "walk." The specific effect spawned when a geometry piece breaks off of an object is specified in a section of the LUA script that contains the following parameters:

Parameters

Parameter	Description
Name	Particle effect name.
Scale	Used to multiply particle sizes.
Count_scale	Used to multiply particle counts.
Count_per_unit	Used to cause particles to be emitted at the same density, regardless of the size of the attached object.

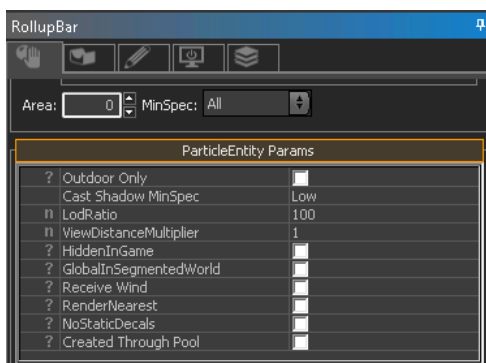
Particle Entity Parameters and Properties

Particle Editor is in preview release and is subject to change.

The following particle entity parameters and properties are accessed using Rollup Bar.

To change particle entity parameters and properties

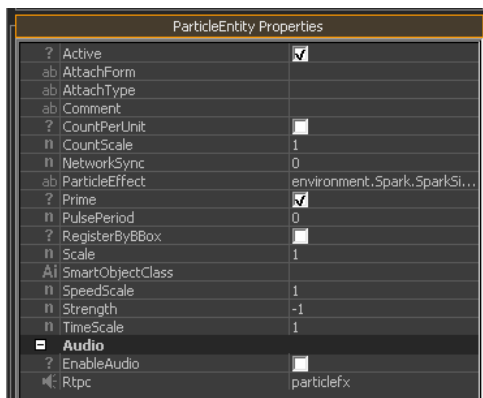
1. In Rollup Bar, on the **Objects** tab, click **Particle Entity**.
2. Under **Browser**, select a suitable particle and then drag it onto the viewport. The particle is displayed.
3. Under **ParticleEntity Params**, change the following parameter's values as needed for the desired effect.



Particle Entity Parameters

Parameter	Description
Outdoor Only	When set, object will not be rendered when inside a VisArea.
Cast Shadow MinSpec	When set, object casts a shadow on the selected quality setting and above.
LodRatio	Defines how far from the current camera position that different Level Of Detail (LOD) models for the object are used.
ViewDistanceMultiplier	Defines how far from the current camera position that the object is rendered.
HiddenInGame	When set, object is not shown in game mode.
Receive Wind	When set, object is influenced by wind parameters in the level.
RenderNearest	Used to eliminate Z-buffer artifacts when rendering in first-person view.
NoStaticDecals	If set to true, decals are not rendered for the selected object.
Created Through Pool	Used primarily for AI entities for memory optimization.

4. Under **ParticleEntity Properties**, change the following property's values as needed for the desired effect.



ParticleEntity Properties

Parameter	Description
Active	Sets the initially active or inactive. Can be toggled in the editor for testing.
AttachForm	If AttachType is not empty, this property determines where particles emit from the attached geometry. Set to Vertices, Edges, Surface, or Volume.
AttachType	If this entity is attached to a parent entity, this field can be used to cause particles to emit from the entity's geometry. Set to BoundingBox, Physics, or Render to emit from the applicable geometry.
CountPerUnit	If AttachType is not empty, this multiplies the particle count by the "extent" of the attached geometry. Depending on AttachForm, the extent is either total vertex count, edge length, surface area, or volume.
CountScale	Multiplies the particle counts of the entire emitter.
ParticleEffect	Use to generate the following effects:
Prime	If true, and the assigned ParticleEffect is immortal, causes the emitter to start "primed" to its equilibrium state, rather than starting up from scratch. Very useful for placed effects such as fires or waterfalls, which are supposed to be already running when the level starts. Applies only to immortal, not mortal effects.
PulsePeriod	If not 0, restarts the emitter repeatedly at this time interval. Should be used to create emitters that pulse on and off at somewhat large intervals, a second or so. Do not set a low value such as 0.1 to try to make an instant effect into a continuous one. Make sure the actual library effect is set Continuous and has an appropriate Count.
RegisterByBBBox	Uses the emitter's (automatically computed) bounding box to determine which VisAreas it is visible in. If this is disabled (the default), the emitter's origin alone determines VisArea membership, as the bounding box is hard to exactly control by the designer.

Parameter	Description
Scale	Multiplies the overall size and velocity of the entire emitter.
SpeedScale	Multiplies the particle emission speed of the entire emitter.
Strength	Used by effect parameters to modify their value. If a parameter has an Emitter Strength curve, and the emitter entity's Strength property is not negative, then Strength will be used as input to this curve.
TimeScale	Multiplies the elapsed time used to simulate the emitter. Less than 1 achieves a show-motion effect.
EnableAudio	Toggles sound emission on any sub-effects with an Audio parameter set.

Particle Attributes and Parameters Reference

Particle Editor is in preview release and is subject to change.

Particle parameters are stored in various attributes that you can reposition and resize. These attributes describe how an emitter and its particles look and behave. Aside from a parameter's base value, most numeric parameters also allow random variation over particle or emitter lifetime. The following reference lists particle attributes and associated parameters that can be adjusted for the desired effect. These are available from the **Attributes** panel of the Particle Editor.

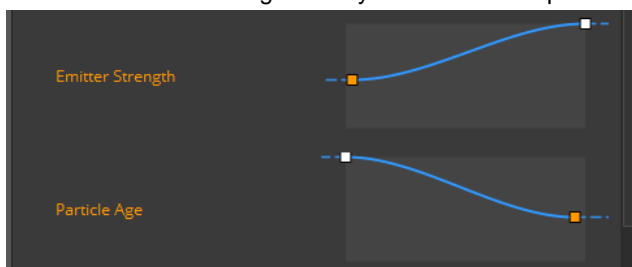
A number of parameters also feature several sub-parameters, as follows:

- **Random** – Specifies how much a particle's parameter value deviates from the default value of 0 (no variation).
- **Emitter Strength** - Controls the alpha strength over the lifetime of the particle. Only works with finite particles. If continuous is set, this has no effect.
- **Particle Age** - Controls the alpha over the individual particles lifetime. For example, use this to fade a smoke particle away to nothing once its lifetime has finished. Depending on where you reduce the value to zero, the particle fades out earlier or later.

For information on GPU particle attributes and parameters, see [GPU Particles \(p. 940\)](#).

Using the Curve Editor

You can use the curve editor to edit the shape of the emitter strength curve as well as the particle age over time. Emitter strength is only active if certain parameters are set.



To edit emitter strength using the curve editor

1. Double-click to set a new key along the curve timeline.
2. Using your mouse, drag the curve to the desired value and shape.

Attribute Comment

Particle Editor is in preview release and is subject to change.

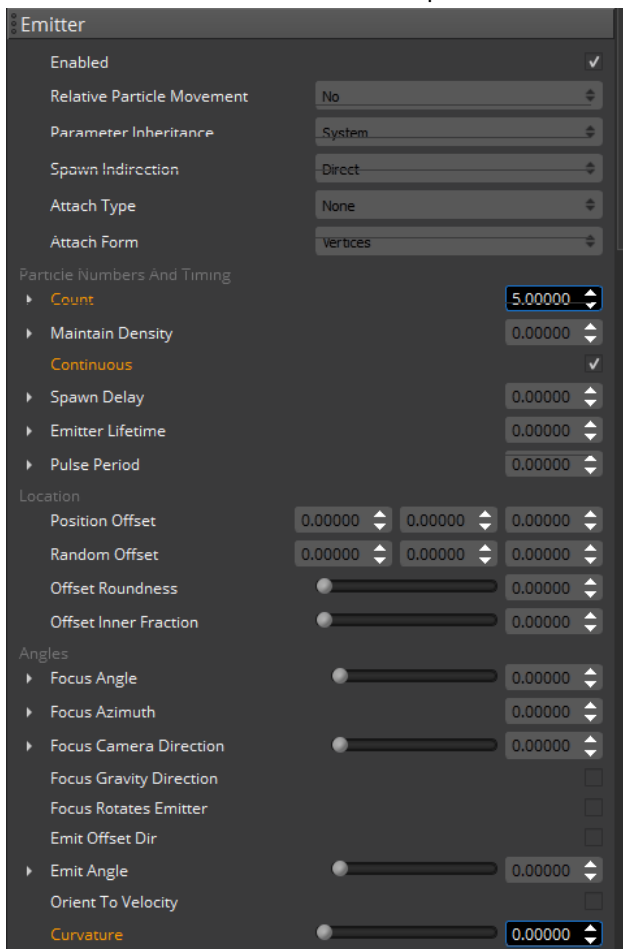
You can use the **Comment** area to save any comments about an attribute. Comments are editable.



Emitter Attribute

Particle Editor is in preview release and is subject to change.

Parameters in this attribute control the particle amount and spawning location of the particles.



Emitter Attribute Parameters

Parameter Function	Description
Relative Particle Movement	<p>Particle motion in the emitter's space. For an example of particles emitted upwards from an emitter:</p> <p>No: The emitted particles drift upwards and fall behind as the emitter moves away, like smoke from the chimney of a train for example.</p> <p>Yes: The emitted particles drift upwards but won't fall behind, resulting in a smoke column going straight up from the chimney, for example.</p> <p>Default: No</p>
Parameter Inheritance	<p>Specifies the source for default (starting) effect parameters:</p> <ul style="list-style-type: none"> • System (default): Reads the <code>System.Default</code> effect to use as defaults. If no such effect exists, uses Standard defaults. • Standard: Uses the hard-coded parameter defaults. These are 0 or off for most parameters, except for the multipliers, for which 1 is the preferred default value. • Parent: Uses the parent particle effect for defaults. One possible use for this is to create a parent effect with one set of parameters, then a variety of sub-effects which alter some of the parameters for variation. Sub-effects can be spawned on their own. Editing the parent effect updates the default values of all sub-effects. <p>The source that is selected has the following consequences:</p> <ul style="list-style-type: none"> • When you create a new effect, it takes its default parameters from the Inheritance source, which by default is System. • The labels of all non-default parameters are highlighted in the Particle Editor. This allows you to quickly see which values you have actually changed. • If you change the Inheritance source, it does not change any other parameters. However, different parameters may be highlighted, as their defaults have changed. • To actually set parameters to their default values, right-click the parameter and then click Reset to default.. This can be done just after creating a new effect and changing its Parameter Inheritance value or at any time during editing, to reset parameters to the selected Inheritance default. The Parameter Inheritance parameter itself is not changed by resetting. • When effects are saved to XML libraries, only non-default values are saved. When they are loaded from XML, the current defaults for the effect's Inheritance are used as a base. • When you edit any parameters of a parent effect, the non-edited parameters of all children (and descendents) which have Parent selected are instantly updated. • To customize the default effects for your game, create a System library, and a Default effect.

Parameter Function	Description
	<ul style="list-style-type: none"> If you edit the System.Default effect, and then Save the System library, the non-edited parameters of all effects and emitters are updated. To customize the default effects for a specific Configuration, create a child of the System.Default effect, give it any name you want, set its Inheritance = Parent (not required, but helpful), set its Configuration parameters to a subset of possible configurations (e.g. VeryHigh only), and then edit its effects. When the engine looks for the default parameters to use, it looks for the deepest effect in the System.Default family which matches the current engine configuration.
Spawn Indirection	<p>This parameter has the following values:</p> <ul style="list-style-type: none"> Direct: Spawns without relying on the parent's input for timing. ParentStart: Spawn once the parent has spawned. ParentCollide: Once the parent particle has collided with an object, this is the trigger to spawn a particle with this setting. ParentDeath: When the parent particle has lived out its lifetime, this is the trigger to spawn a particle with this setting.
Attach Type	<p>Specify the location of emission when the emitter is attached to geometry, or when the parent particle has geometry.</p> <ul style="list-style-type: none"> None: Particles ignore geometry and emit from emitter center as normal. Physics: Particles emit from the geometry of the attached physics object (can be a mesh or simple primitive). Render: Particles emit from the full mesh of the render object (usually static or animated mesh). Generally more CPU-intensive than emitting from physics. <p>Default value: None</p>
Attach Form	<p>When Attach Type is not set to None, specifies the elements of the geometry (box or mesh) that particles emit from.</p> <ul style="list-style-type: none"> Vertices: Emit randomly from the vertices of the geometry. Most efficient form of mesh emission. Edges: Emit randomly from the edges of the geometry. Useful for effects on breaking element pieces. Surface: Emit randomly from the surfaces (faces) of the geometry. Volume: Emit randomly inside the volume of the geometry. <p>Default value: Vertices</p>
Count:	<p>The total number of particles at any one time that are active. Determines the emission rate (Count / Particle Lifetime). Can set a Random value and the Emitter Strength curve.</p> <p>Value range: 0+</p> <p>Default value: 5</p>

Parameter Function	Description
Maintain Density:	<p>Increase emission rate (and particle count) when emitter moves to maintain the same spatial density as when motionless. The increase can be scaled from 0 to 1.</p> <ul style="list-style-type: none"> • Reduce Alpha: When Maintain Density is active, this reduces particle alpha correspondingly, to maintain the same overall emitter alpha. <p>Value range: 0+</p>
Continuous:	<p>If false, all particles are emitted at once, and the emitter then dies. If true, particles are emitted gradually over the Emitter Lifetime. If true, and Emitter Lifetime = 0, particles are emitted gradually, at a rate of Count / Particle Lifetime per second, indefinitely.</p> <p>Default value: False</p>
Spawn Delay:	<p>Delays the start of the emitter for the specified time. Useful to delay sub-effects relative to the overall emitter creation time. Can set a Random value.</p> <p>Value range: 0+</p> <p>Default value: 0</p>
Emitter Lifetime:	<p>If Continuous = true, specifies the lifetime of the emitter. Emitter Lifetime does not apply to non-continuous effects, which always disappear as soon as they have emitted all of their particles. Can set a Random value.</p> <p>Value range: 0+</p> <p>Default value: 0 (infinite lifetime)</p>
Pulse Period:	<p>If greater than 0, the emitter restarts repeatedly at this interval. Can set a Random value.</p> <p>Value range: any</p> <p>Default value: 0</p>
Position Offset:	<p>X, Y, and Z values define the spawning position away from the emitter itself, in emitter space.</p> <p>Value range: any</p> <p>Default values: 0,0,0</p>
Random Offset:	<p>X, Y, and Z values define the range of a random spawning box, in both directions away from the position offset.</p> <p>Value range: any</p> <p>Default values: 0,0,0</p>

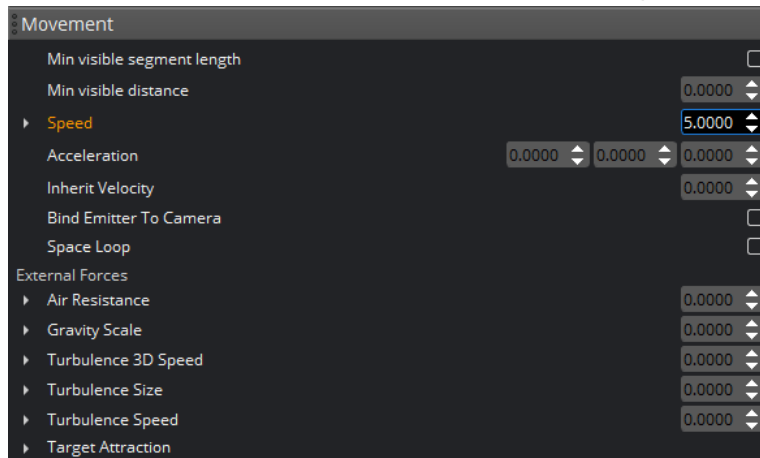
Parameter Function	Description
Offset Roundness:	<p>Fraction of spawning volume corners to round.</p> <p>Value range: 0 (box shape) to 1 (ellipsoid shape)</p> <p>Default value: 0</p>
Offset Inner Fraction:	<p>Ratio of inner to outer spawning volume.</p> <p>Value range: 0 (spawn within entire volume) to 1 (spawn only at surface)</p> <p>Default value: 0</p>
Focus Angle:	<p>The number of degrees to rotate from the Y axis.</p> <p>Value range: 0 (straight up) to 180 (straight down).</p> <p>Default value: 0</p>
Focus Azimuth:	<p>The number of degrees to rotate the new axis about the Y axis.</p> <p>Value range: any (0, 360 = North, 90 = West, 180 = South, 270 = East).</p> <p>Default value: 0</p>
Focus Camera Direction:	<p>Set focus direction to face camera. Can set a Random value and the Emitter Strength curve.</p> <p>Value range: 0-1</p> <p>Default value: 0</p>
Focus Rotates Emitter:	<p>Default value: false</p>
Emit Offset Direction:	<p>If true, change each particles emission direction to be aligned with its offset from the origin.</p> <p>Default value: false</p>
Emit Angle:	<p>The angle deviation of an emitted particle from the default focus (+Y) axis. (0 = straight up, 90 = horizontal, 180 = straight down). This is the maximum angle from the focus. Can set a Random value (determines minimum angle) and the Emitter Strength curve.</p> <ul style="list-style-type: none"> • To emit in all directions, set Emit Angle = 180, Random = 1. • To emit in the top hemisphere, set Emit Angle = 90, Random = 1. • To emit in a horizontal circle, set Emit Angle = 90, Random = 0. <p>Value range: 0-180</p> <p>Default value: 0</p>
Orient to Velocity:	<p>Forces the particle X-axis aligned to the velocity direction. Use Rotation parameters to rotate it further.</p> <p>Default value: false</p>

Parameter Function	Description
Curvature:	<p>Sets how far the vertex normals for Facing=Camera particles are bent into a spherical shape, which affects lighting.</p> <p>Value range: 0 (flat) to 1 (hemispherical shape)</p> <p>Default value: 1</p>

Particles Attribute

Particle Editor is in preview release and is subject to change.

Parameters in this attribute control the basic appearance of the particle. This attribute should be set up first as it includes the **Texture** slot, which is used for most particles.



Particles Attribute Parameters

Parameter Function	Description
Particle Life Time	<p>The lifetime of individual particles. Even after the emitter's lifetime has expired, spawned particles live out their own lifetime.</p> <p>Value range: 0+</p> <p>Default value: 0</p>
Remain While Visible	<p>Particles do not die until the entire emitter is out of view.</p> <p>Default value: false</p>
Facing	<p>Applies only to 2D particles. Determines how the sprite is oriented in space. Texture orientation is further modified by rotational parameters:</p> <ul style="list-style-type: none"> • Camera (default): Faces the viewer, texture X and Y aligned with screen X and Y. In this mode only, particles are assumed to represent spherical objects, and are lit accordingly (see Curvature below). In all other modes, particles are lit as flat polygons.

Parameter Function	Description
	<ul style="list-style-type: none"> • CameraX: Rotates about local Y axis only, to face camera as much as possible. • Free: Rotates freely in 3D. (Remember to give it some rotation; the default orientation is equal to the emitter's.) • Velocity: Faces direction of movement. • Water: Faces upward, moved and aligned to nearest water plane. • Terrain: Faces upward, moved and aligned to nearest terrain location. • Decal: Renders the particle as an actual deferred decal, projected onto the nearest surface. (The Thickness parameter controls the projection depth.) (Only works with Materials, does not work with textures.) <p>Default value: camera</p>
Material	<p>Use to open the Asset Browser and assign a material used for 2D sprite particles.</p> <p>Default value: empty</p>
Blend Type	<p>Applies only to 2D particles. Determines how the sprite blends with the background.</p> <ul style="list-style-type: none"> • Alpha Based: Final Color = Particle Color * Particle Alpha + Background Color * (1 - Particle Alpha). • Additive: Final Color = Particle Color + Background Color. • Multiplicative: Final Color = Particle Color * 2 * Background Color.
Texture	<p>Use to open the Asset Browser and assign a texture used for 2D sprite particles. Displays a preview of the texture when the mouse cursor is over the input box.</p> <p>Default value: empty</p>

Parameter Function	Description
Texture Tiling	<p>Splits the texture into tiles, for variation and animation:</p> <ul style="list-style-type: none"> • Tiles X, Y: Number of tiles the texture is split into. Value range: 1-256 Default value: 1 • First Tile: The first of the range of tiles used by the particle. Value range: 0-255 Default value: 0 • Variant Count: Number of consecutive tiles in the texture the particle randomly selects from. Value range: 1-256 Default value: 1 • Anims Frame Count: How many tiles make up an animation sequence. Variant Count and Anim Frames Count can be used together. For example, if Variant Count = 2 and Anim Frames Count = 8, then the particle randomly chooses between using tiles 0 through 7, or 8 through 15, as an animated sequence. Value range: 1-256 Default value: 1 • Anim Framerate: Frames per second for the animation. If 0, then the animation runs through one sequence in the particle lifetime. Value range: 0+ Default value: 1 • Anim Cycle: This parameter has three values: <ul style="list-style-type: none"> • Once: Animation plays once, and holds on the last frame • Loop: Animation loops indefinitely • Mirror: Animation alternates cycling forward and backward indefinitely Default value: Once • Anim Blend: Renders the particle blended between the two adjacent anim frames. This has a performance impact. Default value: false • Flip Chance: Specifies the fraction of particles that are rendered and mirrored in texture X. Value range: 0-1 Default value: 0 • Anim Curve: Used to set the curve.

Parameter Function	Description
Color	<p>Used to select the color to apply to the particle:</p> <ul style="list-style-type: none">• Random: How much a particle's initial color varies downward from the default. 0 = no variation, 1 = random black to default. <p>Value range: 0-1</p> <ul style="list-style-type: none">• Random Hue: Causes the Random color variation to occur separately in the 3 color channels. If false, variation is in luminance only. <p>Default value: false</p> <ul style="list-style-type: none">• Emitter Strength: Define the color of the particle over the emitter's lifetime. Double-clicking opens the Gradient Editor.• Particle Age: Defines the color of the particle over the particle's lifetime. Double-clicking opens the Gradient Editor.

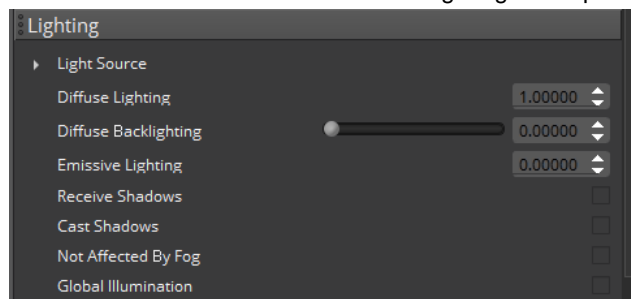
Parameter Function	Description
Alpha clip	<p>A set of parameters to customize how the particle Alpha value controls opacity or alpha test values. Each parameter below has 2 values, corresponding to their values when particle Alpha = 0 and 1. They are interpolated for each particle with its Alpha value, and then used in the shader with the following equation: $\text{FinalOpacity} = \text{saturate}(\text{TextureAlpha} - \text{SourceMin}) / \min(\text{SourceWidth}, 1 - \text{SourceMin}) * \text{Scale}$</p> <ul style="list-style-type: none"> • Scale: A multiplier for the final alpha value. Defaults to (0, 1), so that particle Alpha directly scales final opacity. Value range: 0+ • Source Min: Specifies the minimum source (texture) alpha to be rendered (alpha test); values below become transparent. Defaults to (0, 0), corresponding to no alpha test. Value range: 0+ • Source Width: Specifies the feathering range of alpha clipping; 0 specifies hard-clipping, 1 soft-clipping. Defaults to (1, 1), corresponding to full utilization of texture alpha. Value range: 0+ <ul style="list-style-type: none"> • Default: Alpha controls opacity, no alpha clipping: Scale = (0, 1), Source Min = (0, 0), Source Width = (1, 1). • Hard clipping at texture alpha = C, no feathering: Scale = (1, 1), Source Min = (C, C), Source Width = (0, 0). • Hard clipping, controlled by particle alpha: Scale = (1, 1), Source Min = (0, 1), Source Width = (0, 0). • Feathered clipping, with width F, controlled by particle alpha: Scale = (1, 1), Source Min = (0, 1), Source Width = (F, F). • Soft clipping, test value controlled by particle alpha: Scale = (1, 1), Source Min = (0, 1), Source Width = (1, 1) • Clipping and opacity scale, controlled by particle alpha: Scale = (0, 1), Source Min = (0, 1), Source Width = (1, 1) <p>Default value: 0 for all</p>
Tessellation	<p>If supported by hardware (DirectX 11 minimum), enables tessellation, rendering more vertices within the sprite. This is useful when Receive Shadows is set, increasing the resolution of shadows; or when Tail Length or Connection are set, creating smoother curves in connected particles. This also helps for receiving light from point lights, as the lighting is more accurate.</p> <p>Default value: false</p>

Parameter Function	Description
Soft Particles	Applies rendering that softens the intersection between sprites and nearby objects to prevent unnatural seams. Slightly more expensive, so use sparingly on particles that need it, such as smoke. Use the Softness sub-parameter to define the amount of rendering applied. Default value: false
Geometry	Opens the Preview window to select a 3D object to use for the particles. Default value: empty
Geometry in Pieces	If Whole is not selected, and the Geometry asset contains multiple sub-objects, the geometry is emitted in split-up pieces, one set per particle Count , originating at each piece's location in the asset. Default value: Whole
Geometry No Offset	For geometry particles, uses the geometry pivot for centering. Default value: false
Octagonal Shape	Renders sprites as octagons instead of quads, reducing pixel cost. Only use with textures that fit within an octagon, otherwise clipping occurs. Default value: false

Lighting Attribute

Particle Editor is in preview release and is subject to change.

Parameters in this attribute control the lighting of the particle.



Lighting Attribute Parameters

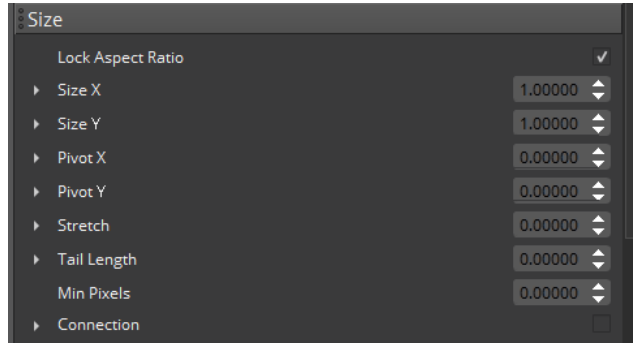
Parameter Function	Description
Light Source	Causes each particle to create a deferred light, where color is equal to the Color value. <ul style="list-style-type: none"> Affects This Area Only – For use with Clip Volumes. When enabled, the particle lights do not exceed the volume boundary.

Parameter Function	Description
	<ul style="list-style-type: none"> • Radius – Radius of the light. Can set a Random value and Emitter Strength and Particle Age curves. Value range: 0+ • Intensity – Intensity of the light. Can set a Random value and Emitter Strength and Particle Age curves. Value range: 0+ <p>Default values: false, 0, 0</p>
Diffuse Lighting	<p>Multiplier to the particle color for dynamic (diffuse) lighting.</p> <p>Value range: 0+</p> <p>Default value: 1</p>
Diffuse Backlighting	<p>Fraction of diffuse lighting that is applied to unlit particle directions.</p> <p>Value range: 0 (standard diffuse, normals facing the light are lit the most) to 1 (omnidirectional diffuse, light affects all normals equally).</p> <p>Default value: 0</p>
Emissive Lighting	<p>Multiplier to the particle color for constant emissive lighting. When you add a value, this can make a particle appear as if it's glowing.</p> <p>Value range: 0+</p> <p>Default value: 0</p>
Receive Shadows	<p>Allows shadows to be cast on the particles.</p> <p>Default value: false</p>
Cast Shadows	<p>Allows particles to cast shadows (Currently only for geometric particles).</p> <p>Default value: false</p>
Not Affected By Fog	<p>Causes particles to ignore scene fog.</p> <p>Default value: false</p>
Global Illumination	<p>Allows the particle to receive global illumination from the environment.</p> <p>Default value: false</p>

Size Attribute

Particle Editor is in preview release and is subject to change.

Parameters in this attribute control the size and shape of the sprite.



For the **Size**, **Pivot**, and **Stretch** parameters, you can set a **Random** value and **Emitter Strength** and **Particle Age** curves.

Size Attribute Parameters

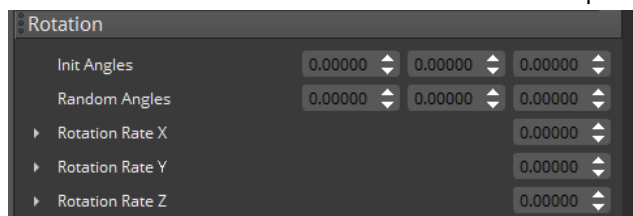
Parameter Function	Description
Lock Aspect Ratio	Maintain particle aspect ratio. Default value: false
Size X, Y	For 2D particles, the world sprite radius. Value range: 0+ Default value: 1
Pivot X, Y	Moves the pivot point of the sprite. Value range: -1 to +1 Default value: 0 (texture center)
Stretch	The amount of stretch applied to the particle in the direction of travel, in seconds (based on current velocity). Stretches in both directions by default. Offset Ratio: Adjusts the center of stretching. 0 = stretch both directions, 1 = stretch backward only, -1 = stretch forward only. Value range: 0+ Default value: 0
Tail Length	Length of particle's tail in seconds. Particle texture is stretched out through the tail. Value range: 0+ Tail Steps: Number of segments for tail. A higher number produces smoother tail curves for non-linear-moving particles. Value range: 0+ Default value: 0
Min Pixels	Adds this many pixels to particles true size when rendering. This is useful for important effects that should always be visible even at distance.

Parameter Function	Description
	Value range: 0+ Default value: 0
Connection	<p>Causes all particles to be rendered in a connected line, in sequence. Emission sequences separated by a Pulse Period produce separate polygons. Indirect child effects produce a separate polygon for each parent particle.</p> <ul style="list-style-type: none"> • Connect To Origin – Additionally connect the newest particle to the emitter origin, with the parameters of a particle of age 0. • Texture Mapping – This and the next parameter specify how textures are repeated over the stream. <ul style="list-style-type: none"> • PerParticle sets a default frequency of one texture per particle. • PerStream sets a default frequency of one texture stretched over the whole stream. • Texture Mirror – Option which causes adjacent texture tiles to alternate direction; if false, they wrap at each repetition. Default = true. • Texture Frequency – Multiplies the texture repeating frequency specified above. Can be less than 0 or greater than 0, which determines texture direction. <p>Default value: false</p>

Particle Rotation Parameters

Particle Editor is in preview release and is subject to change.

Parameters in this attribute control the rotation of the particle.



Rotation Attribute Parameters

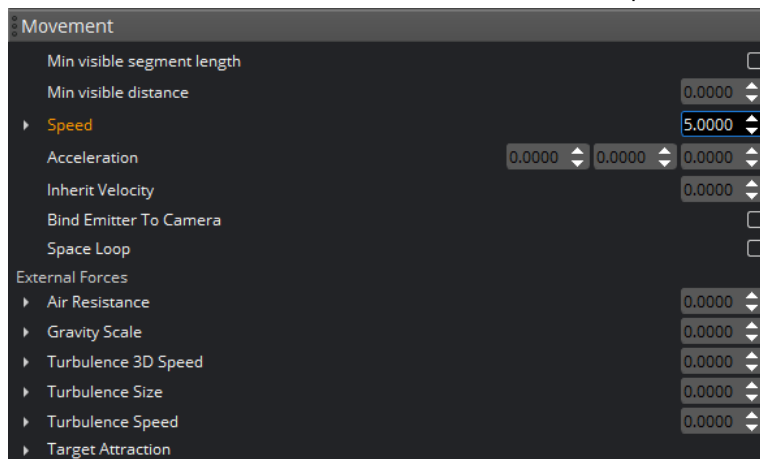
Parameter Function	Description
Init Angles	<p>X, Y, and Z values define the initial angle applied to the particles upon spawning, in degrees. For Facing = Camera particles, only the Y axis is used, and refers to rotation in screen space. For 3D particles, all three axes are used, and refer to emitter local space.</p> <p>Value range: any Default value: 0</p>

Parameter Function	Description
Random Angles	X, Y, and Z values define the random variation (bidirectional) to Init Angles , in degrees. Value range: 0+ Default value: 0
Rotation Rate X, Y, Z	Constant particle rotation, in degrees/second. The axes are the same as for Init Angles . Can set a Random value and Emitter Strength and Particle Age curves. Value range: any Default value: 0

Movement Attribute

Particle Editor is in preview release and is subject to change.

Parameters in this attribute control the movement of the sprite.



For the **Air Resistance**, **Gravity Scale**, **Turbulence 3D Speed**, **Turbulence Size**, and **Turbulence Speed** parameters, you can set a **Random** value and **Emitter Strength** and **Particle Age** curves.

Movement Attribute Parameters

Parameter Function	Description
Min visible segment length	Enables and disables the feature.
Min visible distance	The minimal distance between the start and end of a trail segment. Segments smaller than this value will become transparent.
Speed	The initial speed of particles. You can set a Random value and Emitter Strength curve. Value range: any Default values: 5

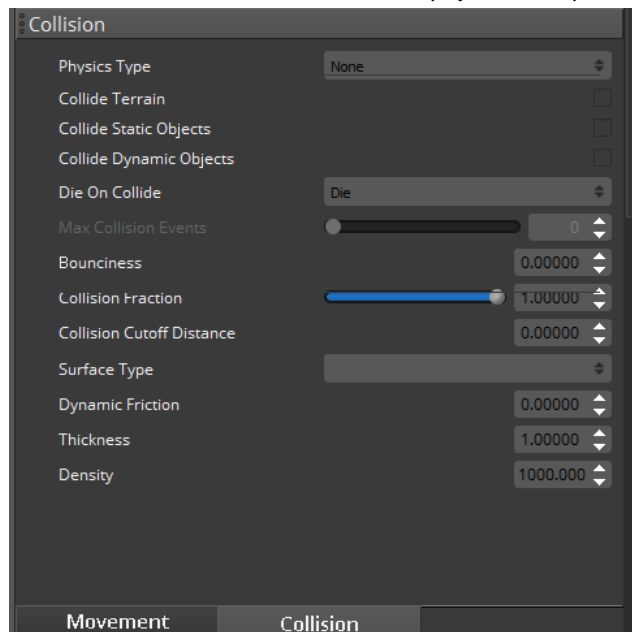
Parameter Function	Description
Acceleration	X, Y, and Z values define the constant acceleration applied to particles in world space. Value range: any Default values: 0,0,0
Inherit Velocity	What fraction of initial velocity is inherited from the particle's parent. For indirect particles, the parent particle's velocity is inherited. For direct particles, the emitter's velocity is inherited. Value range: any Default value: 0
Bind Emitter to Camera	Forces the emitter to relocate to the main camera's position. Useful (with Space Loop) for making a rain or snow effect, which the player cannot pass by. Default value: false
Space Loop	Particles loop within a region around the camera, defined by Camera Min/Max Distance (under the visibility tab). This is useful to make rain or snow effect, which has an effective infinite spawning area. Default value: false
Air Resistance	Value range: Default value:
Gravity Scale	Multiple of world gravity to apply to particles. Most physicalized particles should be set to 1 (use Air Resistance to provide drag). Set to a negative value for buoyant particles such as smoke. Value range: any Default values: 0,0,0
Turbulence3DSpeed	Adds a 3D random turbulent movement to the particle, with the specified average speed. Value range: 0+ Default value: 0
Turbulence Size	Adds a spiral movement to the particles, with the specified radius. The axis of the spiral is set from the particle's velocity. Value range: 0+
Turbulence Speed	When Turbulence Size is greater than 0, the angular speed, in degrees/second, of the spiral motion. Value range: any Default value: 0

Parameter Function	Description
Target Attraction	<p>Specifies how particles behave if the emitter is attached to a target. By default, all particles are attracted to any target the emitter is linked to. These parameters customize that behavior.</p> <ul style="list-style-type: none"> • Target <ul style="list-style-type: none"> • External = Particles attracted to a target entity, if the emitter is linked to one (default). • OwnEmitter = Particles are attracted to their own emitter's origin. • Ignore = Particles ignore any external attractor. • Extend Speed – Particles speed up to reach the target in their lifetime. Otherwise, they move at a real-world natural speed toward the target, and may not reach it. • Shrink – Particles shrink as they approach the target. • Orbit – Particles orbit around target when reached. Otherwise, they disappear into the target. • Radius – Distance from the target that particles either orbit around, or disappear. You can set a Random value and Emitter Strength and Particle Age curves. <p>Value range: any</p>

Collision Attribute

Particle Editor is in preview release and is subject to change.

Parameters in this attribute control the physical setup for the particles.



Collision Attribute Parameters

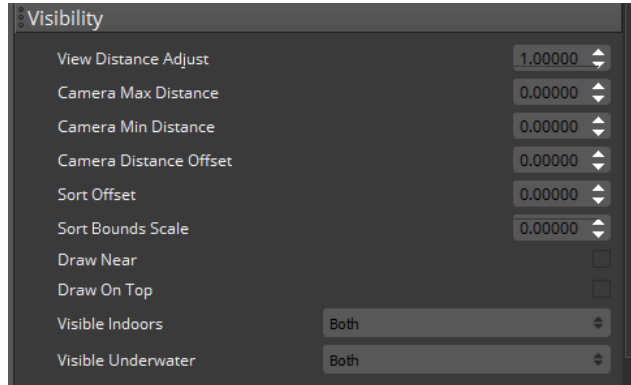
Parameter Function	Description
Physics Type	<p>How the particle interacts physically.</p> <ul style="list-style-type: none"> • None – No collisions or other physics. Default. • SimpleCollision – Particle collides with the static environment using simple physics. This is the most simple mode. • SimplePhysics – Particle created as an entity in the physics system, and collides using a spherical particle model. • RigidBody – Particle created as an entity in the physics system, and collides using the full geometry. A geometry asset must be set to the physicalized model in engine for this particle. This is most expensive mode.
Collide Terrain	<p>Includes terrain in particle collisions.</p> <p>Default value: false</p>
Collide Static Objects	<p>Includes non-terrain, static objects in particle collisions. This is expensive.</p> <p>Default value: false</p>
Collide Dynamic Objects	<p>Includes non-terrain, dynamic objects in particle collisions. This is expensive.</p> <p>Default value: false</p>
Die on Collide	<p>Upon impact with the static environment, the particle dies.</p> <ul style="list-style-type: none"> • Die: • Ignore: • Stop: <p>Default value: Die</p>
Max Collision Events	<p>Limits the number of collisions the particle can have in its physics simulation. Only affects particles that have their Physics Type set to Rigid Body.</p> <p>Value range: 0-255</p> <p>Default value: 0</p>
Bounciness	<p>Controls the elasticity for collision response. Overridden by Surface Type, if set. (Special value: if -1, particle dies on first collision). Only affects particles that have their Physics Type set to Simple Collision.</p> <p>Value range: any</p> <p>Default value: 0</p>
Collision Fraction	<p>Fraction of emitted particles that actually perform collisions.</p> <p>Value range: 0-1</p>

Parameter Function	Description
	Default value: 1
Collision Cutoff Distance	Maximum distance from camera at which collisions are performed (0 = infinite). Value range: 0+ Default value: 0
Surface Type	Select from a variety of surface material types for collision behavior. If set, overrides Bounciness and Dynamic Friction below. Default value: none
Dynamic Friction	The coefficient of dynamic friction. Overridden by Surface Type if set. Only affects particles that have their Physics Type set to Simple Collision . Value range: 0+ Default value: 1
Thickness	Control the fraction of the particle's visible radius to use for the physical radius. Only affects particles that have their Physics Type set to Simple Physics . Value range: 0+ Default value: 1
Density	Control the density of particle, in kg/m ³ . An example of a physically correct value is Water = 1000. Only affects particles that have their Physics Type set to Simple Physics or Rigid Body . Value range: 0+ Default value: 1000

Visibility Attribute

Particle Editor is in preview release and is subject to change.

Parameters in this attribute control the visibility of the particles.



Visibility Attribute Parameters

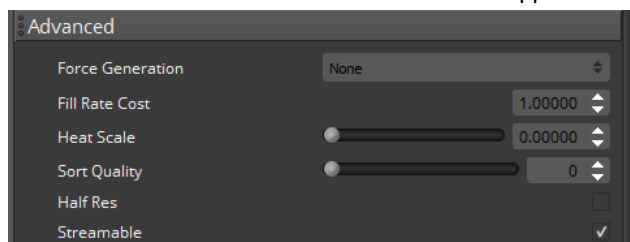
Parameter Function	Description
View Distance Adjust	Multiplier to the automatically computed fade-out camera distance. Range: 0+ Default value: 1
Camera Min/Max Distance	The camera range that particles render in. Defaults are 0, specifying unlimited range. Range: 0+ Default value: 0
Camera Distance Offset	Offsets the emitter away from the camera. Range: any Default value: 0
Sort Offset	Bias the distance used for sorting. Can be used to customize the sort order within an emitter tree: By default, sub-emitters render in the order they are listed in the effect. A bias of 0.01 or greater overrides that order. Larger biases can be used to adjust the sorting order with respect to other transparent objects in the level. Range: any Default value: 0
Sort Bounds Scale	Specify point in emitter for sorting; 1 = bounds nearest, 0 = origin, -1 = bounds farthest. Range: any Default value:0
Draw Near	Render particles in a near 1st-person space (with weapons etc). Default value: false
Draw On Top	Render particles on top of everything (no depth test). Default value: false
Visible Indoors	For use in VisAreas: <ul style="list-style-type: none"> • If_False – Hides particles when indoors. • If_True – Hides particles when outdoors. • Both – Show particles always.

Parameter Function	Description
Visible Underwater	For use with the Ocean and with Water Volumes: <ul style="list-style-type: none"> • If_False – Hides particles when under water. • If_True – Hides particles when above water. • Both – Show particles always.

Advanced Attribute

Particle Editor is in preview release and is subject to change.

Parameters in this attribute include advanced appearance and optimization settings.



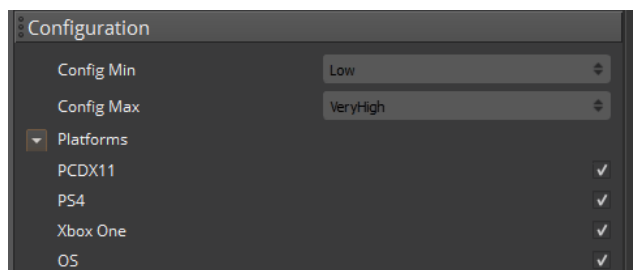
Advanced Attribute Parameters

Parameter Function	Description
Force Generation	Adds an additional force generated by the emitter: <ul style="list-style-type: none"> • None – Does not add any additional force. • Wind – Creates a physical wind force, approximately following the velocity, direction, volume, and timing of the emitter's particles. This wind affects all particles and objects in its region, except particles in the emitter group. Setting the emitter's Speed to negative creates the wind force in the opposite direction, which can be used to create a vacuum force. • Gravity – Creates a physical gravity force, similar to the wind, but creates a gravitational acceleration force, instead of wind velocity. <p>Default value: None</p>
Fill Rate Cost	Multiplier to this emitter's contribution to total fill rate, which affects automatic culling of large particles when the global limit is reached. Set this > 1 if this effect is relatively expensive or unimportant. Set this <, or 0, if the effect is an important one which should not experience automatic culling. <p>Value range: 0+</p> <p>Default value: 1</p>
Heat Scale	Multiplier to thermal vision. Range: 0-4 <p>Default value: 0</p>

Parameter Function	Description
Sort Quality	<p>Specifies more accurate sorting of new particles into emitter's list. Particles are never re-sorted after emission, to avoid popping resulting from changing render particle order. They are sorted only when emitted, based on the current main camera's position, as follows:</p> <ul style="list-style-type: none">• 0 (default, fastest): Particle is placed at either the front or back of the list, depending on its position relative to the emitter bounding box center. Doesn't add any additional force.• 1 (medium slow): Existing particles are sorted into a temporary list, and new particles do a quick binary search to find an approximate position.• 2 (slow): Existing particles are sorted into a temporary list, and new particles do a full linear search to find the position of least sort error. <p>Value range: 0-2 Default value: 0</p>
Half Res	<p>Render particles in separate a half-resolution pass, reducing rendering cost.</p> <p>Default value: False</p>
Streamable	<p>Texture or geometry assets are allowed to stream from storage, as normal.</p> <p>Default value: True</p>
Volume Fog	<p>Enables fog density injection.</p> <p>Default value: False</p>
Volume Thickness	<p>Controls volume thickness.</p> <p>Default value: 1.0000</p>

Configuration Attribute

Particle Editor is in preview release and is subject to change.



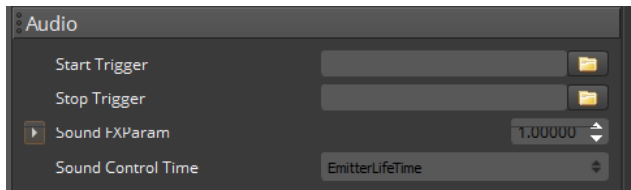
Configuration Attribute Parameters

Parameter Function	Description
Config Min	The minimum system configuration level for the effect. If the config is lower than what is set here, the item is not displayed. Select from Low , Medium , High , or VeryHigh . Default value: Low
Config Max	The maximum system configuration level for the effect. If the config is higher than what is set here, the item is not displayed. Select from Low , Medium , High , or VeryHigh . Default value: VeryHigh
Platforms	Defines what playform the effect should be used with. Default: All true <ul style="list-style-type: none"> • PCDX11 • PS4 • XBox One • OS Default value: all checked (true)

Audio Attribute

Particle Editor is in preview release and is subject to change.

Parameters in this attribute handle what sounds are emitted by the particle system and when.



Audio Attribute Parameters

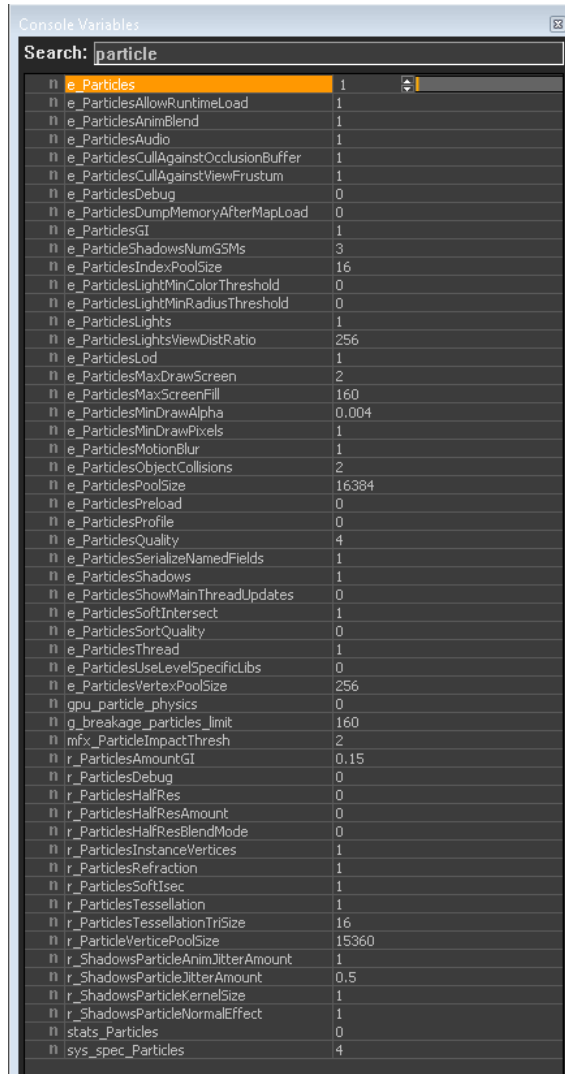
Parameter Function	Description
Start Trigger	Opens a window to select the start trigger sound asset to play with the emitter.
Stop Trigger	Opens a window to select the stop trigger sound asset to play with the emitter.
Sound FXParam	Modulate value to apply to the sound. Its effect depends on how the individual sound's particlefx parameter is defined. Depending on the sound, this value might affect volume, pitch, or other attributes. Can set a Random value and Emitter Strength curve. Value range: 0+

Parameter Function	Description
	Default value: 1
Sound Control Time	<ul style="list-style-type: none"> • EmitterLifeTime – Plays for the length of the emitter's lifetime. • EmitterExtendedLifeTime --Plays for the length of the emitter's lifetime plus all particle's lifetimes (until all particles die). • EmitterPulsePeriod – Plays for the length of the pulse period.

Particle Debugging

Particle Editor is in preview release and is subject to change.

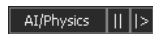
You can use the following console variables to monitor and debug particle system issues. To access **Console Variables**, click the **x** button in the **Console** window at the bottom of your screen.



Physics System

The physics engine of Lumberyard provide a realistic simulation of physical systems, such as collision detection and response, and dynamics for rigid bodies, living entities, dead entities (ragdoll), attachments, cloth, hair, particles, wind, and water.

The selection strip at the bottom of Lumberyard Editor features controls to enable Physics. The **AI/Physics** button turns physics simulation on and off, and allows you to test physics behavior directly without entering game mode.



The pause and next step buttons are used for stepping through the physics system one frame at a time for debugging. To use these correctly, first click the pause button, then click the **AI/Physics** button, then click the next step button.

Make sure to disable the pause button again to return to normal operation.

For information on physics entities, see [Physics Entities \(p. 470\)](#).

For information on character physics, see [Physicalizing Characters \(Ragdoll\) \(p. 157\)](#).

For information on character attachment physics (simulations), see [Secondary Animations \(Simulations\) \(p. 221\)](#).

For information on physics flow graph nodes, see [Physics Nodes \(p. 696\)](#).

Topics

- [Physics Proxies \(p. 981\)](#)
- [Sounds and Physics \(p. 983\)](#)
- [Debugging Physics \(p. 984\)](#)

Physics Proxies

The physics proxy is the geometry that is used for collision detection. It can be part of the visible geometry or linked to it as a separate node. Usually the physics proxy geometry is a simplified version of the render geometry but it is also possible to use the render geometry directly for physics. However,

for performance reasons the collision geometry should be kept as simple as possible since checking for intersections on complex geometry is very expensive, especially if it happens often.

A physics proxy is set up in your DCC tool. The only setup needed in Lumberyard is assigning the surface type to the physics proxy and the render geometry and assigning the **NoDraw** shader in the Material Editor. The surface type gives information about sound and the particle effects of your surface. Make sure that no textures are assigned to your proxy sub material. The physics proxy does not render in Lumberyard Editor except in debug view. Even if you assign an Illum shader it stays invisible. To reload the physics proxy, reload your object, delete it, and then undo delete.

The physics proxy can be part of the render object (in 3ds Max as an Element) or as a separate object, linked to the render object.

Physics proxies are only created for level of detail LOD0. Every successive LOD will automatically take the proxy from LOD0. This also occurs if different config spec quality settings are used, such as **Lowspec** for example.

Geometry Guidelines and Best Practices

The following are some guidelines and best practices that should be taken into consideration when working with physics proxies.

The physics proxies of environment objects such as fences, crates, containers, trees, rocks, ladders, and stairs should be as simple as possible. Crates and fences can usually be approximated with a simple box with 6 sides (12 triangles). The top of stairs should usually be simple ramps, resulting in just 2 triangles. More organic or irregularly shaped objects like rocks and trees can still be approximated with a fairly simple hull by allowing slight and acceptable inaccuracies between the render mesh and the physics proxy.

The physics proxy should not have open edges. Open edges can confuse the physics engine and have a negative effect on performance. It is helpful to assign a bold color to the proxy in order to keep track of it.

Avoiding geometric complexity for physics proxies is not only important to reduce redundant memory requirements and physics computations, but also for making player movement smoother. The more complicated a proxy is, the more memory it takes and the more performance is lost when checking collisions against its polygons. This affects both single player and multiplayer games, including the performance of a dedicated server. Besides the performance issues, a complex proxy with a lot of concavity increases chances that the player can get stuck or bounce undesirably against the proxy.

An ideal proxy is always a primitive, such as a box, sphere, capsule, or cylinder. Lumberyard recognizes primitives from meshes but the default tolerance is very low. In order to force recognition, put the corresponding keyword (such as "box" or "sphere") in the node's user-defined properties. Meshes with several surface types cannot be turned into primitives. Primitives should be considered as an option even for more complex objects. In most cases it is preferable to have a multi-part object (**Merge Nodes** disabled) with primitive parts instead of a single-part mesh object.

The physics proxy is used for blocking character movement as well as first-pass tracing of projectiles. If a hit is detected against the physics proxy, projectile impact and decal locations are refined using the render mesh. The render mesh should be fully encapsulated by the physics proxy, so that the player camera does not intersect the render geometry and first-pass projectile culling does not miss the physical part of the object, even if it hits the visual part of the object. You can also create a special raytrace proxy that can be used for projectiles. This would allow the main proxy to not have to encapsulate the render mesh and thus the proxy could be even simpler.

Debugging Physics Proxy Issues

You can use the following two console variables to help debug physics proxy issues:

p_draw_helpers

```
Same as p_draw_helpers_num, but encoded in letters
Usage [Entity_Types]_[Helper_Types] - [t|s|r|R|l|i|g|a|y|e]_[g|c|b|l|t|#]
Entity Types:
t - show terrain
s - show static entities
r - show sleeping rigid bodies
R - show active rigid bodies
l - show living entities
i - show independent entities
g - show triggers
a - show areas
y - show rays in RayWorldIntersection
e - show explosion occlusion maps
Helper Types
g - show geometry
c - show contact points
b - show bounding boxes
l - show tetrahedra lattices for breakable objects
j - show structural joints (forces translucency on the main geometry)
t(#) - show bounding volume trees up to the level #
f(#) - only show geometries with this bit flag set (multiple f's stack)
Example: p_draw_helpers larRis_g - show geometry for static, sleeping,
active, independent entities and areas
```

e_PhysProxyTriLimit

The `e_PhysProxyTriLimit` console variable shows the maximum allowed triangle count for physics proxies. If you notice your assets wrapped in "No Collisions, Proxy Too Big!" messages, then the physics proxy for that asset is over the triangle count specified in `e_PhysProxyTriLimit`.

Sounds and Physics

The game environment is very interactive, with objects moving, colliding, and breaking. When two materials touch each other, the collision can generate a sound.

Physical events in the game can send parameter information to the sound event. Lumberyard sends the speed and mass of the collision, which then gets passed to the sound event.

For example, an object's speed will cause the collision to change pitch, while an object's mass determines the volume and sound definition used. A smaller mass reduces a sound's roll-off radius. Small collisions won't be heard from as far away as larger collisions.

The interaction between two materials is specified in the `MaterialEffects.xml` file located in the `\Game\Libs\MaterialEffects` directory.

Using this file, Lumberyard looks up actions to be taken on interaction. Each entry in the file table contains text pointing to a description of the sound effect. These effects are described in the `\FXLibs` subfolder.

The following console variables can be used for debugging physics sound events:

- `mfx_Debug` - Enables `MaterialEffects` debug messages (1=Collisions, 2=Breakage, 3=Both).
- `mfx_Enable` - Enables `MaterialEffects`.
- `mfx_EnableFGEffects` - Reloads `MaterialEffects` flow graphs.

- `mfx_ReloadFGEffects` - displays profiling information for the shaders.
- `mfx_ReloadFGEffects` - Reloads the MaterialEffects file.

Debugging Physics

The `p_draw_helpers` console variable is useful for debugging physics issues. The syntax is as follows:

```
p_draw_helpers entity type_helper type
```

Entity Types:

```
t - show terrain  
s - show static entities  
r - show sleeping rigid bodies  
R - show active rigid bodies  
l - show living entities  
i - show independent entities  
g - show triggers  
a - show areas  
y - show rays in RayWorldIntersection  
e - show explosion occlusion maps
```

Helper Types

```
g - show geometry  
c - show contact points  
b - show bounding boxes  
l - show tetrahedra lattices for breakable objects  
j - show structural joints (forces translucency on the main geometry)  
t(#) - show bounding volume trees up to the level #  
f(#) - only show geometries with this bit flag set (multiple f stacks)
```

For the following example:

```
p_draw_helpers larRis_g
```

would show geometry for static, sleeping, active, independent entities, and areas.

In addition, the `p_debug_joints` console variable, if set to 1, logs tensions of breakable objects at their weakest locations.

Project Configurator

The Project Configurator is in preview release and is subject to change.

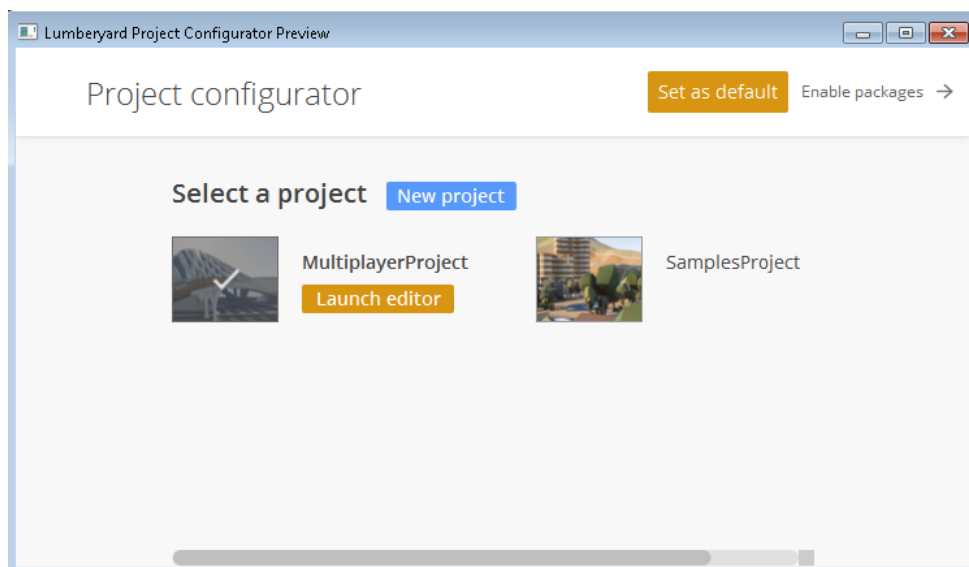
The Project Configurator is a standalone application that allows you to specify to the Waf build system which game projects and assets (Gems) to include in a game build. With it, you can create new projects, save active projects, and enable, disable, or create new Gems. For information about Waf build system, see [Waf Build System \(p. 1318\)](#). For information about Gems, see [Modular Gems System \(p. 779\)](#).

Note

Before you can run the Project Configurator, you must first run Lumberyard Setup Assistant and close Lumberyard Editor.

The following files should be set to editable for the Project Configurator to work:

- `project_asset_folder\gems.json`
- `project_asset_folder\game.cfg`
- `engine_root_folder\bootstrap.cfg`
- `engine_root_folder\dev\game_project_folder\project.json`



Topics

- [Creating and Launching Game Projects \(p. 986\)](#)
- [Enabling Gems \(p. 986\)](#)
- [Using Lmbr.exe \(p. 987\)](#)
- [Troubleshooting the Project Configurator \(p. 988\)](#)

Creating and Launching Game Projects

The Project Configurator is in preview release and is subject to change.

After creating a new project, you must run `lmbr_waf configure` from a command line and build the project before opening Lumberyard Editor with that project.

To create a new game project

1. Go to `engine_root\dev\Bin64\`, then open `ProjectConfigurator`.
2. Choose **New project**.
3. Enter a name and choose **Create project**. Only alphanumeric characters are allowed; no special characters or whitespaces are allowed in the name.
4. Select the new project, and choose **Set as default** to make it the default that Lumberyard Editor loads.
5. From a command line, from `engine_root\dev\`, type `lmbr_waf configure`. This configures Lumberyard correctly.
6. Build the game project. For more information, see [Game Builds \(p. 1365\)](#).

To launch an existing game project

1. Go to `engine_root\dev\Bin64\`, then open `ProjectConfigurator`.
2. Select a project and choose **Set as default** to make it the default that Lumberyard Editor loads.
3. From a command line, from `engine_root\dev\`, type `lmbr_waf configure`. This configures Lumberyard correctly.
4. Build the game project. For more information, see [Game Builds \(p. 1365\)](#).
5. Open Lumberyard Editor by opening `Editor` in the `\Bin64` directory.
6. Wait until Asset Processor loads all the project assets. This may take a few minutes.
7. When Asset Processor is finished, close it.

At this point, your project is configured and you can launch the Lumberyard Editor, process assets, and building the project as needed. For more information, see [Game Builds \(p. 1365\)](#).

Enabling Gems

The Project Configurator is in preview release and is subject to change.

You can enable or disable existing Lumberyard Gems.

Note

After enabling or disabling a gem, you must run `lmbr_waf configure` from a command line, in `engine_root\dev\`, and build the project before opening Lumberyard Editor with that project. For more information, see [Game Builds \(p. 1365\)](#).

For more information about gems, see [Gems \(p. 778\)](#).

To enable or disable a gem

1. Go to `engine_root\dev\Bin64\`, then open ProjectConfigurator.
2. Select the project and choose **Gems package settings** (upper right).
3. Select which gems to include or exclude, then choose **Save**.

Using Lmbr.exe

`Lmbr.exe` is a command-line version of Project Configurator for managing game projects and gems.

`Lmbr.exe` can be run from the Lumberyard root `\dev` folder or from the `\Bin` folder it was built into, such as `\Bin64`, or `\Bin64.Debug`. Examples include:

```
dev\ $ .\Bin64.Debug\lmbr.exe  
dev\Bin64.Debug\ $ lmbr.exe
```

Project Commands

The following commands are used for creating and modifying game projects.

set-active

Sets the active project for building and executing Lumberyard. This command modifies `_WAF_`, `\user_settings.options` and `bootstrap.cfg` to reference the project specified.

```
lmbr projects set-active project_name
```

create

Creates a new project using `EmptyTemplate`, which is located at `dev\ProjectTemplates\EmptyTemplate`, as a template.

```
lmbr projects create project_name
```

list

Lists all projects in the current directory.

```
lmbr projects list
```

Gem Commands

The following commands are used for creating gems and modifying a project's use of gems.

enable

Enables the specified gem in the specified project. If a version is specified, it's used, otherwise the latest version installed is used.

```
lmbd gems enable gem_name project_name (-version version)
```

disable

Disables the specified gem in the specified project. If `-disable-deps` is specified, all dependencies of the gem will also be disabled.

```
lmbd gems disable gem_name project_name (-disable-deps)
```

create

Creates a gem with the given name. If `version` is specified, those will be used. If `-out-folder` is not specified, `name` will be used.

```
lmbd gems create gem_name (-version version) (-out-folder gems \ relative_folder)
```

list

Lists all gems installed or enabled in the specified project.

```
lmbd gems list (-project project_name)
```

Troubleshooting the Project Configurator

The Project Configurator is in preview release and is subject to change.

Review the following if you experience issues when using the Project Configurator.

Cannot create a new project

Make sure that the `engine_root_folder\dev\game_project_folder\project.json` file is editable.

Ensure that the name entered is valid and does not contain special characters or whitespaces.

Cannot enable or disable a Gem

Make sure that the `project_asset_folder\gems.json` file is editable before trying to save changes made to Gems being enabled or disabled.

New project or Gem does not appear in Visual Studio

Make sure that you have run `lmbd_waf configure` from a command line, which regenerates the Visual Studio solution to include the new project or gem.

If the project or gem still does not show up in Visual Studio, ensure that the `enabled_game_projects` field in the `engine_root_folderdev_WAF_user_settings.options` file is set to the name of your project.

Wrong project gets loaded in Lumberyard Editor

Ensure that the `engine_root_folder\dev\bootstrap.cfg` is editable. Then, open the Project Configurator, select the project to open, and choose **Save**.

Also ensure that the `sys_game_folder` field in the `engine_root_folder\dev\bootstrap.cfg` file is set to the name of your project.

Rendering and Graphics

Lumberyard uses physically-based rendering (PBR) shaders that use real-world physical rules and properties to describe how global lighting interacts with objects and how materials get rendered.

Topics

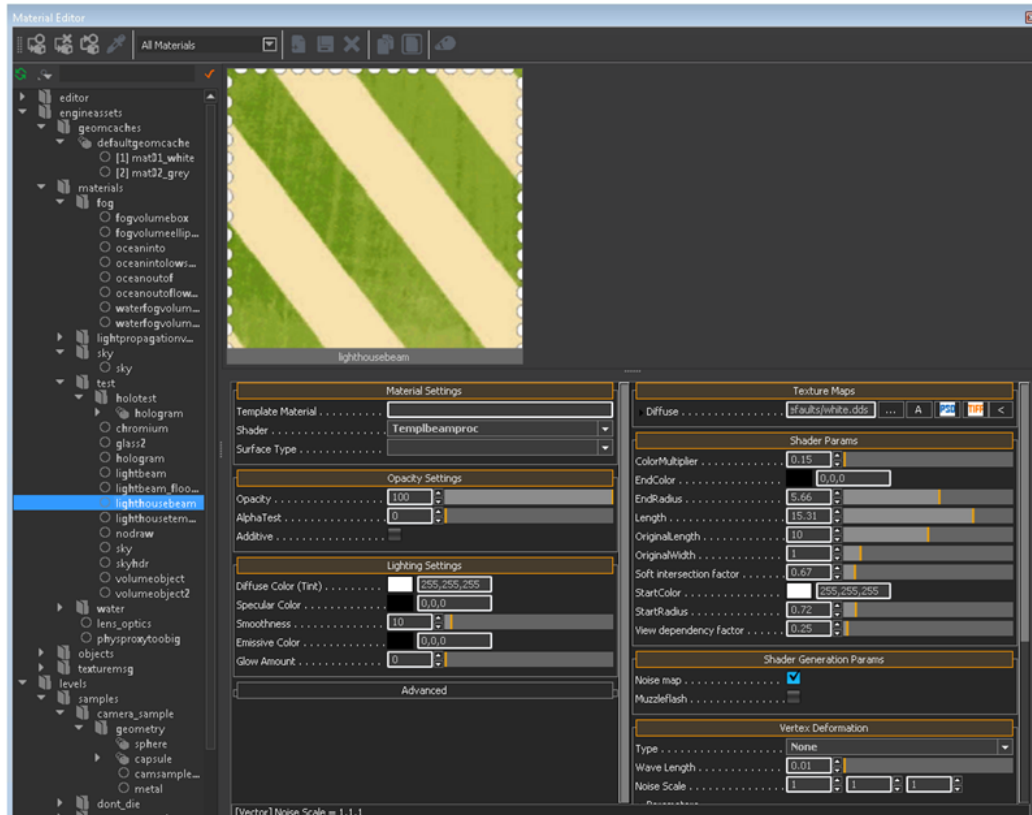
- [Materials and Shaders \(p. 990\)](#)
- [Lighting and Shadows \(p. 1062\)](#)
- [Voxel-based Global Illumination \(SVOGI\) \(p. 1069\)](#)
- [Render Cameras and Effects \(p. 1072\)](#)

Materials and Shaders

There is a close relationship between materials, textures and shaders. For a material, you select a shader and then specify the material's properties and attributes such as color, specularity, and texture that are used by the shader for rendering the object. In this way, the shader entirely defines how the object looks. Lumberyard uses physically-based rendering (PBR) shaders, which use real-world physical rules and properties to describe how light interacts with the surface of objects. This means that game object materials look realistic under all lighting conditions. For more information, see [Shader Rendering System \(p. 991\)](#).

For computer monitors, the sRGB (instead of RGB) color space is used. Using sRGB, you have greater precision for darker colors to which the human eye is more sensitive. sRGB also minimizes any banding artifacts. Always ensure that your monitor is calibrated properly. In sRGB, a 50% mid-gray is not 0.5 or 127 but rather 0.5 raised by the inverse of gamma 2.2, which equals 187 in Adobe Photoshop. For Photoshop, make sure that color management is set to sRGB and Gray-to-Gray Gamma 2.2. By default, Gray is often set to Dot Gain 20%, which results in a color transformation in the alpha channel. A value of 127 comes into Lumberyard as 104 and cause inconsistencies.

The Material Editor is the primary tool used to create materials, texture mapping, setting opacity and lighting effects, setting shader parameters, vertex deformations, tessellation, and more, as shown below.



Topics

- [Shader Rendering System \(p. 991\)](#)
- [Shader Reference \(p. 1002\)](#)
- [Selecting Material Surface Type \(p. 1034\)](#)
- [Setting Material Opacity \(p. 1034\)](#)
- [Setting Material Lighting and Color Settings \(p. 1034\)](#)
- [Material ID Mapping in Autodesk 3ds Max \(p. 1035\)](#)
- [Working with Textures \(p. 1044\)](#)
- [Working with Substances \(p. 1055\)](#)
- [Parallax Mapping \(p. 1057\)](#)
- [Using Vertex Colors \(p. 1059\)](#)
- [Customizing Post-Processing Effects \(p. 1059\)](#)

Shader Rendering System

Lumberyard uses physically-based rendering (PBR) shaders that use real-world physical rules and properties to describe how incoming light interacts with objects. This means that object materials look more convincing under different lighting conditions. A basic understanding of how light interacts with objects in the real world can be very helpful when setting up materials.

Each shader has a unique set of shader parameters (**Shader Params**) and generation parameters (**Shader Generation Params**). Some shader parameters become available (are visible) only if an associated shader generation parameter is first enabled. This is also true for certain texture map slots (file paths) under **Texture Maps**. For a listing of all shaders, see [Shader Reference \(p. 1002\)](#).

There are two categories of materials that are relevant for shader rendering: metals such as iron, gold, copper, and non-metals such as plastic, stone, wood, skin, glass. Each has different diffuse and specular reflectance characteristics.

Shading Metallic Materials - Metal reflects all visible light, hence has specular reflectance. The different types of metal have different specular colors, and should always be above sRGB 180. Metal has no diffuse reflection and thus has a black diffuse color. Rusty metal however needs some diffuse color.

Shading Nonmetallic Materials - In contrast, non-metals have diffuse reflection with weak, monochromatic (gray) specular reflections. Most non-metals reflect only 2%-5% of the light as specular. The sRGB color range for most non-metal materials is between 40 and 60 and should never be above 80. A good clean diffuse map is required for non-metals.

As the variation is so little, it is often enough to use a constant specular color instead of a specular texture map.

Shading Mixed Metal and Nonmetal Materials - Materials that contain both metals and non-metals require a specular map, as metal has a much brighter specular color than non-metal. If a specular map is used, the specular color should be set to white (255/255/255) - as it gets multiplied with the values from the specular map and would otherwise lower the physical values from the map.

To access a shader

1. In Lumberyard Editor, click **View, Open View Pane, Material Editor**.
2. In the left tree pane, select a material to work with.
3. Under **Material Settings, Shader**, make a selection.
4. Locate shader-specific parameters under **Shader Params** and associated **Shader Generation Params**.

Topics

- [Image-Based Lighting \(p. 992\)](#)
- [Environment Probes and Cubemaps \(p. 992\)](#)
- [Height Map Ambient Occlusion \(p. 993\)](#)
- [Developing a Custom Shader \(p. 993\)](#)

Image-Based Lighting

Image-based lighting is a rendering technique where complex lighting is stored in an environment map that is projected onto a scene. In simple words, a light probe or environment map is just an image on a sphere.

If the range of the image colors is within some small defined range (0-255 for monitor displays), the image is LDR (low dynamic range). With HDR (high dynamic range) some rendering effects become more apparent and correct (DOF, motion blur, bloom, dark materials, global illumination). Depending on the image and compression requirements, various texture formats can be useful.

Diffuse lighting can be approximated very well by diffuse-convolving an environment map, which can be stored as a cube map again. Because of bilinear filtering, the texture can be quite low resolution. Mip maps are not required and the result with mip maps can actually look worse as ordinary mip mapping on the GPU is computed for each 2x2 pixel block and 2x2 block artifacts can become noticeable.

Environment Probes and Cubemaps

Cube mapping uses the six faces of a cube as the texture for a material. The cube map is generated by projecting and then rendering the scene six times from a single viewpoint, one for each cube face. In

this way, the local environment can be stored as either six square textures, or unfolded onto six regions of a single texture. This texture is used to store the image of the environment surrounding the object. Cube maps are useful for showing reflections, and are relatively small in size because reflections can be blurry as long as you are not simulating a mirror.

Cube maps control shadow color, ambient diffuse, and particle diffuse as well as reflections. They function as bounce lighting by taking the colors from the surroundings and applying them directly into the diffuse texture of materials inside their specified radius.

For information about using environment probes and cubemaps, see [Environment Lighting \(p. 1062\)](#)

Cube maps use image-based lighting. For more information see [Image-Based Lighting \(p. 992\)](#).

Height Map Ambient Occlusion

Ambient occlusion (AO) is a technique used to calculate how exposed each point in a scene is to ambient lighting. The lighting at each point is a function of other geometry in the scene. For example, the interior of a building is more occluded and thus appears darker than the outside of the building that is more exposed.

Lumberyard uses height map-based ambient occlusion (AO), which is a high-performance and efficient method of providing ambient occlusion in outdoor environments without the need for prebaking. This make it suitable for PC, consoles, and virtual reality headsets.

In combination with screen space directional occlusion (SSDO), height map AO provides additional shading cues that enhance the depth perception of a scene.

To enable height map ambient occlusion

1. In the **Rollup Bar**, click the **Terrain** tab, and then choose **Environment**.
2. Under **Terrain**, select the **Height map AO** check box.

The influence that height map AO provides can be restricted using clip volumes and vis areas. Both of these object types have a **IgnoreHeightMap AO** check box that will locally disable height map AO inside the volume or area.

By default, evaluation is performed at quarter-display resolution. This can be changed using the **r_HeightMapAO** console variable, as listed below.

Heightmap AO uses the following console variables:

- **r_HeightMapAO** – Sets the resolution that evaluation is performed at. Values are : 0=off, 1=quarter resolution, 2=half resolution, 3=full resolution.
- **r_HeightMapAOAmount** – Sets the strength of the occlusion effect when combined with the scene.
- **r_HeightMapAORange**: – Area around the viewer that is affected by height map AO.
- **r_HeightMapAOResolution** – Texture resolution of the height map used for approximating the scene.

Developing a Custom Shader

Most visual effects in Lumberyard are produced by shaders, which employ a number of standard and advanced lighting models like Blinn, Cook-Torrance, Oren-Naye, Kajiya-Kay, and some custom models.

There are two types of shaders used: lighting shaders that interact with scene illumination, and regular shaders that don't calculate any lighting information but for used for post-processing effects. All lighting shaders have a common structure and make use of a unified shading interface. This interface should

always be used to ensure proper usage of the lighting pipeline, minimize code duplication and save a lot of work.

Lumberyard uses an ubershader system with compile-time defines to handle the many different shader permutations that are required for combining numerous shader features. The shader format used that is very similar to High-Level Shader Language (HLSL), DirectX FX, and CgFX.

Shader development is a programming discipline onto itself and requires expert knowledge to optimize as shader code can be performance-critical and platform-dependent.

The easiest way to create new shaders is by using a text editor. Start by copying an existing .ext extension file and associated .cfx effect file. After restarting Material Editor, the new shader will show up and can be assigned to a material.

Topics

- [Shader Development Best Practices \(p. 994\)](#)
- [Shader Rendering Pipeline \(p. 994\)](#)
- [Hot Reloading of Shaders \(p. 995\)](#)
- [Remote Shader Compiler \(p. 995\)](#)
- [Generating Shader Combinations \(p. 997\)](#)
- [Shader Cache and Generation \(p. 998\)](#)

Shader Development Best Practices

Shaders provide the flexibility that is required for realizing the modern rendering effects seen in games today. Unfortunately they have the downside of creating the need to manage large numbers of shader permutations. Each shader can potentially have thousands of permutations. Try to keep the number of new permutations as low as possible.

The shader compiler will parse the code and generate the permutations automatically, so the complexity is hidden, but at the expense of huge memory requirements and long compile times required.

The following guidelines and best practices should be taken into consideration when developing a custom shader for Lumberyard:

- Before creating a new shader, make sure that you can't reuse or parameterize one of the existing shaders.
- Pre-compute as much as possible and place it in either textures or in the vertex shader and pass the data to vertex interpolators.
- For performance reasons, avoid using sincos (8 ALU), normalize (3 ALU), pow (3-9 ALU), and smoothstep. Also, divisions are done per-scalar (3 ALU).
- Pack as much data as possible per-texture instead of doing multiple texture lookups. Texture lookups are expensive on consoles and older hardware.
- Shader code is compiled depending on three different flags: Lumberyard, material and runtime flags. Lots of flags can lead to many shader permutations, so keep the number of flags as small as possible. By using `#ifdefs` with shader flags, it is possible to define several code branches that are compiled and used depending on the flag bitmask. The shader compiler then generates different hardware shader programs for each branch and stores them in the shader cache.

Shader Rendering Pipeline

Lumberyard has a fixed rendering pipeline that is set up in the renderer code. Lumberyard is almost fully deferred and only does forward for hair, eyes, glass, transparencies, and water reflections. Lumberyard makes use of two elements: effects that define parameterized shader code, and materials that customize the shader parameters for a specific mesh.

First, Lumberyard fills the off-screen buffers like reflection buffers and shadow maps. After that, it writes the scene depth to the frame buffer and additionally to a render target. Having access to scene depth is essential for some subsequent rendering steps like screen space ambient occlusion or fog rendering. After the depth is written, Lumberyard does the forward lighting. The shadow contributions are written in a separate step to a texture that combines the shadowing result from several light sources (deferred shadowing). Finally, translucent objects are drawn in a back-to-front order.

When Lumberyard tries to render an object it will first check if a compiled shader is available. If the shader is not available, Lumberyard will try to load it from the global shader cache. If the shader cannot be found in the cache, the rendering thread will issue a request to stream the shader in from disk and will block until the streaming load is complete. This can cause severe stalls due to the relatively long time needed load data from disk.

Hot Reloading of Shaders

Lumberyard supports hot reloading of shaders, so whenever you modify and save a shader file, it will get reloaded automatically and the results can be viewed directly in a test level.

For hot reloading to work, shader files must be copied to the appropriate locations, and the following requirements must also be met:

- Add the following code to the `dev\system.cfg` file:

```
sys_PakPriority=0 <!--ensures the shader files get loaded from the file
system instead of from pak files>
r_ShadersEditing=1 <!--ensures that shader code can be recompiled at
runtime-->
```

- In the Console, type `r_reloadshaders 1`. This is only required in the game executable. In Lumberyard Editor, it will automatically reload a shader when you modify it.
- For Lumberyard, copy the shader files to the `dev\Lumberyard\Shaders` directory.

Remote Shader Compiler

Unlike PCs, many game consoles cannot compile shaders locally. For this reason, Lumberyard provides the remote shader compiler application to handle shader compilation by assigning a server on the local network that can communicate over TCP. The server receives the shader source file from a computer running Lumberyard, compiles it, and sends back the shader, which the game console can then load and use.

The remote shader compiler is also used to store all the shader combinations that have been requested by the game so far, per platform. These are used during shader cache generation, when all the requested shaders are packed into `.pak` files for use by the game.

It is not required to have a central remote shader compile server. You can instead set up the shader compiler locally on a PC.

Running the Remote Shader Compiler

You can find the remote shader compiler at `\Tools\CrySCompileServer\x64\profile\CrySCompileServer.exe`. A configuration file is also available for configuring the TCP port that the server application will listen on.

You can launch the remote shader compiler by starting `CrySCompileServer.exe` manually. However, usually it makes sense to set it up as a service, so that it is always started with the operating system.

Since requests for shaders are executed in parallel, you may notice significant delays in acquiring shaders at runtime.

Remote Shader Compiler Configuration

You configure the remote shader compiler by editing the `config.ini` file. To configure the remote shader compiler, edit the following parameters:

- `MailError` - Set to an internal company e-mail address to which notifications about compilation errors will be sent. The cache `\TempDir` directory in which the binary shaders are stored once they get compiled needs to point to a valid absolute path - the default is `C:\SHADER_CACHE`.
- The `\TempDir` cache directory in which the binary shaders are stored once they got compiled must point to a valid absolute path. The default is `C:\SHADER_CACHE`.
- `port` - TCP port, which has to match the setting in the game `system_<platform>_shader_version.cfg` file. Some examples for this file: `system_windows_pc.cfg`, `system_osx_metal.cfg`, or `system_android_es3.cfg`.
- `MailServer` - Your email server.
- `SCMailAddress` - Email address used in the `From` field of the email sent by the remote shader compiler.

The completed `config.ini` file should look similar to this example:

```
MailError = shadererror@your_company.tld
MailInterval = 1
port = 61453
TempDir = C:\SHADER_CACHE
MailServer = your_email_server
SCMailAddress = RemoteShaderCompiler@your_company.tld
PrintErrors = 1
```

Specific Platforms

In the root directory of the remote shader compiler, each supported platform has its own subfolder with additional subfolders for different version numbers. The paths are hard coded and can be configured in `RenderDll\Common\Shaders\ShaderCache.h` if required.

All paths follow this pattern: `root_folder\Tools\RemoteShaderCompiler\Compiler\<platform_folder>\Vxxx`

You can find information about the path used by the remote shader compiler in the file `ShaderCache.cpp`, under the function `mfGetShaderCompileFlags`.

Lumberyard provides all appropriate shader compilers for you that match the code of that version. Just copy the entire `\RemoteShaderCompiler` directory and run the provided binary.

Shader Cache Lists

The cache subfolder of the remote shader compiler contains different text files of all the combinations requested so far by the game. These text files are named `ShaderList_<platform>.txt` (`ShaderList_DX11.txt` for example) and contain all the shader combinations that have ever been requested on a certain platform for any level. These files are important as the shader `.pak` files cannot be generated without them.

The game submits the requests to the remote shader compiler either during actual gameplay or during loading phases, even when remote shader compiling itself is disabled. This is to ensure that all possible shader combinations are collected and that the shader caches, which are generated during the shader cache generation phase, are as complete as possible.

Game Configuration

Having a remote shader compiler server can provide a performance benefit as it caches the results and sends them out to team members instead of having to compile shaders each time. In addition,

the server keeps track of all shaders used by all people, which can be valuable if you want to make a release build that includes all shaders.

Turning the Remote Shader Compiler On and Off

You can configure whether the game uses the remote shader compiler with the following console variable, which is usually in the `system_platform_shader_version.cfg` file:

```
r_ShadersRemoteCompiler=1
```

If `r_ShadersRemoteCompiler` is set to 0, no remote shader compilation will be performed and Lumberyard will do local shader compilation instead, which will fail on consoles.

Specifying the Remote Shader Compiler Location

When the remote shader compiler is enabled, the game needs the location of the remote shader compiler. To configure the IP address of the server, use the following console variable:

```
r_ShaderCompilerServer=IPv4_of_PC_running_the_RemoteShaderCompiler
```

Using the Remote Shader Compiler Locally

You can set `r_ShaderCompilerServer=localhost` if you are running on a PC and want to use the remote shader compiler locally.

Using Multiple Remote Shader Compilers

It is possible to specify more than one remote shader compiler, as shown in the following example. The IP addresses need to be separated by semicolons as shown:

```
r_ShaderCompilerServer=10.0.0.10;10.0.0.11
```

Note

It is not possible to use the network name of the server instead of the IP address, since no name resolving is performed.

Specifying a Port Number

If the remote shader compile server uses a user-defined port number as specified in the `config.ini` file, you can configure the port number with the following console variable:

```
r_ShaderCompilerPort=portnumber
```

Disabling Request Lines

Submitting request lines to the remote shader compiler can also be disabled with the following console variable. This is useful when experimenting with shaders and you don't want to have these combinations added to the shader cache:

```
r_shaderssubmitrequestline=0
```

Proxying Remote Requests

You can use the Asset Processor to proxy remote requests to the shader compiler server if a device cannot connect to the shader compiler server. In this case, set `r_AssetProcessorShaderCompiler=1`. Now whenever the game would have made a request directly to the shader compiler server, it instead submits the request to the Asset Processor (this can also be over a USB connection), which then forwards it to the shader compiler server.

Generating Shader Combinations

Make sure that the [Remote Shader Compiler \(p. 995\)](#) has been setup successfully first. The remote shader compiler should be accessible by everyone playing the game, especially QA. Try to have everyone who is working on a certain game project share the same remote shader compiler.

Normal game builds should contain shader cache .pak files generated by the shader cache generation phase. At the beginning of a project this could be either completely missing (because no shaders requests have been submitted yet) or the .pak files could still be missing a lot of shaders.

When Lumberyard tries to render an object it will check if the compiled shader is available. When the shader is not available, it will try to load it from the global cache. This can either be loaded directly or through the streaming engine. The direct loading will cause direct disc access from the render thread and this could cause severe stalls due to the streaming thread trying to access the disc at the same time.

By default, when shader compiling is disabled, Lumberyard will stream the shaders from the global cache. The object won't be rendered when shader data is being streamed in. This default behavior can be modified with the following console variable. Note that streaming of shaders is not allowed when shader compiling is enabled, and Lumberyard will automatically disable the following console variable:

```
r_shadersAsyncActivation = 0
```

When the shader is missing from the global cache, a "request line" to store this missing shader is directly sent to the remote shader compiler to be sure that this shader will be available in the next shader cache generation. This happens even when shader compiling is disabled, but the remote shader compiler needs to be active.

When no shader compiler is defined or if the shader compiler is disabled then the request line will be ignored. It is recommended to test the remote shader compiler as much as possible to collect as many shader combinations as possible. The remote shader compiling can be disabled with the following console variable, which is disabled by default in release builds, otherwise is always enabled:

```
r_shadersRemoteCompiler = 0
```

The submission of the shader request lines can be disabled as well:

```
r_shadersSubmitRequestLine = 0
```

When shader compiling is disabled and the shader is missing in the global cache, the object won't be rendered at all. When shader compiling is enabled, and the remote shader compiler is active, an asynchronous request to compile the shader will be sent to the remote shader compiler. If the remote shader compiler is disabled, then the shader will be compiled locally on the PC platform. Other game platforms do not support local compilation.

To keep track of the current shader cache state in game, extra debug information can be enabled using the following console variable:

```
r_displayinfo = 2
```

A shader cache information line can be found on the top right of the screen, which reports the amount of Global Cache Misses (GCM) that have been found so far. It also reports if shader compiling is currently enabled or not.

All the shader cache misses also get written to a text file at the following location: `\Shaders\ShaderCacheMisses.txt`. This information is only used for debugging the current state of the shader cache, and should ideally be empty.

Shader Cache and Generation

This section discusses both the shader cache and how to generate shader cache .pak files.

Shader Cache

The shader cache stores a collection of parsed and precompiled shaders. Since the shader code is written with multiple defines, Lumberyard can generate an enormous number of different shaders. Compiling shaders on demand at runtime is only possible on the PC platform. On-demand shader compiling causes freezes during the gameplay and uses extra memory. In order to reduce this

overhead, all required shader combinations for a game are parsed, compiled, and stored in the shader cache.

The shader cache generally refers to the following files:

- `Shaders.pak` - Contains the shader source files, which is everything inside the `\Engine\Shaders` \ folders excluding `EngineAssets`.

Note

The actual shader source code (`*.cfi` and `*.cfx`) can be removed from this file for the final released version, and is not needed anymore when the binary shaders are valid and available.

- `ShadersBin.pak` - Contains the binary-parsed shader information of the shader source code.
- `ShaderCache.pak` - Contains compiled shaders for all possible combinations that have been submitted to the remote shader compiler.
- `ShaderCacheStartup.pak` - Small subset of the shader cache containing only the shaders that are used during game start. This file is loaded into memory for quicker start up times, but is not required. This cache is often used by developers to contain the minimum set of shaders required to show a loading screen so that the rest of the loading can occur.

ShaderCache.pak File Generation

Creating a `ShaderCache.pak` file consists of running the `BuilderShaderPak_DX11.bat` batch script, which in turn runs `ShaderCacheGen.exe` to ensure the local cache directory contains all the shaders that are listed in the `ShaderList.txt` file. `BuilderShaderPak_DX11.bat` then packs the contents of the cache directory, creates a `ShaderCache.zip` file, and then renames the file to `ShaderCache.pak`.

You can obtain the `ShaderList_<platform>.txt` file either from the remote shader compiler server or from the Lumberyard Editor folder. This file contains the list of all shaders your game uses, which `ShaderCacheGen.exe` uses to produce all the shader combinations your game uses.

When running Lumberyard Editor, individual shaders are created as you view them. As such, you do not strictly need a remote shader compiler server to test game release mode or test shader pack generation, you just need access to the `ShaderList_<platform>.txt` file that is created in the `dev/cache/<game_name>/<platform>/user/cache/shaders` directory when running Lumberyard Editor. However, only the shaders you have viewed on your local computer while running Lumberyard Editor will be listed in the `ShaderList_<platform>.txt` file. For this reason, it is recommended that you use a remote shader compiler server if possible.

Note

During development time when you run the game or run Lumberyard Editor, and before the shaders are packed into shader cache `.pak` files, loose shader files are created in the following directory: `Dev\Cache\<your_game>\<platform>\user\cache`.

The following sections detail the steps used to generate `ShaderCache.pak` files:

ShaderCacheGen.exe

Lumberyard ships with `ShaderCacheGen.exe`, which is located in the `\Bin64` directory. `ShaderCacheGen.exe` is essentially a stripped-down version of the Lumberyard game launcher without the render viewport, and is used to populate the local shader cache directory with all the shaders contained in the `ShaderList.txt` file.

When running `ShaderCacheGen.exe`, it first loads the `ShaderCacheGen.cfg` file, which you can customize to suit your needs.

If you have customized Lumberyard in any way, it is required that you have build Lumberyard and your game using the `all` profile, which will build both `ShaderCacheGen.exe` (and ensure that it is up to date) and the game `.dll` files that it needs. Use the following command to do this:

```
lmbw_waf build_win_x64_profile -p all
```

If you don't want to (or cannot) build using the `all` profile, you can alternatively just build the `game_and_engine` spec and the `shadercachegen` spec using the following commands:

```
lmbw_waf build_win_x64_profile -p game_and_engine
lmbw_waf build_win_x64_profile -p shadercachegen
```

Packing the Shader Cache Using a Batch File

The `BuilderShaderPak_DX11.bat` file is used to generate the `ShaderCache.pak` files, which are saved to the `dev\build\platform\your_game` directory. The batch file works by first calling `ShaderCacheGen.exe` and then calling `Tools\pakShaders.bat`.

Run `BuilderShaderPak_DX11.bat` in a command prompt window from the Lumberyard `\dev` directory, specifying the location to the `ShaderList_platform.txt` file.

For example:

```
F:\Lumberyard_folder\dev\BuildShaderPak_DX11.bat C:
\shader_compiler_server\ShaderList_DX11.txt
```

Once the shader `.pak` files are created, you can move them as needed. For example, if you've already built a release version of your game, you can place them with the rest of the `.pak` files.

When compiling shaders for your own project, you can customize the `BuildShaderPak_DX11.bat` file as needed. The following is an excerpt from a sample `.bat` file:

```
set SOURCESHADERLIST=%1
set GAMENAME=your_game_project
set DESTSHADERFOLDER=Cache\%GAMENAME%\PC\user\Cache\Shaders

set SHADERPLATFORM=PC
rem other available platforms are GL4 GLES3 ORBIS DURANGO METAL
rem if changing the above platform, also change the below directory name
  (D3D11, ORBIS, DURANGO, METAL, GL4, GLES3)
set SHADERFLAVOR=D3D11
```

Packing the Shader Cache Manually

If you want to use more complex build pipelines, you will find it beneficial to pack the shader cache manually. To do so, first run `ShaderCacheGen.exe` to generate the shader cache so you can pack it later.

Next, zip all the shaders up into `ShaderCache.zip`, then rename the file to `ShaderCache.pak`.

Each platform has different `.pak` files. The directory mapping for the different platforms is as follows:

The PC platform should copy data from the following folders:

```
lumberyard_root_folder\dev\cache\your_game\platform\user\shaders\cache\D3D9\
lumberyard_root_folder\dev\cache\your_game\platform\user\shaders\cache\D3D10\
lumberyard_root_folder\dev\cache\your_game\platform\user\shaders\cache\D3D11\
lumberyard_root_folder\dev\cache\your_game\platform\user\shaders\cache\GL4\
```

into the following destination folders:

```
shaders\cache\D3D9\  
shaders\cache\D3D10\  
shaders\cache\D3D11\  
shaders\cache\GL4\
```

The Xbox One platform should copy the data from *lumberyard_root_folder*\dev\cache*your_game*\platform\user\shaders\cache\Durango\.

The Playstation 4 platform should copy the data from *lumberyard_root_folder*\dev\cache*your_game*\platform\user\shaders\cache\Orbis\.

ShaderCache.pak should contain everything from the previously listed subfolders.

ShadersBin.pak should contain only the *.cfxb and *.cfib files.

ShaderCacheStartup.pak should contain the following files:

```
lumberyard_root_folder\dev\cache\your_game\platform\user\shaders\cache  
\<platform>\lookupdata.bin -> Shadercache\<platform>\lookupdata.bin  
lumberyard_root_folder\dev\cache\your_game\platform\user\shaders\cache  
\<platform>\CGPShader\FixedPipelineEmu* -> Shadercache\<platform>\CGPShader  
\FixedPipelineEmu*  
lumberyard_root_folder\dev\cache\your_game\platform\user\shaders\cache  
\<platform>\CGPShader\Scaleform* -> Shadercache\<platform>\CGPShader  
\Scaleform*  
lumberyard_root_folder\dev\cache\your_game\platform\user\shaders\cache  
\<platform>\CGPShader\Stereo* -> Shadercache\<platform>\CGPShader\Stereo*  
lumberyard_root_folder\dev\cache\your_game\platform\user\shaders\cache  
\<platform>\CGVShader\FixedPipelineEmu* -> Shadercache\<platform>\CGVShader  
\FixedPipelineEmu*  
lumberyard_root_folder\dev\cache\your_game\platform\user\shaders\cache  
\<platform>\CGVShader\Scaleform* -> Shadercache\<platform>\CGVShader  
\Scaleform*  
lumberyard_root_folder\dev\cache\your_game\platform\user\shaders\cache  
\<platform>\CGVShader\Stereo* -> Shadercache\<platform>\CGVShader\Stereo*  
lumberyard_root_folder\dev\cache\your_game\platform\user\shaders\cache  
\<platform>\lookupdata.bin -> Shaders\Cache\<platform>\lookupdata.bin  
lumberyard_root_folder\dev\cache\your_game\platform\user\shaders\cache  
\<platform>\Common.cfib -> Shaders\Cache\<platform>\Common.cfib  
lumberyard_root_folder\dev\cache\your_game\platform\user\shaders\cache  
\<platform>\fallback.cfxb -> Shaders\Cache\<platform>\fallback.cfxb  
lumberyard_root_folder\dev\cache\your_game\platform\user\shaders  
\cache\<platform>\fixedpipelineemu.cfxb -> Shaders\Cache\<platform>  
\fixedpipelineemu.cfxb  
lumberyard_root_folder\dev\cache\your_game\platform\user\shaders  
\cache\<platform>\FXConstantDefs.cfib -> Shaders\Cache\<platform>  
\FXConstantDefs.cfib  
lumberyard_root_folder\dev\cache\your_game\platform\user\shaders\cache  
\<platform>\FXSamplerDefs.cfib -> Shaders\Cache\<platform>\FXSamplerDefs.cfib  
lumberyard_root_folder\dev\cache\your_game\platform\user\shaders  
\cache\<platform>\FXSetupEnvVars.cfib -> Shaders\Cache\<platform>  
\FXSetupEnvVars.cfib  
lumberyard_root_folder\dev\cache\your_game\platform\user\shaders\cache  
\<platform>\FXStreamDefs.cfib -> Shaders\Cache\<platform>\FXStreamDefs.cfib
```

```
lumberyard_root_folder\dev\cache\your_game\platform\user\shaders\cache
\<platform>\scaleform.cfxb -> Shaders\Cache\<platform>\scaleform.cfxb
lumberyard_root_folder\dev\cache\your_game\platform\user\shaders\cache
\<platform>\stereo.cfxb -> Shaders\Cache\<platform>\stereo.cfxb
```

Build Platforms

The build platform subfolders listed in the following table are located at `\dev\Cache\your_game\platform\user\cache\shaders\`.

Build Platform	Build Platform Subfolder	
PC, DirectX 11	\D3D11	
XBox One	\DURANGO	
Playstation 4	\ORBIS	
PC, OpenGL 4	\GL4	

Shader Reference

Lumberyard includes the following physically-based rendering (PBR) shaders, which use real-world physical rules and properties to describe how light interacts with the surface of objects. This means that game object materials look realistic under all lighting conditions.

To access a shader

1. In Lumberyard Editor, click **View, Open View Pane, Material Editor**.
2. In the left tree pane, select a material to work with.
3. Under **Material Settings, Shader**, make a selection.
4. Locate shader-specific parameters under **Shader Params** and associated **Shader Generation Params**.

Note

Some shader parameters become available (are visible) only if an associated shader generation parameter is first enabled. This is also true for certain texture map slots (file paths) under **Texture Maps**.

Shader Name	Description
Common.Cloud Shader (p. 1003)	Use to render 3D clouds that use per-vertex gradient lighting and takes sun color, sky color, and viewer position into account.
DistanceClouds Shader (p. 1004)	Use to render cheap 2D clouds that are distantly placed in a sky scene.
Eye Shader (p. 1006)	Use to render realistic eyes that take sclera, cornea, iris, and eye moisture properties into account. Eyelash rendering is done using the Hair Shader (p. 1011) .
GeometryBeam Shader (p. 1007)	Use to create volumetric light beams that feature dust and turbulence effects.
Glass Shader (p. 1009)	Use to render glass surfaces with various refractive, reflective, ripple, tint, and cracking effects.

Shader Name	Description
Hair Shader (p. 1011)	Use to render all hair and fur, imparting different color, stranding, and animation effects. Use to render eyelashes and eyebrows along with the Eye Shader (p. 1006) for realistic eyes.
HumanSkin Shader (p. 1013)	Use to render skin and it's various physical properties including color, oiliness, pores, stubble, and wrinkles.
Illum Shader (p. 1015)	The most common shader - use to create an extremely wide variety of render effects.
Lightbeam.LightBeam Shader (p. 1018)	Use to create volumetric light beams that feature fog and other atmospheric effects.
Monitor Shader	Use to create retro television screen effects such as grain, noise, chroma shift, and interlacing. Useful for in-game displays.
NoDraw Shader	Use mainly for physics proxies, this shader does not render selected geometry.
ParticleImposter Shader (p. 1019)	Use to create particle effects that are not affected by light and hence do not cast shadows or cause reflections.
Particles Shader (p. 1019)	Use to render particle effects for fire, smoke, lightning, sparks, and fog that are affected by light and as such cast shadows and cause reflections.
Sky Shader (p. 1023)	Use to render cheap static sky (SkyBox) effects.
SkyHDR Shader (p. 1023)	Use to render realistic dynamic sky effects that change based on time of day in the level.
TemplBeamProc Shader (p. 1023)	Use to create cheap fog-like effects for light beams.
Terrain.Layer Shader (p. 1025)	Use for painting and blending terrain texture layers in a level.
Vegetation Shader (p. 1026)	Use to render trees, bushes, grass, and other vegetation, as well as imparting various bending motion effects.
VolumeObject Shader (p. 1028)	Use to render various volumetric objects such as clouds, fog, and smoke, and to impart realistic shading and self-shadowing effects.
Water Shader (p. 1029)	Use to render the ocean exclusively, and to impart various reflection, ripple, and foam effects.
Waterfall Shader (p. 1030)	Use to render waterfalls exclusively, and provides layering and tiling, as well as motion effects.
WaterVolume Shader (p. 1032)	Use to render volumetric bodies of water including lakes, pools, and rivers, and to impart various reflection, ripple, and foam effects.

Common.Cloud Shader

The Common.Cloud shader is used exclusively for 3D clouds. It uses per-vertex gradient lighting and takes the sun, cloud and viewer positions into account. Gradient lighting interpolates between the bright color, which is calculated from the HDR Sun color multiplier, and the dark color, which is calculated from the HDR Sky color multiplier. In addition, rim lighting is also applied on a per-pixel

basis to capture the effects of light scattering seen when looking at clouds being lit by the sun from behind.

3D clouds use soft clipping to gradually fade in and out at the near and far clipping plane. This prevents rendering artifacts in the far distance and flickering due to cloud particles entering and leaving the view cone near the camera during a flythrough. Additionally, clouds blend softly against opaque scene geometry.

Shader Parameters

CloudAngularAtten

Defines the angular attenuation factor for rim lighting. The smaller the value the more widespread the rim lighting effect for clouds (partially) covering the sun becomes from the viewer's point of view.

Default value: 30

CloudBacklightingScale

Defines how much to scale rim lighting. Higher values increase the glow of cloud edges.

Default value: 1

CloudOutlineSlope

Defines the slope of the ramp function used to blend in rim lighting. Higher values create harder transitions.

Default value: 1

CloudOutlineThreshold

Defines the cloud's opacity threshold value below which the rim lighting effect is applied. Higher thresholds cause the rim lighting to grow inward.

Default value: 0.4

HDRBrightnessAdjust

Controls brightness of clouds in high dynamic range image format (HDR) (relative to low dynamic range image format (LDR)).

Default value: 1

DistanceClouds Shader

The DistanceClouds shader is a dedicated shader used for 2D clouds that are placed at a far distance.

Shader Parameters

Alpha Multiplier

Alpha multiplier for cloud texture.

This parameter requires that the **Advance distance clouds** shader generation parameter is enabled.

Default value: 1

AlphaSaturation

Controls the alpha saturation of clouds when blending them with the sky. High values make less opaque parts of the cloud texture fade out more.

You can reuse the same texture for slightly different looking clouds by defining several materials with custom **AlphaSaturation** values.

This parameter does not apply if the **Simple distance clouds** shader generation parameter is enabled.

Default value: 2

Attenuation

Controls how strongly sun light is attenuated when traveling through the distance cloud. Light attenuation is computed per pixel.

Use **Attenuation** to blend between current sun color and sky color. Use higher attenuation values to accentuate cloud self-shadowing (for example, strong cloud layers).

This parameter applies if no Shader Generation parameter is enabled.

Default value: 0.6

Cloud Height

Sets the height of the cloud layer.

This parameter requires that the **Advanced distance clouds** shader generation parameter is enabled.

Default value: 0.3

Density Sky

Sets the cloud density that is used for sky light scattering.

This parameter requires that the **Advanced distance clouds** shader generation parameter is enabled.

Default value: 4.5

Density Sun

Sets the cloud density that is used for sunlight scattering.

This parameter requires that the **Advanced distance clouds** shader generation parameter is enabled.

Default value: 1.5

Exposure

Sets exposure amount to enable **HDR** on **LDR** cloud texture.

This parameter requires that the **Simple distance clouds** shader generation parameter is enabled.

Default value: 1

Opacity

Sets opacity modifier for the cloud.

This parameter requires that the **Simple distance clouds** shader generation parameter is enabled.

Default value: 1

SkyColorMultiplier

A value multiplied to the sky color defined for the current time of day.

The result is used in the pixel shader to blend between sun and sky color using the computed light attenuation value.

This parameter applies if no shader generation parameter is enabled.

Default value: 1.5

StepSize

Controls how fast to step through the cloud texture (density) to compute per-pixel light attenuation.

This effect determines the appearance of the gradient. Higher values create smoother and less abrupt gradients, but can also produce unnatural gradient changes over time of day.

This parameter applies if no shader generation parameter is enabled.

Default value: 0.004

SunColorMultiplier

A value multiplied by the sun color that is defined for the current time of day. The result is used in the pixel shader to blend between sun and sky color using the computed light attenuation value.

This parameter applies if no shader generation parameter is enabled.

Default value: 4

Shader Generation Parameters

Simple distance clouds

Enables the use of distance clouds with no volumetric shading computations.

Advanced distance clouds

Enables the use of distance clouds with more accurate shading computations.

Eye Shader

The Eye shader is used to render realistic eyes that take sclera, cornea, iris, and eye moisture properties into account.

Shader Parameters

Cornea Refraction

Controls and optionally animates pupil size.

Default value: 0.01

Cornea Smoothness

Controls the glossiness of corneas reflections.

The default creates smaller and sharper highlights that are more lifelike.

Default value: 1

Indirect bounce color

Sets the amount of indirectly bounced color. Has no effect when the Physically Based Shading (PBR) model is used.

Default value: 136,136,136

Iris Color

Tweaks the iris color without affecting the eye white.

Iris Color can be used for eye variation between characters that use the same texture.

Default value: 187,187,187

Iris Depth

Simulates the actual form of the iris, since the in-game mesh has the shape of a sphere.

Default value: 0.005

Iris Shadowing

Controls iris self-shadowing, which further simulates the actual form of the iris.

Note

This effect is only affected by sunlight and not by other light sources.

Default value: 5

Iris SSS

Controls the subsurface scattering (SSS) amount of the iris, which blurs the shadows. Higher values blur the shading more.

Default value: 0.6

Sclera SSS

Controls the subsurface scattering (SSS) amount of the eye whites, which blurs the shadows. Higher values blur the shading more.

Default value: 0.4

Depth bias scale

Sets the depth bias of the overlay mesh to avoid clipping with the eyes.

This parameter requires that the **Specular overlay** shader generation parameter is enabled.

Default value:

Diffuse occlusion strength

Controls the strength of the occlusion effect on the eyes.

This parameter requires that the **Ambient occlusion overlay** shader generation parameter is enabled.

Default value: 1

Specular occlusion strength

Controls the strength of the occlusion effect on the eyes' specular highlights.

This parameter requires that the **Ambient occlusion overlay** shader generation parameter is enabled.

Default value: 1

Shader Generation Parameters

Environment map

Enables environment map as a separate texture.

If the blending cube map feature isn't used, **Environment map** must be enabled and **nearest_cubemap** must be assigned for the texture's environment.

Ambient occlusion overlay

Enables ambient occlusion overlay rendering.

Must be enabled to use the occlusion mesh that overlays the eye. This mesh gives the eyes a more natural shadowing and integrates them with the head.

Specular overlay

Enables the eye water mesh.

GeometryBeam Shader

Use the GeometryBeam shader to create volumetric light beams that feature dust and turbulence effects.

Shader Parameters

Ambience strength

Controls the general strength of the beam effect.

Default value: 0.12

Base UV scale

Controls the scale or tiling of the object's base UV mapping.

Default value: 1

Brightness

Controls the overall brightness of the beam effect.

Default value: 1

Dust anim speed

Controls the animation speed for the dust turbulence effect, as defined by the Specular texture map.

This parameter requires that the **Dust Turbulence** shader generation parameter is enabled.

Default value: 1

Dust UV rotation

Changes the rotation of the dust turbulence effect, as defined by the Specular texture map.

This parameter requires that the **Dust Turbulence** shader generation parameter is enabled.

Default value: 0

Dust UV scale

Sets the scale or tiling of the UV mapping for the dust turbulence effect, as defined by the Specular texture map.

This parameter requires that the **Dust Turbulence** shader generation parameter is enabled.

Default value: 0.6

End color

Sets the end color for gradient along the U axis.

Default value: 255,255,255

Soft intersection factor

Controls softness of surface interaction with other opaque scene geometry.

Default value: 1

Start color

Sets the start color for gradient along the U axis.

Default value: 255,255,255

Turbulence tiling

Multiplies turbulence, as defined by the Bumpmap texture map.

This parameter requires that the **Dust Turbulence** shader generation parameter is enabled.

Default value: 1

Turbulence visibility

Controls the visibility level of turbulence, as defined by the Bumpmap texture map.

This parameter requires that the **Dust Turbulence** shader generation parameter is enabled.

Default value: 0.55

UV vignetting

Applies a vignetting effect to the edges of the UV map.

This parameter requires that the **UV Vignetting** shader generation parameter is enabled.

Default value: 4

Vertex alpha fading

If you use vertex alpha to fade out the edges, use this slider to control the interpolation curve.

Default value: 0.55

View dependency factor

Determines how beams blend in and out depending on the camera-facing angle.

The higher the value, the longer the beam is visible even when at a nearly 90° angle to camera. Smaller values cause the beam to begin to vanish.

Default value: 2

Volumetric scale

Controls the volumetric features when shadow receiving is enabled. This also has the effect of changing the soft shadow radius.

This parameter requires that the **Receive Shadows** shader generation parameter is enabled.

Default value: 0.7

Shader Generation Parameters

Dust Turbulence

Enables dust and turbulence overlay. **Specular** and **Bumpmap** texture map slots also become available under **Texture Maps** to fine-tune appearance.

Receive Shadows

Enables sun shadows to be cast on the light beams, creating volumetric shafts.

You can use this parameter for an interesting effect, but it might affect your game's performance.

UV Vignetting

Enables vignettes in UV space.

Glass Shader

The Glass shader renders windows and other glass objects, imparting refractive, tint, fog, and cracking effects for both breakable and non-breakable glass objects. Use the [Illum Shader \(p. 1015\)](#) instead if you require non-refractive effects for non-breakable glass objects.

Here are a few things to keep in mind when using the Glass shader:

- Ambient diffuse lighting from cube maps isn't taken into account.
- The shader uses the sky color exclusively for all ambient lighting.
- Except for the sun, all deferred lights don't affect transparent glass objects.
- The shader can't receive sun shadows.

Shader Parameters

Back light scale

Controls the amount of light that gets through the glass.

Default value: 0.5

Blur Amount

Controls the amount of blur.

This parameter requires that the **Blur refraction – PC Only** shader generation parameter is enabled.

Default value: 0.5

Bump Map Tiling

Adjusts tiling of the bump map independently from diffuse.

Default value: 1

Bump Scale

Sets the reflection and refraction bump scale.

Default value: 0.005

Cloudiness Masks Blur

Applies blur to just cloudy areas.

This parameter requires that the **Tint map – Tint/Gloss/Spec** shader generation parameter is enabled.

Default value: 0

Cloudiness Masks Gloss

Makes cloudy areas less glossy.

This parameter requires that the **Tint map – Tint/Gloss/Spec** shader generation parameter is enabled.

Default value: 0.5

DiffAlpha to Spec Bias

Adjusts intensity of specular in opaque and semi-opaque areas.

This parameter requires that the **Use Diffuse map** shader generation parameter is enabled.

Default value: 0

DiffAlpha to Spec Mult

Adjusts intensity of specular in opaque and semi-opaque areas.

This parameter requires that the **Use Diffuse map** shader generation parameter is enabled.

Default value: 1

Fog color

Sets fog color.

This parameter requires that the **Depth Fog** shader generation parameter is enabled.

Default value: 255,255,255

Fog cutoff end depth

Sets the distance, in meters, after which fog doesn't get any stronger.

This parameter requires that the **Depth Fog** shader generation parameter is enabled.

Default value: 20

Fog density

Sets fog density.

This parameter requires that the **Depth Fog** shader generation parameter is enabled.

Default value: 1

Fresnel bias

Sets how reflective the material is.

Default value: 1

Fresnel Scale

Sets the fresnel term scale.

Default value: 1

Indirect bounce color

Sets the amount of indirectly bounced color.

Not used if the **Depth Fog** shader generation parameter is enabled.

Default value: 136,136,136

Tint Cloudiness

Adjusts the cloudiness of tinted areas.

Default value: 0

Tint Color

Applies a tint color to the glass.

Default value: 255,255,255

Shader Generation Parameters

Use Diffuse map

Enables diffuse map for dirt, and so on. Requires alpha channel.

Environment map

Enables environment map as a separate texture.

Tint map – Tint/Gloss/Spec

Enables the RGB spec map to control tinting in red channel, cloudiness in green channel, and specular in blue channel.

Use Tint Color Map

Enables the Tint Color map. Used for multicolored glass, which goes in the custom Tint Color map slot.

Blur refraction – PC Only

Enables the blurring of objects seen through the glass.

Depth Fog

Enables depth fog behind the glass surface.

Disable Lights

Disables the reflection of lights.

Hair Shader

The Hair shader is a dedicated shader for rendering hair and fur, imparting different color, stranding, and animation effects. Hair rendering is a relatively difficult task to achieve in real-time with high-quality results due to the very fine geometry and specific lighting behavior. Depending on the hairstyle, either a simple scalp plane or a more complex shape that defines the volume of the hairstyle is needed. In some cases, breaking up a hairstyle into multiple large patches makes more sense.

Shader Parameters

Alpha Blend Multiplier

Multiplies the alpha map with the result that grayscale values are increased. Useful for the **Thin Hair** shader generation parameter.

Default value: 1

Diffuse Wrap

Allows light to pass through the hair, thus illuminating a wider area.

A tightly woven braid would have a lower **Diffuse Wrap** value (the hair being very dense), whereas sparse, loose hair would have a high **Diffuse Wrap** value.

Default value: 0.5

Indirect bounce color

Sets the amount of indirectly bounced color.

Default value: 136,136,136

Secondary Color

Sets color and intensity of the secondary specular highlight.

Primary highlight color depends on the diffuse color, whereas the secondary highlight usually has a more neutral color.

Default value: 217,217,217

Secondary Shift

Allows the secondary highlight to be shifted over the surface of the hair mesh. Make sure it works with the primary highlight, the position of which can't be shifted.

Default value: 0.1

Secondary Width

Sets the width of the secondary specular highlight.

Default value: 1.5

Shift Variation

Adds variation to the shift of the secondary highlight.

Default value: 0

Soft Intersection

Controls the alpha blending of the hair against skin or scalp.

Default value: 0

Strand Width

Controls the width of the view aligned hair strands. The mesh you exported utilizing this feature from DCC tools is rather thin. The value functions as a multiplier relative to the meshes V coordinate (width) in UV space, which can be used to control strand thickness. For example, you might want thinner strands around the border areas.

This parameter requires that the **View aligned strands** shader generation parameter is enabled.

Default value: 0.01

Thin Hair Threshold

Determines how alpha blending works for screen space effects such as DOF and motion blur. Lower values make the blending harder but can cause artifacts. Higher values soften the blending, but in some cases the hair turns into a blurry mess.

For most gameplay situations, the rather low default value works fine, but in cinematics, manual tweaking might be needed. The value must then be animated throughout the scene.

This parameter requires that the **Thin Hair** shader generation parameter is enabled.

Default value: 0.05

Wind frequency

Sets the speed at which the vertices are deformed.

This parameter requires that the **Wind bending** shader generation parameter is enabled.

Default value: 0

Wind phase

Sets hair animation phase and randomizes the deformation.

This parameter requires that the **Wind bending** shader generation parameter is enabled.

Default value: 1

Wind wave0 amp

Sets the amount or amplitude at which the vertices are deformed.

This parameter requires that the **Wind bending** shader generation parameter is enabled.

Default value: 0

Wind wave2 amp

Sets the amount or amplitude at which the vertices are deformed on a different curve.

This parameter requires that the **Wind bending** shader generation parameter is enabled.

Default value: 0

Shader Generation Parameters

Vertex Colors

Enables vertex colors.

View Aligned Strands

Enables the hair strands to self-align to the camera.

Because this is a global setting for the material, using view-aligned strands requires an extra draw call. For more information, see the **Strand Width** shader parameter.

Thin Hair

For information, see the **Thin Hair Threshold** shader parameter.

Ambient Cubemap

Enables the use of the nearest cube map specified in environment map slot for ambient lighting. Leave this enabled.

Enforce Tiled Shading

Forces hair to be fully affected by tile shading. This effect works as an override for the global tiled shading settings.

With tiled shading off, improper lighting of a scene can cause hair to turn very dark.

Use this effect carefully, as tiled shading for hair is generally quite expensive.

Wind bending

Simulates wind effects. If enabled, various frequency, phase, and amplitude wind options appear under **Shader Parameters**.

HumanSkin Shader

The HumanSkin shader is used to render skin and its various physical properties including color, oiliness, pores, stubble, and wrinkles.

Shader Parameters

Detail bump scale

Controls the strength of the detail normal map.

This parameter requires that the **Detail normal map** shader generation parameter is enabled.

Default value: 0

Displacement bias

For information, see Tessellation and Displacement.

This parameter requires that the **Displacement mapping** shader generation parameter is enabled.

Default value: 0.5

Displacement height scale

For information, see Tessellation and Displacement.

This parameter requires that the **Displacement mapping** shader generation parameter is enabled.

Default value: 1

Indirect bounce color

Sets the amount of indirectly bounced color.

Default value: 136,136,136

Melanin

Controls the amount of pigmentation in the skin.

Default value: 0

SSS Index

Changes the index of subsurface scattering (SSS).

Default value: 1.2

Tessellation face cull

This parameter requires that the **Displacement mapping** shader generation parameter is enabled.

Default value: 0.75

Tessellation factor

This parameter requires that the **Displacement mapping** shader generation parameter is enabled.

Default value: 1

Tessellation factor max

This parameter requires that the **Displacement mapping** shader generation parameter is enabled.

Default value: 32

Tessellation factor min

This parameter requires that the **Displacement mapping** shader generation parameter is enabled.

Default value: 1

Translucency Multiplier

Controls strength of the SSS feature.

Default value: 0

Wrinkles blend

Controls strength of the wrinkle map.

This parameter requires that the **Wrinkle blending** shader generation parameter is enabled.

Default value: 1.0

Shader Generation Parameters

Decal map

Enables the use of a decal map, which is blended on top of the diffuse map.

Detail normal map

Enables the use of a tiled detailed map for pores and tiny details (`_ddn`).

Displacement mapping

Enables the use of displacement mapping, which requires a height map (`_displ`).

Phong tessellation

Enables the use of rough approximation of smooth surface subdivision.

PN triangles tessellation

Enables the use of rough approximation of smooth surface subdivision.

Subsurface Scattering Mask

Enables the use of diffuse map alpha as an SSS amount multiplier.

Wrinkle blending

Enables the use of subsurface map alpha for wrinkle blending.

Illum Shader

The Illum shader is the most commonly used shader and can be used to create an extremely wide variety of effects.

Shader Parameters

Blend Factor

Controls the visibility of the blended layer.

This parameter requires that the **Blendlayer** shader generation parameter is enabled.

Default value: 8

Blend Falloff

Controls falloff of blending.

This parameter requires that the **Blendlayer** shader generation parameter is enabled.

Default value: 32

Blend Layer 2 Spec

Controls specular intensity of the second blend layer.

This parameter requires that the **Blendlayer** shader generation parameter is enabled.

Default value: 0.04

Blend Layer 2 Tiling

Controls tiling of the second blend layer.

This parameter requires that the **Blendlayer** shader generation parameter is enabled.

Default value: 1

Blend Mask Tiling

Controls tiling of the blend mask.

This parameter requires that the **Blendlayer** shader generation parameter is enabled.

Default value: 1

Detail bump scale

Sets detail bump scale.

This parameter requires that the **Detail mapping** shader generation parameter is enabled.

Default value: 0.5

Detail diffuse scale

Sets diffuse detail blend scale.

This parameter requires that the **Detail mapping** shader generation parameter is enabled.

Default value: 0.5

Detail gloss scale

Sets gloss detail blend scale.

This parameter requires that the **Detail mapping** shader generation parameter is enabled.

Default value: 0.5

Dirt Gloss

Controls the fade-out of the gloss map.

This parameter requires that the **Dirtlayer** shader generation parameter is enabled.

Default value: 1

Dirt Map Alpha

Interpolates dirt map opacity between the alpha value and fully opaque.

This parameter requires that the **Dirtlayer** shader generation parameter is enabled.

Default value: 1

Dirt Strength

Controls the fade-out strength of the dirt layer.

This parameter requires that the **Dirtlayer** shader generation parameter is enabled.

Default value: 1

Dirt Tiling

Controls tiling of the dirt layer.

This parameter requires that the **Dirtlayer** shader generation parameter is enabled.

Default value: 1

Dirt Tint

Controls the color tint of the dirt layer.

This parameter requires that the **Dirtlayer** shader generation parameter is enabled.

Default value: 255,255,255

Height bias

Controls the height bias.

This parameter requires that the **Parallax occlusion mapping** shader generation parameter is enabled.

Default value: 0.5

Indirect bounce color

Adds an extra color tint to the reflection.

Default value: 136,136,136

OBM Displacement

Controls the amount of displacement for OBM.

This parameter requires that the **Offset bump mapping** shader generation parameter is enabled.

Default value: 0.01

POM Displacement

Controls the amount of displacement for POM.

This parameter requires that the **Parallax occlusion mapping** shader generation parameter is enabled.

Default value: 0.01

Self shadow strength

This parameter requires that the **Parallax occlusion mapping** shader generation parameter is enabled.

Default value: 3

SSS Index

Controls subsurface scattering profile and amount.

Valid value ranges: 0.01 - 0.99 for marble; 1.00 - 1.99 for skin.

Default value: 1.2

Shader Generation Parameters

Detail mapping

Enables detail mapping.

Offset bump mapping

Enables offset bump mapping. This option requires a height map (`_displ` format).

Vertex Colors

Allows the use of fake ambient occlusion by using vertex colors, or adds more depth and contrast to the model.

Vertex colors must be added to the geometry in the DCC tool.

Decal

Enable if you use a Decal texture map. Decal planes are normally placed very close to other geometry.

Use to avoid flickering and z-fighting when faces are close to each other.

Parallax occlusion mapping

Enables parallax occlusion mapping. This option requires a height map (`_displ` format).

Displacement mapping

Enables displacement mapping. This option requires a height map (`_displ` format).

Phong tessellation

Enables the rough approximation of smooth surface subdivision.

PN triangles tessellation

Enables the rough approximation of smooth surface subdivision.

Dirtlayer

Enables the blending of the dirt layer on top of the base map. This requires an RGBA dirt map placed in the **Custom** slot under **Texture Maps**.

Blendlayer

Enables the blending of the normal-mapped diffuse layer on top of the base material.

DetailMap mask in Diffuse alpha

Enables diffuse map alpha for masking detail maps. This option allows the artist to use the alpha channel in RGBA texture map to mask the decal.

Parallax occlusion mapping with silhouette

For information, see Silhouette POM.

Lightbeam.LightBeam Shader

The LightBeam.LightBeam shader creates various fog-like volumetric and atmospheric effects for light beams.

Shader Parameters

Fade Distance

Defines the distance at which the effect should fade in/out.

This parameter requires that the **Use Falloff** shader generation parameter is enabled.

Default value: 200

Fade Scale

Scales how much the fading effect occurs at defined distance.

This parameter requires that the **Use Falloff** shader generation parameter is enabled.

Default value: 100

Global Density

Controls how dense or thick the fog effect is.

Default value: 1

Jitter Scale

Controls shadow jitter amount. Use to soften shadow artifacts at the cost of shadow accuracy.

Default value: 10

Noise Contrast

Defines the contrast level of the noise effect.

This parameter requires that the **Noise map** shader generation parameter be enabled.

Default value: 1

Noise Coord Scale

Scales noise. Applies to shadow and projector UVs.

This parameter requires that the **Noise map** shader generation parameter be enabled.

Default value: 0.005

Noise Dir X

Defines noise travel along the X-axis.

This parameter requires that the **Noise map** shader generation parameter be enabled.

Default value: 1

Noise Dir Y

Defines noise travel along the Y-axis.

This parameter requires that the **Noise map** shader generation parameter be enabled.

Default value: 0

Noise Dir Z

Defines noise travel along the Z-axis.

This parameter requires that the **Noise map** shader generation parameter be enabled.

Default value: 0

Noise Speed

Controls the speed at which noise travels.

This parameter requires that the **Noise map** shader generation parameter be enabled.

Default value: 5

Shader Generation Parameters

Noise map

Enables the use of a 3D, procedurally-generated noise map.

Use Falloff

Activates the **Fade**-type shader parameters to tweak visual fall-off settings.

Extra Sampling

Reduces aliasing for slightly more expensive rendering.

ParticleImposter Shader

The ParticleImposter shader is used to create particle effects that are not affected by light and hence do not cast shadows or cause reflections.

Particles Shader

The Particles shader is used to render particle effects for fire, smoke, lightning, sparks, and fog that are affected by light, and as such cast shadows and cause reflections.

Shader Parameters

Color lookup amplitude

Sets the color lookup brightness and multiplier.

This parameter requires that the **Color lookup** shader generation parameter is enabled.

Default value: 1

Color lookup color phase

Sets the per-color phase to be used.

This parameter requires that the **Color lookup** shader generation parameter is enabled.

Default value: 1

Global Illumination Amount

Sets the amount of global illumination.

Default value: 1

Perturbation amount

Controls the amount of deformation that is used.

This parameter requires that the **Screen space deformation** shader generation parameter is enabled.

Default value: 0.01

Perturbation anim speed

Controls animation translation speed and frequency that is applied to the deformation map.

This parameter requires that the **Screen space deformation** shader generation parameter is enabled.

Default value: 0.05

Perturbation tiling

Controls the tiling amount of deformation.

This parameter requires that the **Screen space deformation** shader generation parameter is enabled.

Default value: 0.5

Deform amount

Controls deformation multiplier.

This parameter requires that the **Deformation** shader generation parameter is enabled.

Default value: 0

Deform anim speed

Controls deformation animation translation speed and frequency.

This parameter requires that the **Deformation** shader generation parameter is enabled.

Default value: 0

Deform tiling

Controls deformation tiling.

This parameter requires that the **Deformation** shader generation parameter is enabled.

Default value: 0.1

Refraction Bump Scale

Sets the refraction bump scale.

This parameter requires that the **Refraction** shader generation parameter is enabled.

Valid value range: 0 - 2.0

Default value: 0.1

Soft particles scale

Controls soft particle intersection softness for sharper or softer intersections.

Default value: 1

Threshold for writing depth

Sets the threshold for writing depth.

This parameter requires that the **Depth Fixup** shader generation parameter is enabled.

Default value: 0.05

Shader Generation Parameters

Refraction

Enables the use of a bump-map texture as the displacement for refraction.

Refraction Tinting

Enables the use of a color texture to tint refraction.

Screen space deformation

When enabled, the **Refraction Normal** texture map slot also becomes available under **Texture Maps**.

Deformation

When enabled, the **Deformation Normal** texture map slot also becomes available under **Texture Maps**.

Color lookup

Enables the use of the color lookup map for applying color lookup. When enabled, the **Color Lookup Map** texture map slot also becomes available under **Texture Maps**.

Specular Lighting

Enables the calculation of specular lighting in addition to diffuse lighting.

Depth Fixup

Enables writing depth for depth of field and post processing.

Scopes Shader

The Scopes shader is used to render various optical effects for binoculars, telescopes, and weapon sight scopes.

Shader Parameters

Fake glow amount

Sets the amount of fake glow.

This parameter requires that the **Reflex sight new** shader generation parameter is enabled.

Default value: 0.25

Fresnel Bias

Sets the amount of fresnel bias.

This parameter requires that the **Scope zoomed refraction** shader generation parameter is enabled.

Default value: 1

Fresnel Scale

Sets the fresnel scaling amount.

This parameter requires that the **Scope zoomed refraction** shader generation parameter is enabled.

Default value: 1

Hologram depth

Sets the depth of the hologram.

This parameter requires that the **Use halo sight depth** shader generation parameter is enabled.

Default value: 2

Holographic noise scale

Sets the holographic noise scale.

This parameter requires that the **Reflex sight new** shader generation parameter is enabled.

Default value: 0

Noise bias

Sets noise bias.

This parameter requires that the **Reflex sight new** shader generation parameter is enabled.

Default value: 1

Noise scale

Sets noise scale.

Default value: 0.75

Object space UV usage

Sets the amount of usage of object space.

Default value: 0

Refraction Bump Scale

Sets the amount of scaling for refraction bumpiness.

Default value: 0

Scope color multiplier

Sets the scope color multiplier.

Default value: 160

Scope scale

Sets scope scale.

Default value: 4

Shader Generation Parameters

Reflex sight

Use for reflex-style weapon sights. When enabled, the **Diffuse** texture map slot under **Texture Maps** also becomes available.

Reflex sight new

Use for the newer version reflex-style weapon sights. When enabled, the **Diffuse** texture map slot under **Texture Maps** also becomes available.

Scope zoomed refraction

Use to produce light refraction effects for zoomed-in scopes.

Use halo sight depth

Used for holographic-style weapon sights with a depth-field modifier.

Thermal vision scope

Use to produce thermal color effects for night-use scopes.

Sky Shader

The Sky shader is used to render performance-optimized static sky (SkyBox) effects.

Shader Parameters

Indirect bounce color

Adds an extra color tint to the reflection.

Default value: 136,136,136

SSS Index

Subsurface Scattering Index

Default value: 0

SkyHDR Shader

The SkyHDR shader is used to render realistic dynamic sky effects that change based on the time of day in a level.

Shader Parameters

Indirect bounce color

Adds an extra color tint to the reflection.

Default value: 136,136,136

SSS Index

The Subsurface Scattering Index.

Default value: 0

Shader Generation Parameters

No moon

Removes the moon for the dynamic sky.

No night sky gradient

Removes the entire day night effect gradient for the dynamic sky.

No day sky gradient

Removes the entire day sky effect gradient for the dynamic sky.

TempBeamProc Shader

The TempBeamProc shader is used to create inexpensive fog-like light beam effects, enabling control over beam size and blending.

Best Practices

The following are some best practices for using this shader:

- Select the **No Shadow** property under **Advanced**.
- Set **Opacity** to 100%.
- Use a simple grayscale texture with no alpha in the **Diffuse** texture map slot.

- The shader fades out rendering faces that are at a certain angle to the camera. As such, use different sub-materials for the top plane and the intersecting planes to allow control of the angle of visibility.

Shader Parameters

ColorMultiplier

Increases or decreases brightness and blending.

Default value: 1

EndColor

Sets the end color for the gradient.

Default value: 255,255,255

EndRadius

Sets the radius (in meters) of the effect at the end of the object.

Default value: 2

Length

Adjusts the scaling of the rendered effect.

Default value: 10

OriginalLength

Sets the length scaling factor. If the values of **Length** and **OriginalLength** are identical, the object has scale of 100%.

Default value: 10

OriginalWidth

Sets the width scaling factor. If the values of **Width** and **OriginalWidth** are identical, the object has scale of 100%.

Default value: 1

Soft intersection factor

Controls softness of surface interaction with other opaque scene geometry.

Default value: 1

StartColor

Sets the start color for the gradient.

Default value: 255,255,255

StartRadius

Sets the radius (in meters) of the effect at the start of the object.

Default value: 1

View dependency factor

Controls the blending in and out depending on the facing angle to the camera.

The higher the value, the longer the effect is visible even when nearly 90° to camera, the smaller the value the earlier the effect starts to vanish.

Default value: 2

Shader Generation Parameters

Noise map

Enables the use of a 3D animated noise map, which enables a nice motion to the beams. However, this motion cannot be controlled by any parameters.

Muzzleflash

Enables use as a muzzle flash effect.

Terrain.Layer Shader

The Terrain.Layer shader is used for painting and blending terrain texture layers in a level. Besides needing a bump map and high-passed diffuse map, the Terrain.Layer shader also requires a height map with either offset bump mapping (OBM) or parallax occlusion mapping (POM) enabled. Blending uses the height map to determine how the materials blend together. For example, if you have pebbles on one material and dirt as another, you may want the pebbles to accurately stand out from the dirt.

Here are a few notes regarding usage of this shader:

- The Detail normals texture is not an external texture, but rather a texture generated by Lumberyard through code.
- The **Decal** parameters don't appear under **Shader Params** unless you put a texture into the **Decal** slot first. The **Decal Bumpmap** slot also appears after this task.
- Flow map textures go in the **Detail** slot.

Shader Parameters

Blend Factor

Changes the visibility of the blended layer. A height map is required. **OBM** or **OBM** shader generation parameter must be enabled first.

Default value: 0

Blend Falloff

Changes the falloff of blending. A height map is required. **OBM** or **OBM** shader generation parameter must be enabled first.

Default value: 1

Detail bump scale

Detail mapping shader generation parameter must be enabled first.

Default value:

Detail gloss scale

Detail mapping shader generation parameter must be enabled first.

Default value:

DetailTextureStrength

Sets the strength of the diffuse map, which dictates how much detail texture is visible over the layer texture. The higher the value, the more you see only your Diffuse map.

Default value: 1

Height bias

POM shader generation parameter must be enabled first.

Default value: 0.5

Indirect bounce color

Sets the amount of indirectly bounced color

Default value: 136,136,136

Default value:

OBM Displacement

OBM shader generation parameter must be enabled first.

Default value: 0.01

POM Displacement

POM shader generation parameter must be enabled first.

Default value: 0.01

Self shadow strength

POM shader generation parameter must be enabled first.

Default value: 3

Shader Generation Parameters

Offset bump mapping (OBM)

Uses offset bump mapping. Requires a height map (`_displ` format).

Detail mapping

Uses detail mapping.

Parallax occlusion mapping (POM)

Uses parallax occlusion mapping. Requires a height map (`_displ` format).

Vegetation Shader

The Vegetation shader is used to render trees, bushes, grass and other vegetation, as well as imparting various bending motion effects.

Here are a couple of guidelines for best results and performance using this shader:

- Use an **AlphaTest** value of 50 for opacity.
- Use a **Diffuse** color value of 128, 128, 128 for lighting.

Shader Parameters

Back diffuse color scale

Controls the color strength of the backside color of leaves. **Leaves** or **Grass** shader generation parameter must be enabled first.

Default value: 0.85

Back View Dependency

Changes the view dependency of the back diffuse color. Where it starts depends on the point of view earlier or later. **Leaves** or **Grass** shader generation parameter must be enabled first.

Default value: 0.5

Bending branch amplitude

Defines the movement of blue color in the in the complex bending setup.

Default value: -0.5

Bending edges amplitude

Defines the movement of red color in the in the complex bending setup.

Default value: 0.2

Blend Factor

Changes visibility of blending layer. **Blendlayer** generation parameter must be enabled first.

Default value: 0

Blend Falloff

Changes the falloff of blending.

Default value: 1

Blend Layer 2 Spec

Changes specular intensity of second blend layer. **Blendlayer** generation parameter must be enabled first.

Default value:

Blend Layer 2 Tiling

Changes tiling of second blend layer. **Blendlayer** generation parameter must be enabled first.

Default value:

Blend Mask Tiling

Changes tiling of blend mask.

Default value: 1

Cap opacity fall off

Controls the fading of alpha test textures when seen at a steep angle (so they look less like a plane). A value of 1 means it's turned off; 0 means it's fully activated.

Default value: 1

Detail bending frequency

Defines the bending speed for complex (wind) bending. Make sure that this value is in the correct proportion to the wind in your level.

Default value: 1

Indirect bounce color

Sets the amount of indirectly bounced color.

Default value: 136,136,136

Terrain Color Blend

Controls how much of the terrain color is blended into the diffuse color when up close. **Use Terrain Color** for the selected vegetation object must be enabled first, except when **AutoMerge** is enabled.

Default value: 0

Terrain Color Blend Dist

Controls how much of the terrain color is blended into the diffuse color at a distance. **Use Terrain Color** for the selected vegetation object must be enabled first, except when **AutoMerge** is enabled.

Default value: 0.5

Transmittance Color

Applies color tint for translucency. **Leaves** or **Grass** shader generation parameter must be enabled first.

Default value: 255,255,203

Shader Generation Parameters

Leaves

Enables leaf shading and leaves animation. This parameter causes the gaming Lumberyard to use a much more complex (expensive) shading, so activate only for leaves rendering.

Grass

Enables simple and cheap grass rendering. Specular and normal map setting are essentially disabled, so the shading is only diffuse.

Detail bending

Enables detail bending, which simulates wind on vegetation objects. Activate for leaves and grass only. Also, make sure to paint required vertex colors.

Detail mapping

Enables detail mapping.

Blendlayer

Enables normal-mapped diffuse layer blended on top of base material.

Displacement mapping

Enables displacement mapping. Requires a height map (_displ format).

Phong tessellation

Enables rough approximation of smooth surface subdivision.

PN triangles tessellation

Enables rough approximation of smooth surface subdivision.

VolumeObject Shader

The VolumeObject shader is used to render various volumetric objects such as clouds, fog, and smoke, and to impart realistic shading and self-shadowing effects. In addition to the shader parameters listed further on, the following Time-of-Day parameters also affect VolumeObject rendering:

- Alpha Saturation
- Attenuation
- SkyColorMultiplier
- StepSize
- SunColorMultiplier

Shader Parameters

Global Density

The global density.

Default value: 1

Shader Generation Parameters

Soft Intersections

Enhances transparency with opaque scene geometry. Use sparingly due to increased pixel shading cost.

Back Lighting

Enables back lighting of volume objects. The silhouette slightly glows when viewed against the sun.

Jittering

Enables jittering on volume objects.

Soft Jittering

Softens the jittering effect on volume objects.

Use TOD Settings

Enables Time-of-Day (TOD) settings.

Water Shader

The Water shader is a dedicated shader used to render the ocean exclusively, and imparts various reflection, ripple, and foam effects. For lakes, rivers, and other bodies of water, use the [VolumeObject Shader \(p. 1028\)](#) instead.

Shader Parameters

Crest Foam Amount

Sets amount of foam that appears at the crest of a wave. Use for FFT-displaced ocean only on the Very High Spec setting. **Foam** shader generation parameter must be enabled first.

Default value: 1

Detail Normals scale

Sets normal scale.

Default value: 0.5

Detail Tiling

Sets waves detail bump tiling.

Default value: 2.5

Fake camera speed

Causes the surface of the water to scroll in world-space. This parameter gives the impression that a stationary object in the ocean is actually moving through the ocean. **Fake camera movement** shader generation parameter must be enabled first.

Default value: 0

Foam Amount

Multiplier for foam. **Foam** shader generation parameter must be enabled first.

Default value: 1

Foam soft intersection

Very similar to soft intersection, but blends foam on intersection regions. **Foam** shader generation parameter must be enabled first.

Default value: 0.75

Foam tiling

Sets tiling amount for foam. **Foam** shader generation parameter must be enabled first.

Default value: 12

Fresnel gloss

The gloss of the Fresnel effect.

Default value: 0.9

Gradient scale

Applies a more choppy look to waves.

Default value: 0.1

Height scale

Sets scale for height map, which is used for parallax mapping approximation.

Default value: 0.2

Normals scale

Sets overall scale for normals.

Default value: 1.25

Rain ripples tiling

Sets tiling for rain ripples.

Default value: 1

Reflection bump scale

Reflection map bump scale.

Default value: 0.1

Reflection scale

Sets real-time reflection map multiplier or cube map multiplier for water volumes.

Default value: 1

Ripples normals scale

Sets dynamic ripples normals scale.

Default value: 1

Soft intersection factor

Sets water soft intersection with geometry.

Default value: 1

SSS scale

Sets SSS scale.

Default value: 2

Tiling

Sets waves bump tiling.

Default value: 10

Watervol flow speed

Default value:

Sets the flow speed for the water volume flow map. **Water Volume flow** shader generation parameter must be enabled first.

Default value: 10

Shader Generation Parameters

Water Volume flow

Enables water flow along UVs.

Water Volume

Disable this parameter to use the Water shader.

Sunshine

Enables sunshine effects on the ocean surface.

Fake camera movement

Enables fake camera movement for scenes in the ocean.

No refraction bump

Disables refraction bump.

Foam

Enables foam on the ocean surface.

Waterfall Shader

The Waterfall shader is used for waterfalls exclusively, and provides layering, tiling, and motion effects.

Shader Parameters

Alpha blend multiplier

Applies a multiplier amount for alpha blending.

Default value: 1

Foam deform

Deforms the foam texture with a multiplier, based on the bumpmap texture. **Foam** shader generation parameter must be enabled first.

Default value: 0.025

Foam multiplier

Applies a multiplier amount for foam texture. **Foam** shader generation parameter must be enabled first.

Default value: 1

Fresnel bias

The Fresnel bias.

Default value: 0.25

Layer0 bump scale

Scales the bump map texture for the first layer.

Default value: 2

Layer0 speed

Controls the texture rolling speed for the first layer.

Default value: 1

Layer0 tiling

Sets the texture tiling amount for the first layer.

Default value: 1

Layer1 bump scale

Scales the bump map texture for the second layer.

Default value: 1

Layer1 speed

Controls the texture rolling speed for the second layer.

Default value: 2

Layer1 tiling

Sets the texture tiling amount for the second layer.

Default value: 2

Reflect amount

Controls the reflection amount, which comes from the environment map. **Environment map** shader generation parameter must be enabled first.

Default value:

Refraction bump scale

Scale the refraction effect inherited by the bump map texture.

Default value: 0.01

Sun multiplier

Applies a multiplier amount for sun shading. **Sun shading** shader generation parameter must be enabled first.

Default value: 1

Shader Generation Parameters

Environment map

Enables the use of an environment map as a separate texture.

Sun shading

Enables sunlight shading effects.

Foam

Enables foam rendering. Uses diffuse texture.

WaterVolume Shader

The Watervolume shader is used for rendering volumetric bodies of water including lakes, pools, and rivers and imparts various reflection, ripple, and foam effects. For the ocean, use the [Water Shader](#) (p. 1029) instead. Here are a few notes regarding usage of this shader:

- The Detail normals texture is not an external texture, but rather a texture generated by Lumberyard through code.
- The **Decal** parameters don't appear under **Shader Params** unless you put a texture into the **Decal** slot first. The **Decal Bumpmap** slot also appears after this task.
- Flow map textures go in the **Detail** slot.

Shader Parameters

Detail normals scale

Scales the detail bump normals intensity.

Default value: 0.5

Detail tiling

Sets detail bump tiling.

Default value: 2.5

Env projection scale

Controls the projection scale, or the tiling, of the specified environment map.

Default value: 20

Env reflection amount

Controls the reflection amount of the environment map. Can be offset with Specular Color.

Default value: 1

Flow map scale

Controls the scale, or tiling, of the flow map texture.

Default value: 0

Flow speed

Specifies the speed of the flow effect. **Water flow** shader generation parameter must be enabled first.

Default value: 0

Foam amount

Controls the amount of foam placed on the water surface. **Foam** shader generation parameter must be enabled first.

Default value: 1

Foam soft intersection

Controls how the foam behaves from contact areas. Foam forms around intersecting objects and the terrain after it gets close to the surface. **Foam** shader generation parameter must be enabled first.

Default value: 0.75

Foam tiling

Sets the tiling amount of the foam texture. **Foam** shader generation parameter must be enabled first.

Default value: 12

Normals scale

Controls the scale of the normals. Don't confuse this parameter with Detail normals.

Default value: 1.25

Rain ripples tiling

Sets the tiling amount for the rain ripples texture.

Default value: 1

Realtime reflection amount

Controls the reflection amount for the Realtime Reflection.

Default value: 1

Soft intersection factor

Similar to the Foam soft intersection but for the base water surface.

Default value: 1

Tiling

Changes the amount of texture map tiling on the water surface.

Default value: 10

Vertex wave scale

Sets strength of vertex displaced wave animation.

Default value: 0.125

Shader Generation Parameters

Realtime Reflection

Enables approximate real-time reflections.

Water flow

Enables water to flow along geometry UVs.

Water flow map

Enables water flow along a flow map.

Water flow map strength

Enables additional water flow strength controls, which requires the blue channel for strength.

Sun specular

Enables water sunshine.

Debug flow map

Enables visualizing flow map.

Foam

Enables foam.

Selecting Material Surface Type

The surface of a material determines the physical effects and how the material reacts to other materials and its environment. For example, a metal surface is hard, doesn't shatter, reacts to bullets by generating spark particles, and has a unique sound when struck. Contrast this with a grass surface, which is soft, responds to wind, generates grass strands and dirt particles when hit, and sounds different than metal.

To select a material surface type

1. In Lumberyard Editor, click **View, Open View Pane, Material Editor**.
2. In the left pane, click to select the desired asset.
3. Under **Material Settings**, for **Surface Type**, make a selection.

Setting Material Opacity

An object's opacity refers to its transparency level. Opacity is important when using an alpha channel for transparency.

To set opacity for a material

1. In Lumberyard Editor, click **View, Open View Pane, Material Editor**.
2. In the left pane, click to select the desired asset.
3. Under **Opacity Settings**, click and adjust the values of the following parameters:
 - a. **Opacity**: Values below 50 tends more to the white end of the alpha channel map. Values above 50 tends more to the black end of the alpha channel map.
 - b. **AlphaTest**: Used to achieve soft, semi-transparent results. To use AlphaTest, set the Opacity value to 100 and the AlphaTest value to 50.
 - c. **Additive**: When selected, the material color will be added to the scene background color behind the object, with the resulting color being brighter. This is used for almost transparent materials like glass.

Setting Material Lighting and Color Settings

Material color, specular reflection, and lighting effects such as specularity, glossiness, and glow are specified using the Material Editor.

To set lighting and color and settings for a material

1. In Lumberyard Editor, click **View, Open View Pane, Material Editor**.
2. In the left pane, click to select the desired asset.
3. Under **Lighting Settings**, click to set the values for the following parameters:

Material Lighting and Color Settings

Parameter	Description
Diffuse Color	The base color of a material.
Specular Color	The reflective brightness and color of a material when light shines on the object. The greater the value, the shinier the material.

Parameter	Description
	Various gray-scale values provide different black-white reflective brightness levels. This is achieved by using equal values for the RGB settings. For colored reflections using non-equal RGB values, see Anisotropic (Colored) Specular (p. 1035).
Smoothness	The acuity or sharpness of a specular reflection. For values of 10 or less, there is a scattered reflection, while values greater than 10 yield a sharp reflection. You cannot have glossiness without specular color (reflection), as glossiness determines the sharpness of the reflection.
Emissive Color	Enables objects to emit light and be visible in the dark. Can add brightness to objects. Unlike glow, does not emit light onto other objects. Does not work with deferred shading.
Emissive Intensity	Enables objects to glow, and simulates light emitting from extremely bright surfaces. Used in dark scenes for computer monitors, lamps, fire, neon lights, and similar objects. Unlike emissive color, emits light onto other objects. Glow color is specified using a diffuse texture RGB channel, while glow mapping comes from using a diffuse texture alpha channel. This allows you to mask out the pixels where you want less (or no) glow. Glow can be used only with the Cloth, HumanSkin, and Illum shaders. To enable or disable glow, use the <code>r_Glow</code> console variable.

Note

Diffuse color, specular color, and emissive color values can be typed in directly in R,G,B format, or may be selected using the **Colors** dialog box. To use the color picker, click the color square next to the parameter to open the **Colors** dialog box. You can select either standard colors or custom colors where you can specify hue, saturation, and luminance values in addition to the RGB values.

Anisotropic (Colored) Specular

Non-white specular colors (using non-equal RGB values) is only supported using the **Anisotropic Specular** setting. Anisotropic specular means that the reflection is not the same in all directions and has a dependency on the orientation. This is used for materials like brushed metal, fabrics, satin, silk, hair, and compact disks. Anisotropic specular has two parameters:

- **Specular Multiplier** – Defines specular strength.
- **Anisotropic Shape** – Defines the shape of the specular highlights.

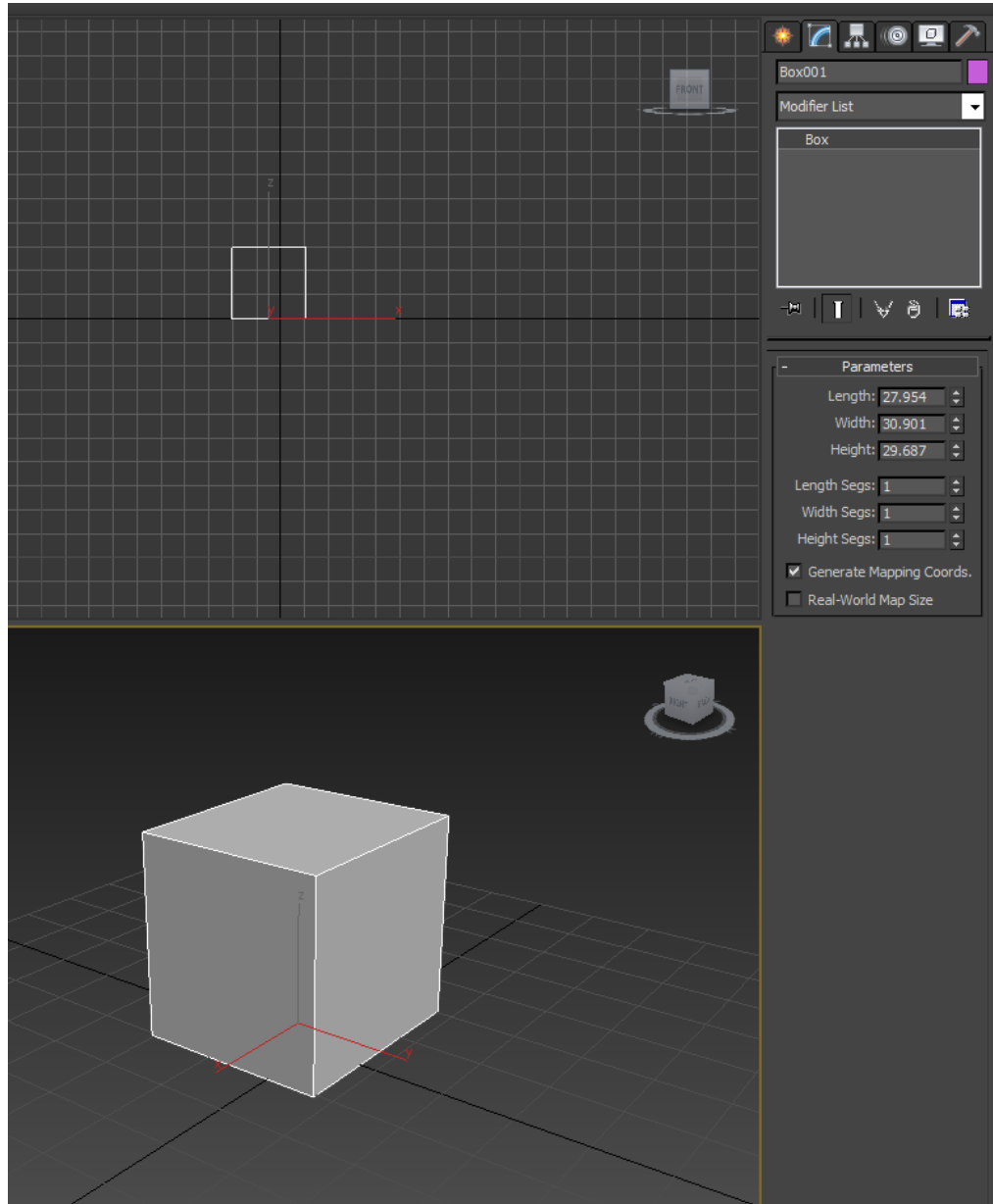
Material ID Mapping in Autodesk 3ds Max

A mesh (.cgf file) can have different materials assigned to different faces. When you work in Autodesk 3ds Max, make sure you have enough submaterials to cover the number of material IDs assigned to faces on the mesh object. Otherwise the material IDs won't get exported correctly to Lumberyard.

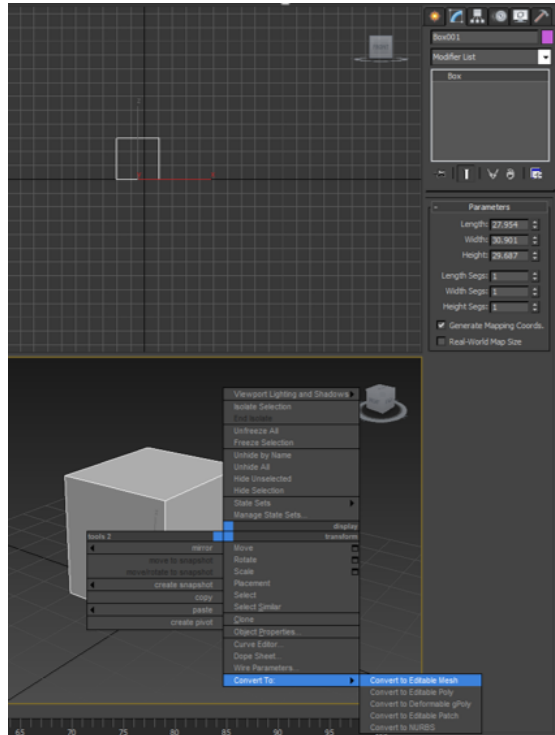
The following procedure presents an example that uses a multimaterial cube.

To map multi-material IDs in 3ds Max

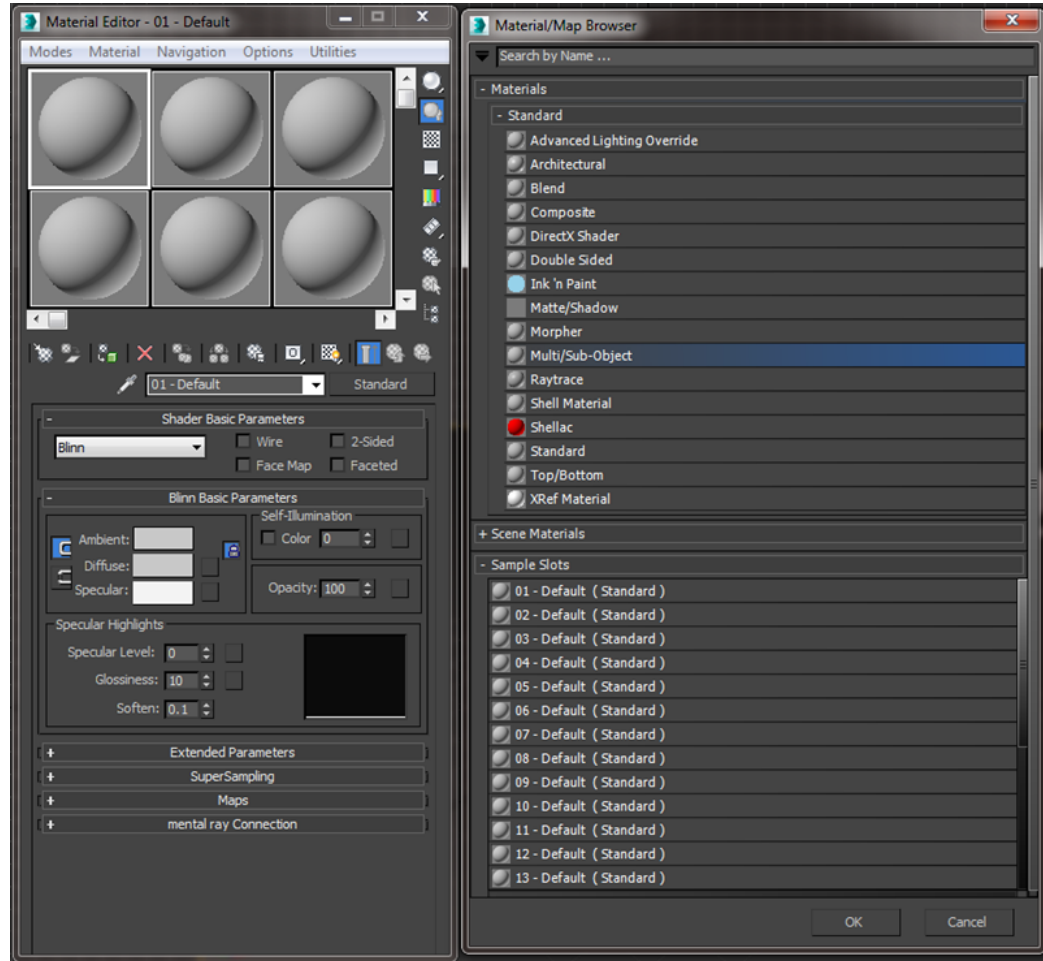
1. Open 3ds Max. Then create and place a cube in the viewport.



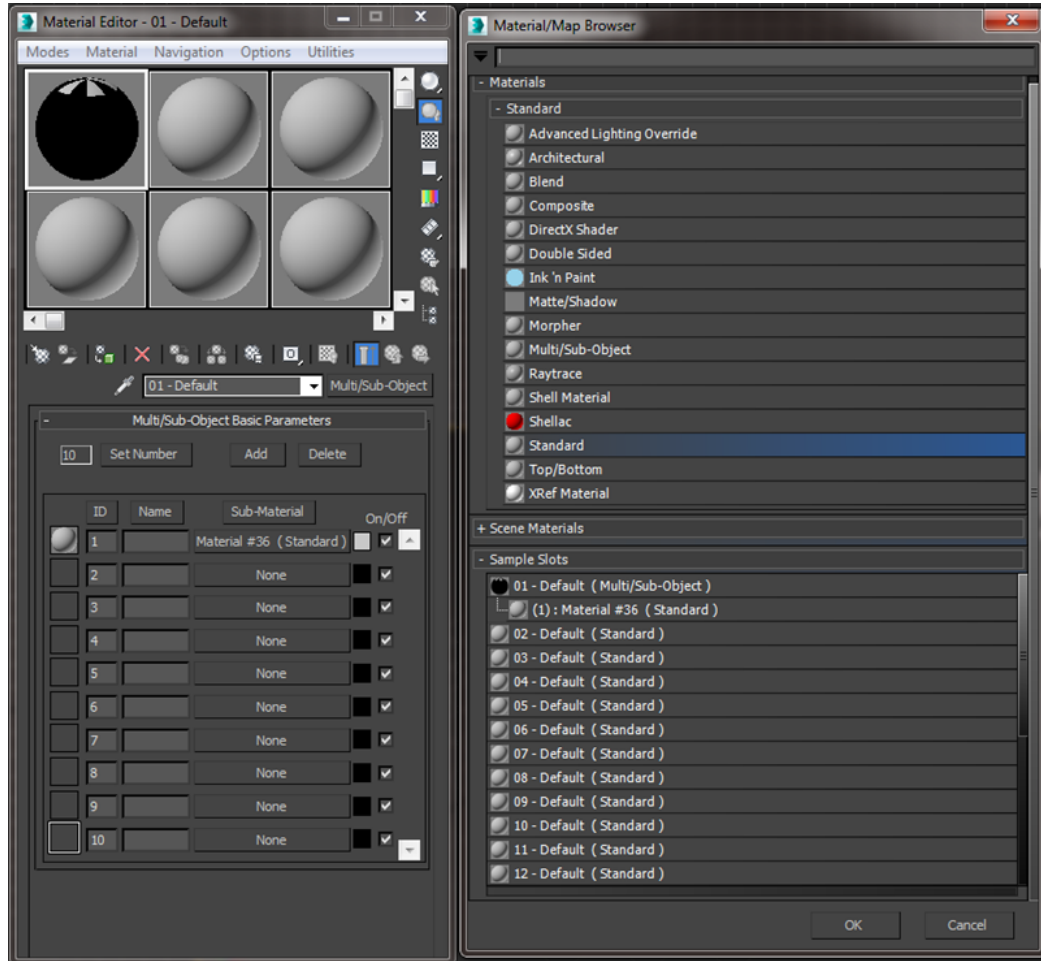
2. Right-click the cube and click **Convert To, Convert to Editable Mesh**. You can now assign different material IDs to the faces.



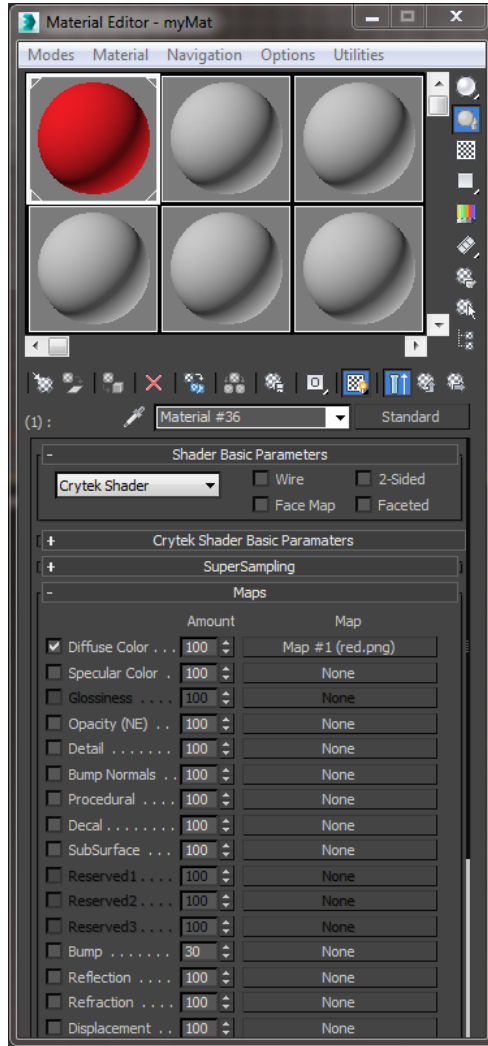
3. From the 3ds Max top menu, choose **Rendering, Material Editor, Compact Material Editor**.
4. From the 3ds Max top menu, choose **Rendering, Material/Map Browser**.



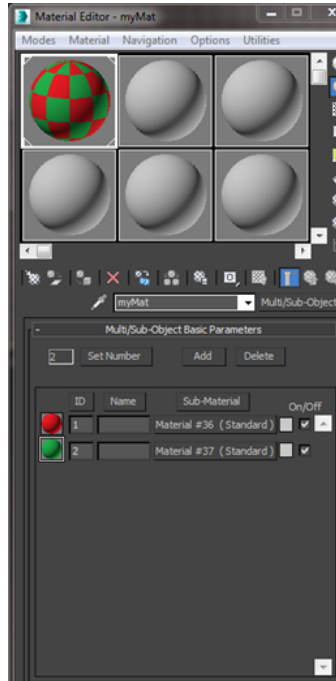
5. In **Material/Map Browser**, under **Materials**, expand **Standard**. Then double-click **Multi/Sub-Object**. In the 3ds Max **Material Editor**, under **Multi/Sub-Object Basic Parameters**, look for a material ID list to fill in. Select the first entry by clicking **None** in the **Sub-Material** column. Select **Standard** under the **Standard** material rollout.



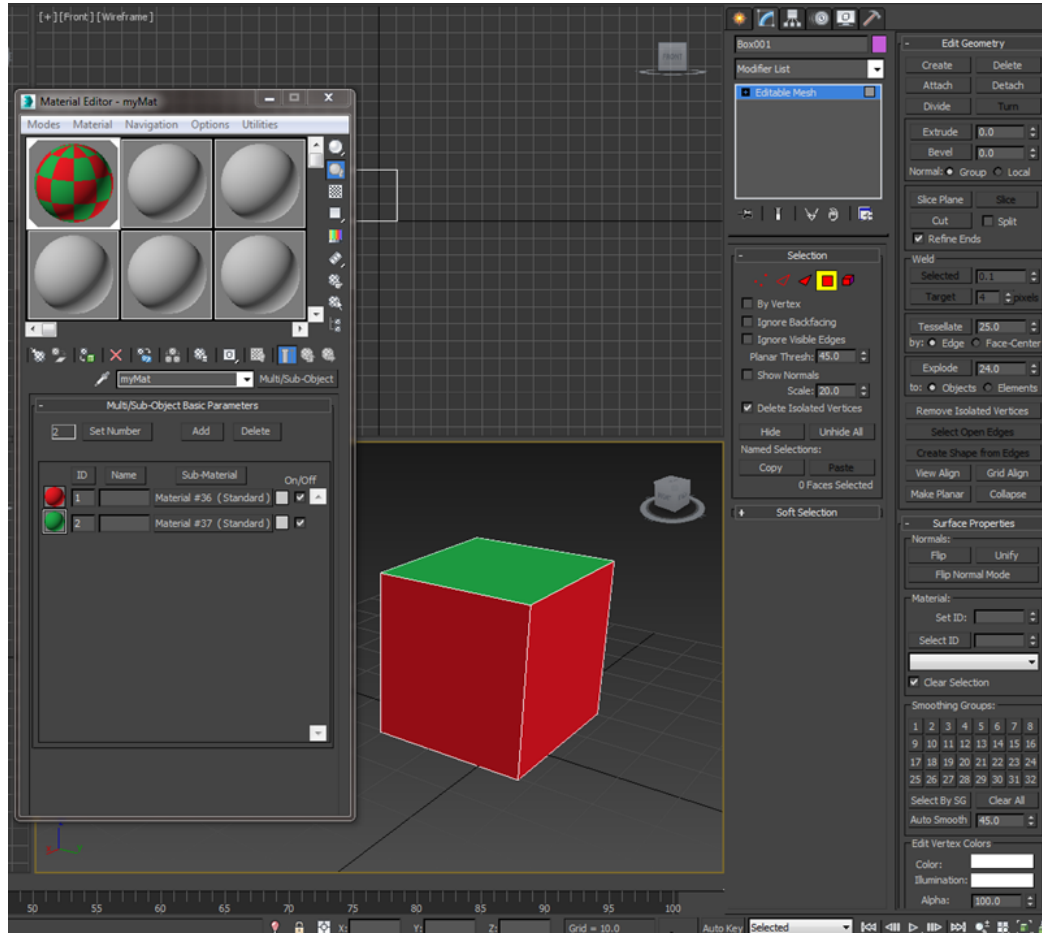
6. In the 3Ds Max **Material Editor**, under **Shader Basic Parameters**, select **Crytek Shader**.
7. Under **Maps**, next to **Diffuse Color**, select **None**.
8. In **Material/Map Browser**, under **Maps**, double-click **Bitmap**. Then double-click to select the desired image file. Afterward the image file appears in the 3DS Max **Material Editor** for the **Diffuse Color** parameter.



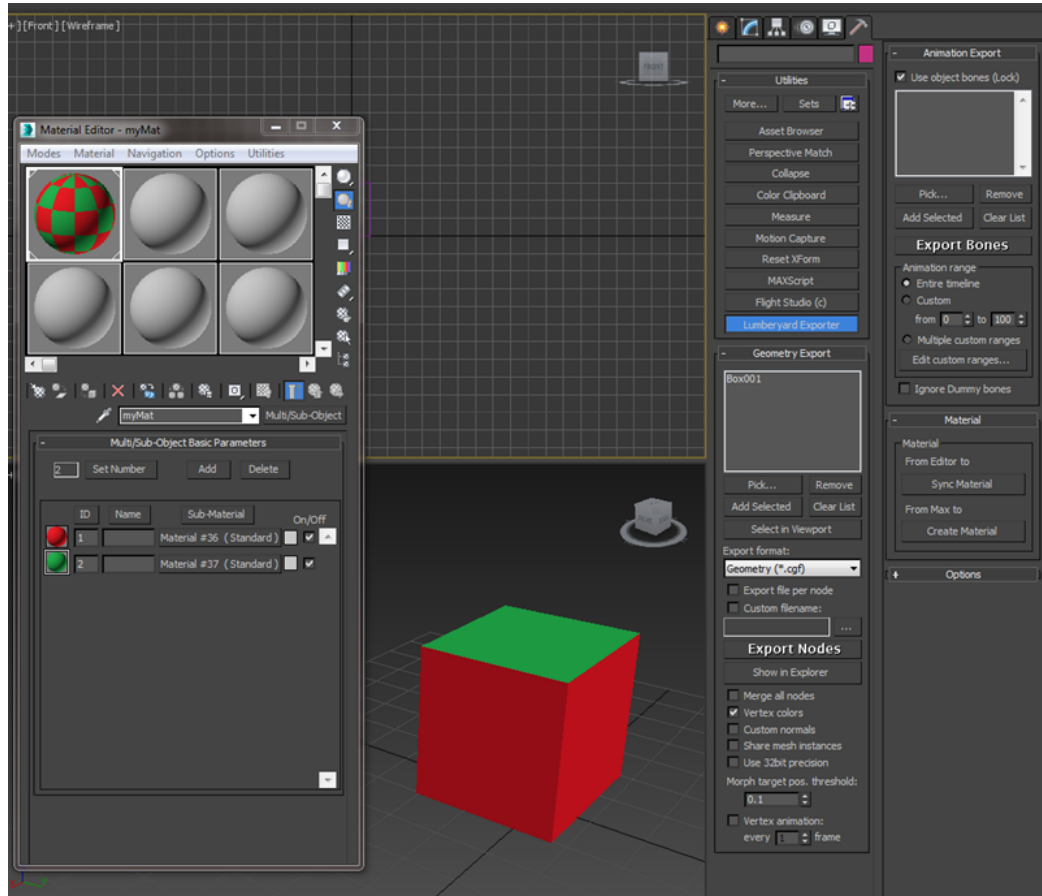
9. While still in **Material Editor**, choose **Navigation, Go to Parent**. Then repeat to get back to the material ID list.
10. Create a second subshader by repeating steps 5 through 9 for the second entry in the list. Click **Set Number**, then type 2 in the **Number of Materials** pop-up window. The list shows only two submaterials.
11. In **Material Editor**, under **Name**, type in a name.



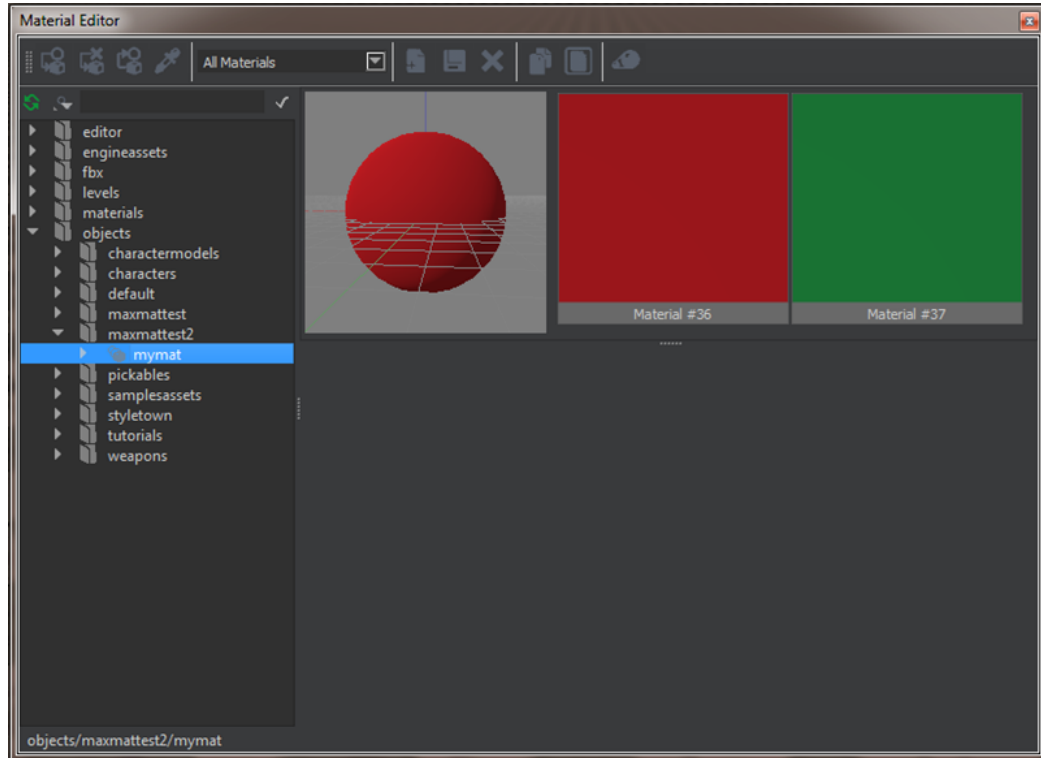
12. With the object selected in the viewport, go to **Material Editor** and choose **Material, Assign to Selection**.
13. Click the hammer icon. Under **Utilities**, select **Lumberyard Export**, select the object, and then choose **Add Selected** to place the object in the **Geometry Export** list.
14. In the 3ds Max panel on the right, under **Modifier List**, select **Editable Mesh, Polygon**.
15. In the viewport, select the top face. Then, under **Surface Properties**, click **Set ID** and set the value to 2. This makes the top face use the second material in the final material group.
16. Select the other faces and set their **Set ID** values to 1. The final face coloring should match the one shown in the following image.



17. Select **Export Nodes** to create a .cgf file.



18. Click **Create Material** to open the Lumberyard **Material Editor** and display a file dialog box.
19. Navigate to the directory where your `.cgf` files are located. Then type the same file name that you specified in 3ds Max. This ensures that the `.cgf` file can automatically find the correct `.mtl` file when loaded in the Lumberyard **Material Editor**.



20. In Lumberyard Editor, create a level and open the `.cgf`. The object should have the correct materials mapped onto its faces.

Working with Textures

Textures can be used to provide color, depth, and details to a surface. For example, a repeating brick-and-mortar texture can be used to simulate a brick wall, rather than creating geometry for each individual brick.

A texture is an image file that consists of a number of pixels, called texels, each occupying a coordinate determined by the width and height of the texture. These coordinates are then mapped into values ranging from 0 to 1 along a U (width) and V (height) axis. This process produces a 2D texture map that is stored in a `.DDS` file.

In turn, the process of mapping the UV coordinates of a texture map to the corresponding UV coordinates at the vertices on a 3D object is called UV mapping. This in effect wraps the 2D texture onto the 3D object.

Textures are dictated by, and applied by, the shader that is selected for a material. There can be multiple textures applied by the shader for a material.

Textures used in Lumberyard are usually created with Adobe Photoshop or other DCC tool.

Topics

- [Texture Map Types \(p. 1045\)](#)
- [Texture Best Practices \(p. 1045\)](#)
- [Working with Diffuse Maps \(p. 1046\)](#)
- [Working with Normal Maps \(p. 1046\)](#)
- [Working with Gloss Maps \(p. 1048\)](#)

- [Working with Detail Maps \(p. 1048\)](#)
- [Working with Decals \(p. 1049\)](#)
- [Displacement Maps and Tessellation \(p. 1053\)](#)

Texture Map Types

Source texture files are converted and compiled in .DDS format by the Resource Compiler (RC). When no presets for the source file are specified, the Resource Compiler will do the following:

- Files with a file suffix of `_ddn` or `_bump` will generate an uncompressed RGBA or U8V8 NormalMap .DDS file with height information in the alpha channel.
- Files with a non-white (less than 255) alpha channel will generate a DXT3-compressed .DDS file.
- Files without an alpha channel will generate DXT1 compressed .DDS file.

Texture Map Types

Texture Map	Filename Suffix	Description
Diffuse map	<code>_diff</code>	Used to define the main color for an object.
Normal map	<code>_ddn</code>	A type of bump map that is used to define the direction of normals on an object surface. A normal direction for each pixel of the texture is stored in the RGB normal map.
Normal with Gloss map	<code>_ddna</code>	Used to achieve physically-correct results. DDNA textures are standard DDN textures with the Gloss map stored inside the Alpha channel.
Environment map	N/A	Used to make an object reflective. The environment map stores the image that is reflected off the object.
Displacement map	<code>_displ</code>	Used in tessellations, parallax occlusion mapping (POM), and offset bump mapping (OBM) to give more depth and definition to an object.
Detail map	<code>_detail</code>	Used to add more detail to a surface. It works like a second material layer and is not affected by the mapping of the object it is used for.
Decal	N/A	Used on top of a diffuse texture for adding cracks, mud or bullet traces to an object.
Blend (layer)	N/A	Blends multiple textures using an adjustable mask texture and a vertex alpha.

Texture Best Practices

When creating textures, consider the following best practices and guidelines:

- Use the fewest number of textures that will do the job.
- For road textures, make sure the texture is horizontal.
- Use detail maps to add detail and crispness to lower-resolution textures. Detail maps can be used to add extra grain to wood, extra cracks to a concrete wall, or small scratches to car paint.

- Reuse normal maps and specular maps when possible to save texture memory. Normal maps are twice as expensive memory-wise compared to regular textures. For example, when using several types of floor tiles, brick walls, concrete walls, create textures so they can use the same normal map and specular map.
- Combine textures for small generic items such as pipes and railings to save on drawcalls. For example, a house can consist of a wall texture, roof texture and a detail sheet with all windows, frames, and doors. This will save on materials and drawcalls.
- Do not make textures bigger than they will appear onscreen. A roof texture on a tall building that either the player or the camera can see at close range should be smaller, for example, than a ground texture.
- Use decals to break up and compensate for lack of texture amount. Dirt and stain decals are an easy way to break up tiled textures.
- Use vertex colors to create variety, depth and color variations. Vertex painting and pre-baked vertex lighting is a relatively cheap way of adding depth to objects and make them look more interesting.
- Use grayscale textures that can be color-tinted to save on texture memory. Objects that can benefit from this technique include cars, fences, barrels, and crates.

Working with Diffuse Maps

When light hits a surface, it splits into two directions: some is reflected immediately off the surface while the rest enters the surface and gets refracted. The refracted light can be absorbed or scattered underneath the surface and exit again at a different angle. This absorbed and refracted light is the diffuse color of an object.

The diffuse color defines how bright a surface is when lit directly by a white light source with an intensity of 100%. Physically speaking, it defines what percentage for each component of the RGB spectrum does not get absorbed when light scatters underneath the surface.

Texture mapping the diffuse color is like applying an image to the surface of the object. For example, if you want a wall object to be made out of brick, you can choose an image file with a photograph of bricks. A diffuse map is always required for objects.

The diffuse map should not contain any lighting, shading or shadowing information, as all this gets added dynamically by Lumberyard. In certain cases, pre-baked ambient occlusion (AO) is required, which is stored in a dedicated AO map in the diffuse channel of the Detail Map. For more information, see [Working with Detail Maps \(p. 1048\)](#).

Diffuse maps can be combined with other texture maps, such as ambient occlusion maps and cavity maps, to create more definition.

Diffuse Mapping Best Practices

- Don't use too light or too dark of a texture that will require too much color compensation.
- Metal objects should have a black diffuse color. Rusty metal however needs some diffuse color.
- Paint, or use occlusion mapping, to darken cracks and holes.
- Use crisp colors and contrast to define variations in shapes in order to break up the image.
- Create UV maps so that there is a decent compromise of space utilization and stretching.

Working with Normal Maps

The illusion of extra depth and detail to objects is achieved by using normal maps, which are a type of bump map. Bump maps and normal maps both add detail without increasing the number of polygons. As such, they are used to "fake" depth and details such as wrinkles, scratches and beveled edges.

Unlike displacement mapping, normal maps affect shading and not the surface itself. The surface remains flat when seen from an angle.

Bump maps store an intensity that represents the relative height (bump) of pixels from the viewpoint of the camera. Traditional normal maps, in addition to storing the height, also store the direction of normals in the RGB values of the texture image. As such, they are more accurate than bump maps.

Lumberyard uses a form of normal mapping, called Tangent Space Normal Mapping, which uses either a height map or is derived from a high-polygon model. In a normal map, a color represents a certain normal vector (surface orientation of a point). For tangent space normal maps the information is relative to the underlying surface.

Tangent space normal maps are independent of the underlying geometry which means the texture can be used on other geometry as well. It will automatically align to the surface regardless of mirroring, rotation, scale or translation. Only the latter two are supported by traditional (object or world) normal maps.

An advantage of tangent space normal maps is that the normals are always pointing outwards, so assuming unit length, the normal z coordinate can be reconstructed from the x and y components. After the coordinate expansion from 0..1 to the -1..1 range, the z component can be computed in the shader with this formula: $z = \sqrt{1 - x^2 - y^2}$. This makes it possible to use two-channel textures (2 bytes per texel) to store normal maps.

Topics

- [Normal Mapping Best Practices \(p. 1047\)](#)
- [Using Normals with Gloss Maps \(p. 1047\)](#)

Normal Mapping Best Practices

The following represent some best practices for consideration when creating normal maps:

- Not all colors represent valid normals. Do not apply bicubic filter, sharpening, or alpha blending with normal maps. Use the CryTIF plugin to visualize such problems.
- If you have to resize a normal map, use bi-linear instead of bi-cubic interpolation.
- Use the CryTIF exporter presets when saving normal maps.
- Lay out UV maps so that there is a decent compromise between space utilization and stretching.
- Render normal maps using tangent-basis calculations with swizzle coordinates in X+Y-Z+.
- Mirroring UVs requires extra work to hide with normal. Be sure the normal directions are the same across seams.
- Give normal maps an extruded edge of pixels. That way, there won't be any errors generated by mip-map levels.
- Do not anti-alias normal maps against the background.
- Don't apply color filters or other manipulation such as grain or noise to a normal map itself.

Using Normals with Gloss Maps

Most materials should have a gloss map as well as a normal map as this can impart a lot of variation to the shading. Gloss maps are closely related to normal maps, as high frequency details in a normal map can create some roughness as well. However, gloss is more the microscale roughness of the material while normal represents macro-scale bumpiness. Gloss maps are treated like diffuse maps.

The Gloss map always goes into the Alpha channel of the Normal map, even if you're using a specular map for metals and metal-embedded surfaces.

If the preset **NormalMapWithGlossInAlpha_highQ** is selected, the Resource Compiler will automatically adjust the gloss map stored in the alpha channel based on the normal variance and lower

the gloss where normals are very bumpy. This can greatly help to reduce shimmering and sparkling highlights artifacts.

Lumberyard uses DDNA textures, which is a standard DDN texture with the addition of a Gloss map in the alpha channel of the normal map. DDNA texture map must use the `_ddna.dds` filename suffix (instead of `_ddn.dds`) for the Resource Compiler to recognize the texture correctly.

Working with Gloss Maps

Gloss defines the roughness of a surface. A low gloss value means that the surface is rough while a high value means the surface is smooth and shiny. The roughness influences the size and the intensity of specular highlights. The smoother and glossier a surface is, the smaller the specular highlight will be. A smaller highlight will at the same time be brighter in order to obey to the rules of energy conservation.

For physically-based shaders, the gloss map is highly important. Most materials should have a gloss map as well as a normal map as this can impart a lot of variation to the shading. Gloss maps are closely related to normal maps, as high frequency details in a normal map can create some roughness as well. However, gloss is more the microscale roughness of the material while normal represents macro-scale bumpiness. Gloss maps are treated like diffuse maps.

The Gloss map always goes into the Alpha channel of the Normal map, even if you're using a specular map for metals and metal-embedded surfaces.

Gloss mapping is more powerful than the traditional specular mask, as gloss influences not only the brightness of a highlight but also its size and the sharpness of reflections.

When working with textures, gloss maps and normal maps are created first, then diffuse maps. Diffuse maps should contain no lighting information.

The gloss map is always stored in the alpha channel of the normal map. If the preset **NormalMapWithGlossInAlpha_highQ** is selected, the Resource Compiler will automatically adjust the gloss map stored in the alpha channel based on the normal variance and lower the gloss where normals are very bumpy. This can greatly help to reduce shimmering and sparkling highlights artifacts.

Gloss Map Best Practices

- Put variation into the gloss map. Not just random noise but really where the object would be less or more rough.
- If an object has the correct physical specular color but does not show specular highlights on top of the diffuse, the gloss is likely set too low. Increase the brightness of the gloss map.
- The Glossiness value must be set to 255, otherwise the gloss map will not work.
- Non-metals should have a specular color value between 53 and 61, based on what looks the best.
- For metals (and for metal parts embedded in non-metals), a dedicated specular texture map is used, with the gloss map going into the alpha channel of the normal map. The gloss map defines the smoothness, reflectivity and tightness of specular highlights. For metals, the shader doesn't control specular color – the specular texture map does. Specular color is physically based. Because of this, set the Specular color value to 255.

Working with Detail Maps

Detail mapping is a simple technique to add macro surface detail at relatively low cost, memory and performance wise. The following best practices should be taken into consideration:

- Use as low a resolution as possible for best performance (512x512 or lower).
- Prevent artifacts by using a higher tiling scale.

- Decrease contrast for the detail diffuse and gloss.

Unified detail mapping (UDM) is basically a reversed detail map. Usually the detail map is used for finer details as you get closer. UDM is the opposite. It helps to define big shapes viewed from the distance. Since close-up detail is provided in tiled textures, larger details are needed to define shapes better when viewed from a distance.

Setting Up Detail Map Textures

Detail map parameters are setup in the Material Editor.

To set Detail Map parameters

1. In Lumberyard Editor, click **View, Open View Pane, Material Editor**.
2. In the left tree, select an applicable texture.
3. In the right pane, under **Shader Generation Params**, click the **Detail Mapping** check box.
4. Under **Shader Params**, set values for the following parameters.
 - a. **Detail bump scale**: Defines how much the normal map is visible. The higher the value, the more the normal map will show through.
 - b. **Detail diffuse scale**: Defines how much the diffuse map (or AO map) visible. The higher the value, the more the normal map will show through.
 - c. **Detail gloss scale**: Defines how much the gloss map is visible. The higher the value, the more the gloss map will show through.

Working with Decals

Decals are non-repeating images or textures that are applied to the surface of an object or terrain with a specified projection. Common examples of decals are product labels and logos, artwork for walls, signs, and surface cracks.

Decals can break up uninteresting textures and bring together such level elements as brushes and terrain. Good decal placement can also create seamless transitions between many different objects. Decals only work with the [Illum Shader \(p. 1015\)](#).

Note

If you apply decals to an object that can be moved by a player, the decal will not move with the object.

Topics

- [Decal Projection Types \(p. 1049\)](#)
- [Placing a Decal \(p. 1050\)](#)
- [Setting Decal Parameters \(p. 1051\)](#)
- [Debugging Decal Mapping Issues \(p. 1052\)](#)

Decal Projection Types

Decals have several different projection types. To change projection type, select the decal and change the **ProjectionType** value.

Planar Projection

Planar projection is the cheapest performance-wise. The decal is displayed in the same location as the center of the object. Use planer projection only on flat surfaces, otherwise the decal may appear to be floating.

Deferred Projection

Deferred projection is a simple method to get decals to follow the contours of objects and is similar to Planar projection, but slower. As such, use Planar projection wherever possible.

Deferred projection is enabled by selecting the **Decal Params, Deferred** check box.

ProjectOnTerrain Projection

The decal is projected directly onto the terrain, ignoring any objects that might otherwise receive the projection.

ProjectOnStaticObjects Projection

The decal is projected onto the geometry of an object along the opposite direction of the blue Z axis. This method is automatically done as a deferred pass.

ProjectOnTerrainAndStaticObjects Projection

A combination of ProjectOnStaticObjects and ProjectOnTerrain, the decal is displayed on both the terrain and on objects. This method is automatically performed as a deferred pass.

Placing a Decal

Do the following to place a decal in your level.

Note

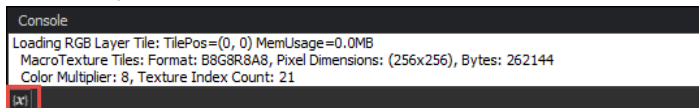
If you apply decals to an object that can be moved by a player, the decal will not move with the object.

To place a decal

1. In Lumberyard Editor, click the top **Follow Terrain** button.
2. In Rollup Bar, click **Objects, Misc, Decal**.
3. Drag the detail into the level and then click to place it.
4. Using the **Edit** menu, move, rotate, or scale the decal as needed.
5. To place a decal manually, select the **Reorientate** check box, and use mouse shortcuts to place the decal as follows. This can speed up placement enormously.
 - a. **Ctrl+Click** : Move the decals to the desired position
 - b. **Alt+Click** : Scales the decal along the X, Y axes
 - c. **Ctrl+Alt+Click** : Rotates the decal around the Z axis

To place a decal on vegetation

1. Enable deferred projection so the decal follows the contours of the vegetation:
 - a. In Lumberyard Editor, in the **Rollup Bar**, under **Objects**, click **Misc, Decal**.
 - b. Under **Decal Params**, select the **Deferred** check box. For information about projection types, see [Decal Projection Types \(p. 1049\)](#).
2. Enable the `r_deferredDecalsOnDynamicObjects` console variable so the decal appears on the vegetation:
 - a. In Lumberyard Editor, click the **X** icon in the **Console** section.



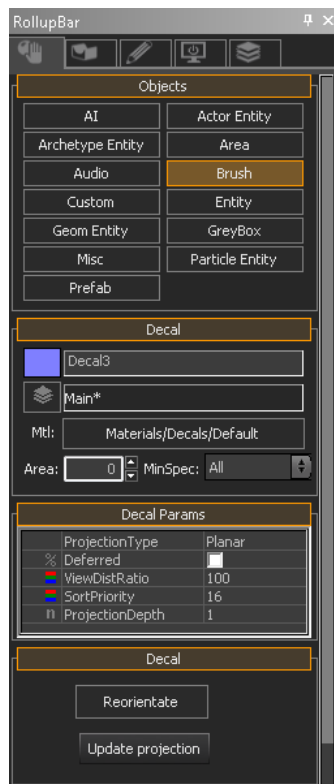
- b. In the **Console Variables** window, search for `r_deferredDecalsOnDynamicObjects`
 - c. Set the value to any positive number, for example 1.
 - d. Close the **Console Variables** window to save the new value.
 3. Follow the instructions above for placing a decal.

Setting Decal Parameters

Complete the following procedures for setting decal mapping parameters.

To set decal parameters in the Rollup Bar

Most of the decal parameters can be found in the Rollup Bar.

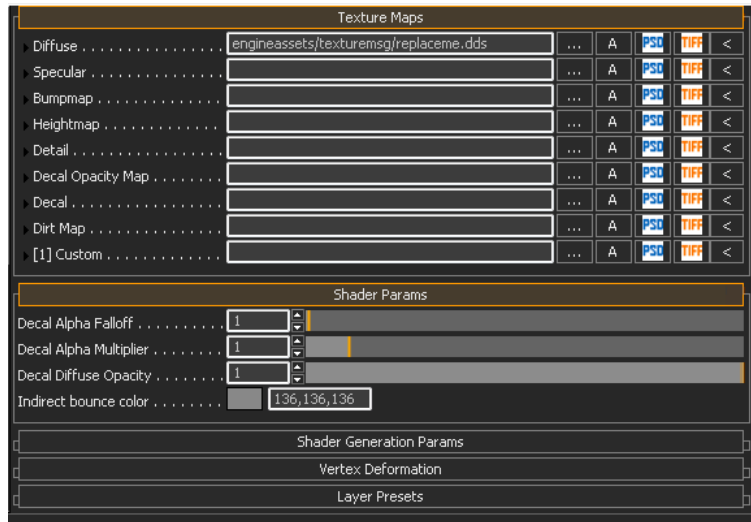


To set decal parameters in the Rollup Bar

1. In the Rollup Bar, under **Objects**, click **Misc, Decal**.
2. Under **Decal Params**, adjust the following parameters:
 - **ProjectionType** – Values range from 0 to 3, corresponding to Planar, ProjectOnStaticObjects, ProjectOnTerrain, and ProjectOnTerrainAndStaticObjects.
 - **Deferred** – Select to enable deferred decal projection.
 - **View Distance Multiplier** – Set the distance at which the decal is visible. The default value is 1. A higher number indicates a longer visibility distance.
 - **SortPriority** – Specifies if the decal will appear on top of another decal.

To set shader decal parameters

A few decal parameters are set using the **Shader Params** panel in Material Editor.



To set decal mapping parameters

1. In Lumberyard Editor, click **View, Open View Pane, Material Editor**.
2. Click the top **Add New Item** button.
3. Select a decals folder, select a subfolder, and then click **Save**. The new material will automatically be selected with the default settings.
4. Under **Shader Generation Params**, select the **Decal** check box.
5. Right-click the decal you created and click **Assign to Selected Objects**.
6. Under **Shader Params**, adjust the values of the following parameters.
 - a. **Decal Alpha Falloff**: Power applied to the decal alpha.
 - b. **Decal Alpha Multiplier**: Multiplier applied to the decal alpha.
 - c. **Decal Diffuse Opacity**: Opacity multiplier for fading out decal diffuse color.

Debugging Decal Mapping Issues

Use the following to debug decals.

Debugging Deferred Decals

The cost of a deferred decal depends on how many objects it will project, how expensive the geometry is, and how many overdraws it will create.

Any deferred decal in the viewport in Lumberyard Editor renders in red, green and blue. These colors show how expensive a deferred decal is for rendering. Place any deferred decals in such a way that they are displayed mostly in blue

- Red = expensive
- Green = medium
- Blue = cheap

Debugging Decal Flicker

If a placed decal is flickering, follow these guidelines to ensure that it has been properly set up.

- Check that all sub-materials have the **Decal** check box selected under **Shader Generation Params** in the Material Editor.

- If still flickering, check for overlapping layers that have the **Decal** check box selected. Use the **SortPriority** parameter to specify which decal will appear on top of the other.
- Other than for decals, the mesh shouldn't have overlapping triangles. Do not offset along the surface normal, they can still break in some situations and will introduce floating parallax effects.

Displacement Maps and Tessellation

Displacement mapping allows you to displace the actual surface geometry of an object to give you extra depth and detail than is available using bump mapping, offset bump mapping or parallax occlusion mapping (POM) techniques, which all "fake" surface detail. Displacement mapping results are dependent on how far the camera is from the object.

Displacement mapping uses a texture map, called the height map, which is used to define the value of vertex height displacement. Specifically, this is a scalar displacement that is stored in the alpha channel of a `_displ` texture file.

In order for displacement mapping to work correctly, you need to also apply tessellation to your object, otherwise there wouldn't be enough geometry to displace. Tessellation increases the geometry count by subdividing polygons into smaller polygons before it gets displaced.

Topics

- [Displacement Mapping Best Practices \(p. 1053\)](#)
- [Setting Displacement Mapping Parameters \(p. 1053\)](#)
- [Tessellation \(p. 1054\)](#)

Displacement Mapping Best Practices

Review the following guidelines and best practices for consideration when creating displacement maps and tessellated geometry.

- Height maps must be stored using the `_displ` suffix (such as `road_displ.tif` for example).
- Do not place the height map in the alpha channel of the normal map. Rather, place the displacement map in the alpha channel of the `_displ` texture. The RGB channels can thus be left empty.
- Set the diffuse and normal texture map textures as usual in the Material Editor. The `_displ` texture will be loaded automatically by checking the name of the texture in the **Bumpmap** (normal) map slot and that there is a corresponding `_displ` texture for it.
- Save the `_displ` texture using the Photoshop CryTIF plugin. The will write the correct metadata to a `.tif` file for it to be converted to a `.dds` file at runtime. In some cases you may need to click **Generate Output** in the dialog box of the plugin.
- When using the CryTIF plugin, use the DisplacementMap preset to store `_displ` textures. Height maps will be converted to A8 textures. If you don't see any displacement, double check the format in the Material Editor texture file dialog preview. If it isn't in A8 format, fix the preset, save and reload.
- Ensure that **Config Spec** is set to **Very High**.
- To enable tessellated shadows for tessellated geometric entities, use the `e_ShadowsTessellateCascades=1` console variable, but keep in mind this comes at a performance cost.

Setting Displacement Mapping Parameters

To apply displacement mapping to an object

1. In Lumberyard Editor, click **View, Open View Pane, Material Editor**.

2. In the left tree, select the desired asset.
3. In the right pane, under **Shader Generation Params**, select **Displacement mapping**.
4. Under **Shader Params**, adjust the values of the following parameters for the desired effect.
 - a. **Displacement bias**: Moves the plane where the displacement is applied. This reduces gaps in meshes, and prevents objects from displacing other objects that are placed above them.
 - b. **Displacement height scale**: Changes the overall height of the displacement.

Tessellation

In order for displacement mapping to work correctly, tessellation is also required, otherwise there wouldn't be enough geometry to displace. Tessellation increases the geometry count by subdividing polygons into smaller polygons before it gets displaced. Phong and PN triangles are the two available tessellation methods.

Phong tessellation approximates smoothing based on surface normals. Surfaces with Phong tessellation applied are not perfectly smooth across patch boundaries, causing the object to look inflated.

PN triangle tessellation is similar to Phong tessellation and is slower, but with better approximation.

Tessellation is only supported for the [Illum Shader \(p. 1015\)](#) and [HumanSkin Shader \(p. 1013\)](#).

Topics

- [Setting Tessellation Parameters \(p. 1054\)](#)
- [Fixing Tessellation Seams \(p. 1055\)](#)

Setting Tessellation Parameters

To apply tessellation to an object and set parameter values, complete this procedure.

To apply tessellation to an object

1. In Material Editor, click **View, Open View Pane, Material Editor**.
2. In the left tree, click to select the desired asset.
3. In the right pane, under **Shader Generation Params**, select either **Phong tessellation** or **PN triangles tessellation**.
4. Under **Shader Params**, adjust the values of the following parameters.

Tessellation Parameters

Parameter	Description
Tessellation face cull	<p>Specifies the extent to which vertices are culled. Because tessellation uses its own face culling, it takes the original (non-tessellated) triangle and checks if it's facing the camera; if not it discards it.</p> <p>This can also be used for 2-sided sorting of polygons. In this case, the 2 Sided check box must also be selected under Advanced in the Material Editor.</p> <p>An issue may arise when there is displacement that is visible from the camera. For example, a bump on a cube that is rotating is still</p>

Parameter	Description
	visible for a while, even though the cube face is no longer facing the camera. Setting this parameter to 0 means no face culling at all, while setting it to 1 will cull anything not facing the camera.
Tessellation factor	Specifies the density of the mesh triangles
Tessellation factor max	Used for objects that are at a fixed distance or range from the camera to get rid of geometry “popping” artifacts. This is useful for cutscenes.
Tessellation factor min	Setting this value to 1 means that it will be always tessellated at level 1, even if the object is far away from camera.

Fixing Tessellation Seams

There are two types of seams or cracks that can become noticeable when using tessellation.

Border Seams

Border seams occur when different meshes are placed close to each other, or when a mesh consisting of sub-meshes causes unpleasant cracks because of using different materials with different displacement (or even same displacement maps with slightly different UV mapping).

The solution involves carefully placing meshes or fade-out displacement by modifying the displacement map as needed.

UV Seams

UV seams occur when two adjacent triangles share an edge but use separate vertices with different UVs. This shared edge will have a different displacement on each side due to sampling different places in the displacement map. Even tiny differences in UV can cause visible seams. This is automatically fixed by Lumberyard if there is no tiling. Otherwise you must change the UV mapping to hide such artifacts where possible.

Phong tessellation and PN Triangle tessellation do not suffer from UV seams as they do not use UV mapping.

Working with Substances

Substances are procedural materials created using Allegorithmic's Substance Designer. Lumberyard has the ability to import Substance `.sbsar` files using the Substance Editor.

Creating Substances for Lumberyard

When creating Substances for Lumberyard using Allegorithmic's Substance Designer, it is recommended to use the **PBR Specular/Glossiness** Substance as the base. This will involve less adjustments to your default outputs for Substances. However, you will need to delete the **Glossiness** output and save the **Gloss** map into the alpha channel for the **Normal** map output in Substance Designer.

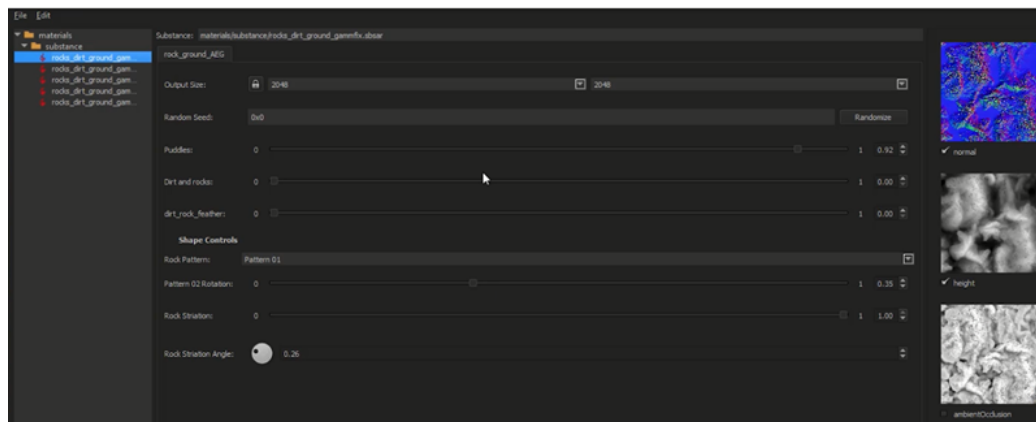
If you want to use a **PBR Metallic/Roughness** Substance and convert it for use in Lumberyard, follow these steps:

- Change the **BaseColor** output node to **Diffuse**.

- Create a **Specular** output node in the Substance Graph.
- Create a **RGB-A Merge** node in the Substance Graph.
 - Connect the node that was originally going into the **Normal** map into the **RGB** input.
 - Note that the **A (Alpha)** input for this node will be connected later on.
 - Connect the output of this merge node into the input for the **Normal** output node.
- Create a **BaseColor/Metallic/Roughness** converter node in the Substance Graph.
 - Connect the node that was originally going into the **BaseColor/Diffuse** map into the **BaseColor** input for this converter node.
 - Connect the node that was originally going into the **Roughness** map into the **Roughness** input for this converter node.
 - Connect the node that was originally going into the **Metallic** map into the **Metallic** input for this converter node.
 - Connect the **Diffuse** output of this converter node into the input for the **Diffuse** output node.
 - Connect the **Specular** output of this converter node into the input for the **Specular** output node.
 - Connect the **Glossiness** output of this converter node into the **A (Alpha)** input for the **RGB-A Merge** node.
- Delete the **Roughness** output node.
- Delete the **Metallic** output node.
- Save the changes to your Substance and then publish the `.sbs` as a `.sbsar` to be imported into Lumberyard.

Working with Substance in Lumberyard

Using Substance Editor, you can edit Substance material properties and visualize substances on objects in real-time. Substance Editor also has the ability to generate and export static textures from Substances.



Here are some things to keep in mind when working with Substances in Substance Editor:

- The Substance Gem needs to be enabled first for the project using [Project Configurator \(p. 985\)](#). For more information on Gems, see [Gems \(p. 778\)](#).
- When importing substance files, you must restart Lumberyard Editor before substance textures are rendered correctly.
- A `.smt.1` (substance material) file and a `.sub` (substance texture) file are generated in the same directory location as the imported `.sbsar` for applying the substance material or substance textures to objects.

- By default, an `.smtl` file will inherit the `.sub` files in the appropriately matching channels based on the outputs in the published `.sbsar` from Substance Designer. For example, a diffuse output texture will map into the diffuse channel for the `.smtl` file.

To use Substance Editor

1. Open Lumberyard Editor and select **View, Open View Pane, Substance Editor**. You can also click the Substance icon in the main toolbar of Lumberyard Editor.
2. To update imported `.sbsar` files, click **Edit, Reimport Substance**. Current changes will not be overwritten.
3. To remove a substance, click **File, Delete Substance**.

Note

This permanently removes the substance and all associated assets from the `.sbsar` project, which cannot be recovered using the Windows Recycle Bin.

Parallax Mapping

Parallax occlusion mapping (POM) is an enhancement of the traditional parallax mapping technique that is used to procedurally create detail in a texture adding the illusion of depth. This depth perception changes based on perspective.

Parallax Occlusion Mapping (POM) and Offset Bump Mapping (OBM) are both similar to displacement mapping and tessellation, but not as expensive performance-wise as the geometry is not increased. However, due to the way in which POM works, it will not always be suitable for every situation.

Use POM for high-spec computers only, and use OBM for anything else, consoles. When using POM, you must enable both enable both shader generation parameters. Lumberyard will automatically default to using OBM for setups cannot run POM.

Topics

- [Parallax Mapping Best Practices \(p. 1057\)](#)
- [Applying Parallax Occlusion Mapping \(POM\) \(p. 1058\)](#)
- [Applying Silhouette Parallax Occlusion Mapping \(SPOM\) \(p. 1058\)](#)
- [Using Blend Layers for Parallax Mapping \(p. 1058\)](#)

Parallax Mapping Best Practices

Review the following guidelines and best practices for consideration when applying POM or SPOM parallax mapping.

- Height maps must be stored using the `_displ` suffix (such as `road_displ.tif` for example).
- Do not place the height map in the alpha channel of the normal map. Rather, place the displacement map in the alpha channel of the `_displ` texture. The RGB channels can thus be left empty.
- Set the diffuse and normal texture map textures as usual in the Material Editor. The `_displ` texture will be loaded automatically by checking the name of the texture in the **Bumpmap** (normal) texture map slot and that there is a corresponding `_displ` texture for it.
- Save the `_displ` texture using the Photoshop CryTIF plugin. This will write the correct metadata to a `.tif` file for it to be converted to a `.dds` file at runtime. In some cases you may need to click **Generate Output** in the dialog box of the plugin.
- When using the CryTIF plugin, use the **DisplacementMap** preset to store `_displ` textures. Height maps will be converted to A8 textures. If you don't see any displacement, double check the format in the Editor's texture file dialog preview. If it isn't in A8 format, fix the preset, save and reload.

- Ensure that **Config Spec** is set to **High** or **Very High**.

Applying Parallax Occlusion Mapping (POM)

To apply POM, complete the following procedure.

To apply Parallax Occlusion Mapping

1. In Lumberyard Editor, click **View, Open View Pane, Material Editor**.
2. In the left tree, select the desired asset.
3. In the right pane, under **Shader Generation Params**, select **Offset bump mapping** and **Parallax occlusion mapping**.
4. Under **Shader Params**, adjust the values of the following parameters.
 - a. **Height bias**: Moves the plane where the displacement is applied. This reduces gaps in meshes, and prevents objects from displacing other objects that are placed above them.
 - b. **POM Displacement**: Sets the POM depth. A larger value adds more depth.
 - c. **Self shadow strength**: Changes the strength of self-shadowing. A larger value imparts more shadowing
5. Under **Texture Maps**, enter the paths to the various textures.

Applying Silhouette Parallax Occlusion Mapping (SPOM)

To apply SPOM, complete the following procedure.

To apply Silhouette Parallax Occlusion Mapping

1. In Lumberyard Editor, click **View, Open View Pane, Material Editor**.
2. In the left tree, select the desired asset.
3. In the right pane, under **Shader Generation Params**, select **Parallax occlusion mapping with silhouette**.
4. Under **Shader Params**, adjust the values of the following parameters.
 - a. **Height bias**: Moves the plane where the displacement is applied. This reduces gaps in meshes, and prevents objects from displacing other objects that are placed above them.
 - b. **Self shadow strength**: Changes the strength of self-shadowing. A larger value imparts more shadowing
 - c. **Silhouette POM Displacement**: Sets the SPOM depth. A larger value adds more depth.
5. Under **Texture Maps**, enter the paths to the various textures.

Using Blend Layers for Parallax Mapping

You can use blend layers for parallax mapping. For both POM and OBM, set the diffuse and normal map as usual. The `_disp` texture will be loaded automatically as long as the [Applying Parallax Occlusion Mapping \(POM\) \(p. 1058\)](#) procedure is first completed.

When using a second blend layer, the diffuse map is placed in the **Custom** texture map slot, the normal map is placed in the **[1] Custom** slot, and the height map is placed in the **SubSurface** slot.

To use a blend layer for parallax mapping

1. Complete the [Applying Parallax Occlusion Mapping \(POM\) \(p. 1058\)](#) procedure.
2. Under **Shader Generation Params**, select **Parallax occlusion mapping** and **Blendlayer**.

3. Under **Texture Maps**, place maps as follows:
 - a. Place the height map in **Second Height Map**.
 - b. Place the height map in **Second Diffuse Map**.
 - c. Place the height map in **Second Bump Map**.
4. Under **Shader Params**, adjust the values of the parameters as needed.

Using Vertex Colors

Vertex color, or vcolor, is just a color with RGB and alpha channel values stored for each vertex of a mesh. Vertex color and alpha can be used for multi-texturing, transparency, or fake ambient occlusion.

Vertex color is typically multiplied against the Diffuse color, coloring or darkening the color map.

When used for non-color effects, typically each color channel is treated as a separate monochrome set of values, so for example vertex color can control three different per-vertex effects.

Vertex Colors is a Shader Generation parameter that can be enabled using the Material Editor, which is part of Lumberyard Editor.

For a good application of vertex colors, see [Defining Vegetation Vertex Colors \(p. 879\)](#).

Customizing Post-Processing Effects

Lumberyard includes post-processing effects that can help improve your game's graphics, lighting, and transitions between effects such as color correction, bloom, and depth of field.

Use XML files with Flow Graph or Lua scripts to customize effects by setting their parameters. You can create prioritized groups of effect parameters in XML and enable or disable them using a Flow Graph node or Lua scripting.

You can also use effect groups to specify the following:

- Blend curves to smoothly transition between effects
- Stay enabled until explicitly disabled
- Make effect strength based on distance from the camera

Note

Creating a new effect requires modifying Lumberyard, while creating a new effect group does not.

Topics

- [Post-Effect Group XML Files \(p. 1059\)](#)
- [Enabling and Disabling Effect Groups \(p. 1060\)](#)
- [Specifying a Blend Curve for Smooth Effect Transitions \(p. 1061\)](#)
- [Setting Effect Strength Based on Camera Distance \(p. 1061\)](#)

Post-Effect Group XML Files

When you open Lumberyard Editor, the effect group located at `\Engine\Libs\PostEffectGroups\Default.xml` automatically loads. The `Default.xml` file includes all available effects and the default values for each parameter. You can modify the default values and copy and paste sections of the `Default.xml` file into custom effect groups.

Example XML file:

```
<PostEffectGroup priority="1" hold="1">
  <Effect name="Global">
    <Param name="User_Brightness" floatValue="0.5"/>
  </Effect>
</PostEffectGroup>
```

Priority

Non-negative integer used to set priorities. Larger priorities override smaller priorities. If multiple effect groups that are enabled have the same priority value, the effect group that was enabled later has the higher priority.

Default value: 0 (for Time of Day and Flow Graph nodes that set effects)

Valid values: 0 - 999

Hold

Indicates if the effect should stay enabled until explicitly disabled.

Default value: 0

Valid values: 0=effect is disabled after blending is complete | 1=effect remains enabled until explicitly disabled

When creating custom effect groups, we recommend creating a directory called `\PostEffectGroups` under `/Engine/Libs`. You can then load the post effect group XML files from any valid CryPath location.

Enabling and Disabling Effect Groups

You can enable and disable effect groups using Flow Graph or Lua scripting.

To enable or disable effect groups using Flow Graph

1. In Lumberyard Editor, open your level.
2. In the menu bar, select **View, Open View Pane, Flow Graph**.
3. In the **Flow Graph** editor, in the menu bar, select **File, New**.
4. In the graph pane, right-click and select **Add Node, Image, EffectGroup**. The **Image:EffectGroup** node should be visible.
5. In the **Image:EffectGroup** node, double-click **GroupName=**. Type the file path for CryPath (example: `Libs\PostEffectGroups\ExtraBright.xml`) and press **Enter**.
6. Connect your ports to the **Enabled** or **Disabled** ports in the **Image:EffectGroup** node.
7. Optionally set a value for **Choose Entity**. For more information, see Fade Distance.
8. Close the **Flow Graph** Editor.

To enable or disable effect groups using Lua

Run the following:

```
System.CachePostFxGroup( "Libs/PostEffectGroups/MyEffectGroup.xml" )
  (Optional) The XML file loads on demand if the function isn't called.
System.SetPostFxGroupEnable( "Libs/PostEffectGroups/MyEffectGroup.xml", true)
  Valid values (second parameter): true = enable | false = disable
System.GetPostFxGroupEnable( "Libs/PostEffectGroups/MyEffectGroup.xml" )
  Return values: true = effect group is enabled | false = effect group is disabled | nil = effect
  group cannot be found
```

Note

You can manually enable or disable an effect group in Lumberyard Editor by running the Lua functions in the Console window. Be sure to prepend each command with the # character to indicate a Lua command.

Specifying a Blend Curve for Smooth Effect Transitions

You can use `BlendIn` and `BlendOut` tags to specify a blend curve that enables smooth transitions between effects.

An example XML file with added `BlendIn` and `BlendOut` tags:

```
<PostEffectGroup priority="1" hold="1">
  <Effect name="SunShafts">
    <Param name="RaysAmount" floatValue="0.2"/>
  </Effect>
  <BlendIn curve="smooth">
    <Key time="0" value="0"/>
    <Key time="0.5" value="1"/>
  </BlendIn>
  <BlendOut curve="smooth">
    <Key time="0" value="1"/>
    <Key time="0.5" value="0"/>
  </BlendOut>
</PostEffectGroup>
```

Priority

Indicates how much the effects should override the lower priority values.

Hold

Determines when the `BlendIn` and `BlendOut` curves play and whether the effect group is enabled or disabled.

Valid values:

0 = Plays the `BlendOut` curve immediately after the `BlendIn` curve finishes playing; when the `BlendOut` curve plays, the effect group is disabled

1 = Plays the `BlendIn` curve; when the `BlendIn` curve plays, the effect group fully overrides lower priority values until the effect group is explicitly disabled

Curve

Available curve types are smooth, linear, and step. If a curve attribute value is not specified, the curve type defaults to smooth. You can include as many key frames in a curve as desired.

Default curve value: smooth

Valid key time values: smooth, linear, step

Key time

Valid values: 0 - 1 (seconds)

Setting Effect Strength Based on Camera Distance

You can use the `fadeDistance` attribute to set the effect strength based on the distance from the camera.

Example opening XML tag using the `fadeDistance` attribute:

```
<PostEffectGroup priority="1" fadeDistance="20">
```

fadeDistance – Indicates how the effects are actualized based on the distance of the camera from the entity.

- When the camera is at the position of the entity, the effects are fully overridden.
- When the camera is less than fade distance from the entity, the effects are blended.
- When the camera is at least fade distance from the entity, the effects are set to the lower priority values.

You can specify an entity in the Flow Graph node and assign it to the graph entity by right-clicking the node and selecting **Assign graph entity**.

To enable an effect group using Lua, set the position at which to apply the effect by using the following function:

```
System.ApplyPostFxGroupAtPosition("Libs/PostEffectGroups/MyEffectGroup.xml",  
self:GetPos())
```

where *self* is the current entity.

This function must be called once per frame while the effect group is enabled. If this function is called multiple times in a single frame, the effect strength increases each time, as if each call applies the effect from a different entity.

Lighting and Shadows

Lumberyard uses physically-based lighting and shading models to implement global illumination and lighting.

For information about the Light entity and the Environment Probe entity used in environment lighting, see [Light Entities \(p. 461\)](#).

For information about using the Time of Day Editor to simulate the changing lighting effects caused by the sun moving across the sky, see [Creating Time of Day Sky Effects \(p. 864\)](#).

Topics

- [Environment Lighting \(p. 1062\)](#)
- [Environment Shadows \(p. 1067\)](#)

Environment Lighting

Lumberyard uses physically based lighting and shading models to implement global illumination and environment lighting.

For information about the Light entity and the Environment Probe entity used in environment lighting, see [Light Entities \(p. 461\)](#).

For information about using the Time of Day Editor to simulate the changing lighting effects caused by the sun moving across the sky, see [Creating Time of Day Sky Effects \(p. 864\)](#).

Topics

- [Illuminance and Auto Exposure Key \(p. 1063\)](#)
- [HDR Settings \(p. 1063\)](#)
- [Global Environment Lighting \(p. 1064\)](#)
- [Local Environment Lighting \(p. 1066\)](#)

Illuminance and Auto Exposure Key

Also known as luminous flux density, illuminance is the total amount of visible light falling on a point on a surface from all directions above the surface in a given time. Proper illuminance values ensures the environment lighting in your level closely models real-world values. Besides simply having good ratios between light and dark, accurate illuminance values ensure that tone-mapping, and eye adaptation works optimally.

The following table lists real-world illuminance values, expressed in luminous flux (lux). Lux is the unit of illuminance and luminous emittance, measuring lux per unit area, and equal to one lumen per square meter.

Illuminance Values

Real-world illuminance	Lux Value	Uniformity Ratio	Artistic Interpretation
Full moon	0.25	0.00005	-
Living room	50	0.01	-
Clear sunrise	400	0.08	-
Office	500	0.1	-
TV studio	1,000	0.2	-
Overcast day	15,000	3.0	~ 1.5
Indirect sunlight (in shadow)	20,000	4.0	~ 2.0
Direct sunlight	100,000	20.0	~ 10.0

The **Auto Exposure Key** setting controls the amount of light exposure and determines whether the tone-mapped image appears relatively bright or dark. This setting is calculated automatically from the average scene illuminance, which is why it is important to use standard real-world illuminance levels. For other settings that affect the tone mapping of a scene, see [HDR Settings \(p. 1063\)](#).

Lumberyard's auto-exposure mode works in exposure value (EV) units and can be enabled using the **r_HDREyeAdaptationMode** console variable.

The following settings are used to achieve the desired illuminance in an environment level. See [Setting Daytime Atmospheric Effects \(p. 861\)](#) for more information.

- Sun color
- Sun color multiplier
- Sun intensity
- Sun intensity multiplier

HDR Settings

As discussed in [Illuminance and Auto Exposure Key \(p. 1063\)](#), the auto exposure key setting controls the amount of scene exposure and determines whether the tone-mapped image appears relatively bright or dark. Several other settings also affect the tone mapping of scene. These are known collectively as HDR (high dynamic range) in the Time of Day Editor.

Film curve parameters in the Time of Day Editor correspond to analogous parameters that exist for camera film. A film curve has three distinct regions with different contrast transfer characteristics:

- The lower part of a film curve that is associated with relatively low exposures is designated the toe, and corresponds to the low-density portions of an image. When an image is exposed so that areas fall within the toe region, little or no contrast is transferred to the image.
- The upper part of a film curve that is associated with relatively high exposures is designated the shoulder, and corresponds to the high-density portions of an image. When an image is exposed so that areas fall within the shoulder region, little or no contrast is transferred to the image.
- The middle part of a film curve with the highest level of contrast is produced within a range of exposures falling between the toe and the shoulder, and is designated the midtones region. This portion of the curve is characterized by a relatively straight and steep slope in comparison to the toe and shoulder regions. You should adjust your image so that important areas fall within this region for maximum contrast.

To set HDR settings parameters

1. In Lumberyard Editor, click **Terrain, Time Of Day**.
2. Under **Time of Day Tasks**, click **Toggle Advanced Properties** to view all settings.
3. Under **HDR Settings, HDR**, click and adjust the values of the following settings:

Film curve shoulder scale

Slope at the tip of the HDR curve (modifies bright values).

Film curve midtones scale

Linearity of the middle of the HDR curve (modifies gray values).

Film curve toe scale

Slope at the base of the curve (modifies dark values).

Film curve whitepoint

Value to be mapped as pure white or reference white in the tone-mapped image.

Saturation

Color saturation before tone-mapping.

Color balance

Overall color of the scene.

Auto Exposure Key

Overall brightness of the scene used for eye adaptation. Eye adaptation causes the exposure of a scene to simulate the way human eyes adjust when going from a brightly lit environment to a dark environment and vice versa. Use lower value for dark scenes and higher values for bright scenes. Default value is 0.18.

Auto Exposure Min

Darkest possible exposure used for eye adaptation.

Auto Exposure Max

Brightest possible exposure used for eye adaptation.

Bloom amount

Controls the amount of bloom that comes from glowing or lit objects.

Global Environment Lighting

To implement global lighting for an entire level, you use a global environment probe (also known as a global light probe) and associated generated cubemap.

Environment probes control many aspects of the physically based lighting in Lumberyard, including accurate shadow colors, ambient diffuse values, and specular reflections. They also provide bounce lighting by taking the colors from the surroundings and applying them directly to the diffuse color of materials inside their radius.

When placing environment probes in a level, pay attention to how probes are layered and sorted going from global to local probes.

Every level should have a global environment probe. Global probes provide the entire level with ambient lighting, which is calculated from the probe's location. In addition to a global probe, a level may have one or more local probes. For more information about local probes, see [Local Environment Lighting](#) (p. 1066).

As shown in the following table, the probe has several configurable properties, which you can adjust in the **Rollup Bar**.

EnvironmentProbe Properties

Active

Enables and disables the probe.

BoxSizeX, BoxSizeY, BoxSizeZ

Specifies the XYZ dimensions of the probe's area of effect. Probes are projected as cubes in the level. For a global probe, set values large enough to span the entire level.

Diffuse

Sets the diffuse color of the light. Set to **255, 255, 255**.

DiffuseMultiplier

Makes the light brighter. Set to **1**.

SpecularMultiplier

Multiplies the specular color brightness. Set to **1**.

AffectsThisAreaOnly

Set parameter to **False** to make lights cover other VisAreas.

AttenuationFalloffMax

Controls the falloff amount (0–1) to create smoother transitions or hard edges. A value of 0.8 means that falloff begins at 80% at the boundaries of the box. Set value to 0 for a global probe (no falloff).

IgnoresVisAreas

Controls whether the light should respond to VisAreas. Set value to **True** for a global probe.

SortPriority

Gives control over which probe has more visual interest and therefore a higher priority. Set the value to 0 for a global probe, then increase the value for local probes, where higher values indicate more localized probes.

deferred_cubemap

Specifies the file location of the cubemap texture.

BoxHeight

Adjusts the height of cubemap box.

BoxLength

Adjusts the length of cubemap box.

BoxProject

When enabled, Lumberyard factors in the size of the cubemap box.

BoxWidth

Adjusts the width of cubemap box.

To generate a global cubemap

1. In **Rollup Bar**, under **Objects**, click **Misc, EnvironmentProbe**.
2. Click to place the probe in your level.
3. Under **EnvironmentProbe Params**, leave **cubemap_resolution** at the default 256. This is the optimal resolution for best performance.
4. Select the **preview_cubemap** check box to see the cubemap in your level.

5. Under **EnvironmentProbe Properties**, adjust the following property values to configure the probe to be global:
 - **BoxSizeX**, **BoxSizeY**, and **BoxSizeZ** values: Large enough to span the entire level
 - **Diffuse** color value: 255, 255, 255
 - **DiffuseMultiplier** and **SpecularMultiplier** values: 1
 - **SortPriority**: 0
 - **AttenuationFalloffMax**: 0
 - **IgnoreVisAreas**: True (check box selected)
6. Click **Generate Cubemap**. Lumberyard creates three textures in **textures\cubemaps\your_level**— one for the diffuse map, one for the specular map, and one for the source `.tif` file.
7. To check your cubemap for accuracy, create and then place a smooth, reflective sphere entity near the probe. If its surface looks different from the environment around it, you need to regenerate the cubemap.
8. Click **Generate Cubemap** again. This incorporates object reflections from the originally generated cubemap for added realism.
9. To hide the sphere entity in your level, select its **HiddenInGame** check box, found under **Entity Params** in the **Rollup Bar**.

Local Environment Lighting

Lumberyard uses local environment probes and their generated cubemaps to implement local lighting. The purpose of local cubemaps is to light smaller areas more accurately. This ensures that all areas in your level have accurate lighting effects that may not be covered by the global cubemap. Lumberyard automatically gives a local probe higher priority within its defined radius and superimposes its effects on those of the global probe. For more information about global probes, see [Global Environment Lighting \(p. 1064\)](#).

When placing environment probes in a level, pay attention to how probes are layered and sorted going from global to local probes.

To generate a local cubemap

1. In **Rollup Bar**, under **Objects**, click **Misc, EnvironmentProbe**.
2. Click to place in the probe in your level.
3. Under **EnvironmentProbe Params**, leave the **cubemap_resolution** at 256, the default. This is the optimal resolution for performance.
4. Select the **preview_cubemap** check box to see the cubemap in your level.
5. Under **EnvironmentProbe Params** and under **EnvironmentProbe Properties**, adjust property values for the desired effect. For more information about these properties, see the table in [Global Environment Lighting \(p. 1064\)](#).
6. Click **Generate Cubemap**.

Lumberyard creates three textures in **textures\cubemaps\your_level**— one for the diffuse map, one for the specular map, and one for the source `.tif` file.

7. To check your cubemap for accuracy, create and then place a smooth, reflective sphere entity near the probe. If its surface looks different from the environment around it, you need to regenerate the cubemap.
8. Click **Generate Cubemap** again. This incorporates object reflections from the originally generated cubemap for added realism.
9. To hide the sphere entity in your level, select its **HiddenInGame** check box, found under **Entity Params** in the **Rollup Bar**.

Environment Shadows

Lumberyard supports shadow casting from all light sources and shadow receiving on all deferred and most forward-rendered geometry. Traditional shadow mapping is used for shadow generation. Light sources can be directional, such as from the sun and moon, or from point and area light sources.

As shadow generation is resource-intensive, Lumberyard offers the following features to mitigate this:

- You can control the degree to which Lumberyard caches shadows and stops dynamically updating the most distant cascaded sun shadows.
- You can set point and area light sources to be updated in intervals, such as every second frame.
- You can use the `r_MergeShadowDrawcalls` console variable to merge submaterials during shadow generation, resulting in fewer drawcalls.

Topics

- [Cached Shadows \(p. 1067\)](#)
- [Object Shadows \(p. 1068\)](#)
- [Shadow Proxies \(p. 1069\)](#)

Cached Shadows

Shadow caching is an effective optimization method to reduce the number of shadow drawcalls and to increase the shadow casting and receiving range.

Starting from a defined cascade number, Lumberyard can render subsequent shadow cascades and then keep them in memory. Once the cached cascade is initialized, no more draw calls are needed for updates. This enables long-range distant shadows with almost no performance cost.

Keep in mind that cached shadows are memory intensive, with the default configuration requiring approximately 130 MB of video memory.

In addition, ensure that all shaders are compiled before triggering an update or all objects may not be rendered into the cached shadow maps.

Placement and Update

Cached shadow cascades are centered around the rendering camera by default, and automatically recenter and update once the camera gets close to the cascade border.

You can override this automated placement by using the **Environment:RecomputeStaticShadows** flow graph node, which takes the world space **Min** and **Max** input positions of the bounding area for the first cached cascade. Bounding boxes for subsequent cached cascades are scaled versions of the preceding cascades and are based on the **NextCascadeScale** input multiplier. The **Trigger** input causes an update of all cached shadow cascades.

Note

To keep you informed, a warning message appears in the console each time a cached shadow cascade is updated.

Dynamic Distance Shadows

Cached shadows work well with static objects, but dynamic objects don't get their shadows updated while moving. To overcome this, you can selectively exclude dynamic objects from the cache and render them to the standard cascades. The performance overhead of enabling this feature for a limited number of entities is generally low.

To enable dynamic distance shadows for an object

- Select the **DynamicDistanceShadows** check box for the entity.

Console Variables

When Lumberyard is set to place shadows automatically, the selected resolution combined with the desired world space pixel density, which is derived from the approximate logarithmic split scheme, determines the world space area covered by each shadow cascade. Lowering the resolution lowers the shadowed range for each cascade while still maintaining shadow quality.

When you place shadows manually, the resolution is uniformly stretched across the shadow cascade. Consequently, lower resolutions result in lower shadow quality at the same world space coverage.

Use the following console variables to control cached shadows, including setting the placement and resolution for individually cached shadow cascades.

- **r_ShadowsCache** – Caches all sun shadow cascades above the value. 0 = no cached cascades, 1 = cache first cascade and up, 2 = cache second cascade and up.
- **r_ShadowsCacheResolutions** – The resolution of the cached cascades.
- **r_ShadowsCacheFormat** – Storage format for cached shadow maps: 0 = D32: 32 bit float, 1 = D16: 16 bit integer.
- **e_ShadowsCacheUpdate** – Triggers updates of cached shadow maps: 0 = no update, 1 = one update, 2 = continuous updates.
- **e_ShadowsCacheObjectLod** – The level of detail (LOD) used for rendering objects into the cached shadow maps.
- **e_ShadowsCascadesDebug** – Enables debug view mode. 0 = disable, 1 = enable.
- **e_DynamicDistanceShadows** – Toggles support for having selected objects cast dynamic shadows.

Object Shadows

With object shadows, you can assign custom shadow maps to selected objects, resulting in increased shadow quality due to higher world space shadow texel (texture element) density and reduced depth range.

The drawbacks of using object shadows are increased memory consumption of the additional shadow maps and increased shadow filtering cost.

Object shadows only affect sun shadows. For performance reasons they are not sampled on forward geometry such as particles, hair, and eyes.

Using Flow Graph

You can use the **Environment:PerEntityShadows** flow graph node and assign the target entity to the **Entity** slot. The **Trigger** input applies the settings to Lumberyard.

Because this node is stateless with respect to the entity, you can add multiple **Environment:PerEntityShadows** nodes for the same entity. The last one to be triggered will be in effect.

Use the following node inputs to tweak the shadow appearance:

- **ConstBias/SlopeBias** – Reduces avoid self-shadowing artifacts.
- **Jittering** – Filters kernel size, which directly affects shadow softness.

- **BBoxScale** – Scale factor for the bounding box of the selected entity. Can be useful in case the bounding box is too small or too large.
- **ShadowMapSize** – Size of the custom shadow map, which is automatically rounded to the next power of two.

Using I3DEngine

The following I3DEngine interface functions can be called from anywhere in game code. The function parameters are equivalent to the parameters for the **Environment:PerEntityShadows** Flow Graph node.

- **AddPerObjectShadow** – Adds an object shadow.
- **RemovePerObjectShadow** – Removes an object shadow.
- **GetPerObjectShadow** – Retrieves object shadow settings for a given `RenderNode`. Do not overwrite the `RenderNode` pointer. Instead use `AddPerObjectShadow` `\RemovePerObjectShadow`.
- **ShadowMapSize**: Size of the custom shadow map, which is automatically rounded to the next power of two.

Console Variables

You can use the **e_ShadowsPerObject** console variable with object shadows. With this variable, 0 = Off, 1 = on, and -1 = don't draw object shadows.

Shadow Proxies

Shadow proxies are a method of significantly reducing shadow performance costs by creating dedicated low-polygon count geometry to cast an object's shadow with minimal visual differences. You can also use shadow proxies to minimize shadow artifacts by controlling which geometry can cast shadows.

Keep in mind that if the shadow proxy mesh aligns closely with the `RenderMesh`, you may notice self-shadow artifacts.

No material setup is required in your DCC tool. Instead you use the Material Editor to set up shadow proxies in the material using Material Editor. Place the shadow proxy on its own submaterial, setting **Opacity** to 0 and ensuring that **No Shadow** is not selected (the default).

The shadow proxy must also be linked as a child node of the `RenderMesh`, and it must be on its own material ID.

For the `RenderMesh` material, set as you normally would, except under the **Advanced** properties, select the **No Shadow** option. This instructs Lumberyard to use the shadow proxy instead of the `RenderMesh` to render the shadows.

Voxel-based Global Illumination (SVOGI)

SVOGI, which stands for sparse voxel octree global illumination, also known as voxel GI, is a global illumination solution based on voxel ray tracing. It does not require prebaking or manual setup of bounce lights or light volumes.

Voxel GI provides the following effects:

- Dynamic indirect light bounce from static objects and many dynamic objects.

- Large-scale ambient occlusion (AO) and indirect shadows from static objects such as brushes, terrain, and vegetation.

For every frame, thousands of rays are traced through voxels and shadow maps to gather occlusion and in-directional lighting.

Integration Modes

Voxel GI can be integrated through a number of different modes.

Mode 0

For mode 0, only opacity is voxelized. The bounced light is sampled directly from shadow maps (extended to RSM) and compute shaders are not used.

Mode 0 has some advantages:

- GPU memory usage is small (~16 MB).
- Indirect lighting is completely dynamic; moving sun does not cause any slowdown.
- Dynamic objects can bounce indirect lighting.

Mode 0 also has some disadvantages:

- Indirect lighting can have low quality (more noise), especially for small point lights.
- Only single bounce is possible.
- Only diffuse GI is possible,
- Environment probes are needed for specular highlights.

Modes 1, 2

For modes 1 and 2, albedo, normals, and several layers of radiance are voxelized together with opacity. Direct lighting is also injected into voxelization, where it is propagated within the voxelization and then sampled during the ray-tracing pass.

Modes 1 and 2 have these advantages:

- They support multiple bounces. The light source can be semistatic with multibounce support or be fully dynamic with single bounce support.
- Mode 2 supports traced speculars.
- They provide higher quality, smoother indirect lighting.

Some disadvantages of using Modes 1 and 2 include:

- They use more GPU memory (64MB+).
- Large semistatic multibounce lights cannot be moved freely, but moving sun may work fine.
- Dynamic objects can not affect GI (but can receive it).

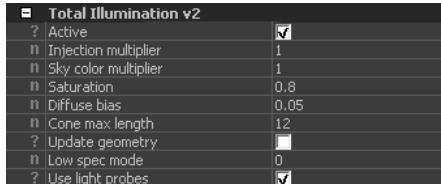
Note

If you get a message that the display driver has stopped responding and has recovered, try this [workaround from Microsoft](#).

Voxel GI Parameters

All the following parameters are global for an entire level. If you need to adjust indirect light intensity locally, use normal ambient lights to modulate or tint it.

In material properties, you use the **Voxel Coverage** parameter to control the transparency of voxels per material and manually fix overoccluded areas.



To enable voxel global illumination

1. In the **Rollup Bar**, on the **Terrain** tab, choose **Environment**.
2. In the **Environment** panel, under **Total Illumination v2**, adjust the following settings as needed.

Parameter	Description
Active	Activates voxel GI for the level.
Injection multiplier	Modulates light injection by controlling the intensity of bounce light.
Sky color multiplier	Controls amount of the sky light. This value may be multiplied with the Time of Day fog color
Saturation	Controls the color saturation of propagated light.
Diffuse bias	Constant ambient value added to GI to prevent completely black areas. If negative, modulates the ambient value with near-range ambient occlusion by preventing constant ambient light in completely occluded indoor areas.
Cone max length	Maximum length of the tracing rays (in meters). Shorter rays work faster.
Update geometry	When enabled, forces single complete revoxelization of the scene. This is needed if terrain, brushes, or vegetation were modified.
Low spec mode	Values greater than 0 simplify shaders and scale down internal render targets. If set to -2 this mode is initialized by the value specified in the <code>sys_spec_Shading.cfg</code> when the level is loaded.
Use light probes	If enabled, environment probes lighting is multiplied with GI. If disabled, diffuse contribution of environment probes is replaced with GI. For integration modes 1–2, this setting enables usage of global environment probe for sky light instead of Time Of Day fog color.

Debugging

Use the following console variable to assist in debugging voxel GI issues.

- `e_svoDebug=6` – Use to visualize the voxels. Ensure all important objects in the scene are voxelized; otherwise they will cast no occlusion and no secondary shadows. Also make sure all unwanted and unnecessary occluders are excluded from voxelization.
- `r_ShowRenderTarget svo_fin` – Use to show the output of voxel GI system.
- `r_profiler 1 | 2` – Use to get GPU profiling information.

Current Limits

The following limitations currently exist for the voxel GI system:

- Large-scale ambient occlusion and indirect shadows are properly cast only by static geometry.
- Voxel GI does not function on some forward-rendering components like particles or water.
- Some artifacts like ghosting, aliasing, light leaking, and noise may be noticeable.
- Procedural vegetation and merged vegetation do not cast occlusion or secondary shadows.
- If a camera is moved to a new location, it may take several seconds until occlusion is working properly.
- Only objects and materials with enabled shadow map casting generate correct bounced light.
- For dynamic objects, indirect light bounce functions only in areas near voxelized static geometry.
- Bounce light may have a noticeable delay of 1 to 2 frames.
- Use of the `r_Supersampling=2` console variable may make voxel GI look strange, but using a lower **LowSpecMode** setting (two times lower) restores the look and speed. In addition, temporal AA (using `r_AntialiasingMode 2/3`) works correctly as well.

Render Cameras and Effects

Topics

- [Fog Systems \(p. 1072\)](#)
- [Rendering Cameras \(p. 1084\)](#)

Fog Systems

Lumberyard supports a standard fog system as well as a voxel-based volumetric fog system. Which one to use for your game comes down to balancing performance over visual quality. Volumetric fog looks superior but comes at a performance cost. The standard fog system is very cheap performance-wise to compute.

You can also add realistic-looking fog above water surfaces, as well as add volumetric fog shadows. For more information, see [Adding Fog Above Water \(p. 856\)](#) and [Adding Volumetric Fog Shadows \(p. 1083\)](#).

Topics

- [Standard Fog \(p. 1072\)](#)
- [Volumetric Fog \(p. 1079\)](#)

Standard Fog

Lumberyard's standard fog system handles sunlight with dynamic shadows and exponential height fog density. However, in dense fog situations the fog's appearance may not be consistent between opaque and transparent materials.

Topics

- [Setting Global \(Time of Day\) Fog \(p. 1073\)](#)
- [Using Fog Volumes \(p. 1075\)](#)
- [Setting Ocean Fog Parameters \(p. 1077\)](#)
- [Setting Fog Environment Parameters \(p. 1078\)](#)
- [Using Console Variables \(p. 1078\)](#)

Setting Global (Time of Day) Fog

Global fog realistically simulates particles distributed uniformly along the ground and falling off exponentially with height above sea level. It also accurately accounts for time of day lighting and for scattered sunlight rays to produce halos around the sun.

Additionally, the effect can cast shadows for both objects and clouds through the fog. For more information, see [Adding Volumetric Fog Shadows \(p. 1083\)](#)

Fog	
Color (bottom)	76,99,128
Color (bottom) multiplier	1
Height (bottom)	0
Density (bottom)	1
Color (top)	76,99,128
Color (top) multiplier	1
Height (top)	400
Density (top)	0.0001
Color height: offset	1
Color (radial)	126,104,60
Color (radial) multiplier	1.5
Radial size	0.2125
Radial lobe	0.25
Final density clamp	1
Global density	1.5
Ramp start	25
Ramp end	1000
Ramp influence	0.7
Shadow darkening	0.2
Shadow darkening sun	0.5
Shadow darkening ambient	1
Shadow range	0.1

To set global fog parameters

1. In Lumberyard Editor, click **Terrain, Time Of Day**.
2. Under **Time of Day Tasks**, click **Toggle Advanced Properties** to view all parameters.
3. Under **Parameters, Fog**, adjust the following parameters as needed:

Parameter	Description
Color (bottom)	Gradient coloring of the global fog. This sets the bottom color of the fog.
Color (bottom) multiplier	A value that is multiplied by the bottom fog color to set the brightness of the top fog color.
Height (bottom)	Specifies a reference height for the vertical fog gradient. This is the height at which the fog color reaches the specified color at the top. For fog density it marks the height at which the vertical density falloff reaches the specified density.
Density (bottom)	Fog density at the bottom. Specifying a density greater than 0 or less than 1 causes the fog to gradually fall off.

Parameter	Description
Color (top)	Specifies the color of the fog component responsible for producing halos around the sun and scattering of sun light.
Color (top) multiplier	Enables gradient coloring of the global fog. This sets the top color of the fog.
Height (top)	Sets the reference height for the vertical fog gradient. For the fog color this marks the height at which it reaches the specified color at the top. For the fog density it marks the height at which the vertical density falloff reaches the specified density.
Density (top)	<p>Density of the fog at the top. Note that it is possible to set the top density to a higher value than the bottom density. This effectively reverses the vertical falloff and produces thick fog in the sky and clear views at the bottom. Also note that both top and bottom density can be equal.</p> <p>It is also important to understand that the volumetric fog computations treat a level as a continuous unbound volume. That means specifying a density greater than 0 at the specified top height doesn't mean that fog suddenly stops there. Instead it continues to fall off gradually. The same is true for the bottom boundary or density values less than 1.</p>
Color height offset	Shifts the color of the vertical fog gradient towards the top or bottom.
Color (radial) and multiplier	Fog color component that is responsible for producing halos around the sun and for scattering of sun light.
Radial size	Size of the radial fog component.
Radial lobe	Amount the radial fog component is affected by distance. Small values affect the horizon only while bigger values make it appear all over the scene.
Final density clamp	Maximum fog density that is allowed for final blending with the scene. This enables the sky, horizon, and other bright distant objects to punch through the fog even if it is dense. However, take care not to set this value too low or it compromises depth perception and results in implausible visuals and apparent artifacts, especially when moving the camera.
Global density	Density of the global volumetric fog. Higher values produce denser fog.

Parameter	Description
Ramp start	Distance from the camera at which the fog starts to be rendered (at 0 density).
Ramp end	Distance from the camera at which the fog starts to be rendered (at 0 density).
Ramp influence	Amount the ramp values affect the rendering of the fog.
Shadow darkening	Amount the fog color (using the settings above) is darkened per pixel based on the volumetric shadow value computed per pixel. The factor is applied after a darkened fog color has been calculated using the sun and ambient darkening factor. See the next two parameters.
Shadow darkening sun	Amount that the sun influences the radial fog color.
Shadow darkening ambient	Amount that the environment is influencing the ambient fog color height gradient.
Shadow range	Distance out that the volumetric shadows are rendered until 10% (0.1) of the level's far clipping plane distance is reached. Smaller values result in more accurate results but shadows won't cast as far.

Using Fog Volumes

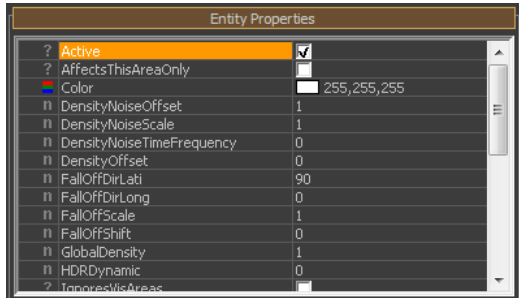
Fog volumes are localized 3D areas that define an area where non-volumetric fog is present. Fog volumes do not change in dynamic, nonuniform ways like smoke does. When alpha-transparent objects are behind fog volumes, each pixel is fogged. However, this is not the case when objects are inside fog volumes.

Unlike global (Time of Day) fog that has an upward falloff direction, fog volumes can have an arbitrary falloff direction. Interesting fog shapes and effects can be achieved, including fog patches that vary in size, color, shape, density, and spacing over time, as well as being influenced by wind.

Observe these best practices when creating fog volumes

- Do not overlap fog volumes.
- Make sure indoor fog volumes don't cover more than one sector or they may be culled when the main sector becomes invisible.
- To avoid inaccurate rendering, don't apply nonuniform scaling to fog volumes.
- When using shadow maps inside fog volumes, make sure the environment **VolFogShadows** parameter is disabled.

You can control fog volume appearance using the **FogVolume** entity properties in Rollup Bar.



To add a fog volume to your level

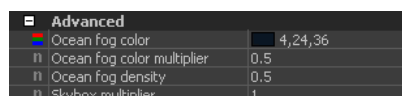
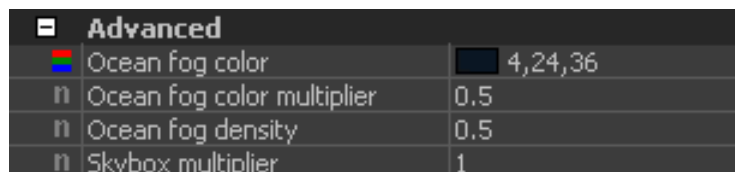
1. In **Rollup Bar**, under **Objects**, click **Entity**.
2. Under **Browser**, expand **Render** and double-click **FogVolume**.
3. Click to place the volume at the desired location in your level.
4. Under **Entity Properties**, adjust the following parameters as needed:

Parameter	Description
Active	Enables fog volumes when selected.
AffectsThisAreaOnly	Disable this setting to have the FogVolume entity effect occurs in multiple VisAreas and ClipVolumes.
Color	Specifies the RGB diffuse color of the fog volume.
DensityNoiseOffset	Offsets the noise value for the fog density.
DensityNoiseScale	Scales the noise value for the fog density.
DensityNoiseTimeFrequency	Controls the time frequency of the noise for the fog density. High frequencies produce fast-changing fog.
DensityOffset	Used in conjunction with the GlobalDensity parameter to offset the density.
FallOffDirLati	Controls the latitude falloff direction of the fog. A value of 90° means the falloff direction is upwards.
FallOffDirLong	Controls the longitude falloff direction of the fog, where 0° represents east. Rotation is counterclockwise.
FallOffScale	Scales the density distribution along the falloff direction. Higher values make the fog fall off more rapidly and generate thicker fog layers along the negative falloff direction.
FallOffShift	Controls how much to shift the fog density distribution along the falloff direction in world units (m). Positive values move thicker fog layers along the falloff direction into the fog volume.

Parameter	Description
GlobalDensity	Controls the density of the fog. The higher the value the more dense the fog.
HDRDynamic	Specifies how much brighter than the default white (RGB 255,255,255) the fog is.
IgnoreVisAreas	Controls whether the FogVolume entity should respond to VisAreas and ClipVolumes.
NearCutoff	Stops rendering the object depending on camera distance to object.
RampEnd	Specifies the end distance of fog density ramp in world units (m).
RampInfluence	Controls the influence of fog density ramp.
RampStart	Specifies the start distance of fog density ramp in world units (m).
SoftEdges	Factor used to soften the edges of the fog volume when viewed from outside. A value of 0.0 produces hard edges. Increasing this value up to 1.0 gradually softens the edges. This property currently has no effect on box type fog volumes as specified in the VolumeType parameter.
UseGlobalFogColor	If selected, ignores the Color parameter and uses the global (Time Of Day) fog color instead.
VolumeType	Produces a box volume for values above 1.0 or a spherical volume for lower values.
WindInfluence	Fog is influenced by the wind.
DensityNoiseFrequency X, Y, Z	Controls the spatial frequency of the noise for the fog density. High frequencies produce highly detailed fog.

Setting Ocean Fog Parameters

You can use several settings to customize the look of fog over the ocean.

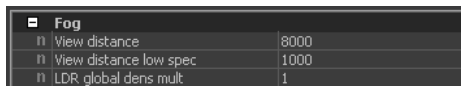


To set ocean fog parameters

1. In Lumberyard Editor, click **Terrain, Time Of Day**.
2. Under **Time of Day Tasks**, click **Toggle Advanced Properties** to access the fog parameters.
3. Under **Parameters**, in the **Advanced** panel, click to adjust ocean fog parameter values for the desired effect, as listed below:
 - **Ocean fog color** – Sets the RGB ocean fog color for a specific time of day.
 - **Ocean fog color multiplier** – Sets the brightness of the ocean fog, which is multiplied by the ocean fog color.
 - **Ocean fog density** – Sets the density of the ocean fog.

Setting Fog Environment Parameters

You can set fog environment properties with just a few simple steps.



To set fog environment properties

1. In Rollup Bar, on the **Terrain** tab, click **Environment**.
2. Under **Fog**, adjust the following values as needed:
 - a. **View distance** - distance at which the fog fades away.
 - b. **View distance low spec** - distance at which the fog fades away using the low spec setting.
 - c. **LDR global dens mult** - sets the low dynamic range global fog density multiplier.

Using Console Variables

The following console variables can be used to control fog:

Parameter	Description
e_Fog	Toggles fog on and off.
e_FogVolumes	Enables local height/distance based fog volumes.
e_FogVolumesTiledInjection	Enables tiled fog volume density injection.
r_FogDepthTest	Enables per-pixel culling for deferred fog pass. Fog computations for all pixels closer than a given depth value will be skipped. 0 = culling disabled. > 0 = fixed linear world space culling depth. < 0 = optimal culling depth will be computed automatically based on camera direction and fog settings.
r_FogShadows	Enabled deferred volumetric fog shadows. 0 - no shadows. 1 = standard resolution. 2 = reduced resolution.
r_FogShadowsMode	Ray-casting mode for shadowed fog. 0 = brute force shadow map sampling. 1 = optimized shadow map sampling.

Parameter	Description
r_FogShadowsWater	Enables volumetric fog shadows over water volumes

Volumetric Fog

Volumetric fog uses volume textures as a view-frustum-shaped [voxel](#) buffer to store incoming light and its properties. Volumetric fog supports regular light and sunlight with dynamic shadows, environment probes, ambient light, as well as variations in fog density. It also supports the application of volumetric fog with respect to opaque and transparent materials.





The **Light** entity has three parameters relating to volumetric fog. For more information, see the **AffectsVolumetricFogOnly**, **FogRadialLobe**, and **VolumetricFog** parameters for the [Light Entity](#) (p. 461) in the [Object and Entity System](#) (p. 416).

In addition, you can also use the **Particle Editor** to place a particle emitter in your level to add fog density to an area. For more information, see the **Volume Fog** and **Volume Thickness** parameters for the **Advanced Attribute** in the [Particle Attributes and Parameters Reference](#) (p. 956).

To add localized nonvolumetric regions of fog, see [Using Fog Volumes](#) (p. 1075).

Topics

- [Guidelines and Best Practices for Volumetric Fog](#) (p. 1080)
- [Setting Global \(Time of Day\) Volumetric Fog](#) (p. 1081)
- [Setting Volumetric Fog Environment Parameters](#) (p. 1083)
- [Adding Volumetric Fog Shadows](#) (p. 1083)
- [Using Console Variables](#) (p. 1084)

Guidelines and Best Practices for Volumetric Fog

Observe the following guidelines and best practices for volumetric fog.

- Make sure that the `r_DeferredShadingTiled` console variable is set to greater than 0. A value of 1 to 2 is recommended. This is required to use volumetric fog.
- To avoid performance problems, use the default values for the **Ramp Start** and **Ramp End** parameters located in the **Time of Day** editor.
- Note that the **Light** entity's **PlanarLight** parameter with the **AmbientLight** parameter enabled is supported. However the **PlanarLight** parameter with the **AmbientLight** parameter disabled is not supported.

- Using large values for the **Range** parameter in the **Time of Day Editor** may cause fog flicker and light leaking behind walls unless you adjust the `r_VolumetricFogTexDepth` console value accordingly
- The default values are `r_VolumetricFogTexDepth=32` for **Range=64**. If you want to use larger ranges such as **Range=256** and with same visual quality, you need to set `r_VolumetricFogTexDepth=64`. When **Range=1024** is used, set `r_VolumetricFogTexDepth=128`.

Setting Global (Time of Day) Volumetric Fog

Global volumetric fog realistically simulates particles distributed uniformly along the ground and falling off exponentially with height above sea level. It also accurately accounts for time of day lighting and for scattered sunlight rays to produce halos around the sun.

Volumetric fog	
Height (bottom)	0
Density (bottom)	1
Height (top)	4000
Density (top)	0.0001
Global density	0.1
Ramp start	0
Ramp end	0
Color (atmosphere)	255,255,255
Anisotropy (atmosphere)	0.6
Color (sun radial)	255,255,255
Anisotropy (sun radial)	0.95
Radial blend factor	1
Radial blend mode	0
Color (entities)	255,255,255
Anisotropy (entities)	0.6
Range	64
In-scattering	1
Extinction	0.3
Analytical fog visibility	0.5
Final density clamp	1

You can use the **Anisotropy** parameters listed after the following procedure to control how much sunlight is scattered through fog and in which direction. Setting the **Anisotropy (atmosphere)** parameter close to 0 achieves a uniform look across the entire sky, while setting the **Anisotropy (sun radial)** parameter close to 1 produces a bloom effect around the sun.

The **Radial blend** parameters blend the **Anisotropy** parameters to create various effects. For example, setting **Radial blend mode** = 1 and **Radial blend factor** = 1 produces sun radial scattering only.

You set global volumetric fog parameters in the Time of Day Editor, which you open from Lumberyard Editor by clicking **Terrain, Time Of Day**.

To set global volumetric fog parameters

1. In Lumberyard Editor, click **Terrain, Time Of Day**.
2. Under **Time of Day Tasks**, click **Toggle Advanced Properties** to view all parameters.
3. Under **Parameters, Volumetric fog**, adjust the following parameters as needed:

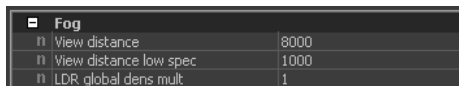
Parameter	Description
Height (bottom)	Specifies a reference height for the vertical fog gradient. This is the height at which the fog color reaches the specified color at the top. For fog density it marks the height at which the vertical density falloff reaches the specified density.

Parameter	Description
Density (bottom)	Fog density at the bottom. Specifying a density greater than 0 or less than 1 causes the fog to gradually fall off.
Height (top)	Sets the reference height for the vertical fog gradient. For the fog color this marks the height at which it reaches the specified color at the top. For the fog density it marks the height at which the vertical density falloff reaches the specified density.
Density (top)	<p>Density of the fog at the top. Note that it is possible to set the top density to a higher value than the bottom density. This effectively reverses the vertical falloff and produces thick fog in the sky and clear views at the bottom. Also note that both top and bottom density can be equal.</p> <p>Volumetric fog computations treat a level as a continuous unbound volume. That means specifying a density greater than 0 at the specified top height doesn't mean that fog suddenly stops there. Instead it continues to fall off gradually. The same is true for the bottom boundary or density values less than 1.</p>
Global density	Density of the global volumetric fog. Higher values produce denser fog
Ramp start	Distance from the camera at which the fog starts to be rendered (at 0 density).
Ramp end	Distance from the camera at which the fog starts to be rendered (at 0 density).
Color (atmosphere)	Specifies the fog albedo color for sun atmosphere scattering.
Anisotropy (atmosphere)	Adjusts the anisotropy for sun atmosphere scattering. When 0 = isotropic, then 1 = perfect forward, and -1 = perfect backward scattering.
Color (sun radial)	Specifies the fog albedo color for sun radial scattering.
Anisotropy (sun radial)	Adjusts the anisotropy for sun radial scattering. When 0 = isotropic, then 1 = perfect forward, and -1 = perfect backward in-scattering.
Radial blend factor	Adjusts the blend factor of blending sun atmosphere and sun radial scattering.
Radial blend mode	Adjusts the blend mode factor of blending sun atmosphere and sun radial scattering.
Color (entities)	Specifies the global fog albedo color for scatterings of all types of light except the sun.

Parameter	Description
Anisotropy (entities)	Adjusts the anisotropy of entities (such as FogVolume) except the global fog. When 0 = isotropic, then 1 = perfect forward, and -1 = perfect backward in-scattering.
Range	Adjusts the maximum distance of volumetric fog. The default setting is 64.
In-scattering	Adjusts the factor of in-scattering of all participating media.
Extinction	Adjusts the factor of extinction of all participating media.
Analytical fog visibility	Adjusts the visibility of analytical volumetric fog. Where 0 = no analytical volumetric fog, 1 = visible analytical volumetric fog.
Final density clamp	Maximum fog density that is allowed for final blending with the scene. This enables the sky, horizon, and other bright distant objects to punch through the fog even if it is dense. However, take care not to set this value too low or it compromises depth perception and results in implausible visuals and apparent artifacts, especially when moving the camera.

Setting Volumetric Fog Environment Parameters

You can set fog environment properties with just a few simple steps.

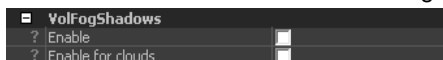


To set volumetric fog environment properties

1. In the **Rollup Bar**, on the **Terrain** tab, click **Environment**.
2. Under **Fog**, adjust the following values as needed:
 - **View distance** – Distance at which the fog fades away.
 - **View distance low spec** – Distance at which the fog fades away using the low spec setting.
 - **LDR global dens mult** – Sets the low dynamic range global fog density multiplier.

Adding Volumetric Fog Shadows

You can add volumetric shadows to fog with just a few simple steps.



To add volumetric fog shadows

1. In the **Rollup Bar**, on the **Terrain** tab, click **Environment**.
2. Under **VolFogShadows**, do the following:

- Click **Enable** to enable volumetric shadows from global fog.
- Click **EnableForClouds** to enable volumetric shadows from clouds.

Using Console Variables

The following console variables can be used to control volumetric fog:

Parameter	Description
e_VolumetricFog	Toggles volumetric fog on and off.
r_VolumetricFogDownscaledSunShadow	Enables replacing sun shadow maps with downscaled shadow maps or static shadow map if possible. This reduces volumetric fog flicker for sun shadows.
r_VolumetricFogDownscaledSunShadowRatio	Sets the downscale ratio for sun shadow maps.
r_VolumetricFogMinimumLightBulbSize	Adjusts the minimum size threshold for light attenuation bulb size for volumetric fog. Small bulb sizes may cause light flicker.
r_VolumetricFogReprojectionBlendFactor	Adjusts the blending factor of the temporal reprojection filter. Higher values cause less flicker, but more ghosting.
r_VolumetricFogReprojectionMode	Sets the mode of ghost reduction for the temporal reprojection filter.
r_VolumetricFogSample	Adjusts the number of sample points.
r_VolumetricFogShadow	Adjusts the shadow sample count per sample point.
r_VolumetricFogTexDepth	Adjusts the internal volume texture depth.
r_VolumetricFogTexScale	Adjusts the internal volume texture width and height. Screen resolution divided by this factor is applied to both the width and height.

Rendering Cameras

You can use rendering cameras to define custom views within your level. You can trigger them using the Track View Editor or the **Image:EffectDepthOfField** flow graph node. Rendering cameras are used frequently for animated sequences.

FOV	60.0001
NearZ	0.25
FarZ	1024
Shake Parameters	
Amplitude A	1, 1, 1
Amplitude A Mult.	0
Frequency A	1, 1, 1
Frequency A Mult.	0
Noise A Amp. Mult.	0
Noise A Freq. Mult.	0
Time Offset A	0
Amplitude B	1, 1, 1
Amplitude B Mult.	0
Frequency B	1, 1, 1
Frequency B Mult.	0
Noise B Amp. Mult.	0
Noise B Freq. Mult.	0
Time Offset B	0
Random Seed	0

To add a render camera to your level

1. In the **Rollup Bar**, on the **Objects** tab, click **Misc, Camera**. Drag the camera object into your level, and then click to position it.
2. To add a targeted camera, hold down the **Shift** key when clicking to position the camera. Then drag the camera to its target. Release the mouse button to position both the camera and its target in the level.
3. Adjust the values of the following parameters as needed.

Parameter	Description
FOV	The vertical field of view of the camera
NearZ	The cut off point closest to the camera
FarZ	The max cut off point of the camera
Shake Parameters	
Amplitude A	Strength of the effect on each axis
Amplitude A Multiplier	Multiplier for the amplitude.
Frequency A	How often the effect plays on each axis
Frequency A Multiplier	Multiplier for the frequency
Noise A Amplitude Multiplier	Adds some noise to the amplitude value
Noise A Frequency Multiplier	Adds some noise to the frequency value
Time Offset A	A time offset
Amplitude B	Strength of the effect on each axis.
Amplitude B Multiplier	Multiplier for the amplitude
Frequency B	How often the effect plays on each axis
Frequency B Multiplier	Multiplier for the frequency
Noise B Amplitude Multiplier	Adds some noise to the amplitude value
Noise B Frequency Multiplier	Adds some noise to the frequency value
Time Offset B	A time offset

Parameter	Description
Random Seed	Apply some random variation to the noise

Depth of Field

Lumberyard uses an efficient gather-based depth of field (DOF) implementation. Depth of field is used to enhance the realism of a scene by simulating the way a real-world camera works. Use a broad depth of field to focus on all or nearly all of a scene. Use a narrow depth of field to focus on objects that are within a certain distance from the camera.





You can enable depth of field by using the `r_depthOfFieldMode` console variable. To control depth of field use the Track View editor or the **Image:EffectDepthOfField** flow graph node.

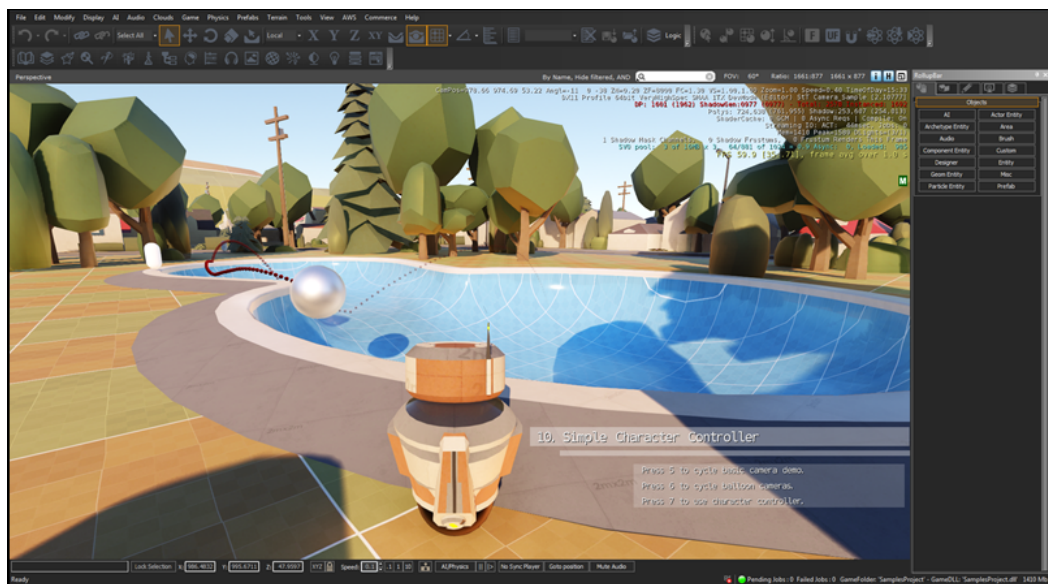
Motion Blur

Lumberyard uses a sample-weighted motion blur implementation whose settings mirror real-world camera shutter speed settings.

Sample Projects and Levels

Lumberyard offers a variety of sample projects, levels, and assets for you, which are located in the `\dev` directory at the root of the Lumberyard installation (`\lumberyard\dev`):

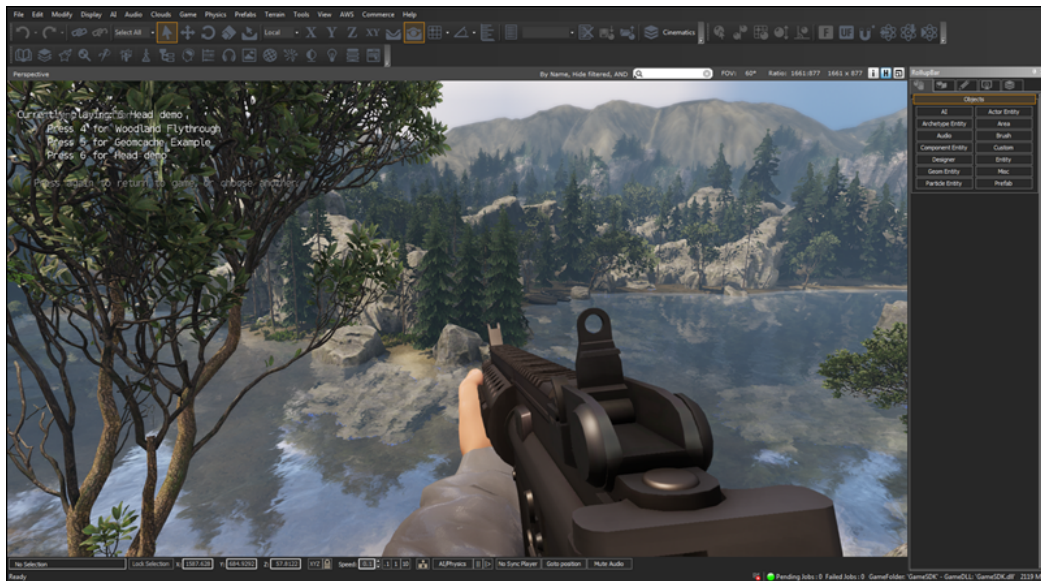
- **Samples Project** – Includes several gameplay levels and content that you need to follow the Lumberyard tutorials.



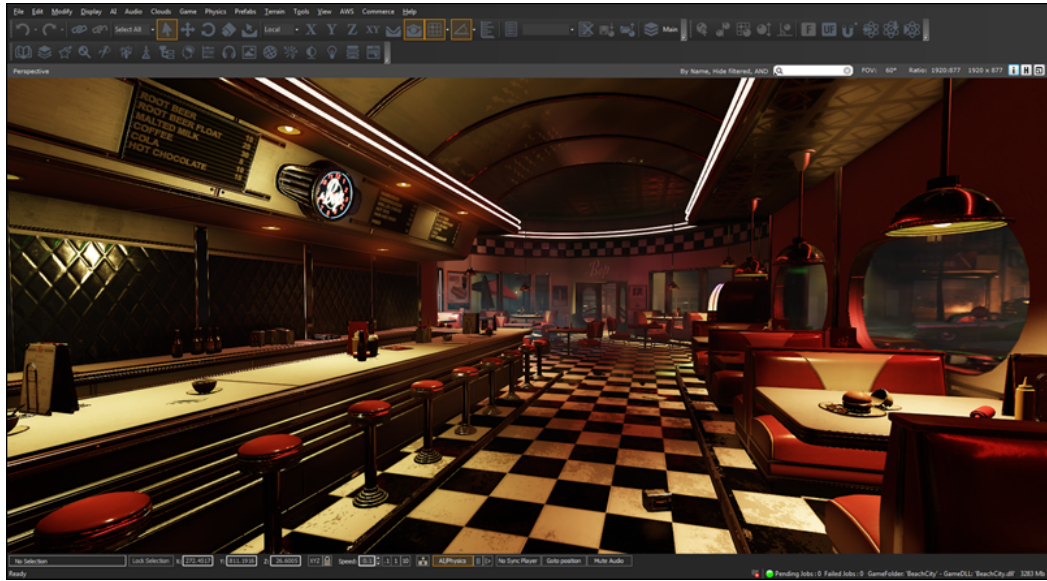
- **Multiplayer Project** – This game enables you to evaluate Amazon GameLift and test Lumberyard's multiplayer capabilities.



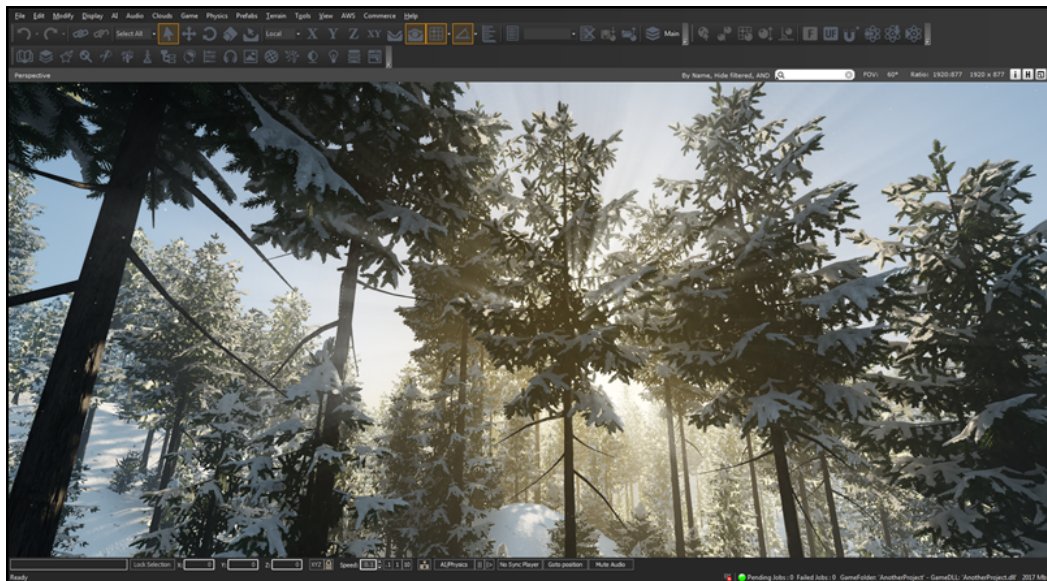
- **Legacy Project (GameSDK)** – Enables you to use GameSDK functionality. GameSDK is available as a separate download.



- **Beach City Night Asset Collection** – Collection of free assets that you can use to try Lumberyard or make your own games. The Beach City Night asset collection is available as a separate download.



- **Woodland Asset Collection** – Free assets for you to use to create your levels. The Woodland asset collection is available as a separate download.



- **FeatureTests** – Collection of levels designed for demonstrating the behavior of a single core feature of Lumberyard.

Topics

- [Samples Project \(p. 1091\)](#)
- [Multiplayer Sample Project \(p. 1103\)](#)
- [Legacy Sample Project \(GameSDK\) \(p. 1105\)](#)
- [Beach City Sample Project \(p. 1106\)](#)
- [Woodland Asset Package \(p. 1107\)](#)
- [FeatureTests Project \(p. 1108\)](#)

Samples Project

The samples project includes a collection of sample levels and code that demonstrates how to use various Lumberyard features. The levels are located in the `\dev\SamplesProject\Levels` directory at the root of the Lumberyard installation (`\lumberyard\dev`).

The **Getting Started**, **Animation**, **Camera**, **Movers**, and **Triggers** projects show you how to use the **Flow Graph** editor to create a variety of scripted events. The examples in these projects show a basic setup and then progressively add more complexity or variation for each additional example. Every script has been annotated to explain what the script does and how each associated flow graph node is used.

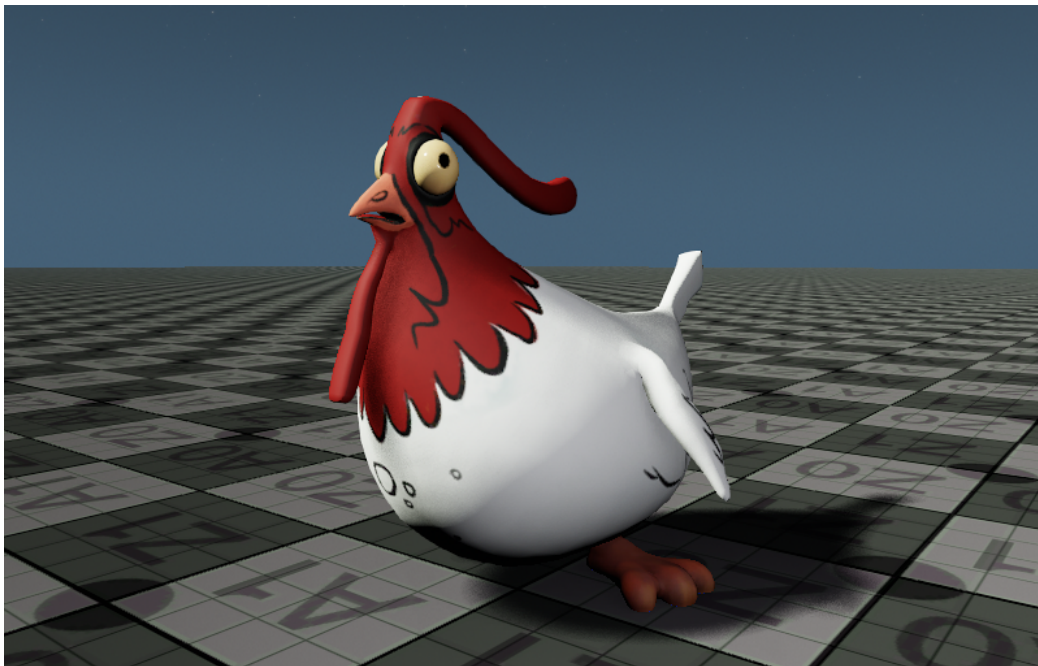
Getting Started Project

The `\GettingStartedFiles` directory contains a complete level that includes terrain, lighting, cameras, objects, materials, flow graph scripting, and code used to complete each step of the *Lumberyard Basics* tutorial. You can use these levels to skip over parts of the tutorial as needed and also to see the completed experience as it should behave based on the walkthrough. For more information, see the [Lumberyard Tutorials](#).

Samples Projects

The `\Samples` subfolder includes a collection of sample projects that demonstrate various Lumberyard features.

Animation Basic Sample



This sample demonstrates how to set up and trigger basic character animations in the **Flow Graph** editor. To play the game, do the following:

- Press **Ctrl+G** to start the level.

- Click the mouse to trigger an animation.
- Press **Esc** to exit the level.

To see the flow graph, choose **View, Open View Pane, Flow Graph**. In the **Graphs** pane, expand **Level, Entities, chicken_flow**, and click **AnimObject**.

Camera Sample



This sample demonstrates how to use a camera entity with flow graph scripts to create different camera events and types for a gameplay experience. Each example is fully annotated within the flow graph scripts of the level file.

Examples for this project include the following:

- **Example 1** – Shows a camera pointed directly at the capsule for a first-person point of view (POV).
- **Example 2** – Shows a camera pointed directly at the capsule but offset to create a third-person POV.
- **Example 3** – Shows a look target pointed at the capsule with a camera parented to the look target, and creating a simple rig example.
- **Example 4** – Shows a top-down POV example.
- **Example 5** – Shows a side scroller POV example.
- **Example 6** – Shows a camera pointed at another entity that is tracking a target. The field of view (FOV) changes based on the distance between the camera and its target.
- **Example 7** – Shows a floating tracking camera. FOV changes are based on the distance between the camera and its target.
- **Example 8** – Shows a simple chase cam.

- **Example 9** – Shows a top-down camera with player controls.
- **Example 10** – Shows a complex example of using flowgraph to create a custom character controller and third-person camera rig.

To play the examples, do the following:

- Press **5** to cycle through camera examples 1 through 10.
- Press **6** to cycle through the balloon camera.
- Press **7** to take control of the robot sphere controller.
- Use the following robot keyboard keys and mouse controls for examples 9 and 10:
 - Press the **W**, **A**, **S**, and **D** keys to move forward, left, backward, and right, respectively.
 - Move the mouse to look around.
 - Press **spacebar** to jump.

Drive the game robot to each numbered display in this sample to see an annotated explanation.

Don't Die

This simplified version of the Don't Die sample illustrates AWS Cloud Canvas functionality and flow graph integration. For more information, see [Don't Die Sample Project](#).

Use the keyboard keys and mouse controls as follows:

- Press the **W**, **A**, **S**, and **D** keys to move forward, left, backward, and right, respectively.
- If shield is active, use the left mouse button to deflect and destroy incoming debris.
- Press the **Y** key to restart the game.

Upon death, a high-score screen displays the top 10 survival times and screenshots.

When the game starts, text at the bottom of the screen shows a message with your Windows login name. Don't Die associates the login name with your Cognito ID and displays it next to your high score. The **Message of the Day** ticker scrolls below the **Don't die username** message.

Movers Sample



This sample demonstrates how to move objects using scripted events to define the motion within a level. Each example is fully annotated within the flow graph scripts of the level file. It contains various examples of moving objects in a scene using the **MoveEntityTo**, **RotateEntity**, and **RotateEntityTo** flow graph nodes.

Examples for this project include the following:

- **Example 1** – Moves the entity to a tag point.
- **Example 2** – Moves the entity to a tag point and loops the operation.
- **Example 3** – Move the entity to a tagpoint, adds additional rotation, and loops the operation.
- **Example 4** – Rotates the entity indefinitely.
- **Example 5** – Accelerates and rotates the entity 180 degrees.
- **Example 6** – Parents the entity to another and sets the parent to rotate indefinitely.
- **Example 7** – Accelerates and rotates up to a maximum speed.
- **Example 8** – Accelerates a rotation and then decelerates.
- **Example 9** – Accelerates a rotation, decelerates, and loops.
- **Example 10** – Links four separate entities to the same parent. Both the parent and its children rotate at different angles and rates.
- **Example 11** – Uses keyboard keys **I**, **K**, **J**, and **L** to move a box around in the viewport. Shows a second entity moving toward the first one.

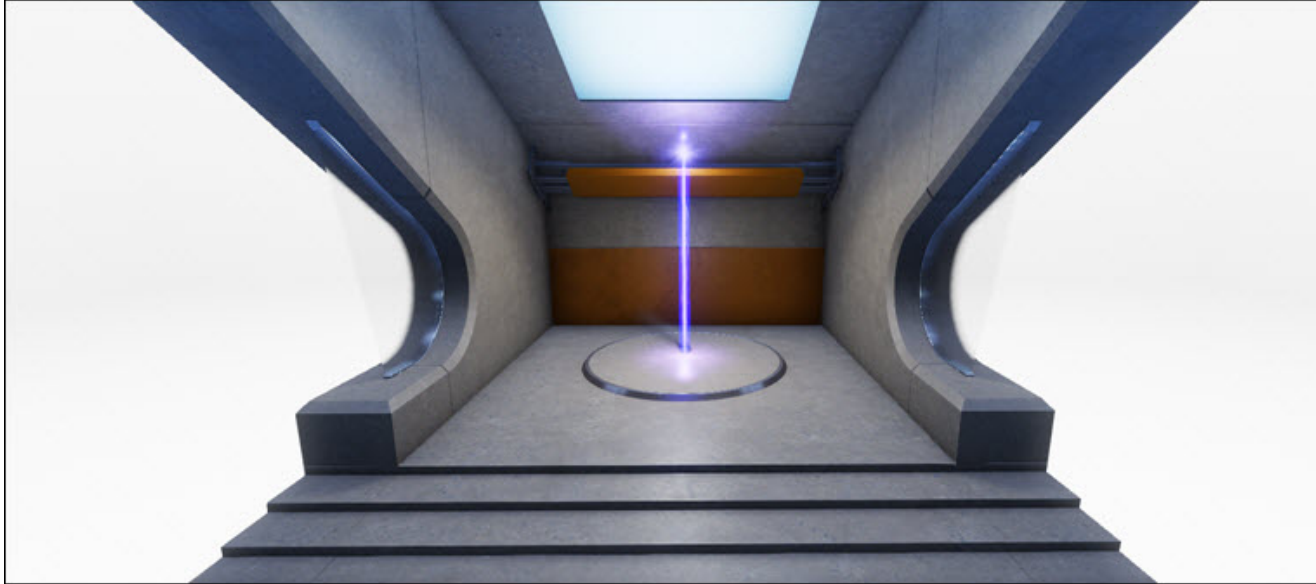
Use the following robot keyboard keys and mouse controls:

- Press **W**, **A**, **S**, and **D** keys to move forward, left, backward, and right, respectively.

- Move the mouse to look around.
- Press the **spacebar** to jump.

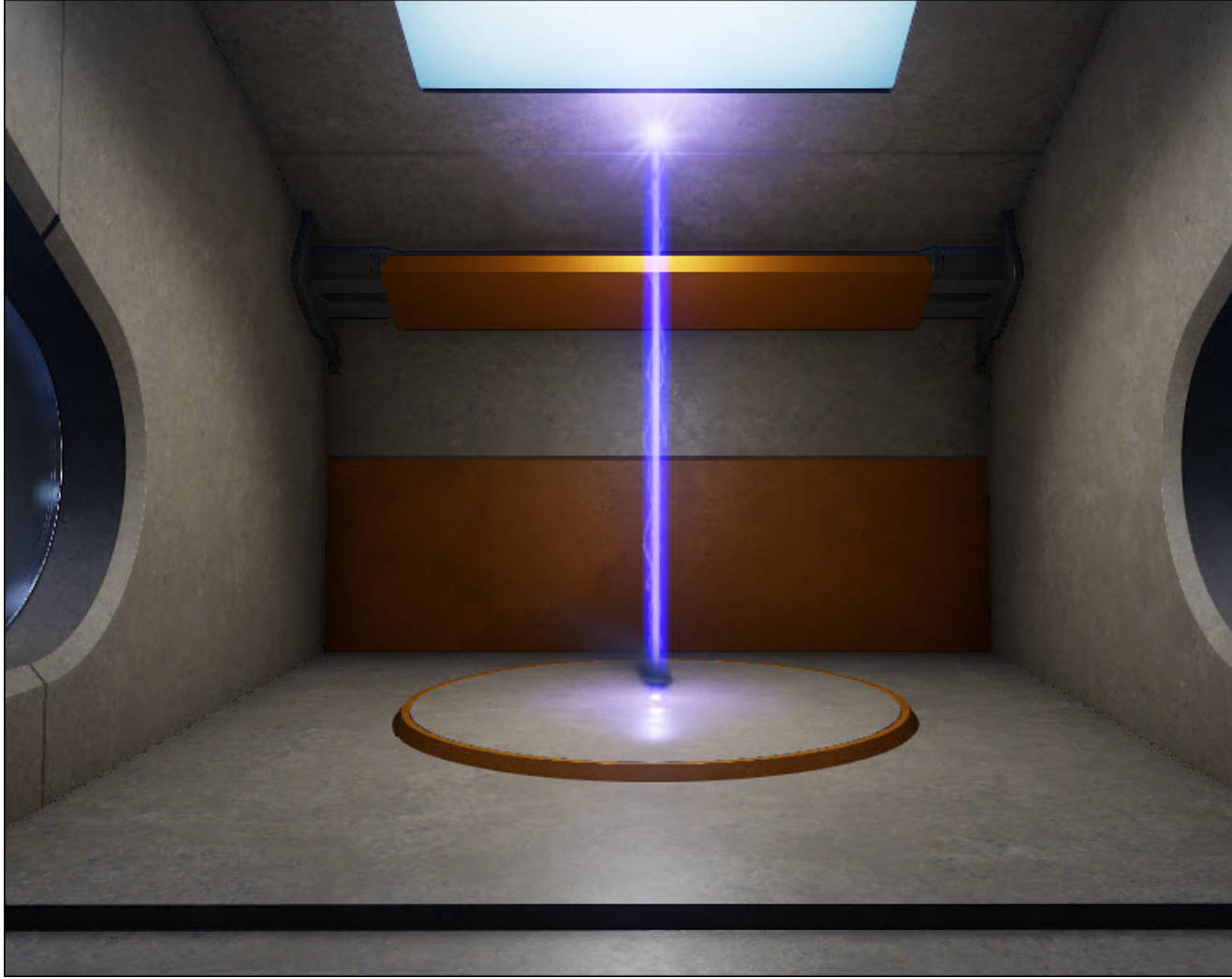
Drive the game robot to each numbered display in this sample to activate the trigger and see an annotated explanation.

Particles Sample

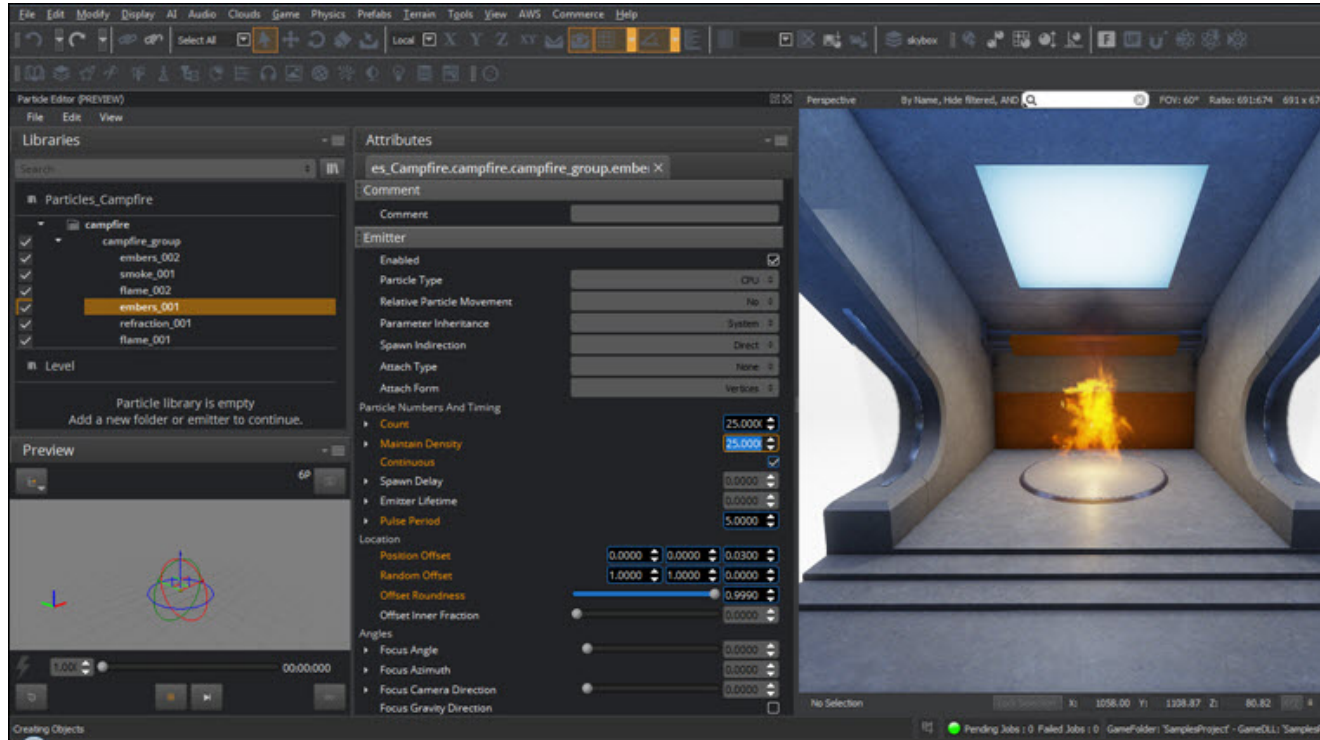


This sample showcases particles from the various Lumberyard systems and demonstrates how to create high-quality effects using simple and advanced features in the **Particle Editor**. Currently the sample level includes a fire effect and a laser effect. More particles will be added in the future.





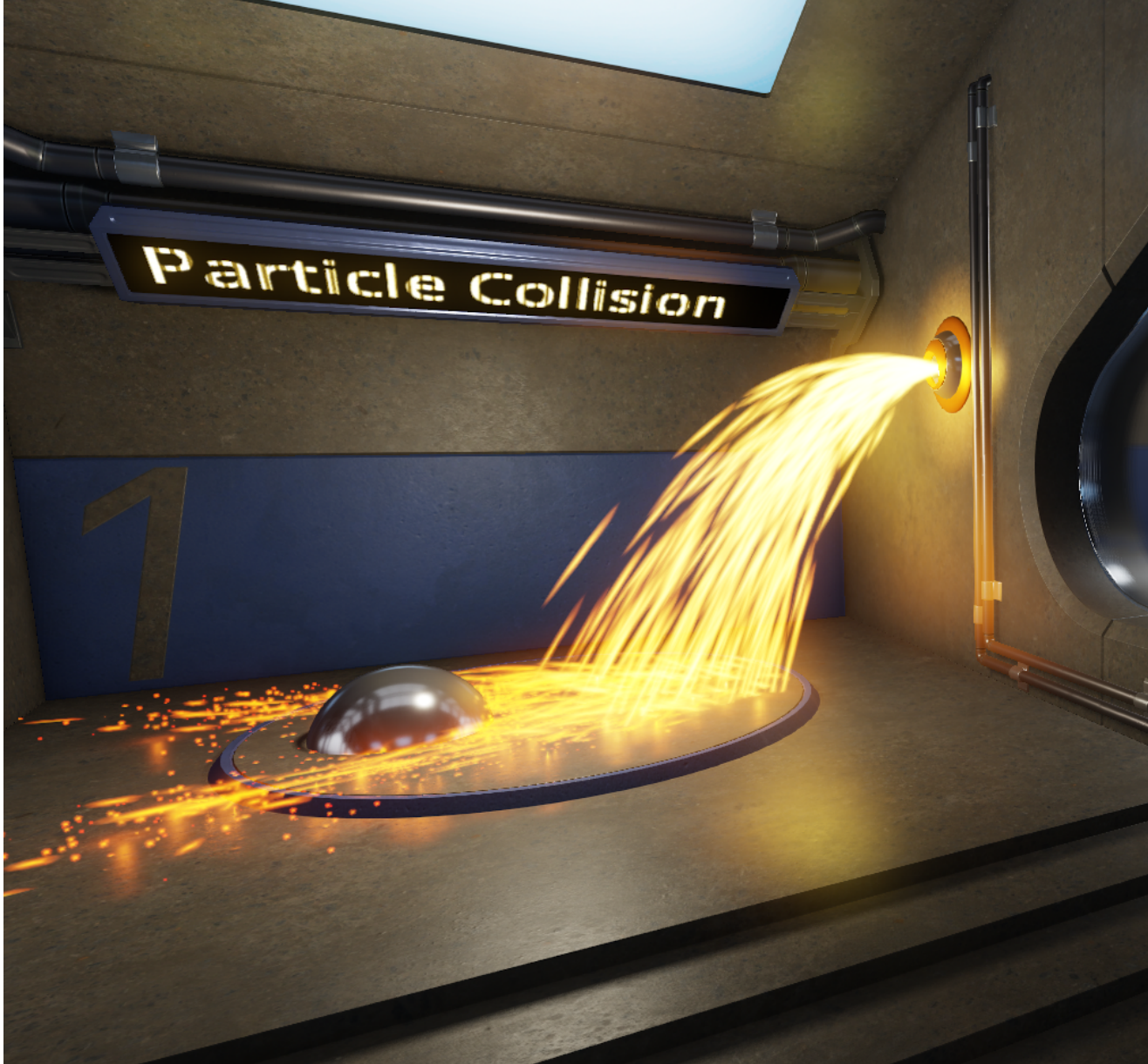
You can view particle effects and interact with individual particles in Lumberyard Editor by clicking **View, Open View Pane, Particle Editor**.



Once the particles sample level is open, in the **Perspective** viewport in Lumberyard Editor, use the following keyboard keys and mouse controls:

- Press **Ctrl+G** to start the level.
- Press the **W**, **A**, **S**, and **D** keys to move forward, left, backward, and right, respectively.
- Move the mouse to look around.
- Press **Esc** to exit game mode.

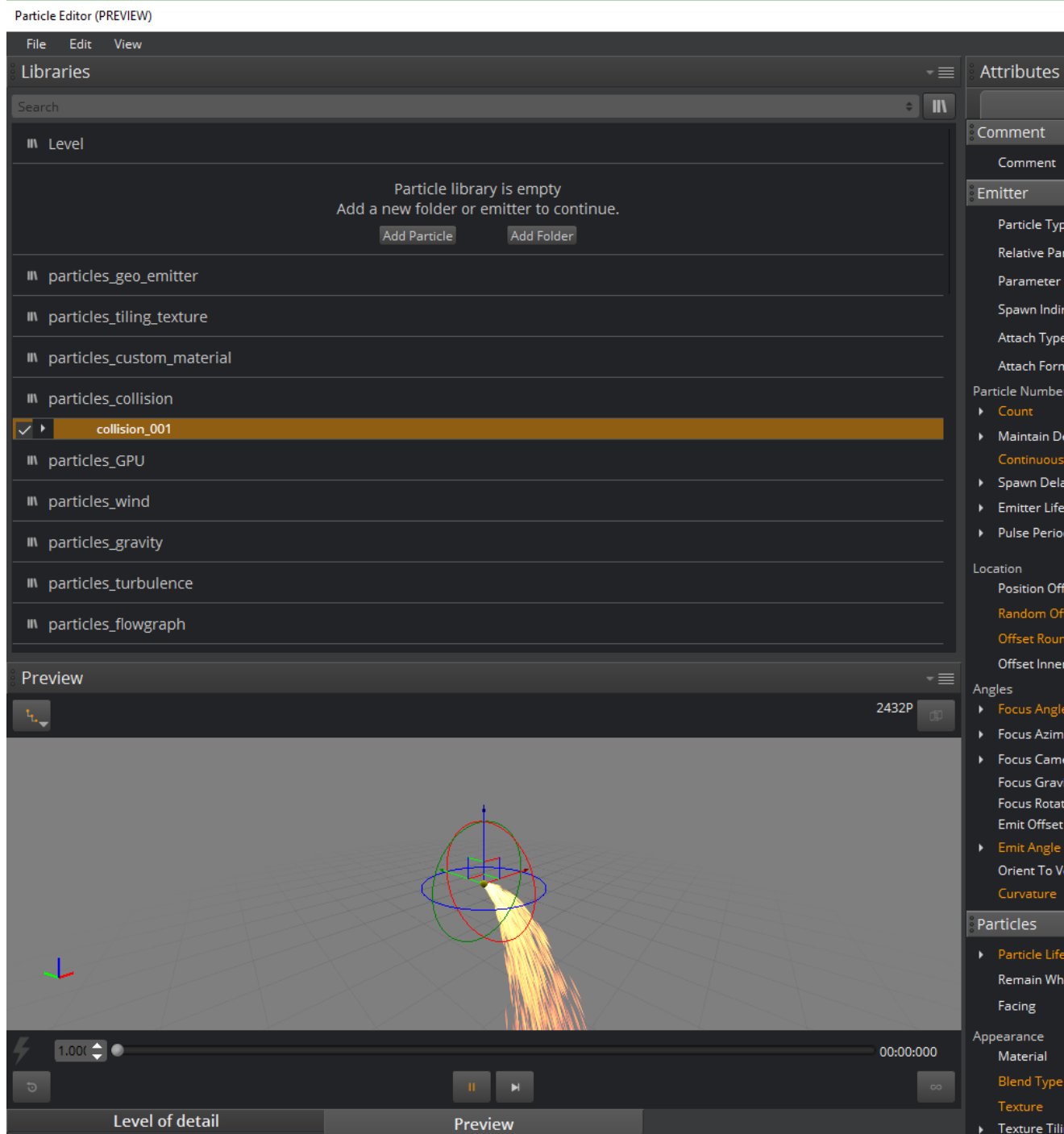
Particles Technical Sample



This sample demonstrates how to create particle systems and effects by changing various attributes in the **Particle Editor**. The sample level includes 10 particle samples that illustrate how to manipulate the particles using two physics entities, flow graph nodes, and entity links.

The sample level is located in the `\dev\SamplesProject\Levels\Samples\Particles_Technical_Sample` directory and the particle libraries are located in the `\dev\SamplesProject\libs\particles` directory.

You can view particle effects and interact with individual particles in Lumberyard Editor by clicking **View, Open View Pane, Particle Editor**. The particle effects are located in the **Libraries** pane.



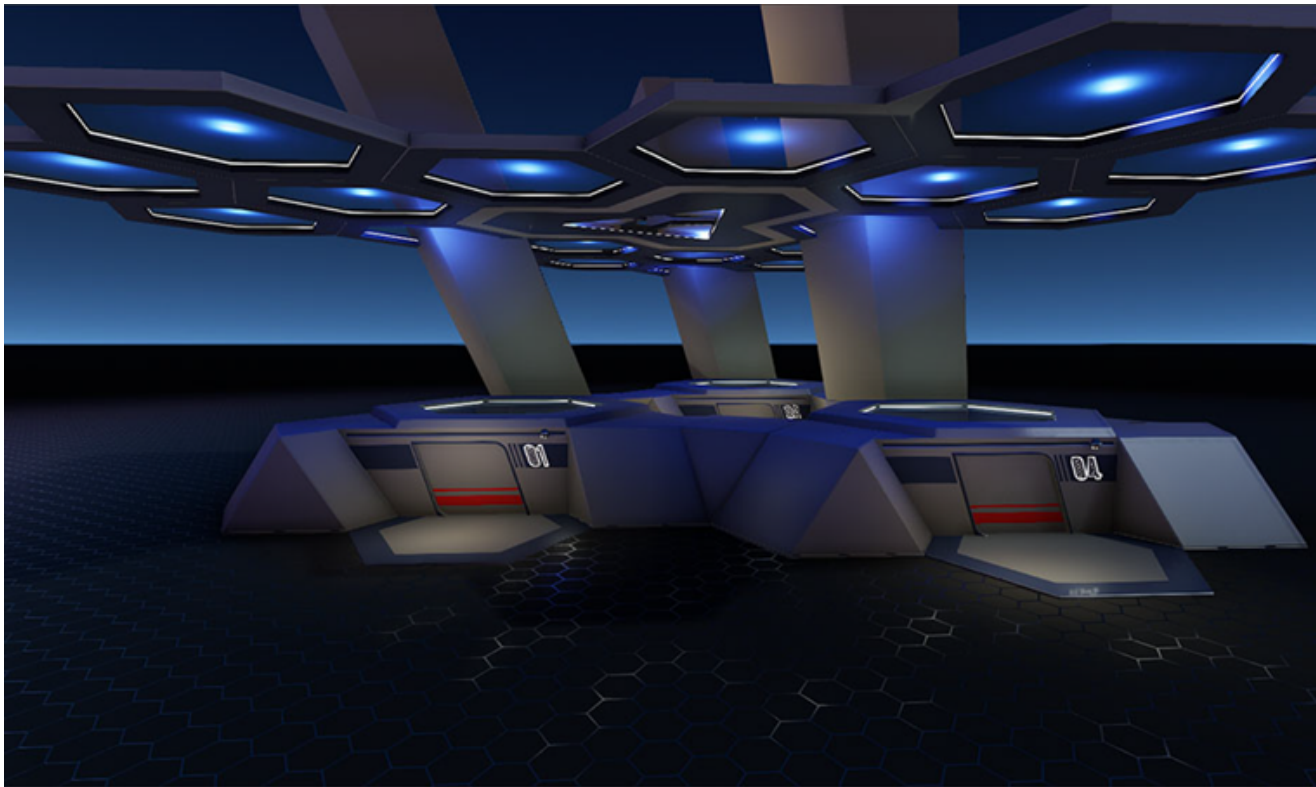
Once the particles technical sample level is open, in the **Perspective** viewport in Lumberyard Editor, you can use the same keyboard keys and mouse controls as [the section called “Particles Sample” \(p. 1095\)](#).

Examples for this sample level include the following:

- **Station 1 – Particle Collision** demonstrates particles that collide with an object.
- **Station 2 – Mesh Particles** demonstrate geometry attached to particles.

- **Station 3 – GPU Particles** demonstrate the particles that are processed and rendered entirely by the graphics card GPU.
- **Station 4 – Gravity Entities** demonstrate particles that are manipulated by the physics entity called **GravitySphere**.
- **Station 5 – Turbulence** demonstrates how the turbulence attributes are used to create a vortex effect.
- **Station 6 – Scripted Wind Speed** demonstrates particles that are manipulated by the physics entity called **WindArea**. The **WindArea** entity is also manipulated by flow graph.
- **Station 7 – Wind Entity** demonstrates particles that are manipulated by the physics entity called **WindArea**.
- **Station 8 – Animated Texture** demonstrates how to use a texture atlas to animate textures that are attached to particles.
- **Station 9 – Custom Particle Material** demonstrates how the appearance of a particle can be changed by using a custom material.
- **Station 10 – Target Attraction** demonstrates how an entity link and flow graph can be used together to make particles target an object and follow it.

Trigger Sample



This sample demonstrates ways to use trigger volumes to activate events within a level. In this sample the event is opening or closing a door. Each example is fully annotated within the flow graph scripts of the level file. It demonstrates various uses for proximity and area triggers.

Examples for this project include the following:

- **Example 1** – Shows a proximity trigger set to only be activated by the player. The metal sphere above the door does not activate the trigger.

- **Example 2** – Shows a proximity trigger with **OnlyPlayer** disabled. Any entity can successfully activate the trigger.
- **Example 3** – Shows a proximity trigger with **OnlyOneEntity** enabled. The first entity must leave before the trigger can be activated again.
- **Example 4** – Shows a proximity trigger with **OnlySelectedEntity** enabled. For the trigger properties, the sphere's name has been added as a string to admit only entities with that specific name.
- **Example 5** – Shows a proximity trigger with simple flow graph logic requiring three entities in the trigger. Both spheres above the door and the player must be in the trigger area for it to activate.
- **Example 6** – Shows an area shape and area box. Both areas are linked to a single trigger.
- **Example 7** – Shows three area triggers that are overlapping. The player must stand in the middle of all trigger areas in order to activate the trigger.
- **Example 8** – Shows three area triggers that must be activated, but in no particular order.
- **Example 9** – Shows three area trigger plates that must be activated in a specific order.

Use the following robot keyboard keys and mouse controls:

- Press the **W**, **A**, **S**, and **D** keys to move forward, left, backward, and right, respectively.
- Move the mouse to look around.
- Press **Spacebar** to jump.

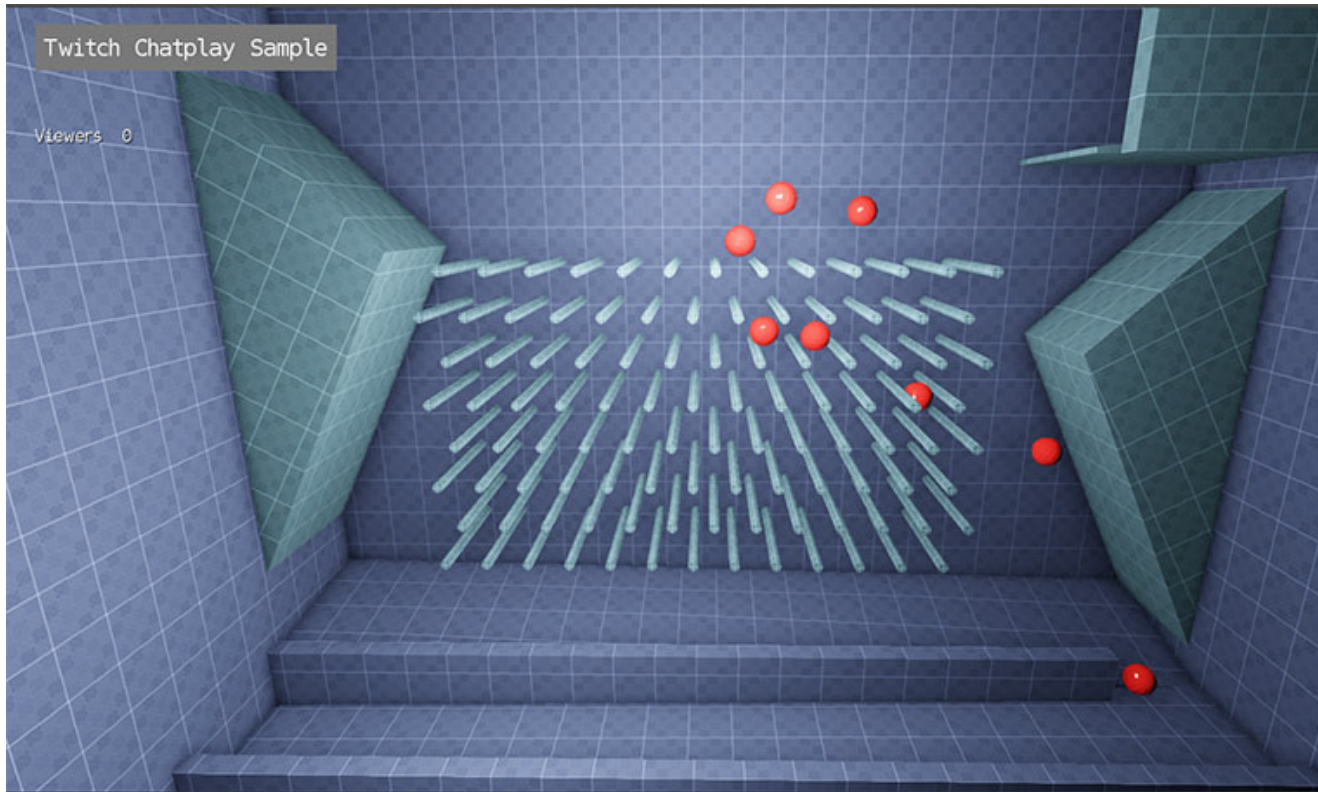
Drive the game robot to each numbered door in this sample to activate the trigger and see an annotated explanation.

UIEditor Sample

This sample demonstrates how to create a basic main menu using the UI Editor. For more information, see [UI System \(p. 1137\)](#).

VR Box Garden Sample

Twitch Chat Basics



Located in the `\TwitchChatBasics` directory, this sample demonstrates Twitch ChatPlay features by demonstrating how to connect a Twitch chat windows text input directly to a trigger event within a game level. In this specific example, Twitch chat users can type in a primary or secondary color and a ball of that color spawns into the level and bounces through a set of pins.

Each section of the **Flow Graph** editor has been annotated to show the steps required to make specific events occur. The essential elements to the chat experience are highlighted as well.

For more information on Twitch-related flow graph nodes, see [Using Flow Graph with Twitch ChatPlay \(p. 1133\)](#).

Also included is a debug script with which you can manually push the number of users up to a set number to verify that the count of users works when no users are available for testing.

Multiplayer Sample Project

The multiplayer sample project enables you to evaluate Amazon GameLift and test Lumberyard's multiplayer capabilities. For more information on GameLift, see the [Amazon GameLift Developer Guide](#).

MultiplayerLobby

This level demonstrates a multiplayer lobby using GridMate networking and LyShine UI. Current features include the following:

- Display list of servers on local LAN.

- Connect to a server.
- Create a new server.

To create a server

1. Enter the server name and map/level name in the **Create a Server** form.
2. Click **Create Server**.

The game automatically starts hosting and loads the selected map.

To connect to a server

1. Click **Refresh** if the server doesn't appear in the server browser list.
2. Click on the row that contains your server name to select it.
3. Click **Connect**.

MultiplayerGame

This level demonstrates a simple multiplayer game using GridMate networking. Current features include:

- Players can connect, reconnect, and disconnect at any time.
- Players can control the movement of an in-game robot, demonstrating delegating network aspects to a client.
- Players can see other player's robots moving, demonstrating network replication of client-delegated physics.
- Players can play robot soccer or football by hitting a ball into a goal. This demonstrates a server Lua script invoking a method in a client Lua script (RMI).
- Players can see the ball in the same place as other players, demonstrating network replication of server-delegated physics.
- The number of goals scored is displayed on a screen in the game.

To play this game you need to create a dedicated server, or you can have a client host the server. After the server is running, you can connect clients to it.

To create a dedicated server

1. Open a command prompt and navigate to the `lumberyard_root_folder\dev` folder.
2. Type the following at the command prompt: `lmbr_waf configure build_win_x64_release_dedicated -p game_and_engine --enabled-game-projects MultiplayerProject --progress.`

Note

For more information on what this command does, see **Step 2: Build the multiplayer project dedicated server** of the Gamelift tutorial: [Tutorial: Packaging your server build](#).

3. Run `Bin64.Dedicated/MultiplayerProjectLauncher_Server.exe`.
4. From a command line prompt, type `mphost`, then press **Enter**.
5. Type `map multiplayergame`, then press **Enter**.

To create a client-hosted server (listen server/peer hosted)

1. Run `Bin64/MultiplayerProjectLauncher.exe`.

2. From a command line prompt, type `map multiplayerlobby`, then press **Enter**.
3. Click the `Create Server` button in the Lobby UI.

To connect clients to a server

1. Run `Bin64/MultiplayerProjectLauncher.exe`.
2. From a command line prompt, type `map multiplayerlobby`, then press **Enter**.

Note

When running the server locally, set `sv_port 0` before calling the map.

3. Click `Refresh` if the server you started doesn't appear in the server browser list.
4. Click on the row that contains your server name, then click **Connect**.

How to play the game

- Use the WASD keys and the mouse to control the robot's movement and orientation.
- Press the spacebar to jump.
- Use the robot to hit the ball down the field and into a goal.
- When the ball enters a goal, the scoreboard updates and the ball returns to the center of the field.

GameLiftLobby

This level demonstrates a multiplayer lobby using GameLift, GridMate Networking, and LyShine UI. Current features include the following:

- Display list of GameLift game sessions.
- Connect to a GameLift game session.
- Create a new GameLift game session.

How to play the game

- Use the WASD keys and the mouse to control the robot's movement and orientation.
- Press the spacebar to jump.
- Use the robot to hit the ball down the field and into a goal.
- When the ball enters a goal, the scoreboard updates and the ball returns to the center of the field.

Legacy Sample Project (GameSDK)

This sample project illustrates the legacy GameSDK functionality.

To download and access GameSDK

1. Download the `GameSDK.zip` package at [Lumberyard Downloads](#) and extract it in your Lumberyard directory.
2. Open the Project Configurator (located in the `lumberyard_root_folder\dev\Bin64\` directory).
3. In the Project Configurator, select **GameSDK**.
4. Click **Set as default**.
5. Click **Launch editor**.

6. Allow the Asset Processor to load all of the project assets. This may take a few minutes. When finished, close the Asset Processor.

Note

Audiokinetic Wave Works Interactive Sound Engine (Wwise) version 2014.1.14 or later is required to access audio for this project.

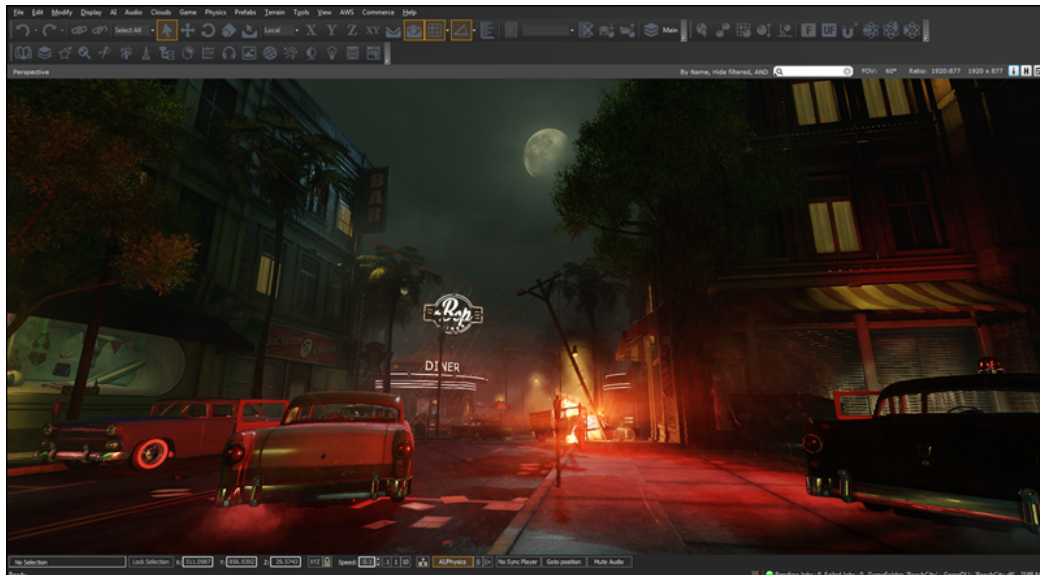
Beach City Sample Project

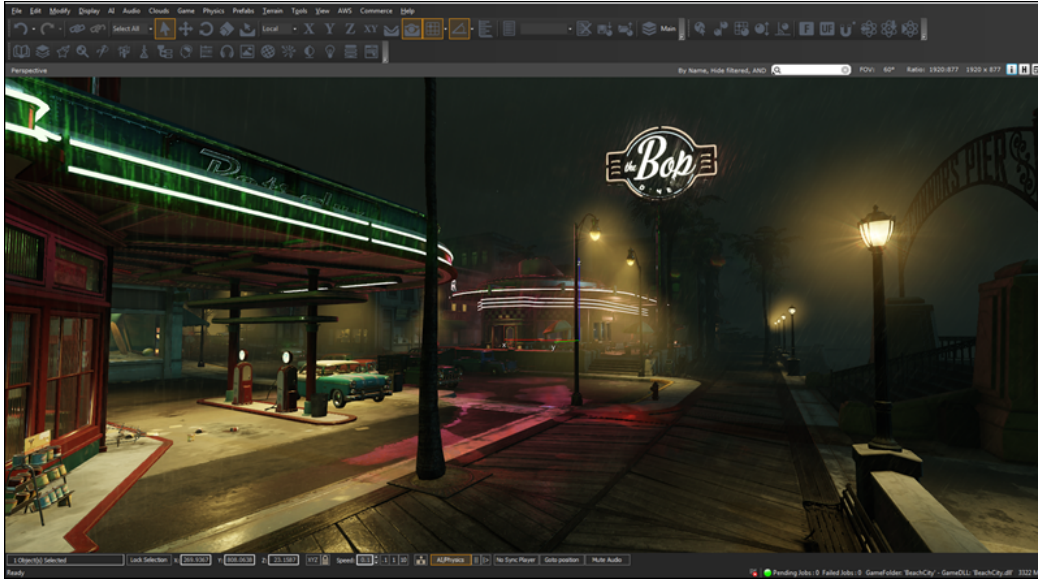
The Beach City sample project includes free assets that you can use to create your own levels. Although the Beach City sample project is intended to be a visual demo and is not a playable demo, you can add player controls to the level if you wish to make it playable.

To install the Beach City sample project

1. Download the `BeachCity.zip` package at [Lumberyard Downloads](#) and extract it in your Lumberyard directory.
2. Open the Project Configurator (located in the `lumberyard\dev\Bin64\` directory).
3. In the Project Configurator, select **BeachCity**.
4. Click **Set as default**.
5. Click **Launch editor**.
6. Allow the Asset Processor to load all of the project assets. When finished, close the Asset Processor.

Sample images from the Beach City sample project:





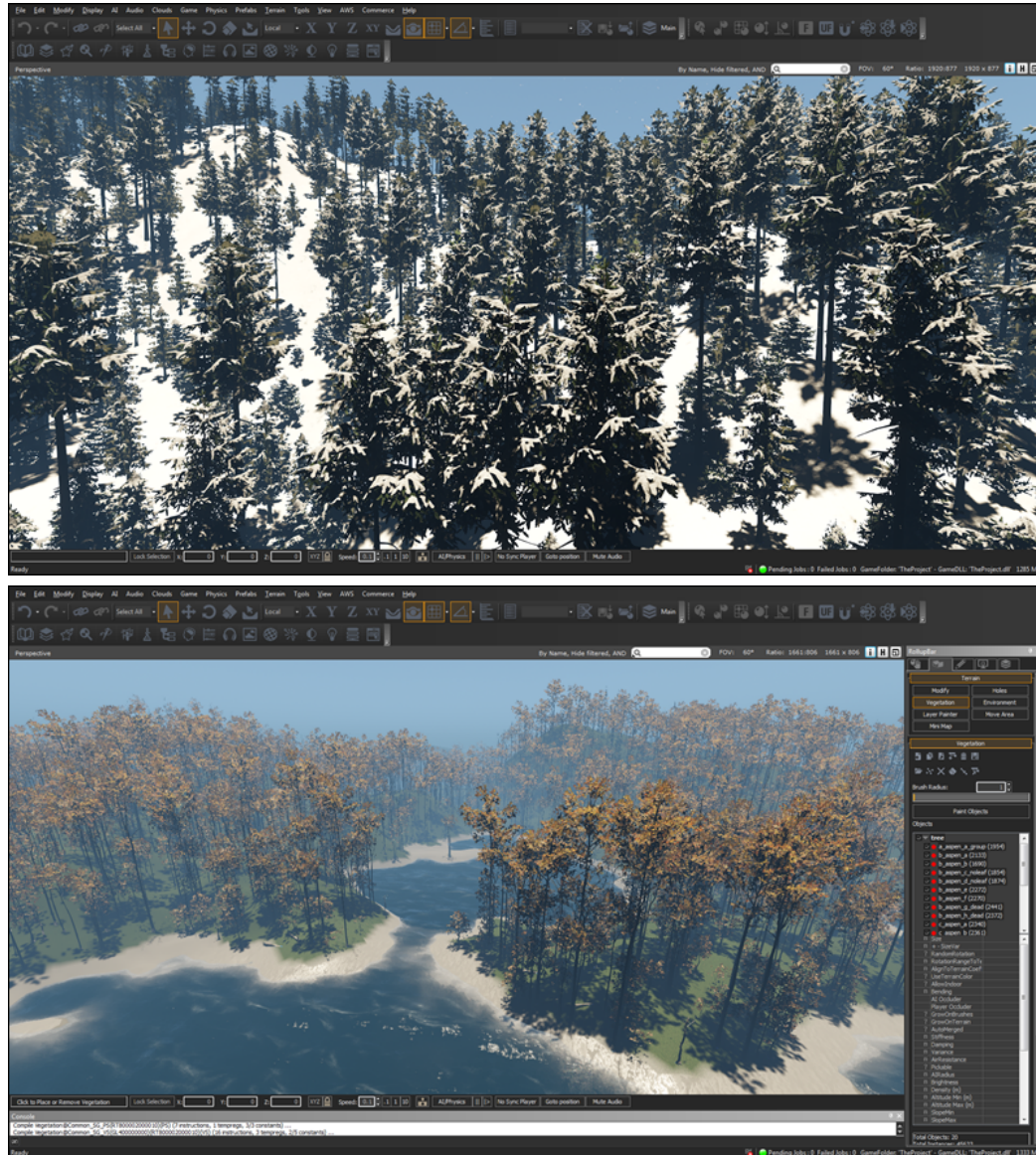
Woodland Asset Package

The Woodland asset package includes free wilderness assets that you can use to create a forest scene or populate your own levels with vegetation and other natural features that might be found in a woodlands scene. The Woodland assets are static art assets; therefore, you will not need to recompile your project after adding these assets.

To install the Woodland asset package

1. Download the Woodland Asset Package at [Lumberyard Downloads](#) and extract it in the `\dev\Gems\AssetCollection_Woodland` directory at the root of your Lumberyard installation. You may need to create this folder or rename the extracted folder.
2. Open the Project Configurator (located in the `\lumberyard\dev\Bin64` directory).
3. In the Project Configurator, under your project, click **Enable Gems**.
4. On the **Gems (extensions)** page, select **Woodland Asset Collection**.
5. Click **Save**.
6. Open Lumberyard Editor and do the following:
 - In the Rollup Bar, click **Geom Entity** and select your meshes.
 - Click **View, Open View Pane, Material Editor** and select your materials and textures.
 - Click **View, Open View Pane, Geppetto** and select your animations.

Sample images from the Woodland asset collection:



FeatureTests Project

The FeatureTests project includes a collection of small, self-contained levels that each demonstrate a single core feature (or small, related subset of features) within Lumberyard across all supported platforms, allowing them to be examined or debugged in relative isolation. The project is located in the `\dev\FeatureTests` directory at the root of the Lumberyard installation (`\lumberyard\dev`).

To load or switch levels, do the following:

- In Lumberyard Editor, choose **File, Open**.
- Modify map `level_to_load` in the `FeatureTests/autoexec.cfg` file before running the standalone game on the Windows, iOS, Android, XboxOne, and PS4 platforms.
- Execute the map `level_to_load` command from the local or remote console while running the standalone game.

FeatureTest Controls

The FeatureTests project includes a simplistic fly camera that is enabled in most of the 3D levels that can be controlled as follows:

- **PC** – Use the mouse to look around and the keyboard AWSD keys to move around.
- **Consoles** – Use the gamepad left thumbstick to move around and the right thumbstick to look around.
- **Mobile** – Use the left side of the screen to move around and the right side of the screen to look around.

Most of the levels in the `Input` directory have different controls designed to test specific input methods.

Note

All of the levels in the `UI` directory are 2D.

FeatureTest Levels

The following feature levels are provided in the `FeatureTests\Levels` directory:

Animation feature levels

- Animation
 - AnimationBasic

Input feature levels

- Gestures
 - GestureClickOrTap
 - GestureDrag
 - GestureHold
 - GesturePinch
 - GestureRotate
 - GestureSwipe
- Keyboard
 - KeyboardBasic
- MotionSensor
 - MotionSensorAccelerometer
 - MotionSensorGyroscope
 - MotionSensorMagnetometer
- Touch
 - TouchBasic
 - TouchRayCast

Rendering feature levels

- AmbientOcclusion

- AmbientOcclusionBasic

- Decals
 - Decals

- GeometryBeam (PC only)
 - GeometryBeam

- HumanFeatures (PC only)
 - HumanFeatureEye
 - HumanFeatureHair
 - HumanFeatureSkin

- Lighting
 - LightingBlend

- Reflections
 - ReflectionsScreenSpace
 - ReflectionsWaterVolume

- ScreenEffects
 - ScreenEffectBlur
 - ScreenEffectChromaShift
 - ScreenEffectColorCorrection
 - ScreenEffectDepthOfField
 - ScreenEffectFader
 - ScreenEffectFrost
 - ScreenEffectGhosting
 - ScreenEffectInterference
 - ScreenEffectRainDrops
 - ScreenEffectVisualArtifacts
 - ScreenEffectVolumetricScattering
 - ScreenEffectWaterDroplets
 - ScreenEffectWaterFlow

- Shadows
 - ShadowsPointLight
 - ShadowsSkybox

- Terrain
 - TerrainAndVegetation
 - TerrainAndWater
 - TerrainAndWaterAndVegetation

- WaterVolume

- VisAreas
 - VisAreaBasic
- Weather
 - WeatherCloudBasic
 - WeatherCloudVolume
 - WeatherRain
 - WeatherSnow

UI feature levels

- UI
 - UiAnimation
 - UiComponents
 - UiFontRendering
 - UiLocalizationExample

Testing, Profiling, and Debugging

Lumberyard includes a number of tools that are used for testing builds, profiling performance, and debugging various issues that may be encountered.

Topics

- [Using AZ Test Scanner \(p. 1112\)](#)
- [Statoscope Profiler \(p. 1116\)](#)
- [Debugging Issues \(p. 1127\)](#)
- [Troubleshooting \(p. 1129\)](#)

Using AZ Test Scanner

The AZ test scanner is a tool for running unit tests that are built into Lumberyard libraries and executables. This tool simplifies testing by automatically finding libraries and executables to test while providing the flexibility for developers to focus on testing the parts of Lumberyard they care about.

The AZ test scanner has two components:

- An AZ test runner executable that loads libraries to test and captures the test results
- An AZ test Python module that performs the scanning and reporting functions

Creating Unit and Integration Test Builds

Unit and integration tests are not included in Lumberyard builds by default as they increase the overall size of a game project. Test code can also have unexpected effects on performance. To build components with tests included, you can use a special test variant that works with each configuration.

To create test builds, use the [Waf build system \(p. 1318\)](#) in the same way that you create regular builds. The only difference is that you add `test` to the platform. You can create a test build on Windows using one of the following examples:

```
// Build with tests using debug configuration. Outputs to the
  \Bin64.Debug.Test folder.
lmbr_waf.bat build_win_x64_debug_test -p all

// Build with tests using profile configuration. Outputs to the \Bin64.Test
  folder.
lmbr_waf.bat build_win_x64_profile_test -p all
```

Note

Only Windows debug and profile builds are supported for testing. Other platforms are not supported; nor are release builds.

Running Unit and Integration Test Builds

A completed test build includes the file `AzTestRunner.exe` in the `\Bin64.Test` folder. Although you can use this to run tests, we recommend that you use the test scanner that uses `AzTestRunner.exe` in an automated manner.

You have two ways to use the scanner:

- Include the AZ test module in your Python path: `python -m aztest`.
- Use the `lmb_r_test.cmd` script located in the Lumberyard `\dev` folder. This automatically includes the AZ test module in your Python path and sends all script parameters to the module.

The following example uses the `lmb_r_test.cmd` scripts. The scanner has several options but only requires one parameter to operate: the build directory to scan. You can use the following command to do a scan of your entire test build:

```
// Scan entire test build and run all found tests
lmb_r_test.cmd scan --dir Bin64.Debug.Test
```

Note

The default scan only tests libraries. It does not attempt to test any executables it finds. This is because executables that are not set up to run tests interrupt the scanner until you close the application.

The scanner produces three types of files. All files are created in the current working directory from which the scanner is called:

- The `aztest.log` file that contains a log of all test output
- Several `.xml` files that contain the test results of each library and executable that has tests, time stamped by default
- An `.html` file that contains a summary of the test results from the entire scan, time stamped by default

The full list of options is shown as follows:

The scanner runs only unit tests by default. This is because unit tests are designed to be fast and do not rely on engine resources. To run integration tests instead, use the `--integ` flag when calling the scanner:

```
// Scan test build and run integration tests on CrySystem.dll
lmb_r_test.cmd scan --dir Bin64.Debug.Test --only CrySystem.dll --integ
```

Note

For best results run integration tests on a single library or use a whitelist as scanning the full build may take hours to complete.

Option	Required?	Description
<code>--dir, -d</code>	Yes	The directory to scan for tests.

Option	Required?	Description
<code>--runner-path</code>	No	Path to the AZ test runner executable (the default is to look in the directory specified by <code>--dir</code>).
<code>--add-path</code>	No	Adds path to system path before running tests; used for resolving library or executable dependencies.
<code>--output-path</code>	No	Sets the path for output folder prefix (the default is <code>\dev\TestResults</code>).
<code>--integ</code>	No	If set, runs integration tests instead of unit tests.
<code>--no-timestamp</code>	No	If set, removes the timestamp from output files.
<code>--wait-for-debugger</code>	No	If set, tells the AZ test runner executable to wait for a debugger to be attached before running tests.
<code>--bootstrap-config</code>	No	Path to a JSON configuration file for bootstrapping applications required by libraries.
<code>--limit, -n</code>	No	Sets a limit for the maximum number of modules to scan.
<code>--only, -o</code>	No	Sets a filter to run tests on only the specified library or executable name.
<code>--whitelist</code>	No	Sets the type of whitelist the scanner should use. O, options are <code>spec</code> , <code>none</code> , or <code>file</code> (the default is <code>spec</code>).
<code>--whitelist-file</code>	Only if <code>--whitelist</code> is set to <code>file</code>	Path to a <code>spec</code> file or new line–delimited file used for whitelisting (the default is <code>\dev_WAF_specs\all.json</code>). The new line–delimited file allows for regular expressions when matching.
<code>--include-gems, -g</code>	No	If set, the scanner will search for gems and include them in the current whitelist.
<code>--include-projects, -p</code>	No	If set, the scanner will search for game projects and include them in the current whitelist.
<code>--blacklist-file</code>	No	Path to a new line–delimited file used for blacklisting. The blacklist takes precedence over the whitelist. The new line–delimited file allows for regular expressions when matching.
<code>--exe</code>	No	If set, causes the scanner to call executables for testing. (The default is to test only libraries).

The scanner also accepts additional parameters that are passed to the testing framework. For Lumberyard, GoogleTest, and GoogleMock for C++ are used for unit testing. You can type parameters in the scanner command line as shown in the following example:

```
// Scan CrySystem.dll and shuffle the test order before running
lmb_r_test.cmd scan --dir Bin64.Test --only CrySystem.dll --gtest_shuffle
```

The scanner can also be called as a chained command using Waf. This means that you can build tests and run them using a single command line. The Waf command `run_tests` will call the scanner on the most recent build folder. For example:

```
// Build a debug test build and then run tests in it
lmbr_waf.bat build_win_x64_debug_test -p all run_tests
```

The `run_tests` command will automatically point to the `\Bin64.Debug.Test` folder to scan as well as use the all spec for whitelisting. The build step is not necessary to use `run_tests`, it will always use whatever the last build was. You can also send all of the scanner parameters through using `--test-params`:

```
// Run tests on the last build with additional parameters (use quotes to
capture as string)
lmbr_waf.bat run_tests --test-params="--include-gems --include-projects --no-
timestamp"
```

You can also use the `--target` flag to build and test just one module:

```
lmbr_waf.bat build_win_x64_debug_test -p all --target CrySystem run_tests
```

Whitelisting and Blacklisting

The test scanner includes the ability to use whitelist and blacklist files to filter out libraries and executables that you do not want to run tests on. The scanner uses Waf spec files for whitelisting as the default method, but a user-defined file can also be used or the whitelist can be turned off entirely. Blacklisting is off by default. In all cases, modules that are blacklisted are never tested even if they are included in the whitelist.

The scanner uses the all spec as the default whitelist to cover the most common scenario for developers. This sets up the scanner so that it only tests libraries and executables created by Waf, not including gems and game projects. The following examples are equivalent:

```
// This...
lmbr_test.cmd scan --dir Bin64.Test --whitelist spec --whitelist-file _WAF_
\specs\all.json
```

```
// is the same as this...
lmbr_test.cmd scan --dir Bin64.Test
```

Gems and game projects are not automatically included because they are not listed in spec files. To include gems and game projects in the whitelist, use the appropriate flags as in the following example:

```
lmbr_test.cmd scan --dir Bin64.Test --include-gems --include-projects
```

To turn whitelisting off, use the following command line:

```
lmbr_test.cmd scan --dir Bin64.Test --whitelist none
```

Both whitelisting and blacklisting can use a new line-delimited text file for defining modules as well. Each line is treated as a regular expression for matching, allowing for easy filtering by modules with similar names or in the same directory. Here is an example file:

```
# List files directly (remember to escape backslashes in regex)
CrySystem.dll
rc\\ResourceCompilerPC.dll
```

```
# Match similar modules using regex (include all gem libraries)
Gem\\.\\.\\.\\.dll
```

```
# Match all in a subdirectory using regex
EditorPlugins\\.\\.\\.*
```

To run the scanner using text files, use the following example:

```
lmbr_test.cmd scan --dir Bin64.Test --whitelist file --whitelist-file
my_whitelist.txt --blacklist-file my_blacklist.txt
```

Statoscope Profiler

Statoscope is a profiling tool that displays per-frame instrumented data. It is used for evaluating performance metrics such as overall CPU time spent, memory usage tracking, and statistics rendering. It records values from Lumberyard and displays how they change over the course of time.

Statoscope will connect to any platform or console that is connected directly to the PC or via an IP address specified in the connection settings dialog.

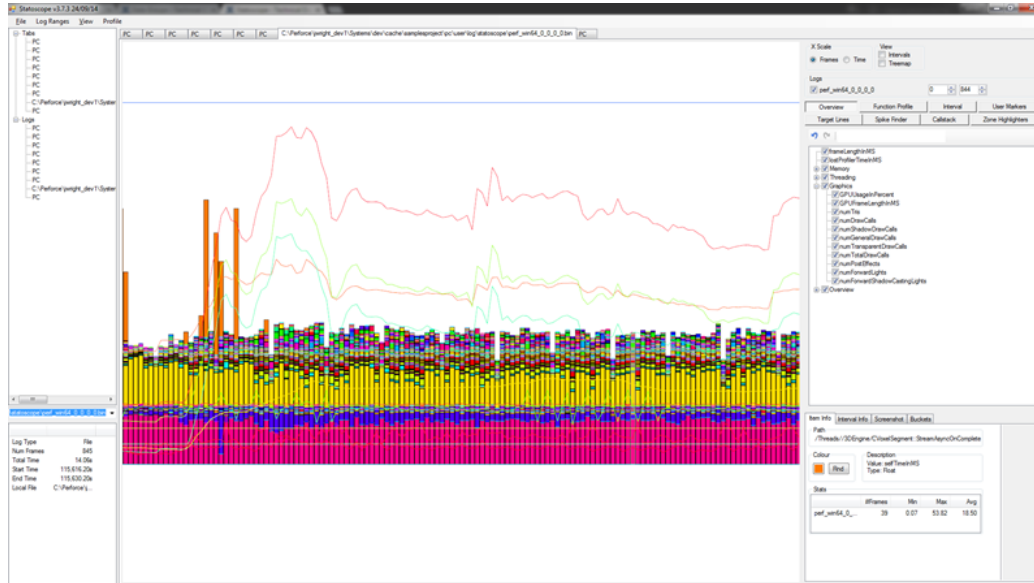
Topics

- [User Interface \(p. 1116\)](#)
- [Logging Data \(p. 1118\)](#)
- [Filtering Data \(p. 1119\)](#)
- [Data Groups \(p. 1120\)](#)
- [Creating Data Groups \(p. 1125\)](#)
- [Guidelines and Best Practices \(p. 1126\)](#)

User Interface

Statoscope data can be displayed as lines (such as for fps, number of drawcalls, memory usage, and threads), as bars (such as for function profiles or per-entity bandwidth statistics), as intervals (such as the status of queued streaming tasks), or as user markers, which are vertical lines displayed when infrequent actions occur (such as invalid file access or level load/unload).

You must select all nodes (and the parent node) down to the selected node in order to see data displayed for that item. When using the data group selection trees, right-click toggles selection of the entire sub-tree.



Here are the basic navigation methods for the Statoscope graph. The x-axis is displayed in both frame numbers and elapsed seconds while the y-axis is displayed in milliseconds.

- To pan: left-click and drag
- To zoom: right-click and drag
- To scale horizontally: right-click hold and drag left/right
- To scale vertically: right-click hold and drag up/down
- To scale along both axes: right-click and drag top right and bottom left

Note

To reset the viewport, select **View, Fit To Frame Records**. This is useful if the data is off-screen from zooming in too much.

Function Profiling

As there are usually many more frames in a log than can easily be shown at once, only a subset of bars are displayed when zoomed out. This is indicated by the bars being displayed at 50% opacity.

The bars displayed are individual frames. The ones selected are the tallest of the range that they represent. This makes it easy to identify unusual spikes even when zoomed out.

Function profiling is enabled using the **e_StatoscopeDataGroups r** console variable.

Clicking on a bar selects that entry on the **Function Profile** tab, with focus moved to the tree view, so you can press the spacebar to unselect and hide that bar quickly. This is useful for eliminating profiling noise.

Hovering

When mouse hovering, a vertical red line clips to the nearest frame and a tooltip will follow your cursor over the window, displaying the following information for a selected frame:

- Top line: frame number, game time, and y-axis value
- Second line: What item you are hovering over
- Third line: The time elapsed in ms

Axes Scaling

The x-axis is linear for number of frames by default. This is useful for function profiling since all bars have the same width. You can also select the x-axis to be linear for time instead.

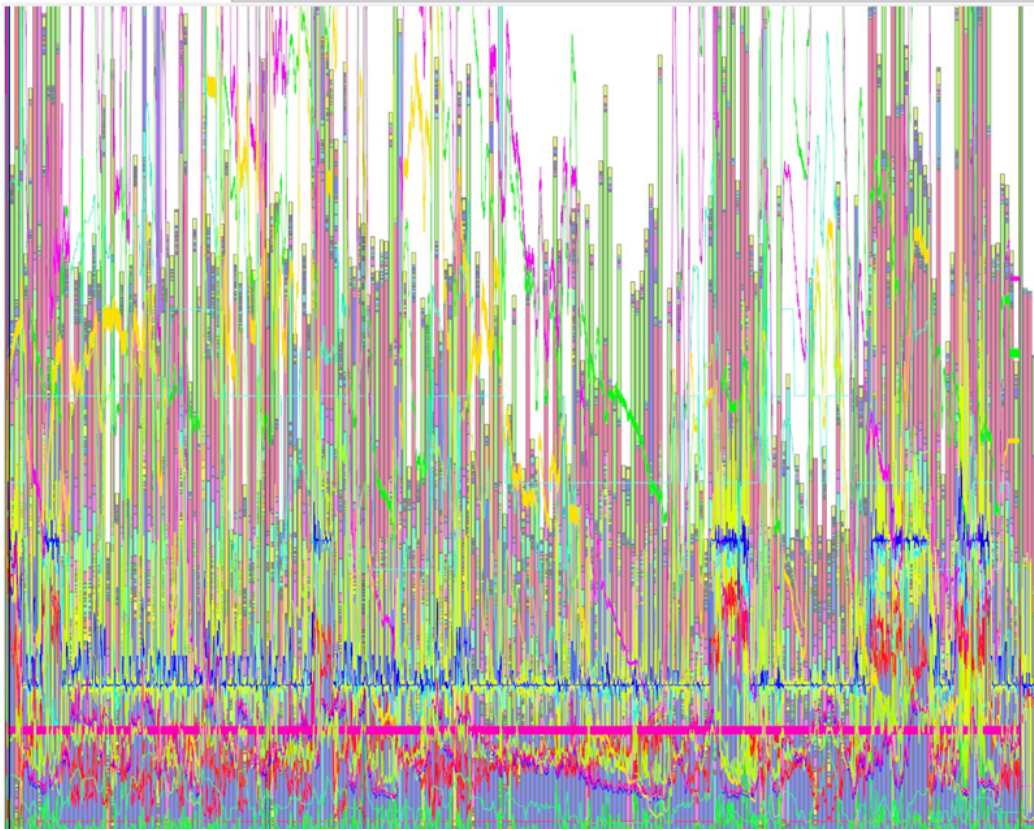
The y-axis can be scaled in order to compare data that varies greatly in value, such as number of drawcalls.

You can also specify which target lines are displayed.

Logging Data

Logging game data using Statoscope involves selecting data groups. Select only the data groups you want to log and display to minimize the performance impact.

The following figure shows the default view with everything enabled and the entire log fit to the viewport graph. You can unselect unneeded data groups and zoom out to make the graph more readable.



You can log data directly to Statoscope using a socket or you can log data to a file.

If you just want to record some data to see how your game is performing, socket logging is recommended. This gives you real-time updates in Statoscope and avoids the maintenance of having log files.

If you want to log QA sessions or compare time demo runs, file logging is recommended.

Logging Data to a Socket

The following procedure shows how to log data directly to the Statoscope application using a socket.

To log data to Statoscope (socket)

1. Run a Profile game client on your chosen platform (such as `SampleProjectLauncher.exe` for example).
2. Set the relevant console variables either after the game client loads or by editing the `bootstrap.cfg` file. The following example would enable logging data from all threads with frame rate limiters disabled:
 - `profile_allthreads 1`
 - `r_Vsync 0`
 - `sys_maxFPS -1`
 - `e_StatoscopeLogDestination 1`
 - `e_StatoscopeEnabled 1`
 - `e_StatoscopeDataGroups your_data_groups`. Default data groups are `fgmtu0`.
3. Run `Statoscope.exe` from `\Tools\Statoscope`.
4. In Statoscope, select **File, Connect**. For Windows, accept all defaults. For consoles, enter the IP of your developers kit.
5. Select **Log to file**, then select the file name to log to. Select the file name quickly or else your session may timeout. For more information, see [Guidelines and Best Practices \(p. 1126\)](#).

You should see your selected data groups being logged.

Logging Data to a File

The following procedure shows how to log data to a file.

To log data to a file

1. Run a Profile game client on your chosen platform (such as `SampleProjectLauncher.exe` for example).
2. Set the relevant console variables either after the game client loads or by editing the `bootstrap.cfg` file. The following example would enable logging data from all threads with frame rate limiters disabled:
 - `profile_allthreads 1`
 - `r_Vsync 0`
 - `sys_maxFPS -1`
 - `e_StatoscopeLogDestination 0`
 - `e_StatoscopeDataGroups your_data_groups` Default data groups are `fgmtu0`.
3. Set the `e_StatoscopeEnabled 1` console variable from the game client to enable Statoscope.
4. Run `Statoscope.exe` from `\Tools\Statoscope`.
5. In Statoscope, select **File, Open the log file**. Log files for Statoscope are located at `\cache\launchername\platform\user\log\statoscope\perf_<config>_0_0_0_0.bin`. For Windows, an example file path would be `\cache\samplesproject\pc\user\log\statoscope\perf_win64_0_0_0_0.bin`.

You should see your selected data groups being logged.

The most recent capture overwrites any existing capture.

Filtering Data

There are a number of data filtering options available in Statoscope.

Use the **Overview** and **Function Profile** tabs to access and then select and deselect data plots. For a plot to be drawn, it and all its parents in the tree must be selected.

There are several shortcuts for selecting items:

- **Ctrl+left-click** label: select just that item
- **Right-click** label: selects or deselects every item under the selected item in the hierarchy.
- **Shift+left-click** label (**Function Profile** tab): collapses all children into a single bar color. This will cause the label to have a gray background. This is useful for seeing the performance cost of a whole thread or profile module.

Item Info tab

This tab is used to control how a data item is displayed and shows some basic data.

Line and bar data colors can be changed by clicking the color swatch button to get a color picker dialog, or by clicking the **Rnd** button to select a random color.

Basic statistics shown include the number of frames the data is present for in the log, and the corresponding minimum, maximum, and average values. In the case of hierarchical bar data, this will represent the total of all selected children.

Line data can be filtered to make it easier to see trends.

Moving Average (MA) shows the same line averaged out using the values from a number of frames on either side of the current frame, with five frames being the default.

Local Maximum (LM) is useful for data that varies consistently for each frame, such as time-sliced shadows for example.

Enabling either of these will hide the base item by default. You can only display the information in one mode at a time: **Off**, **MA**, or **LM**.

Screenshot tab

Screenshots are useful for seeing what happened while the log was being recorded. They are captured at 1/8 resolution to keep the log file size small.

To enable screenshots, set the **e_StatoscopeScreenshotCapturePeriod** console variable and enter the number of seconds between screenshots, with a value of -1 to disable and a value of 0 to capture screenshot frames continuously.

To view the screenshots during a captured session, hover the mouse over the timeline horizontally to view the screenshots updating.

Buckets tab

Available buckets are: Overall fps, RT fps, GPU fps, Triangles, Total Draw Calls, Shadow Draw Calls, Draw Calls and Texture Pool.

The 5fps clamp referred to in some columns treats frames whose length is longer than 200ms (5fps) as if it was 200ms. This is useful to stop very long frames from skewing the data too much.

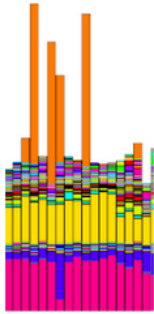
Data Groups

Data groups represent categories or types of data that will be logged in Statoscope, with each group represented by a single lowercase or uppercase letter. Data groups are controlled using the **e_StatoscopeDataGroups** console variable.

You can control which data groups are displayed by selecting groups from the tree on the **Overview** tab on the righthand side.

Select only the data groups you want to display to minimize the performance impact.

The most important data group to verify is `frameprofilers` or `r`, as shown below. Each vertical bar represents one frame, and each color band represents the total time spent inside one profile event for that frame. You can select and unselect entire threads from the **Function Profile** tab.



Lowercase Data Groups

CPU Times 'j'

- physTime
- particleTime
- particleSyncTime
- particleNumEmitters
- animTime
- animNumCharacters
- aiTime
- flashTime

dev buffer 'b'

- written_kb
- read_kb
- creation_time
- io_time
- cpu_flush
- gpu_flush
- cb

frame lengths 'f'

- frameLengthInMS

frame profilers 'r'

- name
- count

- selfTimeInMS

GPU Times 'i'

- Scene
- SceneRSXProfWait
- Shadows
- ZPass
- DeferredDecals
- DeferredLighting
- Ambient
- Cubemaps
- SSAO+GI
- Lights
- Opaque
- Transparent
- Fog
- HDR
- PostFX

graphics 'g'

- GPUUsageInPercent
- GPUFrameLengthInMS
- numTris
- numDrawCalls
- numShadowDrawCalls
- numGeneralDrawCalls
- numTransparentDrawCalls
- numTotalDrawCalls
- numDrawCallsRejectedByConditionalRendering
- numPostEffects
- numForwardLights
- numForwardShadowCastingLights
- numSpriteDIPS
- numSpriteUpdates
- numDoubleSizedSprites
- spriteAtlasSize
- spriteAtlasRequirement
- numSpritePolys
- maxDiffPtrKb
- maxDiffPtrTangKb
- maxRendIndicesKb

memory 'm'

- mainMemUsageInMB

particles 'p'

- numParticlesRendered
- numParticlesActive
- numParticlesAllocated
- numParticlesRequested
- particleScreenFractionRendered
- particleScreenFractionProcessed
- numEmittersRendered
- numEmittersActive
- numEmittersAllocated
- numParticlesReiterated
- numParticlesRejected
- numParticlesCollideTest
- numParticlesCollideHit
- numParticlesClipped

per-cgf gpu profilers 'c'

- totalDrawCallCount
- numInstances

PhysEntities 'w'

- name
- time
- nCalls
- x
- y
- z

streaming 's'

- cgfStreamingMemUsedInMB
- cgfStreamingMemRequiredInMB
- numActiveTextureNodes

streaming textures 'x' – memory numbers yes, bandwidth numbers no

- numUpdated Ups/s
- numRequested Req/s
- numRendered UpsRen/s
- poolMemUsed MB
- poolMemWanted MB

threading 't'

- MTLoadInMS
- MTWaitingForRTInMS

- RTLoadInMS
- RTWaitingForMTInMS
- RTWaitingForGPUInMS
- RTFrameLengthInMS
- RTSceneDrawingLengthInMS

user markers 'u'

- path
- name

Vertex data 'v'

- StaticPolyCountZ
- SkinnedPolyCountZ
- VegetationPolyCountZ

Uppercase Data Groups

art profile 'A'

- GPU
 - ShadowsMS
 - ZPassMS
 - DecalsMS
 - LightingMS
 - OpaqueMS
 - TransparentMS
 - totalMS
 - Detail
 - Lights
 - AmbientMS
 - CubemapsMS
 - DeferredMS
 - ShadowMapsMS
 - ReflectionsMS
 - CausticsMS
 - RefractionOverheadMS
- Budgets
 - GPU
 - ShadowsMS
 - ZPassMS
 - DecalsMS
 - LightingMS
 - OpaqueMS
 - TransparentMS
 - totalMS

- numBatches
- numDrawcalls
- numLightingDrawcalls
- numRSXStallReleases (if ENABLE_ACCURATE_RSX_PROFILING is defined)

Texture Information 'S'

- TexStrm
 - engineassets
 - texturemsg
 - codecoverage
 - textures
 - defaults
 - decals
 - sprites
 - etc...
 - objects
 - props
 - vehicles
 - architecture
 - etc...

Creating Data Groups

When adding a new data group to Statoscope, do not choose a letter that's already in use.

Statoscope doesn't need updating when new data groups are added. You simply create an implementation of `IStatoscopeDataGroup` and register it with `CStatoscope::RegisterDataGroup()`. Here's an example of the simplest data group:

```
struct SFrameLengthDG : public IStatoscopeDataGroup
{
    virtual SDescription GetDescription() const
    {
        return SDescription('f', "frame lengths", "[ '/' (float
frameLengthInMS) ]");
    }
    virtual void Write(IStatoscopeFrameRecord& fr)
    {
        fr.AddValue(gEnv->pTimer->GetRealFrameTime() * 1000.0f);
    }
};
...
RegisterDataGroup(new SFrameLengthDG());
...
```

When this data group is enabled by adding `f` to `e_StatoscopeDataGroups`, `frame lengths` will appear in the `e_StatoscopeDataGroups` help string and for every frame it will output a single float value that appears as `/frameLengthInMS` in the **Overview** tree view.

Below is an example frame profilers data group, which shows how to record bar data:

```
struct SFrameProfilersDG : public IStatoscopeDataGroup
{
    virtual SDescription GetDescription() const
    {
        return SDescription('r', "frame profilers", "['/Threads/$' (int count)
(float selfTimeInMS)]");
    }
    virtual void Enable()
    {
        IStatoscopeDataGroup::Enable();
        ICVar *pCV_profile = gEnv->pConsole->GetCVar("profile");
        if (pCV_profile)
            pCV_profile->Set(-1);
    }
    virtual void Disable()
    {
        IStatoscopeDataGroup::Disable();
        ICVar *pCV_profile = gEnv->pConsole->GetCVar("profile");
        if (pCV_profile)
            pCV_profile->Set(0);
    }
    virtual void Write(IStatoscopeFrameRecord &fr)
    {
        for (uint32 i=0; i<m_frameProfilerRecords.size(); i++)
        {
            SPerfStatFrameProfilerRecord &fpr = m_frameProfilerRecords[i];
            string fpPath = GetFrameProfilerPath(fpr.m_pProfiler);
            fr.AddValue(fpPath.c_str());
            fr.AddValue(fpr.m_count);
            fr.AddValue(fpr.m_selfTime);
        }
        m_frameProfilerRecords.clear();
    }
    virtual uint32 PrepareToWrite()
    {
        return m_frameProfilerRecords.size();
    }
    std::vector<SPerfStatFrameProfilerRecord> m_frameProfilerRecords; // the
most recent frame's profiler data - filled out externally
};
```

With bar data as shown, the same format is output many times per frame, in this case count and selfTimeInMS for each named profiler. The number of items needs to be returned by PrepareToWrite(). To specify the name of each item, place a \$ in the appropriate location in the format string of GetDescription() and the first value output will be used to replace it. For this example, if fpPath is Main/Action/CFlowSystem::Update(), the values output will be attributed to /Threads/Main/Action/CFlowSystem::Update() and hierarchied accordingly

Values can either be float or integer, but are stored as floats.

Guidelines and Best Practices

The following are some guidelines and best practices for consideration.

Pressing **Scroll Lock** pauses capturing data.

You must select a log file name quickly or you will timeout and not be able to connect to another session.

If the Statoscope network state is broken (you cannot connect but Statoscope is enabled and you have selected **log to socket**, you can reset the Statoscope network connection by changing the log destination away from and back to log to socket. To accomplish this, change the following console variables in the following order:

1. `e_StatoscopeEnabled 0`
2. `e_StatoscopeLogDestination 0` (to file logging)
3. `e_StatoscopeLogDestination 1` (back to socket logging, this resets the Statoscope network state)
4. `e_StatoscopeEnable 1`

Debugging Issues

Lumberyard has various built-in debugging and profiling tools that help to locate and fix various problems as well as performance issues.

- [AI debugging \(p. 115\)](#) – Used for debugging AI agent behaviors
- [Character skeleton debugging \(p. 153\)](#) – The `p_draw_helpers` console variable is useful for debugging character skeleton issues
- [Cinematics debugging \(p. 319\)](#) – There are several console variables used for debugging cinematics issues
- [Flow Graph debugging \(p. 775\)](#) – Flow Graph Debugger and console variables are used for debugging Flow Graph issues
- [Mannequin debugging \(p. 277\)](#) – There are several methods used for debugging Mannequin system issues
- [Particle debugging \(p. 980\)](#) – Used for debugging particles
- [Vegetation debugging \(p. 881\)](#) – Used for debugging vegetation objects

Using Console Debug Views

You can use the following console variables and values to generate various viewing modes in the viewport that are useful for debugging:

- `e_camerafreeze 1` – Freezes the camera to see what is rendered from the camera's point of view and what is occluded. Also useful to debug object culling and LOD.
- `e_defaultmaterial 1` – Applies a uniform flat gray material to every surface in the level
- `e_terrainbboxes` – Displays terrain bboxes (bounding boxes)
- `p_debug_joints 1` – Shows the mass of objects in kg and the joint linked to the object. To display joints, enable `p_draw_helpers 1` and `p_draw_helpers 1` first.
- `p_draw_helpers 1` – Shows physics proxy meshes additionally to the render geometry.
- `r_displayinfo 1 | 2 | 3` – Displays memory consumption, frame rate, triangle count, visible light sources, and drawcall count. A value of 2 displays more detailed information, while a value of 3 displays only frames per second (FPS) and frame time in milliseconds.
- `r_wireframe 1 | 2` – Draws the level in 1=wireframe mode, 2=vertex mode, including objects hidden from view.
- `r_showlines 2` – Overlays the wireframe only on the front-facing geometry. Anything behind doesn't get rendered.
- `r_texbindmode 6` – Applies a uniform flat gray material with normal map information to every surface in the level.

Using DebugDraw Console Variables

You can use the following console variables and values to display various information about your level:

- `e_DebugDraw 1` – Displays the name of the `.cgf` used, polycount, and LOD
- `e_DebugDraw 2` – Displays a color-coded polygon count
- `e_DebugDraw 3` – Displays a color-coded LOD count, flashing color indicates no LOD information
- `e_DebugDraw 4` – Displays object texture memory usage
- `e_DebugDraw 5` – Displays a color-coded number of render materials
- `e_DebugDraw 6` – Displays ambient color
- `e_DebugDraw 7` – Display triangle count, number of render materials, and texture memory
- `e_DebugDraw 8` – Displays RenderWorld statistics (with view cones)
- `e_DebugDraw 9` – Displays RenderWorld statistics (with view cones without lights)
- `e_DebugDraw 10` – Displays render geometry with simple lines and triangles
- `e_DebugDraw 11` – Displays render occlusion geometry
- `e_DebugDraw 12` – Displays render occlusion geometry without render geometry
- `e_DebugDraw 13` – Displays occlusion amount (used during AO computations). Warn
- `e_DebugDraw 15` – Displays helpers
- `e_DebugDraw 16` – Displays debug gun
- `e_DebugDraw 17` – Displays streaming info (buffer sizes)
- `e_DebugDraw 18` – Displays streaming info (required streaming speed)
- `e_DebugDraw 19` – Displays physics proxy triangle count
- `e_DebugDraw 20` – Displays object instant texture memory usage
- `e_DebugDraw 21` – Displays animated object distance to camera
- `e_DebugDraw 22` – Display object's current LOD vertex count

Using GBuffer Console Variables

You can use the following console variables and values to display various materials, colors, shadows, albedo, and other characteristics in your level:

- `r_DebugGBuffer 1` – Shows normals of all assets in the level
- `r_DebugGBuffer 2` – Shows how rough or glossy that surfaces are
- `r_DebugGBuffer 3` – Shows the specular color of materials
- `r_DebugGBuffer 4` – Shows the albedo of all surfaces in the level
- `r_DebugGBuffer 5` – Shows the lighting model in the level, where gray = standard, yellow = transmittance, and blue = POM self-shadowing.
- `r_DebugGBuffer 6` – Shows the translucency values set on assets in the level, where black = none.
- `r_DebugGBuffer 7` – Shows self-shadowing of materials that use Offset Bump mapping or Parallax Occlusion Mapping.
- `r_DebugGBuffer 8` – Shows in red and yellow any asset that uses SSS. The brighter the color, the higher the SSS index.
- `r_DebugGBuffer 9` – Shows whether specular colors are in a reasonable range as follows:
 - **Blue** – Specular color too low
 - **Orange** – Specular color too high for dielectric materials
 - **Pink** – Valid only for rusted or oxidized metals

Troubleshooting

Topics

- [Viewing Error Log \(p. 1129\)](#)
- [Error Message Reference \(p. 1129\)](#)
- [Art Assets Errors \(p. 1129\)](#)

Viewing Error Log

Error Message Reference

Art Assets Errors

Twitch ChatPlay System

Twitch ChatPlay provides a flexible framework to create customized game interactions between broadcasters and spectators on Twitch, the world's leading social video platform and community for gamers.

Twitch ChatPlay includes support for chat commands, polls, and surveys that can be triggered by Twitch viewers through the Twitch chat channel. For example, you can create a chat command `#cheer` that triggers celebration animations in your game.

Twitch ChatPlay is implemented by a set of flow graph nodes that establish a connection to a Twitch channel and use incoming traffic as a game input, like any other input device.

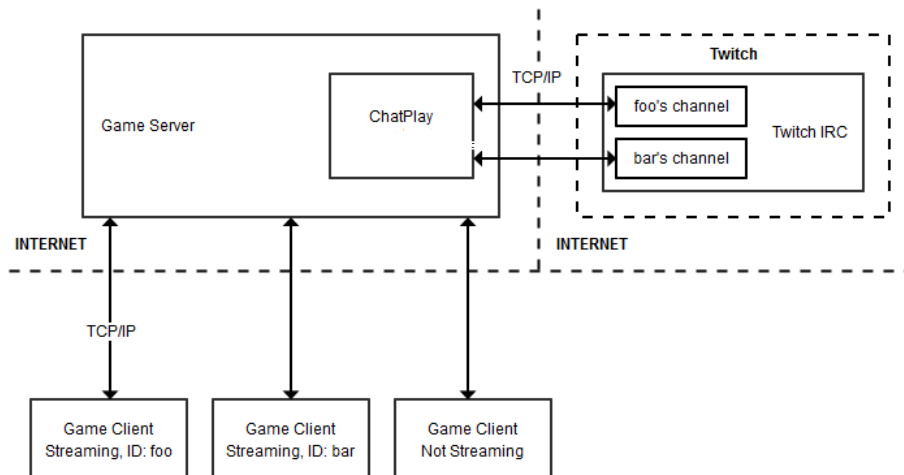
For a tutorial on Twitch ChatPlay, see [Amazon Lumberyard Tutorials](#).

Twitch ChatPlay includes the following components and services:

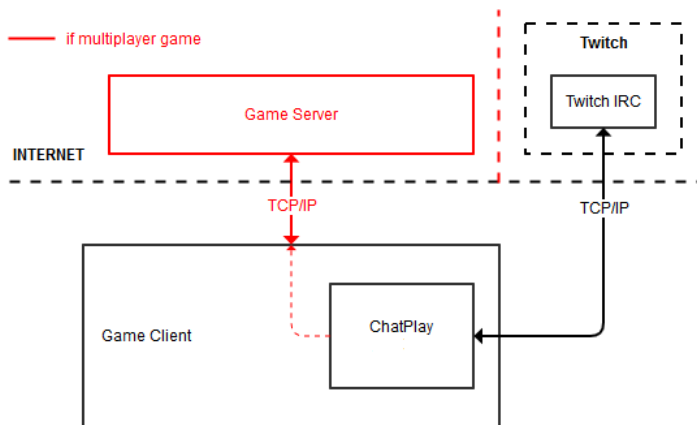
- Twitch IRC servers
- Twitch ID authentication
- Twitch account
- [Twitch Nodes \(p. 730\)](#)

In addition, [Twitch JoinIn \(p. 1135\)](#) enables broadcasting players on Twitch to invite targeted viewers into their game sessions on demand.

The following diagram illustrates Twitch ChatPlay's server-side components.



The following diagram illustrates Twitch ChatPlay's client-side components.



Topics

- [Setting up a Twitch ChatPlay Channel \(p. 1131\)](#)
- [Listening for Twitch Keywords \(p. 1132\)](#)
- [Using Flow Graph with Twitch ChatPlay \(p. 1133\)](#)
- [Twitch ChatPlay Voting \(p. 1133\)](#)
- [Twitch ChatPlay Console Variables \(p. 1133\)](#)
- [Generating and Setting a Twitch Client ID \(p. 1134\)](#)
- [Troubleshooting Twitch ChatPlay \(p. 1135\)](#)
- [Twitch JoinIn \(p. 1135\)](#)
- [Twitch API \(p. 1136\)](#)

Setting up a Twitch ChatPlay Channel

This topic discusses how to set up and connect to a Twitch channel. Go to [Twitch Interactive](#) to set up a new Twitch channel and follow the directions there before starting this procedure.

You need Flow Graph logic to connect to your Twitch channel, listen for keywords, and then act on those keywords.

To create a flow graph for Twitch ChatPlay

1. Open the context (right-click) menu for the object in your level and choose **Create Flow Graph**.
2. In the dialog box, enter a name for the channel and choose **OK**.
3. In the **Flow Graph Editor**, under **Components**, **NodeClass**, **Game**, drag the **Start** node onto the graph.

To connect to a Twitch channel

1. In the **Flow Graph Editor**, under **Components**, **NodeClass**, **Game**, drag the **Start** node onto the graph.
2. Under **Components**, drag the **Twitch:ChatPlay:Channel** node onto the graph.
3. Connect the **output** of the **Game:Start** node to the **Connect** input of the **Twitch:ChatPlay:Channel** node.

To disconnect from a Twitch channel

- To disconnect a single channel, use the **Disconnect** port on the **Twitch:ChatPlay:Channel** node.
- To disconnect from all channels, use the **Twitch:ChatPlay:DisconnectAll** node.

Note

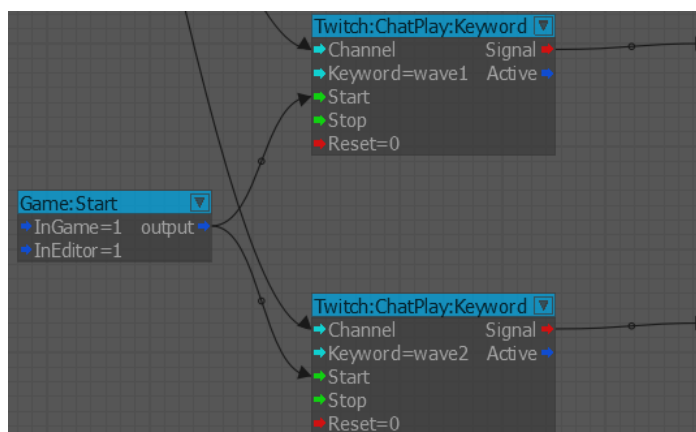
Channels are automatically disconnected when flow nodes are uninitialized. This means that disconnection is automatic in most situations without need for further action.

Listening for Twitch Keywords

This topic discusses how to set up the Flow Graph logic required to listen for keywords from the Twitch chat window.

To listen for keywords

1. In the **Flow Graph Editor**, under **Components**, drag the **Game:Start** node onto the graph.
2. Under **Components**, drag two **Twitch:ChatPlay:Keyword** nodes onto the graph next to the **Game:Start** node.
3. Connect the **output** of the **Game:Start** node to the **Start** inputs of both **Twitch:ChatPlay:Keyword** nodes.

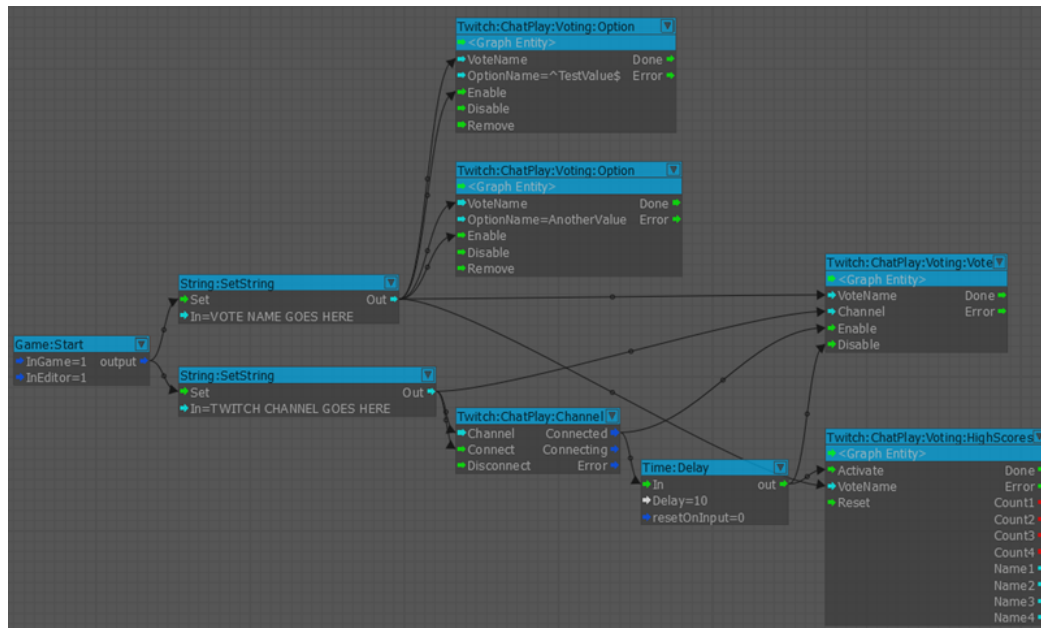


Using Flow Graph with Twitch ChatPlay

There are a number of flow graph nodes you can use to configure Twitch ChatPlay-related settings. For more information, see [Twitch Nodes](#) (p. 730).

Twitch ChatPlay Voting

Twitch ChatPlay voting functionality make it easier to set up polls, surveys, and votes. The following figure shows an example of how Flow Graph voting nodes work together.



For more information about Flow Graph voting nodes for Twitch ChatPlay, see [Twitch Nodes](#) (p. 730).

Twitch ChatPlay Console Variables

Twitch ChatPlay uses the following console variables:

- `chatPlay_clientID` – Client ID for your application. See [Generating and Setting a Twitch Client ID](#) (p. 1134).
- `broadcast_clientID` – Client ID for your application. This value can be the same as the `chatPlay_clientID`. See [Generating and Setting a Twitch Client ID](#) (p. 1134).
- `chatPlay_Server` – Name of the host server. The default is `irc.twitch.tv`.
- `chatPlay_Port` – Port number for the IRC service. The default is port 6667.

Twitch ChatPlay uses the following classes:

- `IChatChannel` – Interface that represents a Twitch ChatPlay channel. Includes keyword callbacks and options for subscribing to the connection state.
- `IChatPlay` – Interface that represents the base system from which you can get handles to Twitch ChatPlay channels.

The Twitch ChatPlay module is implemented as a part of CryAction and is accessible from the `GetChatPlay` method. The main interface is `ChatPlay.h`.

Generating and Setting a Twitch Client ID

In order for Twitch ChatPlay and Twitch API features to function properly, you must set the following console variables to use your application's client ID:

- For Twitch ChatPlay, set `chatPlay_ClientID`
- For Twitch API, set `broadcast_ClientID`

You can use the same value for both console variables.

If you have already registered your application with Twitch, you can locate your client ID on the **Connections** page on the [Twitch website](#). Under **Developer Applications**, click **Edit** under the name of your application.

Generate a Client ID

Generate a client ID by following the instructions below.

To generate the client ID

1. Go to the [Twitch website](#) and log in to your account.
2. In the menu bar, click your user name, **Settings**.
3. On the **Settings** page, click **Connections**.
4. On the **Connections** page, under **Other Connections**, click **Register your application**.
5. Complete the form and click **Register**.
6. Note the generated client ID that you will use to set your console variables.

Set the Client ID

Set the client ID by following the instructions below for your version of Lumberyard.

To set the client ID (Lumberyard 1.6 or later)

1. On your computer, navigate to your project's `game.cfg` file (located in the `\dev\project_name\` directory at the root of your Lumberyard installation).
2. Edit the `game.cfg` file to add the following:

```
chatPlay_clientID = "client ID generated from Twitch"  
broadcast_clientID = "client ID generated from Twitch"
```

To set the client ID (Lumberyard 1.5 or earlier)

1. Modify the `HttpRequestManager.cpp` file (located in the `\dev\Code\CryEngine\CryAction\HttpCaller` directory) to add the following line in the `HttpRequestManager::HandleRequest` function: `HttpRequest->SetHeaderValue("Client-ID", "client ID generated from Twitch");`

It should appear as follows:


```
auto httpRequest = Aws::Http::CreateHttpRequest(uri,
    httpRequestParameters.GetMethod(),
    Aws::Utils::Stream::DefaultResponseStreamFactoryMethod);
httpRequest->SetHeaderValue("Client-ID", "client ID generated from
Twitch");
auto httpResponse = httpClient->MakeRequest(*httpRequest);
```

2. Rebuild the game and engine.

Troubleshooting Twitch ChatPlay

If you run into problems while connecting Twitch ChatPlay to your game, review the following troubleshooting tips for a possible solution.

If your game fails to connect to your Twitch channel, ensure the following:

- You properly entered the name of your Twitch channel into your flow graph.
- You have an active Twitch account set up with the channel name that you're using.
- You have activated the **ChatPlay** node in your flow graph.
- You have an active Internet connection.

If your game fails to connect to your Twitch channel after a successful first attempt, make sure that you have successfully disconnected from your Twitch channel using the **DisconnectAll** node in your flow graph. Failing to do so may result in a successful connection the first time, and then failure to connect afterwards because the first connection was left open.

Twitch JoinIn

Twitch JoinIn enables Twitch broadcasters to invite targeted viewers into their game sessions on demand, using Amazon GameLift session information. Twitch JoinIn provides one flow graph node called `JoinIn::CreateLink` that you can use to create a link that includes all the multiplayer session information necessary for other players to connect to the same session using the generated link. This information is Base64 encoded.

The game must be in a multiplayer session when you create the link. After you create your flow graph logic, you can test the node and your flow graph by exporting the level and launching it from a launcher. To do this in the editor, click **File, Export to Engine**. If you use a launcher such as `SamplesProjectLauncher` or `MultiplayerProjectLauncher`, you must run `mpghost` before attempting to create the link.

Players must have an appropriate launcher that is capable of doing the following:

- Registering with Windows as a URI scheme handler. By default, the URI scheme handler is `game:uri`. You can use the `joinin_uriScheme` console variable to update the scheme in Lumberyard Editor.
- Decoding the Base64 encoded URI and extracting:
 - Game name (if the launcher is designed to launch different games)
 - Launch command (optional)
 - Host address
 - Host port
- Launching the game and connecting to the multiplayer session using the extracted settings.

The JoinIn launcher can be a separate application or be built into the game.

The `Twitch:ChatPlay:Whisper` flow graph node sends information to the viewer client machine. On the viewer client machine, choosing this link decodes the information and launches the game with the appropriate connection settings. For more information about Twitch ChatPlay and Twitch JoinIn flow graph nodes, see [Twitch JoinIn Nodes \(p. 737\)](#).

Twitch API

TwitchAPI is a Twitch-specific implementation of the BroadcastAPI interface that allows developers and designers to make calls to Twitch's REST API from within Lumberyard. For more information, see [Twitch-API](#).

TwitchAPI uses one Flow Graph node. For more information, see [Twitch Nodes \(p. 730\)](#).

To ensure the TwitchAPI feature functions properly, you must set the `broadcast_ClientID` console variable to use the application's client ID (provided by Twitch). For more information, see [Generating and Setting a Twitch Client ID \(p. 1134\)](#).

UI System

You can use the **UI Editor** to create and customize various parts of the user interface, such as images, text, buttons, menus, scroll boxes, and heads-up displays (HUDs). For a tutorial about UI creation, see [Lumberyard Tutorials](#).

Topics

- [Using the UI Editor \(p. 1137\)](#)
- [Working with UI Canvases \(p. 1138\)](#)
- [UI Elements \(p. 1152\)](#)
- [UI Components \(p. 1154\)](#)
- [Implementing New Fonts \(p. 1172\)](#)
- [Using the Animation Editor \(p. 1176\)](#)
- [UI Flow Graph Nodes \(p. 1185\)](#)

Using the UI Editor

You can use the **UI Editor** to create, customize, and animate various user interface elements and components such as menus, buttons, and heads-up displays (HUDs).

The **UI Editor** consists of the following:

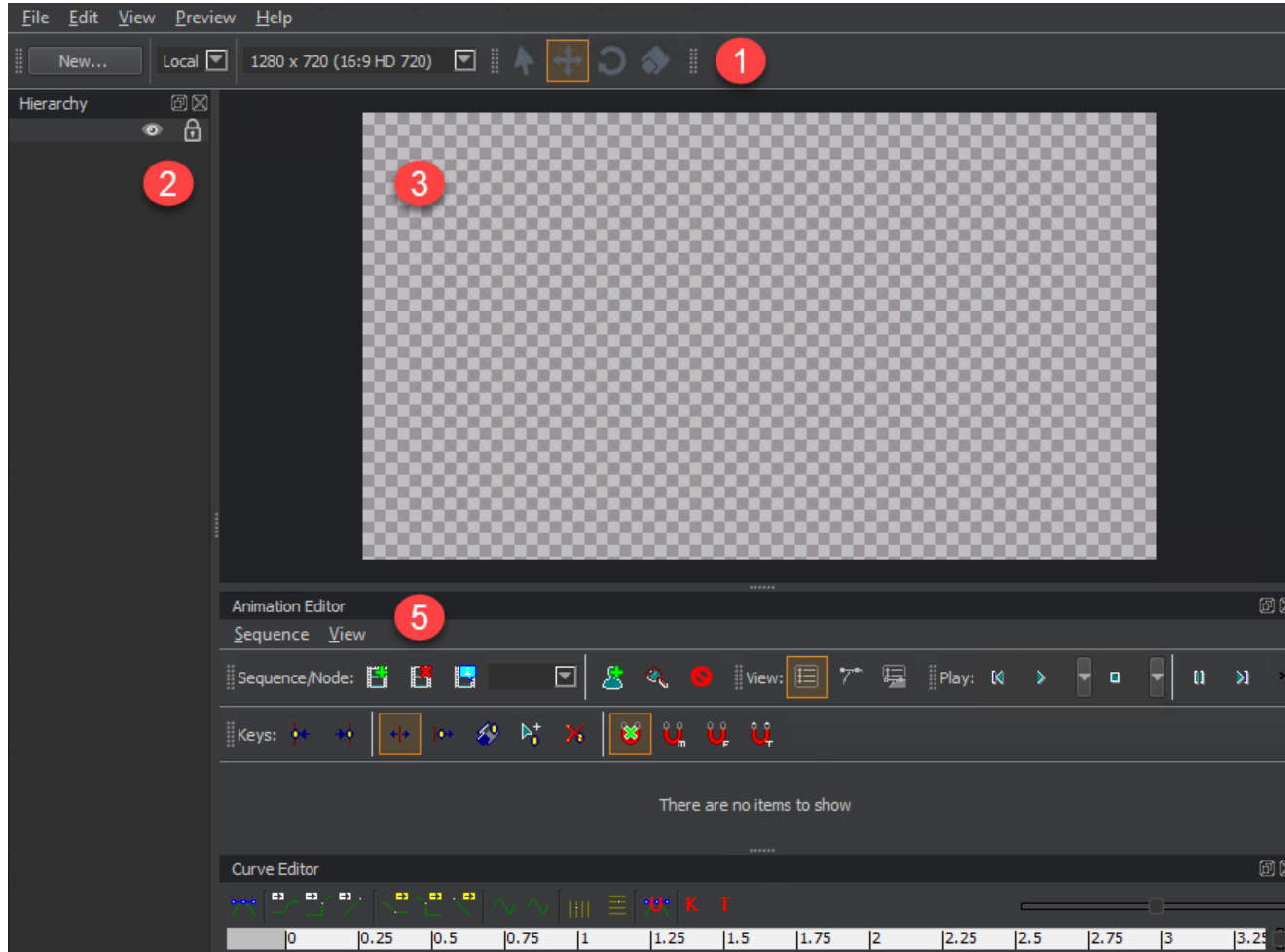
1. **Toolbar** – Commonly used tools and settings
2. **Hierarchy pane** – List of UI elements you create
3. **Viewport** – Display of the UI elements on the current UI canvas
4. **Properties pane** – Component properties for the selected element
5. **Animation Editor** – Tool for animating UI elements

Note

You can tear away and redock the **Hierarchy** pane, **Properties** pane, **Animation Editor**, and sections of the toolbar to customize the **UI Editor**.

To open the UI Editor

- In Lumberyard Editor, select **View, Open View Pane, UI Editor**.



Working with UI Canvases

The **UI Editor** uses the concept of a canvas as an invisible backdrop for your user interface elements. Once you create a canvas, you can add elements such as images, text, and buttons.

To create a UI canvas

1. In Lumberyard Editor, click **View, Open View Pane, UI Editor**.
2. In the UI Editor, add [elements](#) (p. 1152), [components](#) (p. 1154), and [prefabs](#) (p. 1153).
3. Click **File, Save As**. Name the canvas with a `.uicanvas` file extension, and then click **Save**.

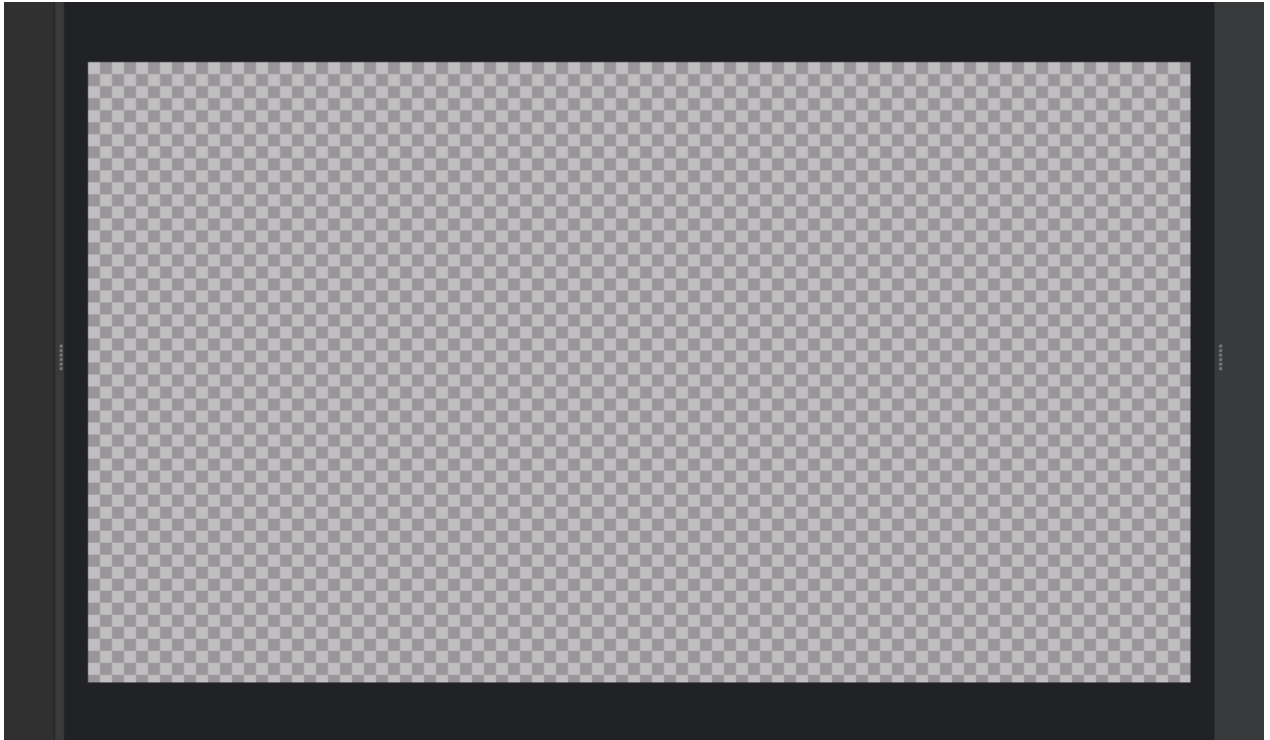
Topics

- [Navigating the Viewport](#) (p. 1139)
- [Changing the Canvas Size](#) (p. 1140)
- [Previewing Canvases](#) (p. 1140)
- [Configuring Canvas Properties](#) (p. 1143)
- [Associating Canvases with UI Flow Graph Nodes](#) (p. 1144)
- [Loading Canvases in the Flow Graph Editor](#) (p. 1145)
- [Loading Canvases in Lua](#) (p. 1147)

- [Placing UI Canvases in the 3D World \(p. 1151\)](#)

Navigating the Viewport

The **UI Editor** features a rectangle with a checkerboard pattern on a dark gray background.



The checkerboard pattern represents empty space within the UI canvas, and the dark gray represents the space outside of the canvas. Anything within the UI canvas space is visible when the canvas is loaded.

To zoom in or out on a UI canvas

Do one of the following:

- Mouse – Scroll the mouse wheel
- Keyboard – Press **Ctrl +** or **Ctrl -**
- Menu – Click **View**, then click **Zoom In** or **Zoom Out**

To pan the view on a UI canvas

- With the mouse on the UI canvas, drag using the middle mouse button
- Press and hold the space bar while dragging the canvas

To toggle common zoom settings

Do one of the following:

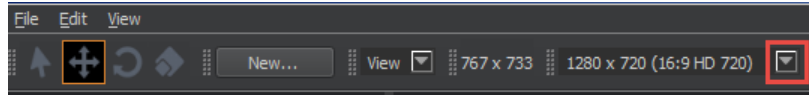
- Fit canvas to current view (default) – Press **Ctrl+0**, or click **View, Fit Canvas**.
- View canvas at actual size – Press **Ctrl+1**, or click **View, Actual Size**.

Changing the Canvas Size

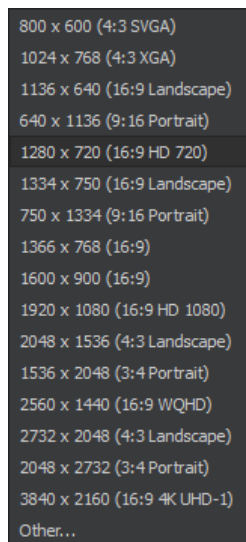
Change your canvas size to visualize how your canvas might look on other displays and devices of varying resolutions. The size at which you save your canvas is the size that is used when you perform the **Scale to Device** action.

To change the canvas size

1. On the toolbar, click the arrow beside the resolution to see a list of commonly used canvas sizes for various platforms.



2. Select the size you want or click **Other** to enter a custom canvas size.



Tip

You can customize the list of canvas sizes that appear in the list by modifying a JSON file stored locally on your machine. In Windows, the canvas size presets file is located in the following directory:

```
C:\Users\<<UserName>\AppData\Local\Amazon\Lumberyard\size_presets.json
```

Previewing Canvases

You can preview your UI canvas to visualize how it might look at different screen resolutions and to see how the interactive elements change states.

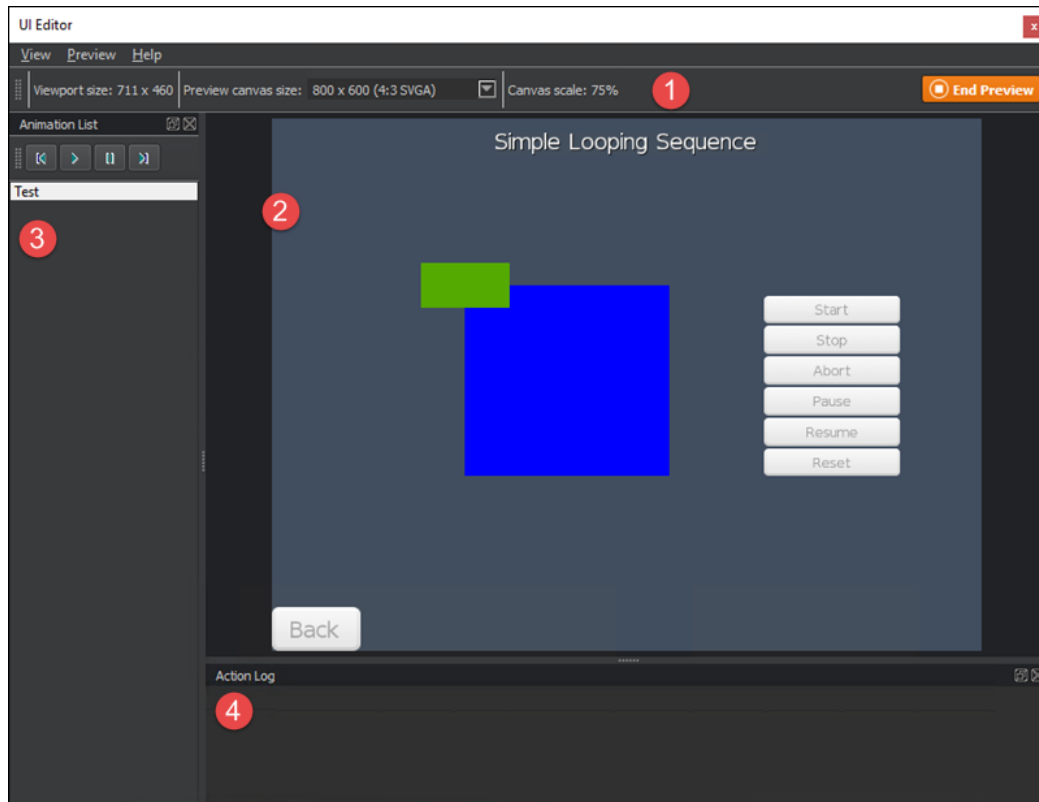
Topics

- [Setting Canvas Size in Preview \(p. 1141\)](#)
- [Previewing Canvas Performance \(p. 1142\)](#)

The **UI Editor Preview** consists of the following:

1. **Toolbar** – Tools to view the current **Viewport size**, **Preview canvas size** (selectable), and **Canvas scale**.

2. **Viewport** – Display of the UI canvas as it would appear at the selected resolution.
3. **Animation List** (p. 1142) – List of the animation sequences in the canvas, which you can control using the playback toolbar. Close this pane to increase the viewport size. Use the **View** menu to restore it.
4. **Action Log** (p. 1142) – Record of actions triggered by the canvas's interactable elements. Close this pane to increase the viewport size. Use the **View** menu to restore it.



To open UI canvas Preview

Do one of the following:

- From the **UI Editor** toolbar, click **Preview**.
- From the **UI Editor** menu, click **Preview, Preview**.
- Press **Ctrl+P**.

To exit the canvas preview, click **End Preview**.

Setting Canvas Size in Preview

Change your canvas size in **UI Editor Preview** to visualize how your canvas might look at different screen resolutions and to see how the interactive elements change state. Changing your canvas size in **Preview** does not affect the canvas size at which you are authoring the UI canvas—that is controlled in the **UI Editor** (p. 1140).

Setting your canvas size in **Preview** is useful when designing games that run on devices that have multiple resolutions. You can see at different resolutions how an element's size and position changes based on the settings of its **Transform2d** (p. 1155) properties, such as anchors, offsets, and the

Scale to Device settings. The **Scale to Device** flag adjusts the size of the element by computing the ratio of the preview canvas size to the reference canvas size and applies that scale to the element. You can see the effects of this computation in the **UI Editor** canvas **Preview**.

The **Canvas scale** in the toolbar shows the scale at which the canvas is displayed. If the **Preview canvas size** selected is larger than the viewport size, the canvas you are previewing is drawn at a reduced scale.

Previewing Canvas Performance

In **UI Editor Preview**, the UI elements in your canvas perform as they would when the game is running.

Try these examples:

- Pause on an interactive element to show its hover state.
- Press (click) an interactive element to show its pressed state.
- Adjust sliders.
- Input and edit text.
- Use keyboard, mouse, or gamepad to interact with the UI.

Note

If the interactive component's **Input enabled** setting is deselected (unchecked), that element is drawn in its disabled state and does not respond to hover or click actions.

Animation List

The **Animation List** pane lists all the UI animation sequences found on the canvas that you are previewing. Select an animation to use the reset, play, pause, and set-to-end controls. Hold **Ctrl** or **Shift** to select and control multiple animations at once. You can also control animations independently and simultaneously so that one may be playing, for example, while you pause another.

Action Log

The **Action Log** pane shows the actions generated by interacting with interactive elements in the UI canvas while in **Preview**. These logged actions help the canvas designer ensure that correct actions are being triggered.

To use this feature, you must type text strings in the **Actions** section of the interactive element's properties.

To enable Action Log entries

1. In the **UI Editor** viewport or **Hierarchy** pane, select the element to which the interactive component is attached.
2. In the **Properties** pane, under the **Actions** category, type a text string for each action for which you want to trigger an action log entry.

The text strings are fully customizable; you can type any string that helps you ensure that the correct actions are being triggered.

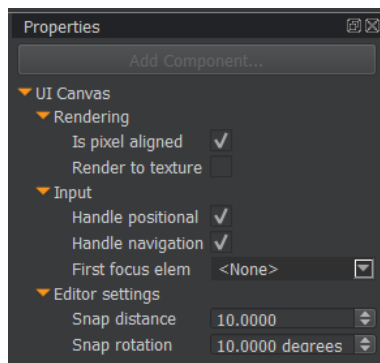
For example, in the picture below, **EnablerChanged** is displayed whenever the **Enable Input** check box changes state (from off to on, or on to off). **EnablerOn** is displayed when the check box is selected, and **EnablerOff** is displayed when it is deselected.



During **Preview**, flow graphs and Lua scripts are not active; actions taken in UI canvas **Preview** have no effect on anything outside of the canvas.

Configuring Canvas Properties

The canvas properties are displayed in the **UI Editor Properties** pane when no elements are selected.



Rendering Properties

The **Is pixel aligned** property, selected by default, makes textures look sharper by rounding to the nearest exact pixel the position of the elements' corners. For example, if, at a particular screen resolution, the position of a corner of an element rectangle is at 123.45, 678.90, then it will be rounded to 123.00, 679.00.

The **Render to texture** property, when selected, causes the UI canvas to be drawn to a texture rather than to the screen. Selecting this property prompts you to type a **Render target name** for the texture. You can type any name, but the convention is to prefix the name with the \$ symbol to distinguish it from texture assets.

Input Properties

The **Handle positional** property, selected by default, causes automatic response to positional input such as mouse movement, mouse button clicks, and touch screen input, as well as keyboard input when an interactive UI element is active (such as an element with a **Text Input** component on it).

You can de-select this property for canvases that don't require input. Another scenario where you may want to de-select this property is if you configure your game to handle all inputs and then pass selected inputs to the UI system.

The **Handle navigation** property, when selected, causes automatic response to navigation input. For example, on a PC, pressing arrow keys will move focus from one interactive UI element to the next, and pressing **Enter** will activate an interactive UI element. We recommend de-selecting this property for canvases placed in the game world.

The **First focus element** property is displayed when **Handle navigation** is selected. **First focus element** specifies which element gains focus when a canvas is first loaded and a mouse is not detected. For more information about element navigation, see [First Focus Element \(p. 1163\)](#).

Editor Properties

The **Snap distance** property controls the distance between positions on the grid when **Snap to grid** is selected in the toolbar.

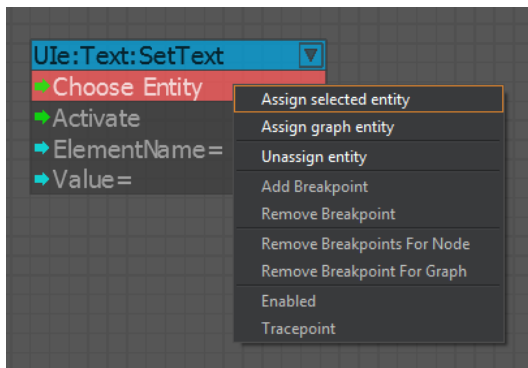
The **Snap rotation** property specifies the number of degrees between each step of rotation when using the rotation gizmo to rotate an element in the viewport when **Snap to grid** is selected in the toolbar.

Associating Canvases with UI Flow Graph Nodes

You must associate all UI flow graph nodes with a UI canvas. There are two sets of flow graph nodes for the UI: **Uie** and **UI**. The **Uie** set of flow graph nodes supersedes the now-legacy **UI** set of flow graph nodes.

In the **Uie** set of nodes, you assign a special entity to the node's **Choose Entity** input using either the new Component Entity system or the legacy Entity system. These procedures are described in this section.

In the legacy **UI** set of nodes, the **CanvasID** comes from the **UI:Canvas:Load** node.



Using the Component Entity system to associate a UI canvas with a Uie flow graph node

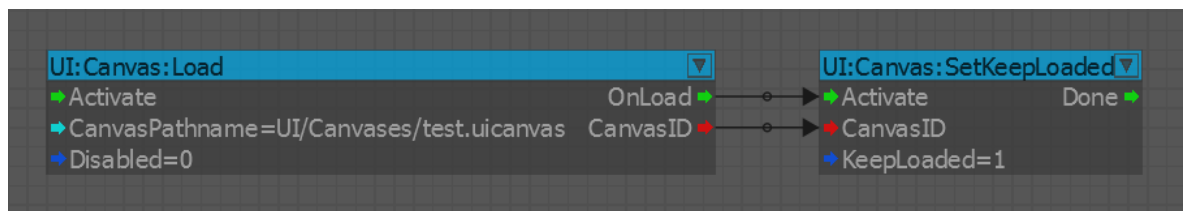
1. In the viewport, right-click and select **Create Component Entity**.
2. Right-click the newly created component and click **Flow Graph, Add**.
Enter a name for the flow graph, or leave it as `Default`.
3. If the **Flow Graph Editor** is not yet open, click **View, Open View Pane, Flow Graph**.
4. In the viewport, right-click the component entity and select **Flow Graph, Open, <flow graph name>**.
5. Select the newly created component entity. In the **Entity Inspector**, add a **UI Canvas Asset Ref** component and enter a path to the canvas you want to associate.
6. In the **Flow Graph Editor**, in the flow graph you created, add any **Uie** flow graph node to the graph.
7. Right-click the node you placed and do one of the following:
 - Click **Assign graph entity** if the canvas you want to reference is selected in the **UI Canvas Asset Ref** component.
 - Select a different entity and then click **Assign selected entity** to reference a different canvas.

Note

This other entity can be either a component entity with the UI Canvas Asset Ref component on it or a legacy entity that is a UiCanvasRefEntity.

Using the Legacy Entity system to associate a UI canvas with a Ule flow graph node

1. In Lumberyard Editor's **Rollup Bar**, on the **Objects** tab, click **Entity**. Expand the **UI** folder and drag **UiCanvasRefEntity** into the viewport.
2. Select the newly created **UiCanvasRefEntity** entity. In its **Entity Properties**, click **CanvasPath** and enter a path to the canvas you want to associate.
3. In Lumberyard Editor, click **View, Open View Pane, Flow Graph**.
4. In the **Flow Graph** Editor's **Graphs** pane, select a flow graph.
5. Add any **Ule** flow graph node to the graph.
6. Right-click the node you placed and do one of the following:
 - Click **Assign graph entity** if the flow graph is associated with the **UiCanvasRefEntity**.
 - If the flow graph you used is not associated with the **UiCanvasRefEntity**, make sure the **UiCanvasRefEntity** is selected in your viewport and then click **Assign selected entity**.



To associate a UI canvas with a legacy UI flow graph node

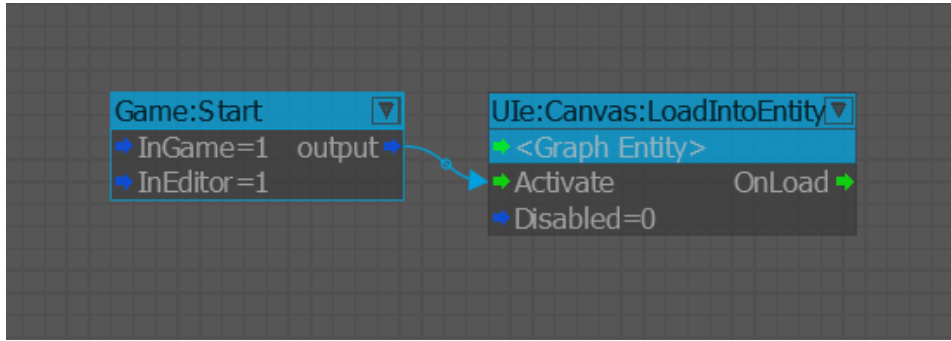
1. Load a canvas in the **Flow Graph** Editor. See [Loading Canvases in the Flow Graph Editor \(p. 1145\)](#) for more information.
2. Add any **UI** flow graph node to the graph.
3. Connect the **CanvasID** output of the **UI:Canvas:Load** node to the **CanvasID** input of the new node.

Loading Canvases in the Flow Graph Editor

You can use the **Flow Graph Editor** to load and unload UI canvases. For more information about using flow graphs, see [Flow Graph System \(p. 487\)](#).

For more information about the flow graph nodes you can use to make elements and components respond to user input, see [UI Flow Graph Nodes \(p. 1185\)](#).

You can load canvases in the **Flow Graph Editor** using either the **Ule** node set (recommended) or the legacy **UI** node set. You can also load a canvas automatically using a component entity (without using flow graph). These procedures are described in this section.



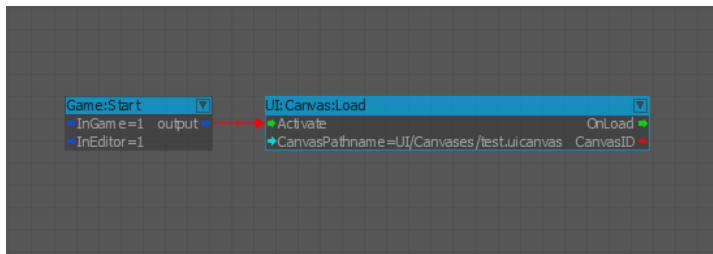
Use the following procedure to load canvases using the **UIe** node set. This is the recommended method of loading canvases in flow graph.

To load a canvas in the Flow Graph Editor using the UIe node set

1. In Lumberyard Editor, click **View, Open View Pane, Flow Graph**.
2. In the **Flow Graph Editor**, select a flow graph from the **Graphs** pane.
3. Right-click anywhere in the graphs pane and select **Add Node, Game, Start**.
4. Right-click anywhere in the graphs pane and select **Add Node, UIe, Canvas, LoadIntoEntity**.
5. Right-click the **UIe:Canvas:LoadIntoEntity** node and select **Assign selected entity** or **Assign graph entity** to assign a UI canvas reference entity to the node.

For more information about assigning a UI canvas reference entity to the node, see [Associating Canvases with UI Flow Graph Nodes \(p. 1144\)](#).

6. Connect the **Game:Start** node output to the **Activate** input on the **UIe:Canvas:LoadIntoEntity** node.



To use the legacy **UI** node set to load canvases, use the following procedure.

To load a canvas in the Flow Graph Editor using the legacy UI node set

1. In Lumberyard Editor, click **View, Open View Pane, Flow Graph**.
2. In the **Flow Graph Editor**, select a flow graph from the **Graphs** pane.
3. Right-click anywhere in the graphs pane and select **Add Node, Game, Start**.
4. Right-click anywhere in the graphs pane and select **Add Node, UI, Canvas, Load**.
5. Connect the **Game:Start** node output to the **Activate** input on the **UI:Canvas:Load** node.
6. Double-click **CanvasPathname** in the **UI:Canvas:Load** node, and type a path in the **CanvasPathname** text box or use the file browser to navigate to the path. The path is relative to the project folder.

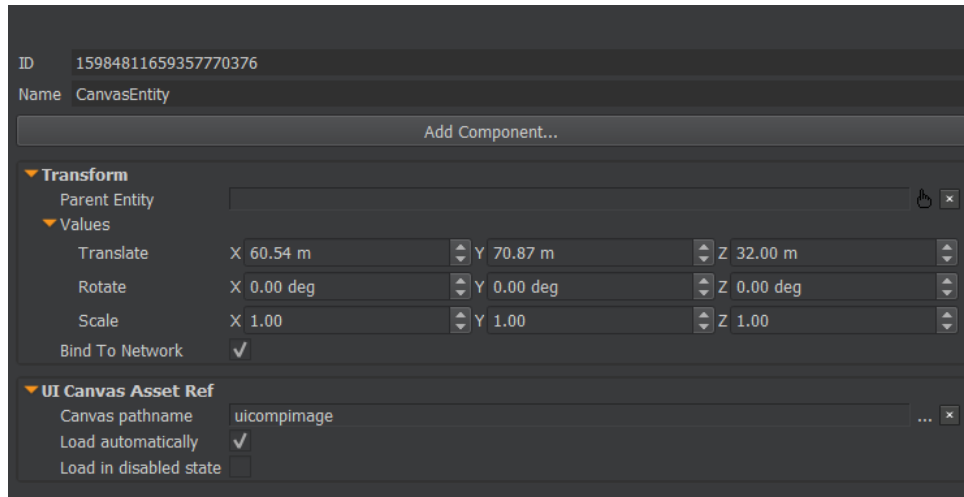
Note

You might need to zoom in to be able to edit **CanvasPathname**.

The following method uses the component entity system to load a canvas, without using any flow graphs.

Using the Component Entity system to load a UI canvas

1. In the level, create a [component entity](#) (p. 410)
2. In the **Entity Inspector**, [add to this component entity](#) (p. 410) a **UI Canvas Asset Ref** to specify the UI canvas and optionally to automatically load it when the level loads.
3. Select the **Load automatically** check box.



The canvas is automatically loaded when the level loads. It can be referenced from any of the **UIe** flow graph nodes, as long as they are in a flow graph that belongs to any component entity.

Loading Canvases in Lua

You can use the Lua scripting language to load and unload UI canvases.

Topics

- [UI Lua Reference](#) (p. 1148)

For general information about using Lua, see [Lua Script Component](#) (p. 365).

To load a canvas in Lua

1. Create a new, plain text file in your game project directory with a `.lua` file extension.
2. Type or paste the following sample script into your new Lua file:

Note

The following script uses a Lua file named `loadcanvas.lua` and loads a canvas file named `menu.uicanvas` saved at the root of the game project directory. Substitute the appropriate file names for your script.

```
loadcanvas =  
{  
  Properties =  
  {  
  },  
}
```

```
}  
  
function loadcanvas:OnActivate()  
    self.uiCanvasLuaProxy = UiCanvasLuaProxy();  
    self.uiCanvasLuaProxy:LoadCanvas("menu.uicanvas");  
end
```

3. In Lumberyard Editor, right-click in the **Viewport** and click **Create Component Entity**.
4. If the **Entity Inspector** does not open automatically, click **View, Open View Pane, Entity Inspector**.
5. Click **Add Component**.
6. Select **Scripting, Lua Script**.
7. Under **Lua Script**, click ... and open the Lua script file that you created.
8. In Lumberyard Editor, click **Game, Switch to Game** to enter game mode. Verify that your canvas file loads.

UI Lua Reference

You can use the following Lua scripting functions when loading and unloading canvases with Lua in Lumberyard Editor.

LyShineLua.ShowMouseCursor

Toggles the visibility of the mouse cursor.

Parameters

visible

Displays 1 if the mouse cursor is displayed or 0 if it is hidden

Returns

None.

UiCanvasLuaProxy:LoadCanvas

Loads the canvas file that you specify and immediately starts rendering it.

Parameters

canvasFilename

The path to the *.uicanvas file to load.

Type: String

Returns

AZ::EntityId for the loaded canvas entity.

UiCanvasLuaProxy:BusConnect

Connects the given canvas entity to the UiCanvasLuaBus.

Parameters

entityId

AZ::EntityId. A canvas entity identifier.

Returns

None.

UiCanvasLuaBusSender:FindElementById

Returns the AZ::EntityId for the given canvas element identifier.

Parameters

id

Represents the identifier of an element stored within the canvas.

Type: Unsigned integer

Returns

AZ::EntityId of the given element.

UiCanvasNotificationLuaProxy:BusConnect

Connects to the given canvas entity's UiCanvasNotificationBus.

Parameters

id

AZ::EntityId. A canvas entity identifier.

Returns

None.

UiCanvasNotificationLuaBus:OnAction

User defined in script. Called when the canvas broadcasts an action name.

Actions are broadcasted by the canvas when they have been configured with an action name. For example, a button can be configured to broadcast an action name when clicked.

Parameters

entityId

AZ::EntityId. The entity identifier that triggered the action.

actionName

The action name that was broadcasted.

Type: String

Returns

None.

UiFaderComponent:HasFaderHandler

Returns true if the given entity identifier has a fader component. Otherwise, false.

Parameters

entityId

AZ::EntityId. The entity identifier to check the fader component for.

Returns

Returns true if the given entity identifier has a fader component. Otherwise, false.

UiFaderBusSender:SetFadeValue

Sets the starting alpha value from which to begin the fade.

Parameters

fade

Alpha value from which to begin the fade.

Type: Float

Returns

None.

UiFaderBusSender:Fade

Executes the fade effect.

Parameters

targetValue

The value at which the fade ends.

Type: Float

Values: 0.0 to 1.0

speed

Speed for the effect to complete. The higher the value, the faster the fade effect takes place., except for 0, which is a special exception that causes instant execution of the fade effect.

For example, 1 would take one second to fade from off to on, 2 takes half that time (twice as fast).

0.5, for example, is half the speed of 1.

Type: Float

Returns

None.

Placing UI Canvases in the 3D World

You can place a UI canvas directly on an object in the 3D world, as opposed to showing it in screen space. To do this, you render a UI canvas to a texture, and then use that texture in a material on a 3D mesh.

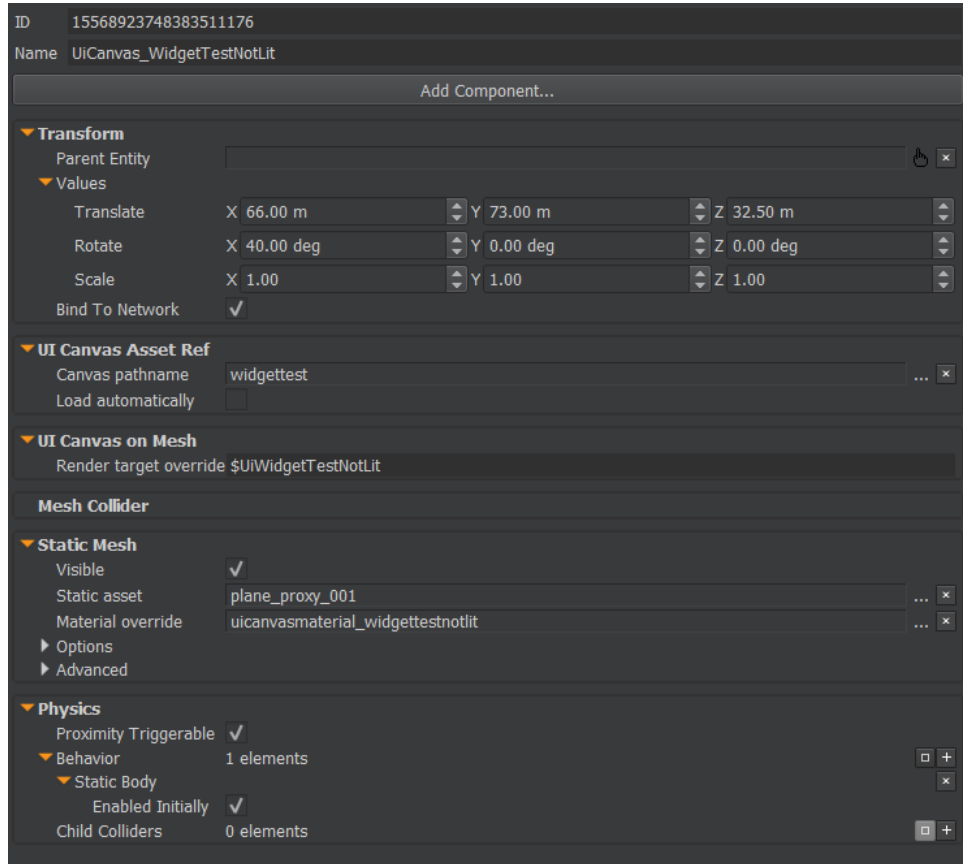
You can use any material on any type of entity to display a texture rendered by a UI canvas. However, if players are to interact with the UI canvas in the 3D world—by clicking with the mouse, for example—you must use a component entity.

To see an example of a UI canvas on an object in a 3D world, open the **Uiln3DWorld** level in the [FeatureTests Project](#) (p. 1108).

Follow all the steps in the following procedure if you need to create a canvas that players can interact with. If the canvas is not to be interactive, then you only need steps 1 through 5.

To place a UI canvas on an object in the 3D world

1. [Create your UI canvas file](#) (p. 1138). In the [canvas properties](#) (p. 1143), select **Render to texture** and type a name in the **Render target** text box.
2. In the level, create a [component entity](#) (p. 410).
3. In the **Entity Inspector**, [add to this component entity](#) (p. 410) a **UI Canvas Asset Ref** to specify the UI canvas and optionally to load it automatically when the level loads. See [Loading Canvases in the Flow Graph Editor](#) (p. 1145) for other ways to load the canvas.
4. In the **Material Editor** (p. 1034), create a material that uses the render target texture that is rendered by your canvas.
5. Add a [Static Mesh](#) (p. 402) component to the component entity and choose the mesh asset onto which you want to map your canvas. Use the **Material override property** to select the material that you created.
6. Add a [Physics component](#) (p. 384) and a [Mesh Collider component](#) (p. 386). In the physics component properties, click the + icon next to the **Behavior** property and add a **Static Body** behavior. Physics is required on this entity because a ray cast is used to translate a mouse or touch input into a position on the UI canvas that is at that point in the world.
7. Add a **UI Canvas on Mesh** component. Type a canvas name in the **Render target override** property if you want to load several instances of the UI canvas on different meshes and have them display different states. Otherwise, leave this property blank.



UI Elements

UI elements are entities to which you can attach multiple components. You can start with an empty element and add components to it, such as a button, image, slider, text, and so on. Or you can [add an existing pre-fabricated \(prefab\) element \(p. 1153\)](#), such as a scrollbox, which is an element with components already attached. You can also [create your own prefab elements \(p. 1153\)](#).

Every UI element has a required component called **Transform2D**. The **Transform2D** component defines the positioning, spacing, and size of the element relative to its parent (whether its parent is the canvas or another element). Each UI element can also have one [visual component \(p. 1158\)](#) (image or text), one [interactive component \(p. 1160\)](#) (button, check box, scrollbox, slider, or text input), and one [layout component \(p. 1170\)](#) (layout column, layout row, or layout grid). The [remaining components \(p. 1171\)](#) are the mask and fader, of which UI elements can attach either or both.

For each of the following procedures, use the **UI Editor** to manage UI elements.

Managing UI Elements in the UI Editor

Task	Steps
To create an element	In the UI Editor toolbar, click New, Empty element . The element appears in the Hierarchy pane and viewport.
To move, rotate, or resize an element	Select the element, then click the Move, Rotate, or Resize tool in the toolbar.

Task	Steps
	Select Snap to grid to modify elements in increments.
To copy an element	Right-click the element in the Hierarchy pane or viewport and click Copy .
To nudge an element	To nudge, or move, an element one pixel at a time, select the element and click the Move tool. Use arrow keys to nudge elements in the selected direction. Press and hold the Shift key while pressing the arrow keys to nudge elements 10 pixels at a time.
To paste a copied element	Right-click anywhere in the Hierarchy pane or viewport and click Paste . If an element is selected, the Paste as sibling and Paste as child options appear.
To delete an element	Right-click the element in the Hierarchy pane or viewport and click Delete .
To hide an element	Click the eye icon (to the right of the element name) in the Hierarchy pane or viewport. Click again to unhide the element.
To hide all elements	To hide all elements, deselect any currently selected items and then click the eye icon in the topmost row of the Hierarchy pane.
To prevent selection of an element in the viewport	Click the padlock icon to the right of the element name in the Hierarchy pane. This prevents selection only of that particular element; its children are still selectable.
To prevent selection of all elements in the viewport	Deselect any currently selected elements (click in a blank area of the Hierarchy pane) and then click the padlock icon in the topmost row of the Hierarchy pane.
To rename an element	Double-click the element in the Hierarchy pane, type the new name, and press Enter .
To nest an element	Select the element in the Hierarchy pane and drag it on top of the parent element.
To change the element draw order	Select and drag elements up or down in the Hierarchy pane. Elements are drawn in order starting from the top of the hierarchy list, so elements at the bottom of the list are displayed in front of elements at the top of the list.

Configuring UI Anchors and Offsets

Each UI element's position is determined by the **Transform2D** component. The **Transform2D** component sets a UI element's position and size relative to its parent's edges. The parent may be another element (if the elements are nested), or the canvas.

For more information about the **Transform2D** component, see [Transform2D – Managing UI Anchors and Offsets \(p. 1155\)](#).

Using and Creating UI Prefabs

In the **UI Editor**, prefabs are preconfigured UI elements and compound elements that you can add to a canvas. You can also create custom prefabs.

To add a prefab element

1. In the **UI Editor** toolbar, click **New, Element from prefab**.
2. Select from:
 - **Button**
 - **Checkbox**
 - **Image**
 - **LayoutColumn**
 - **LayoutGrid**
 - **LayoutRow**
 - **ScrollBox**
 - **ScrollBarHorizontal**
 - **ScrollBarVertical**
 - **Slider**
 - **Text**
 - **TextInput**

The new element appears in the **Hierarchy** pane and viewport.

If you have created your own element or modified an existing prefab, you can save it as a custom prefab.

To save a custom prefab element

1. In the **UI Editor**, right-click an element that you have created or modified in the **Hierarchy** pane or viewport.
2. Click **Save as Prefab**.
3. In the **Save As** dialog box, do the following:
 1. Navigate to any location in the project folder where you want to save your prefab.
 2. Name your prefab with a `.uiprefab` file extension.
 3. Click **Save**.

Your prefab now appears in the **New..., Element from prefab...** menu.

UI Components

UI components define the properties of a UI element. For example, every element has a **Transform2D** component that defines its position, rotation, size, and scale. You can give an element additional properties by adding components, such as adding the image component to give an element color or texture. Each UI element can have one visual component (image or text), one interactive component (button, check box, scrollbox, slider, or text input), and one layout component (layout column, layout row, or layout grid). The remaining components are the mask and the fader components, of which UI elements can attach either or both.

To view some samples of completed UI canvases that demonstrate the following components, open the [FeatureTests Project \(p. 1108\)](#). In the **UI Editor**, click **File, Open Canvas**. Then select the appropriate canvas to view the completed UI canvas as stated in the following topics.

You can also see the completed UI canvases in action by switching to game mode (press **Ctrl+G** or from the main menu, **Game, Switch to Game**) in the [FeatureTests Project](#) (p. 1108).

Topics

- [Adding or Deleting Components](#) (p. 1155)
- [Transform2D – Managing UI Anchors and Offsets](#) (p. 1155)
- [Visual Components](#) (p. 1158)
- [Interactive Components](#) (p. 1160)
- [Layout Components](#) (p. 1170)
- [Other Components](#) (p. 1171)

Adding or Deleting Components

You can easily add or delete components in the [UI Editor](#) (p. 1137).

To add a component to an element

1. In the **UI Editor**, select an element in the **Hierarchy** pane and click **Add Component** at the top of the **Properties** pane.
2. Select the component (image, text, button, checkbox, slider, text input, scrollbox, fader, mask, layout column, layout row, or layout grid) that you want to add to the element.
3. Use the instructions for the specific component you are adding in the next section.

To delete a component from an element

- In the **UI Editor**, select an element in the **Hierarchy** pane. Right-click the component in the **Properties** pane and click **Remove**.

Transform2D – Managing UI Anchors and Offsets

You can use anchors and offset settings in the **Transform2D** component to set a UI element's position and size relative to its parent's edges. The **Transform2D** component is a required component in every element.

Anchor values are always 0.00% to 100.00% as defined by the parent's edges. Offsets are expressed in pixels and are relative to the anchors.

Anchors and offsets are useful in a variety of situations:

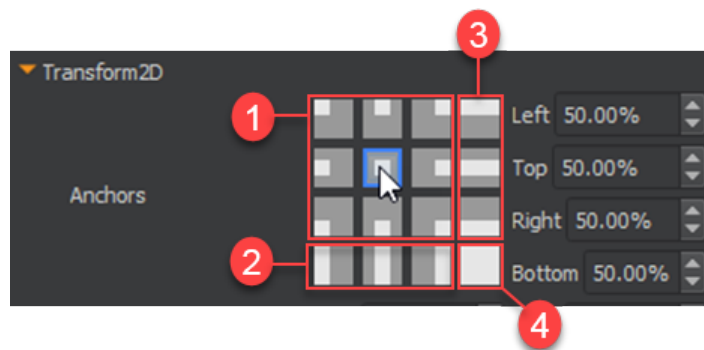
- Ensuring an element maintains a specific padding within its parent's edges, regardless of changes to the parent's size
- Anchoring an element to a corner of its parent, regardless of changes to the parent's size or position
- Building resolution-independent UI elements

For example, you can ensure an element remains full screen regardless of the screen's resolution.

To configure an element's anchors

1. In the **Hierarchy** pane of the **UI Editor** (p. 1137), select the element whose anchors you want to modify.
2. In the **Properties** pane, under **Transform2D**, choose from the selection of commonly used anchor placements.

1. Anchor to the parent's center, corner, or midway along an edge without changing size.
2. Anchor to the left edge, middle, or right edge; vertical size adjusts to parent.
3. Anchor to the top edge, middle, or bottom edge; horizontal size adjusts to parent.
4. Anchor all of the element's edges to the parent; horizontal and vertical size adjusts to parent. You can use this anchor preset to place an element that remains full screen, regardless of a change in resolution (if the canvas is its parent).



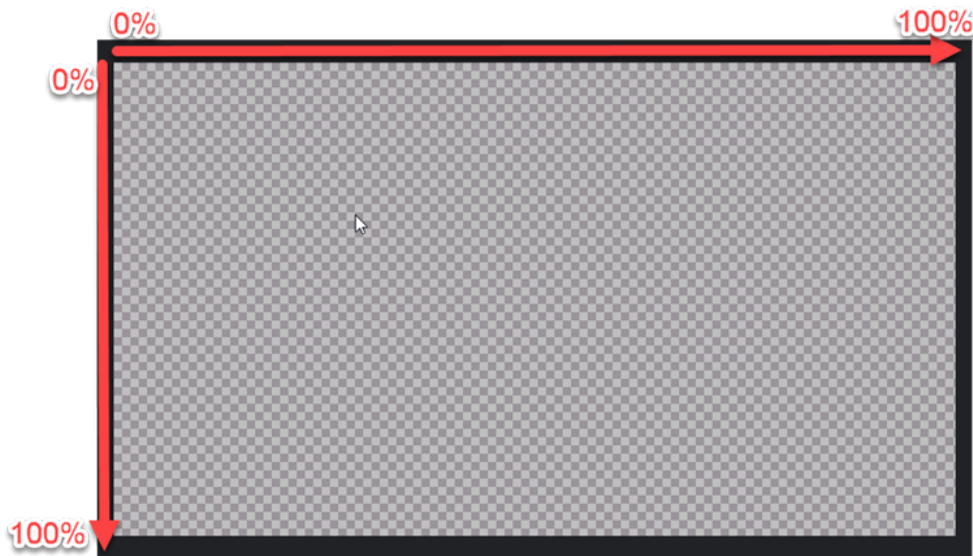
To further edit (fine-tune) an element's anchors

In the **Properties** pane, under **Transform2D**, do the following for **Anchors**, as appropriate:

- For **Left**, enter a value between 0.00% and 100.00%.
- For **Right**, enter a value between 0.00% and 100.00%.
- For **Top**, enter a value between 0.00% and 100.00%.
- For **Bottom**, enter a value between 0.00% and 100.00%.

The anchors' positions can be visualized as points on a grid, plotted in percentages by the length of its parent's edges from left to right and top to bottom. If you want to keep the element's size absolute (so that it doesn't change size when the parent changes size) but want to anchor it a particular vertical or horizontal point relative to the parent's size, make sure the top and bottom (or left and right) anchors have the same number. In this case, the anchors are said to be together.

But if, for example, you want the element's left and right edges to each remain at a fixed percentage relative to its parent and to change size as its parent changes size, then make the numbers different. In this case, the anchors are called split.



To edit an element's position and size

In the **Properties** pane, under **Transform2D**, modify the **Offsets**, as appropriate:

If the element's anchors are together, do the following:

- For **X Pos**, enter a negative or positive value in pixels. This adjusts the horizontal offset relative to the left-right anchor position.
- For **Y Pos**, enter a negative or positive value in pixels. This adjusts the vertical offset relative to the top-bottom anchor position.

When the element's anchors are together, only its position (and not its size) adjusts with the parent's size. Therefore, you can manually adjust its size, which remains consistent when anchors are together:

- For **Width**, enter a value in pixels.
- For **Height**, enter a value in pixels.

If the element's anchors are split, do the following:

- For **Left**, enter a negative or positive value in pixels. This adjusts the size offset relative to the element's left anchor.
- For **Right**, enter a negative or positive value in pixels. This adjusts the size offset relative to the element's right anchor.
- For **Top**, enter a negative or positive value in pixels. This adjusts the size offset relative to the element's top anchor.
- For **Bottom**, enter a negative or positive value in pixels. This adjusts the size offset relative to the element's bottom anchor.

To edit an element's pivot, rotation, and scale

In the **Properties** pane, under **Transform2D**, do the following for **Pivot**, **Rotation**, and **Scale**, as appropriate:

- For **Pivot**, select a pivot preset or enter values for X and Y where 0 and 1 represent the element's edges.
- For **Rotation**, enter a value in degrees.
- For **X Scale**, enter a value to use as a multiplier for the element's width.

- For **Y Scale**, enter a value to use as a multiplier for the element's height.
- Select **Scale to Device** if you want the UI element and its children to scale with the device resolution.

Note

The element rotates around, resizes from, and calculates position from its pivot point. The pivot point is not limited by the element's borders; you can place the pivot outside of the element.

Visual Components

You can add one visual component to an element: **Image** or **Text**.

Image

You can use an image component to add a color tint or texture to an element. To see an example of a completed canvas with the image component, open the `UiCompImage.uicanvas` file in the [FeatureTests Project \(p. 1108\)](#).

To edit an image component

In the **Properties** pane of the **UI Editor (p. 1137)**, expand **Image** and do the following, as appropriate:

SpriteType

Select one of the following:

- **Sprite/Texture asset** – Image displays the asset specified in **Sprite path**.
- **Render target** – Image displays the render target specified in **Render target name**.

Sprite path

Click the ... (folder) icon and select a suitable file.

Click the gear icon next to the **Sprite path** folder icon to open the sprite **Border Editor**. Then define the borders for the sliced image type.

Render target name

Type a name of a render target and press **Enter**.

Color

Click the color swatch to select a different color.

Alpha

Use the slider to choose an alpha value between 0 and 1.

Image type

Select one of the following:

- **Stretched** – Stretches the texture with the element without maintaining aspect ratio
- **Sliced** – Treats the texture as a 9-sliced sprite
- **Fixed** – Makes the texture pixel perfect
- **Tiled** – Tiles the texture to fill the element
- **Stretched to Fit** – Scales to fit while maintaining aspect ratio
- **Stretched to Fill** – Scales to fill while maintaining aspect ratio

Blend Mode

Select one of the following:

- **Normal** – Uses alpha to interpolate colors between elements
- **Add** – Blends colors between elements by adding (lightening) color values together

- **Screen** – Blends colors using inverse source color resulting in a lighter color
- **Darken** – Chooses the darker color channel when blending between elements
- **Lighten** – Chooses the lighter color channel value when blending between elements

Text

You can use a text component to add a text string to an element. To see an example of a completed canvas with the text component, open the `UiCompText.uicanvas` file in the [FeatureTests Project](#) (p. 1108).

To edit a text component

In the **UI Editor Properties** pane, expand **Text** and do the following, as appropriate:

Text

Type the desired text string and press **Enter**. Here, you can apply [text styling markup](#) (p. 1159).

Color

Click the color swatch to select a different color.

Alpha

Use the slider to choose an alpha value between 0 and 1.

Font path

Click the button and select a font `.xml` file.

Font size

Type a font size and press **Enter**.

Font effect

Select an effect from the list. The available font effects are dictated by the font `.xml` file.

Horizontal text alignment

Select **Left**, **Center**, or **Right** to align the text with respect to the element's left and right borders.

Vertical text alignment

Select **Top**, **Center**, or **Bottom** to align the text with respect to the element's top and bottom borders.

Overflow mode

Select **Overflow** to allow the text to display beyond the edges of the element.

Select **Clip text** to hide, or clip, any text that flows beyond the element's edges.

Wrap text

Select **No wrap** to prevent text from wrapping to subsequent lines.

Select **Wrap text** to allow text to be broken into separate lines.

Text Styling Markup

You can customize the appearance of the text in your game UI by using bold and italic styling, multiple text colors, and multiple fonts in a single text string. You enter specific tags directly into the **Font** box, along with your string. The simple markup language used is loosely based on HTML.

To use the text styling markup feature, you must use a font family `*.fontfamily` asset file in the **Font path** setting (rather than an individual `.xml` asset file). For more information about adding font families to your projects, see [Implementing New Fonts](#) (p. 1172).

To use text styling markup

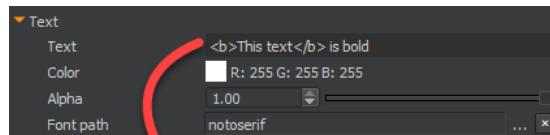
1. In the **UI Editor**, add a text component to an element on your canvas (or modify an existing component).

2. With the element selected, in the **Properties** pane, set the **Font path** property to a *.fontfamily file.
3. Enter a string with markup styling in the **Text** box. See the next section for examples.

Tags and Attributes

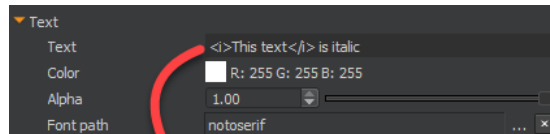
You can use the following tags and attributes when styling text with markup:

Bold tag:



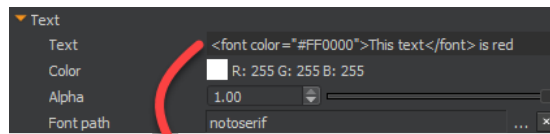
This text is bold

Italic tag: <i>



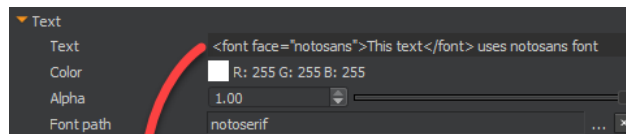
This text is italic

Font color tag:



This text is red

Font face tag:



This text uses notosans font

Interactive Components

Interactive components respond to user input. For example, the user can click a button or drag a slider. You can use [Lua scripts](#) (p. 1147) or [flow graphs](#) (p. 1185) to link the component response to an action.

An interactive element is defined as an element that has an interactive component on it.

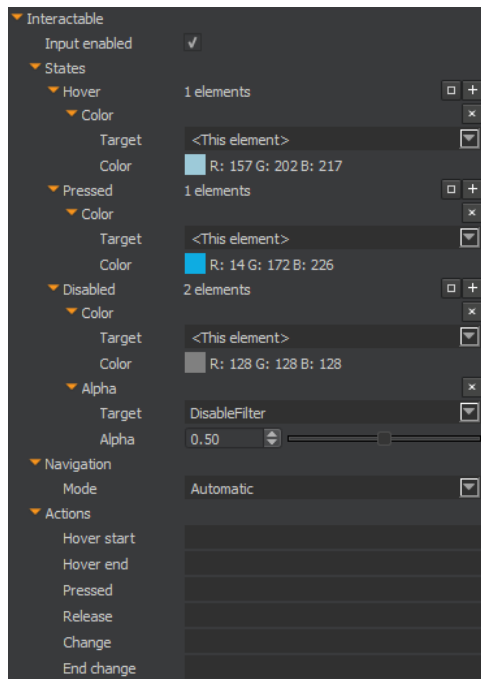
Topics

- [Properties](#) (p. 1161)
- [Button](#) (p. 1164)
- [Check box](#) (p. 1165)
- [Slider](#) (p. 1165)
- [Text Input](#) (p. 1166)
- [ScrollBar](#) (p. 1167)
- [ScrollBar](#) (p. 1169)

Properties

All of the interactive components share a common set of properties. These properties are grouped into the following categories:

- **Input Enabled** (p. 1161) – Check box or flag that determines whether the element can be interacted with.
- **States** (p. 1162) – Settings that determine the appearance of the element when in the **Hover**, **Pressed**, or **Disabled** states.
- **Navigation** (p. 1163) – Settings that determine how the gamepad or arrow keys [navigate between interactive elements](#) (p. 1163).
- **Actions** – Events that are caused by the listed action.



Input Enabled

The **Input Enabled** setting, selected by default, determines whether the component can be interacted with.

To visualize how the interactive element looks in its disabled state, deselect the **Input Enabled** setting, and then use [Preview mode](#) (p. 1140) to preview your canvas.

You can also manipulate the **Input Enabled** flag from C++ or flow graph to enable or disable interactive components while the game is running. Use the flow graph node [UI:Interactable:SetIsHandlingEvents](#) (p. 1256).

States

The **States** group of properties defines the appearance of the interactive element and its child UI elements when the element is in the **Hover**, **Pressed**, and **Disabled** states.

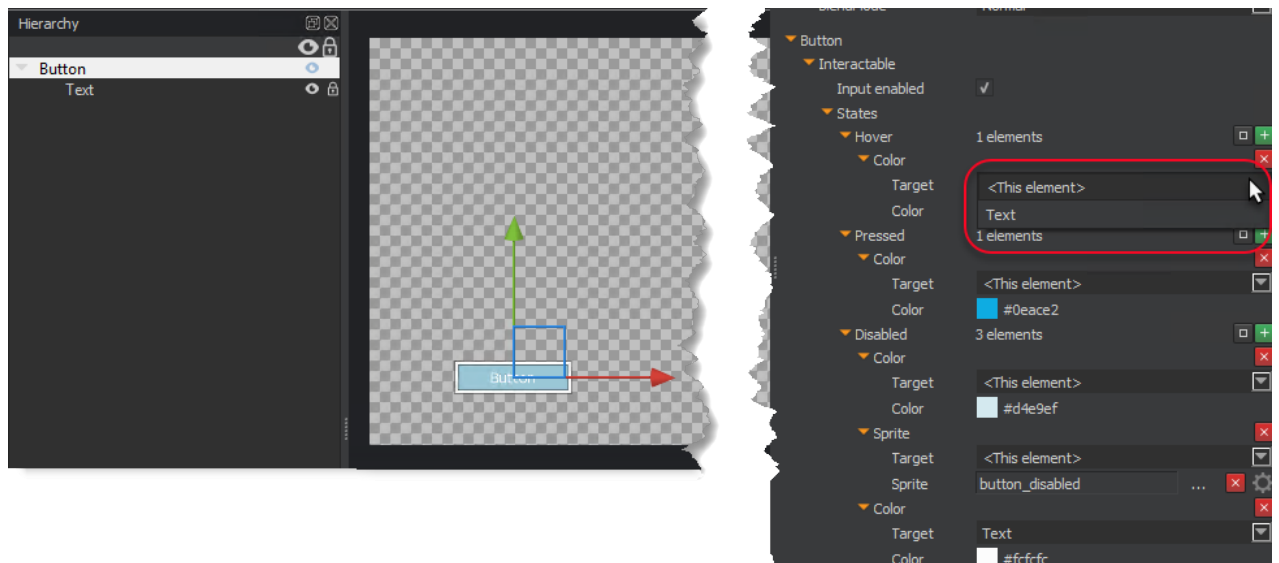
The normal appearance of a visual element (defined as an element with a visual component on it, such as image or text) is defined by the properties of that visual component. Some of the visual component's properties, however, can be overridden by an interactive component that is in the **Hover**, **Pressed**, and **Disabled** states.

The **Hover**, **Pressed**, and **Disabled** states have a list of **state actions**, which define the appearance of that state, and which override the corresponding property on the visual component:

- **Color** – RGB color tint
- **Alpha** – Opacity
- **Sprite** – Texture
- **Font** – Text font and font effect (on a text component, for example)

The **state actions**—**Color**, **Alpha**, **Sprite**, and **Font**—each have a **Target** property that specifies which visual element is to be affected. The elements from which you can choose include the current element—listed as **<This element>**, its child elements, and the descendants of its child elements. Using the **Target** property, you can pick exactly which visual element to override.

For example, the button [prefab](#) (p. 1153) has a top element named **Button** that has a visual component to define color. It also has a child element with a text component to define the text (and its color) on the button. The top element (**Button**) also has the **Interactable** component on it, and the **Target** for the **color** state action can override either the **Button** element's color, or the **Text** element's color, depending on what you pick from the list.

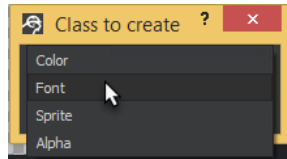


When you first add an interactive component to an element, there are no **state actions** added by default. You must add state actions to the states that you want to use and modify.

To add a state action to a state

In the **UI Editor** (p. 1137), in the **Properties** pane, under the interactive component's name (for example, **Button**), do the following:

1. Under **Interactable**, **States**, click **Add new element** (green +).
2. From the list, choose one of the following: Color, Font, Sprite, Alpha.



To delete a state action

- Click **Remove element** (red x) next to the state action that you want to delete.

To clear all state actions from a state

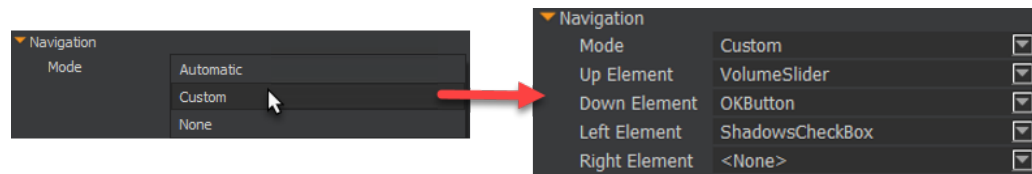
- Click **Clear container** (box icon) next to the state from which you want to clear all the state actions.

Navigation

You can use the **Navigation** group of properties to specify how the arrow keys or gamepad navigates between interactive elements.

For each interactive element, you can set navigation to one of the following:

- **Automatic** – Algorithm determines which interactive elements become focused when up, down, left, or right is pressed.
- **Custom** – You manually specify the interactive elements that become focused when up, down, left, or right is pressed.
- **None** – This option removes navigation capability; using the keyboard or gamepad, the player cannot focus on this element.



First Focus Element

To determine which element receives first focus when a canvas is first loaded, set the **First Focus Element** in the [Canvas Properties](#) (p. 1143). The **First Focus Element** receives focus upon canvas load when no mouse is detected. If you do not set a **First Focus Element**, or if a mouse is detected, no element is focused until the user provides direction input from a keyboard or mouse, whereupon the element closest to the top left corner of the canvas becomes focused.

Once an element is focused, navigation to other elements is controlled by the [navigation properties](#) (p. 1163) defined on each interactive component.

Interactive Element Controls

To interact with a focused element, press **Enter** on the keyboard (**A** on Xbox; **X** on PlayStation).

While an element is interactive, use the following controls:

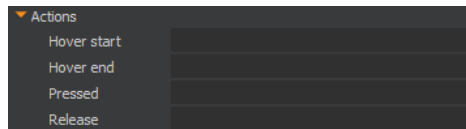
- **Button** and **Checkbox** – **Enter** presses the button or selects/clears the check box, and then returns to navigation automatically. Does not remain active after action.
- **Slider** and **Scrollbar** – Use arrow keys or joystick to move the slider or scroll box. Press **Enter** to return to navigation.
- **TextInput** – While active, use the following (press **Enter** to return to navigation):
 - Arrow keys move the text cursor
 - **Shift**+arrow keys selects text
 - Alphanumerical keys enter text at cursor position
 - **Ctrl+A** selects the entire text string
 - **Backspace** deletes the character to the left of the cursor
 - **Delete** deletes the character to the right of the cursor

Actions

You can use the **Actions** properties to trigger a particular event when one of the listed actions occur. Type a string in the action field. When the listed action occurs (for example, when a game player starts to hover over the element), the listed string is sent as an action. You can set up the [Ule:Canvas:ActionListener \(p. 1185\)](#) flow graph node to listen for this action.

You can strings for the following actions:

- **Hover start**
- **Hover end**
- **Pressed**
- **Release**



Button

You can use a button component to make an element behave like a button. To see an example of a completed canvas with the button component, open the `UiCompButton.uicanvas` file in the [FeatureTests Project \(p. 1108\)](#).

Note the following:

- This component is typically used on an element with an image component; if no visual or image component is present, many of the following properties have no effect.
- If you want to add a text label to a button, add a child element with a text component.
- To define borders for a sliced image type, open the sprite **Border Editor** by clicking the gear icon next to the **Sprite** path folder icon.

To edit a button component

In the [UI Editor \(p. 1137\)](#) **Properties** pane, expand **Button** and do the following, as appropriate:

Interactable

See [Properties \(p. 1161\)](#) to edit the common interactive component settings.

Actions, Click

Type a text string. This string is sent as an action on the UI canvas when the button is clicked. You can listen for this action in the flow graph using the [UI:Canvas:ActionListener Node \(p. 1241\)](#).

Check box

You can use this component to make an element behave like a check box. This component is typically used on an element with two visual child elements—one to display when the check box is selected and another to display when the check box is cleared. To see an example of a completed canvas with the check box component, open the `UiCompCheckBox.uicanvas` file in the [FeatureTests Project \(p. 1108\)](#).

To edit a check box component

In the **Properties** pane of the [UI Editor \(p. 1137\)](#), expand **Checkbox** and do the following, as appropriate:

Interactable

See [Properties \(p. 1161\)](#) to edit the common interactive component settings.

Elements, On

Select an element from the list to provide the entity to be displayed when the check box state is `on` (selected).

Elements, Off

Select an element from the list to provide the entity to be displayed when the check box state is `off` (cleared).

Value, Checked

Click the box to change the initial state of the check box.

Actions, Change

Type a text string. This string is sent as an action on the UI canvas when the check box has any state changes. You can listen for this action in the flow graph using [UI:Canvas:ActionListener Node \(p. 1241\)](#).

Actions, On

Type a text string. This string is sent as an action on the UI canvas when the check box state changes to `on` (selected). You can listen for this action in the flow graph using [UI:Canvas:ActionListener Node \(p. 1241\)](#).

Actions, Off

Type a text string. This string is sent as an action on the UI canvas when the check box state changes to `off` (cleared). You can listen for this action in the flow graph using [UI:Canvas:ActionListener Node \(p. 1241\)](#).

Slider

You can use this component to make an element behave like a slider. This component is typically used on an element with three visual child elements: one immediate child, called **Track**, and two child elements of the track, called **Fill** and **Handle**. To see an example of a completed canvas with the slider component, open the `UiCompSlider.uicanvas` file in the [FeatureTests Project \(p. 1108\)](#).

To edit a slider component

In the **Properties** pane, expand **Slider** and do the following, as appropriate:

Interactable

See [Properties \(p. 1161\)](#) to edit the common interactive component settings.

Elements, Track

Select an element from the list to provide the entity to be displayed as the background of the slider and to limit the movement of the manipulator.

Elements, Fill

Select an element from the list to provide the entity to be displayed as the background of the slider, from the lower limit to the center of the manipulator position.

Elements, Manipulator

Select an element from the list to provide the entity to be displayed as the movable knob of the slider.

Value, Value

Enter the initial value of the slider.

Value, Min

Enter the lower limit of the slider.

Value, Max

Enter the upper limit of the slider.

Value, Stepping

Enter the step value. For example, use 1 to only permit whole integer values.

Actions, Change

Type a text string. This string is sent as an action on the UI canvas when the slider is finished changing values. You can listen for this action in the flow graph using [UI:Canvas:ActionListener Node \(p. 1241\)](#).

Actions, End Change

Type a text string. This string is sent as an action on the UI canvas when the slider is changing values. You can listen for this action in the flow graph using [UI:Canvas:ActionListener Node \(p. 1241\)](#).

Text Input

You can use a text input component to make an element offer player input. This component is typically used on an element with an image component and two child elements with text components (one for placeholder text and one for input text). To see an example of a completed canvas with the text input component, open the `UiCompTextInput.uicanvas` file in the [FeatureTests Project \(p. 1108\)](#).

To edit a text input component

In the **UI Editor Properties** pane, expand **TextInput** and do the following, as appropriate:

Interactable

See [Properties \(p. 1161\)](#) to edit the common interactive component settings.

Elements, Text

Select an element from the list to provide the text component for the input text. The list shows child elements that have text components.

Elements, Placeholder text element

Select an element from the list to provide the text component for the placeholder text. The list shows child elements that have text components.

Text editing, Selection color

Click the color swatch to select a different color for selected text.

Text editing, Cursor color

Click the color swatch to select a different color for the cursor.

Text editing, Max char count

Enter the maximum number of characters allowed in the text input box. Enter -1 for no character limit.

Text editing, Cursor blink time

Enter a value in seconds. Use 0 for no blink, 1 to blink once every second, 2 to blink once every two seconds, etc.

Text editing, Is password field

Check the box and specify the replacement character.

Text editing, Clip input text

Sets the **Overflow mode** of the text element to **Clip text** at runtime.

Actions, Change

Type a text string. This string is sent as an action on the UI canvas whenever a change occurs in the text input, such as typing or deleting a character.

Actions, End edit

Type a text string. This string is sent as an action on the UI canvas whenever the player clicks off the text input or presses **Enter**.

Actions, Enter

Type a text string. This string is sent as an action on the UI canvas when the player presses **Enter**.

ScrollBox

You can use a scroll box component to present content, such as images or text, within a scrollable area. To see an example of a completed canvas with the scroll box component, open the `UiCompScrollBox.uicanvas` and `UiCompScrollBox_More.uicanvas` files in the [FeatureTests Project](#) (p. 1108).

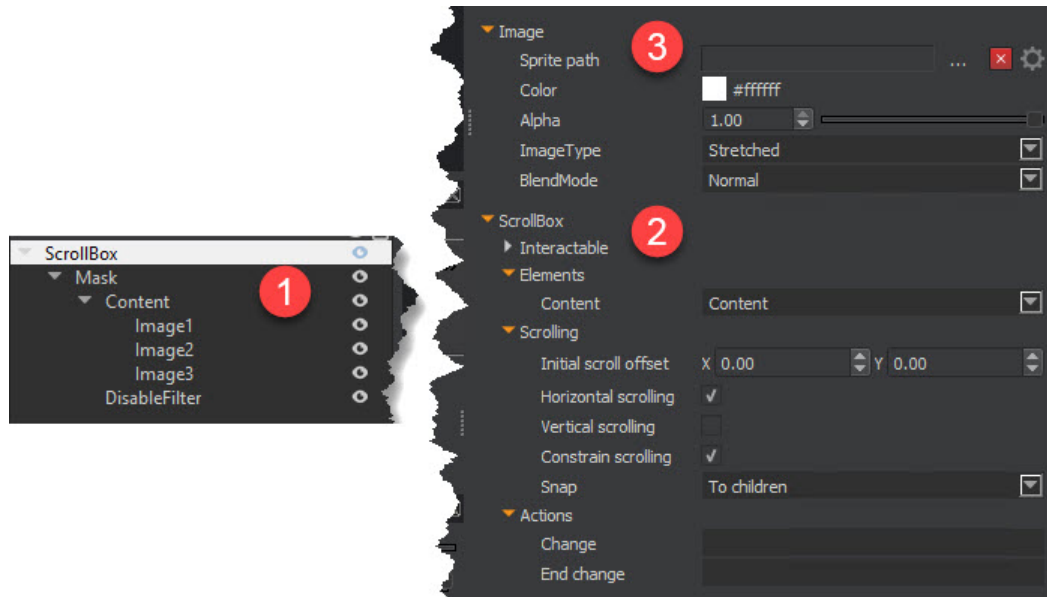
This component is typically used with a mask component, which hides content outside of the masked area.

You can add a prefabricated scroll box element. When you do this, a mask, content, and image elements are automatically created and nested in your **Hierarchy** pane.

To add a ScrollBox element from prefab

- Click **New, Element from prefab, ScrollBox**.

The element named **ScrollBox** (1) has the **ScrollBox** component (2) on it. You can add an image to the **ScrollBox** element's **Image** component (3), which acts as the visual frame for the scroll box. Because the mask element and its child elements are drawn in front of the scroll box element, you see only the edges of the image on the **ScrollBox** component. To increase or decrease the viewable area of this image, adjust the offsets in the mask element's [Transform2D](#) (p. 1155) component.



The element named **Mask** has a **Mask** (p. 1171) component on it, which acts as the viewport through which you can see the content. To specify a custom mask, you can add an image to the **Mask** element's **Image** component. The contents are drawn to the visible area of the mask; the transparent area of the mask hides content.

To edit a scroll box component

In the **Properties** pane, expand **ScrollBox** and do the following, as appropriate:

Interactable

See [Properties](#) (p. 1161) to edit the common interactive component settings.

Content, Content element

Select an element from the list to provide the content to be displayed within the scroll box.

Content, Initial scroll offset

Enter the initial offset value of the content element's pivot point from the parent element's pivot point.

Content, Constrain scrolling

Select the check box to prevent content from scrolling beyond its edges.

Content, Snap

Select a snapping mode:

- **None** – No snapping.
- **To children** – When a drag motion is released, the content element moves in such a way that the closest child element's pivot point is snapped to the parent element's pivot point. You can use this, for example, to center a child element in the scroll box when a drag is released.
- **To grid** – When a drag motion is released, the content element's pivot point is snapped to a multiple of the grid spacing from the parent element's pivot point.

Horizontal scrolling, Enabled

Select the check box to enable content to scroll horizontally. If the element, or its parent, is rotated, then the axis of scrolling is also rotated. You can enable horizontal scrolling simultaneously with vertical scrolling to scroll in both directions.

Horizontal scrolling, Scrollbar element

Select an element from the list to provide the horizontal scrollbar associated with the scroll box.

Horizontal scrolling, Scrollbar visibility

Select the visibility behavior of the horizontal scrollbar:

- **Always visible** – Scroll bar is always visible.
- **Auto hide** – Scroll bar is automatically hidden when not needed. Scroll bar is resized according to visibility of the vertical scroll bar.
- **Auto hide and resize view area** – Same as auto hide, but the view area is also resized smaller when the scroll bar is visible, and larger when the scroll bar is hidden.

Vertical scrolling, Enabled

Select the check box to enable content to scroll vertically. If the element, or its parent, is rotated, then the axis of scrolling is also rotated. You can enable vertical scrolling simultaneously with horizontal scrolling to scroll in both directions.

Vertical scrolling, Scrollbar element

Select an element from the list to provide the vertical scrollbar associated with the scroll box.

Vertical scrolling, Scrollbar visibility

Select the visibility behavior of the vertical scrollbar:

- **Always visible** – Scroll bar is always visible.
- **Auto hide** – Scroll bar is automatically hidden when not needed. Scroll bar is resized according to visibility of the vertical scroll bar.
- **Auto hide and resize view area** – Same as auto hide, but the view area is also resized smaller when the scroll bar is visible, and larger when the scroll bar is hidden.

Actions, Change

Set the action that is triggered during a drag each time the position changes.

Actions, End change

Set the action that is triggered when a drag motion is completed.

ScrollBar

You can use a scroll bar component to add a scrollable bar, or handle, for manipulating settings or scrolling within a scroll box. To see an example of a completed canvas with the scroll bar component, open the `UICompScrollBar.uicanvas` file in the [FeatureTests Project \(p. 1108\)](#).

This is a horizontal scroll bar:



This is an image within a scroll box with both a horizontal and a vertical scroll bar:



You can add a prefabricated horizontal or vertical scroll bar element. When you do this, a handle is automatically created and nested in your **Hierarchy** pane.

To add a ScrollBar element from prefab

- Click **New, Element from prefab, ScrollBarHorizontal** or **ScrollBarVertical**.

To edit a scroll bar component

In the **Properties** pane, expand **ScrollBar** and do the following, as appropriate:

Interactable

See [Properties \(p. 1161\)](#) to edit the common interactive component settings.

Elements, Handle

Select an element from the list to provide the entity to be displayed as the movable handle of the scrollbar.

Values, Orientation

Select the scrollbar's orientation:

- **Horizontal** – Scrollbar's handle moves left and right.
- **Vertical** – Scrollbar's handle moves up and down.

Values, Value

Enter the initial value of the scrollbar (0.0 to 1.0).

Values, Handle size

Enter the size of the handle relative to the scrollbar (0.0 to 1.0).

Values, Min handle size

Enter the minimum size of the handle in pixels.

Actions, Change

Type a text string. This string is sent as an action on the UI canvas when the scroll bar changes values. You can listen for this action in the flow graph using [Ule:Canvas:ActionListener Node \(p. 1185\)](#).

Actions, End Change

Type a text string. This string is sent as an action on the UI canvas when the scroll bar has finished changing values. You can listen for this action in the flow graph using [Ule:Canvas:ActionListener Node \(p. 1185\)](#).

Layout Components

You can add one layout component to an element to organize child elements into uniform columns, rows, or a grid.

Layout Column

You can use a layout column component to organize child elements into a uniform column.

To see an example of a completed canvas with the layout column component, open the `UiCompLayout.uicanvas` file in the [FeatureTests Project \(p. 1108\)](#).

To edit a layout column component

In the **Properties** pane of the [UI Editor \(p. 1137\)](#), expand **LayoutColumn** and do the following, as appropriate:

- For **Padding**, enter values in pixels, relative to the element's borders.
- For **Spacing**, enter values in pixels to adjust spacing between elements.
- For **Order**, select **Top-to-Bottom** or **Bottom-to-Top** to specify the order in which the child elements appear in the column.

Layout Row

You can use a layout row component to organize child elements into a uniform row.

To edit a layout row component

In the **Properties** pane, expand **LayoutRow** and do the following, as appropriate:

- For **Padding**, enter values in pixels, relative to the element's borders.
- For **Spacing**, enter values in pixels to adjust spacing between elements.
- For **Order**, select **Left-to-Right** or **Right-to-Left** to determine the order in which the child elements appear in the row.

Layout Grid

You can use a layout grid component to organize child elements into a uniform grid. To see an example of a completed canvas with the layout column component, open the `UiCompLayout.uicanvas` file in the [FeatureTests Project \(p. 1108\)](#).

To edit a layout grid component

In the **Properties** pane, expand **LayoutGrid** and do the following, as appropriate:

Padding

Enter values in pixels, relative to the element's borders.

Spacing

Enter values in pixels to adjust spacing among elements.

Cell size

Enter values in pixels to specify the size of the child elements.

Order

Do the following as appropriate:

- For **Horizontal**, select **Left-to-Right** or **Right-to-Left** to determine the order in which elements appear horizontally.
- For **Vertical**, select **Top-to-Bottom** or **Bottom-to-Top** to determine the order in which elements appear vertically.
- For **Starting With**, select **Horizontal** or **Vertical** to determine whether elements appear horizontally or vertically first.

Other Components

You can add either or both of the fader and mask components to an element.

Fader

You can use a fader component to simultaneously adjust the transparency of an element and its children. To see an example of a completed canvas with the fader component, open the `UiCompFader.uicanvas` file in the [FeatureTests Project \(p. 1108\)](#).

To edit a fader component

1. In the **Properties** pane of the [UI Editor \(p. 1137\)](#), expand **Fader**.
2. For the **Fade** multiplier, use the slider to select a number between 0 (invisible) and 1 (opaque) and press **Enter**.

Mask

You can add a mask component to an element to show a portion of content in child elements (for example, image or text). To see an example of a completed canvas with the mask component, open the `UiCompMask.uicanvas` file in the [FeatureTests Project \(p. 1108\)](#).

When you add a mask component, the default mask (visible area) is a square. If you want to use a nonrectangular mask, you need a texture or image that contains an [alpha channel \(p. 1372\)](#), which specifies transparent and opaque areas. You can set the image component of the element as a custom image to be used as a mask. The child elements are drawn to (shown by) the visible area of the image and hidden by the transparent area of the image. Masks are most commonly used in conjunction with a [scrollbox prefab element \(p. 1167\)](#).

To add an image to be used as a custom mask

1. In the [UI Editor \(p. 1137\)](#) toolbar, create a new empty element by clicking **New, Empty Element**. This is the parent element.
2. In the **Properties** pane, add an image component by clicking **Add Component, Image**.
3. Add a mask component by clicking **Add Component, Mask**.
4. Add a child element by right-clicking the parent element and then clicking **New, Empty Element**.
5. Select the child element. Add an image component to the child element.
6. Select an image for the child element by clicking the folder icon next to **Image, Sprite Path** in the **Properties** pane. Open an image file that is located within your current project directory.
7. Select the parent element. Select the texture or image to use as a mask by clicking the folder icon next to the **Image, Sprite Path** in the **Properties** pane. Open an image file that is located within your current project directory.

The image that you use as a mask should have opaque areas (which shows the content in child elements) and transparent areas (which hides the content in child elements).

8. In the **Properties** pane, under **Mask**, select **Use alpha test**.

To edit a mask component

In the **Properties** pane, expand **Mask** and use the following settings, as appropriate:

Enable Masking

Enables masking (selected by default). When selected, only the parts of the child elements that are revealed by the mask are visible.

Draw behind

Draws the mask visual behind the child elements. Can be useful for debugging purposes.

Draw in front

Draws the mask visual in front of the child elements. Can be useful for debugging purposes.

Use alpha test

Uses the alpha channel in the masks visual's texture to define the mask. Must be enabled for masks that are anything other than a rectangle.

Mask interaction

Prevents input events from being sent to elements that are outside of the mask.

Implementing New Fonts

You can add fonts to your game UI in Lumberyard by saving the font asset to your game project and creating an `.xml` file that contains specifics for that font, such as the path to the font file and parameters that affect the font's appearance. You can combine multiple font assets into a single font family, and further customize text appearance using [Text Styling Markup \(p. 1159\)](#).

Using the procedures in this section, you can:

- [Add new fonts to your game UI \(p. 1173\)](#)
- [Create font families \(p. 1173\)](#)

- Achieve custom styling
- Configure font rendering quality

Adding New Fonts

For each new font that you want to add, you need the following files:

- Font asset – True Type Font (.ttf) or Open Type Font (.otf) file.
- Font .xml file describing the asset.

To add a new font to your UI

1. Save the font asset (.ttf or .otf file) to your game project directory, such as dev\SamplesProject\Font.
2. Copy an existing font .xml file into your game project Font directory. The following font .xml files are included in the Lumberyard project:
 - dev\Engine\Fonts\default-ui.xml
 - dev\SamplesProject\Fonts\NotoSans-Regular.xml
3. Change the .xml file name (leave the .xml extension unchanged). Use any file name that is descriptive and appropriate for your purposes.
4. Open the .xml file and edit the following line to point to your font asset file name:

```
<font path="yourFont.ttf" w="512" h="256"/>
```

Once the Asset Processor is finished processing your font assets, you can select your font by [loading the font.xml \(p. 1159\)](#) in the **UI Editor** for any text component.

Creating Font Families

You can combine multiple font assets into a single font family group.

The following is an example of a .fontfamily file.

```
<fontfamily name="MyFontFamily">
  <font>
    <file path="myfontfamily-regular.xml" />
    <file path="myfontfamily-bold.xml" tags="b" />
    <file path="myfontfamily-italic.xml" tags="i" />
    <file path="myfontfamily-bolditalic.xml" tags="b,i" />
  </font>
</fontfamily>
```

The UI system uses the font family definitions to determine which font asset to apply when styling text. You can combine the following types of assets:

- **Unstyled** – Font representing text with no styling applied. In the example above, this is myfontfamily-regular.xml.
- **Bold** – Font representing text with bold styling.
- **Italic** – Font representing text with italic styling.
- **Bold-Italic** – Font representing text with both bold and italic styling.

Font Family File XML

To create a new font family file, you can create a new, empty plain text file and enter the contents, or you can modify an existing font family file.

To add a new font family file to your UI

1. To create a new font family file, do one of the following:
 - Open Notepad (or similar program) and save an empty text file with a `.fontfamily` file extension.
 - Copy an existing `.fontfamily` file into your game project's `Fonts` directory. The following `.fontfamily` files are included in the Lumberyard project:
 - `dev\FeatureTests\Fonts\NotoSans\NotoSans.fontfamily`
 - `dev\FeatureTests\Fonts\NotoSerif\NotoSerif.fontfamily`
2. Name your `.fontfamily` file appropriately (leave the `.fontfamily` extension)
3. Open your `.fontfamily` file and edit the contents to configure the font family.

For example:

```
<fontfamily name="MyFontFamily">
  <font>
    <file path="myfontfamily-regular.xml" />
    <file path="myfontfamily-bold.xml" tags="b" />
    <file path="myfontfamily-italic.xml" tags="i" />
    <file path="myfontfamily-bolditalic.xml" tags="b,i" />
  </font>
</fontfamily>
```

Once the Asset Processor has finished processing your font assets, you can select your font family by selecting the `*.fontfamily` file in the UI Editor as the font for any text component. To apply custom styling to text using the font family, see [Text Styling Markup \(p. 1159\)](#).

The `.fontfamily` file uses XML. The UI system supports the following tags and attributes for the `.fontfamily` file:

Tag: `fontfamily`
Attribute: `name`

The unique name of the font family. Each font family name must be unique. You can, however, use the same font assets in multiple font families.

Tag: `font`
Container tag for the `file` tag.

Tag: `file`
Attribute: `path`

Path to the font asset, a TTF or OTF file. The path is relative to the font family file.

Attribute: `tags`

This tag is optional. If omitted, this font file is used when no styling is applied.

Values:

- **b** – indicates `` bold tag
- **i** – indicates `<i>` italic tag

- **b,i** – indicates when both bold and <i> italic tags are applied

Configuring Font Rendering Quality

Lumberyard's built-in UI system, *LyShine*, renders text using font textures. The quality of the on-screen text is affected by the font texture size and the size of the text itself when rendered on the screen.

Use the procedures in this section to configure font size and texture to achieve quality text rendering.

Font Texture Width and Height Attributes

The font .xml file included in the Lumberyard project has the following content:

```
<fontshader>
  <font path="Vera.ttf" w="512" h="256"/>
  <effect name="default">
    <pass>
    </pass>
  </effect>
  <effect name="drop_shadow">
    <pass>
    </pass>
    <pass>
      <color r="0" g="0" b="0" a="1"/>
      <pos x="1" y="1"/>
    </pass>
  </effect>
</fontshader>
```

The font texture resolution is controlled by the following line:

```
<font path="Vera.ttf" w="512" h="256"/>
```

In this example, the font texture has a resolution of 512x256. This resolution size is an important number for determining font rendering quality.

Character Slots

In Lumberyard, a font texture is logically divided up into equally sized slots. The grid size is 16x8. This means that a font texture can hold a maximum of 128 characters.

Font Size

Because a font texture is divided into a logical grid, a simple calculation determines how much real estate each character in the font can use:

- Font texture width / 16 = Slot width
- Font texture height / 8 = Slot height

In the example given in the previous section, the font texture size was 512x256.

- 512 / 16 = 32 (slot width)

- $256 / 8 = 32$ (slot height)

For a 512x256 sized font texture, you can render pixel-perfect characters at 32x32.

Knowing these calculations helps you to determine the right font texture size for the purposes of your game UI.

To determine the right font texture size for your game UI

1. [Create the font .xml file \(p. 1173\)](#) for the font you want to use.
2. Choose an arbitrary font texture size to start with, such as 512x256 as used in the example above.
3. Use the **UI Editor** to mock up a canvas with text elements that use your font .xml file.
4. In the **UI Editor's Properties** pane, under **Text, Font Size**, experiment with the font size to find the ideal size for your use case.
5. Once you have determined the appropriate font size for your purposes, use the following formula to determine the font texture width and height.
 - Texture width = Font size * 16
 - Texture height = Font size * 8
6. Edit your font .xml to use the calculated font texture size.

Note

- Font texture sizes do not necessarily need to be a power of 2 (128, 256, 512, 1024, 2048, and so on), but the width must be a multiple of 16, and the height must be a multiple of 8.
- You can have multiple font .xml files that reference the same font but have different font texture sizes.

For example, you may have some caption text that needs to appear only at a small font size, but you have other screens (perhaps a menu screen) where you want the same look and feel—by using the same font—that needs to be larger, and therefore needs a higher resolution font texture.

Using the Animation Editor

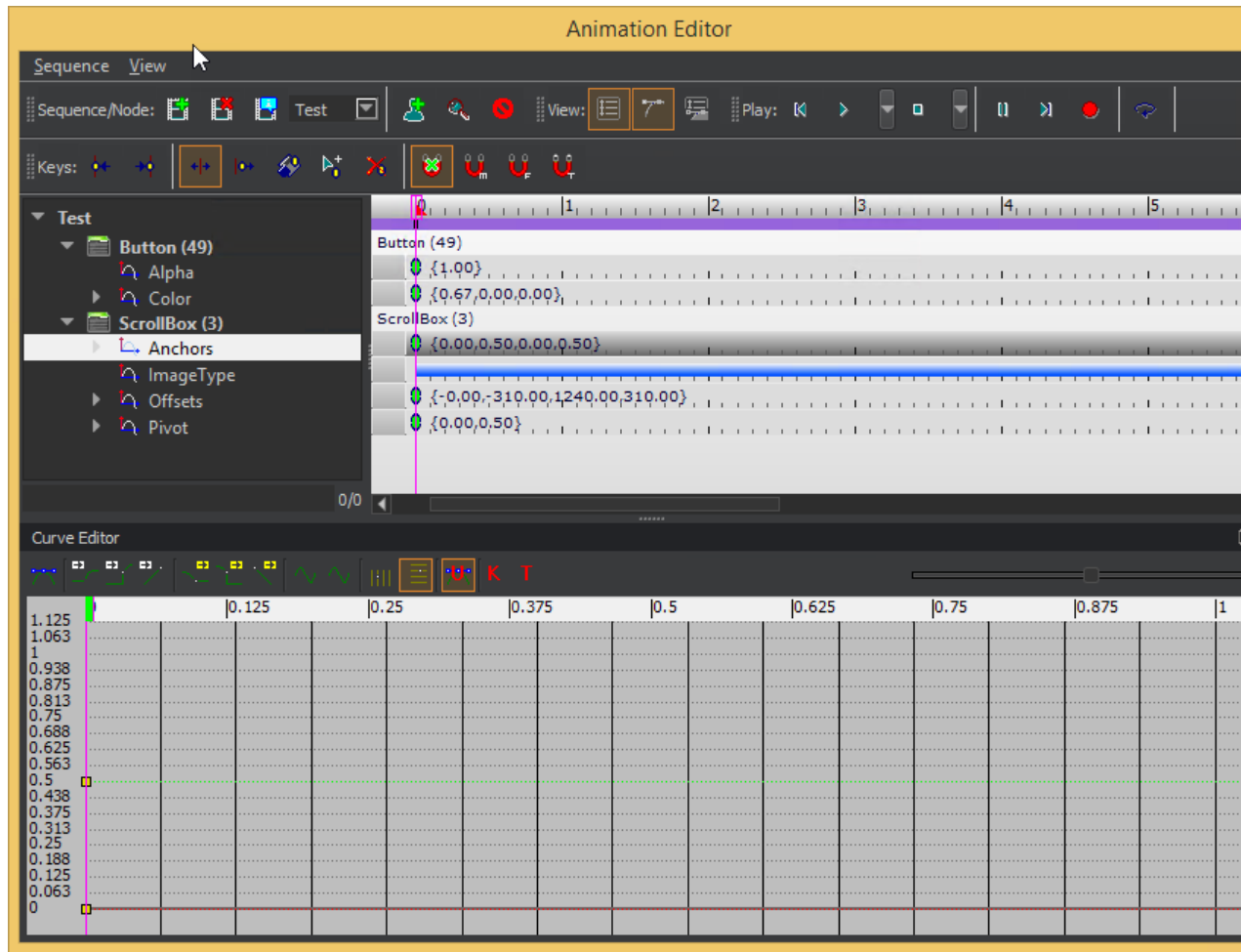
You can use animation sequences to animate UI elements. A UI canvas can contain many named animation sequences.

The **Animation Editor** has the following features:

- **Menu** – Operations for creating new animation sequences and switching between the **Track Editor** and **Curve Editor**.
- **Toolbar** – Tools for the editing and playback of animations. The **Curve Editor** displays an additional toolbar at the top of the pane.
- **Node pane** – Area for showing the active sequence and all of the elements that it is animating. A track for each animated property appears underneath the related element.
- **Editor pane** – Area for either the **Track Editor**, the **Curve Editor**, or both.

To show the Animation Editor editor if it is not already visible

- From the [UI Editor \(p. 1137\)](#) menu, choose **View, Animation Editor**.



To create an animation sequence, you first create a new sequence, assign one or more UI elements to it, and then record changes you make to the UI element(s)—this becomes the animation sequence. You can then edit the animation sequence(s) using the **Animation Editor**. These processes are described in greater detail in the following sections.

Recording Animation Data

Recording animation typically involves three steps:

1. Create a new animation sequence.
2. Add a UI element to that sequence.
3. Turn on animation recording to capture changes in the element properties.

Adding a UI element also adds a node to the sequence. After that any time that you enter record mode, a track is automatically added to your animation for any change you make to this UI element. You do not need to manually add tracks. For more information, see [Using the Node Pane \(p. 1179\)](#).

You can create an animation sequence from the **Animation Editor** menu or toolbar.

To create a new animation sequence

In the [Animation Editor \(p. 1176\)](#), do one of the following:

- From the **Sequence** menu, choose **New Sequence**.
- Click the **Add Sequence** icon on the toolbar.

To add a UI element to the sequence

1. In the [UI Editor \(p. 1137\)](#), select the UI element that you want to animate.
2. In the **Animation Editor**, right-click the sequence that you created and click **Add Selected UI Element(s)**.

To record an animation sequence

1. In the **Animation Editor** toolbar, click the **Record** icon.
2. In the **UI Editor**, use either the **Properties** pane or viewport pane to make changes to the selected UI element.
3. After making all changes, click the **Stop** icon in the **Animation Editor** toolbar.

Note

In the current release, not all component properties can be recorded. For example, enumerated values, such as the image type of an image component, cannot be animated.

After you record a track, it appears beneath its UI element. The node pane lists your current animation sequences. For more information on the **Node Pane**, see [Using the Node Pane \(p. 1179\)](#)

Playing Animation Sequences

You can play back the animation in the **Animation Editor** to preview what it will look like in your game. Playing the animation sequence animates the UI elements in the **UI Editor**.

Tip

You can also play animations in the **Flow Graph Editor**. Use the **UI:Sequence:Play** node to play animation sequences in the game from a flow graph. For more information about this flow graph node, see [UI Animation Node \(p. 1303\)](#).

To control playback of animation in the UI Editor

- In the **Play** toolbar of the **Animation Editor**, use the **Play**, **Pause**, **Stop**, **Go to start of sequence**, and **Go to end of sequence** buttons.

Editing Animation Data

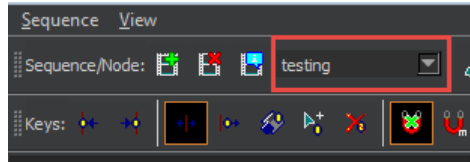
After you create your sequence(s) and record animation data to them, you can use the **Node Pane**, **Track Editor**, and **Curve Editor** in the **Animation Editor** to modify your sequences.

- In the **Node Pane**, you can add or remove UI elements from an animation sequence, edit sequences, and work with keys. For more information, see [Using the Node Pane \(p. 1179\)](#).
- In the **Track Editor**, you can limit your animation preview, manipulate keys, and change your animation's timeline. For more information, see [Using the Track Editor \(p. 1181\)](#).
- In the **Curve Editor**, you can manipulate splines to change the behavior of the transitions between keys. For more information, see [Using the Curve Editor \(p. 1183\)](#).

You can use the toolbar to select a sequence to display and edit.

To select an animation sequence to edit

- In **Animation Editor**, click the arrow next to the name of the current sequence in the toolbar to display a list of active sequences available to edit.



Topics

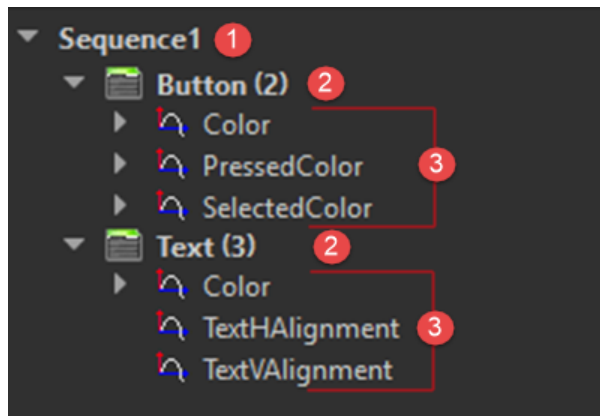
- [Using the Node Pane \(p. 1179\)](#)
- [Using the Track Editor \(p. 1181\)](#)
- [Using the Curve Editor \(p. 1183\)](#)

Using the Node Pane

The **Node Pane** in the [Animation Editor \(p. 1176\)](#) displays all the nodes in the selected animation sequence. Each item listed in the **Node Pane** is considered a node, though they represent different parts of the sequence. You can use the **Node Pane** to add or delete UI element nodes. Track nodes appear beneath its UI element when you record a track.

The animation sequence node, at the top level, contains a list of its UI elements nodes. Each UI element node contains a list of its track nodes.

1. **Animation Sequence** node
2. **UI Element** nodes
3. **Track** nodes



To add a new UI element node

1. In the [UI Editor \(p. 1137\)](#), select one or more elements.
2. In the **Animation Editor**, right-click anywhere in the node pane and select **Add Selected UI Element(s)**.

To remove a UI element node

- In the **Animation Editor**, in the node pane, right-click an element node and click **Delete**.

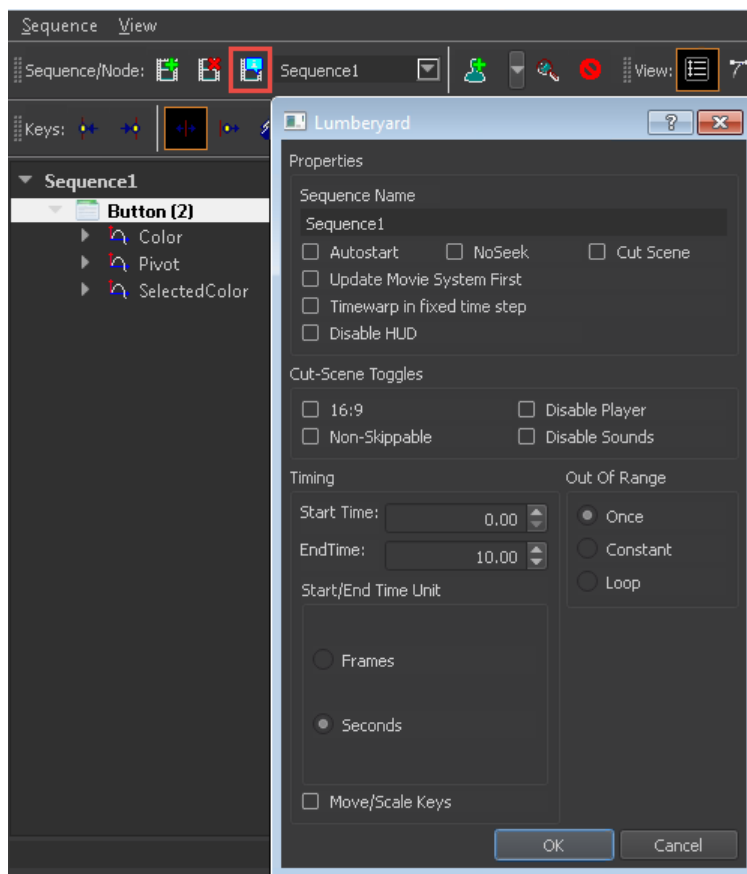
To edit a track

1. In the **Animation Editor**, in the node pane, select a track node.
2. Right-click the track node and choose any of the following:
 - **Copy Keys**
 - **Copy Selected Keys**
 - **Paste Keys**
 - **Disable the track**

You can also use the **Edit Sequence** tool to edit the properties of the sequence directly. You can set various properties, such as the start and end time, whether the sequence loops, and so on.

To open the Edit Sequence tool

- In the **Animation Editor**, click the **Edit Sequence** icon.



Using the Track Editor

The **Track Editor** displays all the tracks in your current animation sequence. The **Track Editor** enables you to do the following:

- Move, delete, copy, and paste keys
- Change the timeline of the animation
- Control the animation preview range

To display the Track Editor

- In the [Animation Editor](#) (p. 1176), choose **View, Track Editor** or **View, Both**.

To zoom in or out

- Scroll the mouse wheel

To pan the view

- With the mouse in the **Track Editor**, drag using the middle mouse button

Topics

- [Working with Keys in the Track Editor](#) (p. 1181)
- [Moving the Play or Record Point in the Track Editor](#) (p. 1182)
- [Previewing in the Track Editor](#) (p. 1183)

Working with Keys in the Track Editor

When you create an animation, key values are automatically recorded. Using the **Track Editor**, you can move, delete, copy, and paste keys. Keys are represented by a green circle on the timeline of each track.

To move a key

- Click a key and drag it to a new time on the timeline.

To constrain movement to time only

- Hold **Shift** as you drag the key to a new time on the timeline.

To scale the selected key frames while moving a key

- Hold **Alt** as you drag the key to a new time on the timeline.

To delete a key

- Right-click a key and click **Delete**.

To copy a key

- Right-click a key and click **Copy**.

To paste a key

- Right-click in the timeline and click **Paste**. Move the key to the desired point on the timeline, then click to place.

The **Track Editor**'s toolbar features a variety of tools to improve your workflow efficiency when editing tracks. Pause over each icon to reveal the tooltips.

Some of the toolbar functions require you to select multiple keys.

To select multiple keys

- In the **Track Editor**, drag to select multiple keys. The selected keys appear as white circles.

You can also use the Track Editor toolbar to select, move, and snap keys. When moving keys, you can choose to snap them to other keys, to frames, or to second ticks.

Working with Keys in the Track Editor Toolbar

Toolbar icon	Function
Go to previous key	Selects the key directly before the currently selected key.
Go to next key	Selects the key directly after the currently selected key.
Slide keys	Moves the currently selected key and slides all the keys after it to the new point on the timeline while maintaining the original spacing.
Move keys	Moves the currently selected key(s) to the new point on the timeline without affecting other keys.
Scale keys	Functions only with multiple keys selected to increase or decrease the space between the selected keys proportionally.
Magnet Snapping	Snaps to keys in other tracks as you get close to them; allows you to place the key anywhere but indicates a red circle on the key you want to snap to.
Frame Snapping	Snaps to frames.
Tick Snapping	Snaps to second ticks.

Moving the Play or Record Point in the Track Editor

The play or record point of the animation sequence is shown as a vertical magenta slider on the timeline. Move the play or record point, and the properties of the UI elements in the [Animation Editor](#) (p. 1176) change to the values specified by the animation tracks.

To move the play or record point in the Track Editor

- Click or drag the vertical magenta slider in the timeline.

Previewing in the Track Editor

The **Track Editor** features a timeline along its top edge. To preview your entire animation, simply click the **Play** button to play your animation at its normal speed. You can also change the speed of preview by clicking the arrow beside the play button and selecting 2, 1, $\frac{1}{2}$, $\frac{1}{4}$, or #. You can also limit your animation preview, as it plays, to a specific time frame.

To limit play preview in the Track Editor

1. In the timeline, at the start of your preferred preview time, right-click to mark the time with a red triangle.
2. In the timeline, at the end of your preferred preview time, right-click again to mark the end time with a red triangle.
3. Click the **Play** button to preview your animation in the time frame specified.

Note

When you preview an animation or move the playback position on the timeline, it moves the UI elements in the **UI Editor**. This means that, if you then save the canvas, these UI elements will be saved in this position.

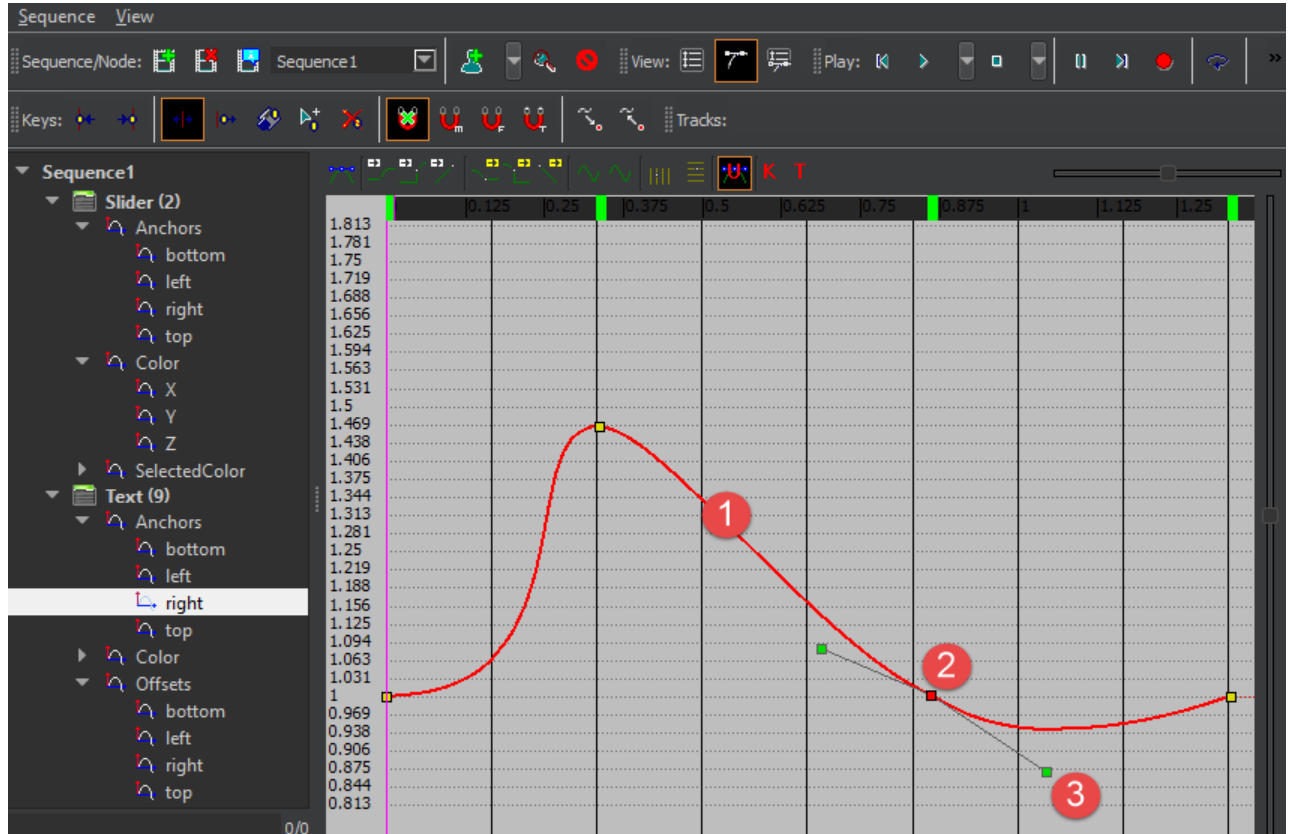
Reposition the timeline or preview a different sequence to position the UI elements at the positions in which you want them to load before you save the canvas.

Using the Curve Editor

The **Curve Editor** displays animations as function curves. Each track's curves represent an animation of a property value (such as anchor, offset, color, or any property of a UI element).

The elements of a curve

1. Curve or spline
2. Spline key
3. Tangent handles



The path of the curve represents the transition of the value between the keyframes. If the value changes in a straight line between each keyframe (linear), transitions between keyframes will not be smooth. The default curve causes the value to smoothly ease in and ease out. Each key has an in tangent and an out tangent. Depending on the preferred effect, you can use the toolbar icons to switch the tangents to auto, zero, step, or linear. You can also manually drag the tangent handles.

By default, animation tracks are recorded with a smooth transition. You can use the buttons in the toolbar at the top of the **Curve Editor** to change how the curves behave on either side of the selected key. You can also drag spline keys to a different point in the timeline.

To display the Curve Editor

- In the **UI Animation** editor, choose **View, Curve Editor** or **View, Both**.

To zoom in or out

- Scroll the mouse wheel

To pan the view

- With the mouse in the **Curve Editor**, drag using the middle mouse button

To adjust a spline key

1. In the **Node Pane**, select a track. The curves for that track appear in the **Curve Editor**.
2. In the **Curve Editor**, select a spline key.
3. Do one or more of the following:

- Drag the spline key to a different point on the timeline.
- Use the toolbar buttons to select a preset: auto, zero, step, or linear.

You can select multiple spline keys to modify at once. Once selected, you can move them all together, set their in and out tangents, and so on.

To select multiple spline keys

- In the **Curve Editor**, drag a selection box over all the spline keys you want to select.

UI Flow Graph Nodes

You can use flow graph nodes to control the game's user interface. For example, you could specify an action that loads a specific UI canvas or set parameters for when to keep a canvas loaded.

Lumberyard features two sets of UI flow graph nodes: **Ule** and **UI**. The improved **Ule** nodes supersede the original (and now legacy) **UI** flow graph nodes. For best results when creating new flow graph nodes, use the **Ule** flow graph node set.

For more information on flow graphs, see [Flow Graph System \(p. 487\)](#).

Topics

- [Ule Flow Graph Nodes \(p. 1185\)](#)
- [UI Flow Graph Nodes \(p. 1241\)](#)

Ule Flow Graph Nodes

The **Ule** flow graph node set supersedes the original **UI** flow graph node set (now legacy). The **Ule** node set behaves the same as the original **UI** node set, but simplifies how the nodes associate with UI canvases and UI elements.

You associate each **Ule** node with a UI canvas by setting the node's **Choose Entity** input. For information on how to associate UI canvases with UI flow graph nodes, see [Associating Canvases with UI Flow Graph Nodes \(p. 1144\)](#).

Topics

- [Ule Canvas Nodes \(p. 1185\)](#)
- [Ule Component Nodes \(p. 1189\)](#)
- [Ule Animation Node \(p. 1241\)](#)

Ule Canvas Nodes

You can use these flow graph nodes to perform actions on a UI canvas.

Ule:Canvas:ActionListener Node

Listens for the specified action on a UI canvas.

Node Inputs

Activate

Initiates listening for the specified action.

ActionName

Name of the action to listen for.

Node Outputs

OnAction

Triggers when the canvas sends the action.

ElementName

Name of the UI element that triggered the action.

Ule:Canvas:LoadIntoEntity Node

Loads the specified UI canvas.

Node Inputs

Activate

Loads the canvas.

Disabled

Sets whether canvas is disabled initially. If disabled, the canvas is not updated or rendered.

Node Outputs

OnLoad

Sends a signal when the canvas is loaded.

Ule:Canvas:UnloadFromEntity Node

Unloads the specified canvas.

Node Inputs

Activate

Unloads the canvas.

Node Output

Done

Sends a signal when the node's action is finished.

Ule:Canvas:GetKeepLoaded Node

Gets the Boolean value of whether the canvas stays loaded when a level is unloaded.

Node Inputs

Activate

Gets whether the canvas stays loaded when the level is unloaded.

Node Output

KeepLoaded

The Boolean value of whether the canvas stays loaded if the level is unloaded. True if the canvas should stay loaded during level unload; otherwise, false.

UIe:Canvas:SetKeepLoaded Node

Determines whether the canvas stays loaded when a level is unloaded.

Node Inputs

Activate

Sets whether the canvas stays loaded when the level is unloaded.

KeepLoaded

If true, causes the canvas to stay loaded when the level is unloaded.

Node Output

Done

Sends a signal when the node's action is finished.

UIe:Canvas:GetDrawOrder Node

Gets the integer draw order value for a UI canvas with respect to other UI canvases.

Node Inputs

Activate

Gets the draw order for the canvas.

Node Output

DrawOrder

Order in which the canvas draws. Higher numbers appear before lower numbers.

UIe:Canvas:SetDrawOrder Node

Sets the draw order for a UI canvas with respect to other UI canvases.

Node Inputs

Activate

Sets the draw order for the canvas.

DrawOrder

Order in which to display the canvas. Higher numbers appear before lower numbers.

Node Output

Done

Sends a signal when the node's action is finished.

UIe:Canvas:GetsPixelAligned Node

Gets the Boolean value of whether the canvas is pixel-aligned.

Node Inputs

Activate

Gets whether visual element's vertices should snap to the nearest pixel.

Node Output

IsPixelAligned

Boolean value. True if the visual element's vertices should snap to the nearest pixel; otherwise, false.

Ule:Canvas:SetIsPixelAligned Node

Sets whether visual element's vertices should snap to the nearest pixel.

Node Inputs

Activate

Sets the pixel-aligned property for the canvas ID.

IsPixelAligned

Boolean value that represents whether a visual element's vertices should snap to the nearest pixel.

Node Output

Done

Sends a signal when the node's action is finished.

Ule:Canvas:GetEnabled Node

Gets the Boolean `enabled` flag of the canvas. Enabled canvases are updated and each frame rendered.

Node Inputs

Activate

Gets the enabled flag of the canvas.

Node Output

Enabled

The enabled flag of the canvas. True if enabled; otherwise, false.

Ule:Canvas:SetEnabled Node

Sets whether the canvas is enabled. Enabled canvases are updated and each frame rendered.

Node Inputs

Activate

Sets the enabled flag of the canvas.

Enabled

True if the canvas should be enabled; otherwise, false.

Node Output

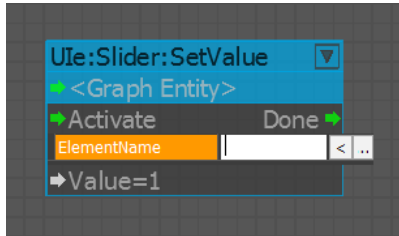
Done

Sends a signal when the node's action is finished.

Ule Component Nodes

The **Ule** component flow graph node set supersedes the original **UI** component flow graph node set (now legacy). The **Ule** node set behaves the same as the original **UI** node set, but simplifies how the nodes associate with UI canvases and UI elements.

You can use these flow graph nodes to perform actions on UI elements through their components.



Each **Ule** component node has an input called **ElementName**. This input represents the name of the UI element in the UI Editor. To edit the **ElementName** input, click the < button (right of the text field) to automatically enter the name of the element that is currently selected in the UI Editor. Click the .. button to launch the UI Editor.

Topics

- [Ule Button Component Nodes \(p. 1189\)](#)
- [Ule Checkbox Component Nodes \(p. 1190\)](#)
- [Ule Element Node \(p. 1194\)](#)
- [Ule Fader Component Nodes \(p. 1194\)](#)
- [Ule Image Component Nodes \(p. 1195\)](#)
- [Ule Interactable Component Nodes \(p. 1198\)](#)
- [Ule Layout Column Component Nodes \(p. 1199\)](#)
- [Ule Layout Grid Component Nodes \(p. 1201\)](#)
- [Ule Layout Row Component Nodes \(p. 1205\)](#)
- [Ule Mask Component Nodes \(p. 1208\)](#)
- [Ule ScrollBox Component Nodes \(p. 1210\)](#)
- [Ule ScrollBar Component Nodes \(p. 1219\)](#)
- [Ule Slider Component Nodes \(p. 1224\)](#)
- [Ule Text Component Nodes \(p. 1229\)](#)
- [Ule Text Input Component Nodes \(p. 1233\)](#)

Ule Button Component Nodes

Use the following flow graph nodes to perform actions on the button component.

Ule:Button:GetActionName Node

Gets the action name string that is emitted when the button is released.

Node Inputs

Activate

Updates the output.

ElementName

Name of the button element.

Node Output

Action

The action name associated with the button.

[Ule:Button:SetActionName Node](#)

Sets the action name string that's emitted when the button is released.

Node Inputs

Activate

Assigns the action name.

ElementName

Name of the button element.

Action

The action name string to assign to the button.

Node Output

Done

Sends a signal when the node's action is finished.

Ule Checkbox Component Nodes

Use the following flow graph nodes to perform actions on the check box component.

[Ule:Checkbox:GetState Node](#)

Gets the Boolean state of the check box.

Node Inputs

Activate

Gets the state of the check box.

ElementName

Name of the check box element.

Node Output

State

Outputs the current Boolean state of the check box.

[Ule:Checkbox:SetState Node](#)

Sets the Boolean state of the check box.

Node Inputs

Activate

Sets the state of the check box.

ElementName

Name of the check box element.

State

The Boolean state of the check box.

Node Output

Done

Sends a signal when the node's action is finished.

[UIe:Checkbox:GetChangedActionName Node](#)

Gets the action triggered when the check box value changed.

Node Inputs

Activate

Gets the changed action name.

ElementName

Name of the check box element.

Node Output

ChangedAction

The action name string value emitted when the check box value changes.

[UIe:Checkbox:SetChangedActionName Node](#)

Sets the action triggered when the check box value changed.

Node Inputs

Activate

Gets the changed action name.

ElementName

Name of the check box element.

ChangedAction

The action name string value emitted when the check box value changes.

[UIe:Checkbox:GetOptionalCheckedEntity Node](#)

Gets the child element to show when the check box is in the `on` state.

Node Inputs

Activate

Updates the output.

ElementName

Name of the check box element.

Node Output

CheckedElement

The child element to show when the check box is selected (in the `on` state).

Ule:Checkbox:SetOptionalCheckedEntity Node

Sets the child element to show when the check box is selected (in the `on` state).

Node Inputs

Activate

Updates the output.

ElementName

Name of the check box element.

CheckedElement

The child element to show when the checkbox is selected (in the `on` state).

Node Output

Done

Sends a signal when the node's action is finished.

Ule:Checkbox:GetOptionalUncheckedEntity Node

Gets the child element to show when the check box is deselected (in the `off` state).

Node Inputs

Activate

Updates the output.

ElementName

Name of the check box element.

Node Output

UncheckedElement

The child element to show when the check box is deselected (off state).

Ule:Checkbox:SetOptionalUncheckedEntity Node

Sets the child element to show when the check box is deselected (in the `off` state).

Node Inputs

Activate

Updates the output.

ElementName

Name of the check box element.

UncheckedElement

The child element to show when the check box is deselected (in the `off` state).

Node Output

Done

Sends a signal when the node's action is finished.

Ule:Checkbox:GetTurnOnActionName Node

Gets the action triggered when the check box is selected.

Node Inputs

Activate

Updates the output.

ElementName

Name of the check box element.

Node Output

TurnOnAction

The action name emitted when the check box is selected (turned on).

Ule:Checkbox:SetTurnOnActionName Node

Sets the action triggered when the check box is selected (turned on).

Node Inputs

Activate

Assigns `TurnOnAction` as the action name that is emitted when the check box is selected.

ElementName

Name of the check box element.

TurnOnAction

The action name emitted when the check box is selected.

Node Output

Done

Sends a signal when the node's action is finished.

Ule:Checkbox:GetTurnOffActionName Node

Gets the action triggered when the check box is deselected (turned off).

Node Inputs

Activate

Update the output.

ElementName

Name of the check box element.

Node Output

TurnOffAction

The action name emitted when the check box is deselected.

Ule:Checkbox:SetTurnOffActionName Node

Sets the action triggered when the check box is deselected (turned off).

Node Inputs

Activate

Assigns `TurnOffAction` as the action name that is emitted when the check box is deselected.

ElementName

Name of the check box element.

TurnOffAction

The action name emitted when the check box is deselected.

Node Output

Done

Sends a signal when the node's action is finished.

Ule Element Node

Use the following flow graph node to enable or disable an element.

Ule:Element:SetIsEnabled Node

Sets the Boolean enabled state of the element. If an element is not enabled, neither it nor any of its children are drawn or interactive.

Node Inputs

Activate

Sets the enabled state to the value of the `State` input.

ElementName

Name of the element.

State

The Boolean enabled state of the element.

Node Output

Done

Sends a signal when the node's action is finished.

Ule Fader Component Nodes

Use the following flow graph nodes to perform actions on the fader component.

Ule:Fader:Animation Node

Animates the fader component on the specified element.

Node Inputs

Activate

Starts a fade animation.

ElementName

Name of the fader element.

StartValue

Value at which the fade starts.

Valid values: 0 = Invisible | 1 = Opaque | -1 = Start from the current value

TargetValue

Value at which the fade ends.

Valid values: 0 = Invisible | 1 = Opaque

Speed

Rate at which the element fades.

Valid values: 0 = Instant fade | 0.5 = Slow fade | 1 = One second fade | 2 = Fade twice as fast

Node Outputs**OnComplete**

Sends a signal when the fade is complete.

OnInterrupted

Sends a signal when the fade is interrupted by another fade starting.

Ule:Fader:GetFadeValue Node

Gets the floating-point fade value of an element.

Node Inputs**Activate**

Updates the output.

ElementName

Name of the fader element.

Node Output**Value**

The floating-point fade value of the element (**ElementID**).

Ule:Fader:SetFadeValue Node

Sets the fade value of an element.

Node Inputs**Activate**

When triggered, assigns **Value** as the fade value of the fader component of the element.

ElementName

Name of the fader element.

Value

The fade value to assign to the fader component for the element.

Node Output**Done**

Sends a signal when the node's action is finished.

Ule Image Component Nodes

Use the following flow graph nodes to perform actions on the image component.

Ule:Image:GetImageSource Node

Replaced by [Ule:Image:GetSprite Node](#) (p. 1196).

Retrieves the texture file path currently used by the specified image element.

Node Inputs

Activate

Updates the output.

ElementName

Name of the image element.

Node Outputs

Value

Outputs the file path of the image that is currently on the element.

Ule:Image:SetImageSource Node

Replaced by [Ule:Image:SetSprite Node \(p. 1196\)](#).

Changes the texture on the specified image element.

Node Inputs

Activate

Set the texture.

ElementName

Name of the image element.

ImagePath

File path of the texture to display.

Node Output

Done

Sends a signal when the node's action is finished.

Ule:Image:GetSprite Node

Gets the texture file path currently used by the specified image element.

Node Inputs

Activate

Updates the output.

ElementName

Name of the image element.

Node Output

Value

Outputs the file path of the image that is currently on the element.

Ule:Image:SetSprite Node

Sets the texture on the specified image element.

Node Inputs

Activate

Sets the texture.

ElementName

Name of the image element.

ImagePath

File path of the texture to display.

Node Output

Done

Sends a signal when the node's action is finished.

Ule:Image:GetImageType Node

Gets the type of the image. Affects how the texture or sprite is mapped to the image rectangle.

Node Inputs

Activate

Updates the output.

ElementName

Name of the image element.

Node Output

ImageType

An integer representing how the image is scaled and placed.

Valid values: 0 = Stretched | 1 = Sliced | 2 = Fixed | 3 = Tiled | 4 = Stretched to fit | 5 = Stretched to fill

Ule:Image:SetImageType Node

Sets the type of the image. Affects how the texture or sprite is mapped to the image rectangle.

Node Inputs

Activate

Updates the output.

ElementName

Name of the image element.

ImageType

An integer representing how the image is scaled and placed.

Valid values: 0 = Stretched | 1 = Sliced | 2 = Fixed | 3 = Tiled | 4 = Stretched to fit | 5 = Stretched to fill

Node Output

Done

Sends a signal when the node's action is finished.

Ule:Image:GetColor Node

Gets the color tint for the image.

Node Inputs

Activate

Updates the output.

ElementName

Name of the image element.

Node Outputs

Color

The RGB value (0 – 255 each for R, G, and B) of the element (**ElementID**).

Alpha

The alpha value (0 – 255) of the element (**ElementID**).

Ule:Image:SetColor Node

Sets the color tint for the image.

Node Inputs

Activate

Updates the output.

ElementName

Name of the image element.

Color

The RGB value (0 – 255 each for R, G, and B).

Alpha

A floating-point alpha value (0 – 255).

Node Output

Done

Sends a signal when the node's action is finished.

Ule Interactable Component Nodes

Use the following flow graph node for the **Interactable** component.

Ule:Interactable:SetIsHandlingEvents Node

Sets the Boolean "is handling events" state of the element.

The **Interactable** flow graph nodes can be used to get or set values on any interactive UI element.

Interactive UI elements are elements that players can interacted with in game, such as button, text input, check box, slider, and so on. The **SetIsHandlingEvents** flow graph node sets whether an interactive UI element should handle input events. If set to false, then the UI element does not respond to input events, and its visual state is also changed to disabled.

Node Inputs

Activate

Sets the "is handling events" state.

ElementName

Name of the element.

State

The Boolean "is handling events" state of the element.

Node Output

Done

Sends a signal when the node's action is finished.

Ule Layout Column Component Nodes

Use the following flow graph nodes to perform actions on the layout column component.

Ule:LayoutColumn:GetOrder Node

Gets the vertical order of the **LayoutColumn** component for an element.

Node Inputs

Activate

Updates the output.

ElementName

Name of the element.

Node Output

Order

An integer representing the vertical order.

Valid values: 0 = Top to bottom | 1 = Bottom to top

Ule:LayoutColumn:SetOrder Node

Sets the vertical order of the **LayoutColumn** component for an element.

Node Inputs

Activate

Sets the vertical order for the element.

ElementName

Name of the element.

Order

An integer representing the vertical order. 0 = Top to bottom | 1 = Bottom to top.

Node Output

Done

Sends a signal when the node's action is finished.

Ule:LayoutColumn:GetPadding Node

Gets the padding (in pixels) inside the edges of the **LayoutColumn** component for an element.

Node Inputs

Activate

Updates the outputs.

ElementName

Name of the element.

Node Outputs

Left

An integer representing the padding inside the left edge of the element.

Right

An integer representing the padding inside the right edge of the element.

Top

An integer representing the padding inside the top edge of the element.

Bottom

An integer representing the padding inside the bottom edge of the element.

Ule:LayoutColumn:SetPadding Node

Sets the padding (in pixels) inside the edges of the `LayoutColumn` component for an element.

Node Inputs

Activate

Updates the outputs.

ElementName

Name of the element.

Left

An integer representing the padding inside the left edge of the element.

Right

An integer representing the padding inside the right edge of the element.

Top

An integer representing the padding inside the top edge of the element.

Bottom

An integer representing the padding inside the bottom edge of the element.

Node Output

Done

Sends a signal when the node's action is finished.

Ule:LayoutColumn:GetSpacing Node

Gets the spacing (in pixels) between child elements of the `LayoutColumn` component for an element.

Node Inputs

Activate

Updates the output.

ElementName

Name of the element.

Node Output

Spacing

A float value of the spacing (in pixels) between child elements of the element (**ElementName**).

Ule:LayoutColumn:SetSpacing Node

Sets the spacing (in pixels) between child elements of the **LayoutColumn** component for an element.

Node Inputs

Activate

Updates the output.

ElementName

Name of the element.

Spacing

A float value of the spacing (in pixels) between child elements of the element (**ElementName**).

Node Output

Done

Sends a signal when the node's action is finished.

Ule Layout Grid Component Nodes

Use the following flow graph nodes to perform actions on the layout grid component.

Ule:LayoutGrid:GetCellSize Node

Gets the size (in pixels) of a child element in the layout.

Node Inputs

Activate

Updates the output.

ElementName

Name of the element.

Node Outputs

CellWidth

The width (in pixels) of a child element of element (**ElementID**).

CellHeight

The height (in pixels) of a child element of element (**ElementID**).

Ule:LayoutGrid:SetCellSize Node

Sets the size (in pixels) of a child element in the layout.

Node Inputs

Activate

Updates the output.

ElementName

Name of the element.

CellWidth

The width (in pixels) of a child element of element (**ElementID**).

CellHeight

The height (in pixels) of a child element of element (**ElementID**).

Node Output

Done

Sends a signal when the node's action is finished.

Ule:LayoutGrid:GetHorizontalOrder Node

Gets the horizontal order for the layout.

Node Inputs

Activate

Updates the outputs.

ElementName

Name of the element.

Node Output

Order

An integer representing the horizontal order.

Valid values: 0 = Left to right | 1 = Right to left

Ule:LayoutGrid:SetHorizontalOrder Node

Sets the horizontal order for the layout.

Node Inputs

Activate

Updates the outputs.

ElementName

Name of the element.

Order

An integer representing the horizontal order.

Valid values: 0 = Left to right | 1 = Right to left

Node Output

Done

Sends a signal when the node's action is finished.

Ule:LayoutGrid:GetPadding Node

Gets the padding (in pixels) inside the edges of the **LayoutGrid** component for an element.

Node Inputs

Activate

Updates the output.

ElementName

Name of the element.

Node Output

Left

An integer representing the padding inside the left edge of the element.

Right

An integer representing the padding inside the right edge of the element.

Top

An integer representing the padding inside the top edge of the element.

Bottom

An integer representing the padding inside the bottom edge of the element.

Ule:LayoutGrid:SetPadding Node

Sets the padding (in pixels) inside the edges of the **LayoutGrid** component for an element.

Node Inputs

Activate

Updates the output.

ElementName

Name of the element.

Left

An integer representing the padding inside the left edge of the element.

Right

An integer representing the padding inside the right edge of the element.

Top

An integer representing the padding inside the top edge of the element.

Bottom

An integer representing the padding inside the bottom edge of the element.

Node Output

Done

Sends a signal when the node's action is finished.

Ule:LayoutGrid:GetSpacing Node

Gets the spacing (in pixels) between child elements of the **LayoutGrid** component for an element.

Node Inputs

Activate

Updates the output.

ElementName

Name of the element.

Node Output

Spacing

A float value of the spacing (in pixels) between child elements of the element (**ElementID**).

Ule:LayoutGrid:SetSpacing Node

Sets the spacing (in pixels) between child elements of the **LayoutGrid** component for an element.

Node Inputs

Activate

Updates the output.

ElementName

Name of the element.

Spacing

A float value of the spacing (in pixels) between child elements of the element (**ElementID**).

Node Output

Done

Sends a signal when the node's action is finished.

Ule:LayoutGrid:GetStartingDirection Node

Gets the starting direction for the layout.

Node Inputs

Activate

Updates the output.

ElementName

Name of the element.

Node Output

Direction

An integer representing the direction.

Valid values: 0 = Horizontal order | 1 = Vertical order

Ule:LayoutGrid:SetStartingDirection Node

Sets the starting direction for the layout.

Node Inputs

Activate

Set the starting direction for the layout.

ElementName

Name of the element.

Direction

An integer representing the horizontal order.

Valid values: 0 = Horizontal order | 1 = Vertical order.

Node Output

Done

Sends a signal when the node's action is finished.

Ule:LayoutGrid:GetVerticalOrder Node

Gets the vertical order for the layout.

Node Inputs

Activate

Updates the outputs.

ElementName

Name of the element.

Node Output

Action

An integer representing the vertical order.

Valid values: 0 = Top to bottom | 1 = Bottom to top

Ule:LayoutGrid:SetVerticalOrder Node

Sets the vertical order for the layout.

Node Inputs

Activate

Sets the vertical order for the layout.

ElementName

Name of the element.

Action

An integer representing the vertical order.

Valid values: 0 = Top to bottom | 1 = Bottom to top

Node Output

Done

Sends a signal when the node's action is finished.

Ule Layout Row Component Nodes

Use the following flow graph nodes to perform actions on the layout row component.

Ule:LayoutRow:GetOrder Node

Gets the horizontal order of the **LayoutRow** component for an element.

Node Inputs

Activate

Updates the output.

ElementName

Name of the element.

Node Output

Order

An integer representing the horizontal order.

Valid values: 0 = Left to right | 1 = Right to left

[Ule:LayoutRow:SetOrder Node](#)

Sets the horizontal order of the **LayoutRow** component for an element.

Node Inputs

Activate

Updates the output.

ElementName

Name of the element.

Order

An integer representing the horizontal order.

Valid values: 0 = Left to right | 1 = Right to left

Node Output

Done

Sends a signal when the node's action is finished.

[Ule:LayoutRow:GetPadding Node](#)

Gets the padding (in pixels) inside the edges of the **LayoutRow** component for an element.

Node Inputs

Activate

Updates the output.

ElementName

Name of the element.

Node Outputs

Left

An integer representing the padding inside the left edge of the element.

Right

An integer representing the padding inside the right edge of the element.

Top

An integer representing the padding inside the top edge of the element.

Bottom

An integer representing the padding inside the bottom edge of the element.

[Ule:LayoutRow:SetPadding Node](#)

Sets the padding (in pixels) inside the edges of the **LayoutRow** component for an element.

Node Inputs

Activate

Sets the padding (in pixels) inside the edges of the LayoutRow.

ElementName

Name of the element.

Left

An integer representing the padding inside the left edge of the element.

Right

An integer representing the padding inside the right edge of the element.

Top

An integer representing the padding inside the top edge of the element.

Bottom

An integer representing the padding inside the bottom edge of the element.

Node Output

Done

Sends a signal when the node's action is finished.

Ule:LayoutRow:GetSpacing Node

Gets the spacing (in pixels) between child elements of the LayoutRow component for an element.

Node Inputs

Activate

Updates the outputs.

ElementName

Name of the element.

Node Output

Spacing

A float value of the spacing (in pixels) between child elements of the element (**ElementName**).

Ule:LayoutRow:SetSpacing Node

Sets the spacing (in pixels) between child elements of the LayoutRow component for an element.

Node Inputs

Activate

Sets the spacing (in pixels) between child elements.

ElementName

Name of the element.

Spacing

A float value of the spacing (in pixels) between child elements of the element (**ElementName**).

Node Output

Done

Sends a signal when the node's action is finished.

UIe Mask Component Nodes

Use the following flow graph nodes to perform actions on the mask component.

UIe:Mask:GetDrawBehind Node

Gets whether mask is drawn behind the child elements.

Node Inputs

Activate

Updates the outputs.

ElementName

Name of the element.

Node Output

DrawBehind

Indicates whether mask is drawn behind the child elements.

UIe:Mask:SetDrawBehind Node

Sets whether mask is drawn behind the child elements.

Node Inputs

Activate

Sets whether mask is drawn behind the child elements.

ElementName

Name of the element.

DrawBehind

Sets whether mask is drawn behind the child elements.

Node Output

Done

Sends a signal when the node's action is finished.

UIe:Mask:GetDrawInFront Node

Gets whether mask is drawn in front of child elements.

Node Inputs

Activate

Updates the outputs.

ElementName

Name of the element.

Node Output

DrawInFront

Indicates whether mask is drawn in front of child elements.

Ule:Mask:SetDrawInFront Node

Sets whether mask is drawn in front of child elements.

Node Inputs

Activate

Sets whether mask is drawn in front of child elements.

ElementName

Name of the element.

DrawInFront

Sets whether mask is drawn in front of child elements.

Node Output

Done

Sends a signal when the node's action is finished.

Ule:Mask:GetIsMaskingEnabled Node

Gets whether masking is enabled.

Node Inputs

Activate

Updates the outputs.

ElementName

Name of the element.

Node Output

IsMaskingEnabled

Indicates whether masking is enabled.

Ule:Mask:SetIsMaskingEnabled Node

Sets whether masking is enabled.

Node Inputs

Activate

Sets whether masking is enabled.

ElementName

Name of the element.

IsMaskingEnabled

Sets whether masking is enabled.

Node Output

Done

Sends a signal when the node's action is finished.

Ule:Mask:GetUseAlphaTest Node

Gets whether to use the alpha channel in the mask visual's texture to define the mask.

Node Inputs

Activate

Updates the outputs.

ElementName

Name of the element.

Node Output

UseAlphaTest

Indicates whether to use the alpha channel in the mask visual's texture to define the mask.

Ule:Mask:SetUseAlphaTest Node

Sets whether to use the alpha channel in the mask visual's texture to define the mask.

Node Inputs

Activate

Sets whether to use the alpha channel in the mask visual's texture to define the mask.

ElementName

Name of the element.

UseAlphaTest

Sets whether to use the alpha channel in the mask visual's texture to define the mask.

Node Output

Done

Sends a signal when the node's action is finished.

Ule ScrollBox Component Nodes

Use the following flow graph nodes to perform actions on the **ScrollBox** component.

Ule:ScrollBox:FindClosestContentChildElement Node

Finds the child of the content element that is closest to the content anchors.

Node Inputs

Activate

Updates the outputs.

ElementName

Name of the element.

Node Output

ClosestElement

The element currently closest to the focused element.

Ule:ScrollBox:GetContentEntity Node

Gets the content element for the **ScrollBox**.

Node Inputs

Activate

Updates the outputs.

ElementName

Name of the element.

Node Output

Content

The element that the **ScrollBar** scrolls.

Ule:ScrollBar:SetContentEntity Node

Sets the content element for the **ScrollBar**.

Node Inputs

Activate

Sets the content element for the **ScrollBar**.

ElementName

Name of the element.

Content

The element that the **ScrollBar** scrolls.

Node Output

Done

Sends a signal when the node's action is finished.

Ule:ScrollBar:GetIsHorizontalScrollingEnabled Node

Gets whether the **ScrollBar** allows horizontal scrolling.

Node Inputs

Activate

Updates the outputs.

ElementName

Name of the element.

Node Output

Enabled

Indicates whether horizontal scrolling is enabled.

Ule:ScrollBar:SetIsHorizontalScrollingEnabled Node

Sets whether the **ScrollBar** allows horizontal scrolling.

Node Inputs

Activate

Sets whether the **ScrollBar** allows horizontal scrolling.

ElementName

Name of the element.

Enabled

Sets whether horizontal scrolling is enabled.

Node Output

Done

Sends a signal when the node's action is finished.

[UIe:ScrollBox:GetIsScrollingConstrained Node](#)

Gets whether the **ScrollBox** restricts scrolling to the content area.

Node Inputs

Activate

Updates the outputs.

ElementName

Name of the element.

Node Output

IsConstrained

Indicates whether scrolling is constrained.

[UIe:ScrollBox:SetIsScrollingConstrained Node](#)

Sets whether the **ScrollBox** restricts scrolling to the content area.

Node Inputs

Activate

Sets whether the **ScrollBox** restricts scrolling to the content area.

ElementName

Name of the element.

IsConstrained

Sets whether scrolling is constrained.

Node Output

Done

Sends a signal when the node's action is finished.

[UIe:ScrollBox:GetIsVerticalScrollingEnabled Node](#)

Gets whether the **ScrollBox** allows vertical scrolling.

Node Inputs

Activate

Updates the output.

ElementName

Name of the element.

Node Output

Enabled

Indicates whether vertical scrolling is enabled.

Ule:ScrollBar:SetIsVerticalScrollingEnabled Node

Sets whether the **ScrollBar** allows vertical scrolling.

Node Inputs

Activate

Sets whether the **ScrollBar** allows vertical scrolling.

ElementName

Name of the element.

Enabled

Sets whether vertical scrolling is enabled.

Node Output

Done

Sends a signal when the node's action is finished.

Ule:ScrollBar:GetScrollOffset Node

Gets the scroll offset of the **ScrollBar**.

Node Inputs

Activate

Updates the outputs.

ElementName

Name of the element.

Node Output

HorizOffset

The horizontal scroll offset of the element identified by **ElementName**.

VertOffset

The vertical scroll offset of the element identified by **ElementName**.

Ule:ScrollBar:SetScrollOffset Node

Sets the scroll offset of the **ScrollBar**.

Node Inputs

Activate

Sets the scroll offset of the **ScrollBar**.

ElementName

Name of the element.

HorizOffset

The horizontal scroll offset of **ElementName**.

VertOffset

The vertical scroll offset of **ElementName**.

Node Output

Done

Sends a signal when the node's action is finished.

Ule:ScrollBar:GetScrollOffsetChangedActionName Node

Gets the action triggered when the **ScrollBar** drag is completed.

Node Inputs

Activate

Updates the output.

ElementName

Name of the element.

Node Output

ChangedAction

The action name.

Ule:ScrollBar:SetScrollOffsetChangedActionName Node

Sets the action triggered when the **ScrollBar** drag is completed.

Node Inputs

Activate

Sets the action triggered when the **ScrollBar** drag is completed.

ElementName

Name of the element.

ChangedAction

The action name.

Node Output

Done

Sends a signal when the node's action is finished.

Ule:ScrollBar:GetScrollOffsetChangingActionName Node

Gets the action triggered while the **ScrollBar** is being dragged.

Node Inputs

Activate

Updates the output.

ElementName

Name of the element.

Node Output

ChangingAction

The action name.

Ule:ScrollBar:SetScrollOffsetChangingActionName Node

Sets the action triggered while the **ScrollBar** is being dragged.

Node Inputs

Activate

Sets the action triggered while the **ScrollBar** is being dragged.

ElementName

Name of the element.

ChangingAction

The action name.

Node Output

Done

Sends a signal when the node's action is finished.

Ule:ScrollBar:GetSnapGrid Node

Gets the snapping grid of the **ScrollBar**.

Node Inputs

Activate

Updates the outputs.

ElementName

Name of the element.

Node Outputs

HorizSpacing

The horizontal grid spacing of the element identified by **ElementName**.

VertSpacing

The vertical grid spacing of the element identified by **ElementName**.

Ule:ScrollBar:SetSnapGrid Node

Sets the snapping grid of the **ScrollBar**.

Node Inputs

Activate

Sets the snapping grid of the **ScrollBar**.

ElementName

Name of the element.

HorizSpacing

The horizontal grid spacing of the element identified by **ElementName**.

VertSpacing

The vertical grid spacing of the element identified by **ElementName**.

Node Output

Done

Sends a signal when the node's action is finished.

Ule:ScrollBar:GetSnapMode Node

Gets the snap mode for the **ScrollBar**.

Node Inputs

Activate

Updates the output.

ElementName

Name of the element.

Node Output

SnapMode

An integer representing the snap mode state.

Valid values: 0 = None | 1 = Children | 2 = Grid

Ule:ScrollBar:SetSnapMode Node

Sets the snap mode for the **ScrollBar**.

Node Inputs

Activate

Sets the snap mode for the **ScrollBar**.

ElementName

Name of the element.

SnapMode

An integer representing the snap mode state.

Valid values: 0 = None | 1 = Children | 2 = Grid

Node Output

Done

Sends a signal when the node's action is finished.

Ule:ScrollBar:GetHorizontalScrollBarVisibility Node

Gets horizontal scrollbar visibility behavior.

Node Inputs

Activate

Updates the outputs.

CanvasID

Unique identifier of the element's canvas.

ElementID

Unique identifier of the element.

Node Output

ScrollBarVisibility

An integer that represents the scrollbar visibility behavior.

Valid values: 0 = AlwaysVisible | 1 = AutoHide | 2 = AutoHideAndResizeViewArea

Ule:ScrollBox:SetHorizontalScrollBarVisibility Node

Sets horizontal scrollbar visibility behavior.

Node Inputs

Activate

Sets horizontal scroll bar visibility behavior.

CanvasID

Unique identifier of the element's canvas.

ElementID

Unique identifier of the element.

ScrollBarVisibility

An integer representing the scrollbar visibility behavior.

0 = AlwaysVisible | 1 = AutoHide | 2 = AutoHideAndResizeViewArea.

Node Output

Done

Sends a signal when the node's action is finished.

Ule:ScrollBox:GetVerticalScrollBarVisibility Node

Gets vertical scrollbar visibility behavior.

Node Inputs

Activate

Updates the outputs.

CanvasID

Unique identifier of the element's canvas.

ElementID

Unique identifier of the element.

Node Output

ScrollBarVisibility

An integer that represents the scrollbar visibility behavior.

Valid values: 0 = AlwaysVisible | 1 = AutoHide | 2 = AutoHideAndResizeViewArea

Ule:ScrollBox:SetVerticalScrollBarVisibility Node

Sets vertical scrollbar visibility behavior.

Node Inputs

Activate

Sets vertical scroll bar visibility behavior.

CanvasID

Unique identifier of the element's canvas.

ElementID

Unique identifier of the element.

ScrollBarVisibility

An integer representing the scrollbar visibility behavior.

0 = AlwaysVisible | 1 = AutoHide | 2 = AutoHideAndResizeViewArea.

Node Output

Done

Sends a signal when the node's action is finished.

[Ule:ScrollBar:GetHorizontalScrollBarEntity Node](#)

Gets the horizontal scrollbar element for the **ScrollBar**.

Node Inputs

Activate

Updates the outputs.

CanvasID

Unique identifier of the element's canvas.

ElementID

Unique identifier of the element.

Node Output

HorizontalScrollBar

The element that scrolls the **ScrollBar** horizontally.

[Ule:ScrollBar:SetHorizontalScrollBarEntity Node](#)

Sets the horizontal scrollbar element for the **ScrollBar**.

Node Inputs

Activate

Sets the horizontal scrollbar element for the **ScrollBar**.

CanvasID

Unique identifier of the element's canvas.

ElementID

Unique identifier of the element.

HorizontalScrollBar

The element that scrolls the **ScrollBar** horizontally.

Node Output

Done

Sends a signal when the node's action is finished.

[Ule:ScrollBar:GetVerticalScrollBarEntity Node](#)

Gets the vertical scrollbar element for the **ScrollBar**.

Node Inputs

Activate

Updates the outputs.

CanvasID

Unique identifier of the element's canvas.

ElementID

Unique identifier of the element.

Node Output

VerticalScrollBar

The element that scrolls the **ScrollBar** vertically.

Ule:ScrollBar:SetVerticalScrollBarEntity Node

Sets the vertical scrollbar element for the **ScrollBar**.

Node Inputs

Activate

Sets the vertical scrollbar element for the **ScrollBar**.

CanvasID

Unique identifier of the element's canvas.

ElementID

Unique identifier of the element.

VerticalScrollBar

The element that scrolls the **ScrollBar** vertically.

Node Output

Done

Sends a signal when the node's action is finished.

Ule ScrollBar Component Nodes

Use the following flow graph nodes to perform actions on the **Scrollbar** component.

Ule:Scrollbar:GetHandleEntity Node

Gets the handle element of the scroll bar.

Node Inputs

Activate

Updates the output.

CanvasID

Unique identifier of the element's canvas.

ElementName

Name of the element.

Node Output

Handle

The handle element.

Ule:Scrollbar:SetHandleEntity Node

Sets the handle element of the scroll bar.

Node Inputs

Activate

Sets the handle element.

CanvasID

Unique identifier of the element's canvas.

ElementName

Name of the element.

Handle

The handle element.

Node Output

Done

Sends a signal when the node's action is finished.

Ule:Scrollbar:GetValue Node

Gets the value of the scrollbar.

Node Inputs

Activate

Updates the output.

CanvasID

Unique identifier of the element's canvas.

ElementName

Name of the element.

Node Output

Value

The scrollbar value of the element identified by `ElementName`.

Ule:Scrollbar:SetValue Node

Sets the value of the scroll bar.

Node Inputs

Activate

Sets the value of the scrollbar.

CanvasID

Unique identifier of the element's canvas.

ElementName

Name of the element.

Value

The scrollbar value of the element identified by `ElementName`.

Node Output**Done**

Sends a signal when the node's action is finished.

Ule:Scrollbar:GetHandleSize Node

Gets the size of the handle.

Node Inputs**Activate**

Updates the output.

CanvasID

Unique identifier of the element's canvas.

ElementName

Name of the element.

Node Output**HandleSize**

The size of the handle of the element identified by `ElementName`.

Ule:Scrollbar:SetHandleSize Node

Sets the size of the handle.

Node Inputs**Activate**

Sets the size of the handle.

CanvasID

Unique identifier of the element's canvas.

ElementName

Name of the element.

HandleSize

The size of the handle of the element identified by `ElementName`.

Node Output**Done**

Sends a signal when the node's action is finished.

Ule:Scrollbar:GetMinHandlePixelSize Node

Gets the minimum size in pixels of the handle.

Node Inputs**Activate**

Updates the output.

CanvasID

Unique identifier of the element's canvas.

ElementName

Name of the element.

Node Output**MinHandleSize**

The minimum size in pixels of the handle of the element identified by `ElementName`.

Ule:Scrollbar:SetMinHandlePixelSize Node

Sets the minimum size in pixels of the handle.

Node Inputs**Activate**

Sets the minimum size in pixels of the handle.

CanvasID

Unique identifier of the element's canvas.

ElementName

Name of the element.

MinHandleSize

The minimum size in pixels of the handle of the element identified by `ElementName`.

Node Output**Done**

Sends a signal when the node's action is finished.

Ule:Scrollbar:GetValueChangedActionName Node

Gets the action triggered when the value is done changing.

Node Inputs**Activate**

Updates the output.

CanvasID

Unique identifier of the element's canvas.

ElementName

Name of the element.

Node Output**ValueChangedAction**

The action name.

Ule:Scrollbar:SetValueChangedActionName Node

Sets the action triggered when the value is done changing.

Node Inputs

Activate

Updates the output.

CanvasID

Unique identifier of the element's canvas.

ElementName

Name of the element.

ValueChangedAction

The action name.

Node Output

Done

Sends a signal when the node's action is finished.

Ule:Scrollbar:GetValueChangingActionName Node

Gets the action triggered while the value is changing.

Node Inputs

Activate

Updates the output.

CanvasID

Unique identifier of the element's canvas.

ElementName

Name of the element.

Node Output

ValueChangingAction

The action name.

Ule:Scrollbar:SetValueChangingActionName Node

Sets the action triggered while the value is changing.

Node Inputs

Activate

Updates the output.

CanvasID

Unique identifier of the element's canvas.

ElementName

Name of the element.

ValueChangingAction

The action name.

Node Output

Done

Sends a signal when the node's action is finished.

Ule Slider Component Nodes

Use the following flow graph nodes to perform actions on the slider component.

Ule:Slider:GetFillEntity Node

Gets the fill element.

Node Inputs

Activate

Updates the output.

ElementName

Name of the element.

Node Output

FillElement

The fill element.

Ule:Slider:SetFillEntity Node

Sets the fill element.

Node Inputs

Activate

Sets the fill element.

ElementName

Name of the element.

FillElement

The fill element.

Node Output

Done

Sends a signal when the node's action is finished.

Ule:Slider:GetManipulatorEntity Node

Gets the manipulator element.

Node Inputs

Activate

Updates the output.

ElementName

Name of the element.

Node Output

ManipulatorElement

The manipulator element.

UIe:Slider:SetManipulatorEntity Node

Sets the manipulator element.

Node Inputs

Activate

Sets the manipulator element.

ElementName

Name of the element.

ManipulatorElement

The manipulator element.

Node Output

Done

Sends a signal when the node's action is finished.

UIe:Slider:GetMaxValue Node

Gets the maximum value of the slider.

Node Inputs

Activate

Updates the output.

ElementName

Name of the element.

Node Output

MaxValue

The slider maximum value of the element identified by **ElementName**.

UIe:Slider:SetMaxValue Node

Sets the maximum value of the slider.

Node Inputs

Activate

Sets the maximum value of the slider.

ElementName

Name of the element.

MaxValue

The slider maximum value of the element identified by **ElementName**.

Node Output

Done

Sends a signal when the node's action is finished.

UIe:Slider:GetMinValue Node

Gets the minimum value of the slider.

Node Inputs

Activate

Updates the output.

ElementName

Name of the element.

Node Output

MinValue

The slider minimum value of the element identified by **ElementName**.

Ule:Slider:SetMinValue Node

Sets the minimum value of the slider.

Node Inputs

Activate

Sets the minimum value of the slider.

ElementName

Name of the element.

MinValue

The slider minimum value of the element identified by **ElementName**.

Node Output

Done

Sends a signal when the node's action is finished.

Ule:Slider:GetStepValue Node

Gets the smallest increment allowed between values. Zero means no restriction.

Node Inputs

Activate

Updates the output.

ElementName

Name of the element.

Node Output

StepValue

The smallest increment allowed between values of the element identified by **ElementName**. Zero means no restriction.

Ule:Slider:SetStepValue Node

Sets the smallest increment allowed between values. Zero means no restriction.

Node Inputs

Activate

Sets the smallest increment allowed between values. Zero means no restriction.

ElementName

Name of the element.

StepValue

The smallest increment allowed between values of the element identified by **ElementName**. Zero means no restriction.

Node Output

Done

Sends a signal when the node's action is finished.

Ule:Slider:GetTrackEntity Node

Gets the track element.

Node Inputs

Activate

Updates the output.

ElementName

Name of the element.

Node Output

Track

The track element.

Ule:Slider:SetTrackEntity Node

Sets the track element.

Node Inputs

Activate

Sets the track element.

ElementName

Name of the element.

Track

The track element.

Node Output

Done

Sends a signal when the node's action is finished.

Ule:Slider:GetValue Node

Gets the value of slider.

Node Inputs

Activate

Updates the output.

ElementName

Name of the element.

Node Output

Value

The slider value of the element identified by **ElementName**.

Ule:Slider:SetValue Node

Sets the value of the slider.

Node Inputs

Activate

Sets the value of the slider.

ElementName

Name of the element.

Value

The slider value of the element identified by **ElementName**.

Node Output

Done

Sends a signal when the node's action is finished.

Ule:Slider:GetValueChangedActionName Node

Gets the action triggered when the value is done changing.

Node Inputs

Activate

Updates the output.

ElementName

Name of the element.

Node Output

ValueChangedAction

The action name.

Ule:Slider:SetValueChangedActionName Node

Sets the action triggered when the value is done changing.

Node Inputs

Activate

Updates the output.

ElementName

Name of the element.

ValueChangedAction

The action name.

Node Output

Done

Sends a signal when the node's action is finished.

Ule:Slider:GetValueChangingActionName Node

Gets the action triggered while the value is changing.

Node Inputs

Activate

Updates the output.

ElementName

Name of the element.

Node Output

ValueChangingAction

The action name.

Ule:Slider:SetValueChangingActionName Node

Sets the action triggered while the value is changing.

Node Inputs

Activate

Updates the output.

ElementName

Name of the element.

ValueChangingAction

The action name.

Node Output

Done

Sends a signal when the node's action is finished.

Ule Text Component Nodes

Use the following flow graph nodes to perform actions on the text component.

Ule:Text:GetColor Node

Gets the color to draw the text string.

Node Inputs

Activate

Updates the outputs.

ElementName

Name of the element.

Node Outputs

Color

The RGB value (0 – 255 each for R, G, and B) of the element identified by **ElementName**.

Alpha

The alpha value (0 – 255) of the element identified by **ElementName**.

Ule:Text:SetColor Node

Sets the color to draw the text string.

Node Inputs

Activate

Updates the output.

ElementName

Name of the element.

Color

The RGB value (0 – 255 each for R, G, and B) of the element identified by **ElementName**.

Alpha

The alpha value (0 – 255) of the element identified by **ElementName**.

Node Output

Done

Sends a signal when the node's action is finished.

Ule:Text:GetFont Node

Gets the path to the font.

Node Inputs

Activate

Updates the output.

ElementName

Name of the element.

Node Output

Font

The path to the font used by the element.

Ule:Text:SetFontNode

Sets the path to the font.

Node Inputs

Activate

Updates the output.

ElementName

Name of the element.

Font

The path to the font used by the element identified by **ElementName**.

Node Output

Done

Sends a signal when the node's action is finished.

Ule:Text:GetFontSize Node

Gets the font size in points.

Node Inputs

Activate

Updates the output.

ElementName

Name of the element.

Node Output

FontSize

The font size of the element identified by **ElementName**.

Ule:Text:SetFontSize Node

Sets the font size in points.

Node Inputs

Activate

Updates the output.

ElementName

Name of the element.

FontSize

The font size of the element identified by **ElementName**.

Node Output

Done

Sends a signal when the node's action is finished.

Ule:Text:GetOverflowMode Node

Gets the overflow behavior of the text.

Node Inputs

Activate

Updates the output.

ElementName

Name of the element.

Node Output

OverflowMode

An integer representing how overflow text is handled.

Valid values: 0 = Overflow text | 1 = Clip text

Ule:Text:SetOverflowModeNode

Sets the overflow behavior of the text.

Node Inputs

Activate

Updates the output.

ElementName

Name of the element.

OverflowMode

An integer representing how overflow text is handled.

Valid values: 0 = Overflow text | 1 = Clip text

Node Output

Done

Sends a signal when the node's action is finished.

Ule:Text:GetText Node

Gets the text string that the element displays.

Node Inputs

Activate

Updates the output.

ElementName

Name of the element.

Node Output

Value

The text string being displayed by the element identified by **ElementName**.

Ule:Text:SetText Node

Sets the text string being displayed by the element.

Node Inputs

Activate

Updates the output.

ElementName

Name of the element.

Value

The text string being displayed by the element identified by **ElementName**.

Node Output

Done

Sends a signal when the node's action is finished.

Ule:Text:GetWrapText Node

Gets whether text is wrapped.

Node Inputs

Activate

Updates the output.

ElementName

Name of the element.

Node Output

WrapTextSetting

An integer representing how long text lines are handled.

Valid values: 0 = No wrap | 1 = Wrap

Ule:Text:SetWrapText Node

Gets whether text is wrapped.

Node Inputs

Activate

Updates the outputs.

ElementName

Name of the element.

WrapTextSetting

An integer representing how long text lines are handled.

Valid values: 0 = No wrap | 1 = Wrap

Node Output

Done

Sends a signal when the node's action is finished.

Ule Text Input Component Nodes

Use the following flow graph nodes to perform actions on the text input component.

Ule:TextInput:GetChangeAction Node

Gets the action triggered when the text is changed.

Node Inputs

Activate

Updates the output.

ElementName

Name of the element.

Node Output

ChangeAction

The action name.

Ule:TextInput:SetChangeAction Node

Sets the action triggered when the text is changed.

Node Inputs

Activate

Updates the output.

ElementName

Name of the element.

ChangeAction

The action name.

Node Output

Done

Sends a signal when the node's action is finished.

Ule:TextInput:GetCursorBlinkInterval Node

Gets the cursor blink interval of the text input.

Node Inputs

Activate

Updates the output.

ElementName

Name of the element.

Node Output

CursorBlinkInterval

The cursor blink in interval of the element identified by **ElementName**.

Ule:TextInput:SetCursorBlinkInterval Node

Gets the cursor blink interval of the text input.

Node Inputs

Activate

Updates the output.

ElementName

Name of the element.

CursorBlinkInterval

The cursor blink in interval of the element identified by **ElementName**.

Node Output

Done

Sends a signal when the node's action is finished.

Ule:TextInput:GetEndEditAction Node

Gets the action triggered when the editing of text is finished.

Node Inputs

Activate

Updates the output.

ElementName

Name of the element.

Node Output

EndEditAction

The action name.

Ule:TextInput:SetEndEditAction Node

Sets the action triggered when the editing of text is finished.

Node Inputs

Activate

Updates the output.

ElementName

Name of the element.

EndEditAction

The action name.

Node Output

Done

Sends a signal when the node's action is finished.

Ule:TextInput:GetEnterAction Node

Gets the action triggered when **Enter** is pressed.

Node Inputs

Activate

Updates the output.

ElementName

Name of the element.

Node Output

EnterAction

The action name.

UIe:TextInput:SetEnterAction Node

Sets the action triggered when **Enter** is pressed.

Node Inputs

Activate

Updates the output.

ElementName

Name of the element.

EnterAction

The action name.

Node Output

Done

Sends a signal when the node's action is finished.

UIe:TextInput:GetIsPasswordField Node

Gets whether the text input is configured as a password field.

Node Inputs

Activate

Updates the output.

ElementName

Name of the element.

Node Output

IsPasswordField

Boolean. Whether the element identified by **ElementName** is configured as a password field.

UIe:TextInput:SetIsPasswordField Node

Sets whether the text input is configured as a password field.

Node Inputs

Activate

Updates the output.

ElementName

Name of the element.

IsPasswordField

Boolean. Whether the element identified by **ElementName** is configured as a password field.

Node Output

Done

Sends a signal when the node's action is finished.

UIe:TextInput:GetMaxStringLength Node

Gets the maximum number of characters that can be entered.

Node Inputs

Activate

Updates the output.

ElementName

Name of the element.

Node Output

MaxStringLength

An integer representing the maximum number of characters that can be entered.

Valid values: 0 = none allowed | -1 = unlimited

Ule:TextInput:SetMaxStringLength Node

Sets the maximum number of characters that can be entered.

Node Inputs

Activate

Updates the output.

ElementName

Name of the element.

MaxStringLength

An integer representing the maximum number of characters that can be entered.

Valid values: 0 = none allowed | -1 = unlimited

Node Output

Done

Sends a signal when the node's action is finished.

Ule:TextInput:GetPlaceholderTextEntity Node

Gets the placeholder text element.

Node Inputs

Activate

Updates the output.

ElementName

Name of the element.

Node Output

PlaceholderTextElement

The placeholder text element.

Ule:TextInput:SetPlaceholderTextEntity Node

Sets the placeholder text element.

Node Inputs

Activate

Updates the output.

ElementName

Name of the element.

PlaceholderTextElement

The placeholder text element.

Node Output

Done

Sends a signal when the node's action is finished.

Ule:TextInput:GetText Node

Gets the text string that the element is displaying or allowing to be edited.

Node Inputs

Activate

Updates the output.

ElementName

Name of the element.

Node Output

Value

The text string being displayed or edited by the element

Ule:TextInput:SetText Node

Sets the text string that the element is displaying or allowing to be edited.

Node Inputs

Activate

Updates the output.

ElementName

Name of the element.

Value

The text string being displayed or edited by the element

Node Output

Done

Sends a signal when the node's action is finished.

Ule:TextInput:GetTextCursorColor Node

Gets the color to be used for the text cursor.

Node Inputs

Activate

Updates the output.

ElementName

Name of the element.

Node Outputs

Color

The RGB value (0 – 255 each for R, G, and B) of the element identified by **ElementName**.

Alpha

The alpha value (0 – 255) of the element identified by **ElementName**.

Ule:TextInput:SetTextCursorColor Node

Sets the color to be used for the text cursor.

Node Inputs

Activate

Updates the output.

ElementName

Name of the element.

Color

The RGB value (0 – 255 each for R, G, and B) of the element identified by **ElementName**.

Alpha

The alpha value (0 – 255) of the element identified by **ElementName**.

Node Output

Done

Sends a signal when the node's action is finished.

Ule:TextInput:GetTextEntity Node

Gets the text element.

Node Inputs

Activate

Updates the output.

ElementName

Name of the element.

Node Output

TextElement

The text element.

Ule:TextInput:SetTextEntity Node

Gets the text element.

Node Inputs

Activate

Updates the output.

ElementName

Name of the element.

TextElement

The text element.

Node Output

Done

Sends a signal when the node's action is finished.

Ule:TextInput:GetTextSelectionColor Node

Gets the color to be used for the text background when it is selected.

Node Inputs

Activate

Updates the output.

ElementName

Name of the element.

Node Outputs

Color

The RGB value (0 – 255 each for R, G, and B) of the element identified by **ElementName**.

Alpha

The alpha value (0 – 255) of the element identified by **ElementName**.

Ule:TextInput:SetTextSelectionColor Node

Gets the color to be used for the text background when it is selected.

Node Inputs

Activate

Updates the output.

ElementName

Name of the element.

Color

The RGB value (0 – 255 for R, G, and B) of the element identified by **ElementName**.

Alpha

The alpha value (0 – 255) of the element identified by **ElementName**.

Node Output

Done

Sends a signal when the node's action is finished.

Ule Animation Node

The UI animation node consists of the following node inputs and outputs:

Ule:Sequence:Play Node

Controls playback of a UI animation sequence.

Node Inputs

Start

Starts playing the sequence from the beginning and triggers the **OnStarted** output.

Stop

Jumps the animation to the end and stops playing and triggers the **OnStopped** output.

Abort

Jumps the animation to the end and stops playing and triggers the **OnAborted** output.

Pause

Pauses the animation.

Resume

Continues playing a previously paused animation.

Reset

Resets the animation to the start. This applies all the key values for the first key frame of the animation.

SequenceName

The name of the sequence to play.

Node Outputs

OnStarted

Triggers when the sequence starts playing.

OnStopped

Triggers an output when the sequence stops playing, either because the end of the animation is reached or because the sequence is forced to stop (for example, by using the **Stop** node input).

OnAborted

Triggers an output when the sequence is aborted (for example, by using the **Abort** node input).

UI Flow Graph Nodes

The **UI** flow graph node set is the original, and now legacy, version of the **UI** flow graph nodes.

Use the **Ule** (p. 1185) flow graph node set for best results when creating new flow graph nodes for your user interface.

Topics

- [UI Canvas Nodes](#) (p. 1241)
- [UI Component Nodes](#) (p. 1246)
- [UI Animation Node](#) (p. 1303)

UI Canvas Nodes

The **UI Canvas** flow graph nodes have been superseded by the **Ule Canvas** (p. 1185) flow graph nodes. For best results, use the **Ule Canvas** (p. 1185) flow graph nodes.

You can use these flow graph nodes to perform actions on a UI canvas.

UI:Canvas:ActionListener Node

Listens for the specified action on a UI canvas.

Node Inputs

Activate

Initiates listening for the specified action.

CanvasID

Unique ID of the canvas to listen to.

ActionName

Name of the action to listen for.

Node Outputs

OnAction

Triggers when the canvas sends the action.

ElementID

ID of the UI element that triggered the action.

UI:Canvas:Load Node

Loads the specified UI canvas.

Node Inputs

Activate

Loads the canvas.

CanvasPathname

Path of the canvas to load.

Disabled

Sets whether canvas is disabled initially. If disabled, the canvas is not updated or rendered.

Node Outputs

OnLoad

Sends a signal when the canvas is loaded.

CanvasID

Outputs the unique canvas ID when the canvas is loaded.

UI:Canvas:Unload Node

Unloads the specified canvas.

Node Inputs

Activate

Unloads the canvas.

CanvasID

Unique ID of the canvas to unload.

Node Output

Done

Sends a signal when the node's action is finished.

UI:Canvas:FindLoaded Node

Finds the canvas ID for the UI canvas file path.

Node Inputs

Activate

Finds the canvas using the UI canvas file path.

CanvasPathname

Path of the canvas to find.

Node Outputs

CanvasID

The ID of the canvas that was found (if it was found).

Found

True if the canvas was found; otherwise, false.

UI:Canvas:GetKeepLoaded Node

Gets the Boolean value of whether the canvas stays loaded when a level is unloaded.

Node Inputs

Activate

Gets whether the canvas stays loaded when the level is unloaded.

CanvasID

Unique ID of the canvas to keep loaded.

Node Output

KeepLoaded

The Boolean value of whether the canvas stays loaded if the level is unloaded. True if the canvas should stay loaded during level unload; otherwise, false.

UI:Canvas:SetKeepLoaded Node

Determines whether the canvas stays loaded when a level is unloaded.

Node Inputs

Activate

Sets whether the canvas stays loaded when the level is unloaded.

CanvasID

Unique ID of the canvas to keep loaded.

KeepLoaded

If true, causes the canvas to stay loaded when the level is unloaded.

Node Output

Done

Sends a signal when the node's action is finished.

UI:Canvas:GetDrawOrder Node

Gets the integer draw order value for a UI canvas with respect to other UI canvases.

Node Inputs

Activate

Gets the draw order for the canvas.

CanvasID

Unique ID of the canvas to get the draw order from.

Node Output

DrawOrder

Order in which the canvas draws. Higher numbers appear before lower numbers.

UI:Canvas:SetDrawOrder Node

Sets the draw order for a UI canvas with respect to other UI canvases.

Node Inputs

Activate

Sets the draw order for the canvas.

CanvasID

Unique ID of the canvas whose draw order you are setting.

DrawOrder

Order in which to display the canvas. Higher numbers appear before lower numbers.

Node Output

Done

Sends a signal when the node's action is finished.

UI:Canvas:GetsPixelAligned Node

Gets the boolean value of whether the canvas is pixel-aligned.

Node Inputs

Activate

Gets whether visual element's vertices should snap to the nearest pixel.

CanvasID

Unique ID of the canvas.

Node Output

IsPixelAligned

Boolean value. True if the visual element's vertices should snap to the nearest pixel; otherwise, false.

UI:Canvas:SetIsPixelAligned Node

Sets whether visual element's vertices should snap to the nearest pixel.

Node Inputs

Activate

Sets the pixel-aligned property for the canvas ID.

CanvasID

Unique ID of the canvas to receive the pixel-aligned property value.

IsPixelAligned

Boolean value that represents whether a visual element's vertices should snap to the nearest pixel.

Node Output

Done

Sends a signal when the node's action is finished.

UI:Canvas:GetEnabled Node

Gets the boolean `enabled` flag of the canvas. Enabled canvases are updated and each frame rendered.

Node Inputs

Activate

Gets the enabled flag of the canvas.

CanvasID

Unique ID of the canvas to obtain the enabled flag from.

Node Output

Enabled

The enabled flag of the canvas. True if enabled; otherwise, false.

UI:Canvas:SetEnabled Node

Sets whether the canvas is enabled. Enabled canvases are updated and each frame rendered.

Node Inputs

Activate

Sets the enabled flag of the canvas.

CanvasID

Unique ID of the canvas to obtain the enabled flag from.

Enabled

True if the canvas should be enabled; otherwise, false.

Node Output

Done

Sends a signal when the node's action is finished.

UI Component Nodes

The **UI** component flow graph nodes have been superseded by the **UIe Component** (p. 1189) flow graph nodes. For best results, use the **UIe Component** (p. 1189) flow graph nodes.

These flow graph nodes perform actions on UI elements through their components.

Topics

- [UI Button Component Nodes](#) (p. 1246)
- [UI Checkbox Component Nodes](#) (p. 1247)
- [UI Element Node](#) (p. 1251)
- [UI Fader Component Nodes](#) (p. 1252)
- [UI Image Component Nodes](#) (p. 1253)
- [UI Interactable Component Nodes](#) (p. 1256)
- [UI Layout Column Component Nodes](#) (p. 1257)
- [UI Layout Grid Component Nodes](#) (p. 1259)
- [UI Layout Row Component Nodes](#) (p. 1264)
- [UI Mask Component Nodes](#) (p. 1267)
- [UI ScrollBox Component Nodes](#) (p. 1270)
- [UI ScrollBar Component Nodes](#) (p. 1280)
- [UI Slider Component Nodes](#) (p. 1284)
- [UI Text Component Nodes](#) (p. 1291)
- [UI Text Input Component Nodes](#) (p. 1295)

UI Button Component Nodes

The **UI Button** component flow graph nodes have been superseded by the **UIe Button** (p. 1189) flow graph nodes. For best results, use the **UIe Button** (p. 1189) component flow graph nodes.

Use the following flow graph nodes to perform actions on the button component.

[UI:Button:GetActionName](#) Node

Gets the action name string that is emitted when the button is released.

Node Inputs

Activate

Updates the output.

CanvasID

Unique identifier of the element's canvas.

ElementID

Unique identifier of the button element.

Node Output

Action

The action name associated with the button.

UI:Button:SetActionName Node

Sets the action name string that's emitted when the button is released.

Node Inputs

Activate

Assigns the action name.

CanvasID

Unique identifier of the element's canvas.

ElementID

Unique identifier of the button element.

Action

The action name string to assign to the button.

Node Output

Done

Sends a signal when the node's action is finished.

UI Checkbox Component Nodes

The **UI** component flow graph nodes have been superseded by the **UIe Checkbox (p. 1190)** component flow graph nodes. For best results, use the **UIe Checkbox (p. 1190)** component flow graph nodes.

Use the following flow graph nodes to perform actions on the check box component.

UI:Checkbox:GetState Node

Gets the Boolean state of the check box.

Node Inputs

Activate

Gets the state of the check box.

CanvasID

Unique identifier of the element's canvas.

ElementID

Unique identifier of the check box element.

Node Output

State

Outputs the current Boolean state of the check box.

UI:Checkbox:SetState Node

Sets the Boolean state of the check box.

Node Inputs

Activate

Sets the state of the check box.

CanvasID

Unique identifier of the element's canvas.

ElementID

Unique identifier of the check box element.

State

The Boolean state of the check box.

Node Output

Done

Sends a signal when the node's action is finished.

UI:Checkbox:GetChangedActionName Node

Gets the action triggered when the check box value changed.

Node Inputs

Activate

Gets the changed action name.

CanvasID

Unique identifier of the element's canvas.

ElementID

Unique identifier of the check box element.

Node Output

ChangedAction

The action name string value emitted when the check box value changes.

UI:Checkbox:SetChangedActionName Node

Sets the action triggered when the check box value changed.

Node Inputs

Activate

Gets the changed action name.

CanvasID

Unique identifier of the element's canvas.

ElementID

Unique identifier of the check box element.

ChangedAction

The action name string value emitted when the check box value changes.

UI:Checkbox:GetOptionalCheckedEntity Node

Gets the child element to show when the check box is in the `on` state.

Node Inputs

Activate

Updates the output.

CanvasID

Unique identifier of the element's canvas.

ElementID

Unique identifier of the check box element.

Node Output**CheckedElement**

The child element to show when the check box is selected (in the `on` state).

UI:Checkbox:SetOptionalCheckedEntity Node

Sets the child element to show when the check box is selected (in the `on` state).

Node Inputs**Activate**

Updates the output.

CanvasID

Unique identifier of the element's canvas.

ElementID

Unique identifier of the check box element.

CheckedElement

The child element to show when the checkbox is selected (in the `on` state).

Node Output**Done**

Sends a signal when the node's action is finished.

UI:Checkbox:GetOptionalUncheckedEntity Node

Gets the child element to show when the check box is deselected (in the `off` state).

Node Inputs**Activate**

Updates the output.

CanvasID

Unique identifier of the element's canvas.

ElementID

Unique identifier of the check box element.

Node Output**UncheckedElement**

The child element to show when the check box is deselected (off state).

UI:Checkbox:SetOptionalUncheckedEntity Node

Sets the child element to show when the check box is deselected (in the `off` state).

Node Inputs**Activate**

Updates the output.

CanvasID

Unique identifier of the element's canvas.

ElementID

Unique identifier of the checkbox element.

UncheckedElement

The child element to show when the check box is deselected (in the `off` state).

Node Output

Done

Sends a signal when the node's action is finished.

UI:Checkbox:GetTurnOnActionName Node

Gets the action triggered when the check box is selected.

Node Inputs

Activate

Updates the output.

CanvasID

Unique identifier of the element's canvas.

ElementID

Unique identifier of the check box element.

Node Output

TurnOnAction

The action name emitted when the check box is selected (turned on).

UI:Checkbox:SetTurnOnActionName Node

Sets the action triggered when the check box is selected (turned on).

Node Inputs

Activate

Assigns `TurnOnAction` as the action name that is emitted when the check box is selected.

CanvasID

Unique identifier of the element's canvas.

ElementID

Unique identifier of the check box element.

TurnOnAction

The action name emitted when the check box is selected.

Node Output

Done

Sends a signal when the node's action is finished.

UI:Checkbox:GetTurnOffActionName Node

Gets the action triggered when the check box is deselected (turned off).

Node Inputs

Activate

Update the output.

CanvasID

Unique identifier of the element's canvas.

ElementID

Unique identifier of the check box element.

Node Output

TurnOffAction

The action name emitted when the check box is deselected.

UI:Checkbox:SetTurnOffActionName Node

Sets the action triggered when the check box is deselected (turned off).

Node Inputs

Activate

Assigns `TurnOffAction` as the action name that is emitted when the check box is deselected.

CanvasID

Unique identifier of the element's canvas.

ElementID

Unique identifier of the check box element.

TurnOffAction

The action name emitted when the check box is deselected.

Node Output

Done

Sends a signal when the node's action is finished.

UI Element Node

The **UI Element** flow graph nodes have been superseded by the **UIe Element** (p. 1194) flow graph nodes. For best results, use the **UIe Element** (p. 1194) flow graph nodes.

Use the following flow graph node to enable or disable an element.

UI:Element:SetIsEnabled Node

Sets the Boolean enabled state of the element. If an element is not enabled, neither it nor any of its children are drawn or interactive.

Node Inputs

Activate

Sets the enabled state to the value of the `State` input.

CanvasID

Unique identifier of the element's canvas.

ElementID

Unique identifier of the element.

State

The Boolean enabled state of the element.

Node Output**Done**

Sends a signal when the node's action is finished.

UI Fader Component Nodes

The **UI Fader** component flow graph nodes have been superseded by the **UIe Fader** (p. 1194) component flow graph nodes. For best results, use the **UIe Fader** (p. 1194) component flow graph nodes.

Use the following flow graph nodes to perform actions on the fader component.

UI:Fader:Animation Node

Animates the fader component on the specified element.

Node Inputs**Activate**

Starts a fade animation.

CanvasID

Unique identifier of the fader element's canvas.

ElementID

Unique identifier of the fader element.

StartValue

Value at which the fade starts.

Valid values: 0 = Invisible | 1 = Opaque | -1 = Start from the current value

TargetValue

Value at which the fade ends.

Valid values: 0 = Invisible | 1 = Opaque

Speed

Rate at which the element fades.

Valid values: 0 = Instant fade | 0.5 = Slow fade | 1 = One second fade | 2 = Fade twice as fast

Node Outputs**OnComplete**

Sends a signal when the fade action is finished.

OnInterrupted

Sends a signal when the fade is interrupted by another fade starting.

UI:Fader:GetFadeValue Node

Gets the floating-point fade value of an element.

Node Inputs**Activate**

Updates the output.

CanvasID

Unique identifier of the element's canvas.

ElementID

Unique identifier of the element.

Node Output

Value

The floating-point fade value of the element (**ElementID**).

UI:Fader:SetFadeValue Node

Sets the fade value of an element.

Node Inputs

Activate

When triggered, assigns **Value** as the fade value of the fader component of the element.

CanvasID

Unique identifier of the element's canvas.

ElementID

Unique identifier of the element.

Value

The fade value to assign to the fader component for the element.

Node Output

Done

Sends a signal when the node's action is finished.

UI Image Component Nodes

The **UI Image** component flow graph nodes have been superseded by the [UI Image \(p. 1195\)](#) component flow graph nodes. For best results, use the [UI Image \(p. 1195\)](#) component flow graph nodes.

Use the following flow graph nodes to perform actions on the image component.

UI:Image:GetImageSource Node

Replaced by [UI:Image:GetSprite Node \(p. 1254\)](#).

Retrieves the texture file path currently used by the specified image element.

Node Inputs

Activate

Updates the output.

CanvasID

Unique identifier of the element's canvas.

ElementID

Unique identifier of the image element.

Node Outputs

Value

Outputs the file path of the image that is currently on the element.

UI:Image:SetImageSource Node

Replaced by [UI:Image:SetSprite Node](#) (p. 1254).

Changes the texture on the specified image element.

Node Inputs

Activate

Set the texture.

CanvasID

Unique identifier of the element's canvas.

ElementID

Unique identifier of the image element.

ImagePath

File path of the texture to display.

Node Output

Done

Sends a signal when the node's action is finished.

UI:Image:GetSprite Node

Gets the texture file path currently used by the specified image element.

Node Inputs

Activate

Updates the output.

CanvasID

Unique ID of the element's canvas.

ElementID

Unique ID of the image element.

Node Output

Value

Outputs the file path of the image that is currently on the element.

UI:Image:SetSprite Node

Sets the texture on the specified image element.

Node Inputs

Activate

Sets the texture.

CanvasID

Unique ID of the element's canvas.

ElementID

Unique ID of the image element.

ImagePath

File path of the texture to display.

Node Output**Done**

Sends a signal when the node's action is finished.

UI:Image:GetImageType Node

Gets the type of the image. Affects how the texture or sprite is mapped to the image rectangle.

Node Inputs**Activate**

Updates the output.

CanvasID

Unique identifier of the element's canvas.

ElementID

Unique identifier of the image element.

Node Output**ImageType**

An integer representing how the image is scaled and placed.

Valid values: 0 = Stretched | 1 = Sliced | 2 = Fixed | 3 = Tiled | 4 = Stretched to fit | 5 = Stretched to fill

UI:Image:SetImageType Node

Sets the type of the image. Affects how the texture or sprite is mapped to the image rectangle.

Node Inputs**Activate**

Updates the output.

CanvasID

Unique identifier of the element's canvas.

ElementID

Unique identifier of the image element.

ImageType

An integer representing how the image is scaled and placed.

Valid values: 0 = Stretched | 1 = Sliced | 2 = Fixed | 3 = Tiled | 4 = Stretched to fit | 5 = Stretched to fill

Node Output**Done**

Sends a signal when the node's action is finished.

UI:Image:GetColor Node

Gets the color tint for the image.

Node Inputs

Activate

Updates the output.

CanvasID

Unique identifier of the element's canvas.

ElementID

Unique identifier of the image element.

Node Outputs

Color

The RGB value (0 – 255 each for R, G, and B) of the element (**ElementID**).

Alpha

The alpha value (0 – 255) of the element (**ElementID**).

UI:Image:SetColor Node

Sets the color tint for the image.

Node Inputs

Activate

Updates the output.

CanvasID

Unique identifier of the element's canvas.

ElementID

Unique identifier of the image element.

Color

The RGB value (0 – 255 each for R, G, and B).

Alpha

A floating-point alpha value (0 – 255).

Node Output

Done

Sends a signal when the node's action is finished.

UI Interactable Component Nodes

The **UI** Interactable component flow graph nodes have been superseded by the **UIe Interactable** (p. 1198) component flow graph nodes. For best results, use the **UIe Interactable** (p. 1198) component flow graph nodes.

Use the following flow graph node for the **Interactable** component.

UI:Interactable:SetIsHandlingEvents Node

Sets the Boolean "is handling events" state of the element.

The **Interactable** flow graph nodes can be used to get or set values on any interactive UI element.

Interactive UI elements are elements that players can interacted with in game, such as button, text input, check box, slider, and so on. The **SetIsHandlingEvents** flow graph node sets whether an interactive UI element should handle input events. If set to false, then the UI element does not respond to input events, and its visual state is also changed to disabled.

Node Inputs

Activate

Sets the "is handling events" state.

CanvasID

Unique identifier of the element's canvas.

ElementID

Unique identifier of the element.

State

The Boolean "is handling events" state of the element.

Node Output

Done

Sends a signal when the node's action is finished.

UI Layout Column Component Nodes

The **UI** Layout column component flow graph nodes have been superseded by the **UIe** Layout column (p. 1199) component flow graph nodes. For best results, use the **UIe** Layout column (p. 1199) component flow graph nodes.

Use the following flow graph nodes to perform actions on the layout column component.

UI:LayoutColumn:GetOrder Node

Gets the vertical order of the **LayoutColumn** component for an element.

Node Inputs

Activate

Updates the output.

CanvasID

Unique identifier of the element's canvas.

ElementID

Unique identifier of the element.

Node Output

Order

An integer representing the vertical order.

Valid values: 0 = Top to bottom | 1 = Bottom to top

UI:LayoutColumn:SetOrder Node

Sets the vertical order of the LayoutColumn component for an element.

Node Inputs

Activate

Sets the vertical order for the element.

CanvasID

Unique identifier of the element's canvas.

ElementID

Unique identifier of the element.

Order

An integer representing the vertical order. 0 = Top to bottom | 1 = Bottom to top.

Node Output

Done

Sends a signal when the node's action is finished.

UI:LayoutColumn:GetPadding Node

Gets the padding (in pixels) inside the edges of the **LayoutColumn** component for an element.

Node Inputs

Activate

Updates the outputs.

CanvasID

Unique identifier of the element's canvas.

ElementID

Unique identifier of the element.

Node Outputs

Left

An integer representing the padding inside the left edge of the element.

Right

An integer representing the padding inside the right edge of the element.

Top

An integer representing the padding inside the top edge of the element.

Bottom

An integer representing the padding inside the bottom edge of the element.

UI:LayoutColumn:SetPadding Node

Sets the padding (in pixels) inside the edges of the **LayoutColumn** component for an element.

Node Inputs

Activate

Updates the outputs.

CanvasID

Unique identifier of the element's canvas.

ElementID

Unique identifier of the element.

Left

An integer representing the padding inside the left edge of the element.

Right

An integer representing the padding inside the right edge of the element.

Top

An integer representing the padding inside the top edge of the element.

Bottom

An integer representing the padding inside the bottom edge of the element.

Node Output

Done

Sends a signal when the node's action is finished.

UI:LayoutColumn:GetSpacing Node

Gets the spacing (in pixels) between child elements of the **LayoutColumn** component for an element.

Node Inputs

Activate

Updates the output.

CanvasID

Unique identifier of the element's canvas.

ElementID

Unique identifier of the element.

Node Output

Spacing

A float value of the spacing (in pixels) between child elements of the element (**ElementID**).

UI:LayoutColumn:SetSpacing Node

Sets the spacing (in pixels) between child elements of the **LayoutColumn** component for an element.

Node Inputs

Activate

Updates the output.

CanvasID

Unique identifier of the element's canvas.

ElementID

Unique identifier of the element.

Spacing

A float value of the spacing (in pixels) between child elements of the element (**ElementID**).

Node Output

Done

Sends a signal when the node's action is finished.

UI Layout Grid Component Nodes

The **UI** Layout grid component flow graph nodes have been superseded by the [UIe Layout grid \(p. 1201\)](#) component flow graph nodes. For best results, use the [UIe Layout grid \(p. 1201\)](#) component flow graph nodes.

Use the following flow graph nodes to perform actions on the layout grid component.

UI:LayoutGrid:GetCellSize Node

Gets the size (in pixels) of a child element in the layout.

Node Inputs

Activate

Updates the output.

CanvasID

Unique identifier of the element's canvas.

ElementID

Unique identifier of the element.

Node Outputs

CellWidth

The width (in pixels) of a child element of element (**ElementID**).

CellHeight

The height (in pixels) of a child element of element (**ElementID**).

UI:LayoutGrid:SetCellSize Node

Sets the size (in pixels) of a child element in the layout.

Node Inputs

Activate

Updates the output.

CanvasID

Unique identifier of the element's canvas.

ElementID

Unique identifier of the element.

CellWidth

The width (in pixels) of a child element of element (**ElementID**).

CellHeight

The height (in pixels) of a child element of element (**ElementID**).

Node Output

Done

Sends a signal when the node's action is finished.

UI:LayoutGrid:GetHorizontalOrder Node

Gets the horizontal order for the layout.

Node Inputs

Activate

Updates the outputs.

CanvasID

Unique identifier of the element's canvas.

ElementID

Unique identifier of the element.

Node Output

Order

An integer representing the horizontal order.

Valid values: 0 = Left to right | 1 = Right to left

[UI:LayoutGrid:SetHorizontalOrder Node](#)

Sets the horizontal order for the layout.

Node Inputs

Activate

Updates the outputs.

CanvasID

Unique identifier of the element's canvas.

ElementID

Unique identifier of the element.

Order

An integer representing the horizontal order.

Valid values: 0 = Left to right | 1 = Right to left

Node Output

Done

Sends a signal when the node's action is finished.

[UI:LayoutGrid:GetPadding Node](#)

Gets the padding (in pixels) inside the edges of the **LayoutGrid** component for an element.

Node Inputs

Activate

Updates the output.

CanvasID

Unique identifier of the element's canvas.

ElementID

Unique identifier of the element.

Node Output

Left

An integer representing the padding inside the left edge of the element.

Right

An integer representing the padding inside the right edge of the element.

Top

An integer representing the padding inside the top edge of the element.

Bottom

An integer representing the padding inside the bottom edge of the element.

[UI:LayoutGrid:SetPadding Node](#)

Sets the padding (in pixels) inside the edges of the **LayoutGrid** component for an element.

[Node Inputs](#)

Activate

Updates the output.

CanvasID

Unique identifier of the element's canvas.

ElementID

Unique identifier of the element.

Left

An integer representing the padding inside the left edge of the element.

Right

An integer representing the padding inside the right edge of the element.

Top

An integer representing the padding inside the top edge of the element.

Bottom

An integer representing the padding inside the bottom edge of the element.

[Node Output](#)

Done

Sends a signal when the node's action is finished.

[UI:LayoutGrid:GetSpacing Node](#)

Gets the spacing (in pixels) between child elements of the **LayoutGrid** component for an element.

[Node Inputs](#)

Activate

Updates the output.

CanvasID

Unique identifier of the element's canvas.

ElementID

Unique identifier of the element.

[Node Output](#)

Spacing

A float value of the spacing (in pixels) between child elements of the element (**ElementID**).

[UI:LayoutGrid:SetSpacing Node](#)

Sets the spacing (in pixels) between child elements of the **LayoutGrid** component for an element.

Node Inputs

Activate

Updates the output.

CanvasID

Unique identifier of the element's canvas.

ElementID

Unique identifier of the element.

Spacing

A float value of the spacing (in pixels) between child elements of the element (**ElementID**).

Node Output

Done

Sends a signal when the node's action is finished.

UI:LayoutGrid:GetStartingDirection Node

Gets the starting direction for the layout.

Node Inputs

Activate

Updates the output.

CanvasID

Unique identifier of the element's canvas.

ElementID

Unique identifier of the element.

Node Output

Direction

An integer representing the direction.

Valid values: 0 = Horizontal order | 1 = Vertical order

UI:LayoutGrid:SetStartingDirection Node

Sets the starting direction for the layout.

Node Inputs

Activate

Set the starting direction for the layout.

CanvasID

Unique identifier of the element's canvas.

ElementID

Unique identifier of the element.

Direction

An integer representing the horizontal order.

Valid values: 0 = Horizontal order | 1 = Vertical order.

Node Output

Done

Sends a signal when the node's action is finished.

UI:LayoutGrid:GetVerticalOrder Node

Gets the vertical order for the layout.

Node Inputs

Activate

Updates the outputs.

CanvasID

Unique identifier of the element's canvas.

ElementID

Unique identifier of the element.

Node Output

Action

An integer representing the vertical order.

Valid values: 0 = Top to bottom | 1 = Bottom to top

UI:LayoutGrid:SetVerticalOrder Node

Sets the vertical order for the layout.

Node Inputs

Activate

Sets the vertical order for the layout.

CanvasID

Unique identifier of the element's canvas.

ElementID

Unique identifier of the element.

Action

An integer representing the vertical order.

Valid values: 0 = Top to bottom | 1 = Bottom to top

Node Output

Done

Sends a signal when the node's action is finished.

UI Layout Row Component Nodes

The **UI** Layout row component flow graph nodes have been superseded by the **UIe** Layout row (p. 1205) component flow graph nodes. For best results, use the **UIe** Layout row (p. 1205) component flow graph nodes.

Use the following flow graph nodes to perform actions on the layout row component.

UI:LayoutRow:GetOrder Node

Gets the horizontal order of the **LayoutRow** component for an element.

Node Inputs

Activate

Updates the output.

CanvasID

Unique identifier of the element's canvas.

ElementID

Unique identifier of the element.

Node Output

Order

An integer representing the horizontal order.

Valid values: 0 = Left to right | 1 = Right to left

UI:LayoutRow:SetOrder Node

Sets the horizontal order of the **LayoutRow** component for an element.

Node Inputs

Activate

Updates the output.

CanvasID

Unique identifier of the element's canvas.

ElementID

Unique identifier of the element.

Order

An integer representing the horizontal order.

Valid values: 0 = Left to right | 1 = Right to left

Node Output

Done

Sends a signal when the node's action is finished.

UI:LayoutRow:GetPadding Node

Gets the padding (in pixels) inside the edges of the **LayoutRow** component for an element.

Node Inputs

Activate

Updates the output.

CanvasID

Unique identifier of the element's canvas.

ElementID

Unique identifier of the element.

Node Outputs

Left

An integer representing the padding inside the left edge of the element.

Right

An integer representing the padding inside the right edge of the element.

Top

An integer representing the padding inside the top edge of the element.

Bottom

An integer representing the padding inside the bottom edge of the element.

UI:LayoutRow:SetPadding Node

Sets the padding (in pixels) inside the edges of the LayoutRow component for an element.

Node Inputs

Activate

Sets the padding (in pixels) inside the edges of the LayoutRow.

CanvasID

Unique identifier of the element's canvas.

ElementID

Unique identifier of the element.

Left

An integer representing the padding inside the left edge of the element.

Right

An integer representing the padding inside the right edge of the element.

Top

An integer representing the padding inside the top edge of the element.

Bottom

An integer representing the padding inside the bottom edge of the element.

Node Output

Done

Sends a signal when the node's action is finished.

UI:LayoutRow:GetSpacing Node

Gets the spacing (in pixels) between child elements of the LayoutRow component for an element.

Node Inputs

Activate

Updates the outputs.

CanvasID

Unique identifier of the element's canvas.

ElementID

Unique identifier of the element.

Node Output

Spacing

A float value of the spacing (in pixels) between child elements of the element (**ElementID**).

UI:LayoutRow:SetSpacing Node

Sets the spacing (in pixels) between child elements of the LayoutRow component for an element.

Node Inputs

Activate

Sets the spacing (in pixels) between child elements.

CanvasID

Unique identifier of the element's canvas.

ElementID

Unique identifier of the element.

Spacing

A float value of the spacing (in pixels) between child elements of the element (**ElementID**).

Node Output

Done

Sends a signal when the node's action is finished.

UI Mask Component Nodes

The **UI Mask** component flow graph nodes have been superseded by the [UIe Mask \(p. 1208\)](#) component flow graph nodes. For best results, use the [UIe Mask \(p. 1208\)](#) component flow graph nodes.

Use the following flow graph nodes to perform actions on the mask component.

UI:Mask:GetDrawBehind Node

Gets whether mask is drawn behind the child elements.

Node Inputs

Activate

Updates the outputs.

CanvasID

Unique identifier of the element's canvas.

ElementID

Unique identifier of the element.

Node Output

DrawBehind

Indicates whether mask is drawn behind the child elements.

UI:Mask:SetDrawBehind Node

Sets whether mask is drawn behind the child elements.

Node Inputs

Activate

Sets whether mask is drawn behind the child elements.

CanvasID

Unique identifier of the element's canvas.

ElementID

Unique identifier of the element.

DrawBehind

Sets whether mask is drawn behind the child elements.

Node Output

Done

Sends a signal when the node's action is finished.

UI:Mask:GetDrawInFront Node

Gets whether mask is drawn in front of child elements.

Node Inputs

Activate

Updates the outputs.

CanvasID

Unique identifier of the element's canvas.

ElementID

Unique identifier of the element.

Node Output

DrawInFront

Indicates whether mask is drawn in front of child elements.

UI:Mask:SetDrawInFront Node

Sets whether mask is drawn in front of child elements.

Node Inputs

Activate

Sets whether mask is drawn in front of child elements.

CanvasID

Unique identifier of the element's canvas.

ElementID

Unique identifier of the element.

DrawInFront

Sets whether mask is drawn in front of child elements.

Node Output

Done

Sends a signal when the node's action is finished.

UI:Mask:GetsMaskingEnabled Node

Gets whether masking is enabled.

Node Inputs

Activate

Updates the outputs.

CanvasID

Unique identifier of the element's canvas.

ElementID

Unique identifier of the element.

Node Output

IsMaskingEnabled

Indicates whether masking is enabled.

UI:Mask:SetIsMaskingEnabled Node

Sets whether masking is enabled.

Node Inputs

Activate

Sets whether masking is enabled.

CanvasID

Unique identifier of the element's canvas.

ElementID

Unique identifier of the element.

IsMaskingEnabled

Sets whether masking is enabled.

Node Output

Done

Sends a signal when the node's action is finished.

UI:Mask:GetUseAlphaTest Node

Gets whether to use the alpha channel in the mask visual's texture to define the mask.

Node Inputs

Activate

Updates the outputs.

CanvasID

Unique identifier of the element's canvas.

ElementID

Unique identifier of the element.

Node Output

UseAlphaTest

Indicates whether to use the alpha channel in the mask visual's texture to define the mask.

UI:Mask:SetUseAlphaTest Node

Sets whether to use the alpha channel in the mask visual's texture to define the mask.

Node Inputs

Activate

Sets whether to use the alpha channel in the mask visual's texture to define the mask.

CanvasID

Unique identifier of the element's canvas.

ElementID

Unique identifier of the element.

UseAlphaTest

Sets whether to use the alpha channel in the mask visual's texture to define the mask.

Node Output

Done

Sends a signal when the node's action is finished.

UI ScrollBox Component Nodes

The **UI Scrollbox** component flow graph nodes have been superseded by the [UIe Scrollbox \(p. 1210\)](#) component flow graph nodes. For best results, use the [UIe Scrollbox \(p. 1210\)](#) component flow graph nodes.

Use the following flow graph nodes to perform actions on the **ScrollBox** component.

UI:ScrollBox:FindClosestContentChildElement Node

Finds the child of the content element that is closest to the content anchors.

Node Inputs

Activate

Updates the outputs.

CanvasID

Unique identifier of the element's canvas.

ElementID

Unique identifier of the element.

Node Output

ClosestElement

The element currently closest to the focused element.

UI:ScrollBox:GetContentEntity Node

Gets the content element for the **ScrollBox**.

Node Inputs

Activate

Updates the outputs.

CanvasID

Unique identifier of the element's canvas.

ElementID

Unique identifier of the element.

Node Output

Content

The element that the **ScrollBox** scrolls.

[UI:ScrollBox:SetContentEntity Node](#)

Sets the content element for the **ScrollBox**.

Node Inputs

Activate

Sets the content element for the **ScrollBox**.

CanvasID

Unique identifier of the element's canvas.

ElementID

Unique identifier of the element.

Content

The element that the **ScrollBox** scrolls.

Node Output

Done

Sends a signal when the node's action is finished.

[UI:ScrollBox:GetsHorizontalScrollingEnabled Node](#)

Gets whether the **ScrollBox** allows horizontal scrolling.

Node Inputs

Activate

Updates the outputs.

CanvasID

Unique identifier of the element's canvas.

ElementID

Unique identifier of the element.

Node Output

Enabled

Indicates whether horizontal scrolling is enabled.

[UI:ScrollBox:SetsHorizontalScrollingEnabled Node](#)

Sets whether the **ScrollBox** allows horizontal scrolling.

Node Inputs

Activate

Sets whether the **ScrollBox** allows horizontal scrolling.

CanvasID

Unique identifier of the element's canvas.

ElementID

Unique identifier of the element.

Enabled

Sets whether horizontal scrolling is enabled.

Node Output

Done

Sends a signal when the node's action is finished.

[UI:ScrollBox:GetsScrollingConstrained Node](#)

Gets whether the **ScrollBox** restricts scrolling to the content area.

Node Inputs

Activate

Updates the outputs.

CanvasID

Unique identifier of the element's canvas.

ElementID

Unique identifier of the element.

Node Output

IsConstrained

Indicates whether scrolling is constrained.

[UI:ScrollBox:SetIsScrollingConstrained Node](#)

Sets whether the **ScrollBox** restricts scrolling to the content area.

Node Inputs

Activate

Sets whether the **ScrollBox** restricts scrolling to the content area.

CanvasID

Unique identifier of the element's canvas.

ElementID

Unique identifier of the element.

IsConstrained

Sets whether scrolling is constrained.

Node Output

Done

Sends a signal when the node's action is finished.

[UI:ScrollBox:GetsVerticalScrollingEnabled Node](#)

Gets whether the **ScrollBox** allows vertical scrolling.

Node Inputs

Activate

Updates the output.

CanvasID

Unique identifier of the element's canvas.

ElementID

Unique identifier of the element.

Node Output

Enabled

Indicates whether vertical scrolling is enabled.

UI:ScrollBox:SetIsVerticalScrollingEnabled Node

Sets whether the **ScrollBox** allows vertical scrolling.

Node Inputs

Activate

Sets whether the **ScrollBox** allows vertical scrolling.

CanvasID

Unique identifier of the element's canvas.

ElementID

Unique identifier of the element.

Enabled

Sets whether vertical scrolling is enabled.

Node Output

Done

Sends a signal when the node's action is finished.

UI:ScrollBox:GetScrollOffset Node

Gets the scroll offset of the **ScrollBox**.

Node Inputs

Activate

Updates the outputs.

CanvasID

Unique identifier of the element's canvas.

ElementID

Unique identifier of the element.

Node Output

HorizOffset

The horizontal scroll offset of the element identified by **ElementID**.

VertOffset

The vertical scroll offset of the element identified by **ElementID**.

UI:ScrollBox:SetScrollOffset Node

Sets the scroll offset of the **ScrollBox**.

Node Inputs

Activate

Sets the scroll offset of the **ScrollBox**.

CanvasID

Unique identifier of the element's canvas.

ElementID

Unique identifier of the element.

HorizOffset

The horizontal scroll offset of **ElementID**.

VertOffset

The vertical scroll offset of **ElementID**.

Node Output

Done

Sends a signal when the node's action is finished.

UI:ScrollBox:GetScrollOffsetChangedActionName Node

Gets the action triggered when the **ScrollBox** drag is completed.

Node Inputs

Activate

Updates the output.

CanvasID

Unique identifier of the element's canvas.

ElementID

Unique identifier of the element.

Node Output

ChangedAction

The action name.

UI:ScrollBox:SetScrollOffsetChangedActionName Node

Sets the action triggered when the **ScrollBox** drag is completed.

Node Inputs

Activate

Sets the action triggered when the **ScrollBox** drag is completed.

CanvasID

Unique identifier of the element's canvas.

ElementID

Unique identifier of the element.

ChangedAction

The action name.

Node Output

Done

Sends a signal when the node's action is finished.

UI:ScrollBox:GetScrollOffsetChangingActionName Node

Gets the action triggered while the **ScrollBox** is being dragged.

Node Inputs

Activate

Updates the output.

CanvasID

Unique identifier of the element's canvas.

ElementID

Unique identifier of the element.

Node Output

ChangingAction

The action name.

UI:ScrollBox:SetScrollOffsetChangingActionName Node

Sets the action triggered while the **ScrollBox** is being dragged.

Node Inputs

Activate

Sets the action triggered while the **ScrollBox** is being dragged.

CanvasID

Unique identifier of the element's canvas.

ElementID

Unique identifier of the element.

ChangingAction

The action name.

Node Output

Done

Sends a signal when the node's action is finished.

UI:ScrollBox:GetSnapGrid Node

Gets the snapping grid of the **ScrollBox**.

Node Inputs

Activate

Updates the outputs.

CanvasID

Unique identifier of the element's canvas.

ElementID

Unique identifier of the element.

Node Outputs

HorizSpacing

The horizontal grid spacing of the element identified by **ElementID**.

VertSpacing

The vertical grid spacing of the element identified by **ElementID**.

UI:ScrollBox:SetSnapGrid Node

Sets the snapping grid of the **ScrollBox**.

Node Inputs

Activate

Sets the snapping grid of the **ScrollBox**.

CanvasID

Unique identifier of the element's canvas.

ElementID

Unique identifier of the element.

HorizSpacing

The horizontal grid spacing of the element identified by **ElementID**.

VertSpacing

The vertical grid spacing of the element identified by **ElementID**.

Node Output

Done

Sends a signal when the node's action is finished.

UI:ScrollBox:GetSnapMode Node

Gets the snap mode for the **ScrollBox**.

Node Inputs

Activate

Updates the output.

CanvasID

Unique identifier of the element's canvas.

ElementID

Unique identifier of the element.

Node Output

SnapMode

An integer representing the snap mode state.

Valid values: 0 = None | 1 = Children | 2 = Grid

UI:ScrollBox:SetSnapMode Node

Sets the snap mode for the **ScrollBox**.

Node Inputs

Activate

Sets the snap mode for the **ScrollBox**.

CanvasID

Unique identifier of the element's canvas.

ElementID

Unique identifier of the element.

SnapMode

An integer representing the snap mode state.

Valid values: 0 = None | 1 = Children | 2 = Grid

Node Output

Done

Sends a signal when the node's action is finished.

UI:ScrollBox:GetHorizontalScrollBarVisibility Node

Gets horizontal scrollbar visibility behavior.

Node Inputs

Activate

Updates the outputs.

CanvasID

Unique identifier of the element's canvas.

ElementID

Unique identifier of the element.

Node Output

ScrollBarVisibility

An integer that represents the scrollbar visibility behavior.

Valid values: 0 = AlwaysVisible | 1 = AutoHide | 2 = AutoHideAndResizeViewArea

UI:ScrollBox:SetHorizontalScrollBarVisibility Node

Sets horizontal scrollbar visibility behavior.

Node Inputs

Activate

Sets horizontal scroll bar visibility behavior.

CanvasID

Unique identifier of the element's canvas.

ElementID

Unique identifier of the element.

ScrollBarVisibility

An integer representing the scrollbar visibility behavior.

0 = AlwaysVisible | 1 = AutoHide | 2 = AutoHideAndResizeViewArea.

Node Output

Done

Sends a signal when the node's action is finished.

UI:ScrollBox:GetVerticalScrollBarVisibility Node

Gets vertical scrollbar visibility behavior.

Node Inputs

Activate

Updates the outputs.

CanvasID

Unique identifier of the element's canvas.

ElementID

Unique identifier of the element.

Node Output

ScrollBarVisibility

An integer that represents the scrollbar visibility behavior.

Valid values: 0 = AlwaysVisible | 1 = AutoHide | 2 = AutoHideAndResizeViewArea

UI:ScrollBox:SetVerticalScrollBarVisibility Node

Sets vertical scrollbar visibility behavior.

Node Inputs

Activate

Sets vertical scroll bar visibility behavior.

CanvasID

Unique identifier of the element's canvas.

ElementID

Unique identifier of the element.

ScrollBarVisibility

An integer representing the scrollbar visibility behavior.

0 = AlwaysVisible | 1 = AutoHide | 2 = AutoHideAndResizeViewArea.

Node Output

Done

Sends a signal when the node's action is finished.

UI:ScrollBox:GetHorizontalScrollBarEntity Node

Gets the horizontal scrollbar element for the **ScrollBox**.

Node Inputs

Activate

Updates the outputs.

CanvasID

Unique identifier of the element's canvas.

ElementID

Unique identifier of the element.

Node Output

HorizontalScrollBar

The element that scrolls the **ScrollBox** horizontally.

[UI:ScrollBox:SetHorizontalScrollBarEntity Node](#)

Sets the horizontal scrollbar element for the **ScrollBox**.

Node Inputs

Activate

Sets the horizontal scrollbar element for the **ScrollBox**.

CanvasID

Unique identifier of the element's canvas.

ElementID

Unique identifier of the element.

HorizontalScrollBar

The element that scrolls the **ScrollBox** horizontally.

Node Output

Done

Sends a signal when the node's action is finished.

[UI:ScrollBox:GetVerticalScrollBarEntity Node](#)

Gets the vertical scrollbar element for the **ScrollBox**.

Node Inputs

Activate

Updates the outputs.

CanvasID

Unique identifier of the element's canvas.

ElementID

Unique identifier of the element.

Node Output

VerticalScrollBar

The element that scrolls the **ScrollBox** vertically.

[UI:ScrollBox:SetVerticalScrollBarEntity Node](#)

Sets the vertical scrollbar element for the **ScrollBox**.

Node Inputs

Activate

Sets the vertical scrollbar element for the **ScrollBox**.

CanvasID

Unique identifier of the element's canvas.

ElementID

Unique identifier of the element.

VerticalScrollBar

The element that scrolls the **ScrollBox** vertically.

Node Output

Done

Sends a signal when the node's action is finished.

UI ScrollBar Component Nodes

The **UI Scrollbar** component flow graph nodes have been superseded by the [UIe Scrollbar \(p. 1219\)](#) component flow graph nodes. For best results, use the [UIe Scrollbar \(p. 1219\)](#) component flow graph nodes.

Use the following flow graph nodes to perform actions on the **Scrollbar** component.

[UI:Scrollbar:GetHandleEntity Node](#)

Gets the handle element of the scroll bar.

Node Inputs

Activate

Updates the output.

CanvasID

Unique identifier of the element's canvas.

ElementID

Unique identifier of the element.

Node Output

Handle

The handle element.

[UI:Scrollbar:SetHandleEntity Node](#)

Sets the handle element of the scroll bar.

Node Inputs

Activate

Sets the handle element.

CanvasID

Unique identifier of the element's canvas.

ElementID

Unique identifier of the element.

Handle

The handle element.

Node Output**Done**

Sends a signal when the node's action is finished.

UI:Scrollbar:GetValue Node

Gets the value of the scrollbar.

Node Inputs**Activate**

Updates the output.

CanvasID

Unique identifier of the element's canvas.

ElementID

Unique identifier of the element.

Node Output**Value**

The scrollbar value of the element identified by `ElementID`.

UI:Scrollbar:SetValue Node

Sets the value of the scroll bar.

Node Inputs**Activate**

Sets the value of the scrollbar.

CanvasID

Unique identifier of the element's canvas.

ElementID

Unique identifier of the element.

Value

The scrollbar value of the element identified by `ElementID`.

Node Output**Done**

Sends a signal when the node's action is finished.

UI:Scrollbar:GetHandleSize Node

Gets the size of the handle.

Node Inputs**Activate**

Updates the output.

CanvasID

Unique identifier of the element's canvas.

ElementID

Unique identifier of the element.

Node Output

HandleSize

The size of the handle of the element identified by `ElementID`.

[UI:Scrollbar:SetHandleSize Node](#)

Sets the size of the handle.

Node Inputs

Activate

Sets the size of the handle.

CanvasID

Unique identifier of the element's canvas.

ElementID

Unique identifier of the element.

HandleSize

The size of the handle of the element identified by `ElementID`.

Node Output

Done

Sends a signal when the node's action is finished.

[UI:Scrollbar:GetMinHandlePixelSize Node](#)

Gets the minimum size in pixels of the handle.

Node Inputs

Activate

Updates the output.

CanvasID

Unique identifier of the element's canvas.

ElementID

Unique identifier of the element.

Node Output

MinHandleSize

The minimum size in pixels of the handle of the element identified by `ElementID`.

[UI:Scrollbar:SetMinHandlePixelSize Node](#)

Sets the minimum size in pixels of the handle.

Node Inputs

Activate

Sets the minimum size in pixels of the handle.

CanvasID

Unique identifier of the element's canvas.

ElementID

Unique identifier of the element.

MinHandleSize

The minimum size in pixels of the handle of the element identified by `ElementID`.

Node Output

Done

Sends a signal when the node's action is finished.

UI:Scrollbar:GetValueChangedActionName Node

Gets the action triggered when the value is done changing.

Node Inputs

Activate

Updates the output.

CanvasID

Unique identifier of the element's canvas.

ElementID

Unique identifier of the element.

Node Output

ValueChangedAction

The action name.

UI:Scrollbar:SetValueChangedActionName Node

Sets the action triggered when the value is done changing.

Node Inputs

Activate

Updates the output.

CanvasID

Unique identifier of the element's canvas.

ElementID

Unique identifier of the element.

ValueChangedAction

The action name.

Node Output

Done

Sends a signal when the node's action is finished.

UI:Scrollbar:GetValueChangingActionName Node

Gets the action triggered while the value is changing.

Node Inputs

Activate

Updates the output.

CanvasID

Unique identifier of the element's canvas.

ElementID

Unique identifier of the element.

Node Output

ValueChangingAction

The action name.

UI:Scrollbar:SetValueChangingActionName Node

Sets the action triggered while the value is changing.

Node Inputs

Activate

Updates the output.

CanvasID

Unique identifier of the element's canvas.

ElementID

Unique identifier of the element.

ValueChangingAction

The action name.

Node Output

Done

Sends a signal when the node's action is finished.

UI Slider Component Nodes

The **UI Slider** component flow graph nodes have been superseded by the **UIe Slider** (p. 1224) component flow graph nodes. For best results, use the **UIe Slider** (p. 1224) component flow graph nodes.

Use the following flow graph nodes to perform actions on the slider component.

UI:Slider:GetFillEntity Node

Gets the fill element.

Node Inputs

Activate

Updates the output.

CanvasID

Unique identifier of the element's canvas.

ElementID

Unique identifier of the element.

Node Output

FillElement

The fill element.

UI:Slider:SetFillEntity Node

Sets the fill element.

Node Inputs

Activate

Sets the fill element.

CanvasID

Unique identifier of the element's canvas.

ElementID

Unique identifier of the element.

FillElement

The fill element.

Node Output

Done

Sends a signal when the node's action is finished.

UI:Slider:GetManipulatorEntity Node

Gets the manipulator element.

Node Inputs

Activate

Updates the output.

CanvasID

Unique identifier of the element's canvas.

ElementID

Unique identifier of the element.

Node Output

ManipulatorElement

The manipulator element.

UI:Slider:SetManipulatorEntity Node

Sets the manipulator element.

Node Inputs

Activate

Sets the manipulator element.

CanvasID

Unique identifier of the element's canvas.

ElementID

Unique identifier of the element.

ManipulatorElement

The manipulator element.

Node Output

Done

Sends a signal when the node's action is finished.

[UI:Slider:GetMaxValue Node](#)

Gets the maximum value of the slider.

Node Inputs

Activate

Updates the output.

CanvasID

Unique identifier of the element's canvas.

ElementID

Unique identifier of the element.

Node Output

MaxValue

The slider maximum value of the element identified by **ElementID**.

[UI:Slider:SetMaxValue Node](#)

Sets the maximum value of the slider.

Node Inputs

Activate

Sets the maximum value of the slider.

CanvasID

Unique identifier of the element's canvas.

ElementID

Unique identifier of the element.

MaxValue

The slider maximum value of the element identified by **ElementID**.

Node Output

Done

Sends a signal when the node's action is finished.

[UI:Slider:GetMinValue Node](#)

Gets the minimum value of the slider.

Node Inputs

Activate

Updates the output.

CanvasID

Unique identifier of the element's canvas.

ElementID

Unique identifier of the element.

Node Output

MinValue

The slider minimum value of the element identified by **ElementID**.

UI:Slider:SetMinValue Node

Sets the minimum value of the slider.

Node Inputs

Activate

Sets the minimum value of the slider.

CanvasID

Unique identifier of the element's canvas.

ElementID

Unique identifier of the element.

MinValue

The slider minimum value of the element identified by **ElementID**.

Node Output

Done

Sends a signal when the node's action is finished.

UI:Slider:GetStepValue Node

Gets the smallest increment allowed between values. Zero means no restriction.

Node Inputs

Activate

Updates the output.

CanvasID

Unique identifier of the element's canvas.

ElementID

Unique identifier of the element.

Node Output

StepValue

The smallest increment allowed between values of the element identified by **ElementID**. Zero means no restriction.

UI:Slider:SetStepValue Node

Sets the smallest increment allowed between values. Zero means no restriction.

Node Inputs

Activate

Sets the smallest increment allowed between values. Zero means no restriction.

CanvasID

Unique identifier of the element's canvas.

ElementID

Unique identifier of the element.

StepValue

The smallest increment allowed between values of the element identified by **ElementID**. Zero means no restriction.

Node Output

Done

Sends a signal when the node's action is finished.

UI:Slider:GetTrackEntity Node

Gets the track element.

Node Inputs

Activate

Updates the output.

CanvasID

Unique identifier of the element's canvas.

ElementID

Unique identifier of the element.

Node Output

Track

The track element.

UI:Slider:SetTrackEntity Node

Sets the track element.

Node Inputs

Activate

Sets the track element.

CanvasID

Unique identifier of the element's canvas.

ElementID

Unique identifier of the element.

Track

The track element.

Node Output

Done

Sends a signal when the node's action is finished.

UI:Slider:GetValue Node

Gets the value of slider.

Node Inputs

Activate

Updates the output.

CanvasID

Unique identifier of the element's canvas.

ElementID

Unique identifier of the element.

Node Output

Value

The slider value of the element identified by **ElementID**.

UI:Slider:SetValue Node

Sets the value of the slider.

Node Inputs

Activate

Sets the value of the slider.

CanvasID

Unique identifier of the element's canvas.

ElementID

Unique identifier of the element.

Value

The slider value of the element identified by **ElementID**.

Node Output

Done

Sends a signal when the node's action is finished.

UI:Slider:GetValueChangedActionName Node

Gets the action triggered when the value is done changing.

Node Inputs

Activate

Updates the output.

CanvasID

Unique identifier of the element's canvas.

ElementID

Unique identifier of the element.

Node Output

ValueChangedAction

The action name.

UI:Slider:SetValueChangedActionName Node

Sets the action triggered when the value is done changing.

Node Inputs

Activate

Updates the output.

CanvasID

Unique identifier of the element's canvas.

ElementID

Unique identifier of the element.

ValueChangedAction

The action name.

Node Output

Done

Sends a signal when the node's action is finished.

UI:Slider:GetValueChangingActionName Node

Gets the action triggered while the value is changing.

Node Inputs

Activate

Updates the output.

CanvasID

Unique identifier of the element's canvas.

ElementID

Unique identifier of the element.

Node Output

ValueChangingAction

The action name.

UI:Slider:SetValueChangingActionName Node

Sets the action triggered while the value is changing.

Node Inputs

Activate

Updates the output.

CanvasID

Unique identifier of the element's canvas.

ElementID

Unique identifier of the element.

ValueChangingAction

The action name.

Node Output

Done

Sends a signal when the node's action is finished.

UI Text Component Nodes

The **UI Text** component flow graph nodes have been superseded by the **UIe Text** (p. 1229) component flow graph nodes. For best results, use the **UIe Text** (p. 1229) component flow graph nodes.

Use the following flow graph nodes to perform actions on the text component.

UI:Text:GetColor Node

Gets the color to draw the text string.

Node Inputs

Activate

Updates the outputs.

CanvasID

Unique identifier of the element's canvas.

ElementID

Unique identifier of the element.

Node Outputs

Color

The RGB value (0 – 255 each for R, G, and B) of the element identified by **ElementID**.

Alpha

The alpha value (0 – 255) of the element identified by **ElementID**.

UI:Text:SetColor Node

Sets the color to draw the text string.

Node Inputs

Activate

Updates the output.

CanvasID

Unique identifier of the element's canvas.

ElementID

Unique identifier of the element.

Color

The RGB value (0 – 255 each for R, G, and B) of the element identified by **ElementID**.

Alpha

The alpha value (0 – 255) of the element identified by **ElementID**.

Node Output

Done

Sends a signal when the node's action is finished.

UI:Text:GetFont Node

Gets the path to the font.

Node Inputs

Activate

Updates the output.

CanvasID

Unique identifier of the element's canvas.

ElementID

Unique identifier of the element.

Node Output

Font

The path to the font used by the element.

UI:Text:SetFontNode

Sets the path to the font.

Node Inputs

Activate

Updates the output.

CanvasID

Unique identifier of the element's canvas.

ElementID

Unique identifier of the element.

Font

The path to the font used by the element identified by **ElementID**.

Node Output

Done

Sends a signal when the node's action is finished.

UI:Text:GetFontSize Node

Gets the font size in points.

Node Inputs

Activate

Updates the output.

CanvasID

Unique identifier of the element's canvas.

ElementID

Unique identifier of the element.

Node Output

FontSize

The font size of the element identified by **ElementID**.

UI:Text:SetFontSize Node

Sets the font size in points.

Node Inputs

Activate

Updates the output.

CanvasID

Unique identifier of the element's canvas.

ElementID

Unique identifier of the element.

FontSize

The font size of the element identified by **ElementID**.

Node Output

Done

Sends a signal when the node's action is finished.

UI:Text:GetOverflowMode Node

Gets the overflow behavior of the text.

Node Inputs

Activate

Updates the output.

CanvasID

Unique identifier of the element's canvas.

ElementID

Unique identifier of the element.

Node Output

OverflowMode

An integer representing how overflow text is handled.

Valid values: 0 = Overflow text | 1 = Clip text

UI:Text:SetOverflowModeNode

Sets the overflow behavior of the text.

Node Inputs

Activate

Updates the output.

CanvasID

Unique identifier of the element's canvas.

ElementID

Unique identifier of the element.

OverflowMode

An integer representing how overflow text is handled.

Valid values: 0 = Overflow text | 1 = Clip text

Node Output**Done**

Sends a signal when the node's action is finished.

UI:Text:GetText Node

Gets the text string that the element displays.

Node Inputs**Activate**

Updates the output.

CanvasID

Unique identifier of the element's canvas.

ElementID

Unique identifier of the element.

Node Output**Value**

The text string being displayed by the element identified by **ElementID**.

UI:Text:SetText Node

Sets the text string being displayed by the element.

Node Inputs**Activate**

Updates the output.

CanvasID

Unique identifier of the element's canvas.

ElementID

Unique identifier of the element.

Value

The text string being displayed by the element identified by **ElementID**.

Node Output**Done**

Sends a signal when the node's action is finished.

UI:Text:GetWrapText Node

Gets whether text is wrapped.

Node Inputs

Activate

Updates the output.

CanvasID

Unique identifier of the element's canvas.

ElementID

Unique identifier of the element.

Node Output

WrapTextSetting

An integer representing how long text lines are handled.

Valid values: 0 = No wrap | 1 = Wrap

UI:Text:SetWrapText Node

Gets whether text is wrapped.

Node Inputs

Activate

Updates the outputs.

CanvasID

Unique identifier of the element's canvas.

ElementID

Unique identifier of the element.

WrapTextSetting

An integer representing how long text lines are handled.

Valid values: 0 = No wrap | 1 = Wrap

Node Output

Done

Sends a signal when the node's action is finished.

UI Text Input Component Nodes

The **UI** Text Input component flow graph nodes have been superseded by the **UIe** Text Input (p. 1233) component flow graph nodes. For best results, use the **UIe** Text Input (p. 1233) component flow graph nodes.

Use the following flow graph nodes to perform actions on the text input component.

UI:TextInput:GetChangeAction Node

Gets the action triggered when the text is changed.

Node Inputs

Activate

Updates the output.

CanvasID

Unique identifier of the element's canvas.

ElementID

Unique identifier of the element.

Node Output

ChangeAction

The action name.

UI:TextInput:SetChangeAction Node

Sets the action triggered when the text is changed.

Node Inputs

Activate

Updates the output.

CanvasID

Unique identifier of the element's canvas.

ElementID

Unique identifier of the element.

ChangeAction

The action name.

Node Output

Done

Sends a signal when the node's action is finished.

UI:TextInput:GetCursorBlinkInterval Node

Gets the cursor blink interval of the text input.

Node Inputs

Activate

Updates the output.

CanvasID

Unique identifier of the element's canvas.

ElementID

Unique identifier of the element.

Node Output

CursorBlinkInterval

The cursor blink in interval of the element identified by **ElementID**.

UI:TextInput:SetCursorBlinkInterval Node

Gets the cursor blink interval of the text input.

Node Inputs

Activate

Updates the output.

CanvasID

Unique identifier of the element's canvas.

ElementID

Unique identifier of the element.

CursorBlinkInterval

The cursor blink in interval of the element identified by **ElementID**.

Node Output

Done

Sends a signal when the node's action is finished.

UI:TextInput:GetEndEditAction Node

Gets the action triggered when the editing of text is finished.

Node Inputs

Activate

Updates the output.

CanvasID

Unique identifier of the element's canvas.

ElementID

Unique identifier of the element.

Node Output

EndEditAction

The action name.

UI:TextInput:SetEndEditAction Node

Sets the action triggered when the editing of text is finished.

Node Inputs

Activate

Updates the output.

CanvasID

Unique identifier of the element's canvas.

ElementID

Unique identifier of the element.

EndEditAction

The action name.

Node Output

Done

Sends a signal when the node's action is finished.

UI:TextInput:GetEnterAction Node

Gets the action triggered when **Enter** is pressed.

Node Inputs

Activate

Updates the output.

CanvasID

Unique identifier of the element's canvas.

ElementID

Unique identifier of the element.

Node Output

EnterAction

The action name.

UI:TextInput:SetEnterAction Node

Sets the action triggered when **Enter** is pressed.

Node Inputs

Activate

Updates the output.

CanvasID

Unique identifier of the element's canvas.

ElementID

Unique identifier of the element.

EnterAction

The action name.

Node Output

Done

Sends a signal when the node's action is finished.

UI:TextInput:GetIsPasswordField Node

Gets whether the text input is configured as a password field.

Node Inputs

Activate

Updates the output.

CanvasID

Unique identifier of the element's canvas.

ElementID

Unique identifier of the element.

Node Output

IsPasswordField

Boolean. Whether the element identified by **ElementID** is configured as a password field.

UI:TextInput:SetIsPasswordField Node

Sets whether the text input is configured as a password field.

Node Inputs

Activate

Updates the output.

CanvasID

Unique identifier of the element's canvas.

ElementID

Unique identifier of the element.

IsPasswordField

Boolean. Whether the element identified by **ElementID** is configured as a password field.

Node Output

Done

Sends a signal when the node's action is finished.

UI:TextInput:GetMaxStringLength Node

Gets the maximum number of characters that can be entered.

Node Inputs

Activate

Updates the output.

CanvasID

Unique identifier of the element's canvas.

ElementID

Unique identifier of the element.

Node Output

MaxStringLength

An integer representing the maximum number of characters that can be entered.

Valid values: 0 = none allowed | -1 = unlimited

UI:TextInput:SetMaxStringLength Node

Sets the maximum number of characters that can be entered.

Node Inputs

Activate

Updates the output.

CanvasID

Unique identifier of the element's canvas.

ElementID

Unique identifier of the element.

MaxStringLength

An integer representing the maximum number of characters that can be entered.

Valid values: 0 = none allowed | -1 = unlimited

Node Output

Done

Sends a signal when the node's action is finished.

UI:TextInput:GetPlaceholderTextEntity Node

Gets the placeholder text element.

Node Inputs

Activate

Updates the output.

CanvasID

Unique identifier of the element's canvas.

ElementID

Unique identifier of the element.

Node Output

PlaceholderTextElement

The placeholder text element.

UI:TextInput:SetPlaceholderTextEntity Node

Sets the placeholder text element.

Node Inputs

Activate

Updates the output.

CanvasID

Unique identifier of the element's canvas.

ElementID

Unique identifier of the element.

PlaceholderTextElement

The placeholder text element.

Node Output

Done

Sends a signal when the node's action is finished.

UI:TextInput:GetText Node

Gets the text string that the element is displaying or allowing to be edited.

Node Inputs

Activate

Updates the output.

CanvasID

Unique identifier of the element's canvas.

ElementID

Unique identifier of the element.

Node Output

Value

The text string being displayed or edited by the element

UI:TextInput:SetText Node

Sets the text string that the element is displaying or allowing to be edited.

Node Inputs

Activate

Updates the output.

CanvasID

Unique identifier of the element's canvas.

ElementID

Unique identifier of the element.

Value

The text string being displayed or edited by the element

Node Output

Done

Sends a signal when the node's action is finished.

UI:TextInput:GetTextCursorColor Node

Gets the color to be used for the text cursor.

Node Inputs

Activate

Updates the output.

CanvasID

Unique identifier of the element's canvas.

ElementID

Unique identifier of the element.

Node Outputs

Color

The RGB value (0 – 255 each for R, G, and B) of the element identified by **ElementID**.

Alpha

The alpha value (0 – 255) of the element identified by **ElementID**.

UI:TextInput:SetTextCursorColor Node

Sets the color to be used for the text cursor.

Node Inputs

Activate

Updates the output.

CanvasID

Unique identifier of the element's canvas.

ElementID

Unique identifier of the element.

Color

The RGB value (0 – 255 each for R, G, and B) of the element identified by **ElementID**.

Alpha

The alpha value (0 – 255) of the element identified by **ElementID**.

Node Output**Done**

Sends a signal when the node's action is finished.

UI:TextInput:GetTextEntity Node

Gets the text element.

Node Inputs**Activate**

Updates the output.

CanvasID

Unique identifier of the element's canvas.

ElementID

Unique identifier of the element.

Node Output**TextElement**

The text element.

UI:TextInput:SetTextEntity Node

Gets the text element.

Node Inputs**Activate**

Updates the output.

CanvasID

Unique identifier of the element's canvas.

ElementID

Unique identifier of the element.

TextElement

The text element.

Node Output**Done**

Sends a signal when the node's action is finished.

UI:TextInput:GetTextSelectionColor Node

Gets the color to be used for the text background when it is selected.

Node Inputs

Activate

Updates the output.

CanvasID

Unique identifier of the element's canvas.

ElementID

Unique identifier of the element.

Node Outputs

Color

The RGB value (0 – 255 each for R, G, and B) of the element identified by **ElementID**.

Alpha

The alpha value (0 – 255) of the element identified by **ElementID**.

UI:TextInput:SetTextSelectionColor Node

Gets the color to be used for the text background when it is selected.

Node Inputs

Activate

Updates the output.

CanvasID

Unique identifier of the element's canvas.

ElementID

Unique identifier of the element.

Color

The RGB value (0 – 255 for R, G, and B) of the element identified by **ElementID**.

Alpha

The alpha value (0 – 255) of the element identified by **ElementID**.

Node Output

Done

Sends a signal when the node's action is finished.

UI Animation Node

The **UI** animation flow graph node has been superseded by the **UIe Animation** (p. 1241) flow graph node. For best results, use the **UIe Animation** (p. 1241) flow graph node.

The UI animation node consists of the following node inputs and outputs:

UI:Sequence:Play Node

Controls playback of a UI animation sequence.

Node Inputs

Start

Starts playing the sequence from the beginning and triggers the **OnStarted** output.

Stop

Jumps the animation to the end and stops playing and triggers the **OnStopped** output.

Abort

Jumps the animation to the end and stops playing and triggers the **OnAborted** output.

Pause

Pauses the animation.

Resume

Continues playing a previously paused animation.

Reset

Resets the animation to the start. This applies all the key values for the first key frame of the animation.

CanvasID, SequenceName

Unique ID of UI canvas that contains the animation sequence.
The name of the sequence to play.

Node Outputs

OnStarted

Triggers when the sequence starts playing.

OnStopped

Triggers an output when the sequence stops playing, either because the end of the animation is reached or because the sequence is forced to stop (for example, by using the **Stop** node input).

OnAborted

Triggers an output when the sequence is aborted (for example, by using the **Abort** node input).

Virtual Reality

Lumberyard's [virtual reality \(p. 1376\)](#) system integrates the use of the OculusRift, HTC Vive, and Open Source Virtual Reality (OSVR) head-mounted displays (HMD) on PC gaming systems. Before using these head-mounted displays, read each manufacturer's safety guide:

- [Oculus Rift Health and Safety Warning](#)
- [HTC Vive Safety and Regulatory Guide](#)

To activate Lumberyard's virtual reality support, add the appropriate [Virtual Reality Gem\(s\) \(p. 1305\)](#) in the Project Configurator and then [rebuild your project \(p. 778\)](#). By enabling the appropriate Virtual Reality Gem(s), your project becomes capable of working with the supported virtual reality device(s), after some additional configuration. You can also [add new gems \(p. 778\)](#) for other head-mounted devices.

Use [console variables \(CVARs\) \(p. 1306\)](#) to activate and modify configurable features of the virtual reality system, such as resolution and performance specifications.

You can use flow graph modules for the initial game setup and game play scripting, for example to customize such features as the position of the camera, tracking of the attached virtual reality device, current view depending on height of the player, and more.

Topics

- [Configuring your Project for Virtual Reality \(p. 1305\)](#)
- [Configuring Required Console Variables \(p. 1306\)](#)
- [Setting Up Virtual Reality with Flow Graph \(p. 1307\)](#)
- [Previewing your Virtual Reality Project \(p. 1316\)](#)
- [Debugging your Virtual Reality Project \(p. 1316\)](#)

Configuring your Project for Virtual Reality

Add one or more Virtual Reality Gems available in Lumberyard Editor to enable virtual reality for supported head-mounted displays (HMDs). You can add the gem(s) to new or existing projects. If you add more than one gem, the system automatically detects which HMD is connected, and uses the appropriate gem code to control the specific HMD and any associated virtual reality (VR) controllers.

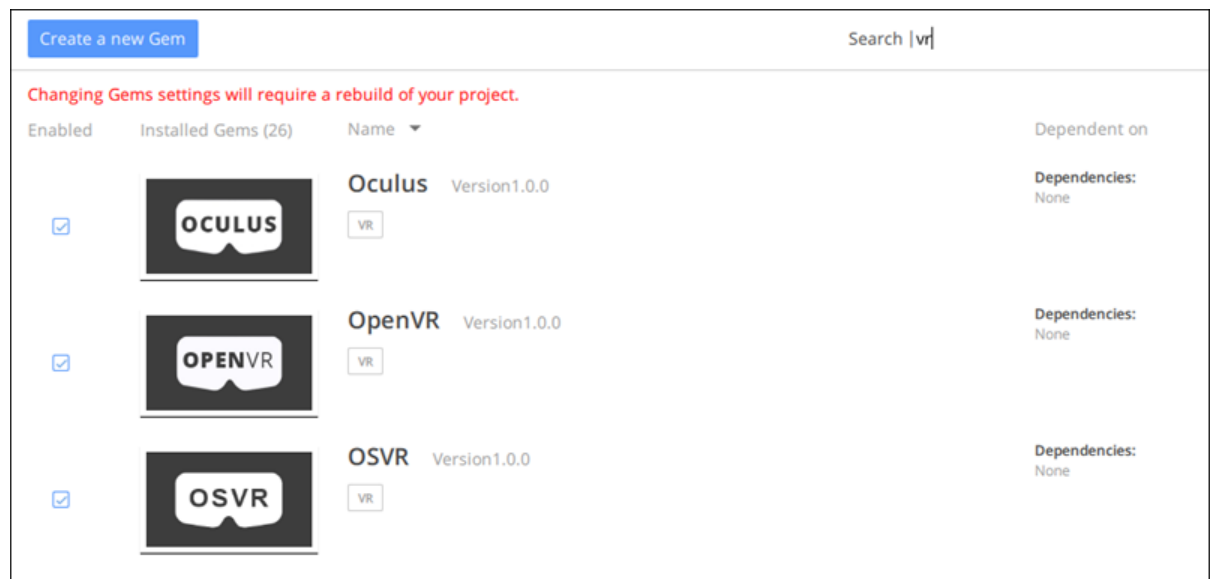
Supported HMDs include:

- **Oculus** – Oculus Rift HMD
- **OpenVR** – HTC Vive HMD
- **OSVR** – Open Source Virtual Reality (OSVR) HDK1 and HDK2

To add the Virtual Reality Gem(s)

1. Use the **Lumberyard Setup Assistant** to open the **Project Configurator**.
2. Select the project you want to add the Virtual Reality Gem to, or create a new project. Then click **Set as Default**.
3. Click **Enable Gems** below the project name.
4. Select one or more of the Virtual Reality Gems:

- **Oculus**
- **OpenVR**
- **OSVR**



5. Click **Save**.

After you enable the gem(s), you must [rebuild your project \(p. 778\)](#) before the gem(s) will function in Lumberyard Editor.

Configuring Required Console Variables

[Console variables \(CVARs\) \(p. 57\)](#) are a type of variable that you can manipulate in Lumberyard's [console interface \(p. 56\)](#).

You must set the following console variable to enable your project's capability to support the head-mounted display.

output_to_hmd = 1

Enables output to head-mounted display (HMD). Allows users to toggle stereoscopic output while playing the game. With this variable enabled, the height and width resolution for the connected headset is detected and set automatically.

Set the following console variables to 0 in order to turn them off. These features are either unnecessary for virtual reality or too resource-intensive for a virtual reality environment.

r_DepthOfField = 0

Disables the depth of field setting. 0 = disabled; 1 = enabled; 2 = hdr time of day enabled.

r_MotionBlur = 0

Disables the motion blur setting. 0 = no motion blur; 1 = camera and object motion blur; 2 = debug mode.

r_ResolutionScale

Float value. Scales the resolution for better performance. For example, set to 0.5 to scale the resolution by 50% in width and height (retains the aspect ratio).

e_gi = 0

Disables the global illumination setting. 0 = disabled; 1 = enabled.

Optional Console Variables

The following console variables are optional but strongly recommended. Disabling the following rendering features ensures better performance in a virtual reality environment. If you need certain rendering features that are explicitly disabled by these example variables, you may turn them back on at the cost of performance.

sys_spec = 2

Sets the system configuration specification to medium. 0 = custom; 1 = low; 2 = medium; 3 = high; 4 = very high; 5 = Xbox One; 6 = PS4; 7 = Mobile.

r_ssdoHalfRes = 3

Applies [screen space directional occlusion \(SSDO\)](#) (p. 1376) bandwidth optimizations to half resolution output. 0 = full resolution; 1 = lower resolution; 2 = low res depth (except for small camera field of views) to avoid artifacts; 3 = half resolution output.

r_Refraction = 0

Disables refraction. 0 = disabled; 1 = enabled.

r_CBufferUseNativeDepth = 0

Disables use of the depth buffer as the coverage buffer. 0 = disabled; 1 = enabled.

r_DeferredShadingTiled = 0

Disables tiled shading. 0 = disabled; 1 = tiled forward shading for transparent objects; 2 = tiled deferred and forward shading; 3 = tiled deferred and forward shading with debug info; 4 = light coverage visualization.

r_SSReflections = 0

Disables glossy screen space reflections. 0 = disabled; 1 = enabled.

Setting Up Virtual Reality with Flow Graph

You can use flow graph modules to set up or script your virtual reality game.

The following flow graph modules are available for any attached head-mounted display.

VR:ControllerTracking

Provides up-to-date information about any attached motion controller's transform (position and rotation) information in game world space. If an entity is specified in the node, all positions and rotations are specified relative to the entity.

Node Inputs

Enabled

Enables or disables the node.

Scale

Scales the controller's movements.

Node Outputs

Left pos

Position of the left controller.

Left Rot (PRY)

Rotation of the left controller in degrees (PRY – pitch, roll, yaw).

Left data ok

Valid data output from left controller. This means that the controller is connected and active.

Right pos

Position of the right controller.

Right Rot (PRY)

Rotation of the right controller in degrees (PRY – pitch, roll, yaw).

Right data ok

Valid data output from right controller. This means that the controller is connected and active.

VR:DeviceInfo

Gets information about the currently connected device.

Node Input

Activate

Updates the output.

Node Outputs

Name

The name of the active HMD.

RenderWidth

The render width for a single eye (in pixels).

RenderHeight

The render height for a single eye (in pixels).

VerticalFOV

The vertical field of view (FOV) for the HMD in degrees.

HorizontalFOV

The combined horizontal field of view (FOV) for both eyes in degrees.

VR:TransformInfo

Gives up-to-date information about the current HMD transform (position and rotation) in the game world space.

Node Outputs

Camera pos

Position of the camera after being translated by the HMD.

Camera rot

Rotation of the camera after being rotated by the HMD in degrees (pitch, yaw, roll).

HMD pos

Position of the HMD relative to the HMD's recentered pose. This is not the position of the HMD within the game world, as no camera transform has been applied.

HMD rot

Rotation of the HMD relative to the HMD's recentered pose. This is not the rotation of the HMD within the game world, as no camera transform has been applied.

VR:Dynamics:Controllers

Gives up-to-date information about the current HMD transform (position and rotation) in the game world space.

Node Input

Activate

Updates the outputs.

Node Outputs

Left Controller Active

Boolean. Whether left controller is active and being tracked.

Left Linear Velocity

Vector. Linear velocity of the left controller in local space.

Left Linear Acceleration

Vector. Linear acceleration of the left controller in local space.

Left Angular Velocity

Vector. Angular velocity of the left controller in local space.

Left Angular Acceleration

Vector. Angular acceleration of the left controller in local space.

Right Controller Active

Boolean. Whether right controller is active and being tracked.

Right Linear Velocity

Vector. Linear velocity of the right controller in local space.

Right Linear Acceleration

Vector. Linear acceleration of the right controller in local space.

Right Angular Velocity

Vector. Angular velocity of the right controller in local space.

Right Angular Acceleration

Vector. Angular acceleration of the right controller in local space.

VR:Dynamics:HMD

Provides information about the current angular and linear dynamics of the HMD.

Node Input

Enabled

Enables the node.

Node Outputs

Linear Velocity

Linear velocity of the HMD in local space.

Linear Acceleration

Linear acceleration of the HMD in local space.

Angular Velocity

Angular velocity of the HMD in local space.

Angular Acceleration

Angular acceleration of the HMD in local space.

VR:OpenVR:Playspace

Provides information about the HMD's playspace.

Node Input

Activate

Updates the outputs.

Node Outputs

Corner0

The world-space position of corner 0.

Corner1

The world-space position of corner 1.

Corner2

The world-space position of corner 2.

Corner3

The world-space position of corner 3.

Center

The world-space center of the playspace. Note that the center is on the floor.

Dimensions

The width (x) and height (y) of the playspace in meters.

IsValid

If true, the playspace data is valid and configured correctly.

VR:RecenterPose

Recenters the view coordinate system for the attached HMD to the current view.

VR:VREnabled

Queries whether VR output is enabled and active in the system. A `true` output from this node means that an HMD is connected, properly initialized, and being rendered to.

VR:SetTrackingLevel

Sets the current tracking level of the attached VR device to either **Head** or **Floor**. These options determine how the HMD's origin is calculated for every frame.

VR:TransformInfo

Provides information about the orientation and position of the camera and the HMD.

Node Input

Enabled

Enables the node.

Node Outputs

Camera pos

The position of the current camera in world coordinates.

Camera rot (PRY)

Vector. The orientation of the current camera in world coordinates in degrees (PRY – pitch, roll, yaw).

HMD pos

The position of the HMD with respect to the recentered pose of the tracker.

HMD rot (PRY)

The orientation of the HMD in world coordinates in degrees (PRY – pitch, roll, yaw).

VR:VREnabled

Whether VR rendering is enabled.

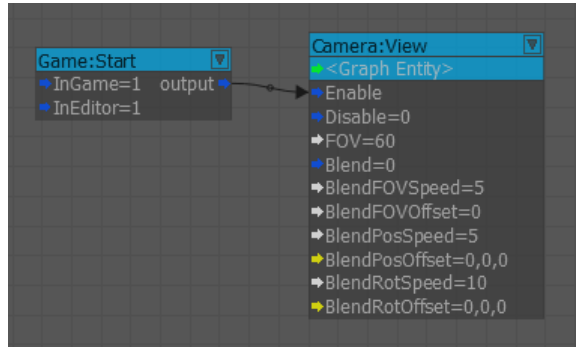
Setting Up a Basic Virtual Reality Flow Graph

The following [flow graph \(p. 487\)](#) examples guide you through a basic setup of a virtual reality level and its accompanying head-mounted display(s).

When you create a new level, the default point of origin is 0,0,0—the location at which the level starts during game play. You can specify a custom starting point by [placing a camera \(p. 296\)](#) at a specific location and enabling it with flow graph.

To specify a custom starting point

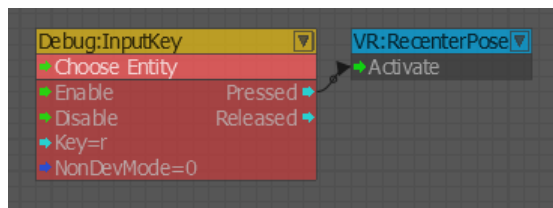
1. Place a [game play camera \(p. 296\)](#) at the desired location in your level. This is your default camera.
2. Right-click the camera entity and click **Create Flow Graph** to open the **Flow Graph** editor.
3. Drag a **Game:Start** node and **Camera:View** node onto your flow graph canvas.
4. Connect the **Game:Start** node's **output** output to the **Camera:View** node's **Enable** input.
5. Right-click **Choose Entity** and click **Assign graph entity**.



During virtual reality game play, a player may need to recenter their game play world around themselves, and start from a known position in space, regardless of their current position. Using flow graph, you can add a keyboard shortcut that the player can use to accomplish this.

To add a keyboard shortcut for recentering

1. Open the [Flow Graph](#) (p. 487) editor.
2. Drag a **Debug:InputKey** node and **VR:RecenterPose** node onto the canvas.
3. Connect the **Debug:InputKey** node's **Pressed** output to the **VR:RecenterPose** node's **Activate** input.
4. On **Debug:InputKey** node, click on **Key=** and set the key to the shortcut key you want to use.



For your virtual reality game, you may want to place a graphical, virtual controller to represent where a physical controller is within the 3D space. You can use flow graph to add this graphical representation of a controller (for example, hands, weapons, and so on).

For this procedure, the default camera is the game play camera that you placed in the [custom starting point](#) (p. 1311) procedure. Assigning the default camera entity to the **VR:ControllerTracking** node ensures that the motion controllers are aligned in the same space.

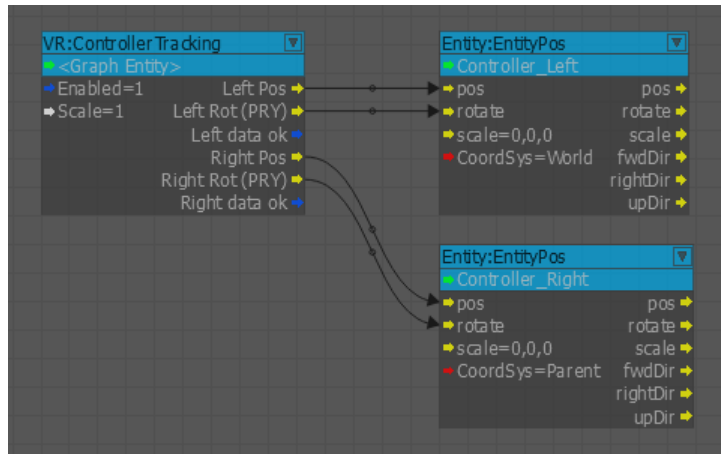
To add virtual controllers and assign the default camera

1. In the Perspective viewport, select the default camera.
2. In **Flow Graph** editor, drag the **VR:ControllerTracking** node onto the flow graph canvas.
3. Right-click **Choose Entity** and click **Assign Graph Entity**.
4. In the **Perspective** viewport, place one or more entities that you want to use as controllers into your level. Ensure that you keep the entity selected.
5. In the **Flow Graph** editor, drag one or two **Entity:EntityPos** nodes onto your flow graph canvas.
6. On the **Entity:EntityPos** node, right-click **Choose Input** and click **Assign selected entity**.

If you placed another entity that you want to assign as the other controller, select the entity and repeat this step for the other **Entity:EntityPos** node.

7. Connect **VR:ControllerTracking** node's **Lef Pos** output to the **Entity:EntityPos** node's **pos** input.

8. Connect **VR:ControllerTracking** node's **Left Rot (PRY)** output to the **Entity:EntityPos** node's **rotate** input.
9. Repeat the previous two steps for the **Right Pos** and **Right Rot (PRY)**, if applicable.



Setting Up a Custom Playspace with Flow Graph

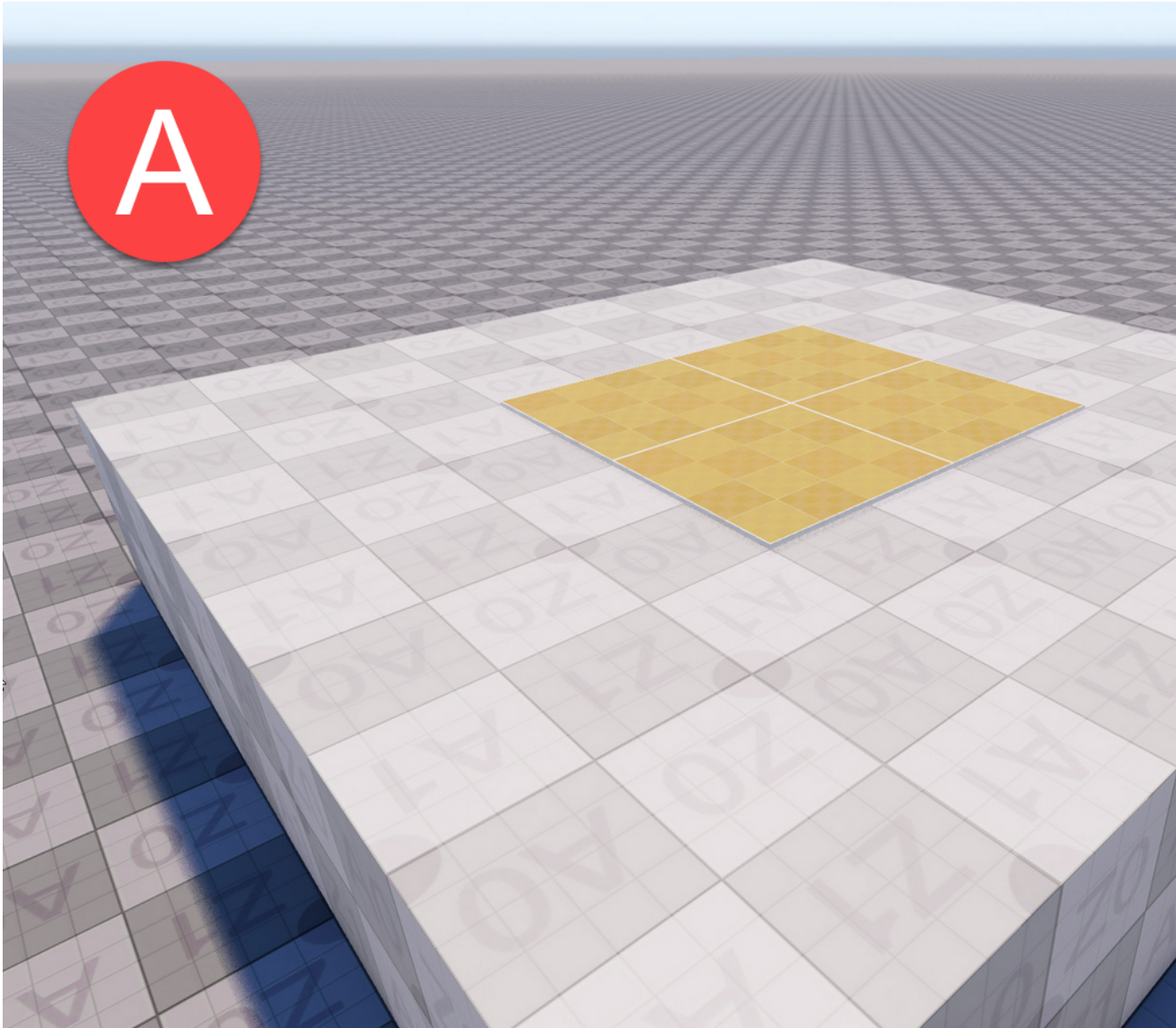
You can use flow graph to set up a custom playspace for OpenVR (Vive HTC head-mounted display [HMD]).

When a user first sets up their Vive HTC HMD, they must configure their playspace, or play area, according to their available space and room configuration. You can set up a flow graph to enable your game to access this information in order to spawn game objects within the user's reach and to create a visual area for the user to move within.

Lumberyard has included a sample virtual reality level. Open **VR_BoxGarden_Sample** in the [Samples Project \(p. 1091\)](#). You may copy or modify this sample level for your own uses.

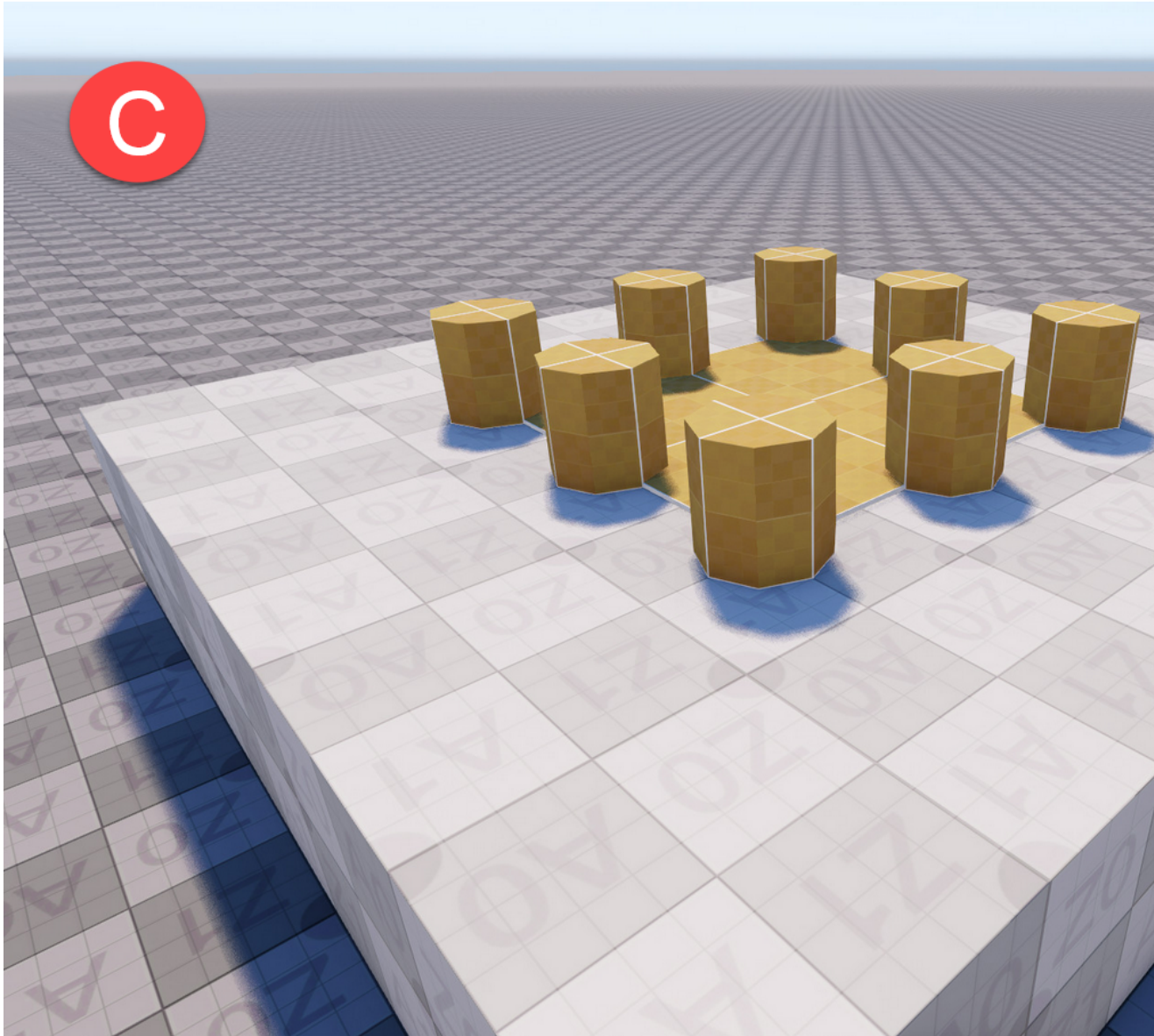
The following pictures show examples of:

- **A** – 2m x 2m playspace
- **B** – 4m x 3m playspace

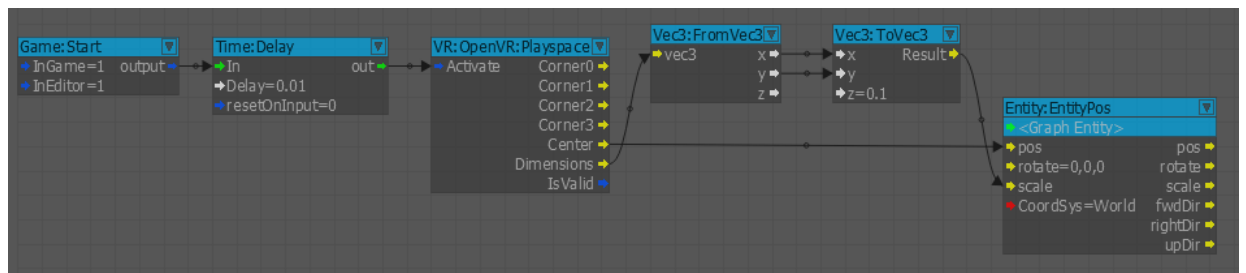


The following pictures show examples of spawning posts around the perimeter of a playspace with an area of:

- **C** – 2m x 2m
- **D** – 4m x 3m



The following flow graph from **VR_BoxGarden_Sample** in the [Samples Project \(p. 1091\)](#) shows an example of how to scale the user playspace to create the box shown in the above pictures. This flow graph uses the user playspace width (x) and length (y), and scales the height (z=0.1) to create an area equivalent to the user playspace, but at the height of a floorboard.



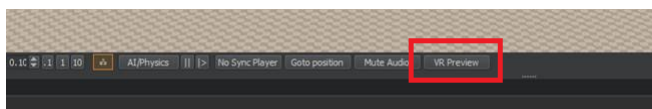
Previewing your Virtual Reality Project

You can preview your virtual reality project for any project that has one or more of the virtual reality head-mounted display gems enabled. As you work in Lumberyard Editor, use your head-mounted display to preview your virtual reality game. The preview display inside of Lumberyard Editor is a preview only; it is not a good indicator of how fast the application will perform outside of the editor.

To gauge the game performance outside of Lumberyard Editor, [create a release build \(p. 1366\)](#) to run your game in standalone mode.

To preview your virtual reality project

1. In Lumberyard Editor, click **VR Preview** on the bottom toolbar.



2. Enter game mode by doing one of the following:
 - Press **Ctrl + G**
 - On the main menu, click **Game, Switch to Game**.

To exit virtual reality preview mode

1. Exit game mode by pressing **Esc**.
2. Click **VR Preview** if you want to return to the default PC game preview mode.

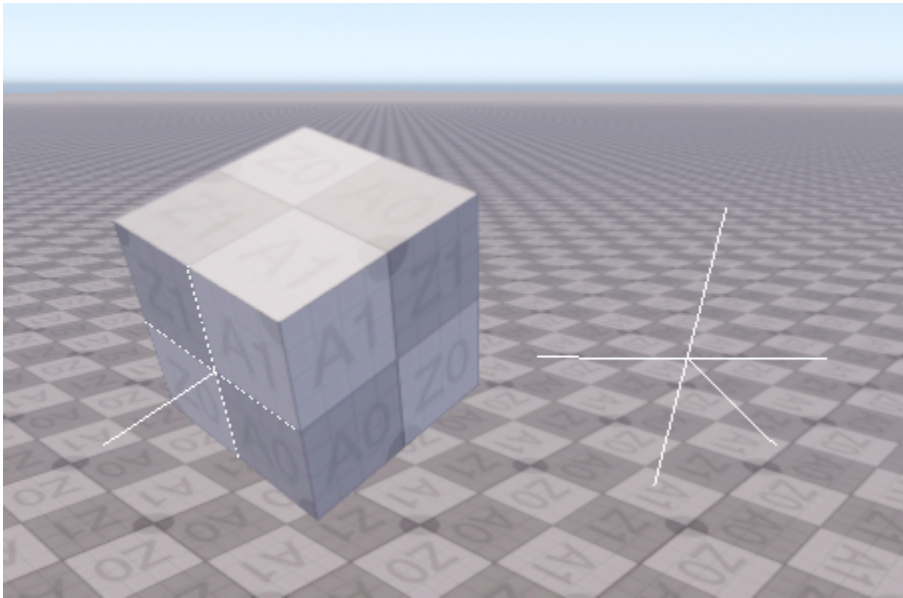
Debugging your Virtual Reality Project

You can debug your virtual reality project either through a running instance of the game or through the Lumberyard Editor. The head-mounted displays that Lumberyard supports outputs debugging information when debugging is enabled.

To enable debugging

- Set [console variable \(p. 1306\)](#) `hmd_debug` to 1 (enabled).

When in debug mode, motion controllers are rendered as white crosshairs. That is, if you assigned an object or entity to represent the motion controller in the game play world, then you will see it rendered with the white crosshairs. The following picture shows two controllers, one with render geometry assigned, and the other without.



Waf Build System

Lumberyard uses the Waf build system to allow you to switch between various build pipelines and to ensure you build only what is needed. You can use extensions, such as automatic project generation, or a simple GUI to modify the command line base system for your project requirements.

You can run Waf using the following methods:

- Command line window
- Waf-generated, Visual Studio solution file. Waf creates a Visual Studio solution file along with the projects specified in the selected project specs. If more than one spec file includes the same project, only one project file is created to prevent duplicates. Waf uses the project specs to determine the projects, project filters, and possible build configurations. Waf uses the wscript files to identify individual project definitions.

Note

Waf requires Python 2.6 or later.

Lumberyard includes the [Project Configurator \(p. 985\)](#), a standalone application that allows you to specify to Waf which game templates and assets (gems) to include in the game build.

Topics

- [Waf File System \(p. 1318\)](#)
- [Waf Commands and Options \(p. 1334\)](#)
- [Waf Supported Platforms and Compilers \(p. 1338\)](#)
- [Waf Project Settings \(p. 1339\)](#)
- [Waf Extensions \(p. 1342\)](#)
- [Using Waf \(p. 1344\)](#)
- [Adding User Settings to Waf \(p. 1357\)](#)
- [Adding Qt 5 Content to Waf \(p. 1360\)](#)
- [Using Uber Files \(p. 1362\)](#)
- [Debugging Waf \(p. 1363\)](#)

Waf File System

You can find global configurations and project specs in the `_WAF_` directory at the root project path. Three subfolders represent settings specific to the following build systems: `Android`, `iOS`, and `MSBuild`. Defined specs are located in the `specs` directory.

In addition to the configurations specified in the `_WAF_` directory, you can find other Waf settings in the `waf_branch_spec.py` file in the root directory. You can modify this file if you need to include support for additional platforms or configurations.

The Waf build file system can be grouped into three categories:

- Waf Module files (wscript)
- Waf file list (*.waf_files)
- Project and compilation files such as *.h, *.cpp, and so on

Topics

- [Waf File List \(*.waf_files\)](#) (p. 1319)
- [Waf Branch Spec \(waf_branch_spec.py\)](#) (p. 1320)
- [Waf Projects File \(project.json\)](#) (p. 1321)
- [Waf Spec Files \(*.json\)](#) (p. 1324)
- [Waf Module Files \(wscript\)](#) (p. 1326)
- [Waf Default Settings \(default_settings.json\)](#) (p. 1328)
- [Waf User Settings \(user_settings.options\)](#) (p. 1328)

Waf File List (*.waf_files)

Waf files are JSON-based and used to represent all files in the build plus their uber file and the VisualStudioFilter. By default, the uber file option is set to false. When the uber option is false, all files are treated as individual compilation units. `NoUberFile` is a fixed key that represents files that are individually compiled regardless of the uber file flag state.

Files are organized hierarchically into three levels:

- **Level 1 – Uber file target file**

The first level represents the uber file designation for the source files that are defined in the group. Uber file names must include the extension of the compilation types for the files defined. Only `.cpp` is supported. You can use the reserved name **NoUberFile** to prevent grouping the files defined into a single uber file, regardless of the Uber File option setting.

- **Level 2 – Visual Studio filter name**

The second level represents the Visual Studio project filter, which helps organize files in the group into user-defined folders and subfolders. Folder filter names can be shared across multiple uber file groupings because the folder groupings are not tied to uber file grouping definitions. The reserved name **root** represents the base of the project in the hierarchy.

- **Level 3 – List of source files**

The third level below each Visual Studio filter name group includes the source file names, relative to the current project folder.

The following is an example *.waf_files content file used by CryFont:

```
{
  "CryFont_Uber_0.cpp" :
  {
    "Source Files" :
    [
      "CryFont.cpp" ,
```

```
        "FFont.cpp",
        "FFontXML.cpp",
        "FontRenderer.cpp",
        "FontTexture.cpp",
        "GlyphBitmap.cpp",
        "GlyphCache.cpp",
        "ICryFont.cpp",
        "NullFont.cpp"
    ],
    "Header Files":
    [
        "CryFont.h",
        "FFont.h",
        "FontRenderer.h",
        "FontTexture.h",
        "GlyphBitmap.h",
        "GlyphCache.h",
        "NullFont.h",
        "resource.h",
        "FBitmap.h",
        "StdAfx.h"
    ]
},
"NoUberFile":
{
    "Root":
    [
        "StdAfx.cpp"
    ]
}
}
```

Waf Branch Spec (waf_branch_spec.py)

The `waf_branch_spec.py` is the topmost configuration level of the Waf build system. It specifies which platforms and configurations are available for all projects and specs.

The following is an example `waf_branch_spec.py` file:

```
#####
## Build Layout
BINTEMP_FOLDER = 'BinTemp'

#####
## Build Configuration
COMPANY_NAME = 'My Company'
COPYRIGHT = '(c) My Company'

#####
## Supported branch platforms/configurations
## This is a map of host -> target platforms
PLATFORMS = {
    'darwin' : [ 'darwin_x64', 'android_armv7_gcc', 'ios' ],
    'win32' : [ 'win_x64', 'win_x64_vs2012', 'win_x64_vs2010', 'durango',
    'android_armv7_gcc' ],
    'linux' : [ 'linux_x64_gcc', 'linux_x64_clang' ]
}
```

```
## And a list of build configurations to generate for each supported platform
CONFIGURATIONS = [ 'debug',          'profile',          'performance',
                   'release',
                   'debug_dedicated', 'profile_dedicated',
                   'performance_dedicated', 'release_dedicated' ]

## what conditions do you want a monolithic build ? Uses the same matching
rules as other settings
## so it can be platform_configuration, or configuration, or just platform
for the keys, and the Value is assumed
## false by default.
## monolithic builds produce just a statically linked executable with no
dls.

MONOLITHIC_BUILDS = {
    'release' : True,
    'release_dedicated' : True,
    'performance_dedicated' : True,
    'performance' : True,
    'ios' : True
}
```

The `waf_branch_spec.py` file manages the following global values:

Global values

Value	Description
BINTEMP_FOLDER	Subfolder under the base of the project where Waf stores all intermediate and temporary files
COMPANY_NAME	Company name to embed in the built executables
CONFIGURATIONS	List of possible build configurations
COPYRIGHT	Copyright header to embed in the built executables
MONOLITHIC_BUILDS	Build configurations mapped to monolithic flag values
PLATFORMS	Supported host platforms mapped to corresponding build platforms

Waf Projects File (project.json)

The `project.json` file (located in each game project directory) is used to store game project-specific data. The **enabled_game_projects** settings (`user_settings.options`) and the **enable-game-projects** build parameter use the project names defined in this file.

The `project.json` file is structured as follows:

- **First level** – Represents the project based on its name
- **Second level** – Presents attributes that you can set for each game project

The following is an example `project.json` file:

```
{
  "SamplesProject": {
    "product_name"      : "Samples Project",
    "executable_name"   : "SamplesProjectLauncher",
    "code_folder"       : "Code/SamplesProject",
    "project_directory" : "SamplesProject",
    "modules"           : [ "SamplesProject" ],

    "android_settings": {
      "package_name"   : "com.cryengine.sdk",
      "orientation"    : "landscape"
    }
  },
  "MultiplayerProject" : {
    "product_name"      : "Multiplayer Project",
    "executable_name"   : "MultiplayerProjectLauncher",
    "code_folder"       : "Code/MultiplayerProject",
    "project_directory" : "MultiplayerProject",
    "modules"           : [ "MultiplayerProject" ],

    "android_settings": {
      "package_name"   : "com.cryengine.sdk",
      "orientation"    : "landscape"
    }
  }
}
```

You can configure the following settings in the `project.json` file:

General settings

Value	Description
<code>android_folder</code>	(Android builds) Folder that includes the Android launcher project
<code>android_package</code>	(Android builds) Package name for the Android project
<code>code_folder</code>	Game code folder location, relative to the root of the SDK
<code>durango_settings</code>	(Durango) Root for the Durango-specific settings
<code>executable_name</code>	Name of the built executable file: <ul style="list-style-type: none"> • Dedicated server executables – <code>'_Server'</code> is appended to the name • Unit test executables – <code>'_UnitTest'</code> is appended to the name
<code>modules</code>	(List) Base modules for the game
<code>orbis_settings</code>	(Orbis) Root for the Orbis-specific settings
<code>product_name</code>	Externally-facing name of the product
<code>project_directory</code>	Project directory for the game project

The following values are only valid under the **durango_settings** key:

Durango settings

Value	Description
app_id	app_id value to set in the Appxmanifest.xml file
appxmanifest	appxmanifest entry defines the name of the Appxmanifest.xml template in the resource folder
background_color	background_color value to set in the Appxmanifest.xml file
description	description value to set in the Appxmanifest.xml file
display_name	display_name value to set in the Appxmanifest.xml file
foreground_text	foreground_text value to set in the Appxmanifest.xml file
logo	Path to the logo image to set in the Appxmanifest.xml file; the path must match a file in the resource folder
package_name	package_name value to set in the Appxmanifest.xml file
publisher	publisher value to set in the Appxmanifest.xml file
scid	scid value to set in the Appxmanifest.xml file; this value is also written to the durango_title_id.h file
small_logo	Path to the small_logo image to set in the Appxmanifest.xml file; the path must match a file in the resource folder
splash_screen	Path to the splash_screen image to set in the Appxmanifest.xml file; the path must match a file in the resource folder
store_logo	Path to the store_logo image to set in the Appxmanifest.xml file; the path must match a file in the resource folder
titleid	titleid value to set in the Appxmanifest.xml file; this value is also written to the durango_title_id.h file
version	version value to set in the Appxmanifest.xml file

The following values are only valid under the **orbis_settings** key:

Orbis settings

Value	Description
data_folder	Name of the data folder used for Orbis projects
nptitle_dat	Location of the <code>nptitle.dat</code> file to copy to the Orbis output folder
param_sfo	Location of the <code>param.sfo</code> file to copy to the Orbis output folder
trophy_trp	Location of the <code>trophy00.trp</code> file to copy to the Orbis output folder

Waf Spec Files (*.json)

You use Waf spec files to specify which modules to include in a build configuration. All settings are mandatory if not explicitly stated otherwise.

A typical spec includes all modules that are required to build a game project. Lumberyard includes the following with the engine SDK:

- `game_and_engine.json` – Specs to build the sample game and engine
- `resource_compiler.json` – Specs to build the Resource Compiler
- `pipeline.json` – Specs to build the pipeline tools
- `all.json` – Specs to build all projects

The following is an example `*.json` file that illustrates a spec file layout:

```
{
  "description"      : "Configuration to build my game",
  "visual_studio_name" : "My Game",
  "comment"         : "This is the build spec for my game",
  "disable_game_projects" : false,
  "platforms"       : ["win_x64"],
  "configurations"  : ["debug", "profile", "performance", "release"],
  "modules" :
  [
    "WindowsModule1",
    "WindowsModule2",
    "CommonModule"
  ]
}
```

Note

The `disable_game_projects` keyword does not compile the games specified in the `project.json` file. The default value is `false`, which means the specs compile the game projects by default.

Platform-specific Entry Values

You can apply the entry values in the table to targeted platforms and/or configurations. For example, a spec can build specific modules for Durango vs `win_x64` or a spec can build different modules in certain configurations.

- **modules** – Includes in the build all modules defined by this key, regardless of platform and configuration.
- **win_x64_modules** – Includes in the win_64 build all modules defined by this key, regardless of configuration.
- **durango_debug_defines** – Includes in the Durango debug build all defines specified by this key.
- **durango_modules** – Includes in the Durango build all modules defined by this key, regardless of configuration.

Overlapping lists are combined into a single list based on the build command. For example, for durango builds, the above example includes in the build the unique set of modules that is defined in both **modules** and **durango_modules**.

Spec File Format Specification

The general format of the JSON-based spec file is a dictionary of keyword values. The following table lists the possible keywords and their description.

Keyword Value	Description
comment	Additional comments to add to the spec file.
configurations	The list of configurations that this spec supports. In other words, the spec only builds the modules listed in the spec if the current configuration exists in the list of configurations. This is an AND condition with the platforms value.
description	Description of the spec file.
disable_game_projects	Flag that indicates that no game projects (as defined in <code>project.json</code>) are included in the build for this spec.
platforms	The list of platforms that this spec supports. In other words, the spec only builds the modules listed in the spec if the current target platform exists in this list of platforms.
<i>platform_configuration_defines</i>	<ul style="list-style-type: none">• <i>platform</i> and <i>configuration</i> are optional values.• Possible values for <i>platform</i> and <i>configuration</i> can be determined from the <code>waf_branch_spec.py</code> file.• The build uses the entry that matches the combination of these values and the build command.
visual_studio_name	Name of the generated Visual Studio solution that is used to distinguish this build spec from a build configuration.

Waf Module Files (wscript)

Wscript files are Python source files that have a fixed name and defined rules for the project folder. Waf picks up and processes the wscript file in each folder. Files can recurse into one or more subdirectories, define the build script for one or more modules, or both.

Wscript files are the main project script files for projects and can include the following:

- Specialized behavior for various Waf commands
- Different module types and entries
- Build rules for the folder
- Project- or target platform-specific definitions for compile, link, or other settings

Lumberyard includes a wscript file at the root folder that is used for the following:

- Loading all supported modules and tools relevant to a platform
- Importing all scripts necessary for configuring and building the engine
- Setting the available options that can be passed through the command line or in the default user options file located at `_WAF_/user_settings.options`
- Recursing into the `Code` and `Engine` folders at the root level

At the root is a compiled python script called `lmb_r_waf.bat` that executes the Waf commands through the root `wscript` file.

Lumberyard Engine Build Modules

The Lumberyard Waf system includes the following predefined build modules that can help define the build rules for system modules:

Build Module	Description	Consumers	Project Type
CryConsoleApplication	Build module for generic console applications	ShaderCacheGen	Executable
CryDedicatedServer	Build module for dedicated (server) game project launchers	ETDedicatedLauncher	Executable
CryEditor	Build module for Lumberyard Editor project	Editor	Executable
CryEngineModule	Build definition for CryEngine modules. Standard CryEngine modules autogenerate an RC file, if applicable.	Cry3DEngine, CryAction, CryAISystem, CryAnimation, CryEntitySystem, CryFont, CryInput, CryLiveCreate, CryMovie, CryNetwork, CryLobby, CryPhysics, CryScriptSystem, CrySoundSystem, CryAudioImplMiles,	Shared Library (non-release), Static Library (performance, release)

Build Module	Description	Consumers	Project Type
		CryAudioImplNoSound, CryAudioImplSDLMixer, CryAudioImplWwise, CrySystem, CryRenderD3D11, CryRenderOpenGL, CryRenderNULL, CryD3DCompilerStub	
CryEngineNonRCModule	Version of the CryEngineModule that does not attempt to create an RC file	CrySoundUnitTests, LyShine, AssetTaggingTools	Shared Library
CryEngineStaticModule	Build module to create static libraries	lua, md5, LZSS, Lzma, expat, DBAPI, zlib, lz4, PRT	Static Library
CryFileContainer	Build module that acts as a placeholder for source files	CryCommon, CryAudioCommon, EditorAudioControlsBrowser	Non
CryLauncher	Build module for game project launchers	ETPCLauncher	Executable
CryPipelineModule	Build module for pipeline components	CryTIFPluginCS4_11, CryExport2014, CryExport2015, CryExport2016, MayaCryExport22014, MayaCryExport22015, MayaCryExport22016	Custom
CryPlugin	Build module for Lumberyard Editor plugins	AssetTagging, CryDesigner, EditorDesc, EditorAnimation, EditorFbxImport, EditorGameDatabase, SchematycPlugin	Shared Library
CryPluginModule	Build module for Lumberyard Editor plugin modules	EditorCommon, PerforcePlugin	Shared Library
CryResourceCompiler	Build module for the resource compiler application	ResourceCompiler	Executable
CryResourceCompilerModule	Build module for resource compiler modules	CryPhysicsRC, CryXML, CryPerforce, ResourceCompilerABC, ResourceCompilerFBX, ResourceCompilerImage, ResourceCompilerPC, ResourceCompilerXML	Shared Library

Build Module	Description	Consumers	Project Type
CryStandAlonePlugin	Build module for Lumberyard Editor standalone plugins (does not link to any engine shared libraries)	EditorAudioControlsBrowser, EditorNoSound, EditorWwise, FBXPlugin, FFMPEGPlugin, MetricsPlugin, PrototypeEditorPlugin, StateMachineEditorPlugin, UiEditor	SharedLibrary
CryUnitTestLauncher	Build module for unit test launchers	UnitTestLauncher	Executable

Waf Default Settings (default_settings.json)

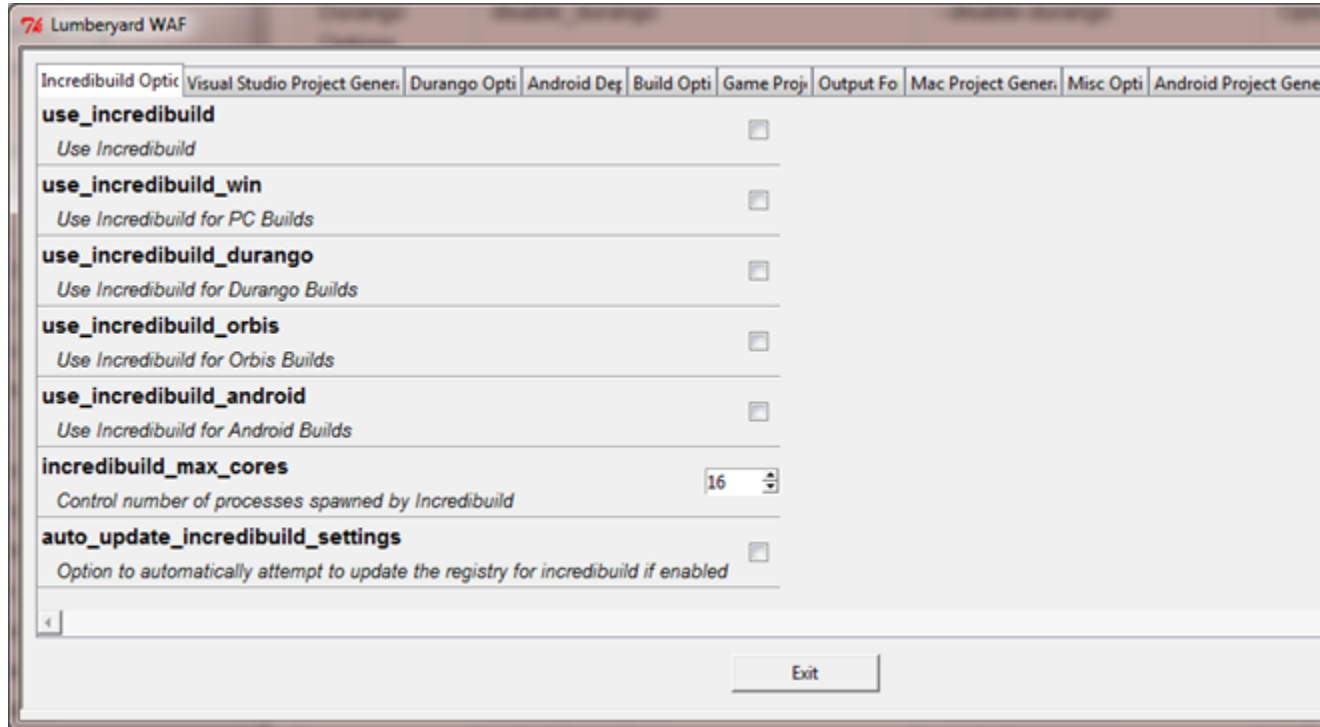
The Waf default settings file includes the default values for configurable Waf options. These values are used if custom values are not entered in the command line or `user_settings.options` cache file.

Waf User Settings (user_settings.options)

Global Waf build system settings are specified in the `user_settings.options` file located in the `_WAF_` subfolder. This file is automatically generated from the `default_settings.json` file if it does not exist. Every build that is run will refer to this file to get the option values specific to the builds. Any of the values can be overridden at a command prompt using **Override Parameter** column value in the table below. When a value is overridden, it is not updated in the `user_settings.options` file.

The settings listed below can be modified in the file directly, or through the **Lumberyard WAF Settings** dialog. To invoke the Settings dialog, type `show_option_dialog` command into Waf as follows:

```
lmb_r_waf.bat show_option_dialog
```



The tabs shown represent each section in the `user_settings.option` file.

Attribute	Override Parameter	Description	Default
Game Projects			
<code>enabled_game_projects</code>	<code>--enabled-game-projects</code>	Comma-separated list of game projects to enable for compiling	GameSDK, SamplesProject, MultiplayerProject, FeatureTests
Incredibuild Options			
<code>use_incredibuild</code>	<code>-i --use-incredibuild</code>	Use Incredibuild if available. This is a general flag; the specific platform value needs to be specified as per below.	False
<code>use_incredibuild_win</code>	<code>--use-incredibuild-win</code>	Use Incredibuild for Windows PC builds. This requires at a minimum the Make and Build tools package.	False
<code>use_incredibuild_durango</code>	<code>--use-incredibuild-durango</code>	Use Incredibuild for Durango builds. This requires at a minimum the Make and Build tools package and the Xbox One package.	False
<code>use_incredibuild_orbis</code>	<code>--use-incredibuild-orbis</code>	Use Incredibuild for Orbis builds. This requires at a minimum the Make and Build tools package and the PS4 package.	False

Attribute	Override Parameter	Description	Default
use_incredibuild_and_rose	use-incredibuild-android	Use Incredibuild for Android builds. This requires at a minimum the Make and Build tools package.	False
incredibuild_max_cores	incredibuild-max-cores	Control the number of processes spawned by Incredibuild.	128
auto_update_incredibuild_settings	auto-update-incredibuild-settings	Option to automatically attempt to update the registry for Incredibuild if needed. These registry updates are needed to configure Incredibuild to work properly with the WAF build system.	False
Build Options			
version	--force-version	The version of the game project to embed in the game launchers.	0.0.0.0
generate_debug_info	info-generate-debug-info	Option to generate debug symbols and .pdb files for the build.	True
generate_map_file	--generate-map-file	Generate a map file during linking if the platform supports it.	False
use_precompiled_header	use-precompiled-header	Use a precompiled header for compilation where applicable.	True
use_uber_files	--use-uber-files	Use uber files for compilation.	False
uber_file_size	--uber-file-size	Maximum content size of auto-generated uber files.	307200
max_parallel_link	--max-parallel-link	Controls the number of C++ linking operations that happen in parallel.	2
gems_optional	--gems-optional	Allows building of projects without gems.json files.	False
use_debug_code_generator	use_debug_code_generator	Uses the version of the code generator in the \Bin64.Debug folder instead of the \Bin64 folder.	False
Output Folder			
out_folder_win64	--output-folder-win64	Absolute or relative Win64 build output path. May have configuration-based extensions to the name based on the additional options listed below.	Bin64
out_folder_durango	--output-folder-durango	Absolute or relative Xbox One (Durango target) platform build output path. May have configuration-based extensions to the name based on the additional options listed below.	BinDurango

Attribute	Override Parameter	Description	Default
out_folder_mac64	--output-folder-mac64	Absolute or relative Mac (Darwin) target platform build output path. May have configuration-based extensions to the name based on the additional options listed below.	BinMac64
out_folder_orbis	--output-folder-orbis	Absolute or relative PS4 (Orbis) target platform build output path. May have configuration-based extensions to the name based on the additional options listed below.	BinOrbis
out_folder_ios	--output-folder-ios	Absolute or relative iOS target platform build output path. May have configuration-based extensions to the name based on the additional options listed below.	BinIos
out_folder_android_armv7	--output-folder-android-armv7	Absolute or relative Android/Armv7 build target platform output path. May have configuration-based extensions to the name based on the additional options listed below.	BinAndroid
output_folder_ext_debug	--output-folder-ext-debug	The output folder name extension for debug builds. This will be appended to the corresponding output folder based on the target platform builds.	Debug
output_folder_ext_profile	--output-folder-ext-profile	The output folder name extension for profile builds. This will be appended to the corresponding output folder based on the target platform builds.	
output_folder_ext_performance	--output-folder-ext-performance	The output folder name extension for performance builds. This will be appended to the corresponding output folder based on the target platform builds.	Performance
output_folder_ext_release	--output-folder-ext-release	The output folder name extension for release builds. This will be appended to the corresponding output folder based on the target platform builds.	Release
Misc Options			
max_cores	--max-cores	Number of parallel processes for local builds.	8
bootstrap_tool_params	bootstrap-tool-param	Optional parameters to pass to SetupAssistantBatch.exe as part of the bootstrap process.	
bootstrap_third_party_3rdparty_path	bootstrap-3rdparty-path	Optional parameter to pass the location of the \3rdParty folder as part of the bootstrap process.	

Attribute	Override Parameter	Description	Default
Visual Studio Project Generator			
generate_vs_projects	generate-automatically vs-projects-automatically	Automatically generate Visual Studio solutions.	True
visual_studio_solution_name	visual-studio-solution-name visual-studio-solution-name	Name of the generated Visual Studio solution.	LumberyardSDK
visual_studio_solution_folder	visual-studio-solution-folder visual-studio-solution-folder	Name of the folder in which the generated Visual Studio solution should be stored.	Solutions
specs_to_include_in_project_generation	specs-to-include-in-project-generation include-in-project-generation	List of specs to include in Visual Studio solution generation.	all, game, game_and_engine, resource_compiler
msvs_version	--msvs-version	Version of the Visual Studio Solution to generate. For more information, see https://en.wikipedia.org/wiki/Microsoft_Visual_Studio). (Don't include Without the decimal point).	12
Android Project Generator			
generate_android_projects	generate-automatically android-projects-automatically	Automatically generates Android projects.	False
android_projects_folder	android-projects-folder android-projects-folder	Solutions/android	Name of the folder in which the generated Android projects should be stored.
Android Deploy			
deploy_android	--deploy-android	Deploy to an Android device.	True
deploy_android_clean_device	deploy-android-clean-device android-clean-device	Removes any previous assets for the game project that were copied. If the deploy-android-executable option is specified as well then the package specified for deploy-android-package-name will also be uninstalled.	True
deploy_android_executable	deploy-android-executable android-executable	Deploys the executable .apk to the Android device.	True
deploy_android_replace_apk	deploy-android-replace-apk android-replace-apk	When installing the .apk to the Android device use the -r option to force the replacement of the package.	True
deploy_android_root_dir	deploy-android-root-dir android-root-dir	Root folder to deploy the assets to on the Android device	/storage/emulated/legacy

Attribute	Override Parameter	Description	Default
deploy_android_install_options	deploy-android-install-options	Additional options to specify for the install command.	
deploy_android_paks	deploy-android-paks	Forces .pak files to be built in non-release builds.	False
iOS Project Generator			
generate_ios_projects_automatically	generate-ios-projects-automatically	Automatically generates iOS projects.	True
ios_project_name	--ios-project-name	Name of the generated iOS project.	LumberyardiOSSDK
ios_project_folder	--ios-project-folder	Name of the folder in which the generated iOS projects should be stored.	Solutions
Mac Project Generator			
generate_mac_projects_automatically	generate-mac-projects-automatically	Automatically generates Darwin projects.	True
mac_project_name	--mac-project-name	Name of the generated project	LumberyardSDK
mac_project_folder	--mac-project-folder	Name of the folder in which the generated Darwin projects should be stored.	Solutions
Durango Options			
disable_durango	--disable-durango	Option to disable Durango.	False
deploy_durango_clean_target	deploy-durango-clean-target	Clean up the Durango target before deploying.	True
deploy_durango	--deploy-durango	Deploy to Durango.	True
deploy_durango_kits	--deploy-durango-kits	Deploy to a list of Durango kits. Does not include the default kit by default.	
durango_paks	--durango-paks	Forces .pak files to be built in non-release builds.	False
Orbis Options			
disable_orbis	--disable-orbis	Disables Orbis.	False
deploy_orbis_clean_target	deploy-orbis-clean-target	Clean up the Orbis target before deploying.	True
deploy_orbis	--deploy-orbis	Deploy to Orbis.	True

Attribute	Override Parameter	Description	Default
deploy_orbis_executable	--deploy-orbis-executable	Deploy the executable to the dev kit. This is not needed for debugging as the debugger downloads it when running. Disabling this improves debug iteration time, however if you want to run from the console without a debugger then you should leave this enabled.	True
deploy_orbis_kits	--deploy-orbis-kits	Deploy to a list of Orbis kits. Does not include the default kit by default.	
orbis_paks	--orbis-paks	--orbis-paks	False

Waf Commands and Options

Before building a project using Waf, you must run `configure` from the command line. The `configure` command recursively processes all of the `wscript` configuration files starting from the root directory and generates a Visual Studio solution file for the entire project. You can set an option to generate a solution file during the `configure` command.

Note

The Waf script automatically runs Lumberyard Setup Assistant to ensure the correct third-party libraries are available and the proper links are created to compile the game code, engine and asset pipeline, and editor and tools.

Waf Configuration

To run the Waf executable, run the following command at `engine_root\dev\` of your project:
`lubr_waf configure`

This command iterates through all the Waf project configuration files and sets up the project-specific settings in the Waf cache, which is used in subsequent build commands. It also uses the host environment to determine which platforms are available to build.

The following example shows the output of the `lubr_waf configure` command:

```
[WAF] Executing 'configure'
Running SetupAssistant.exe...
--- Lumberyard Setup Assistant ---
SDK location: d:/lumberyard_engine/dev
Third party location: d:/lumberyard_engine/dev/3rdParty
Capabilities Available, [x] enabled - [ ] disabled:
[ ] runtime - Run your game project
[ ] runeditor - Run the Lumberyard Editor and tools
[X] compilegame - Compile the game code
[X] compileengine - Compile the engine and asset pipeline
[X] compilesandbox - Compile the Lumberyard Editor and tools
[ ] compileandroid - Compile for Android devices
[ ] compileios - Compile for iOS devices
Successfully executed
[INFO] Configure "win_x64 - [debug, profile, performance, release,
debug_dedicated, profile_dedicated, performance_dedicated,
release_dedicated]"
```



```
[INFO] Configure "win_x64_vs2012 - [debug, profile, performance,
release, debug_dedicated, profile_dedicated, performance_dedicated,
release_dedicated]"
Unable to find Visual Studio 2012, removing build target
[INFO] Configure "win_x64_vs2010 - [debug, profile, performance,
release, debug_dedicated, profile_dedicated, performance_dedicated,
release_dedicated]"
Unable to find Visual Studio 2010, removing build target
[INFO] Configure "durango - [debug, profile, performance, release,
debug_dedicated, profile_dedicated, performance_dedicated,
release_dedicated]"
[INFO] Configure "android_armv7_gcc - [debug, profile, performance,
release, debug_dedicated, profile_dedicated, performance_dedicated,
release_dedicated]"
[WARN] android_armv7_gcc setup failed, removing target platform
[WAF] 'configure' finished successfully (10.335s)
[WAF] Executing 'generate_uber_files' in 'd:\ws\lyengine\dev\BinTemp'
[WAF] 'generate_uber_files' finished successfully (2.177s)
[WAF] Executing 'msvs' in 'd:\ws\lyengine\dev\BinTemp'
```

The `configure` command uses the settings defined in the `user_settings.options` file that is located in the `_WAF_` subfolder. You can edit this file in a text editor or by using the built-in settings editor: `lmb_r_waf.bat show_option_dialog`.

If you set the option to generate a Visual Studio solution to `true`, a solution file is created in the directory specified in the `user_settings.option` file. If you do not modify the `user_settings.option` file, the Visual Studio solution is in `root_folder/Solutions/LumberyardSDK.sln` by default.

Build Configuration

After configuring Waf, you can run the `build` command.

Here is an example showing syntax: `lmb_r_waf.bat build_<platform_configuration> -p spec`

The following commands and options are available:

- `configure` – Must be run before any `clean` or `build` command. Loads all modules, configs, and project specs; validates and sets up the working cached build Python file.
- `build_*` – Builds the specified project spec for the specified platform and configuration.
- `clean_*` – Cleans out intermediate and target files that were generated for the particular platform and configuration.

Here's an example of how to build release for Windows x64: `lmb_r_waf.bat build_win_x64_release -p all`

Here's an example of how to clean the build release for Windows x64: `lmb_r_waf.bat clean_win_x64_release -p all`

Note

Combining the `clean_*` and `build_*` commands is the equivalent of performing a rebuild.

Configure command options

Command	Command Option	Description
<code>build_*</code> , <code>clean_*</code>	<code>-p <spec name> --project-spec=<spec name></code>	The spec name to use to build or clean a project.

Command	Command Option	Description
	<code>--targets = <i>target1,target2,...</i></code>	Optional flag to filter on which targets to build. The targets must be included in the project spec above in order for this to work.
<code>build_*</code> , <code>clean_*</code> , <code>configure</code>	<code>--profile-execution = (True False)</code>	The option to run the build process through the Python execution profiler. As this will produce a large output to the console, we recommend that you redirect the output of this command to a log file.
<code>build_*</code>	<code>-- execsolution=<i>VS_solution</i></code>	This internally-generated command line is a Visual Studio solution that provides a way to build Waf commands invoked from the VS IDE to apply additional overrides that can be defined in the <code>.vcxproj</code> files themselves.
<code>build_*</code>	<code>--internal-dont-check- recursive-execution= (True False)</code>	This internally-generated command prevents infinitely recursing the Waf command line when Incredibuild is enabled. Incredibuild is invoked by passing the entire command line that was entered into the Incredibuild BuildConsole (or XgConsole). This flag is added so that the script will know that the command is already being processed by Incredibuild and does not need to be passed in again. This option should never be explicitly specified anywhere besides the main Wscript.
<code>build_*</code>	<code>--file- filter=<i>source_files</i></code>	An option to pass in a comma-separated list of absolute paths to source files to filter the build on. This option is useful to build specific files.
<code>build_*</code>	<code>--show-includes=(True False)</code>	Option to show the <code>#include</code> tree that a file uses during compilation. This option is only valid when <code>--file-filter</code> is specified.
<code>build_*</code>	<code>--show-preprocessed- file = (True False)</code>	Option to generate a preprocessor output for a source file (but not actually build the file). The generated file is saved in the variant folder under <code>\bintemp</code> based on the location of the source file. This option is only valid when <code>--file-filter</code> is specified.
<code>build_*</code>	<code>--show-disassembly = (True False)</code>	Option to generate an Assembler output for a source file (but not actually build the file). The generated file is saved in the variant folder under <code>\bintemp</code> based on the location of the source file. This option is only valid when <code>--file-filter</code> is specified.

Command	Command Option	Description
configure	--update-settings = (True False)	Option to update the <code>user_settings.options</code> file with any values that are modified from the command line. For instance, if you want to modify the value of <code>use_uber_files</code> in <code>user_settings.options</code> , set <code>--use-uber-files=True</code> in the command line for <code>configure</code> and add <code>--update-settings=True</code> to apply the changes to <code>user_settings.options</code> .

You can set the command options at build time. These options override the values set in the `user_settings.option` file. For more information, see [Project Configurator \(p. 1328\)](#).

Only modules that support each project configuration are built from the project spec. If a module is defined in the spec that only can be built in debug or profile, building in performance mode excludes that project from compilation.

Project configurations parameters

Configuratio	Asserts	Profiling	Optimization	Logging	Description
debug	Yes	All	Minimum	Yes	Slowest – Focuses on debugging with asserts enabled, all profiling features enabled, and logging enabled.
profile	No	All	Medium	Yes	Fast – Strikes a balance between debugging and performance with all profiling features and logging enabled.
performance	No	Few	Maximum	No	Very fast – Performance similar to release but has some profiling features enabled; difficult to debug; no logging.
release	No	None	Maximum	No	Fastest – Highest performance; most difficult to debug; no profiling features; no logging.

Build command project spec options

Spec	Platform	Configuration	Description
all	win_x64	Debug, profile, performance, release	Configuration to build the engine, editor, plugins, and tools
game_and_engine	win_x64	Debug, profile, performance, release	Configuration to build the engine and game project

Spec	Platform	Configuration	Description
pipeline	win_x64	Debug, profile	Configuration to build tools for the asset pipeline
resource_compiler	win_x64	Debug, profile	Configuration to build the Resource Compiler only

Build configuration options

Option	Description
--progress	Shows the build progress and updates in real time.
--project-spec	Specifies the project spec to use when cleaning or building the project.
--show-includes	Shows the includes for each compiled file.
--target	Specifies the target to build and its dependencies. The target must exist in the specified project spec; otherwise, all targets in the project spec are built.

Multiplayer Configuration

Before you can build multiplayer information, you must build the dedicated server. This creates a directory called `Bin64.Dedicated` that includes the binaries directory and configuration files for dedicated server.

To build the dedicated server, run the following command:

```
lmb_r_waf.bat build_win_x64_profile_dedicated -p dedicated_server
```

Waf Supported Platforms and Compilers

This topic provides information about the platforms and compilers that Waf supports. For more information about supported configurations, see [Waf Commands and Options \(p. 1334\)](#)

Supported platforms

Platform	Build Environment	Waf Short Name
64-bit Windows	MSBuild / Visual Studio 2013	win_x64
64-bit Windows	MSBuild / Visual Studio 2012	win_x64_vs2012
64-bit Windows	MSBuild / Visual Studio 2010	win_x64_vs2010

The following compilers are supported based on the build platform.

Supported compilers

Compiler	Windows 64-Bit
MSVC 10.0 (Visual Studio 2010)	Yes (only for CryExport2014)
MSVC 11.0 (Visual Studio 2012)	Yes (only for CryExport2015)
MSVC 12.0 (Visual Studio 2013)	Yes (except for CryExport2014 and CryExport2015)
GCC	No
Clang	No

Waf Project Settings

When defining a project's build settings (wscript), you can specify several different project settings for the build modules to configure the correct parameters for the project.

The following table provides the valid attributes for the different build modules.

Build attributes

Attribute	Description	Target to Platform or Configuration
additional_manifests	Additional manifests to add to MSVC applications	Y
additional_settings	<p>Container that groups compile settings and acts upon them recursively; useful for specifying options for particular files in a project</p> <p>For example, you can disable precompiled headers for a specific file using the following:</p> <pre>... additional_settings = Settings (files = 'my_file.cpp', disable_pch=True)</pre>	Y
build_in_dedicated	True by default; if False, the module will not be built when building in dedicated server mode	N
cflags	Additional C flags to pass to the compiler	Y
create_appdata	(Durango) Creates and/or collects all resources to create a Durango application	N
cxxflags	Additional CXX flags to pass to the compiler	Y
defines	List of additional pre-processor defines for the project	Y

Attribute	Description	Target to Platform or Configuration
export_definitions	List of export definitions to export using the /DEF: compiler option	Y
features	Additional custom features to apply to the project during the build	Y
file_list	List of file specs that contain the files to include in the project	Y
files	List of files to include for the module	N
force_dynamic_crt	Forces the use of dynamic runtime CRT for the project	N
force_static_crt	Forces the use of static runtime CRT for the project	N
framework	(Darwin) Specifies the framework to use	Y
includes	Additional include paths for the module	Y
lib	Additional input libraries to link against	Y
libpath	Additional library paths for the module	Y
linkflags	Additional linker flags to pass to the linker	Y
meta_includes	Additional meta includes for WinRT using the /AI compiler option	Y
meta_includes	(Durango) Specifies the include path for Durango WinRT include files	Y
need_deploy	Hint to deploy the module before debugging in Visual Studio	N
output_sub_folder	Optional subfolder under the target output folder in which to copy the module binary	N
pch	Specifies the precompiled header (PCH) file, if in use	N
platforms	List of platforms to restrict the module to build on; if missing, a specific platform will not be targeted at the project definition level Example: platforms = ['durango'],	N
priority_includes	Same as the includes paths, except this include list is added prior to the ones defined in the includes paths	N
qt_to_moc_files	List of files for the QT5 moc processor to process; these files must reside in the file list specified by the file_list attribute (attribute required if the qt5 feature is applied to the module)	N

Attribute	Description	Target to Platform or Configuration
source	List of source files to add directly to the project	N
target	Project name of the target	N
use	List of static library modules that are part of the Waf build to which you can add dependencies and static links	Y
use_module	List of static library modules that are part of the Waf build to which you can add dependencies and static links	Y
vs_filter	Folder filter in the generated solution file where this project exists	N

Platform and Configuration Targeting

If allowed (refer to the third column in the table above), you can set an attribute value to apply only under certain target platforms and configurations. Each attribute can universal for all builds or targeted specifically to a platform/configuration combination:

- **[Attribute]** – Applies to any target platform/configuration for the attribute
- **[target_platform]_[attribute]** – Applies to any configuration for a specific target platform for the attribute
- **[configuration]_[attribute]** – Applies to a specific configuration for any target platform for the attribute
- **[target_platform]_[configuration]_[attribute]** – Applies to a specific target platform and configuration for the attribute

Features

The Lumberyard Waf system allows the use of custom features to add functionality to a project's build pipeline.

Build features

Feature	Description
qt5	Passes files through the QT5 moc processor; if this feature is set for a project, you must also pass the qt_to_moc_files
generate_rc_file	Creates an RC file and copies the resources, such as the icon file; win_x64 only
wwise	Sets the following for building and linking against Wwise: environment, includes, libraries, and library paths
GoogleMock	Sets the following for building and linking against Google Mock: environment, includes, libraries, and library paths

Feature	Description
AWSNativeSDK	Sets the following for building and linking against the AWS Native SDK library: environment, includes, libraries, and library paths
AWSGameLift	Sets the following for building and linking against the AWS GameLift library: environment, includes, libraries, and library paths
GridMate	Sets the following for building and linking against the GridMate library: environment, includes, libraries, and library paths

Waf Extensions

Compiling with Incredibuild

Waf supports IncrediBuild 6.0 or later, and allows for distributed network builds for compiling larger projects.

You must have the appropriate package for your platform:

- **Windows or Android** – IncrediBuild for Make and Build
- **Durango (Xbox One)** – IncrediBuild for Xbox One
- **Orbis (PlayStation 4)** – IncrediBuild for PlayStation 4

To verify which package is configured for your machine, run the following command (located in C:\Program Files (x86)\Xoreax\IncrediBuild): `xgConsole.exe /QUERYLICENSE`

The following is output:

```
> xgConsole.exe /QUERYLICENSE

License details:
-----
Registered to: My Game Company
Up to XX Agents allowed
Maintenance expires on XX/XX/XXXX

Packages installed:
-----
- IncrediBuild for Make && Build Tools
```

To do this	Run this at a command line
Enable or disable IncrediBuild builds	<p><code>use_incredibuild</code> Instructs Waf to use <code>incredibuild</code> to distribute and parallelize the build, if possible. You need to specify the type of <code>incredibuild</code> package based on the platform.</p> <p>For Win x64, you must add a <code>--use-incredibuild-win</code> parameter and install the <code>Make and Build Tools</code> package.</p>

To do this	Run this at a command line
	<p>For Durango, you must add a <code>--use-incredibuild-durango</code> parameter and install the Xbox One Extension package.</p> <p>For Orbis, you must add a <code>--use-incredibuild-orbis</code> parameter and install the Playstation Extension package.</p>
Adjust the maximum number of parallel tasks	<code>incredibuild_max_cores</code>
Determine which IncrediBuild package is configured for your machine	<code>xgConsole.exe /QUERYLICENSE</code>

Waf requires certain packages and the Windows registry key settings below to run IncrediBuild. Run `lmb_r_waf.bat` in **Administrator** mode to edit the registry.

Modify the settings in the Windows registry under the following key:

`HKEY_LOCAL_MACHINE\\Software\\Wow6432Node\\Xoreax\\Incredibuild\\Builder`

Registry Settings

Name	Value	Description
<code>PdbForwardingMode</code>	0	Controls the way IncrediBuild handles PDB files. Required for Waf.
<code>MaxConcurrentPDBs</code>	0	Controls how many files can be processed in parallel. This optimization is also useful for MSBuild.
<code>AllowDoubleTargets</code>	0	Controls whether or not remote processes can be restarted on the local machine when possible. This option is required to prevent any compiler crashes.

To enable IncrediBuild

1. Open the `user_settings.options` file located in `/_WAF_/Settings`.
2. In the `user_settings.options` file, under **[IncrediBuild Options]**, do the following:
 - Set the `use_incredibuild` flag to `True`.
 - Set the `use_incredibuild_win` flag to `True`.
3. Save your changes.

Compiling with QT

Waf supports compiling QT5 `.moc` Meta-Object-Compiler files. To enable or disable compiling of particular files, add the `qt5` feature to your Waf Module (wscript) file and then add the list of files to be compiled.

The following example shows a Waf Module (wscript) file:

```
# wscript relative path
```

```
QT_TO_MOC_FILES = [  
    'MyQTFile.h',  
    'MyOtherQTFile.h',  
    ...  
]  
  
def build(bld):  
    bld.CryPlugin(  
        target      = 'MyQTPlugin',  
        vs_filter   = 'Plugins',  
        file_list   = 'file_list.waf_files',  
  
        features    = ['qt5'], # add the QT5 moc feature to this Waf  
module  
    )
```

Compiling with Visual Studio

Waf has limited support for the Visual Studio 2013 IDE. Once you run the `configure` command to generate a Visual Studio solution, you can invoke Waf through the IDE and open the solution file in Visual Studio 2013.

Waf creates a Visual Studio solution file along with the projects specified in the selected project specs. If more than one spec file includes the same project, only one project file is created to prevent duplicates. Waf uses the project specs to determine the projects, project filters, and possible build configurations. Waf uses the `wscript` files to identify individual project definitions.

To select the active solution configuration

1. Open the solution file in Visual Studio 2013.
2. Select **Build, Configuration Manager**.
3. In the Configuration Manager dialog box, select **[All] Debug** from the **Active solution configuration** drop-down list. This option builds all x64 modules in debug mode.
4. Click **Close**.

Once the active solution configuration is set, you can build the solution.

To build the solution in Visual Studio 2013

1. Select **Build, Build Solution**. This builds all modules defined in the all project spec.
2. Once the build is successful, you can choose different solution configurations based on your active projects. For example, if you are working on the game (`game_and_engine` spec), you wouldn't need to build everything. Or if you want to build a profile configuration of the build, you can use **[All] Profile**.

Using Waf

This topic demonstrates how you can use Waf the following ways:

- [Adding a Game Project \(p. 1345\)](#)
- [Adding a Spec \(p. 1347\)](#)
- [Adding a Build Module \(p. 1349\)](#)

Adding a Game Project

The simplest and recommended method to add a game project to the Lumberyard Waf build system is to use the Project Configurator. The Project Configurator is a standalone application for telling the Waf build system which game projects and assets to include in a game build. For more information, see [Project Configurator \(p. 985\)](#).

You can also add a game project with the following steps:

- Create the project definition
- Create a game module
- Update the user settings to include the game

Note

You can build your game project by creating a game project first (see steps below) and then creating a spec for just the game (no modules, just basic spec values):

```
{
  "description": "Configuration to build the My Game",
  "visual_studio_name": "My Game"
}
```

When the project is properly defined and all source files are in the correct locations, you can set the **enabled_game_projects** value in the `user_settings.options` file. Configuring this value limits the Visual Studio solution to the launcher projects and your game project.

Creating the Project Definition

In the following procedure you set `Code/MyGame` as the project source folder and `MyGame` as the project folder. The `code_folder` points to your game's module root and the `project_directory` points to the game-specific assets. You can define any number of game projects in this file and you can configure which ones to build.

To create the project definition

1. Navigate to the SDK root and locate the `Code` folder and `project_directory`. Typically your game code folder should reside under these locations.
2. Determine the name for your project. For this example use **My Game**.
3. Add the definitions for the new game project to the `project.json` file (located in the game project folder under the `\dev` directory). For this example add **My Game** to the SDK:

```
{
  "project_name": "My Game",
  "product_name": "My Game",
  "executable_name": "MyGame",
  "code_folder": "Code/MyGame",
  "modules" : [ "MyGame" ]
}
```

Creating a Game Module

You can create a game module after setting the game project definition. Game modules include `wscript` files, source files, and a `waf_files` configuration file. You must create separate folders for the game source code and for the resources. Both should reside under the `code_folder` specified earlier.

For this example you create folders called **GameSource** and **Resources** under the `Code/MyGame` directory.

Create a wscript file

Because Waf searches for and discovers wscript files recursively through other wscript files, you must include a simple wscript file in the `Code/MyGame` folder that recurses to the `GameSource` folder.

Create a file with the following:

```
SUBFOLDERS = [ 'GameSource' ]

def build(bld):
    bld.recurse(SUBFOLDERS)
```

Next you must create the source code in the `GameSource` folder. Include in this folder all of your source files and the corresponding Waf source file configuration (for example, `MyGame.waf_files`) to include your game files.

Create a wscript in the `GameSource` folder to define the build configuration for your game:

```
def build(bld):
    bld.CryEngineModule(

        target      = 'MyGame',
        vs_filter    = 'Game/MyGame',
        file_list    = 'MyGame.waf_files',
        pch          = 'StdAfx.cpp',
        includes     = [ '.', '..',
                        Path('Code/CryEngine/CryCommon'),
                        Path('Code/CryEngine/CryAction'),
                        Path('Code/CryEngine/CryNetwork') ]

    )
```

Create source files

All game projects first need a source file. If you intend to use pre-compiled headers you must create standard `StdAfx.h` and `StdAfx.cpp` files. For this example you create a single C++ file and a corresponding header file (`MyGameMain.cpp` and `MyGameMain.h`).

Create a waf_files configuration file

You use the `waf_files` configuration file to include the source files into the game module. For this example you create a file called `MyGame.waf_files` and specify it for the project. This file includes the four files you created from the previous step.

Create a `waf_files` configuration file called `MyGame.waf_files` with the following:

```
{
    "auto":
    {
        "Source Files":
        [
            "MyGameMain.cpp"
        ],
        "Header Files":
        [
            "MyGameMain.h"
        ]
    }
}
```

```
    ]  
  },  
  "none":  
  {  
    "Root":  
    [  
      "StdAfx.h",  
      "StdAfx.cpp"  
    ]  
  }  
}
```

Updating the User Settings

The final step is to update `enabled_game_projects` to include or exclusively set the new game project. You can do this one of the following ways:

- Hand edit the `user_settings.options` file to set the value for the `enabled_game_projects`. The following example sets `MyGame` as the only game project generated. You can use a comma-separated list to include multiple game projects in the final solution.

```
[Game Projects]  
enabled_game_projects = MyGame
```

- Update game projects using the Lumberyard Waf GUI. Run the `show_options_dialog` command, click **Game Projects** in the Lumberyard Waf window, and select your new project. You can select more than one project.
- Build the project during the build step. Use `--enabled-game-projects=MyGame` to override every build command. This does not include the project in the generated solution, but it sets specific game projects to build during the build commands.

```
lmbr_waf.exe build_win_x64_debug -p game_and_engine --enabled-game-  
projects=MyGame
```

Adding a Spec

The Waf spec system provides a template to create Visual Studio solutions and describes a build filter that determines which modules to build for particular platforms and configurations. The nature of the generic Waf build system is to be all projects that are defined through the wscript system, which acts recursively on the root directory structure. If no spec is specified when you execute a build or clean command, the Waf build system system attempts to build all modules that are supported by the selected target platform and configuration. The platform and configuration support is defined in each of the module's wscript definitions. For more information, see [Adding a Build Module \(p. 1349\)](#).

Project spec files are a collection of modules and definitions for a specific build pipeline. These files are useful for including existing modules or adding new ones as part of the build dependencies for your game project.

When you build in debug or profile configurations and their `_dedicated` counterparts, a spec file is not required. This is because these two configurations build out of Lumberyard as modular shared components. In performance and release configurations, however, all the modules that are marked as `CryEngineModules` are built monolithically, which means that they are built into a single executable. This causes problems with similar modules that support the same platform and configuration. Currently, the spec file is required for this scenario in order to target specific modules to build into the monolithic `.exe` files.

Adding a project requires these steps:

- [Creating a New Project Spec JSON File \(p. 1348\)](#)
- [Adding the Spec File to the Visual Studio Solution Generator \(p. 1349\)](#)
- [Building the Spec \(p. 1349\)](#)

Creating a New Project Spec JSON File

In the following example a spec file called `my_game` includes the game engine modules as a base as well as custom modules for Windows. The spec file also sets a custom `#define` for Windows builds.

You need to configure the values for the modules that you want to include in the spec file (and optionally the target platform and configuration). The spec file can isolate `target_platform` modules for multiplatform builds.

Create a spec file called `my_game.json` with the following:

```
{
  "description"      : "Configuration to build my game",
  "visual_studio_name" : "My Game",
  "comment"         : "This is the build spec for my game",
  "disable_game_projects" : false,
  "platforms"       : ["win_x64"],
  "configurations"  : ["debug", "profile", "performance", "release"],
  "modules" :
  [
    "WindowsModule1",
    "WindowsModule2",
    "CommonModule"
  ]
}
```

The spec files are located in the `_WAF_\specs` directory and have the `.json` file extension. For more information on Waf spec files, see [Waf Spec Files \(*.json\) \(p. 1324\)](#).

Spec	Description
<code>all.json</code>	Configuration for all targets.
<code>dedicated_server.json</code>	Configuration to build dedicated servers for the enabled projects.
<code>external_sdks.json</code>	Configuration to build externally distributed binary-only libraries.
<code>game_and_engine.json</code>	Configuration to build the engine and game projects.
<code>pipeline.json</code>	Configuration to build Pipeline Only for building Resource Compiler, and also Maya, 3ds Max, and Photoshop plugins if Visual Studio 2010 and 2012 are installed. Build only in Profile or Debug mode (Release mode is only for the 3ds Max plugin)
<code>resource_compiler.json</code>	Configuration to build only the Resource Compiler.
<code>shadercachegen.json</code>	Configuration to build only the shadercache generator.
<code>tools.json</code>	Configuration to build nonessential tools.

Adding the Spec File to the Visual Studio Solution Generator

Adding the spec file to the Visual Studio solution is optional.

To add the spec file to the Visual Studio solution

1. Edit the `specs_to_include_in_project_generation` value in the `user_settings.options` file to add your spec file to the Visual Studio solution:

```
[Visual Studio Project Generator]
generate_vs_projects_automatically = True
visual_studio_solution_name = LumberyardSDK
visual_studio_solution_folder = Solutions
specs_to_include_in_project_generation = MySpec1, MySpec2, MySpec3
```

2. Regenerate the Visual Studio solution by running the following command: `lمبر_waf.bat configure`

Building the Spec

After saving the new spec, do one of the following:

- Build the spec using Visual Studio (if you followed the steps above to add the spec to Visual Studio).
- Build the spec from the command line by running the following command: `lمبر_waf.bat build_win_x64_profile -p MySpec`

The `build` command builds the game project specified in the `user_settings`, even if the module is not defined in the spec. The exception is if the option `disable_game_projects` is set to **True**.

Adding a Build Module

You can create a custom build module in the Lumberyard Waf build system. You can use predefined build modules to add any shared library or plugin into the Lumberyard engine SDK.

The default Waf system defines modules and methods that will take various keywords into Waf commands to build applications and shared and static libraries as well as serving as a project container for files. The `cryengine_modules.py` file defines functions that wrap these modules with additional keywords and logic to extend the behavior of standard Waf into a system that supports the requirements of Lumberyard. In addition to providing standard Waf build functionality, the functions in the various modules add support for precompiled headers (`pch`), content file support (`.waf_files`), monolithic build capability, uber file support, and Microsoft Visual Studio (`msvs`) generation.

Creating a module requires the following steps:

1. Create the source folder and script
2. Create a basic wscript module
3. Create the `.waf_files` content file
4. Specify additional include paths and external library linking
5. Add a project dependency

Creating a New Module

You can create and add the following types of modules to the Lumberyard Waf build system:

Build Module	Description	Project Type
CryEngineModule	<p>Modules that are dynamically loaded at runtime as part of the lumberyard engine module system. For Performance and Release configurations, all projects that are built using these modules are included monolithically to the final build output. If the libraries are not linked in, the source from these modules is included in the build.</p> <p>For Debug and Profile configurations, these modules are built as shared libraries. For the Windows platform, versioning information is injected as defined in the <code>waf_branch_spec.py</code> file located in the root folder. As such, a Windows resource (<code>.rc</code>) file as needed as part of the <code>waf_files</code> content.</p>	Shared Library (Non-Release), Static Library Performance (Performance, Release)
CryEngineSharedLibrary	<p>Used to define a shared library that any other module can use inside Waf. Provided they are located in the same directory path as the dependent project, these modules are included as a dependency to other modules by use of the <code>use</code> keyword.</p>	Shared Library
CryEngineStaticLibrary	<p>Used to define a static library that can be used by any other module inside Waf. Provided they are located in the same folder path as the dependent project, these modules are included as a dependency to other modules by use of the <code>use</code> keyword.</p>	Static Library
CryLauncher	<p>Used to define the build definition for launchers, which are created for each game project defined per supported platform. All supported launchers that can be generated based on availability against the current platform are located in the <code>\Code\Launcher</code> subfolder. If an additional platform is included, a new launcher</p>	Executable

Build Module	Description	Project Type
	project would be added in this subfolder and use the CryLauncher build module.	
CryDedicatedServer	Similar to the CryLauncher module, except used for dedicated server projects.	Executable
CryConsoleApplication	Used to build console applications. On the Windows platform, it builds a console application instead of a Windows application.	Executable
CryBuildUtility	Used to define build utility projects, such as AZCodeGenerator. Build utilities are separated into a <code>build_utilities</code> group that are built before the regular build group.	Executable
CryFileContainer	Used to set a file container for projects.	None
CryEditor	Used by Lumberyard Editor projects.	Executable
LumberyardApp		
CryEditorUiQt	Used by the <code>CryEditorUI_QT</code> plugin.	
CryPlugin	Used by Lumberyard Editor plugin projects. It is automatically placed in the <code>\EditorPlugins</code> subfolder and automatically loaded by Lumberyard Editor at runtime.	Shared Library
CryStandAlonePlugin	Used by Lumberyard Editor plugin projects. The difference between this module and <code>CryPlugin</code> is that it does not import any <code>SANDBOX</code> or <code>EDITOR_COMMON</code> imports, <code>RTTI</code> is enabled, and <code>nodefaultlib:/</code> is set to <code>libcmt</code> .	Shared Library
CryPluginModule	Used to define shared libraries that can be used by a Lumberyard Editor plugin. Plugins that need to link to a Cryengine plugin module use the <code>use</code> feature of Waf.	Shared Library

Build Module	Description	Project Type
CryResourceCompiler	Used by the Resource Compiler to implicitly set the target name to <code>rc</code> and the subfolder to <code>\rc</code> under the <code>\configure</code> output folder.	Executable
CryResourceCompiler	Used by the Resource Compiler to implicitly set the target name to <code>rc</code> and the subfolder to <code>rc</code> under the <code>\configure</code> output folder.	Shared Library
CryPipelineModule	Used to define pipeline modules such as for the 3ds Max and Maya exporters.	Custom
CryQtApplication	Used to define Qt 5 applications that can be launched by Lumberyard Editor, such as the Asset Processor.	Executable
CryQtConsoleApplication	Used to define Qt 5 console applications that can be launched by Lumberyard Editor, such as the Asset Processor batch file.	Executable

In this topic's example you create a `CryEngineModule`.

Build Module Keywords

The following describes the general keywords that are supported by the build modules. The listed targetable keywords can be specific to a platform or a configuration. The keyword by itself is used for all supported platforms and configurations, but if you need keywords that are specific to a platform or configuration, you must include the name of the platform or configuration in the name..

Other things to consider:

- The general pattern for platform plus configuration-specific values is `<platform>_<configuration>_<keyword>`.
- The general pattern for platform-specific values is `<platform>_<keyword>`
- The general pattern for configuration-specific values is `<configuration>_<keyword>`

You can use the following keyword macros to reduce the verbosity of `wscript` files:

autod_uselib

This macro is used in conjunction with the `uselib` keyword and adds a `D` to the suffix of all of the `uselib` names in the list. This eliminates the need to duplicate the same debug versions of the lib for every configuration. This only works for `uselib` modules that use a trailing `D` suffix to distinguish between debug and nondebug version.

<platform>_ndebug_<keyword>

This macro eliminates the need to repeatedly specify certain nondebug flags. Lumberyard has one debug configuration and three nondebug configurations.

Keyword	Description	Targetable?
target	Name of the target project.	
platforms	The list of platforms to restrict this module to. If not specified, then defaults to <code>all</code> , which assumes all supported target platforms on the current host.	No
configurations	<p>The list of configurations to restrict this module to. If not specified, then default to <code>all</code>.</p> <p>In addition to the standard configurations (<code>debug</code>, <code>profile</code>, and <code>release</code>), configurations can be specific to a particular platform. This is done by appending the platform name with a colon separator in front of the configuration. For example, if a module supports only <code>debug</code> and <code>profile</code> for the <code>iOS</code> platform, then the configuration list would include the values <code>ios:debug</code> and <code>ios:profile</code>.</p>	
file_list	The <code>.waf_files</code> JSON file that contains the file list definition for the project.	Yes
pch	The name of the precompiled header. If present, then precompiled headers are enabled.	
use	Additional projects to link as a use dependency.	Yes
uselib	Additional libraries to use.	Yes
defines	Additional preprocessor defines for the project.	Yes
includes	Additional include paths.	Yes
cflags	Additional C flags.	Yes
cxxflags	Additional C++ flags.	Yes
lib	Additional libraries to link to.	Yes
libpath	Additional library include path.	Yes
stlib	Boolean flag that indicates a static library module.	Yes

Keyword	Description	Targetable?
stlibpath	Lib path for static libs (generally the same for any lib).	Yes
linkflags	Additional link flags during the linker phase.	Yes
export_definition	Export definition filename (.def file).	Yes
features	Any additional features to tag this project to.	Yes
output_file_name	An output file name used to override the default output file based on the target.	Yes
framework	Additional frameworks (Darwin).	No
frameworkpath	Additional framework paths (darwin).	No
export_defines	Additional preprocessor defines that are added to any module that uses the current module as a project dependency.	No
export_includes	Additional library include paths that are added to any module that uses the current module as a project dependency.	No
additional_settings	Additional settings added for specific files.	Yes
meta_includes	Meta includes for WinRT.	yes
files	Another way to pass in files for processing a build project.	Yes
winres_includes	Additional include paths for the winres compiler.	No
winres_defines	Additional defines for the winres compiler.	No
enable_rtti	Flag to enable rtti settings for a project.	Yes
rpath	Additional relative library paths (Darwin).	No

Creating a Basic Wscript Module

The wscript file specifies the name of the module (`target`), `.waf_files` content file (`file_list`), Visual Studio filter (`vs_filter`), and precompiled headers (`pch`).

Create a wscript module with the following:

```
def build(bld):  
  
    bld.CryEngineModule(  
        target      = 'MyEngineModule',  
        vs_filter   = 'LyEngine',  
        file_list   = 'myenginemodule.waf_files',  
        pch         = 'StdAfx.cpp'  
    )
```

In order for the Lumberyard Waf build system to pick up the new folder and script, you must add the new folder to the list of subfolders to recurse. Because you are adding this project under `root_folder/Code/CryEngine/MyEngineModule`, you need to update the wscript located in the parent `root_folder/Code/CryEngine` folder.

Update the wscript located in the `root_folder/Code/CryEngine` folder with the following:

```
SUBFOLDERS = [  
    'CryInput',  
    'Cry3DEngine',  
    ...  
    'MyEngineModule',  
]  
  
def build(bld):  
    # Recursive into all sub projects  
    bld.recurse(SUBFOLDERS)
```

Creating the .waf_files Content File

In the example wscript, you specified a file called `myenginemodule.waf_files` as the project content file. The project content file can be one of the following:

- A single file that defines the source files for the project
- A list of files that define the source files for the project
- Platform/configuration, where certain files are included only for a particular platform (for example, console-specific files)

The following `myenginemodule.waf_files` example demonstrates a simple module with six files:

```
{  
    "NoUberFile":  
    {  
        "Root":  
        [  
            "StdAfx.cpp",  
            "StdAfx.h"  
        ]  
    },  
    "myenginemodule_uber_0.cpp":  
    {  
        "Root":  
        [  
            "myenginecore.cpp",  
            "myenginecore.h",  
            "myengineextras.cpp",  
        ]  
    }  
}
```

```
        "myengineextras.h"  
    ]  
}  
}
```

Specifying Additional Include Paths and External Library Linking

To configure the module to link to external modules, you need to update the wscript to specify the include path and link related project settings flags such as includes, lib, libpath, and linkflags.

In this example, you add the following to your module:

1. Google mock libraries for Win x64
2. Preprocessor DEFINE called **USE_GMOCK** to inject into the compile based on the platform Win x64
3. Link-time code generation flag to enable instrumentation (/LTCG:PGOPTIMIZE)

Add the following to your wscript module:

```
def build(bld):  
  
    bld.CryEngineModule(  
        target          = 'MyEngineModule',  
        vs_filter       = 'LyEngine',  
        file_list       = 'myenginemodule.waf_files',  
        pch              = 'StdAfx.cpp',  
  
        win_includes    = [Path('Code/SDKs/GoogleMock/include')],  
        win_lib         = ['gmock'],  
        win_linkflags   = ['/LTCG:PGOPTIMIZE'],  
        win_defines     = ['USE_GMOCK'],  
        win_x64_debug_libpath = [Path('Code/SDKs/GoogleMock/bin/x64/  
Debug')],  
        win_x64_profile_libpath = [Path('Code/SDKs/GoogleMock/bin/x64/  
Release')],  
        win_x64_performance_libpath = [Path('Code/SDKs/GoogleMock/bin/x64/  
Release')],  
        win_x64_release_libpath = [Path('Code/SDKs/GoogleMock/bin/x64/  
Release')]  
    )  
)
```

Note

The following are duplicated to cover all possible configurations that you specified in the waf_branch_spec: win_x64_profile_libpath, win_x64_profile_performance, and win_x64_release_libpath.

Adding and Linking to a Project Dependency

If you want to link to another module that is built within the system, you can use the `use` parameter for the build.

Update your wscript module to link to the CryPerforce module:

```
def build(bld):
```

```
bld.CryEngineModule(  
    target          = 'MyEngineModule',  
    vs_filter       = 'LyEngine',  
    file_list       = 'myenginemodule.waf_files',  
    pch              = 'StdAfx.cpp',  
    use              = ['CryPerforce'],  
  
    win_includes    = [Path('Code/SDKs/GoogleMock/include')],  
    win_lib          = ['gmock'],  
    win_defines     = ['USE_GMOCK'],  
    win_x64_debug_libpath = [Path('Code/SDKs/GoogleMock/bin/x64/  
Debug')],  
    win_x64_profile_libpath = [Path('Code/SDKs/GoogleMock/bin/x64/  
Release')],  
    win_x64_performance_libpath = [Path('Code/SDKs/GoogleMock/bin/x64/  
Release')],  
    win_x64_release_libpath = [Path('Code/SDKs/GoogleMock/bin/x64/  
Release')]  
    )  
)
```

Adding User Settings to Waf

You can add a new user setting to the `default_settings.json` file in the `waf` folder located at the root. Use the standards established in this file and customize as needed. After you have added a user setting, you need to add a minimum of three utility functions for the GUI and console to validate your new setting.

To define utility functions, add the following to `default_settings.py`:

- **Getter** – Retrieves the value of your new setting and performs necessary transformations
- **Validator** (optional) – Validates new values
- **Hint** (optional) – Tells GUI the available options

See the sections below for more information about these functions.

You can also add these functions to any new `.py` file as long as you add the module during build and configure. Be sure to load the file using the following command:

```
(opt.load('<YOUR PYTHON NAME>', tooldir='<DIRECTORY WHERE ITS STORED>')
```

Getter Function

Waf calls the getter function to retrieve the value of your new setting and perform any necessary transformations.

Follow these guidelines:

- Implement the `@register_attribute_callback` function for your type.
- Use the same name for your function as your property name that's defined in the `default_settings` file. For example, if your property name is called `my_setting`, the function must be called `my_setting()`.
- Choose attribute names that are unlikely to conflict.

In the example below of a getter/setter function, the current value is the input and the return value is the value with any validation and transformations applied. We expect a list of comma-separated values. The first half of the function returns the value quickly and the second half is where Waf runs in interactive or GUI mode.

```
@register_attribute_callback
def enabled_game_projects(ctx, section_name, option_name, value):
    """ Configure all Game Projects enabled by user """
    if ctx.options.execsolution or not
    ctx.is_option_true('ask_for_user_input'):
        return value

    if LOADED_OPTIONS.get('enabled_game_projects', 'False') == 'False':
        return ''

    info_str = ['Specify which game projects to include when compiling and
    generating project files.']
    info_str.append('Comma separated list of Game names, from the
    project.json root (SamplesProject, MultiplayerProject) for example')
    # GUI
    if not ctx.is_option_true('console_mode'):
        return ctx.gui_get_attribute(section_name, option_name, value,
        '\n'.join(info_str))
    # Console
    info_str.append("\nQuick option(s) (separate by comma):")
    project_list = ctx.game_projects()
    project_list.sort()
    for idx, project in enumerate(project_list):
        output = ' %s: %s: ' % (idx, project)
        while len(output) < 25:
            output += ' '
        output += ctx.get_launcher_product_name(project)
        info_str.append(output)
    info_str.append("(Press ENTER to keep the current default value shown in
    [])")
    Logs.info('\n'.join(info_str))
    while True:
        projects = _get_string_value(ctx, 'Comma separated project list',
        value)
        projects_input_list = projects.replace(' ', '').split(',')
        # Replace quick options
        options_valid = True
        for proj_idx, proj_name in enumerate(projects_input_list):
            if proj_name.isdigit():
                option_idx = int(proj_name)
                try:
                    projects_input_list[proj_idx] = project_list[option_idx]
                except:
                    Logs.warn('[WARNING] - Invalid option: "%s" ' %
                    option_idx)
                    options_valid = False
        if not options_valid:
            continue
        projects_enabled = ','.join(projects_input_list)
        (res, warning, error) =
        ATTRIBUTE_VERIFICATION_CALLBACKS['verify_enabled_game_projects'](ctx,
        option_name, projects_enabled)
        if error:
```



```
    Logs.warn(error)
    continue
    return projects_enabled
```

In the example below, the function is simpler because it's a simple string entry and there are no enumerations like bool and no validation.

```
@register_attribute_callback
def out_folder_linux64(ctx, section_name, option_name, value):
    """ Configure output folder for linux x64 """
    if not _is_user_input_allowed(ctx, option_name, value):
        Logs.info('\nUser Input disabled.\nUsing default value "%s" for
option: "%s"' % (value, option_name))
        return value

    # GUI / console mode
    if not ctx.is_option_true('console_mode'):
        return ctx.gui_get_attribute(section_name, option_name, value)

    return _get_string_value(ctx, 'Linux x64 Output Folder', value)
```

Validator Function

Waf only requires the getter function; however, to validate input or provide the GUI with more than raw strings, you'll need to implement other functions like the validator.

Follow these guidelines:

- Implement the `@register_verify_attribute_callback` function and name it **`verify_(your_option_name)`**.
- Pass into the function the value parameter, which is the current raw value.
- Return a tuple of Bool, String, ErrorString. The first bool specifies whether or not validation is okay.

In the example below of a validator function, we make sure not to trigger the duplicate check (for example with a list like "SamplesProject,SamplesProject,SamplesProject") or provide a list that won't be accepted (for example with a list like "ASDJASUIDIASJDA").

```
#####
@register_verify_attribute_callback
def verify_enabled_game_projects(ctx, option_name, value):
    """ Configure all Game Projects which should be included in Visual Studio
    """
    if not value:
        return True, "", "" # its okay to have no game project
    if (len(value) == 0):
        return True, "", ""
    if (value[0] == '|' and len(value) == 1):
        return True, "", ""
    project_list = ctx.game_projects()
    project_list.sort()
    project_input_list = value.strip().replace(' ', '').split(',')
    # Get number of occurrences per item in list
    num_of_occurrences = Counter(project_input_list)
    for input in project_input_list:
        # Ensure spec is valid
```

```
if not input in project_list:
    error = ' [ERROR] Unkown game project: "%s".' % input
    return (False, "", error)
# Ensure each spec only exists once in list
elif not num_of_occurrences[input] == 1:
    error = ' [ERROR] Multiple occurrences of "%s" in final game
project value: "%s"' % (input, value)
    return (False, "", error)
return True, "",
```

Hinter Function

Waf uses the optional hinter function to provide the GUI with a list of available options. For example, you might want to use the hinter function if you have a string list that can have multiple or single values that must be specific (enums).

Follow these guidelines:

- Implement the `@register_hint_attribute_callback` function and name it `hint_(your_option_name)`.
- Ignore the value parameter passed, which is the current value.
- Return a tuple of display value list, actual value list, help text list, multi or single. All three input lists should be the same length. The values in these lists are what's displayed in the GUI, the values to set if selected, and the text to display as extra information for an option, respectively.

The example below is for a hinter function.

```
#####
@register_hint_attribute_callback
def hint_enabled_game_projects(ctx, section_name, option_name, value):
    """ Hint list of specs for projection generation """
    project_list = ctx.game_projects()
    project_list.sort()
    desc_list = []
    for gameproj in project_list:
        desc_list.append(ctx.get_launcher_product_name(gameproj))
    return (project_list, project_list, desc_list, "multi")
```

You can also see how Waf uses hinting by engaging Waf in GUI mode and entering the following command: `lmb_r_waf.bat show_option_dialog`

This displays an options dialog box that you can review to determine hinting.

Adding Qt 5 Content to Waf

You can add Qt 5 content into the Waf build system. Typically you use an IDE (integrated development environment) tool such as Qt Designer to create and edit the Qt source file. As with all files that are processed through the Waf build system, the Qt source file must be included in the corresponding `*.waf_files` file for each project.

Intermediate files that need additional compilation such as the `.rcc` file from the `.qrc` compiler do not need to be specified explicitly in these files or any other source file. In addition, intermediate `.rcc` files are never included in any uber files (if the uber file option is enabled) since they are not compatible with uber files in general.

MOC (Meta-Object Compiler) Files

When header files need to be processed by the Meta-Object Compiler (MOC) as part of the build process, the build system identifies them by including their MOC output file inside the source `.cpp` file. For example, if `foo.h` is a file that is to be processed by MOC, then the source `foo.cpp` file also needs to include the corresponding `#include` for the `.moc` file that is generated.

For example:

```
...
#include "foo.h"

..
..

#include <foo.moc>
```

The `#include` for the `.moc` file requires angled brackets because the generated `.moc` file does not reside in the local project directory but rather is located in an intermediate directory. Also, the include path that is added to the project is based on the mirrored project base in the intermediate directory. If the header file exists in a relative subdirectory, that subdirectory needs to be included in the `#include` for the `.moc`, regardless of where the `.cpp` file is located.

For example, if `foo.h` and `foo.cpp` are moved into the `\test` subdirectory, the result looks like the following:

```
...
#include "foo.h" // This can still be relative to the current source file

...
..

#include <test/foo.cpp> // This needs to be relative to the base path for the
project in the intermediate directory.
```

QRC (QT Resource Collection) files

Qt resource collection (`.qrc`) files are processed by the Qt `.qrc` compiler. The output file has the same source name but with an `.rcc` extension. The resulting `.rcc` file is stored in the projects intermediate directory relative to any subdirectory that it exists in.

For example, if the file `foo.qrc` is located in the `\test` subdirectory, the generated `.rcc` file is stored in the `\test` subdirectory under the project's intermediate directory structure. There is no need to explicitly include the generated `.rcc` file into any source file as it is added as a build task for the project.

UI Files

Designer UI files are processed by the Qt UIC (user interface compiler). The output file has an `.h` header extension to it, and `ui_` is also added to the name of the source. The resulting header file is created in the project's intermediate directory relative to its location in the project.

For example, if the file `foo.ui` is located in a `\test` subfolder, the generated `ui_test.h` file will be located in the `\test` subfolder under the project's intermediate folder structure.

When including the generated header file, using the same rule as the moc include applies as follows:

```
...
#include "foo.h"
...
#include <test/ui_foo.h> // Path is relative to the project root
```

Qt Linguist (TS) files

Qt Linguist files (.ts) are processed by Qt and output as .qm files. The .qm files are automatically included into a single .qrc file specified by the langname attribute in the wscript file. The .qrc file is automatically added as a build task like other .qrc files for the project.

The following example demonstrates adding the required langname attribute to a wscript file:

```
...
def build(bld):
    bld.CryPlugin(
        ...
        langname = 'en-us',
        ...
```

The .qm files are loaded using the QTranslator module, and the Qt resource directory is the same relative to the source directory. For example, if there a foo_en-us.ts file in a \test subdirectory, then that is the same directory that you use when loading the resource, as shown in the following example:

```
...
#include <QTranslator>
...

...
void main() {

...
    QTranslator* translator = new QTranslator();
    translator->load("foo_en-us.qm", ":/test");
...
}
```

Using Uber Files

Uber files combine multiple C and CPP files into a single compilation unit, which is intended to reduce input/output impact on compilation time and help accelerate build time.

The code in uber files must meet the following coding standards:

- No global statics in the global namespace
- No global 'using namespace' declarations

Waf compile jobs include files from the *.waf_files lists. These files have the following format:

```
{
  "<uber_file>": {
    "<source_filter_name>": [
      "file1.h",
      "file1.cpp"
    ]
  }
}
```

Valid values for `uber_file` are:

- **none** – Files in this list are banned from uber files. If you want your module to use precompiled headers, you must include them in this list.
- **auto** – Files in this list are combined into modules that are optimized for compile time by Waf. Files that are automatically combined are sorted by absolute path and then combined until the file size path is reached. The combination must be deterministic given the same input files and file size limit.

File size limits vary depending on the compilation:

- 200K – Suggested for compiling remotely using Incredibuild (**incredibuild_max_cores** = 64, **max_parallel_link** = 4)
- 300K – Default setting and suggested for compiling locally using an SSD
- 400-500K – Suggested for compiling using an HDD

You can specify the file size by updating the **uber_file_size** value in the `user_settings` file or by running the following command: `--uber-file-size`

- **somefilename.cpp** – Files in this list are combined into `somefilename.cpp`. This action is useful when certain files can only be combined together or when you want to combine platform-specific code.

Most `waf_files` lists should include one `none` section with the precompiled header and an `auto` section with everything else.

Configuring Waf

To help obtain the most optimal compile times, use the following:

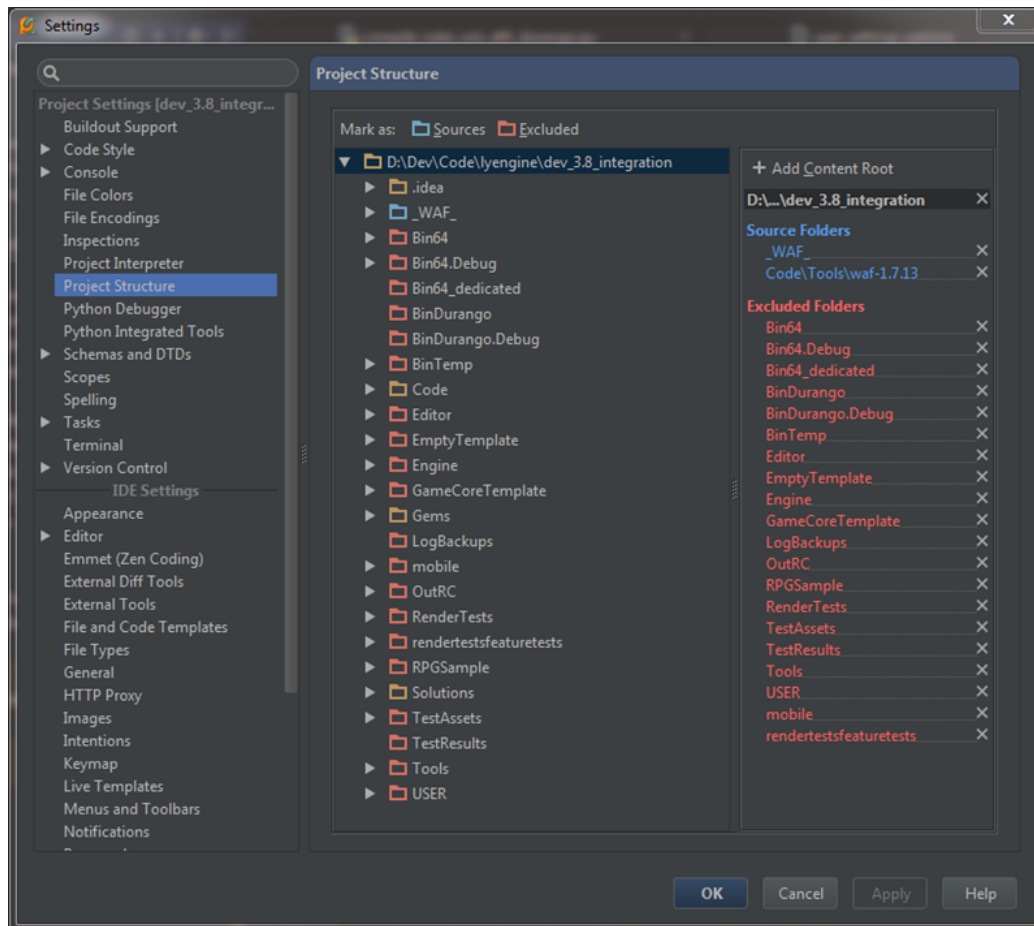
- **use_uber_files** = True
- **max_parallel_link** = 4
- **use_incredibuild** = True
- **use_incredibuild_win** = True
- **incredibuild_max_cores** = 64

Debugging Waf

If you encounter issues that are not related to configuration, it is important to debug the internal Waf library. For a Python callstack, you typically need to debug either in `Code/Tools/waf-#.##.##/waflib` or `Code/Tools/waf-#.##.##/crywaflib`.

Using PyCharm, an IDE for Python development, you can browse to a file where you are having problems, set a breakpoint, and click the bug icon to start debugging. Execution time may be slower when running PyCharm.

Opening the root directory creates file indexing. You can use PyCharm to specify folders to exclude from the project structure, as shown in the example image.



You can also debug the way you would any native Visual Studio solution-based project. Right-click the project you want to debug and select **Set as Startup Project**. Continue the debugging process as you normally would. If you receive a warning saying the `_WAF_` project is outdated but your project is already up-to-date, click **No** to build.

Game Builds

You can create a variety of different game builds, including a release build. Following are definitions for the different build mode types:

Profile mode builds for developers, designers, and artists

- Provides an optimized build meant for development
- Contains performance instrumentation and debugging output
- Can compile shaders and textures
- Communicates with the Asset Processor and compiles as needed
- Has logging, crash reporting, metrics, and other developer features

Debug mode builds for developers

- Provides a nonoptimized version of profile mode meant for debugging
- Has additional memory checks and tests
- Contains obfuscated code that may be hard to follow

Release mode builds for customer previews, demos, and launches

- Can only load from `.pak` files, so assumes these have been created using the Asset Processor and packed from a build script
- Can't compile shaders, so assumes you've already built them
- Can't use VFS or remote asset access
- Doesn't communicate with the Asset Processor as this developer tool doesn't ship with the game
- Strips all logging, instrumentation, profiling, and other measurement metrics
- Strips all developer features, such as console usage, cheat commands, command-line parsing, and batch mode processing
- Combines everything into a single executable file instead of DLLs
- May enable other release features

Topics

- [Compiling Game Code](#) (p. 1366)
- [Creating Release Builds for PC](#) (p. 1366)
- [Creating Minimal Release Builds](#) (p. 1368)
- [Compiling Shaders for Release Builds](#) (p. 1369)
- [Adding Custom Game Icons](#) (p. 1369)
- [Universal Remote Console](#) (p. 1370)

Compiling Game Code

If you choose the **Compile the game code** option in Lumberyard Setup Assistant, you must create a game spec file that includes the configuration to build your game project.

To compile game code

1. In Lumberyard Setup Assistant, select **Compile the game**. Follow the instructions on each page. For more information, see [Running Lumberyard Setup Assistant](#) (p. 15).
2. On the **Summary** page, click **Configure project** to create your game project using Project Configurator. For more information, see [Project Configurator](#) (p. 985).
3. In a command line window, run the following to generate the Visual Studio solution: `lmbr_waf configure`
4. Build your project by doing one of the following:
 - In Visual Studio, select one of the **[Game]** specs from the **Build Configuration** drop-down menu. You can use **[Game] Profile** to start.
 - In a command line window, run the following: `lmbr_waf.bat build_win_x64_profile -p game`

Creating Release Builds for PC

You can create a release build of your game for multiple platforms, including PC, iOS, and Android. This topic describes how to create a PC build; however, other platforms follow a similar process with slight alterations to the batch files.

There are two options for creating a release build:

- Create a standalone image of your game in a directory.

This option allows you to generate a complete image of your game in a directory that can be deployed without requiring the Asset Processor or other files. This image will not contaminate your build or source.

- Create a formal, shippable release build.

This option requires you to use the shader compiler server and shader builder to build a list of shaders from a file. You can then use an automated batch file to pack up the generated shaders directory into a `.pak` file called `shadercache.pak`. Shaders must be packed because release builds cannot load loose shader files. For more information, see [Compiling Shaders for Release Builds](#) (p. 1369).

To create a release build for PC

1. In a command line window, navigate to `\dev` in the directory where you installed Lumberyard.

2. Generate all tools in profile mode by typing: `lmbr_waf build_win_x64_profile -p all`
Alternatively, you can use Visual Studio to build `all` in profile mode.
3. Build `game_and_engine` in release mode by typing: `lmbr_waf build_win_x64_release -p game_and_engine`
Alternatively, you can use Visual Studio to build `game_and_engine` in release mode. This builds the actual release version into `Bin64.Release` and copies all required `.dll` files.
4. Run `BuildSamplesProject_Paks_PC.bat` (located in the `\dev` directory). Alter the path as needed to reference your game if you do not want to include `SamplesProject`. The `.bat` file generates a `\samplesproject_pc_paks` directory that includes all files required to run your game, excluding shaders and executables.
5. (Optional) If you are shipping profile or debug mode executables, edit the `bootstrap.cfg` file (located in the `\samplesproject_pc_paks` directory) to change the `connect_to_remote` value from 1 to 0. This prevents the Asset Processor from starting. The Asset Processor is not required because all necessary assets have been packaged and preprocessed.
6. Build and pack the shaders by doing one of the following:
 - Use a shader compiler server to obtain the shader list from the `shaderlist.txt` file.
 - Generate the shader list by opening Lumberyard Editor and navigating through the levels that you want to ship while in game mode until all the shaders are generated. Then close Lumberyard Editor.
7. Run `BuildShaderPak_DX11.bat` (located in the `\dev` directory). Note the following:
 - If you use Lumberyard Editor to generate the shader list, the batch file looks in `Cache\SamplesProject\PC\user\cache\shaders\shaderlist.txt`.
 - If you use the `shaderlist.txt` file from the shader compiler server, you must specify the path as the first parameter in the batch file. For example, `BuilderShaderPak_DX11 f:\shader_compiler_server\shaderlist.txt`.
 - If the `7za.exe` file is missing from your `dev\Tools` directory, you can download and install the 7-Zip Extra (standalone console version) tool from the [7-Zip](#) website.
8. Copy the resulting `.pak` files (the batch file output will specify where they are located) to the `SamplesProject` directory located in the `samplesproject_pc_paks` directory with the rest of the `.pak` files.
9. Copy the `Bin64.Release` directory to the `samplesproject_pc_paks` directory such that it has its own `Bin64.Release` directory. Optionally rename this directory.

Alternatively, you can create a directory junction (symlink) from `samplesproject_pc_paks` folder\`Bin64.Release` to `Lumberyard_root_folder\Bin64.Release` by typing the following in a command line window: `mklink /j D:\source_folder\samplesproject_pc_paks\Bin64.Release D:\destination_folder\Bin64.Release`
10. You have now created a `samplesproject_pc_paks` directory that contains a standalone release build of your game. This standalone release build does not require the Asset Processor, Lumberyard, or Lumberyard Editor to run.

To run the standalone release build

1. Navigate to the `samplesproject_pc_paks` directory (or the name of your game directory) that contains the standalone release build of your game.
2. Run the game executable.
3. Because the console does not work properly in a release build, you must add `+map (MAPNAME)` to the command line parameters for launching the executable. Add it manually or create a Windows shortcut, batch file, or `autoexec.cfg` file that contains the line, if your game has no menu or other code.

Note

If the build does not run, please check the `dev.log`, `user.log`, `log.log`, or `game.log` files for more information.

4. Release builds require a player login method. Write your own solution or use the provided sample code in the User Login: Default Gem (located in the `\dev\Gems\UserLoginDefault` directory at the root of your Lumberyard installation).

Running a Build from Visual Studio

Before you can run a release build from Visual Studio, you must change the following debugging properties for the project (using `SamplesProject` as an example):

- **Command** to `lumberyard_root_folder\samplesproject_pc_paks\Bin64.release\SamplesProjectLauncher.exe`
- **Command Arguments** to `+map (MAPNAME)`
- **Working Directory** to `lumberyard_root_folder\samplesproject_pc_paks\Bin64.release`

Note

`Bin64.release` is generated for release builds. For profile builds, you must use `Bin64`.

Creating Minimal Release Builds

In the [Creating Release Builds for PC \(p. 1366\)](#) topic, you don't need to copy the entire `\Bin64` directory when creating the release build. However, you do need a `\Bin64` directory (or subfolder containing your binaries) that includes the following:

- Your game executable file
- `D3DCOMPILER_47.DLL`
- `AWS-CPP-SDK-*.DLL`
- (Optional) `DBGHELP.DLL` (without which call stacks are not available if dumping crashes)

You should end up with a directory structure like this:

```
\samplesproject_pc_paks (contains engine bootstrap files)
  \bin64 (contains your executable DLLs)
  \gems
    \gem folders (each containing only gem.json)
  \samplesproject
    (pak files)
  \user
    \cache
      \shaders
        \shader folders
```

Any other folders can be removed.

If you want to package the shaders instead of keeping them loose, zip the `user\cache` directory, rename it `shadercache.pak`, and place it with the rest of the `.pak` files in the `\SamplesProject` directory. This `.zip` file, if correctly formed, would contain a `\Shaders` folder at the root directory. Again, for a release build, you need to use the shader compiler server method.

Using Visual Studio

In order to run the release build from Visual Studio using the above method, you must change some of the debugging properties of the launcher project. Specifically, you must change the following, using SamplesProject as the example:

- Change command to `engine_install_location/samplesproject_pc_paks/Bin64/SamplesProjectLauncher.exe`
- Change Command Arguments to `+map (MAPNAME)`
- Change Working Directory to `engine_install_location/samplesproject_pc_paks/Bin64/`

Compiling Shaders for Release Builds

Shaders for release builds of projects that are built using Lumberyard should be compiled (packaged) into `.pak` files.

Console and mobile platforms – On console and mobile platforms, runtime shader compilation is not supported for release builds. Shaders will compile at runtime only if you are running in profile mode or debug mode and can connect to a shader compiler server.

Windows DirectX platform – On Windows builds that use the DirectX module, runtime shader compilation is supported for release builds. Nevertheless, it is highly recommended that you compile shaders into `.pak` files for performance reasons. Compiling shaders at runtime can cause unwanted frame rate fluctuations. In addition objects that use shaders compiled at runtime may fail to appear until the shaders have been successfully compiled.

The following shader `.pak` files are required for release builds:

- `Shaders.pak` – Only required if you want to support runtime compilation. Source shaders are located in the `dev\Engine\Shaders\` directory.
- `ShaderCache.pak` – Compiled shaders of all possible combinations used by Lumberyard.
- `ShaderCacheStartup.pak` – Compiled shaders that are used during startup.

During development, it is more convenient to use a shader compiler server or to compile shaders locally.

Generating Shader `.pak` Files

To generate shader `.pak` files use the following tools:

- **Shader Compiler** – The shader compiler server generates the `ShaderList.txt` file that contains the list of all shaders used by the game. This server can run locally or on a remote PC. For more information, see [Remote Shader Compiler \(p. 995\)](#).
- **ShaderCacheGen.exe** – Used to populate the local shader cache folder with all the shaders contained in the `ShaderList.txt` file. For more information, see [ShaderCache.pak File Generation \(p. 999\)](#).
- **BuildShaderPak_DX11.bat** – Batch file used to generate the `ShaderCache.pak` files. For more information, see [ShaderCache.pak File Generation \(p. 999\)](#).

Adding Custom Game Icons

You can add a custom icon that appears in the top left title bar window of your game.

To add a custom game icon

1. Create an icon and name it `default_icon`. You can save it in `.tif`, `.png`, `.tga`, or `.bmp` format.
2. Save the icon file to your game's `\textures` directory.

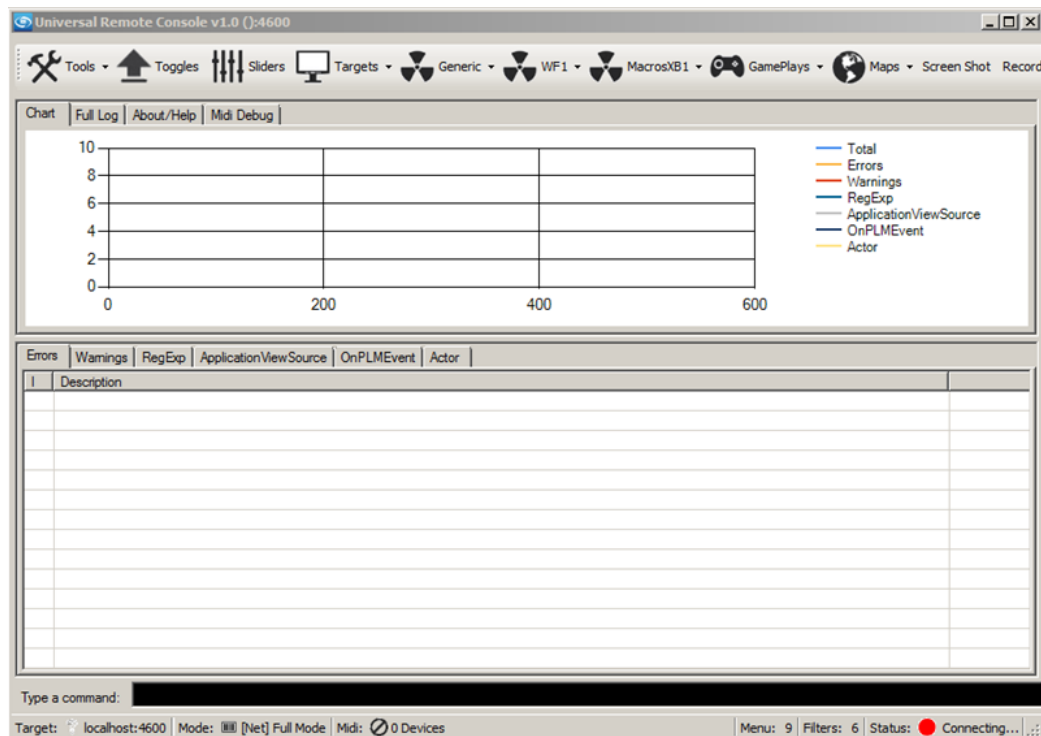
Universal Remote Console

You can use **Console** commands to modify and configure the Lumberyard run-time application. On a PC, the **Console** is available from Lumberyard Editor or the game. But for mobile platforms you must use a separate Windows-based application called the Universal Remote Console. With the Universal Remote Console you can use the IP address of the machine running the Lumberyard game to connect to a remote instance of Lumberyard.

Universal Remote Console requires the use of a PC and works with both Android and iOS. Your mobile device and the PC will need to be on the same network and your firewall should be configured to allow traffic through port 4600.

To start the Universal Remote Console

1. Run `Lumberyard_root_folder\dev\Tools\RemoteConsole\x64\RemoteConsole.exe`



2. To see output from the Lumberyard logging system, click the **Full Log** tab.

To connect to a Lumberyard game on a mobile device

1. Click **Targets** on the toolbar.
2. Type the IP address of the device under **Custom IP**.

If your network allows you to assign fixed IP addresses per device, you can edit the `params.xml` file and add the new target devices, as illustrated in the following example. This file is located in the same directory as Universal Remote Console, and you can edit it with the application running.

```
<Targets>
<Target name="PC" ip="localhost" port="4600"/>
<Target name="Android" ip="192.168.1.247" port="4600"/>
</Targets>
```

This lets you select from a list of devices instead of entering the IP address each time. Once successfully connected, the status indicator in the lower right corner will turn green.

Issuing Commands

In the **Type a command** box at the bottom of the window, type a command like the ones that follow. This control features autocomplete and, for certain commands (like `map`), can also detect available options.

Commands include the following:

- `cl_DisableHUDText` – Disables HUD text
- `g_debug_stats` – Enables gameplay events debugging
- `r_DisplayInfo` – Displays rendering information
- `r_ProfileShaders` – Displays profiling information for the shaders

Glossary

actor	A specialized entity (p. 1373) that is the basis for characters in a game.
additive animation	An animation that can be attached to a base animation to extend its behavior.
agent	An autonomous entity used in artificial intelligence (AI) that uses sensors to observe its environment and directs its activity towards achieving one or more goals.
aim pose	Part of a collection of parametric-blended poses for making a character take aim at specified points in the game.
alpha channel	An extension of RGB color values for specifying the opacity of an object. A value of 0.0 indicates fully transparent while a value of 1.0 indicates fully opaque.
Amazon GameLift	A fully managed AWS (p. 1372) service for deploying, operating, and scaling session-based multiplayer game servers in the cloud.
archetype entity	A special type of entity (p. 1373) with linked instances. If a parameter of the archetype entity is changed, all other instances of that entity parameter are automatically updated.
asset	Any art, texture, 3D model, sound effect, or other digital data that is presented to the user in the game.
attachment	A hierarchical object that is attached to characters, respond to real-world physics, and can be attached, detached, or replaced at runtime in the game. Character attachments include clothing, weapons, tools, or entire body parts such as heads or hands.
AWS	Amazon Web Services, an infrastructure web services platform in the cloud for companies of all sizes. See Also http://aws.amazon.com .
baked	Performs and stores all calculations for a scene element so that the element does not need to be processed or rendered in real time in the game. Often used for lighting or physics. Also referred to as <i>prebaked</i> .
bind pose	The pose that a character has when you bind the mesh (skin) to the skeleton. The skeleton determines the pose.
blend shape	Method that stores a deformed version of a mesh as a series of vertex positions. In each keyframe of an animation, the vertices are interpolated

	between these stored positions. Also known as <i>morph target animation</i> or <i>per-vertex animation</i> .
blend space	Animation blending that is treated as geometry. A character's kinematic, physical, and other high-level motion-related parameters are mapped onto corresponding features that are stored in animation clips. By storing such motion as parameters, controllable interactive animations are possible. Specifically, an animation is associated with a 1D, 2D, or 3D location in the blend space. Also known as a <i>bspace</i> .
boids	Entities that mimic living animals and that have simulated group behavior and obstacle avoidance.
brush	A simple 3D shape that is tied to an entity, and that provides a specific appearance. Brushes are used for static objects.
bspace	See blend space .
bump map	A grayscale image that allows more realistic rendering of an object by introducing small displacements of its surface without changing its geometry. This is done by perturbing the surface normals of a rendered object during lighting. The amount of perturbation is specified by the values in the bump map.
Cloud Canvas	A tool for building connected gameplay by using the Lumberyard flow graph and AWS services, such as Amazon Cognito, Amazon DynamoDB, AWS Lambda, Amazon S3, Amazon SNS, and Amazon SQS.
collision proxy	A simplified geometric shape for approximating a more complex piece of geometry for purposes of a fast first-pass collision detection.
cubemap	A set of six squares that represent reflections from the environment. The six squares form the faces of an imaginary cube that surrounds an object.
cutscene	A noninteractive cinematic game sequence that is typically used to promote plot during gameplay.
damping	The gradual reduction of movement, vibration, or intensity.
DCC	Digital content creation; related to a third-party product such as Autodesk 3ds Max or Autodesk Maya for creating digital assets.
decal	A 2D texture placed on a piece of flat geometry.
detail map	An image for adding up-close surface details to an object.
diffuse map	An image for defining the base color and pattern of an object's surface.
displacement map	A type of heightmap (p. 1374) that modifies the position of vertices of a surface by a specified amount.
DOF	Depth of field. The degree to which distant objects are in focus relative to closer ones.
EBus	A general-purpose system for dispatching messages between objects in C++ code. Also known as <i>event bus</i> .
emitter	An entity that specifies the location from which particles are emitted.
entity	A game object with one or more components that provide some behavior or functionality. An entity consists of a unique ID and a container.

environment probe	A technique that uses cube maps to provide a game level or location with realistic ambient lighting.
Gem	A package that contains code and assets used to provide a single feature or multiple, tightly scoped functions.
gloss map	An image that represents the microscale roughness of a surface. The gloss map is located in the alpha channel of the normal map.
heightmap	A grayscale image used to modify vertex positions of a surface. Lumberyard uses heightmaps to store terrain surface height data. White areas represent the high areas while black areas represent the low areas of the terrain.
HDR tone mapping	The process of converting the tonal values of an image from a high dynamic range (HDR) to a lower range.
helper	Visual icons attached to objects in the Lumberyard Editor that provide object-specific functionality.
IK	Inverse kinematics. The use of kinematics equations to calculate the positions and orientations of joints of a character's skeleton so that a specific part of the skeleton (the end effector) reaches a defined target point.
IBL	Image-based lighting. A rendering technique that involves capturing lighting information, storing it in an environment probe, and projecting it onto a scene.
imposter	Procedurally created 2D sprites that are rendered to look like 3D objects. In essence, imposters are 2.5D objects.
keyframe	An animation frame that specifies exact positions and orientations of geometry affected by the animation. Animation frames that exist between keyframes are interpolated based on animation curves.
level	A world or map that represents the space or area available to the player during the course of completing a discrete game objective. Most games consist of multiple levels.
locomotion locator	The Y vector of the character root joint quaternion, which is typically the direction in which the character is facing. The locomotion locator is needed for motions that translate in nonuniform ways, such as stop or start transitions that have changes in acceleration.
LOD	Level of detail. A technique for increasing performance and reducing draw calls by displaying progressively less-detailed objects the farther they are from the camera.
look pose	Part of a collection of parametric-blended poses for making a character look at specified points in the game.
mesh	A collection of vertices that define the surface of an object.
minimap	A miniature map placed at a screen corner in the game to aid players in orienting themselves in the world.
mipmap	A precalculated, optimized sequence of textures, each of which is a progressively lower resolution representation of the same image. Used in conjunction with LOD (p. 1374) processing.
morph target	A snapshot of vertex locations for a specific mesh that have been deformed in some way.

morph target animation	See blend shape .
navmesh	A navigation mesh, or navmesh, defines the areas of an environment in which a character can move freely without obstructions such as trees, lavas, or other environmental barriers.
normal	The vector that is orthogonal to a surface defined by a set of vertices.
normal map	An image whose pixel values are interpreted as the normal vectors for each point on the surface to which the image is mapped.
null bone	The character bone associated with a null or root object.
parallax mapping	A technique that is used to create detail in a texture adding the illusion of depth. This depth perception changes based on perspective.
PBR	Physically based rendering. PBR uses real-world physical rules and properties to define how light interacts with the surface of objects. Used by the Lumberyard rendering system.
per-vertex animation	See blend shape .
POM	Parallax occlusion mapping. POM uses a displacement map to encode surface detail information in a texture. In this way self-occlusion and self-shadowing of an object is possible without changing the surface geometry.
prebaked	See baked .
prefab	A game object template that stores an asset or a group of assets and all associated properties.
procedural vegetation	A technique used to automatically cover a large area of terrain with vegetation objects using texture layers.
project	The collection of levels, assets, and code that make up a game.
ragdoll	Physical rules used to simulate the realistic movement of a skeletal character.
rigging	The process of building a skeleton hierarchy of bone joints for a character mesh.
rope	Used for attaching cloth, hair, or ropes to a character so that the objects can dangle and move realistically against the character.
retargeting	Applying animations that were created for one model to another.
shadow map	A technique for controlling how shadows are added to a scene. You can use multiple, cascaded shadow maps to control how sun shadows look at varying distances.
skinning	The process of binding bone joints to a model's mesh (skin).
skybox	A cube without the bottom side that contains the environment around a scene. Usually viewed from the inside of the cube.
slices	Cascaded data management system for entities. They are a superset to what are also known as prefabs, and represent the structure in which nearly all entity data is managed.
socket	A pivot point on a character where attachments are connected. Attachments dangle or move according to the properties of the socket.

specular map	An image that determines the shininess of each area of a surface.
SPOM	Silhouette parallax occlusion mapping. SPOM is similar to POM (p. 1375) , but affects the silhouette of a mesh similar to tessellation, without the object actually being tessellated.
sprite	A 2D bitmap image. Multiple sprites can be grouped into a single image known as a sprite sheet.
SSDO	Screen Space Directional Occlusion is a method for approximating real time global illumination (GI).
SSS index	Subsurface scattering index. SSS is used to simulate the diffusion and scattering of light transmitted through translucent objects.
tessellation	The deformation of a surface using one or more geometric objects with no overlaps or gaps. Tessellation increases the geometry count of the mesh by subdividing polygons into smaller polygons before it gets displaced.
texture mapping	The application of an image to a surface.
TOD	The time of day in a level. TOD is used to simulate the changing lighting conditions as the sun crosses the sky.
UV mapping	The projection of texture coordinates onto a 3D surface.
vertex color	A method for adding variety, depth, and color variations to an object surface.
virtual reality	Technology that replicates the gaming environment and simulates a user's presence in it, allowing the player to feel as if they are in the game world as they interact with the environment, characters, and objects.
voxel	A volumetric point in a 3D space, similar to a pixel in a 2D space.
Waf	Game build system that allows you to automatically compile a game that targets all supported platforms.
white point	The reference value used to indicate true white in an image or level.

Lumberyard Blog, Forums, and Feedback

As we continue to improve Lumberyard, we want to thank everyone in our developer community. Without your participation in the forums, your messages, and your bug reports, Lumberyard wouldn't be as strong as it is.

- Keep sending your feedback to <lumberyard-feedback@amazon.com>.
- If you haven't spoken up on the [forums](#) yet, we would love to have you.
- You can also keep up with new changes on our [blog](#) and leave comments to let us know what you think.

Legal

The Amazon Lumberyard engine, integrated development environment, and related assets and tools are licensed as "Lumberyard Materials" under the terms and conditions of the [AWS Customer Agreement](#) and the [Lumberyard Service Terms](#). Please see these terms and conditions for details.

Topics

- [Lumberyard Redistributables \(p. 1378\)](#)
- [Alternate Web Services \(p. 1380\)](#)

Lumberyard Redistributables

For purposes of the [Lumberyard Service Terms](#), the Lumberyard materials in the directories listed below are designated as "Lumberyard Redistributables." Unless subdirectories of a directory are specified, all files in the directory listed are deemed Lumberyard Redistributables.

Note

Restrictions on use and distribution of the Lumberyard materials, including in source code form, are specified in the [Service Terms](#).

Lumberyard

- \3rdParty\GameLift
- \dev_WAF_
- \dev\Bin64
- \dev\Code\CryEngine
- \dev\Code\Framework
- \dev\Code\Launcher
- \dev\Code\MultiplayerProject
- \dev\Code\SamplesProject
- \dev\Code\Sandbox
- \dev\Code\Tools
- \dev\Code\Tools\AssetTagging
- \dev\Code\Tools\ClangReflect

- \dev\Code\Tools\CryCommonTools
- \dev\Code\Tools\CryD3DCompilerStub
- \dev\Code\Tools\CrySCompilerServer
- \dev\Code\Tools\CryXML
- \dev\Code\Tools\DBAPI
- \dev\Code\Tools\GemRegistry
- \dev\Code\Tools\HLSLCrossCompiler
- \dev\Code\Tools\LUARemoteDebugger
- \dev\Code\Tools\PRT
- \dev\Code\Tools\RC
- \dev\Code\Tools\ShaderCacheGen
- \dev\Code\Tools\SphericalHarmonics
- \dev\Code\Tools\AssetProcessor
- \dev\Code\Tools\waf-1.7.13
- \dev\Editor
- \dev\Engine
- \dev\FeatureTests
- \dev\Gems
- \dev\MultiplayerProject
- \dev\ProjectTemplates
- \dev\SamplesProject
- \dev\AssetProcessorPlatformConfig.ini
- \dev\bootstrap.cfg
- \dev\editor.cfg
- \dev\engineeroot.txt
- \dev\libr_aws.cmd
- \dev\libr_waf.bat
- \dev\libr_waf.exe
- \dev\SetupAssistantConfig.json
- \dev\system_BuildShaderPak_DX11.cfg
- \dev\system_BuildShaderPak_GL4.cfg
- \dev\system_windows_pc.cfg
- \dev\waf_branch_spec.py
- \dev\wscript

Asset Collection – Woodland

- All directories

Asset Collection – Beach City

- All directories

Legacy Sample (GameSDK)

- All directories

Alternate Web Services

For purposes of the [Lumberyard Service Terms](#), "Alternate Web Service" means any non-AWS compute, database, storage, or container service that is similar to or can act as a replacement for the following services: Amazon EC2, Amazon Lambda, Amazon DynamoDB, Amazon RDS, Amazon S3, Amazon EBS, Amazon EC2 Container Service, or Amazon GameLift.