Lumberyard

User Guide

Version 1.11

Lumberyard: User Guide

Table of Contents

What Is Lumberyard?

Amazon Lumberyard is a free, cross-platform, 3D game engine that allows you to create high-quality games, connect your games to the compute and storage of the AWS Cloud, and engage fans on Twitch. With Lumberyard, you can spend more time creating great gameplay and building communities of fans, and less time on the heavy lifting of building a game engine and managing servers.

Lumberyard offers everything a professional game developer can expect, such as a full-featured editor, native code performance and stunning visuals. It also includes hundreds of other ready-to-use features like networking, character and animation editors, the **Particle Editor**, the **UI Editor**, audio tools, and more.

Image from Starter Game. For more sample projects and levels, see Sample Projects and Levels (p. 1465).

Professional-Grade AAA Engine

Lumberyard helps you build rich, engaging, world-class games through its comprehensive and proven toolset and highly optimized runtime performance.

Beautiful Worlds

The visuals technology of Lumberyard can bring to life any virtual environment. Your artists get a powerful toolbox to create high quality visual elements, such as physically based shaders, dynamic global illumination, particle effects, lifelike vegetation, real-time dynamic water caustics, and volumetric fog. They can also produce cinematic features like color grading, motion blur, depth of field, and integrated HDR lens flares.

For more information, see the following topics:

- Levels and Environment (p. 1154)

- Particle Effects System (p. 1272)
- Rendering and Graphics (p. 1350)

Asset Changes

With the Lumberyard's Asset Processor, you can quickly get assets into the engine. Save a file (for example, from Maya or Photoshop) into a directory, and the Asset Processor automatically processes that file from source art into game-ready assets. If you edit an asset, Lumberyard detects the change and automatically updates it in the background.

For more information, see Using the Asset Processor (p. 163).

Component Entities

The component entity system helps you compose complex entities out of simpler entities. Content creators can drag and drop components to build behaviors, edit component settings live in the editor, and create Lua scripts to quickly change or extend an entity's behavior. To build and iterate for complex scenes, entities can be grouped together into slices. Those slices can in turn be used to create more complex slices, resulting in a fully cascading prefab system. If you change a slice, you can share your changes to all the slice instances, or keep your changes exclusive to just that instance.

For more information, see the following topics:

- Component Entity System (p. 437)
- Working with Slices (p. 597)

Compelling Characters

Artists can use Lumberyard to create believable characters and high-fidelity performances. Lumberyard's character tool, Geppetto, combines animation, attachments, and physics simulations with blend shape, blend space, and animation layering. Animators can use Lumberyard's animation tool, Mannequin, to bring characters and creatures to life with features that include animation sequencing, transitions, game logic procedures, ragdoll physics, and more.

For more information, see Characters and Animation (p. 203).

Robust Networking

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

For more information, see Networking System in the *Amazon Lumberyard Developer Guide*.

Real-Time Gameplay Editing

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

Modular Gems

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

For more information, see Gems (p. 1060).

Wwise LTX

Lumberyard includes a version of Audiokinetic's advanced, feature-rich sound engine. Sound designers and composers can author rich soundscapes for your games.

For more information, see Audio System (p. 181).

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

Amazon GameLift is an AWS service for deploying, operating, and scaling session-based multiplayer games. You can scale high-performance game servers up and down to meet player demand without additional engineering effort.

For more information, see the Amazon GameLift Developer Guide.

Cloud Canvas

You can build live, online game features, such as a community news feed, daily gifts, or in-game notifications, in minutes with 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.

For more information, see Cloud Canvas in the *Amazon Lumberyard Developer Guide*.

AWS SDK for C++

The AWS SDK for C++ provides C++ API operations 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 integrate AWS components into your game.

For more information, see the 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.

For more information, see Twitch ChatPlay System (p. 1529).

Twitch JoinIn

The Twitch JoinIn feature within Lumberyard helps you build multiplayer games that allow Twitch broadcasters to 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.

For more information, see Twitch JoinIn (p. 1537).

Free with Source

Lumberyard is free, including source code. You can customize Lumberyard for your team and vision for your project today, and for future projects. There are no seat fees, subscription fees, or requirements to share revenue. Only pay for the AWS services that you choose to use.

For more information, see the Lumberyard Licensing FAQ.

Lumberyard Systems

Lumberyard consists of the following major systems that help you develop levels within the editor:

- AI System (p. 106)

  Artificial Intelligence (AI) is a technology or system that endows seemingly intelligent actions and behaviors to an agent or character, called the AI agent. An AI agent is a game entity that uses information to make decisions in pursuit of one or more goals.

- Audio System (p. 181)

  The audio translation layer (ATL) provides an interface between Lumberyard and third-party audio middleware so you can change your audio implementation without affecting the game logic.

- Characters and Animation (p. 203)

  The character animation system combines skeletal-based deformation of meshes with morph-based vertex deformation to facilitate complex animation. You can create realistic character movements 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.

- Cinematics System (p. 392)

  Cinematics are interactive movie animations. 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 players can view these from their own (first person) or another's (third person) perspective.

- Component Entity System (p. 437)

  The component entity system provides a modular and intuitive construction of game elements, such as lights, cameras, trigger areas, and objects. The component entity system works at both the system level and the entity level. It employs reflection, serialization, event bus (EBus) messaging, fully cascading slices, and the ability to drag and drop and edit entities and their components in Lumberyard Editor.

- Object and Entity System (p. 613)

  The object and entity system is now considered legacy; it will be fully replaced by the component entity system at a future date.

- Flow Graph System (p. 754)

  Flow graph is a visual scripting system with which you can implement complex game logic without touching code. An extensive library of nodes provides everything that you need to fully control entities and AI agents in a level. Flow graph can prototype gameplay, effects, and sound design. A level can contain multiple flow graphs that perform different tasks at the same time.

- Gems (p. 1060)

  Gems are packages that contain code and/or assets to augment your game projects, such as the ChatPlay and Woodland Asset Collection gems. You can select gems to include in your project through the Lumberyard Project Configurator (p. 28) and from the command line.

- Levels and Environment (p. 1154)

  A level, also known as world or map, represents the space or area available to the player while completing a game objective. A level's environment includes lighting, terrain, bodies of water, vegetation, sky, and weather effects.

- Materials (p. 1350)

  You can use the **Material Editor** to create and apply materials, map textures, set opacity and lighting effects, set shader parameters, create vertex deformations, tessellation, and more.

- Particle Effects System (p. 1272)

  The particle effects system simulates explosions, fire, smoke, sparks, water spray, fog, snow, rain, and other effects.

- Script Canvas (p. 682)

  Create game logic and behaviors with Lumberyard's new visual scripting environment.

- Shader Rendering System (p. 1351)

  Lumberyard's physically based rendering (PBR) shaders 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.

- Twitch ChatPlay System (p. 1529)

  The Twitch ChatPlay Gem 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.

- UI System (p. 1539)

  With the **UI Editor**, you can create and customize various parts of the game user interface, such as images, text, buttons, menus, scroll boxes, and heads-up displays (HUDs).

- Virtual Reality (p. 1782)

  Virtual reality (VR) is a technology that replicates the gaming environment and simulates a user's presence in it. With virtual reality players, feel as if they are in the game world as they interact with the environment, characters, and objects. Lumberyard's virtual reality system integrates the use of the Oculus Rift, HTC Vive, and Open Source Virtual Reality (OSVR) head-mounted displays (HMD) on PC gaming systems.

The following Lumberyard tools are outside of the main editor and are used for project and game development:

- Asset Pipeline (p. 162)

  The Asset Pipeline converts source art and other assets into platform-specific, game ready data.

- Lmbr.exe (p. 1208)

  `Lmbr.exe` is a command-line tool for managing capabilities, game projects, and gems.

- Mobile Support (p. 1211)

  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 5s, iPhone 6s, iPhone 6s Plus, iPad Air 2, and iPad Pro.

- macOS Support (p. 1260)

  You can use Lumberyard to build macOS applications.

- Linux Support (p. 1266)

Lumberyard supports compiling and deploying the Windows client for a multiplayer project on a Linux dedicated server.

- Project Configurator (p. 28)

  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, configure advanced settings, and enable, disable, or create new gems.

- Testing, Profiling, and Debugging (p. 1508)

  Lumberyard includes a number of tools for testing builds, profiling performance, and debugging.

- Waf Build System (p. 1797)

  With the Waf build system, you can switch between various build pipelines and ensure that 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.

Lumberyard Editors and Tools

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

Animation Editor (p. 223)

Manages character animations, attachments, and physics simulations along with blendspace and animation layering.

Asset Browser (p. 179)

Displays all game assets available for use.

Asset Processor (p. 163)

Runs in the background when you launch Lumberyard Editor, monitoring input directories for changes in source files and automatically generating platform-specific game assets as they change.

Audio Controls Editor (p. 186)

Manages audio translation layer (ATL) controls and events for the audio system.

Geppetto (p. 276) (Legacy)

Manages character animations, attachments, and physics simulations along with blendspace and animation layering.

Console (p. 93)

Runs editor commands and lists available console variables.

Database View (p. 1181)

Displays various object libraries such as entities, particles, and prefabs.

Entity Inspector (p. 441)

Displays the ID and name for component entity system objects.

Entity Outliner (p. 437)

Displays all component entities used for a level.

FBX Settings (p. 212)

Imports single static meshes and materials from FBX.

Flow Graph (p. 754) (Legacy)

Implements complex game logic using a visual scripting system.

Layer Editor (p. 1178)

Creates and manages layers for levels.

Lens Flare Editor

Creates and manages camera lens flare effects.

Sun Trajectory Tool (p. 1182)

Creates and manages dynamic sky lighting effects.

LOD Generator

Generates geometry and material level of detail (LOD).

Lumberyard Editor (p. 67)

Acts as the main workspace editor and game viewport; loads the **Rollup Bar** and console by default.

Lumberyard Setup Assistant (p. 20)

Ensures that you have the necessary runtime software and SDKs installed to successfully run Lumberyard.

Mannequin Editor (p. 329)

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 (p. 1350)

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 (p. 177)

Searches for assets that have moved and references their new locations.

Gems System (p. 1060)

Provides a library of prebuilt features that you can use to quickly start new projects or prototype ideas.

Particle Editor (p. 1272)

Creates and simulates explosions, fire, sparks, and other visual effects.

Project Configurator (p. 28)

Standalone application used to tell the Waf build system which gems to include in the game build.

Resource Compiler (p. 163)

Compresses and processes source game asset files and creates package files.

Rollup Bar (p. 88)

Accesses and places objects, vegetation, modified terrain, and modeling tools; includes display options, profile tools, and layer controls.

Script Canvas (p. 682)

Create game logic and behaviors with Lumberyard's new visual scripting environment.

Script Terminal

Runs various scripts in a terminal window.

Smart Objects Editor

Creates and manages smart objects, which can interact with other objects according to complex rules.

Substance Editor

Imports substance `.sbsar` files, edits material properties, and exports them as textures.

Terrain Editor (p. 1155)

Generates terrain and sculpts terrain elements in your level.

Terrain Texture Layers (p. 1161)

Creates and paints terrain texture layers in your level.

Time of Day Editor (p. 1182)

Creates and manages day–night cycles and other dynamic sky effects.

Track View Editor (p. 393)

Creates and manages cinematic scenes and sequences; consists of the **Track Editor** and **Curves Editor**.

UI Editor (p. 1539)

Creates, manages, and simulates user interface elements for your game, such as menus and heads-up displays (HUD).

Universal Remote Console (p. 1852)

Used to connect to a remote instance of Lumberyard running on mobile devices.

Lumberyard Asset File Types

See the following tables for supported asset data file typesin Lumberyard.

Lumberyard supports the `.xml` file format and the following image file formats:

- `.bmp`
- `.jpg`
- `.pgm`

- `.png`
- `.raw`
- `.r16`
- `.tga`
- `.tif`

3D Art Asset File Types

The following file formats are used for static geometry:

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

Material and Texture File Types

The following files are used for the **Material Editor**. For more information, see Materials and Shaders (p. 1350).

File Type	Where Created	Description
`*.mtl` (Material File)	DCC Tool	Contains settings for shaders, surface types, and references to textures.
`*.dds` (DirectDraw Surface)	DCC tool	Contains compressed source texture files.

File Type	Where Created	Description
`*.sbsar` (Substance Files)	Allegorithmic Substance Designer	Contains procedural materials.

Animation File Types

The following file types are used for the **Animation Editor**. For more information, see Animation Editor File Types (p. 233).

File Type	Where Created	Description
`*.actor` (Actor File)	DCC tool	A character with at least one bone.
`*.motion` (Motion File)	DCC tool	Individual animation clips for a character, such as walk, run, and so on.
`*.motionset` (Motion Set File)	Lumberyard	Contains a list of motion files for a character. For example, you can create a motion set named `Run.motionset` that contains the `run.motion`, `sprint.motion`, and `jog.motion` files.
`*.animgraph` (Animation Graph File)	Lumberyard	Contains the state machines, transitions, conditions, blend trees, and so on.
`*.assetinfo` (Asset Info File)	Lumberyard	Contains the configuration and settings for the `.actor` and `.motion` files. **Animation Editor** and the **FBX Settings** tool can create this file.

The following file types are used for the Geppetto and Mannequin systems. For more information on these file types, see Character Animation Files (p. 304).

File Type	Where Created	Description
`*.adb` (Animation Database File)	Lumberyard	The Mannequin system uses this file 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. This file has the same name as the `.chr` file to which it refers to.

File Type	Where Created	Description
`*.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 creates this file, which is 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

The following file types are used for the audio system. For more information, see Audio System (p. 181).

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.

Navigating the Lumberyard Documentation

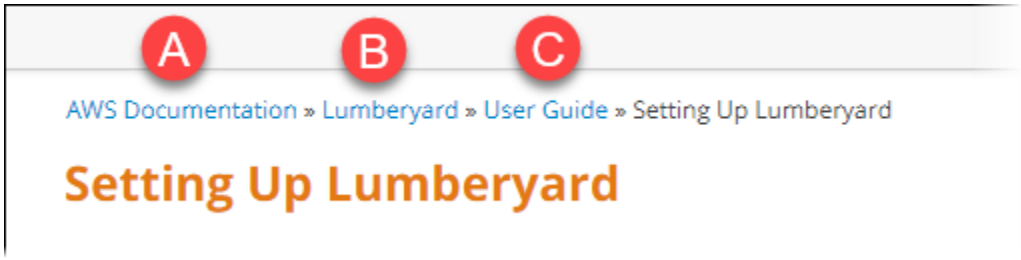
Amazon Lumberyard offers the following guides:

- **Amazon Lumberyard Getting Started Guide** – Get familiar with Lumberyard basics, such as navigating the editor, building terrain, and lighting a scene.
- **Amazon Lumberyard Developer Guide** – If you're a programmer, learn more about working programmatically with Lumberyard.
- **Amazon Lumberyard C++ API Reference** – Learn more about the fundamental C++ API operations of the Lumberyard component entity system.
- **Amazon Lumberyard Release Notes** – Learn more about improvements, highlights, and known issues for Lumberyard features and systems.

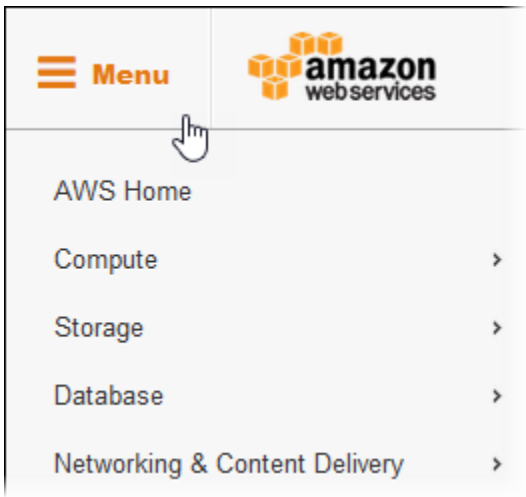
To navigate the Lumberyard documentation

1. To search in a specific guide, go to the guide that you want (for example, the *Amazon Lumberyard Getting Started Guide*) and in the search box, type your keyword.

2. To search all Lumberyard documentation, in the drop-down menu, choose **Documentation – This Product**, and in the search box, type your keyword.
3. In the header, use the path to navigate to other pages:



   A. Choose **AWS Documentation** for all other AWS services.
   B. Choose **Lumberyard** to go to the main Lumberyard documentation page.
   C. Choose the guide name to go back to the home page.
4. You can also use the menu icon to see the documentation for other AWS services.



Setting Up Lumberyard

Lumberyard supports the following platforms: PC, Xbox One, Android, iOS, and macOS. In order to develop games for the Xbox One, you must pass Microsoft's screening process. For information about console support, see Developing Games for Xbox One. For information about developing for mobile devices, see Mobile Support (p. 1211). For information about developing for macOS, see macOS Support (p. 1260).

Topics

System Requirements

Lumberyard requires the following hardware and software:

- One of the following operating systems:
  - Windows 7 64-bit
  - Windows 8.0
  - Windows 10
- 3GHz minimum quad-core processor
- 8 GB RAM minimum
- 2 GB minimum DirectX 11 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
- One or both of the following: (required to compile Lumberyard Editor and tools)
  - Visual Studio 2015 Update 3 or later

    Visual Studio 2015 default installation does not include **Common Tools for Visual C++ 2015** and **Microsoft Foundation Classes (MFC) for C++**. You must select it in the features dialog box during installation, under **Programming Languages**, **Visual C++**. To verify your current installation, click Control Panel, Programs and Features, Microsoft Visual Studio 2015. Next, select **Modify** to view or add features.
  - Visual Studio 2013 Update 4 or later

    Visual Studio 2013 default installation does not include the **Microsoft Foundation Classes (MFC) for C++**. You must select it in the optional features dialog during installation. To verify your current installation, click Control Panel, Programs and Features, Microsoft Visual Studio 2013. Next, select **Modify** to view or add MFC support.

For Visual Studio 2013, you must also install the multibyte character set (MBCS) version of the Microsoft Foundation Class (MFC) Library.

Note
Lumberyard does not support using only Visual C++ Build Tools 2015.

- Visual C++ Redistributable Packages for Visual Studio 2012, 2013, and 2015

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

  - After you have installed Lumberyard, run the redistributable installers from the Visual Studio (2012, 2013, and 2015) directories in the following location:`\dev\Tools\Redistributables\`
  - Download and run the installer directly from Microsoft:
    - Visual C++ Redistributable Packages for Visual Studio 2012
    - Visual C++ Redistributable Packages for Visual Studio 2013
    - Visual C++ Redistributable Packages for Visual Studio 2015

For information about installing third-party software and SDKs, see Using Lumberyard Setup Assistant to Set Up Your Development Environment (p. 20).

Downloading Lumberyard

This topic includes information about downloading Lumberyard using the Lumberyard Installer, and links to more information about installing required third-party software. After installation, you can 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. 14).

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
  - `Bin64vc120` – Binaries directory and configuration files for Visual Studio 2013
  - `Bin64vc140` – Binaries directory and configuration files for Visual Studio 2015
  - `Code` – Source files directory and solution files
  - `Editor` – Editor assets
  - `Engine` – Engine assets
  - `Gems` – Optional systems and assets
  - `MultiplayerSample` – Multiplayer sample project that demonstrates how to build a multiplayer game with the new component entity system
  - `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
- `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 installation 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. An internet connection is required.

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. 16).

> **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.

   > **Note**
   > If you want to install Lumberyard from a location other than the default (C drive), click **Options** and then specify another location.
4. Follow the instructions to complete your installation.
5. On the **Installation Successfully Completed** page, click **Launch Lumberyard Setup Assistant** to install required third-party software and SDK. For information, see Running Lumberyard Setup Assistant (p. 20).

Using GitHub to Download Lumberyard

Each Lumberyard release exists as a separate branch in GitHub. Follow the instructions on the Amazon Lumberyard page on GitHub to download Lumberyard.

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. 15).
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. 19). For information about the Lumberyard directory structure, see Downloading Lumberyard (p. 15).
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. 17).

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. 15).
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. 20).
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 one of the following:
   - If you are using Visual Studio 2015: `lmbr_waf build_win_x64_vs2015_profile -p all`
   - If you are using Visual Studio 2013: `lmbr_waf build_win_x64_vs2013_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 or Visual Studio 2015 runtime. If you do not already have Visual Studio runtime installed, download and install the runtime directly from Microsoft.

Topics

Running Lumberyard Setup Assistant

Before you run Lumberyard Setup Assistant, verify that `3rdParty.txt` appears in the `\3rdParty` directory and that `engineroot.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. Do one of the following:
   - Double-click the **Setup Assistant** desktop shortcut.
   - Click **Setup Assistant** in the Start menu.
   - Open the directory where you extracted Lumberyard and run `SetupAssistant.bat`.
2. In the **Custom Install** box, click **Customize**.
3. Verify that the engine root path is correct.
4. 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 that you have made.
   - **Compile the engine and asset pipeline*** – Compile the engine code and asset pipeline to include any changes that you have made.

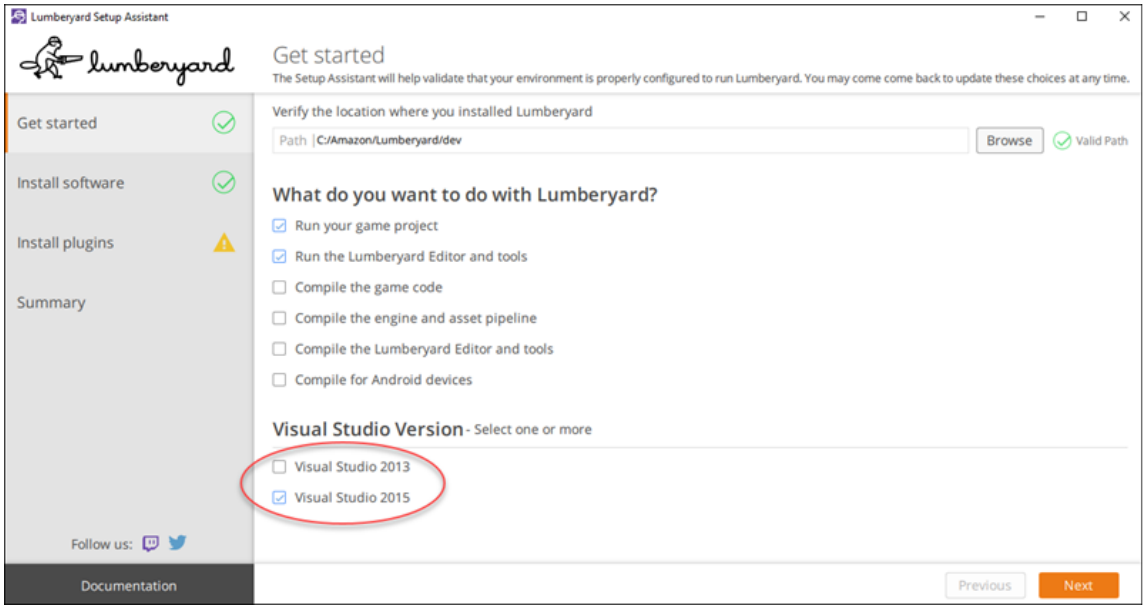
- **Compile the Lumberyard Editor and tools*** – Compile Lumberyard tools to include any changes that you have made.
- **Compile for Android devices***
- **Setup for Linux Dedicated Server***

*If you select any of these 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.

To create new capabilities and enable and disable these capabilities in a command line environment, use Lmbr.exe (p. 1208).

5. Select **Visual Studio 2013**, **Visual Studio 2015**, or both.

   Note
   By default, the Visual Studio 2015 installation does not include C++ as an installed language. In order to build, you must select **C++**, its child options, and **MFC** during the Visual Studio 2015 installation. To verify your current installation, in Windows, open the **Control Panel** and click **Programs and Features**, **Microsoft Visual Studio 2015**. Next, select **Modify** to view or add C++ and MFC support.
   To use both versions of Visual Studio, see Configuring Visual Studio 2013 and 2015 for Lumberyard (p. 21).



6. Click **Next**.
7. Follow the instructions on each page.
8. When you have all the required software and SDKs installed for your implementation, click **Configure project** or **Launch Editor**. For more information about configuring your project, see Using the Project Configurator (p. 28).
9. Log in to your existing Amazon account or create a new account to access the editor.

Configuring Visual Studio 2013 and 2015 for Lumberyard

If you are an advanced user and want to use both Visual Studio 2013 and Visual Studio 2015, you must configure the Lumberyard build system (Waf) for each version. Lumberyard uses Waf to generate Visual Studio solutions and to build.

To configure the Waf settings for each Visual Studio version

1. Navigate to the `\dev\_WAF_` directory at the root of your Lumberyard installation.
2. Use a text editor to open the `user_settings.options` file.
3. Under `[Visual Studio Project Generator]`, set `msvs_version` to **`12`** for Visual Studio 2013.
4. In a command line window, navigate to the `\dev` directory.
5. Type the following command to configure your build:

   ```
   C:Lumberyard\lyengine\dev>lmbr_waf configure
   ```

   This generates the Visual Studio solution and build for the version that you specified in the `user_settings.options` file. The solution is located in the `\dev\Solutions` directory.
6. Repeat these steps for Visual Studio 2015. For step 3, set `msvs_version` to **`14`** for Visual Studio 2015.

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\Tools\LmbrSetup\Win` directory as an executable file called `SetupAssistantBatch.exe`.

You can also use Lmbr.exe (p. 1208) to create new capabilities, enable and disable capabilities, and to modify your projects and gems.

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\Tools\LmbrSetup\Win`
3. Run the `SetupAssistantBatch.exe`.

   Example: `D:\lumberyard-build\dev\Tools\LmbrSetup\Win\SetupAssistantBatch.exe`
4. Modify as needed. See the commands list below.

Commands

Command	Description
`--help`	Lists all commands and descriptions
`--3rdpartypath`	Sets the third-party directory to the specified parameter
`--sdkpath`	Sets the location of the Lumberyard SDK to the specified parameter **Note** This command expects a root where `Lumberyardroot.txt` is located.

Command	Description
`--disablecapability`	Disables the specified tasks (capabilities): • 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.
`--enablecapability`	Enables the specified tasks: • 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
`--all`	Enables all tasks
`--none`	Disables all tasks
`--no-modify-environment`	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 --3rdpartypath "d:\myLumberyard\3rdParty" --sdkpath "d:
\myLumberyard\dev" --none --enablecapability rungame
```

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 `identiferName`, 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, `$TOOLSFOLDER$` 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" :["$TOOLSFOLDER$/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.

Using the Perforce Plugin with Lumberyard

Lumberyard integrates with Perforce as a source control solution. The engine uses the `p4 set` command to configure settings, and the Perforce visual client (P4V) to selectively sync and submit changed assets.

You can use the **Perforce Settings** dialog box to configure how Lumberyard connects to Perforce. The following settings are cached and populated when the dialog box opens:

- **Server** (`P4PORT`)
- **User** (`P4USER`)
- **Workspace** (`P4CLIENT`)
- **Charset** (`P4CHARSET`)

  Note
  `P4_<P4PORT>_CHARSET` is also cached. This value is used if it matches your current `P4PORT` value; otherwise, the value for `P4CHARSET` is used. For example, if `P4PORT` is set to **`my.perforce.server.com:1666`**, the value **`P4_my.perforce.server.com:1666`** would be used.

Certain values may not be modifiable if your Perforce connection settings are configured using a method that overrides the `p4 set` command. The following connection methods may override the ability to modify a setting:

- Config – A configuraton file overrides this connection setting. If detected, the path to the configuration file is displayed. If undetected, you can check the setting for `P4CONFIG`.
- Environment – Your system environment overrides this connection setting. You can check your system's control panel to remove these overrides.

To use the Perforce plugin menu

1. In Lumberyard Editor, click the P4 icon in the bottom toolbar.

   Note
   Hover over the icon to display the connection status.
2. In the drop-down menu, you can do the following:

   - Click **Enable** or **Disable** to toggle the plugin. The **Enable** setting allows you to work online. The **Disable** setting forces you to work offline.

     Note
     Changes are not tracked in offline mode. If you work offline, you must manually reconcile your work when you reconnect to Perforce.
   - Click **Settings** to view or modify your Perforce settings.

   To restore default settings, click **Reset** for each value. When finished, click **OK** to apply your changes.

Uninstalling Amazon Lumberyard

Follow these steps to uninstall Lumberyard.

To uninstall Lumberyard

1. Go to the **Control Panel** and navigate to the list of programs on your computer.
2. Click ***`Lumberyard <version number>`*** and then click **Uninstall**.
3. When prompted, click **Uninstall**. After the uninstallation process, the **Uninstall Successfully Completed** message appears.

   Uninstalling removes Lumberyard components, including the Lumberyard Setup Assistant, Project Configurator, and Lumberyard Editor.

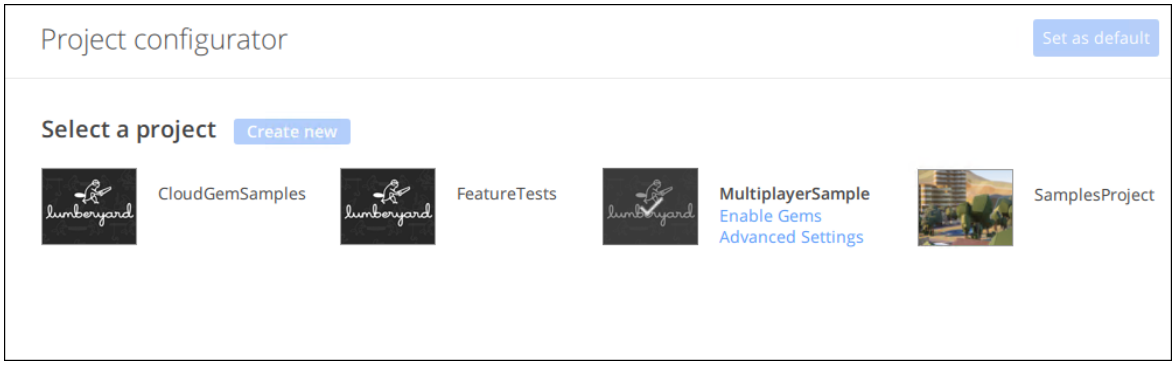
Using the Project Configurator

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 (p. 1060)) to include in a game build. With it, you can create new projects, save active projects, configure advanced settings, and enable, disable, or create new Gems (p. 1060). You can also use Lmbr.exe (p. 1208) to perform these functions in a command line environment.

For information about Waf build system, see Waf Build System (p. 1797). For information about Gems, see Gems (p. 1060).

Note
Before you can run the Project Configurator, you must first run Lumberyard Setup Assistant and close Lumberyard Editor.



Topics

Creating and Launching Game Projects

The Project Configurator is in preview release and is subject to change.

You can create a game project in the Project Configurator or from a command line (`Lmbr.exe` (p. 1208)). When you create a game project, the following are also created:

- A game gem that includes your game-specific code. The gem is named after your game project and is located in the `MyNewProject\Gem` directory.

- A Visual Studio solution file specific to your game.

Creating a Game Project in the Project Configurator

Lumberyard includes **Empty** and **Default** templates that you can use as a starting point when you create your game project in the Project Configurator. We recommend that you use the **Default** template. The **Empty** template is useful to compare with other game project templates to see what features are optional for a base project.

Empty template

The **Empty** template includes the minimum features required for the editor to load and run the game project:

- **CryLegacy** – Enables the editor and launcher to load a game project that contains legacy code.
- **Legacy Game Interface** – Enables the editor and launcher to load a game project that contains legacy game interface code.
- **LyShine** – Provides access to the in-game UI system for Lumberyard.
- **Maestro** – Provides access to cinematics features.

Default template

The **Default** template builds on the **Empty** template by enabling the following gems to provide basic features for game development:

- **Amazon GameLift** – Provides flow graph nodes for using this service and creating game sessions.
- **Camera** – Includes a basic camera component for runtime rendering.
- **ChatPlay** – Includes the interface for triggering events based on Twitch chat activity.
- **Cloud Canvas** – Provides visual scripting capabilities to power your game backend with AWS services.
- **Gestures** – Allows gesture-based input, including click/tap, drag, hold, pinch, rotate, and swipe.
- **Http Requestor** – Adds support to handle HTTP and HTTPs requests.
- **In-App Purchases** – Provides the in-app purchasing API for Android and iOS.
- **Input Management Framework** – Converts input to user-defined gameplay events.
- **LyShine** – Provides access to the in-game UI system for Lumberyard.
- **PBS Reference Materials** – Includes a set of physically based shading reference materials and texture assets.
- **Physics Entities** – Provides physics entity modifiers to simulate physical events.
- **Primitive Assets Gem** – Provides primitive objects to manipulate in your level.

For a list of gems that are enabled with the **Default** template, choose **Enable Gems** for your project as described in the following procedure.

The **Default** template also includes a Simple level, which is a starting point for you to manipulate objects in the editor and experiment with other assets in a neutral, gray environment. The Simple level includes a camera, a single light, an environment probe for reflections, and primitive objects with physics enabled. The objects are provided by the Primitive Assets Gem (p. 1133).

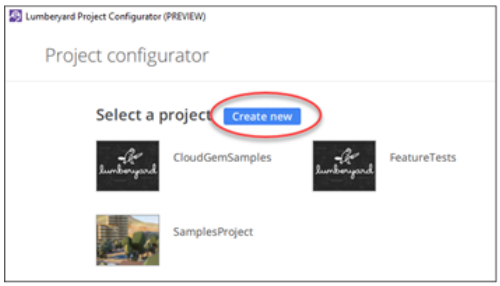
To create a game project in the Project Configurator

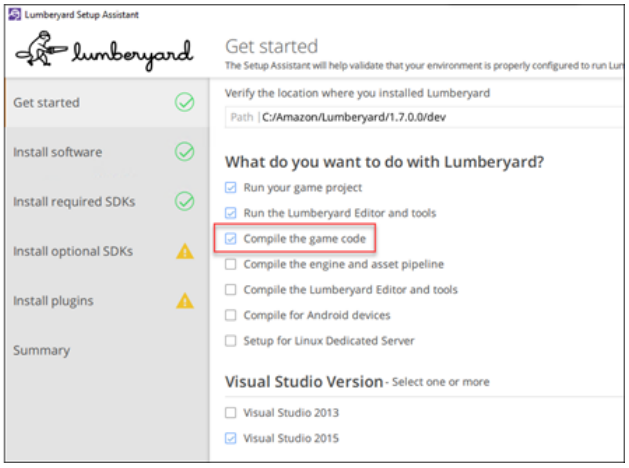
1. Open the Project Configurator by doing one of the following:

- Open Lumberyard Setup Assistant. On the **Summary** page, click **Configure project**.
- Navigate to the `lumberyard_version\dev\Tools\LmbrSetup\Win\` directory and then start `ProjectConfigurator`.

2. In the Project Configurator, choose **Create new**.

   Note
   If **Create new** is disabled, reopen Lumberyard Setup Assistant and select **Compile the game code**.



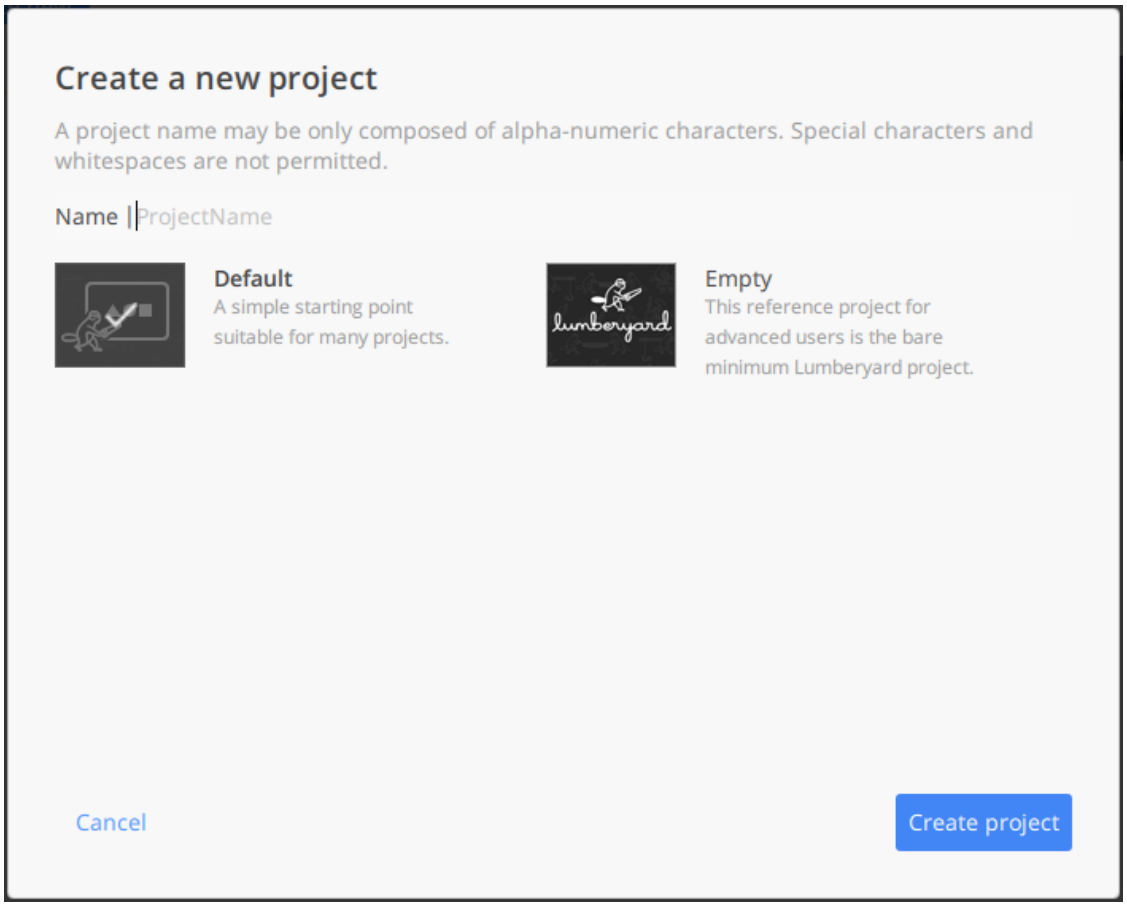


3. In the **Create a new project** window, do the following:

   a. Type a name for your project. Only alphanumeric characters are allowed. Do not use special characters or spaces in the name.

   b. Choose one of the following: **Default** or **Empty**.

   c. Click **Create project**.



4. When your project has been created, click **Continue**.
5. In the Project Configurator, choose the new project and click **Set as default** to make it the default project that Lumberyard Editor loads.
6. (Optional) Add gems to your project by choosing **Enable Gems**. If you choose **Code & Assets** gems, you must rebuild your project:

   a. In a command line window, change the directory to `lumberyard_version\dev\`. Type `lmbr_waf configure` to configure Lumberyard correctly.

   b. Build the game project. For more information, see Game Builds (p. 1847).
7. Close the Project Configurator.
8. Open Lumberyard Editor by doing one of the following:

   - Open Lumberyard Setup Assistant. On the **Summary** page, click **Launch editor**.
   - Start `Editor.exe` from one of the following directories:
     - If you are using Visual Studio 2013: `lumberyard_version\dev\Bin64vc120`
     - If you are using Visual Studio 2015: `lumberyard_version\dev\Bin64vc140`

Creating a Game Project from a Command Line

When you create a game project from a command line, you can type `lmbr` for a list of all possible commands that you can use with `Lmbr.exe`.

To create a game project from a command line (using Lmbr.exe (p. 1208))

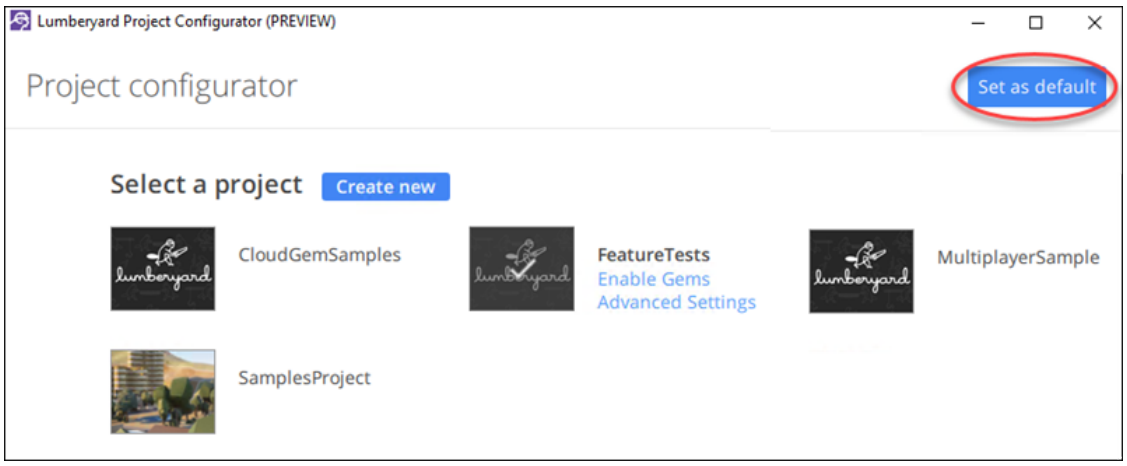
1. In a command line window, navigate to `lumberyard_version\dev\Tools\LmbrSetup\Win\`
2. Type `lmbr projects create MyNewProject`.
3. To set your newly created project as the default or active project, type `lmbr projects set-active MyNewProject`.
4. To configure Lumberyard to run this project, navigate to the `lumberyard_version\dev\` directory. Then type `lmbr_waf configure`.
5. Build the game project. For more information, see Game Builds (p. 1847).

Launching a Game Project

Use the Project Configurator to set a game project as the default project that the editor loads.

To launch an existing game project

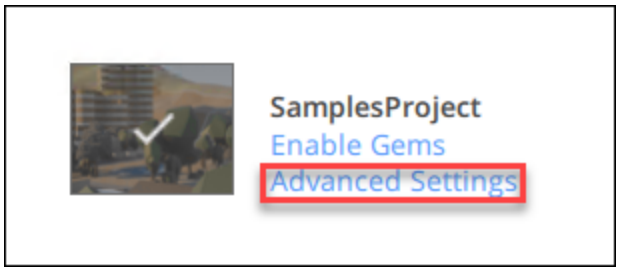
1. Open the Project Configurator by doing one of the following:
   - Open Lumberyard Setup Assistant. On the **Summary** page, click **Configure project**.
   - Go to the `lumberyard_version\dev\Tools\LmbrSetup\Win\` directory. Then start `ProjectConfigurator`.
2. Select a project and choose **Set as default** to make it the default project that Lumberyard Editor loads.



3. (Optional) Add gems to your project by choosing **Enable Gems**. If you choose **Code & Assets** gems, you must rebuild your project:
   a. In a command line window, change the directory to `lumberyard_version\dev\`. Type `lmbr_waf configure` to configure Lumberyard correctly.
   b. Build the game project. For more information, see Game Builds (p. 1847).
4. Close the Project Configurator.
5. Open Lumberyard Editor by doing one of the following:
   - Open Lumberyard Setup Assistant. On the **Summary** page, click **Launch editor**.
   - Start `Editor.exe` from one of the following directories:
     - If you are using Visual Studio 2013: `lumberyard_version\dev\Bin64vc120`
     - If you are using Visual Studio 2015: `lumberyard_version\dev\Bin64vc140`

Configuring Advanced Settings

To configure advanced settings for your selected project, click **Advanced Settings** in the **Project Configurator**.



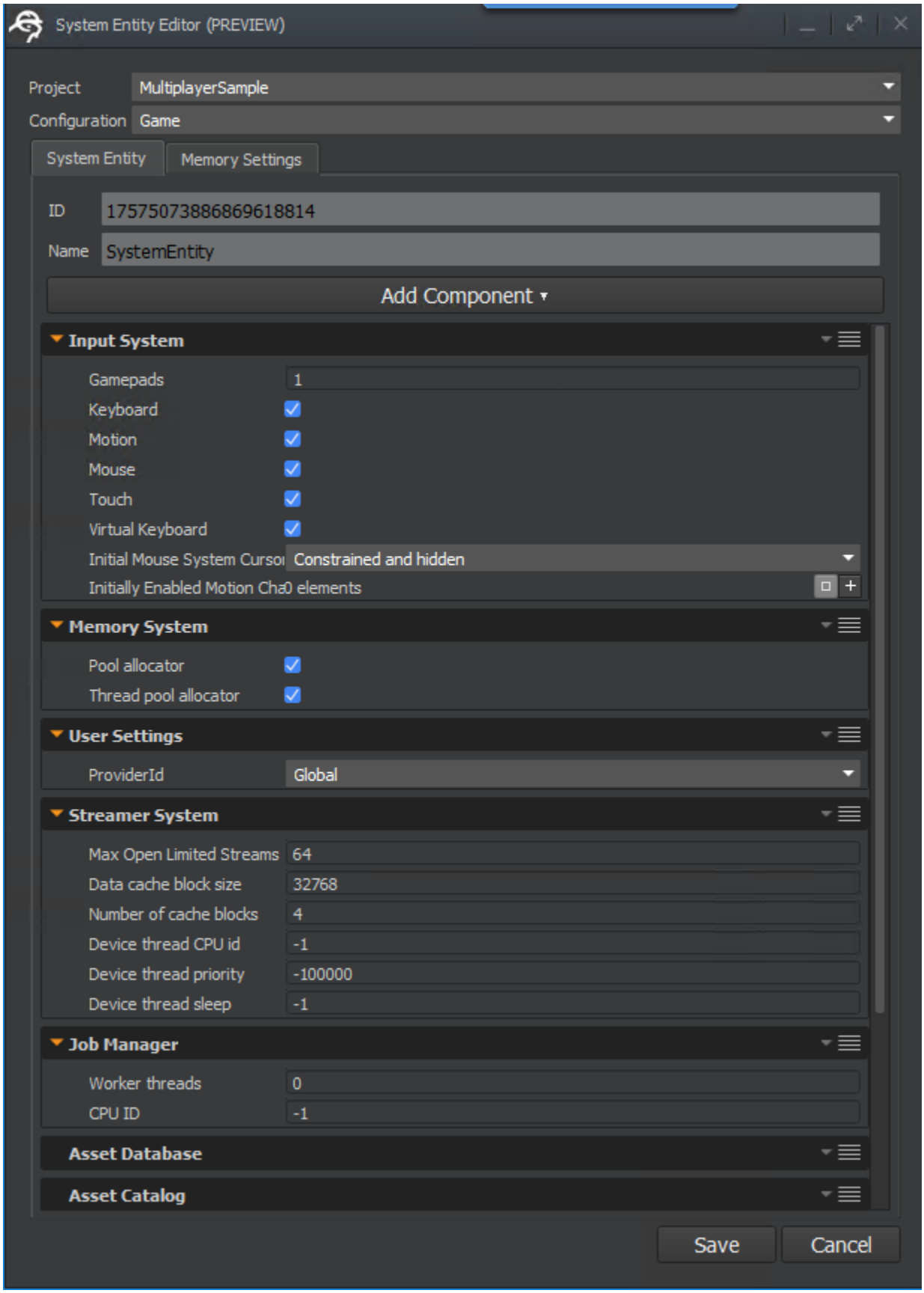
Upon launch, the **System Entity Editor** detects any required system components that are missing and adds them. **System Entity Editor** displays a dialog that lists all of the components that it added to your project.

The **System Entity Editor** interface has the following tabs:

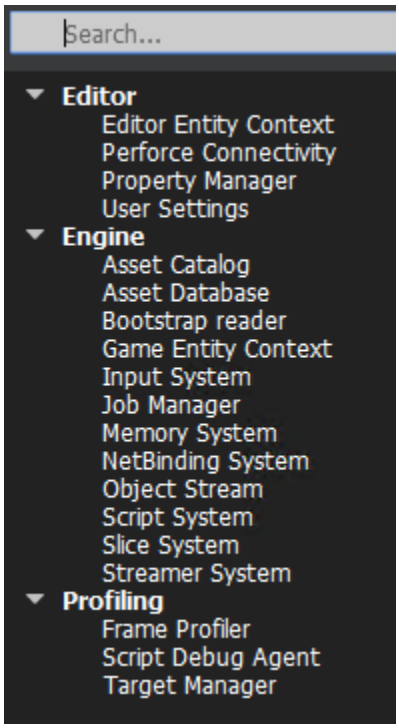
- **System Entity**
- **Memory Settings**

System Entity Editor

A single entity, shown in the following picture as `SystemEntity`, makes up the core of every Lumberyard application. This entity's components, known as system components, power major systems within Lumberyard. Using the **System Entity Editor**, developers can customize the project's components and configuration settings.

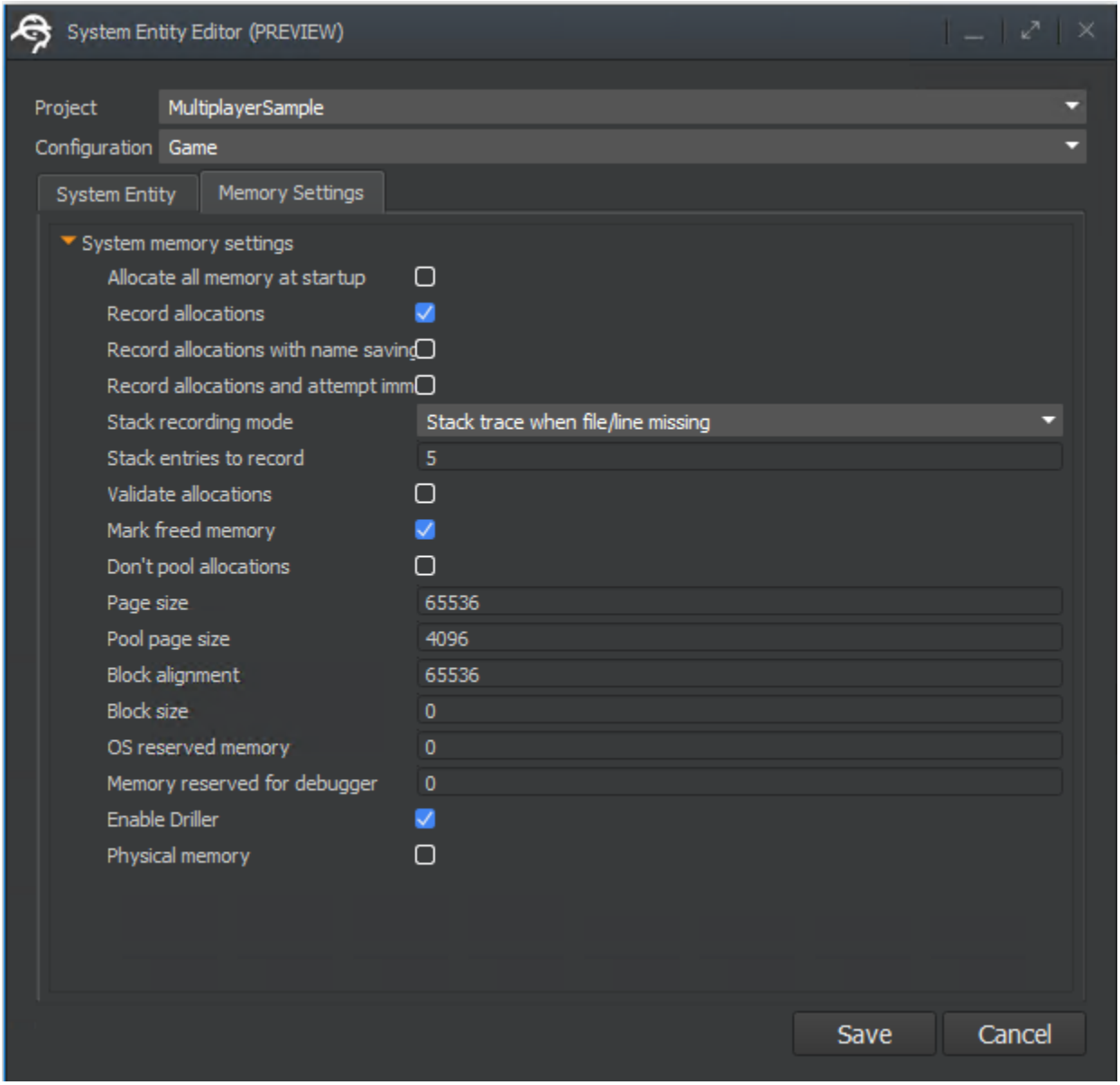

System Entity Editor (PREVIEW)
Project
MultiplayerSample
Configuration
Game
System Entity
Memory Settings
ID
17575073886869618814
Name
SystemEntity
Add Component
Input System
Gamepads
1
Keyboard
Motion
Mouse
Touch
Virtual Keyboard
Initial Mouse System Cursor
Constrained and hidden
Initially Enabled Motion Cha0 elements
Memory System
Pool allocator
Thread pool allocator
User Settings
ProviderId
Global
Streamer System
Max Open Limited Streams
64
Data cache block size
32768
Number of cache blocks
4
Device thread CPU id
-1
Device thread priority
-100000
Device thread sleep
-1
Job Manager
Worker threads
0
CPU ID
-1
Asset Database
Asset Catalog
Save
Cancel

To add a system component, click **Add Component**.



Memory Settings

Click the **Memory Settings** tab to configure and manage your project's memory settings.



How the Project Configurator Works

The Project Configurator is in preview release and is subject to change.

Understanding the inner workings of Lumberyard's Project Configurator may help you to troubleshoot and debug your projects.

The following describes what Project Configurator is doing when you perform certain actions.

Set Default (Active) Project

When you set the default or active project, Project Configurator modifies the following files with the following information:

`\dev\bootstrap.cfg`

The property `sys_game_folder` specifies the project that the editor and PC launcher attempts to load.

`\dev\_WAF_\user_settings.options`

The property `enabled_game_projects` is a comma-separated list of one or more projects to include in a build.

Create a New Project

When you create a new project in Project Configurator, several things happen:

- `EmptyTemplate` is used to create a new project. This template is located in `dev/ProjectTemplates/EmptyTemplate`.
- The contents of `/dev/ProjectTemplates/EmptyTemplate/code/EmptyTemplate` are copied to `/dev/code/NewProjectName`.
- The contents of `/dev/ProjectTemplates/EmptyTemplate/EmptyTemplate` are copied to `/dev/NewProjectName`.
- `EmptyTemplate` is replaced in both new locations with the name of your new project, including file names and file contents.

Enable or Disable Gems

When you use Project Configurator to enable or disable a gem in a project, Project Configurator updates the list of enabled gems that is maintained in `dev\`*`Project_Asset_Folder`*`\gems.json`.

- Disabling a gem removes that gem name from the list in `gems.json`.
- Enabling a gem adds that gem name to the list in `gems.json`.

Create a New Gem

When you use Project Configurator to create a new gem, the following takes place:

- The content of the gem template (which is part of Project Configurator resources and is not directly accessible) is copied into `\dev\Gems\`*`Gem_Name`*.
- Strings with the gem name in the copied content are substituted with your gem's name to make it a valid gem.

Troubleshooting

> The Project Configurator is in preview release and is subject to change.

Review the following if you experience issues when using the Project Configurator.

Perforce integration

If you have enabled Perforce integration with the tooling, then the Project Configurator automatically checks the following files out of Perforce. If not, then update the read only flag manually:

- *project_asset_folder*\gems.json
- *project_asset_folder*\game.cfg
- *engine_root_folder*\bootstrap.cfg
- *engine_root_folder*\dev*game_project_folder*\project.json

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.

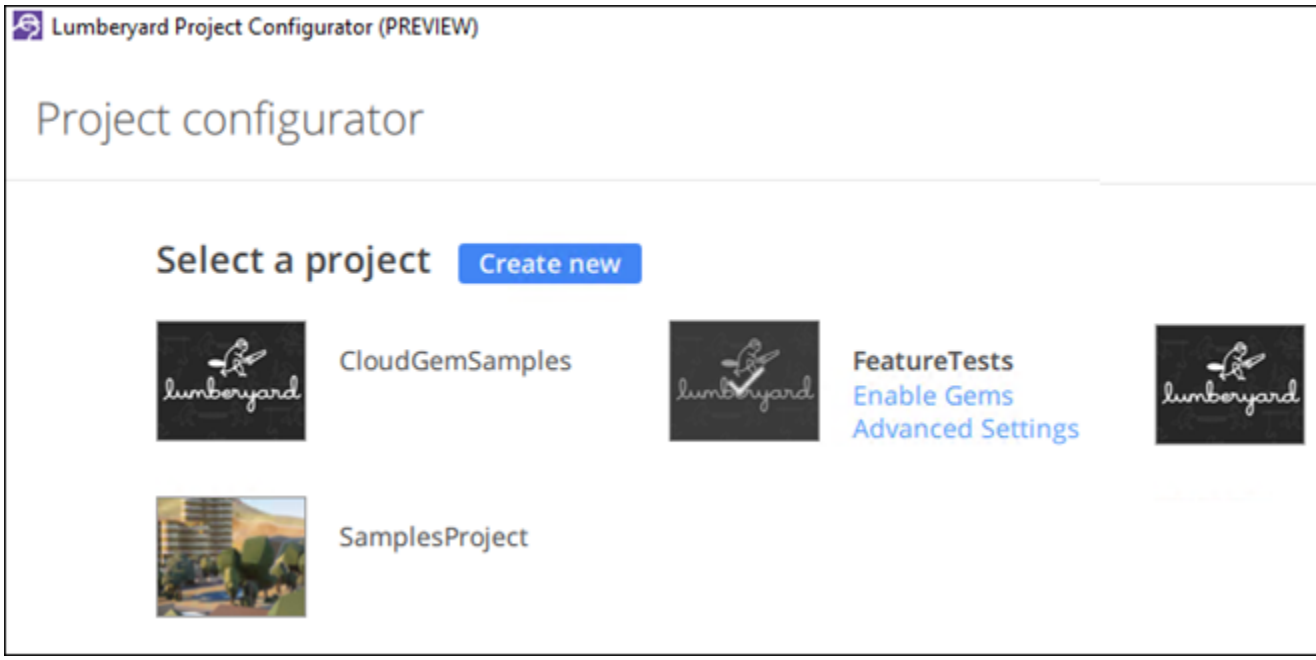
New project or gem does not appear in Visual Studio

Make sure that you have run `lmbr_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_folder*dev_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 **Set as default**.



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.

Migrating Lumberyard Projects

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

Topics

Lumberyard 1.10

Use the following instructions if you are upgrading your projects to Lumberyard 1.10.

Upgrading Projects

If you created your project in Lumberyard 1.9 or earlier, you must enable the **LmbrCentral Gem** in order to compile your project using Lumberyard 1.10.

If you do not follow these steps before configuring and compiling your project, you see the following error message:

```
Could not find a task generator for the name 'LmbrCentral'. Please verify that your selected
project is configured to be built. Current Settings: Spec: '', Config: 'project_generator', Platform
'project_generator'
```

For more information, see Gems (p. 1060).

To enable the LmbrCentral Gem in Project Configurator

1. In the Project Configurator, select your project and click **Set as default**.
2. Click **Enable Gems**.
3. The Project Configurator automatically adds the **LmbrCentral Gem** and notifies you of this change. Older `LmbrCentral` entries are also automatically removed from the `Game.xml` and `Editor.xml` configuration files.
4. Click **Save**.
5. Open a command line window and navigate to the `\dev` directory at the root of your Lumberyard installation.
6. Type `lmbr_waf configure`.
7. Do one of the following to build your project:
   - In Visual Studio, for **Build Configuration**, select one of the **[Game]** specs. You can use **[Game] Profile** to start.
   - In a command line window, do one of the following:
     - For Visual Studio 2013, type: `lmbr_waf build_win_x64_vs2013_profile -p game`

- For Visual Studio 2015, type: `lmbr_waf build_win_x64_vs2015_profile -p game`

To update your project using the command line

1. Navigate to the `\dev\Tools\LmbrSetup\Win` directory.
2. Launch the `lmbr.exe` command line tool.
3. Type the following to set the active project to the project that you want to update: `lmbr.exe projects set-active <project name>`
4. (Optional) Type the following to validate that the active project is set correctly: `lmbr.exe projects get-active`
5. Type the following command: `lmbr.exe projects populate-appdescriptors`
6. If there is no output, the `gems.json` and configuration files are updated.

Updating the Editor and Game Configuration Files

When you upgrade your project from Lumberyard 1.9 to Lumberyard 1.10, you may need to remove `LmbrCentral` references from the `editor.cfg` and `game.cfg` files for your project. `LmbrCentral` is now a gem.

The Project Configurator automatically updates the `editor.cfg` and `game.cfg` files for your project. However, you may receive compile errors if you do not run the Project Configurator.

To automatically update the configuration files

1. Open the **Project Configurator**. For more information, see Creating and Launching Game Projects (p. 28).
2. In the **Project Configurator**, choose your project and click **Set as default**.
3. Under your project name, choose **Enable Gems**.
4. On the **Enable Gems** page, click **Save**.

For reference, the previous XML appeared as follows:

```
<Class name="DynamicModuleDescriptor" field="element" type="{D2932FA3-9942-4FD2-
A703-2E750F57C003}">
<Class name="AZStd::string" field="dynamicLibraryPath" value="LmbrCentral" type="{EF8FF807-
DDEE-4EB0-B678-4CA3A2C490A4}"/>
</Class>
```

The new XML appears as follows:

```
<Class name="DynamicModuleDescriptor" field="element" type="{D2932FA3-9942-4FD2-
A703-2E750F57C003}" specializationTypeId="{D2932FA3-9942-4FD2-A703-2E750F57C003}">
<Class name="AZStd::string" field="dynamicLibraryPath" type="{EF8FF807-DDEE-4EB0-
B678-4CA3A2C490A4}" value="Gem.LmbrCentral.ff06785f7145416b9d46fde39098cb0c.v0.1.0"
 specializationTypeId="{189CC2ED-FDDE-5680-91D4-9F630A79187F}"/>
</Class>
```

Upgrading Your Starter Game Project

Starter Game has been updated for Lumberyard 1.10 compatibility.

If you have been making your own changes to Lumberyard 1.9 Starter Game, you must upgrade your existing Starter Game project to avoid losing your work when upgrading to Lumberyard 1.10.

- Material lifetime has been updated in order to address stability issues. As a result, any code or classes that store an IMaterial* in C++ are no longer valid. You must convert any code that acquires and stores an IMaterial* to `_smart_ptr<IMaterial>`.
- Enable the LmbrCentral Gem (p. 40) to compile your project in 1.10.
- Rebuild your project (p. 1847).

Converting AZ::IO::Print to PrintV

In Lumberyard 1.9 and earlier, the `AZ::IO::Print` function had two implementations:

- `int64_t Print(HandleType fileHandle, const char* format, ...);`
- `int64_t Print(HandleType fileHandle, const char* format, va_list arglist);`

In Lumberyard 1.10, the second implementation is now `PrintV` instead of `Print`: `int64_t PrintV(HandleType fileHandle, const char* format, va_list arglist);`

If you use the second implementation, you must update the function name to use `PrintV`:

- Lumberyard 1.9 and earlier: `AZ::IO::Print(fileHandle, format, arglist);`
- Lumberyard 1.0: `AZ::IO::PrintV(fileHandle, format, arglist);`

Lumberyard 1.9

Lumberyard 1.9 includes an upgrade script that helps you to change the version of the AWS Native SDK that you use.

AWS Native SDK Upgrade Script

To help automate the process of downloading and installing `.zip` files from the AWS SDK for C++ prebuild site, you can run an upgrade script. The script downloads files from the prebuild site and adds them to your `3rdParty` directory. By default, the script installs prebuilds of all platforms and services that are included with Lumberyard.

You can use the script to do the following:

- Change the version of the `AWSNativeSDK` that Lumberyard uses.
- Add or remove the list of services available within Lumberyard.

When you change the version or add or remove services, you must also modify the corresponding configuration files `SetupAssistantConfig.json`, `aws_native_sdk_shared.json`, and `aws_native_sdk_static.json` files.

Running the Upgrade Script

The script is located at `Tools\AWSNativeSDK\Upgrader\Upgrade.py`. The Python executable file must be in your Windows environment path.

To run the script, type the following syntax:

```
python upgrade.py <version> <optional destination>
```

The following example command specifies SDK version 1.0.74.

```
python upgrade.py 1.0.74 F:\3rdParty\AWS\AWSNativeSDK\1.0.74
```

Parameters

<version> – Specifies the SDK version number. To change the version that Lumberyard uses, you must also change the corresponding version numbers in the `dev\SetupAssistantConfig.json` file. For more information, see Changing Version Numbers (p. 43).

<optional destination> – Specifies the AWS Native SDK installation location. The default location is the Lumberyard `3rdParty` directory. If you use a different directory, specify the directory path in this parameter.

Changing Version Numbers

To change SDK versions, you must change the version numbers in the `dev\SetupAssistantConfig.json` file. Modify the locations in the file as indicated in the following example.

```
{
    "identifier": "AWSNativeSDK",
    "name": "AWSNativeSDKName",
    "version": "1.0.74",                          <-----   Change version number
    "source": "AWS/AWSNativeSDK/1.0.74",          <-----   Change version number
    "optional": 0,
    "description": "AWSNativeSDKDescriptionSummary",
    "detailedInstructions": "AWSNativeSDKDetailedInstructions",
    "roles": [ "compilegame", "compileengine", "compileeditor", "compileios",
 "compileandroid", "setuplinux" ],
    "symlinks": [
        {
            "source": "AWS/AWSNativeSDK/1.0.74",   <-----   Change version number here and
 in
                                                             similar following occurrences.
```

Customizing Platforms and Services

To customize platforms and services, modify the code in the `Upgrade.py` script.

Customizing Platforms

To customize the list of platforms that you use, modify the `get_platform_list():` call. The following code shows the default platform list.

```
def get_platform_list():
    platform_list = []

    platform_list.append(get_windows_vs2015())
    platform_list.append(get_windows_vs2013())
    platform_list.append(get_darwin())
    platform_list.append(get_ios())

    platform_list.append(get_appletv())
    platform_list.append(get_android())
    platform_list.append(get_linux())
```

Customizing Services

To customize services for all platforms or individual platforms, modify the script as in the following example for Microsoft Visual Studio 2015 on Windows.

```
def get_windows_library_list():
    return_list = get_default_library_list()

    ## Libraries by customer request
    return_list.append('access-management')
    return_list.append('transfer')

    return return_list

def get_windows_vs2015():
    vs2015 = {}
    vs2015['platform'] = 'windows'
    vs2015['zipfile'] = 'aws-sdk-cpp_x86_64_visual_cpp-14x.zip'
    vs2015['libraries'] = get_windows_library_list()
    vs2015['libextensions'] = ['.dll', '.lib']

    return vs2015
```

To add a service to Windows, add the service to `get_windows_library_list()`. The following example adds the `polly` service to Windows.

```
return_list.append('polly')
```

Making a Service Accessible to Wscript

To make a service accessible within your `.wscript` files, add the service name to the `dev\_WAF_\3rdParty\aws_native_sdk_shared.json` and `dev\_WAF_\3rdParty\aws_native_sdk_static.json` files.

Lumberyard 1.8

In Lumberyard 1.8, the behavior context replaces the script context. The behavior context works with serialize context, edit context, and network context to provide rich C++ reflection. Behavior context focuses on the run-time aspects of C++ code and allows you to manipulate C++ code and objects while they are created. All script bindings, including Lua, use this reflection. Reflection is also used to modify objects while in running state (such as animating object properties) and read current properties for component state transitions. You can have multiple behavior contexts that are specialized for different purposes, and you can unreflect the behavior contexts in order to implement reloading. With the introduction of behavior context, you can no longer reflect directly into the script context.

After upgrading your projects to Lumberyard 1.8, do the following to migrate your existing projects into the behavior context format:

1. Convert your C++ code from the script context to the behavior context.
2. Convert your Lua script syntax.

Converting C++ Code from the Script Context to the Behavior Context

The script context is still used to manage a Lua VM instance; however, it now binds to a behavior context instance. The behavior context uses the property, method, class, EBus, and attributes primitives. For more information, see Behavior Context in the *Amazon Lumberyard Developer Guide*. You must convert any event buses that expose functionality to Lua using the script context.

To convert your C++ code from the script context to the behavior context

1. Convert any EBus senders to behavior context handlers.
2. Use the behavior context in your component's `Reflect` method to expose the EBus handlers to Lua.
3. Remove the old `AZ_SCRIPTABLE_EBUS` macro calls.

Migrating Globals, Classes, and EBus

You can see all support script attributes in `AZ::Script::Attributes`, for example `Storage`, `MethodOverride`, `ConstructorOverride`, and `Operator`. This section includes examples for certain migrations.

You can use the same method as before in order to implement generic functionality, such as accepting any number and type of arguments. The script context binding to the behavior context provides some automation. For example, if a function signature accepts `ScriptDataContext` as the last parameter, the signature always uses that function. Instead of unpacking the arguments from the script VM, it wraps the arguments in `ScriptDataContext` and allows you to handle the arguments directly.

Method

Conversion is required for methods if you use class operators or have default values, as in the following example.

```
scriptContext.Method("GlobalFunction",&GlobalFunction, scriptContext.DefaultValues(555),
 "Method description for authors");behaviorContext.Method("GlobalFunction",&GlobalFunction,
 BehaviorMakeDefaultValues(555),
      "Method description for authors");
```

Fields

Fields are no longer supported. Each value is now a property that you can control. Read/write values are determined with setters and getters, as in the following example.

```
scriptContext.Field("globalField", &s_globalField);behaviorContext.Property("globalField",
 BehaviorValueProperty(&s_globalField));
```

Attributes

In the behavior context, both methods and properties support attributes. Because the behavior context is more generic, you may want to reflect a C++ function for C++ calls like animating a value. You may also want to have a different function for the script context. You can reflect the function that you used to reflect only to the script context. The following script attribute is still supported.

```
scriptContext.Method("MethodFooForScriptOnly",&MethodFooForScriptOnly);behaviorContext.Method("MethodFo
```

Classes

Classes are generic and no longer require specific ownership or default constructors during class declaration. These values have been moved to attributes.

Constructors

Similar to methods, the script context provides automation with constructors by using the `ScriptDataContext` parameter and applying the same operations for a method. By default, script owns any objects that are created in script. C++ runtime owns any objects that are passed into script. You can change this behavior by using the `Acquire` or `Release` functions in C++ or Lua. The `MyClass` example below has a `MyClass(int)` constructor and a custom allocator/deallocator and is owned by

script. You can provide the class name; however, by default the `AzTypeInfo` is used to determine the class name. You can specify the `int` argument for constructor to include as many as you want.

```
scriptContext.Class<MyClass, void(int), ScriptContext::SP_RAW_RUNTIME_OWN>("MyClass",
 &MyClassAllocate, &MyClassDestructor)->
        ... methods, enums, constants and properties
behaviorContext.Class<MyClass>()
        ->Constructor<int>()
        ->Allocator(&MyClassAllocate, &MyClassDestructor)
        ... methods, enums, constants and properties

// If the class is passed by value
scriptContext.Class<MyValueClass, ScriptContext::SP_VALUE>("MyValueClass")
        ... methods, enums, constants and properties
behaviorContext.Class<MyValueClass>()
        -
>Attribute(AZ::Script::Attributes::Storage,AZ::Script::Attributes::StorageType::Value)
        ... methods, enums, constants and properties

// Operators are script/Lua-specific and are now moved to attributes
scriptContext.Class<Uuid, void (ScriptDataContext&), ScriptContext::SP_VALUE>("Uuid")
        ->Operator(ScriptContext::OPERATOR_TOSTRING, &Uuid::ToString)
        ...
behaviorContext.Class<Uuid>()
        -
>Attribute(AZ::Script::Attributes::Storage,AZ::Script::Attributes::StorageType::Value)
        ->Method("ToString",&Uuid::ToString)
            -
>Attribute(AZ::Script::Attributes::Operator,AZ::Script::Attributes::OperatorType::ToString)
        ...
```

EBus

Migrating the EBus binding requires different steps than methods and classes. See the following EBus class `MyEvents` example for more information.

```
// EBus class MyEvents : public AZ::EBusTraits
{
    float OnEvent(float a, float b) = 0;
};

using MyEBus = AZ::EBus<MyEvents>;

// Script context
AZ_SCRIPTABLE_EBUS(
        MyEBus ,   // Generates class MyEBusHandler
        MyEBus ,   // Generates class MyEBusSender
        "{8165B431-DEAB-4033-984C-2400A74C69F8}", // Handler type UUID
        "{6B160DDE-36CB-4876-B09D-E848503C7AE0}", // Sender type UUID
        AZ_SCRIPTABLE_EBUS_EVENT_RESULT(float,0.0f,OnEvent, float, float) // Reflects an
 event with result float; the default result is 0.0f. The "OnEvent" name must match and has
 two float parameters.
)

// Reflecting the EBus (usually a component reflect function) required a call
 ScriptableEBus_MyEBus::Reflect(scriptContext).

// In Lua the code appeared as follows
-- To send messages
myEBusSender = MyEBusSender()
result = myEBusSender:OnEvent(10.0,20.0) -- call on event

-- To listen to messages
MyEBusHandlerTable = {
```

```
    OnEvent = function(self,a,b)
        -- do something
        return result;
    end
};

myEBusHandler = MyEBusHandler(MyEBusHandlerTable /* or , ID when the bus has an ID */);

-- To prevent disconnection
myEBusHandler:Disconnect()

// The implementation above had a sender that required an ID (0 for no ID) and did not
 support broadcast versus event or queuing of any events functions.
// In addition, when the event returned const ClassX& it was necessary to implement
 the class without the macros in order to create temporary storage and return that as a
 reference.

// With the behavior context, sender is no longer required and the handler is written as an
 EBus listener.
// Note that a sender and handler are not required and you can provide either depending on
 your requirements.
// It is common to have one bus for requests and one bus for notifications. This pattern is
 used in most components.

// MyEBus behavior context handler class
class MyEBusBehaviorHandler : public MyEBus::Handler, public AZ::BehaviorEBusHandler
{
public:
    AZ_EBUS_BEHAVIOR_BINDER(MyEBusBehaviorHandler, "{19F5C8C8-4260-46B1-
B624-997CD3F10CBD}", AZ::SystemAllocator, // Helper macro to set the handler type ID and
 allocator
                                OnEvent); // Comma-separated names of events to handle

    float OnEvent(float a, float b) override   // Implements MyEBus::Handler OnEvent
    {
        float result = 0.0f; // Sets the default value for the result if CallResult doesn't
 call anything
        CallResult(result, FN_OnEvent, a, b); // FN_OnEvent is generated from the
 macro above. You can cache the function index if you want static int eventIndex =
 GetFunctionIndex("OnEvent"); CallResult(result,eventIndex,a,b); if you prefer
        return results;
    }
};

// Wherever you reflect the EBus (usually a component or system component reflect function)
behaviorContext.EBus<MyEBus>("MyEBus")
    ->Handler<MyEBusBehaviorHandler >() // Allows systems that use the behavior context to
 create handlers for this EBus each time they need to listen for events. You can reflect a
 bus without a handler so that behavior context users can only send events.
    ->Event("OnEvent",&MyEBus::Events::OnEvent) // Allows the behavior context system to
 send "OnEvent" events. The code automatically generates Broadcast, Event, QueueBroadcast,
 QueueEvent, and QueueFunctions if the EBus configuration supports them. You aren't
 required to provide events; you can provide only a handler if you don't have the behavior
 context systems to send events.

-- In Lua
-- You no longer need to create sender objects. You can send events using the same method
 as in C++.
result = MyEBus.Broadcast.OnEvent(10.0,20.0) -- Or you can use result =
 MyEBus.Event.OnEvent(id,10.0,20.0) for issues on an ID, or you can queue events to
 MyEBus.QueueBroadcast.OnEvent(10.0,20.0)

-- To listen
MyEBusHandlerTable = {
    OnEvent = function(self,a,b)
        -- do something
```

```
        return result;
    end
};

myEBusHandler = MyEBus.Connect(MyEBusHandlerTable) -- You can pass an ID when the bus
 supports it

-- To prevent disconnection
myEBusHandler:Disconnect()

// The implementation above is generic. The handler is shared whether you use Lua, Visual
 Script, or another behavior context-based system. You can now use most of the EBus
 functionality that is available
// in C++.
```

Converting the Lua Script Syntax

Several improvements to the Lua syntax provide better flexibility and more similarity to the C++ syntax. The older script context has been replaced with a new behavior context that provides a more robust EBus script reflection. As a result, the interfaces for sending events and receiving notifications has changed. For more information, see Writing Lua Scripts for the Component Entity System in the *Amazon Lumberyard Developer Guide*.

Using Events Instead of Senders

With the behavior context, sender objects are no longer used to send events or make requests on an EBus. You can now send an event directly using an entity ID or EBus as the destination address. Each bus has a new event table that defines the functions for sending the appropriate events and requests.

Script context syntax (previous):

```
self.uiCanvasLuaBusSender = UiCanvasLuaBusSender(self.canvasEntityId)
local entityWithFader = self.uiCanvasLuaBusSender:FindElementById(2)
local otherEntity = self.uiCanvasLuaBusSender:FindElementById(3)
```

Behavior context syntax (new):

```
local entityWithFader = UiCanvasLuaBus.Event.FindElementById(self.canvasEntityId, 2)
local otherEntity = UiCanvasLuaBus.Event.FindElementById(self.canvasEntityId, 3)
```

Using a Connect Method for Handlers

The syntax for handlers now uses a `Connect` method that is similar to C++.

Script context system (previous):

```
self.uiCanvasNotificationHandler = UiCanvasNotificationLuaBusHandler(self, canvasEntityId)
```

Behavior context system (new):

```
self.uiCanvasNotificationHandler = UiCanvasNotificationLuaBus.Connect(self, canvasEntityId)
```

If you do not want to connect immediately, use the following:

```
self.uiCanvasNotificationHandler = UiCanvasNotificationLuaBus.CreateHandler(self)
```

Then, if you want to connect later, use the following:

```
self.uiCanvasNotificationHandler:Connect(canvasEntityId)
```

Creating a Local Instance of Your Script Component

You can now create a local instance of your script component and return it at the end of the file. The examples below refer to `MyScriptFile.lua`.

In the previous script context system, the table name had to match the file name:

```
myscriptfile = {}
```

In the new behavior context system, the table name does not have to match the file name:

```
local ScriptName = {}
```

Use the following Lua script code to return the table that the engine uses.

```
return ScriptName
```

Treating Entity References as Any Other Property

Entity references no longer use a special syntax and can be treated the same as any other property that contains a reflected type.

Script context system (previous):

```
myscriptfile = {
    Properties = {
        Property = { entity = '' },
    }
}
```

Behavior context system (new):

```
local ScriptName = {
    Properties = {
        Property = { default = EntityId() },
        AltProp = EntityId(),
    }
}

return ScriptName
```

Registering to Receive Notifications

The syntax for registering to receive notifications has changed to use the `Connect()` functions that are defined by each notification bus to connect to a notification bus. Previously you were required to create a globally defined handler table. The `Connect()` functions return an object that you must hold in order to disconnect from the notification bus. You can also use the `CreateHandler()` function to create the handler object and then use the `Connect()` function on that handler object, if you want to separate handler creation from the bus connection.

Script context system (previous):

```
function myscript:OnActivate()
 if self.uiCanvasNotificationHandler == nil then
  self.uiCanvasNotificationHandler = UiCanvasNotificationLuaBusHandler(self,
 canvasEntityId)
 end
end

function myscript:OnAction(entityId, actionName)
    Debug.Log("Action Notification: " .. entityId.id .. ": " .. actionName)
end

function myscript:OnDeactivate()
 if self.uiCanvasNotificationHanlder ~= nil then
  self.uiCanvasNotificationHandler:Disconnect()
  self.uiCanvasNotificationHanlder = nil
 end
end
```

Behavior context system (new):

```
function myscript:OnActivate()
 self.uiCanvasNotificationHandler = UiCanvasNotificationLuaBus.Connect(self,
 canvasEntityId)

end

function myscript:OnAction(entityId, actionName)
    Debug.Log("Action Notification: " .. tostring(entityId) .. ": " .. actionName)
end

function myscript:OnDeactivate()
 if self.uiCanvasNotificationHanlder ~= nil then
  self.uiCanvasNotificationHandler:Disconnect()
  self.uiCanvasNotificationHandler = nil
 end
end
```

Alternate option for the behavior context system (new):

```
function myscript:OnActivate()
    self.uiCanvasNotificationHandler = UiCanvasNotificationLuaBus.CreateHandler(self)
    self.uiCanvasNotificationHandler:Connect(canvasEntityId)
end
```

Passing the EntityId Object

The `.id` property is now deprecated and can no longer be used to access or pass entity IDs. You can now pass the `EntityId` object.

Converting the Main Lua Script Table

Lumberyard Lua scripting is now aligned with standard Lua practices. Previously the engine could only recognize the main script table if it had the same name as the script file. Now, the main script table can be local to the file and named as you prefer, as long as the table is returned at the end of the file.

Script context system (previous):

```
myscriptfile = {}

function myscriptfile:OnActivate()
```

```
end

function myscriptfile:OnDeactivate()
end
```

Behavior context system (new):

```
local ScriptName =
{}

function ScriptName:OnActivate()
end

function ScriptName:OnDeactivate()
end

return ScriptName;
```

Importing FBX Files with Multiple UV Streams

When upgrading to Lumberyard 1.8, any `.fbx` files that you previously imported in Lumberyard 1.7 or earlier now imports two UV channels, if present. Previous versions of Lumberyard imported only the first UV channel. This can result in an additional, unexpected UV channel on certain meshes. If you do not want the additional channel, you can use a DCC tool such as Autodesk 3ds Max, Autodesk Maya, or Blender to remove the channel.

Converting Material Assets

Lumberyard 1.8 introduces a feature called independent texture tiling. With this feature you can author independent texture modulator values in the second diffuse, emittance, or detail maps. Lumberyard provides a conversion script to ensure your material asset files (`.mtl`) work properly with the new feature. You must run the conversion script if you have authored material assets that modified the tiling, rotation, or oscillation values in the main diffuse map. You can choose not to run the conversion script if you haven't changed material values. However, the script converts only updated materials and is safe to run as a precaution. You only need to run the script once per project.

To download the conversion script

1. Download the conversion script here.
2. Save the `1.8_IndependentTilingConvertor.py` file to the `\dev\Editor\Scripts\migration\1.8` directory. You might need to create this directory.

To run the conversion script from a command prompt

1. In a command line window, navigate to the Python directory for your Lumberyard installation. For example, *`lumberyard_root`*`\dev\Tools\Python\2.7.11\windows`
2. Run the conversion script by typing the following: `python` *`lumberyard_root`*`\dev\Editor\Scripts\migration\1.8\1.8_IndependentTilingConvertor.py`
3. (Optional) Run the script on a single project or multiple projects.
   - To run the script on a single project, run the following command: `python` *`lumberyard_root`*`\dev\Editor\Scripts\migration\1.8\1.8_IndependentTilingConvertor.py "my_project"`
   - To run the script on multiple projects at once, use commas to separate the project names and run the following command: `python` *`lumberyard_root`*`\dev\Editor\Scripts\migration\1.8\1.8_IndependentTilingConvertor.py "my_project,my_other_project"`

Note
You can convert all materials in all projects within your Lumberyard build by omitting the project name.

To run the conversion script from Lumberyard Editor

1. In Lumberyard Editor, click **Tools**, **Other**, **Python Scripts**.
2. In the **Python Scripts** window, navigate to and click **1.8_IndependentTilingConvertor**.
3. Click **Execute**.

Lumberyard 1.7

If you are upgrading your projects to Lumberyard 1.7, use the following instructions to migrate your projects with AWS resources managed by the Cloud Canvas Resource Manager and created using previous versions of Lumberyard.

To update your projects with AWS resources to work with Lumberyard 1.7

1. Use the `lmbr_aws update-project-code` command to update the contents of the `\project\AWS\project-code` directory.

   Note
   If you use source control, you must check out these files before executing the `update-project-code` command.

   After executing the `update-project-code` command, you must add any new files in this directory to your source control system.
2. Edit the `project-template.json` file (located in the `\project\AWS` directory) to make the following changes:

   a. In the `Parameters` object, add a `CloudCanvasStack` parameter as shown below. The default value for this parameter is Project. The `ConfigurationKey` parameter should be present and remain unchanged.

   ```
   "Parameters": {
           "CloudCanvasStack": {
               "Type": "String",
               "Description": "Identifies this stack as a Lumberyard Cloud Canvas
    managed stack.",
               "Default": "Project"
           },
           "ConfigurationKey": {
               "Type": "String",
               "Description": "Location in the configuration bucket of configuration
    data."
           }
       },
   ```

   b. In the `Configuration` bucket resource definition, remove the `DeletionPolicy` property as shown below.

   ```
   "Resources": {

         ...
   ```

```
        "Configuration": {
            "Type": "AWS::S3::Bucket",
            "DeletionPolicy": "Retain", <==== DELETE THIS LINE
            "Properties": {
...
```

3. Edit the `deployment-access-template.json` file (located in the `\project\AWS` directory) to add the `CloudCanvasStack` parameter as shown below. The default value for this parameter is DeploymentAccess.

   The following parameters should be present and remain unchanged: `ProjectResourceHandler`, `ConfigurationBucket`, `ConfigurationKey`, `ProjectPlayerAccessTokenExchangeHandler`, `ProjectStack`, `DeploymentName`, `DeploymentStack`, and `DeploymentStackArn`.

```
"Parameters" : {

        "CloudCanvasStack": {
            "Type": "String",
            "Description": "Identifies this stack as a Lumberyard Cloud Canvas managed
 stack.",
            "Default": "DeploymentAccess"
        },

        "ProjectResourceHandler": {
            "Type": "String",
            "Description": "Service token of the custom resource handler."
        },

  ...
```

4. Edit the `deployment-template.json` file (located in the `\project\AWS` directory) to make the following changes:

   a. Add a `CloudCanvasStack` parameter as shown below. The default value for this parameter is `Deployment`.

      The following parameters should be present and remain unchanged: `ProjectResourceHandler`, `ConfigurationBucket`, `ConfigurationKey`, `DeploymentName`, and `ProjectStackId`.

```
"Parameters" : {

        "CloudCanvasStack": {
            "Type": "String",
            "Description": "Identifies this stack as a Lumberyard Cloud Canvas
 managed stack.",
            "Default": "Deployment"
        },

        "ProjectResourceHandler": {
            "Type": "String",
            "Description": "Service token of the custom resource handler."
        },

        ...
```

   b. For each of the `AWS::CloudFormation::Stack` resource definitions in the `Resources` object, add the following parameters as shown below: `DeploymentStackArn`, `DeploymentName`, and `ResourceGroupName`.

      You must change the value for the `ResourceGroupName` parameter for each stack resource to use the resource name.

The following parameters should be present and remain unchanged: `ProjectResourceHandler`, `ConfigurationBucket`, and `ConfigurationKey`.

```
"Resources": {

        ...
        "DontDieAWS": {
            "Type": "AWS::CloudFormation::Stack",
            "Properties": {
                "TemplateURL": { "Fn::GetAtt": [ "DontDieAWSConfiguration",
 "TemplateURL" ] },
                "Parameters": {
                    "DeploymentStackArn": { "Ref": "AWS::StackId" },
                    "DeploymentName": { "Ref": "DeploymentName" },
                    "ResourceGroupName": "DontDieAWS", <== Change this value to
 match the resource name for each stack resource.
                    "ProjectResourceHandler": { "Ref": "ProjectResourceHandler" },
                    "ConfigurationBucket": { "Fn::GetAtt":
 [ "DontDieAWSConfiguration", "ConfigurationBucket" ] },
                    "ConfigurationKey": { "Fn::GetAtt":
 [ "DontDieAWSConfiguration", "ConfigurationKey" ] }
                }
            }
        }
        ...
```

5. Edit the `resource-template.json` file (located in the *`project`*`\resource-group\`*`resource-group-name`* directory for each of your projects) to add the following parameters as shown below: `CloudCanvasStack`, `DeploymentStackArn`, `DeploymentName`, and `ResourceGroupName`. The default value for the `CloudCanvasStack` parameter is ResourceGroup.

   The following parameters should be present and remain unchanged: `ProjectResourceHandler`, `ConfigurationBucket`, and `ConfigurationKey`. Any other resource group-specific parameters must also remain unchanged.

```
"Parameters": {
        "CloudCanvasStack": {
            "Type": "String",
            "Description": "Identifies this stack as a Lumberyard Cloud Canvas managed
 stack.",
            "Default": "ResourceGroup"
        },
        "DeploymentStackArn": {
            "Type": "String",
            "Description": "ARN of the deployment stack that owns this resource group
 stack."
        },
        "DeploymentName": {
            "Type": "String",
            "Description": "Name of the resource group's deployment."
        },
        "ResourceGroupName": {
            "Type": "String",
            "Description": "Name of the resource group."
        },
        "ProjectResourceHandler": {
            "Type": "String",
            "Description": "Service token of the custom resource handler."
        },
         ...
```

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.

Topics

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 EBuses 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. Modifying 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
`CRY_UNIT_TEST_SUITE(SuiteName)`	No replacement. You can safely remove these from your code.
`CRY_UNIT_TEST(TestName)`	`TEST(SuiteName, TestName)`
`CRY_UNIT_TEST_FIXTURE(FixtureName)`	`class FixtureName : public ::testing::Test`

CryUnitTest	GoogleTest
`CRY_UNIT_TEST_WITH_FIXTURE(TestName, FixtureName)`	`TEST_F(FixtureName, TestName)`
`CRY_UNIT_TEST_ASSERT(Condition)`	`ASSERT_TRUE(condition)`
`CRY_UNIT_TEST_CHECK_CLOSE(ValueA, ValueB, Epsilon)`	`ASSERT_NEAR(ValueA, ValueB, Epsilon)`
`CRY_UNIT_TEST_CHECK_EQUAL(ValueA, ValueB)`	`ASSERT_EQ(ValueA, ValueB)`
`CRY_UNIT_TEST_CHECK_DIFFERENT(ValueA, ValueB)`	`ASSERT_NE(ValueA, ValueB)`

B. Moving 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. 1508).

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. Building and Running Tests

After you complete these migration steps, you can build and run your new tests. For more information, see Using AZ Test Scanner (p. 1508).

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`.

Migrating from CryEngineNonRCModule to CryEngineModule

`CryEngineNonRCModule` has been removed. If you are upgrading your projects from Lumberyard 1.4 or earlier, you must update all references of `CryEngineNonRCModule` to `CryEngineModule` in your wscript files.

Lumberyard 1.5

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

Topics

Upgrading Decal Components

Lumberyard 1.5 introduces improvements to the decal component. These changes may affect the rendering of legacy and pre-1.5 decal components. Manually check each existing decal to ensure that they are rendering as intended, and that attributes are set correctly.

The following changes to the decal component were made for Lumberyard 1.5:

- Projection type **OnStaticObjects** has been removed
- The decal component now has the following projection types:
  - **Planar** (forward rendered)
  - **On Terrain** (forward rendered)
  - **On Terrain and Static Objects** (deferred rendered)

Use the following procedure to run a script that aids you in upgrading your legacy decals from Lumberyard 1.4 and earlier to Lumberyard 1.5 and later.

To run the legacy decal upgrade script

1. Create a backup of the project(s) for which you intend to run this script. Once you run this script, you cannot back out of the changes it performs. To protect against the loss of data, be sure to perform this step.
2. Download the script. Extract it to Lumberyard's `\dev` directory.
3. Open a command prompt. Navigate to Lumberyard's `dev` directory.
4. Use the Lumberyard bundled version of python to execute the script on your project or your project's level.

   For example:

   - `\dev\Tools\Python\2.7.11\windows\python.exe FixDecalsPy27.py D:\Lumberyard\1.6.0.0\dev\BeachCity`
   - `\dev\Tools\Python\2.7.11\windows\python.exe FixDecalsPy27.py D:\Lumberyard\1.6.0.0\dev\BeachCity\Levels\BeachCity_NightTime`
   - `\dev\Tools\Python\2.7.11\windows\python.exe FixDecalsPy27.py D:\Lumberyard\1.6.0.0\dev\BeachCity\Levels\BeachCity_NightTime\levels\Layers\`

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\lmbr.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\lmbr.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. 437). 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. Renaming 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. Modifying 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`:

      i. Copy the all lower-case UUID from your `gem.json` file.

      ii. 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. Editing 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. Migrating Your Configuration 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:

```
lmbr.exe projects populate-appdescriptors
```

Migrating from CryEngineNonRCModule to CryEngineModule

`CryEngineNonRCModule` has been removed. If you are upgrading your projects from Lumberyard 1.4 or earlier, you must update all references of `CryEngineNonRCModule` to `CryEngineModule` in your wscript files.

Migrating from AZCore to AzCore

In Lumberyard 1.5, AzCore replaces AZCore. In code you will notice this update in AzCore header files:

- Previously: `#include <AZCore/AZCore/headername.h>`
- Updated: `#include <AzCore/AzCore/headername.h>`

Update your code to replaces all AZCore references with AzCore.

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 one of the following directories and double-clicking `Editor.exe`:

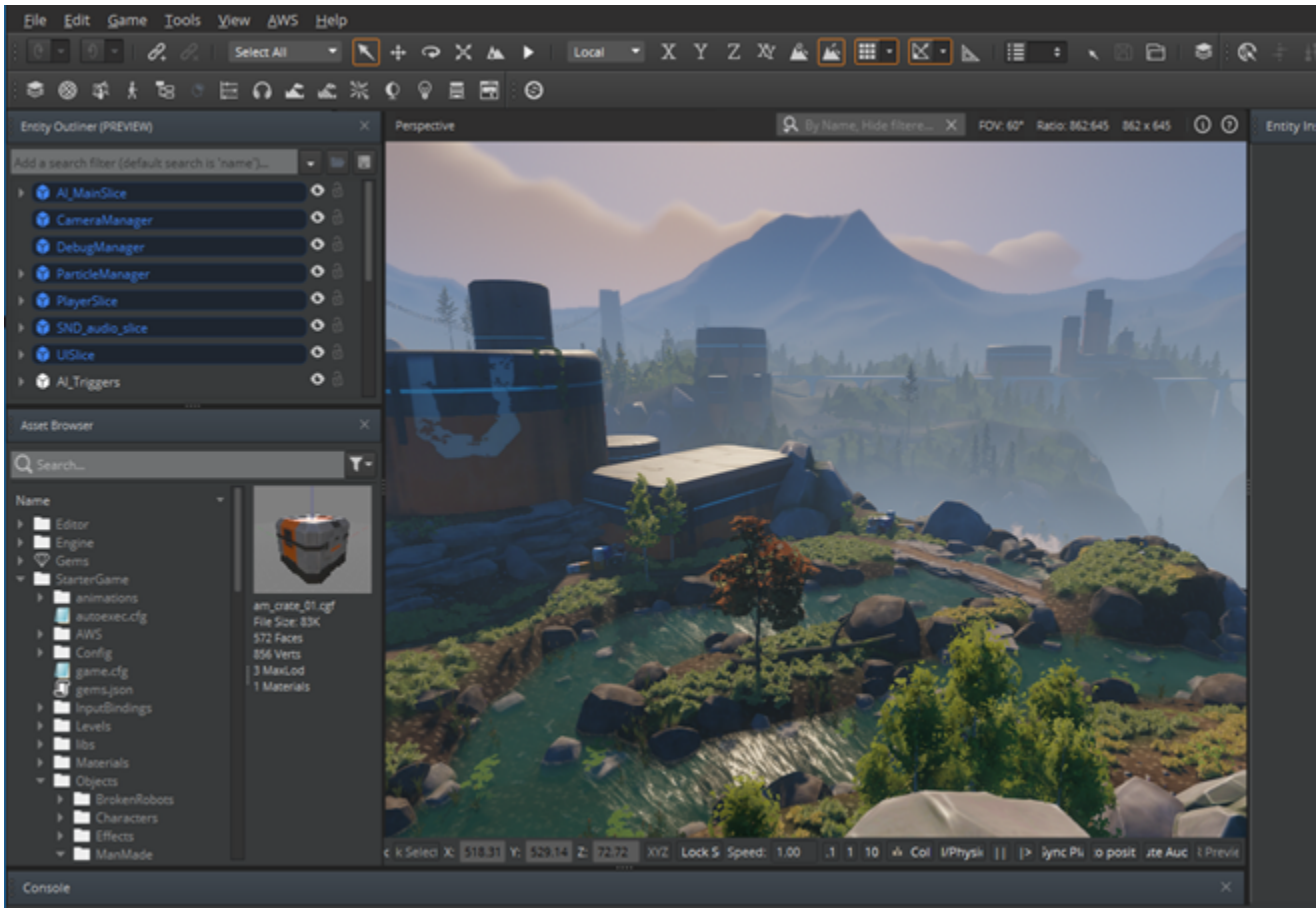
- If you are using Visual Studio 2013, go to the `\dev\Bin64vc120` directory.
- If you are using Visual Studio 2015, go to the `\dev\Bin64vc140` directory.

> **Note**
> When starting Lumberyard for the first time, if you encounter errors with the `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 opens the following:

- **Entity Outliner**
- **Entity Inspector**
- **File Browser**

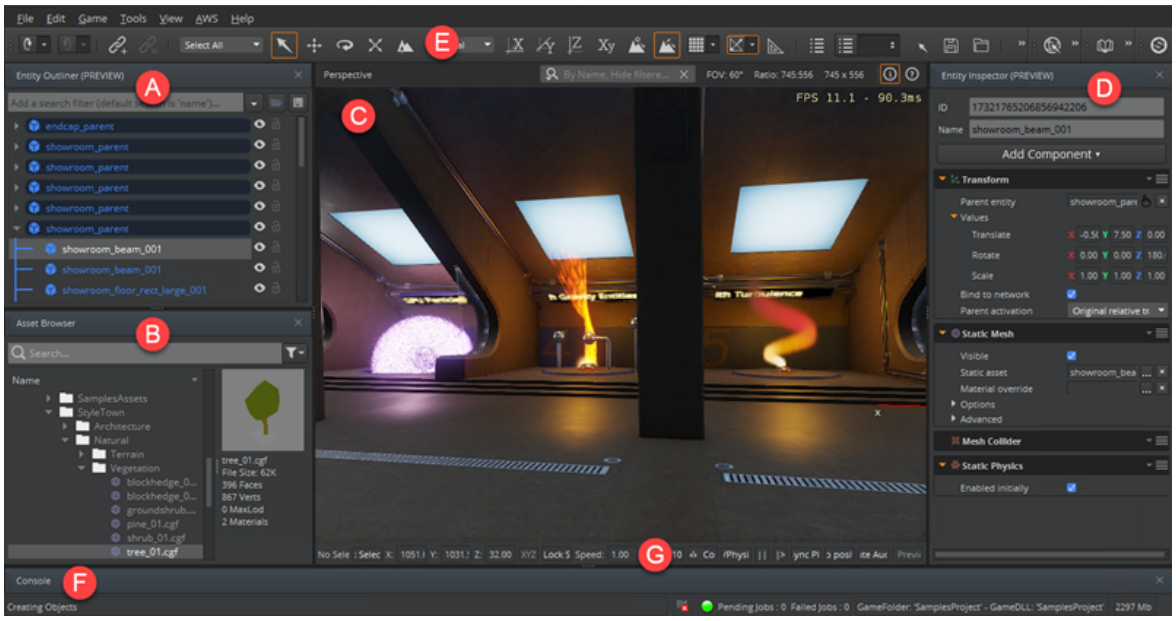
The **Rollup Bar** is available as a tab in the same panel as the **Entity Inspector**.



Topics

Lumberyard Editor Interface

Lumberyard Editor has the following panels and toolbars in the default layout:



A. **Entity Outliner** – The **Entity Outliner** lists the component entities that are included in the current level. For more information, see Entity Outliner (p. 437).

B. **Asset Browser** – The **Asset Browser** displays your assets in a tree view that mirrors your assets directory. For more information, see Asset Browser (Preview) (p. 179).

C. **Perspective** – The 3D viewport window displays the game environment and allows you to view, create, and interact with assets. For more information, see Using the Viewport (p. 85).

D. **Entity Inspector** – The right pane defaults to the **Entity Inspector** view, which allows you to add component and entities and modify their settings and properties. For more information, see Entity Inspector (p. 441).

E. **Top Toolbar** – The top toolbar provides quick access to the most commonly used functions and features. You can customize the toolbar. For more information, see Using the Top Toolbar (p. 79).

F. **Console** – The console window allows you to enter commands 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. You can access the console while in-game by pressing the tilde (**~**) key. For more information, see Using the Console Window (p. 93).

G. **Bottom Toolbar** – 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. 81).

Using the Menu Bar

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 use these features by clicking the toolbar buttons or using the keyboard. For more information, see Using Keyboard Shortcuts (p. 82).

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	Create a new level
Open	Open an existing level
Open Recent	Open a level from the list of recently opened levels
Save	Save the level that is currently open
Save as	Save the open level with a new name
Save Level Resources	Save all assets that are used in the open level
Save Level Statistics	Save statistics for the open level to an `.xml` file
Save Modified External Layers	Save only the external layers that have been modified since the last save
Configure Gems	Configure Gems and settings for your projects
Project Settings	Open menu options **Switch Projects**, **Configure Gems**, and **Input Mapping**
Show Log File	Show the log file that contains all text printed in the console
Exit	Close Lumberyard Editor; you will be asked to save any unsaved changes

Edit Menu

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

Menu Item	Description
Undo	Revert the last action
Redo	Apply the last action
Duplicate	Duplicate the selected object
Delete	Delete the selected object
Select All	Select all visible, non-frozen objects
Deselect All	Deselect the objects that are currently selected
Next Selection Mask	Select the next selection mask
Invert Selection	Invert the selection so that the unselected object is selected and the currently selected objects are deselected
Hide Selection	Hide the selected object

Menu Item	Description
Show Selection	Show the selected object
Show Last Hidden	Show the last hidden object
Unhide All	Unhide all hidden objects
Group	Group multiple selected objects and draw a green box around those objects
Modify	Perform actions on the selected objects: Link, Unlink, Align, Constrain, Snap, Transform Mode, Convert to, Fast Rotate, Sub Object Mode, Load Objects, and Update Procedural Vegetation
Lock Selection	Lock the selected object
Unlock All	Unlock all frozen objects
Rename Object(s)	Rename selected object(s)
Set Object(s) Height	Move the object to a specified height (in meters) above the terrain
Editor Settings	See the Editor Settings Menu (p. 72) section

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
Link	Create hierarchies between objects
Unlink	Remove the connection between linked objects
Align	Align an object to the grid, to another object, or to the selected surface, which moves the pivot point of the object
Constrain	Limit movement to the XYZ axes, XY planes, or to the surface of the terrain and objects
Snap	Snap an object to the grid or a rotational increment
Transform Mode	Select an object and apply one of the following: • **Select Mode** – Select an object • **Move** – Select and move an object • **Rotate** – Select and rotate an object • **Scale** – Select and scale an object • **Select Terrain** – Switch to terrain selection mode
Convert to	Convert the selected object to a brush, geometry entity, designer object, static entity, game volume, or component entity
Fast Rotate	Quickly rotate the selected object on the specified axis or with the degree value that you specify for **Rotate Angle**

Menu Item	Description
Sub Object Mode	Select and edit various geometry components, if an object is selected using the edit mesh function
Save Object(s)	Save the selected objects
Load Object(s)	Browse to and open objects from the game directory
Update Procedural Vegetation	Update all procedural vegetation

Editor Settings Menu

The **Editor Settings** menu allows you to modify global settings for the editor, switch between viewport quality settings, and customize keyboard settings.

Menu Item	Description
Global Preferences	Modify the global settings for Lumberyard Editor and its tools
Graphics Performance	Select your preferred display setting: • **PC - Very High** – Enable very high resolution display settings (some DX11 specific) • **PC - High** – Enable high resolution display settings • **PC - Medium** – Enable medium resolution display settings • **PC - Low** – Enable low resolution display settings • **Android** – Emulate the Android display settings • **iOS** – Emulate the Apple iOS display settings
Keyboard Customization	Configure toolbars, menus, and keyboard shortcuts; customize your keyboard settings, with the option to import or export saved custom keyboard settings

Game Menu

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

Menu Item	Description
Play Game	Switch to game mode (press **Esc** to exit game mode)
Enable Physics/AI	Enable physics and AI in your open level
Export to Engine	Export the level data to a `level.pak` file in order to play the level in game mode
Export Selected Objects	Save the selected geometry to an `.obj` or `.fbx`file
Export Occlusion Mesh	Export the occlusion mesh
Terrain Collision	Collide the camera with the terrain to prevent flying under the terrain surface

Menu Item	Description
Edit Equipment Packs	Open the **Edit Equipment Packs** window to add, delete, rename, or reorder the equipment for an actor
Toggle SP/MP GameRules	Toggle between single player and multiplayer game rules
Synchronize Player with Camera	Set the player position relative to the camera position
AI	See the AI Menu (p. 73) section
Audio	See the Audio Menu (p. 74) section
Clouds	See the Clouds Menu (p. 74) section
Physics	See the Physics Menu (p. 74) section
Prefabs	See the Prefabs Menu (p. 75) section
Terrain	See the Terrain Menu (p. 75) section
Debugging	See the Tools Menu (p. 77) section

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	Generate all AI navigation and perform the tasks below
Generate Triangulation	Generate triangulation of the navigation mesh that is used for outdoor levels
Generate 3D Navigation Volumes	Generate 3D navigation data for 3D volumes that are used by alien AI agents; volumes are defined by `AINavigationModifier` and a `Volume` navigation type
Generate Flight Navigation	Generate 2.5D navigation data for volumes used by flying AI agents; volumes are defined by `AINavigationModifier` and a `Flight` navigation type
Generate Waypoints	Generate links for indoor waypoints
Validate Navigation	Check the navigation data for various problems (for example, bad object placement, overlapping forbidden areas, or corruptions) and display warnings if any problems are found
Clear All Navigation	Remove all navigation information from the level
Generate Spawner Entity Code	Search for AI entity classes and generate an `.ent` file for each; associate an entity class name with the Lua base file for that entity
Generate 3D Debug Voxels	Generate debugging information for volume navigation regions when the `ai_DebugDraw` console variable is enabled
Create New Navigation Area	Create a navigation area

Menu Item	Description
Request a full MNM rebuild	Perform a full rebuild of all MNM mesh data
Show Navigation Areas	Display the MNM navigation areas
Add Navigation Seed	Add a navigation seed entity
Continuous Update	Continuously updates MNM data; if disabled, you must request a rebuild in order for the mesh data to update
Visualize Navigation Accessibility	Display inaccessible areas in red and accessible areas in blue
MediumSizedCharacters	Toggle the navigation debug display
Generate Cover Surfaces	Generate cover surface data
AIPoint Pick Link	Combine AI navigation modifier points
AIPoint Pick Impass Link	Restrict AI navigation modifier points to prevent AI from walking on the points

Audio Menu

The **Audio** menu includes commands to show the music that is currently playing and to access the sound and dialog browsers.

Menu Item	Description
Stop All Sounds	Silence all sounds in the level
Refresh Audio	Refresh 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	Create a new cloud asset
Destroy	Delete a custom cloud asset
Open	Open the selected cloud asset
Close	Close the selected cloud asset

Physics Menu

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

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

Prefabs Menu

The **Prefabs** 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)	Create a prefab from selected objects
Add Selected Object(s) to Prefab	Add the selected objects to the prefab
Clone Selected Object(s)	Clone the selected objects
Extract Selected Object(s)	Extract the selected objects from the prefab
Open All	Open all prefabs
Close All	Close all prefabs
Reload All	Reload 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
Generate Terrain Texture	Generate the terrain surface texture in a compressed format into the `terraintexture.pak` file; you must do this in order for changes made by the terrain painter to be visible in game mode
Generate Terrain	Open the **Terrain Editor** to generate terrain for the level
Edit Terrain	Open the **Terrain Editor** to modify the terrain settings
Export/Import Megaterrain Texture	Export or import the megaterrain texture
Export Terrain Block	Export a section of the terrain to a terrain block `.trb` file
Import Terrain Block	Import terrain from a saved `.trb` file
Resize Terrain	Open the **Terrain Resize** tool to modify the terrain size
Terrain Modify	Flatten or smooth the terrain

Menu Item	Description
Edit Vegetation	Open the **Vegetation** section on the **Terrain** tab in the **Rollup Bar** in order to modify the vegetation
Paint Layers	Open the **Layer Painter** section on the **Terrain** tab in the **Rollup Bar** in order to modify settings on the layer
Refine Terrain Texture Tiles	Divide the terrain tiles into smaller sections
Export Terrain Area	Export the selected terrain area to an `.obj` or `.fbx`file
Export Terrain Area with Objects	Export the selected terrain area and all objects within to an `.obj` or `.fbx`file

Debugging Menu

The **Debugging** menu allows you to reload scripts, textures, geometry, and terrain. Other debugging options include configuring user commands and checking the level for errors.

Menu Item	Description
Reload Scripts	Reload all scripts or scripts for actor, AI, entity, item, and UI
Reload Textures/Shaders	Reload all textures and shaders used in the level
Reload Geometry	Reload all geometry used in the level
Reload Terrain	Reload the selected terrain
Resolve Missing Objects/ Materials	Check the level and resolve all object and material issues
Enable File Change Monitoring	Monitor file changes
Check Object Positions	Check the position of all objects in the level
Clear Registry Data	Clear the registry data for all custom toolbars
Check Level for Errors	Check the level for errors—such as duplicate objects and missing assets—and display a list in the console window
Save Level Statistics	Save level statistics to the *`yourlevelname`*`.xml` file in the `TestResults` directory
Compile Script	Compile an entity script
Reduce Working Set	Reduce memory consumption
Update Procedural Vegetation	Update all procedural vegetation
Configure Toolbox Macros	Open the **Tools Configuration** window to create shortcuts for the console commands
Toolbox Macros	Display the shortcuts for the console and Lumberyard Editor commands that you created
Script Help	Open the **Script Help** window, which lists all commands, descriptions, and examples

Tools Menu

The **Tools** menu lists the Lumberyard Editor tools and plugins. For more information, see Lumberyard Editors and Tools (p. 7).

View Menu

The **View** menu allows you to customize Lumberyard Editor, including the layout and viewport options.

Menu Item	Description
Cycle Viewports	Change the viewport to the next view type
Center on Selection	Select an object to zoom to the boundaries of the object; you can then press the **Alt** key and use the mouse to pan around the object, which remains centered on the screen
Show Quick Access Bar	Display the quick access bar
Enter Full Screen Mode	Display the viewport in full screen mode
Layouts	Change the Lumberyard Editor layout: • **Component Entity Layout** – Display the **Entity Outliner**, **Entity Inspector**, and **File Browser** alongside the viewport • **Legacy Layout** – Display the **Rollup Bar** and viewport only • **Save Layout** – Save the current layout • **Restore Default Layout** – Restore the layout to the default (**Component Entity Layout**)
Viewport	See the Viewport Menu (p. 77) section
Refresh Style	Refresh the editor stylesheet

Viewport Menu

The **Viewport** 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
Top	Open an orthographic top view of your scene
Front	Open an orthographic front view of your scene
Left	Open an orthographic left view of your scene
Perspective	Open a perspective view of your scene
Map	Open a map view of your scene
Wireframe	Use the wireframe rendering view
Ruler	Use the ruler tool to measure distance

Menu Item	Description
Grid Settings	Set grid line spacing, angle snapping, and rotation and translation settings
Configure Layout	Select a preconfigured layout
Goto Coordinates	Specify the camera position in XYZ coordinates, and move the camera to that position
Goto Selection	Go to the currently selected object in the viewport
Goto Location	Go to one of 10 predefined locations in the viewport
Remember Location	Save up to 10 locations in the viewport
Change Move Speed	Change the movement speed for all objects in the level
Switch Camera	Change the camera for the viewport: • **Default Camera** – Select the default camera • **Sequence Camera** – Select the camera that is used in a Track View sequence • **Selected Camera Object** – Select the camera entity • **Cycle Camera** – Select the next camera
Show/Hide Helpers	Show or hide all helper objects

AWS Menu

The **AWS** menu allows you to sign up for an Amazon Web Services (AWS) account, access AWS services using Cloud Canvas and Amazon GameLift, and open the **Cloud Gem Portal**.

Menu Item	Description
Credentials manager	Add or edit an AWS profile
Cloud Canvas	Select a deployment or access the Cloud Canvas Resource Manager or Dynamic Content Manager
Commerce	Learn how to submit your game to Amazon's Digital Software store using Merch by Amazon or Publishing on Amazon
GameLift	Access the Amazon GameLift console or learn more about Amazon GameLift
Open AWS Console	Open the AWS Management Console and access Amazon Cognito, Amazon DynamoDB, Amazon S3, and Amazon Lambda
Open Cloud Gem Portal	Open the Cloud Gem Portal

Help Menu

The **Help** menu includes web links to the technical documentation and tutorials, support contact information, and Lumberyard Editor version information.

Menu Item	Description
Getting Started	Open the *Lumberyard Getting Started Guide*
Tutorials	Open the **Amazon GameDev Tutorials** website to view written or video tutorials
Documentation	Open the Lumberyard glossary, release notes, Lumberyard documentation landing page, or the Amazon GameLift documentation landing page
GameDev Resources	Access the GameDev blog, Twitch channel, forums, and AWS support
Give Us Feedback	Send feedback to the Lumberyard feedback email alias
About Lumberyard	View copyright, build, and version information for Lumberyard Editor

Using the Top Toolbar

Lumberyard Editor provides a toolbar that allows you to easily access various editor tools and features. The toolbar is docked at the top of the editor by default, but you can also dock it vertically on the edges of the editor or undock it from the editor. To customize the toolbar, right-click anywhere on the toolbar and select **Customize** from the context menu. You can choose which toolbars, views, or modes to include. You can also add commands to a toolbar.

You can access Lumberyard tools and features using one of the following methods:

- Toolbar buttons
- Tools menu
- Keyboard shortcuts (p. 82)

For information about the bottom toolbar, see Using the Bottom Toolbar (p. 81).

EditMode Toolbar

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

- **A** – Revert or apply the last command
- **B** – Link or unlink the selected object
- **C** – Filter what you can select in the viewport: all, brushes, no brushes, entities, prefabs, areas, shapes, AI points, decals, solids, or no solids
- **D** – Use the translation tools to select, move, rotate, or scale an object or object type; and select or rotate a terrain area
- **E** – Select the reference coordinate system
- **F** – Specify the axis constraint by locking on the x-, y-, or z-axis or xy-plane
- **G** – Use the object placement tools to follow the terrain, snap to objects, snap to grid, snap to angle, or show the ruler

- **H** – Use the asset organization tools to open the object selector, create a selection, delete a selection, save selected objects, or load selected objects
- **I** – Select the current layer

Object Toolbar

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

- **A** – Go to the selected object
- **B** – Align the selection to an object by choosing the source object, clicking the tool, and then clicking the target object
- **C** – Align the object to the grid
- **D** – Set the object's height
- **E** – Align the object to the terrain surface normal (press and hold **Ctrl** for object surface normal alignment)
- **F** – Freeze or unfreeze the selected object
- **G** – Apply vertex snapping for the selected object
- **H** – Reset or get the physics state for the selected object or simulate physics on the selected object

Editors Toolbar

The **Editors** toolbar allows you to access various editor tools:

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

Substance Toolbar

The **Substance** toolbar includes a button that opens the **Substance Editor**.

Using the Bottom Toolbar

Lumberyard Editor includes a bottom toolbar that provides status as well as the features below.

Status

The **status** bar (A) displays the number of selected object(s) and provides functional hints for buttons or menu items in Lumberyard Editor.

Lock Selection

The **Lock Selection** button (B) 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 (C) 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 (D) 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 (E) 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 (F) toggles terrain collision. You can enable terrain collision to inhibit camera movement below the terrain surface.

AI/Physics

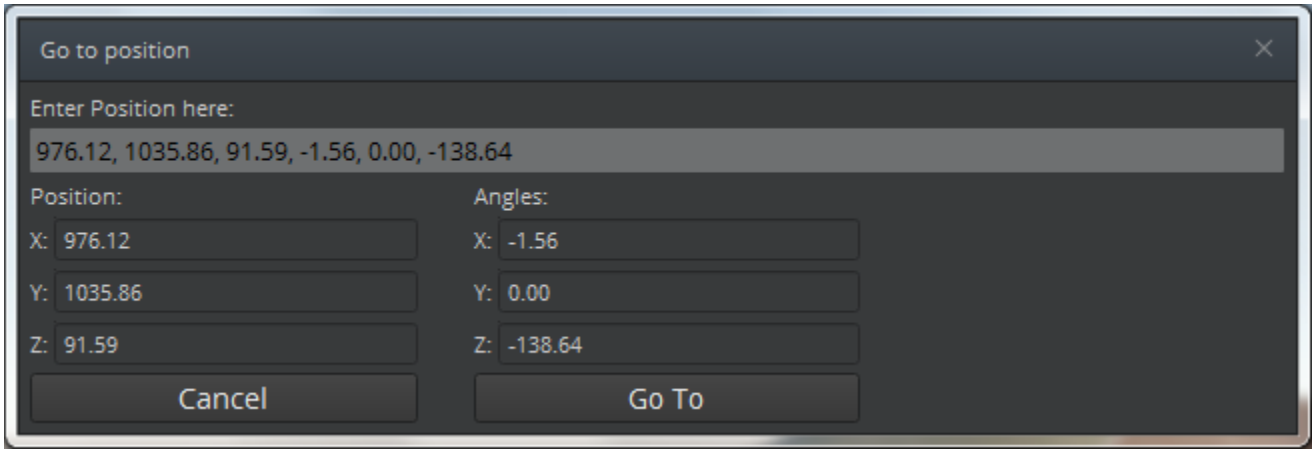
The **AI/Physics** button (G) toggles physics simulation and AI, allowing you to test physics and AI behavior directly in the editor without entering game mode.

No Sync Player

The **No Sync Player** button (H) 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 **Goto Position** button (I) 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 (J) mutes audio and all sounds in the level.

VR Preview

The **VR Preview** button (K) previews your game project in virtual reality mode (p. 1794) when a virtual reality (p. 1782) gem is enabled.

Using Keyboard Shortcuts

Lumberyard supports the following keyboard shortcuts.

File Menu

Keyboard shortcut	Function
Ctrl+O	Open a level
Ctrl+S	Save the level
Ctrl+Alt+F	Restore the saved state
Ctrl+Alt+H	Save the current state
Ctrl+Shift+L	Load objects from the game directory
Ctrl+Shift+S	Save the selected object

Edit Menu

Keyboard shortcut	Function
G	Toggle snap-to-grid
Ctrl+D	Duplicate the selected object
Ctrl+Z	Undo the last operation
Ctrl+Shift+Z	Redo the last operation
Ctrl+Shift+Space	Lock the selection

Game Menu

Keyboard shortcut	Function
Ctrl+E	Export the level
Ctrl+G	Enter game mode (Esc to exit)
Ctrl+P	Enable AI or physics

Tools Menu

Keyboard shortcut	Function
M	Open the Material Editor
~	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

Select Toolbar

Keyboard shortcut	Function
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

Hide Toolbar

Keyboard shortcut	Function
F	Freeze the selected object
H	Hide the selected object
Ctrl+F	Unfreeze all objects
Ctrl+H	Unhide all hidden objects

Constraint Toolbar

Keyboard shortcut	Function
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

Perspective Viewport

Keyboard shortcut	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
Q	Toggle camera or terrain collision
Z	Focus the camera to the selected object in the viewport
F3	Toggle the wireframe view
Alt+middle mouse button	Rotate around the selected object

Keyboard shortcut	Function
Alt+right click	Zoom in and out
Ctrl+Tab	Cycle the viewport perspective
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
Shift+Spacebar	Show or hide helpers

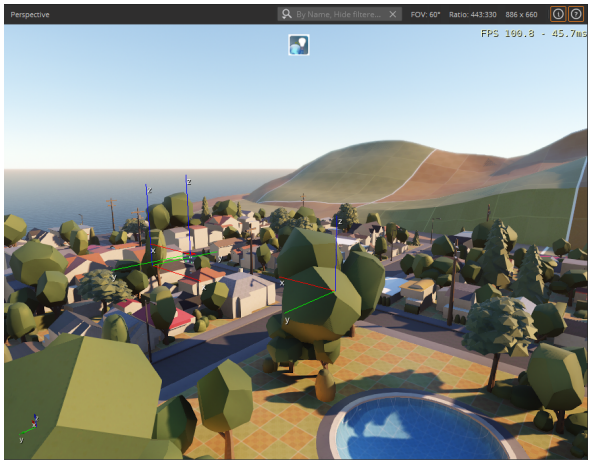
Using the Viewport

The viewport window (called **Perspective** in Lumberyard Editor) displays the scene that is rendered by the engine. The majority of level design occurs in the viewport, including 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.

The **Perspective** header includes a search box, field of view (FOV), screen ratio information, and options to show or hide debug information.

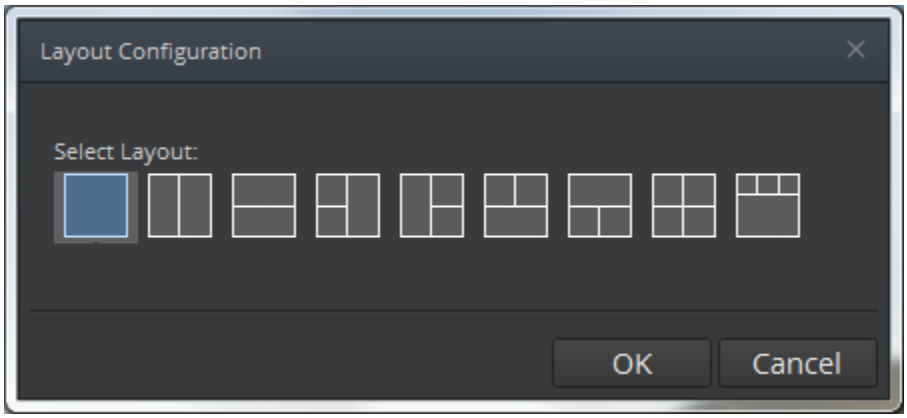
Topics

- Changing the Field of View (FOV) (p. 86)
- Changing the Perspective (p. 86)
- Changing the Camera View (p. 87)



To customize the viewport

1. In Lumberyard Editor, in the viewport title bar, right-click **Perspective** and choose **Configure Layout**.
2. In the **Layout Configuration** dialog box, select your preferred layout.



3. Click **OK** to save your changes.

Changing the Field of View (FOV)

You can change the default camera's FOV for the Lumberyard Editor viewport or the FOV for your game's camera. Follow the steps to change the default camera's FOV. To change the FOV for your game's camera, see Camera (p. 467).

To change the default camera's FOV

1. In Lumberyard Editor, choose **Edit**, **Editor Settings**, **Global Preferences**.
2. In the **Preferences** window, under **Viewports**, click **General**.
3. In the right pane, under **General Viewport Settings**, for **Perspective View FOV**, set the desired value in degrees.
4. Click **OK** to save your changes.

Changing the Perspective

You can change the perspective of the viewport to use a top, front, left, perspective, or map view.

To change the perspective of the viewport

In Lumberyard Editor, click **View**, **Viewport**, and choose one of the following views:

- **Top** – A top-down view of your level, including bounding boxes and line-based helpers. Terrain geometry is not shown.
- **Front** – A front view of your level, including bounding boxes and line-based helpers. Terrain geometry is not shown.
- **Left** – A view of your level from the left, including bounding boxes and line-based helpers. Terrain geometry is not shown.
- **Perspective** – A view of your level that uses the default camera perspective, showing all level content. This is the default and most commonly used view.
- **Map** – An overhead map of the level, including terrain, texture, and helper information.

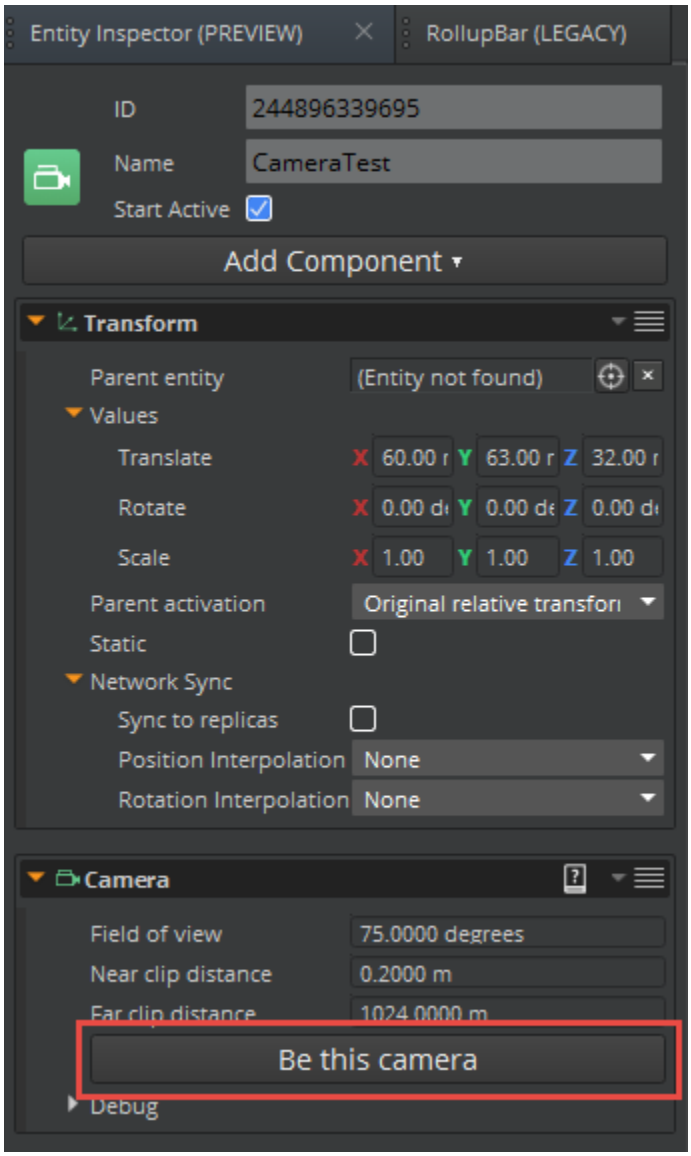
Changing the Camera View

You can use the **Viewport Camera Selector** to quickly position and orient a camera in your game. You can choose between all in-game cameras and the editor camera. When you possess the camera, you can use the editor controls to manipulate the camera.

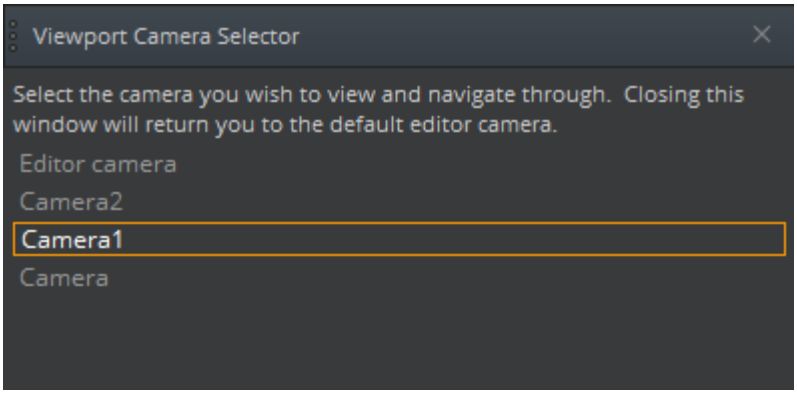
To possess the camera and move around

1. In Lumberyard Editor, in the **Perspective** viewport, select an entity with a Camera component or create one if it doesn't exist.
2. In the **Entity Inspector**, under **Camera**, choose **Be this camera**. This allows the editor to use the selected camera as its view.

   Note
   After choosing **Be this camera**, moving the view in the editor changes the transform for the camera entity. As you look around in the viewport while possessing a camera, the corresponding position and orientation for the transform is updated accordingly.



3. Do any of the following:
   - Change to the default view: In the **Entity Inspector**, under **Camera**, choose **Return to default editor camera**.
   - Choose another camera to possess in the **Viewport Camera Selector**.



   - Disable the **Viewport Camera Selector** controls by entering game mode (**Ctrl+G**).

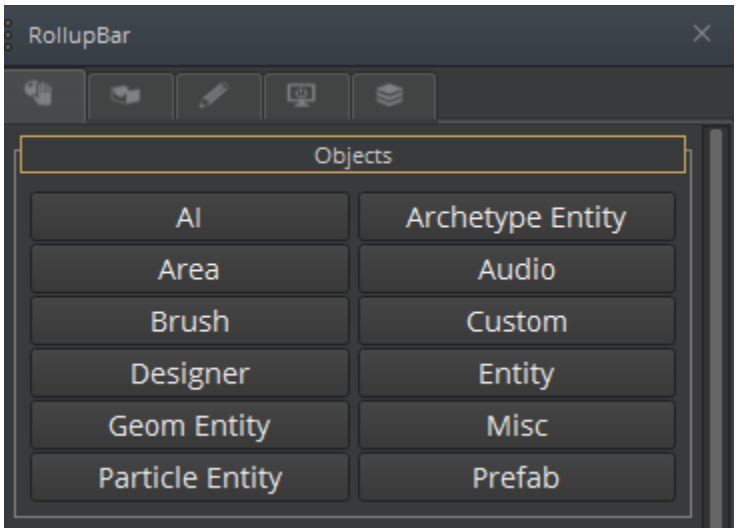
Using the Rollup Bar

The Rollup Bar is on a tab in the right pane of Lumberyard Editor and provides access to display options, profile tools, layer controls, objects, terrain tools, and modeling tools. The Rollup Bar includes the following tabs: **Objects**, **Terrain**, **Modeling**, **Render/Debug**, and **Layers**.

Topics

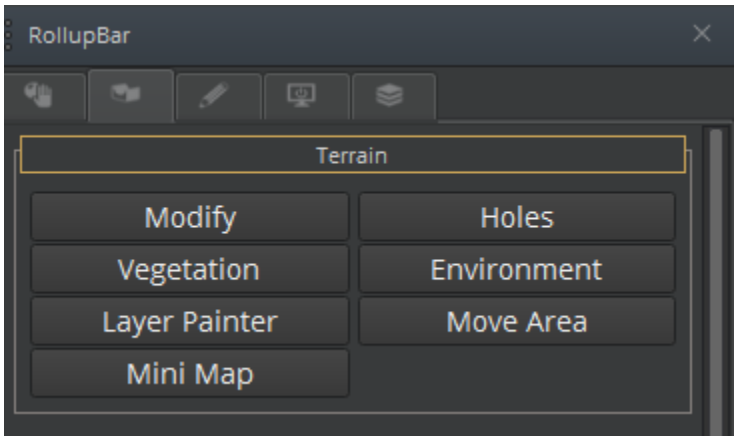
Objects Tab

The **Objects** tab includes the majority of objects and entities, and the interfaces to various local object and brush databases. This tab includes the following buttons: **AI**, **Archetype Entity**, **Area**, **Audio**, **Brush**, **Custom**, **Designer**, **Entity**, **Geom Entity**, **Misc**, **Particle Entity**, and **Prefab**.



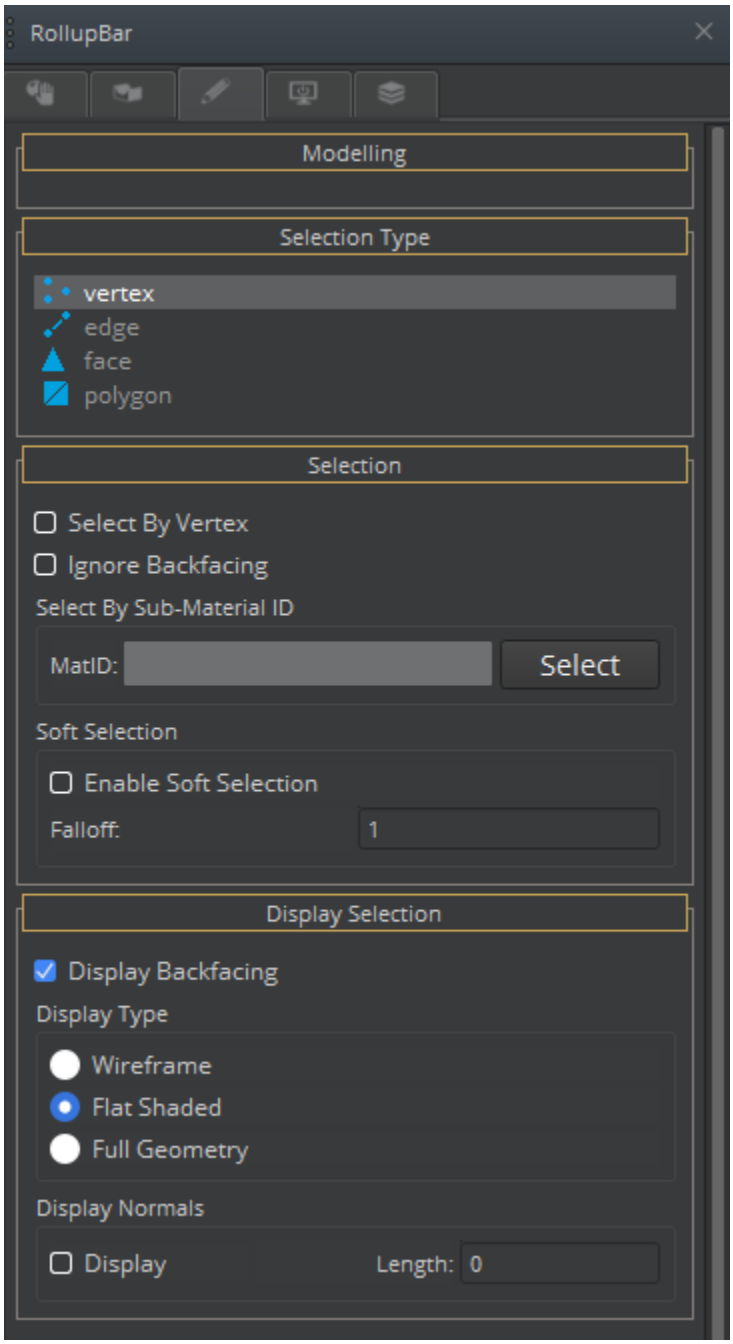
Terrain Tab

The **Terrain** tab includes tools to modify terrain and vegetation. This tab includes the following buttons: **Modify**, **Holes**, **Vegetation**, **Environment**, **Layer Painter**, **Move Area**, and **Mini Map**.



Modeling Tab

The **Modeling** tab includes tools to modify display options and parameters for the selection. This tab includes the following sections: **Selection Type**, **Selection**, **Display Selection**.

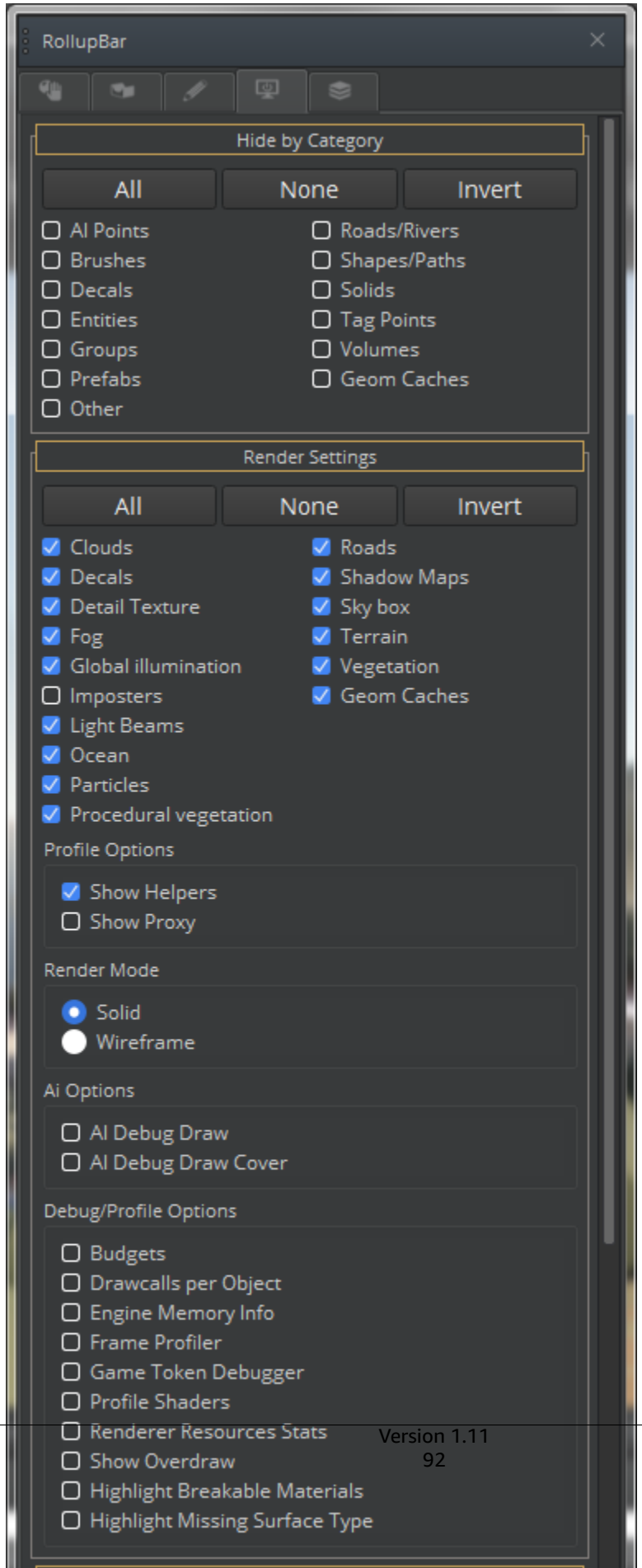


Render/Debug Tab

The **Render/Debug** tab includes 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.

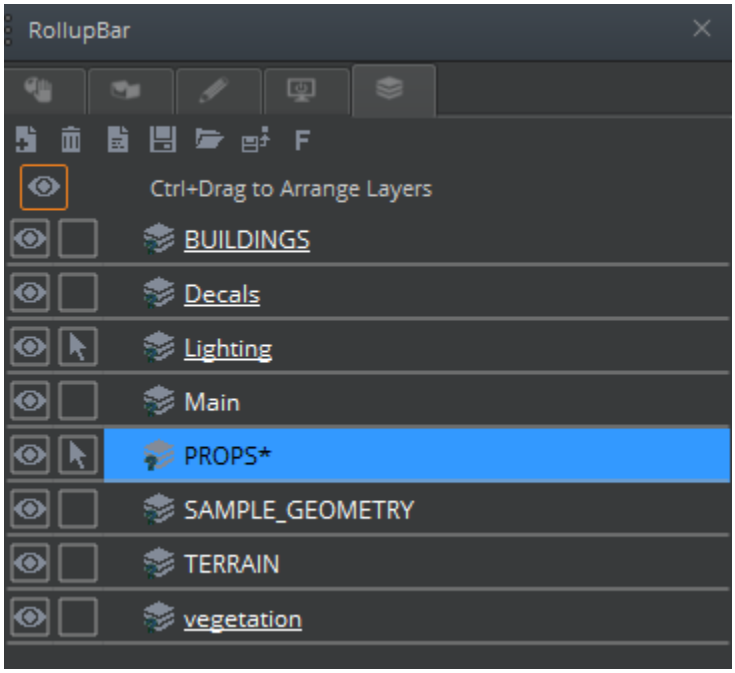
This tab includes the following sections and subpanels: **Hide by Category**, **Render Settings**, and **Stereo Settings**.

Note
You must enable (or disable) the **Shadow Maps** setting before you can enable (or disable) the **Global Illumination** setting. These two settings work together.


RollupBar
Hide by Category
All
None
Invert
AI Points
Brushes
Decals
Entities
Groups
Prefabs
Other
Roads/Rivers
Shapes/Paths
Solids
Tag Points
Volumes
Geom Caches
Render Settings
All
None
Invert
Clouds
Decals
Detail Texture
Fog
Global illumination
Imposters
Light Beams
Ocean
Particles
Procedural vegetation
Roads
Shadow Maps
Sky box
Terrain
Vegetation
Geom Caches
Profile Options
Show Helpers
Show Proxy
Render Mode
Solid
Wireframe
Ai Options
AI Debug Draw
AI Debug Draw Cover
Debug/Profile Options
Budgets
Drawcalls per Object
Engine Memory Info
Frame Profiler
Game Token Debugger
Profile Shaders
Renderer Resources Stats
Show Overdraw
Highlight Breakable Materials
Highlight Missing Surface Type

Layers Tab

The **Layers** tab includes the tools to create and manage level layers. This tab includes the following buttons: **New**, **Delete**, **Rename**, **Export**, **Import**, **Save External Layers**, and **Freeze**.



Using the Console Window

In Lumberyard Editor, the console window shows a running list of all editor commands, processes, and output. For example, when you delete an entity, the console shows the action and the output.

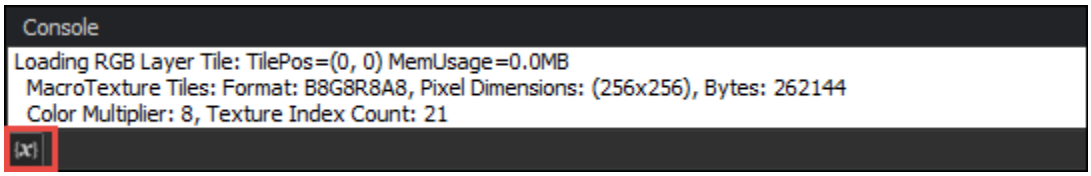
Topics

Viewing the Console

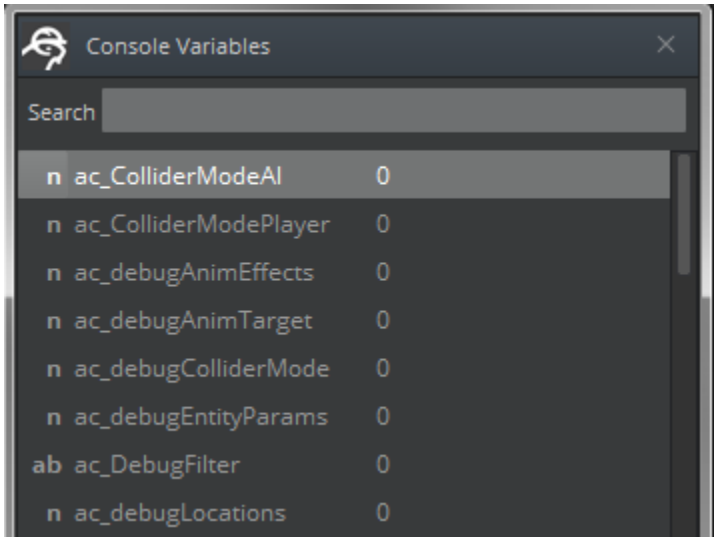
With the console, you can also type commands and view and edit console variables (CVARs). Console variables are a type of variable that you can manipulate in Lumberyard Editor.

To view the console window

1. Choose **Tools** and then choose **Console**.
2. Click the **X** icon in the left corner to open the **Console Variables** window.



3. The **Console Variables** window lists all available console variables. You can search for a specific console variable. To learn more about the variable, pause on the name.



4. To export a list of all console commands and variables, in the **Console** window, type: **DumpCommandsVars**. This creates the `consolecommandandvars.txt` file and is saved to the *`lumberyard_version`*`\dev\` directory.

Configuring Console Variables

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

- Configuration files:
  - The `game.cfg` file in your project directory
  - The *`lumberyard_version`*`\dev\system_`*`gamesystem`*`.cfg` file for your game system
  - The *`lumberyard_version`*`\dev\engine\config\user.cfg` file
  - The `level.cfg` file in your project's level directory
- Code
- Flow graphs
- Console variables typed directly into the console

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

Configuring Console Variables in the Console

You can specify values for console variables in the console to apply changes to your level.

To configure console variables in the console

1. In the **Console Variables** window, search for the variable name.
2. Double-click the value and then type the value that you want.
3. Close the **Console Variables** window.

Configuring Console Variables in Configuration Files

You can specify values for console variables in the configuration files, such as the level configuration file (`level.cfg`).

To configure console variables with a configuration file

1. Go to the directory that has the configuration file. For example, if you want to configure the `level.cfg` file, go to the *`lumberyard_version`*`\dev\`*`ProjectName`*`\Levels\`*`level_name`* directory.
2. Use a text editor to edit the file or to create one.
3. Specify the console variable name and the value. For example: `r_DisplayInfo=1` shows debugging information in the viewport.

```
r_DipslayInfo=1
```

4. Save the file.

Customizing Lumberyard Editor

You can customize your workspace by adjusting how the windows and tools are docked, customizing which toolbars and menus display, and updating the global editor settings. To change the size setting for the toolbar icon, edit the `ed_toolbarIconSize` parameter in the `Editor.cfg` file. By default, the toolbar icon size is set to `0` (32 pixels).

Topics

Docking Windows and Toolbars

When you drag a window or toolbar over an interface element or the edges of the editor, docking targets appear to show you where you can dock them. These targets appear for the top, bottom, left, and right quadrants of the pane.

To dock and undock windows and toolbars

Do any of the following:

- To split a row or column, drop the window or toolbar on a docking target.
- To dock a window or toolbar as a tab, drop it on a docking target in the middle of a pane.
- To undock a window or toolbar, drag the title bar and move the selection window away. Avoid the docking targets to prevent from accidentally redocking the window. To help prevent accidental

docking, a brief delay occurs before a docking target becomes active. You can also undock a window by right-clicking the title bar and choosing **Undock**.

- To prevent a window from docking, press and hold **Ctrl** while moving the window.

You can dock windows relative to any open pane, whether it is already docked, floating as a tab, or split in a column or row.

To snap windows

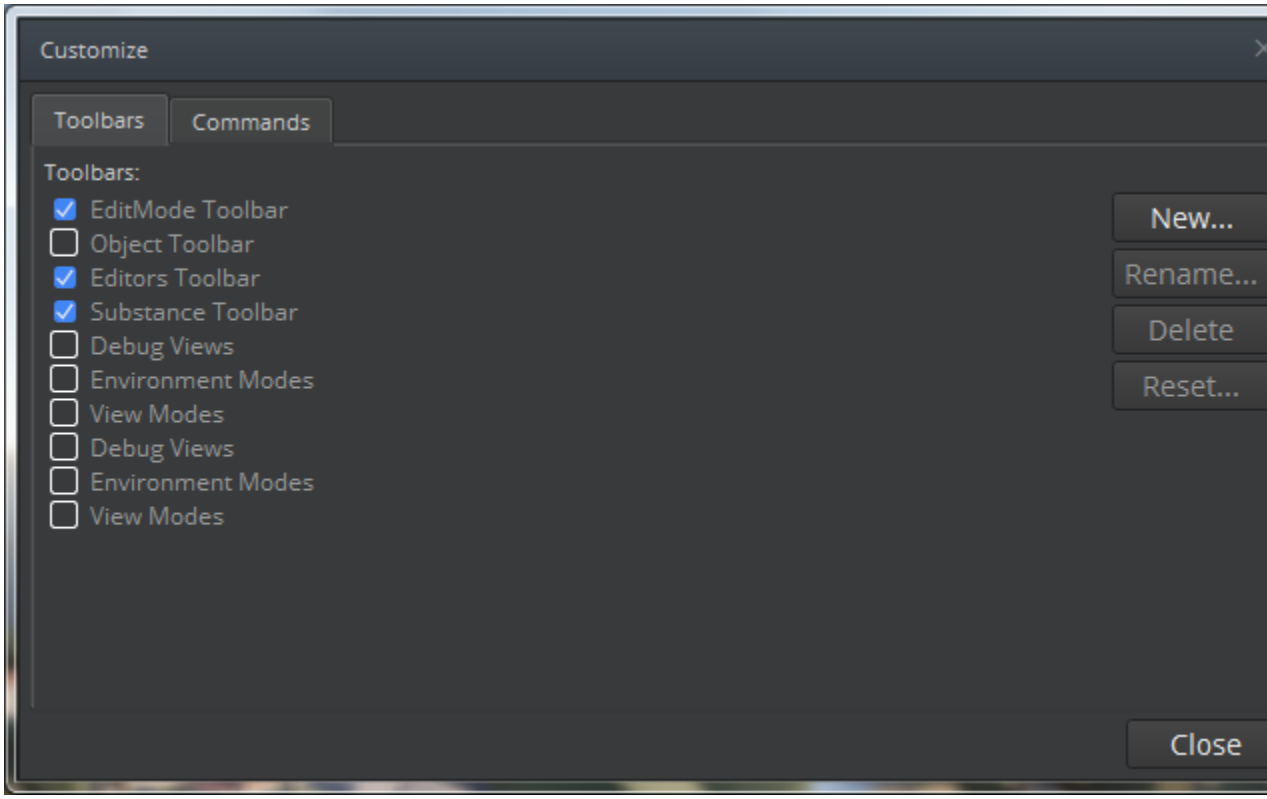
To snap a window in place, move the window close to a stationary window. Snapping works on the top, bottom, left, and right borders of the pane.

Customizing Toolbars and Menus

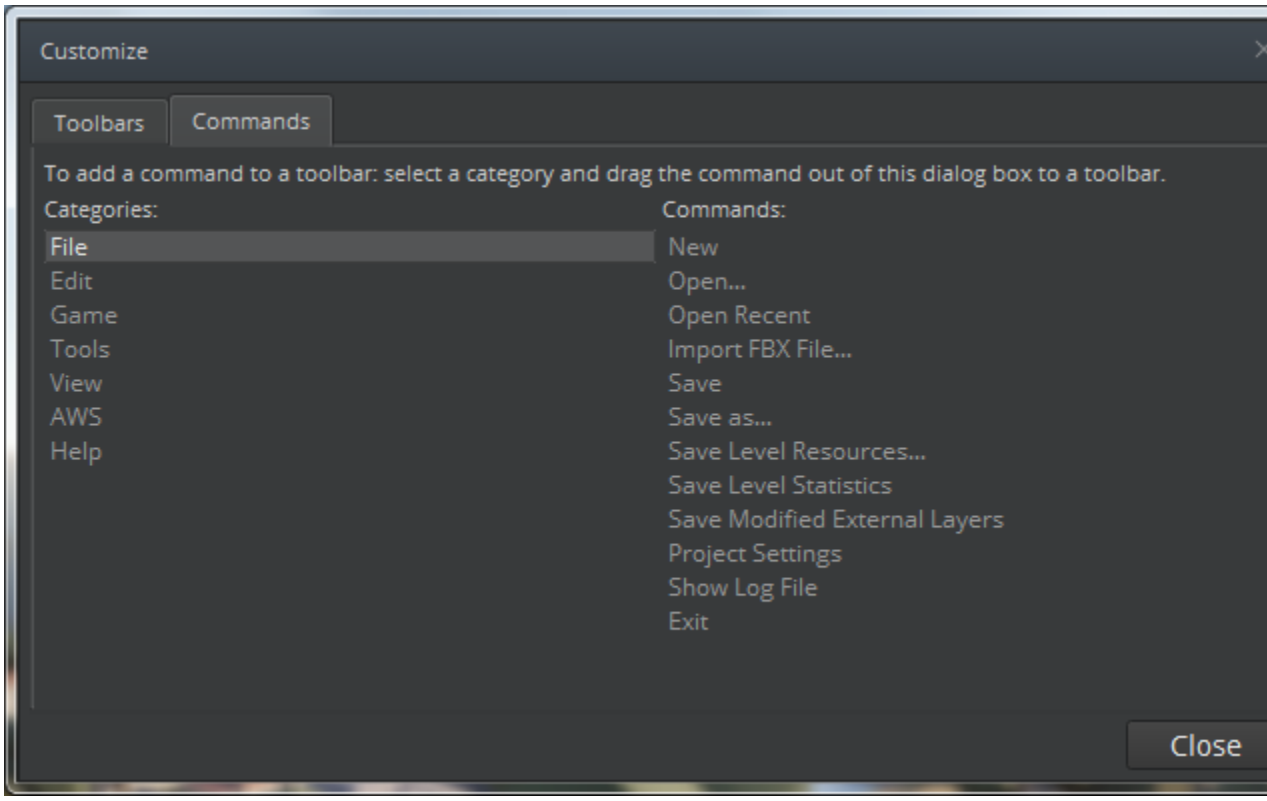
You can also give toolbars and menus your own personal touch.

To customize preset toolbars or create custom toolbars and menus

1. Right-click the top toolbar and click **Customize**.
2. In the **Customize** window, use the **Toolbars** tab to create, rename, and delete any custom toolbars and menus, and reset them to the default settings.



3. Use the **Commands** tab to drag and drop menu commands to any menu category.

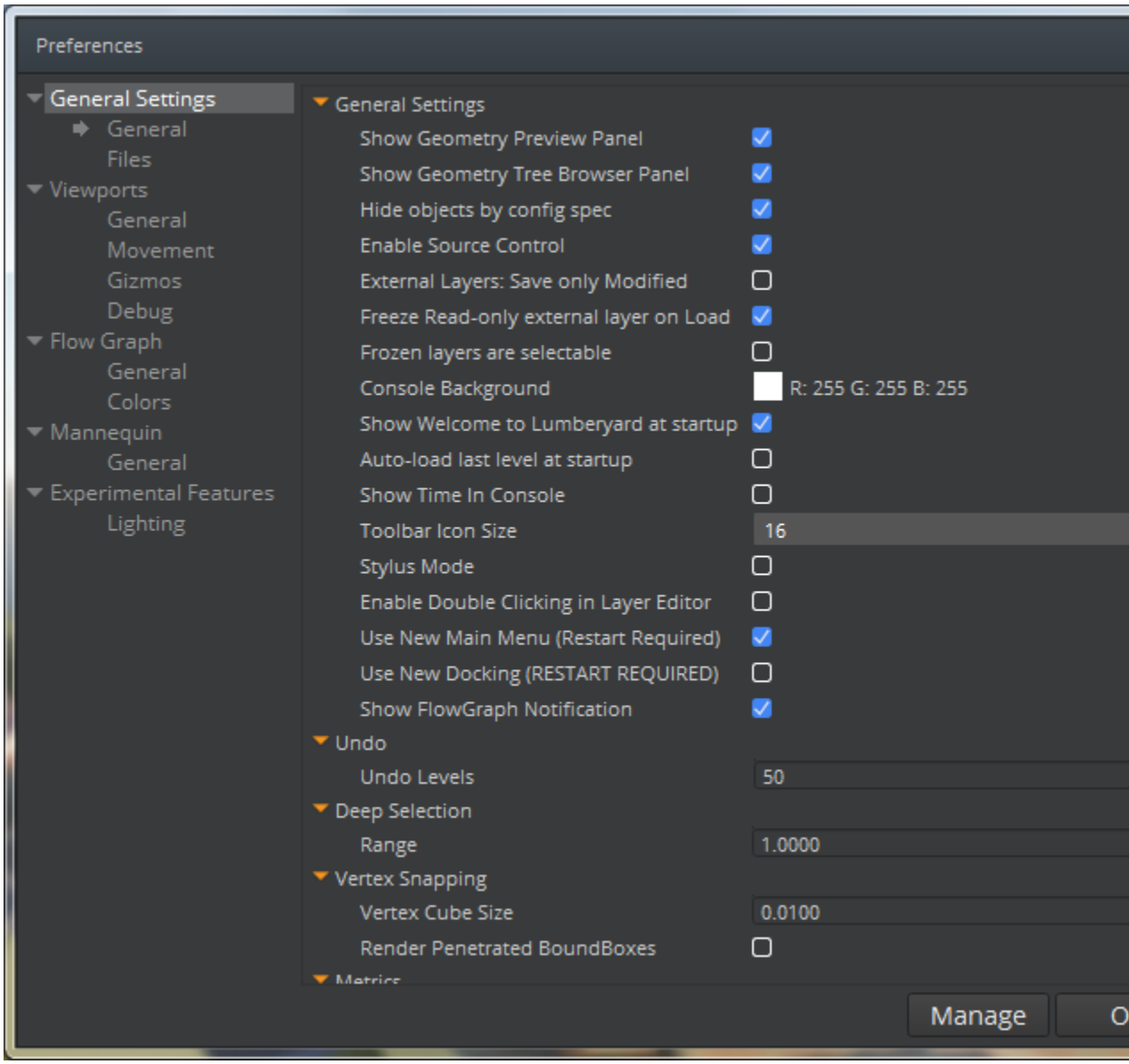


Changing Preferences

You can change the default settings to customize the look and functionality of the editor.

To customize the look and features of Lumberyard Editor

- In Lumberyard Editor, choose **Edit**, **Editor Settings**, **Global Preferences**.



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

Parameter	Description
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 at Startup	Display the **Welcome to Lumberyard** dialog box at startup
Autoload Last Level at Startup	Load the level that was last loaded
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 Boundboxes	Render penetrated bound boxes

Files 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 default temporary directory to use; default = `[root]\Temp`
Auto Save 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 blend spaces
Texture Editor	Specify the program to use for textures
Animation Editor	Specify the program to use for animations

Parameter	Description
Enable	Enable auto backup
Time Interval	Specify the frequency of auto backup (in minutes)
Maximum Backups	Specify the maximum number of auto backups
Remind Time	Specify the frequency of auto backup 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 pointer 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 bounding box around each object
Always Draw Entity Labels	Display entity names
Always Show Trigger Bounds	Display the trigger boundary helpers

Parameter	Description
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
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 highlight 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 highlight alpha to add to the geometry
Child Geometry Highlight Alpha	Specify the amount of highlight 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 movement while you control the viewport camera
Fast Movement Scale	Specify the multiplier for the camera speed; for example, a value of two doubles the movement speed of the camera
Wheel Zoom Speed	Specify the speed of the camera zoom when using the mouse wheel
Invert Y Axis	Invert the direction that the camera moves on the y-axis when holding the right button on the mouse and moving the mouse up or down

Parameter	Description
Invert Pan	Invert the direction that the camera moves when holding the middle button on the mouse and moving the mouse left or right

Gizmos 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
Ruler Sphere Transparency	Specify the transparency level of the locator sphere when using the **Ruler** tool

Debug Settings

Parameter	Description
Show Mesh Statistics	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 the Flow Graph editor.

General Settings

Parameter	Description
Automatic Migration	Update and reconnect port connection changes
Show NodeIDs	Display an identifier for each node
Show Tooltip	Display a tooltip for each node on pause over
Edges on Top of Nodes	Enable edges on top of nodes

Parameter	Description
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
- 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
Size of Tracks	Specify the height of the tracks for the dope sheet; minimum = 14, maximum = 32
Ctrl to Snap Scrubbing	Snap scrubbing by pressing and holding the **Ctrl** key
Timeline Wheel Zoom Speed	Specify the speed of the mouse wheel when zooming on the Mannequin timeline

Experimental Features

You can change the default settings for experimental features such as total illumination.

Lighting Settings

Parameter	Description
Total Illumination	Enable the total illumination lighting feature

Restoring the Default Layout for Lumberyard Editor

If you customized your workspace, you can reset Lumberyard Editor to use the default layout. This restores your editors, tools, and windows to the default view that you see when you open Lumberyard Editor for the first time. You can restore the default layout at any time.

Contents

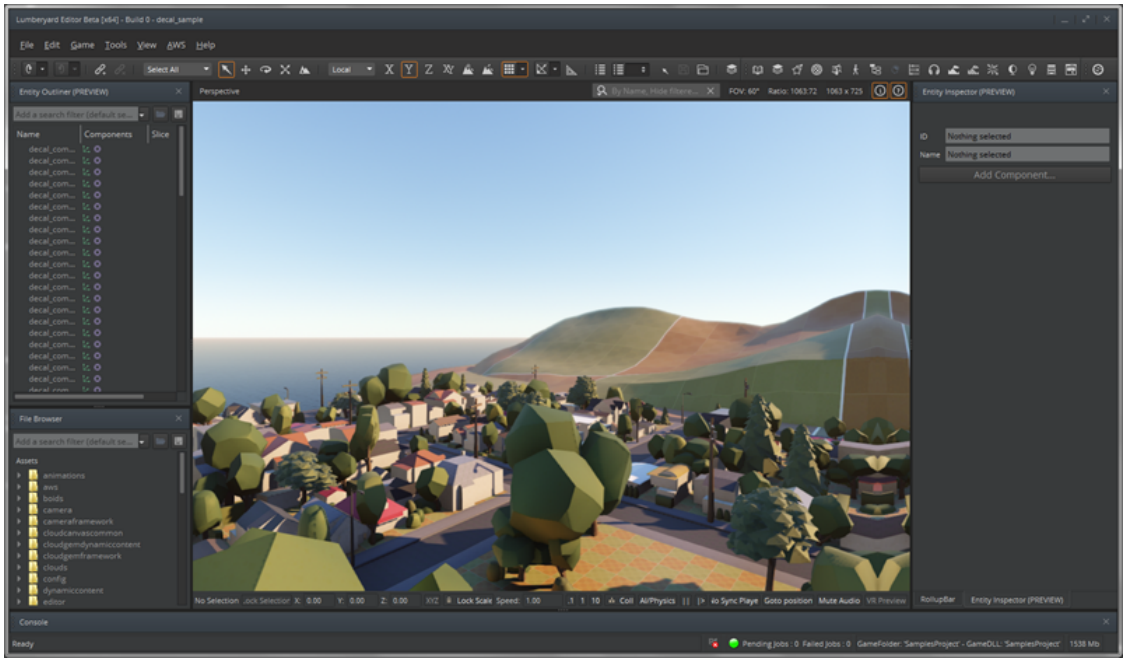
Restoring the Component Entity Layout

The component entity layout is the default view for Lumberyard Editor. If you customize your settings and tools, you can restore the default layout for Lumberyard Editor.

To use the component entity layout (default)

- In Lumberyard Editor, do one of the following:
  - Click **View**, **Layouts**, **Component Entity Layout**.
  - Click **View**, **Layouts**, **Restore Default Layout**.

The following example shows the component entity layout in Lumberyard Editor:



Note
You can enable the CryEntity Removal gem to disable all legacy features from Lumberyard Editor. Legacy features will eventually be removed. If you enable this gem, the legacy layout

view is removed from Lumberyard Editor. For more information, see CryEntity Removal Gem (p. 1076).

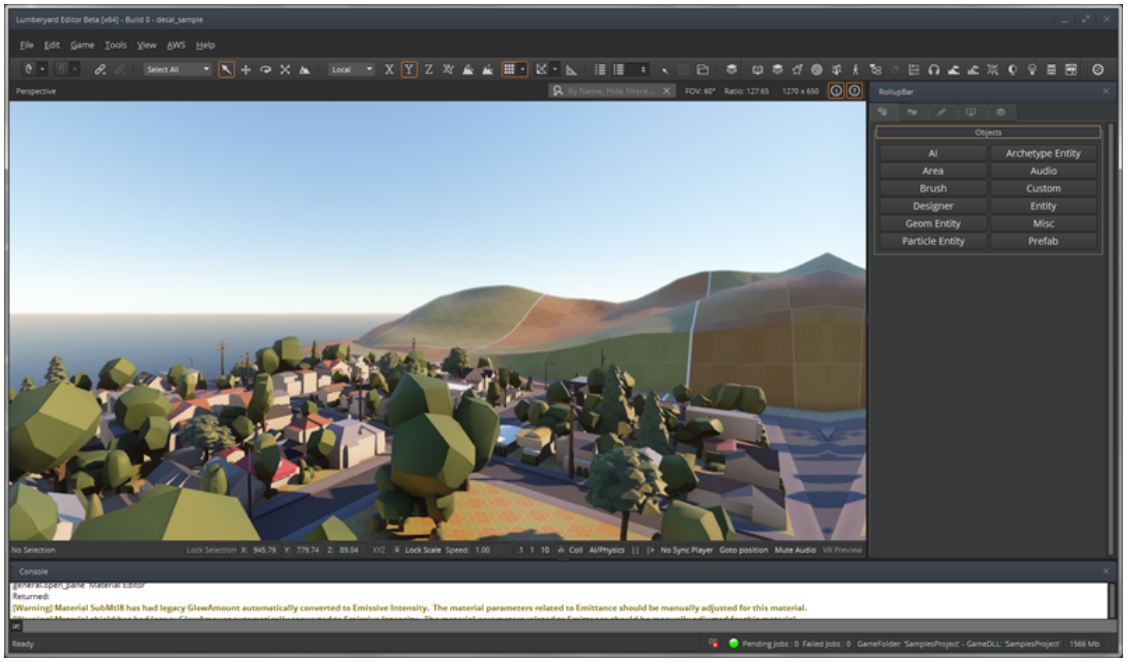
Restoring the Legacy Layout

If you don't enable the CryEntity Removal gem, you can switch back to the legacy layout.

To use the legacy layout

- In Lumberyard Editor, click **View**, **Layouts**, **Legacy Layout**.

  The following example shows the legacy layout in Lumberyard Editor:



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 can 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**, go to `HKEY_CURRENT_USER\Software\Amazon\Lumberyard\Editor`.
3. Right-click the `fancyWindowLayouts` and `mainWindowLayouts` folder icons and choose **Delete**. This deletes custom layout settings that you have for Lumberyard Editor.

   The default settings are restored the next time you start 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. 634).

For information on AI Navigation, see Navigation (p. 523).

For information on AI Behavior Tree, see Behavior Tree (p. 465).

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

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 **Tools**, **Other**, **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 **Tools**, **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

- Debugging AI Navigation (p. 114)

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 **Tools**, **Other**, **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 **Tools**, **Flow Graph**.

Some of 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.

AI:RayCastMNM – Performs a raycast to the AI multilayer nav mesh relative to an entity.

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

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

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

For more information, see Flow Graph Node Reference (p. 767).

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 **Tools**, **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

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 **Tools**, **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 (**...**) icon.
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. 1506).
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_MNMProfileMemory

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

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.

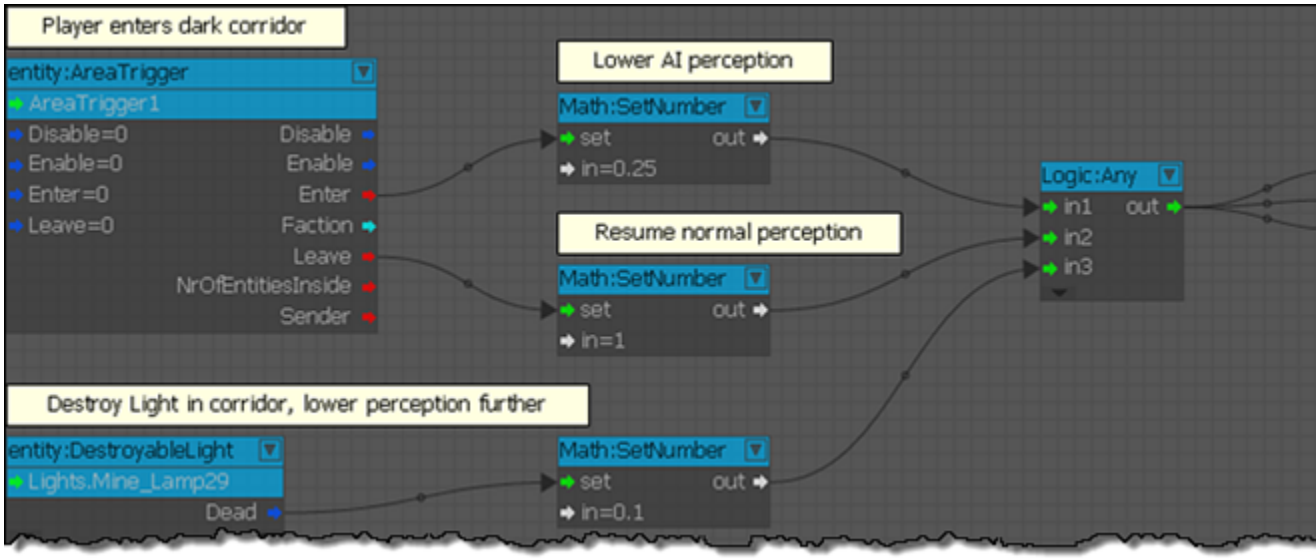
- **AI:AIAwarenessToPlayer** – The degree to which an AI agent is aware of the player's faction. Red is the most aware, while green is the least aware.

- **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.
- **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 (**...**) icon in the 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 **Tools**, **Other**, **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. 153).

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

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 **Tools**, **Other**, **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"/>
               </Channel>
           </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. 118)) 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" expirity="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.
- **expirity** (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. 119).

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 **Tools**, **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 **Tools**, **Other**, **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

The following standard Modular Behavior Tree nodes are supported. Thse 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 it 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.

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">
```

> **Warning**
> The LuaGate node uses the legacy scripting context and does not work with the new component entity system.

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>
```

Warning
The LuaWrapper node uses the legacy scripting context and does not work with the new component entity system.

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)" />
```

Warning
The ExecuteLua node uses the legacy scripting context and does not work with the new component entity system.

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()" />
```

Warning
The AssertLua node uses the legacy scripting context and does not work with the new component entity system.

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.
- **minSecondsBeweenUpdates**: (Optional) The minimum amount of delay between updates.
- **useReferencePointForInitialDirectionAndPivotPosition**:

Behavior

The node does not succeed or fail.

Example

```
<AimAroundWhileUsingAMachingGun minSecondsBeweenUpdates="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 irregardless 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** paramter 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 onthe 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 deined 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 anddisables 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>
```

InflateAgentCollisionRadiusUsingPhysicTricksTrick 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

When	Lua variable	Overridden XML tag
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

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.

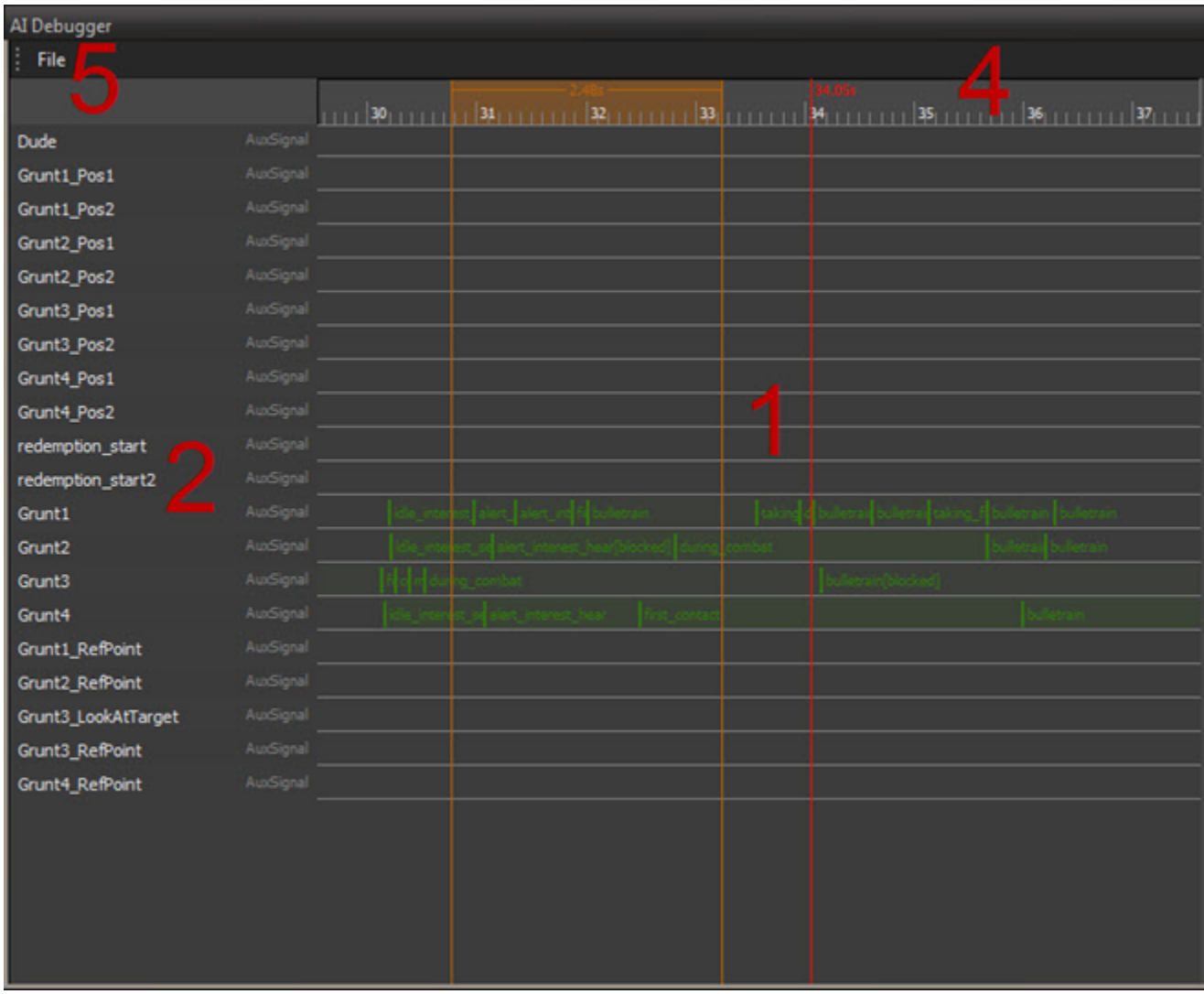
For example, a game project may connect a keyboard input to log event data on the Bookmark stream whenever a button is pressed, which informs the QA team to push the button whenever odd behavior from the AI is observed.

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 **Tools**, **Other**, **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.

Menu Item	Description
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 **Tools**, **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

Value	Description
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
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 diplays 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 (**...**) icon 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

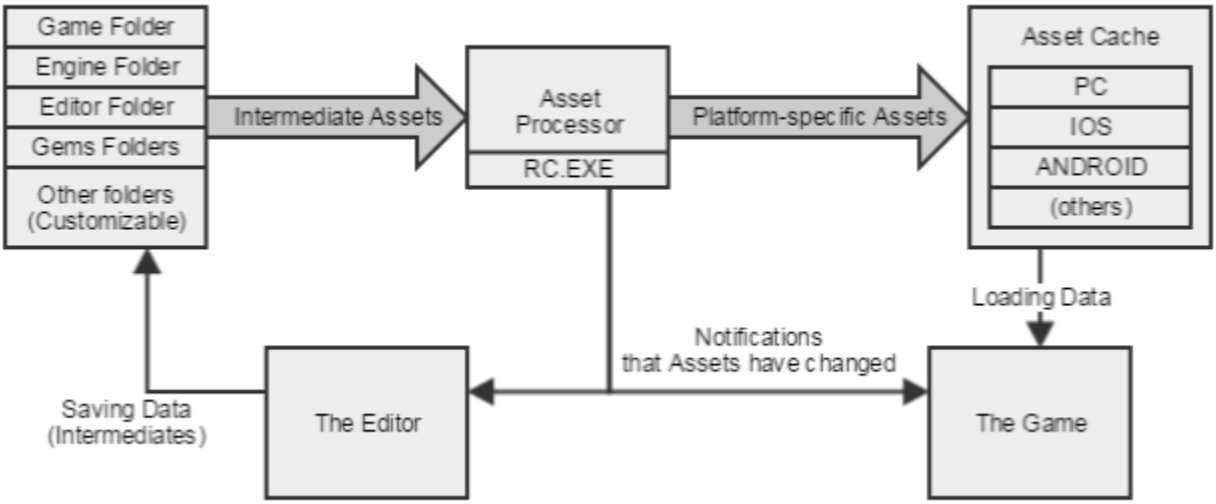
- Goalpipe selected
- Goalpipe inserted
- Goalpipe reset
- RefPoint position set
- Stimulus registered
- AI System reset
- OnEnemyHeard
- OnEnemyMemory
- OnEnemySeen
- OnInterestingSoundHeard
- OnLostSightOfTarget
- OnMemoryMoved
- OnNoTarget
- OnObjectSeen
- OnSomethingSeen
- OnSuspectedSeen
- OnSuspectedSoundHeard
- 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 OS-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 operating systems.

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 OS 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.

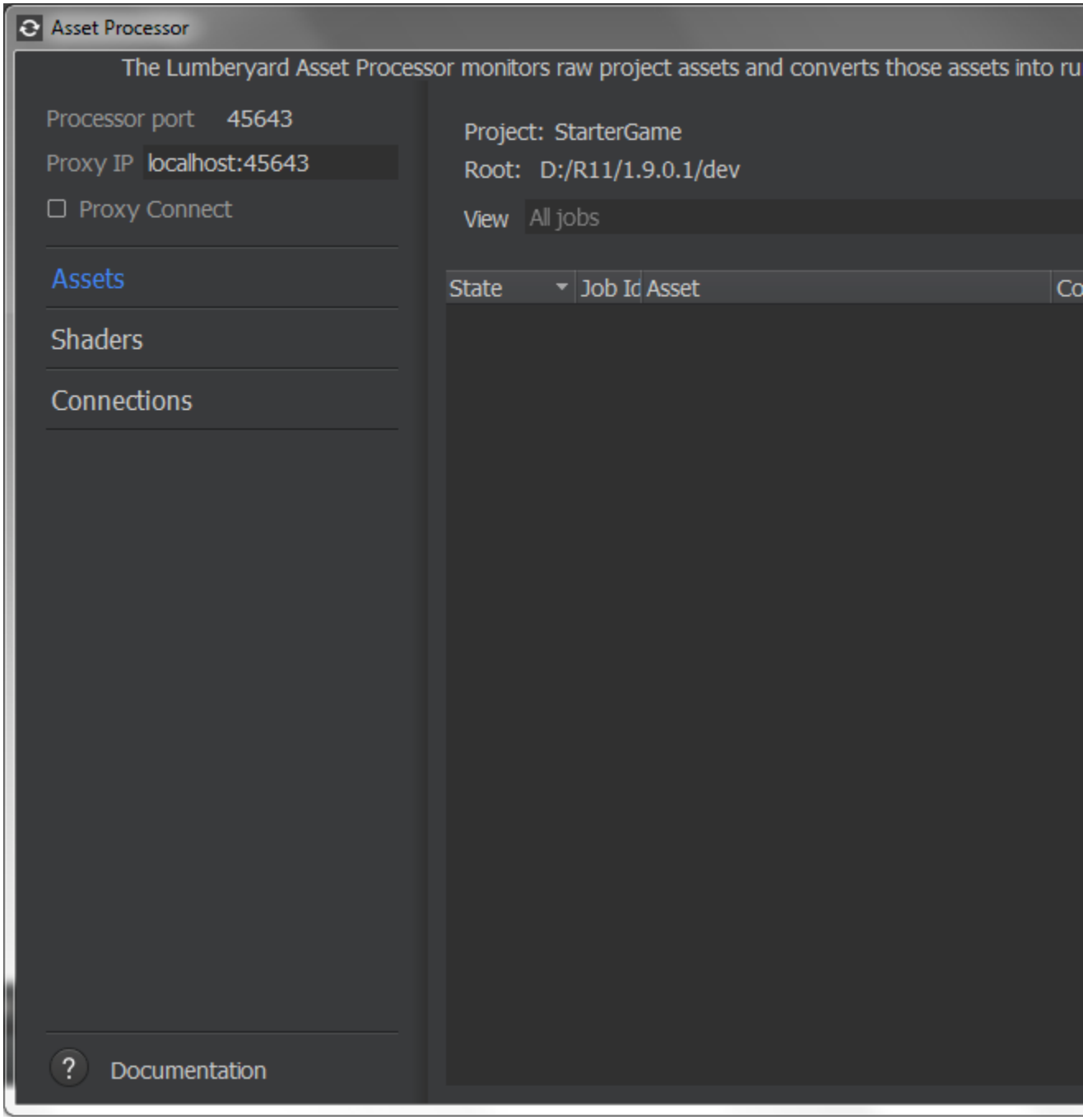
Each output directory in the asset cache represents a full image of all files (except for executables and related files) needed to run the game. The Asset Processor curates the directory to keep it up to date, ensuring that new files are ready to use in the game and Lumberyard Editor as soon as possible. Game runtimes load assets only from the asset cache and never directly from your input source folders.

Topics

- Technical Information: Asset IDs and File Paths (p. 177)
- Asset Browser (Preview) (p. 179)

Using the 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 enables games to run on other platforms without deploying assets to that platform. Instead, the assets are accessed from the asset cache on a connected Windows or macOS system. With this feature, you can also run games that use someone else's assets.

By proxying requests through itself, the Asset Processor communicates with an iOS or Android shader compiler server through a USB cable on iOS and Android.

On Windows, 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 that it needs to operate or if you retrieve a new version.

On macOS, you must manually start the Asset Processor (located in the `/dev/BinMac64` directory) from a command line window.

> **Note**
> Symbolic links are not supported when using the Asset Processor on macOS. To ensure that the Asset Processor works properly on macOS, follow these guidelines:
>
> - Do not use a symbolic link for your cache directory when you store compiled assets in a central location.
> - Do not store your source project assets in a symbolic link directory.
> - Use a unique cache directory. Do not share the cache directory with a Windows system that is also running the Asset Processor.

You do not need to close the Asset Processor when getting latest from source control. Nor must you 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 exit the Asset Processor by right-clicking its icon in the notification area on the taskbar (Windows) or menu bar (macOS).

The Asset Processor can also serve files directly to running console games so that the assets aren't required 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.

Modifying the Asset Processor Configuration File

Use the `AssetProcessorPlatformConfig.ini` configuration file (located in the root Lumberyard installation directory) to perform the following tasks:

- Add new file types for the Asset Processor to feed to the Resource Compiler, copy into the cache, or alter existing file type rules.
- Alter the ignore list.
- Alter which platforms are currently enabled. The default value is the host platform that the Asset Processor runs on. The Asset Processor automatically builds assets for the host platform. For example, if the Asset Processor is running on Windows, it builds Windows assets even if **`pc`** is not enabled in the `.ini` file. If the Asset Processor is running on macOS, it builds macOS assets even if **`osx_gl`** is not enabled in the `.ini` file. You can build assets for other platforms by editing the `.ini` file and enabling the desired platforms.
- Add additional folders for the Asset Processor to watch. For example, if you want to share particle libraries and associated textures between projects.
- Alter which files trigger related files to be rebuilt. This is called metafile fingerprinting.

To add game-specific overrides, you can 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 ignore list, platforms, and file types.

For more information about these configuration files, see Configuring the Asset Pipeline (p. 165).

Using the Asset Processor Batch File

The `AssetProcessorBatch.exe` batch file compiles all assets for the current project and enabled platforms. If the batch file succeeds without errors, it exits with a 0 code. You can use the batch file as part of your build system for automation.

The `AssetProcessorBatch.exe` batch file 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 the Asset Processor

Use the following techniques to debug Asset Processor issues:

- Exit Asset Processor and then restart it from the project or branch that you're currently working in. You may need to quit the Asset Processor in the notification area on the Windows taskbar. The close button hides the application.
- Clear the asset cache by deleting the `Cache` folder located in the Lumberyard root directory when the Asset Processor is not running. Then restart the Asset Processor to rebuild all assets.

Configuring the Asset Pipeline

> **Important**
> The Asset Builder SDK is now preferred over the legacy `rc.exe` program for adding asset types to the pipeline. Instead of using the `rc.exe` program, make a builder module that you derive from the `BuilderSDK`. These modules are self configuring. For instructions and examples on how to write builders that process your own asset types, see the Asset Builder API documentation. We recommend that you do not rely on the old `rc.exe` pipeline, although its still available if you have legacy code.

You can configure the Lumberyard asset pipeline by editing the `\dev\AssetProcessorPlatformConfig.ini` file that `rc.exe` program uses. You can add your own asset types to it by modifying the sections of the file described in this document. When you check in your changes to the config file, the version of the assets on your collaborators' computers is updated automatically. This removes the need for you to manually refresh the cache on each coworker's computer.

The `AssetProcessorPlatformConfig.ini` consists of six sections. The `.ini` file uses standard Qt/Windows `.ini` file formatting rules. Comments are preceeded by a semicolon, and named sections are designated by square brackets.

> **Important**
> Backslashes in `.ini` files have a special meaning. To use a regular backslash character, you must prefix it with another backslash. To avoid problems with file paths, the asset processor and asset pipeline use forward slashes for path names. However, if you need to use backslashes in regular expressions, you must also escape them so that they can be recognized by the regex system. For example, you must specify the regular expression `.*\/Levels\/.*` like this:

```
.*\\/Levels\\/.*
```

In addition to the `AssetProcessorPlatformConfig.ini` file, you can also have an `AssetProcessorGamePlatformConfig.ini` file. You can add this file to your project subfolder to override any configurations specifically for a project. The final configuration is the result of the merging of both files. Because the `AssetProcessorGamePlatformConfig.ini` file is read last, it takes priority.

Platforms Section

Use the `Platforms` section to enable and disable operating systems for the entire project. Note that "disabling" simply means that the game project does not use the specified operating system. When you disable an operating system, the related assets are removed, and the associated space on your hard drive is freed up.

In the following example, PC is enabled and other operating systems are commented out.

```
[Platforms]
pc=enabled
;es3=enabled
;ios=enabled
;osx_gl=enabled
```

Because the default value for an operating system is `disabled`, the operating systems in the example that are commented out are not enabled.

If you want to enable an operating system that is already listed in the `[Platforms]` section, simply remove the semicolon to uncomment the corresponding line.

If an entry for a game OS that you want is not in the list, you can add it. However, you must also handle OS related differences like image formats. To do so, you must change the code in the asset processor (and possibly the image compiler and other builders).

If you are using the `rc.exe` pipeline, specified operating systems are passed as parameters to the `rc.exe` program.

Jobs Section

Use the `Jobs` section to control how many parallel jobs to run, as in the following example.

```
; ---- The number of worker jobs, 0 means use the number of logical cores
[Jobs]
minJobs=1
maxJobs=0
```

Setting `maxJobs` to zero specifies using as many cores as are available. A number other than zero limits the cores used to no more than the number that you specify.

MetaDataTypes Section

Use the `MetaDataTypes` section to tell the asset system that certain file types are associated with other files in the same folder. These specifications control the compilation of side-by-side assets, as in the following example.

```
[MetaDataTypes]
exportsettings=
animsettings=i_caf
```

```
Animations/SkeletonList.xml=i_caf
cbc=abc
fbx.assetinfo=fbx
```

Entries on the left and right sides of the equals sign specify file extensions of asset files in the same folder. If a file with the extension on the left changes, then the file with the extension on the right must also be rebuilt if it has the same filename. For example, the line `animsettings=i_caf` means that if a file called `example.animsettings` changes, `example.i_caf` will be recompiled.

The line `exportsettings=` means that when any file with the extension `.exportsettings` changes, any asset file that has the same filename as the file with the `.exportsettings` extension is invalidated. For example, a change in the `MyImage.TIF.exportsettings` file invalidates the `MyImage.TIF` file.

In the example `Animations/SkeletonList.xml=i_caf`, the left side specifies not an extension, but a specific file. Whenever the `Animations/SkeletonList.xml` file changes (note that the forward slash indicates a directory path), all files with the extension `.i_caf` are invalidated.

> **Note**
> If you use the Asset Builder SDK, you can declare your dependencies on other files explicitly. This makes the `[MetaDataTypes]` section less important.

ScanFolder Section

Use the `ScanFolder` section to direct the Asset Processor to monitor the assets in specific folders. The following example directs the Asset Processor to monitor the `Editor` folder.

```
[ScanFolder Editor]
watch=@ROOT@/Editor
output=editor
recursive=1
order=30000
```

You can add as many scan folders as you want, but each folder must have a unique name. Because the scan folders are stored in a hash table using the name specified in square brackets, make sure that the name following `ScanFolder` is unique.

You can use the aliases `@root@` and `@gamename@` as placeholders to enable portability to the computers of other users who are working on the same project.

The `ScanFolder` section has the following parameters.

Entry	Description
`watch=`*`<foldernamewatch>`*	Watch this specific folder for assets.
`output=`*`<foldernameout>`*	Put the contents of this watch folder into the subfolder of the `@assets@` folder called *`<foldername>`*.
`recursive=1`	Recurse into subfolders.
`order=30000`	Declares a priority order. The lower the number, the more "important" a folder is. The game folder for your project is always considered 0, the most important. **Note** The `order` parameter affects only assets with the same name. For example, suppose you have an asset called `MyTexture.TIF` in two separate scan folders. If both asset files map to the same output file, then the asset file with the lower order number overrides the one with the higher.

Notes

- In most cases, you do not have to specify an output folder. The output folder remaps source folders into subfolders of the cache. Usually folders that contain assets go into the cache directly, without requiring a subfolder.
- It is not considered an error if a scan folder is missing. This behavior is by design because it lets you have optional folders for assets. For example, this might be useful for test cases.
- Removing folders from the `ScanFolder` sections removes any corresponding assets from the cache. If the assets specified were overriding other assets, the overridden assets are reinstated and become primary assets again.

Exclude Section

Use the `Exclude` section to add file path patterns to ignore. As in the rest of the `.ini` file, backslashes must be prefixed with an extra backslash to escape them from `.ini` file processing.

The following example excludes alembic compression templates and temporary animation compression files.

```
[Exclude AlembicCompressionTemplates]
pattern=.*Presets\\/GeomCache\\/.*

[Exclude TmpAnimationCompression]
pattern=.*Editor\\/Tmp\\/AnimationCompression\\/.*
```

Notes

- The regular expressions are standard `STD::regex` in extended format. Standard `STD::regex` rules apply.
- The input paths are always absolute paths. If you don't want to filter by absolute path, start your regular expressions with `.*`, as in the example.
- If you want to add new exclude rules, give them a unique name. The actual name does not matter as long as each is unique.

RC Section

Use the `RC` section to specify files to be processed by the `rc.exe` program or to be copied as-is into the asset cache without processing. The `RC` section is only for use by legacy RC modules and for specifying simple file copies to cache.

The `RC` section consists of a series of *recognizer descriptors*. Each descriptor specifies a set of files (by glob or by pattern) and what to do with the specified files. Changing the fields of the recognizer invalidates assets according to the change made.

> **Important**
> Because they do not use the legacy `rc.exe` program, builders implemented as builder modules do not use the `RC` section. Instead, they derive their configuration programmatically or read it from a custom config file. If you create your own `BuilderSDK` builder, do not add anything to the `RC` section.

The following code block shows the syntax of the `RC` section.

```
[RC (recognizer name)]
```

```
; ---- Choose either pattern or glob. You cannot choose both.
pattern=(pattern to use to recognize these files)
glob=(glob to use to recognize these files)
params=(command line params/copy/skip)
(platformname)=(params)
lockSource=(true/false)
priority=(0...n) ; Higher numbers are more important.
critical=(true, false)
version=(0...n)
```

The following table describes each parameter and its options.

Parameter	Description
`pattern`	A regular expression that specifies the files to process. When you use regular expressions, remember to escape any backslashes.
`glob`	A wildcard expression like `*.tif` that specifies binary glob files to process.
`params`	The `params` parameter can take one of the following three options. 1. The default parameters to pass to the `rc.exe` program to process the kind of asset specified. 2. `copy` – Copies the file as-is into the cache. It does not invoke `rc.exe`. **Note** Copy jobs are the most common type of job. For example, in the `dev\assetprocessorplatformconfig.ini` file that is included with Lumberyard, most `[RC]` sections specifiy `params=copy`. 3. `skip` – Skips the specified file type entirely. The `skip` option is typically more useful in the `platformname` parameter.
`(platformname)=(params)`	Specifies OS-specific parameters. You can use the `params` parameter to specify default parameters, and then override them for specific operating systems when required by using `platformname=params`. For example, the statement `pc=/TEST` overrides the default parameters for PC and passes the parameter `/TEST` to the `rc.exe` program.
`lockSource`	Can be `true` or `false`. When `true`, causes the job to wait until it can gain an exclusive read/write lock on the source file. The `lockSource` parameter is useful for dealing with applications that hold onto a file and then slowly stream data into it. For example, if a program creates very large files over a long period of time, you can set `lockSource=true` to avoid processing an asset until the other application releases it. Use of this parameter is relatively rare and is generally expensive, so you should avoid using it unless absolutely required.
`priority`	Specifies job priority. A larger number gives a job greater priority in the queue. Normally, you should assign a larger number to assets that are likely to be needed from the start or that affect gameplay. This ensures that they get compiled sooner. **Note** `params copy` jobs have an default priority of 1.
`critical`	Can be `true` or `false`. Critical jobs cause the editor splash screen to continue displaying and pause the startup of the runtime until every critical job has been completed. Marking jobs as critical ensures they are complete before the editor is allowed to start.

Parameter	Description
	You can specify entire types of assets as critical. This can be useful for files that are used during startup, cause bad behavior if they are not ready during bootstrap, or cannot be reloaded live. **Important** Because critical jobs can delay the startup of the editor for the first time, not having critical jobs is always the preferred choice. Alternative approaches include: • Making the editor or runtime capable of reloading the asset live after it is compiled. • Making a call to compile the asset on demand using the asset system bus. You can use the public function `CompileAssetSync` to do this. See the Lumberyard source code for examples. **Note** `params copy` jobs are critical by default.
`version`	An arbitrary versioning number. The default is 0. Changing the version number invalidates the assets specified and causes them to be rebuilt. The `version` parameter provides a convenient way to cause a rebuild of all assets of a particular kind. For example, you might make changes to the compiler that builds a particular kind of asset. Then, when you check in your changes to the `.ini` file, local assets of workers receiving the update are rebuilt for them automatically.

The following example specifies how `.tiff` files are to be processed.

```
[RC tif]
pattern=.*\\.tiff?
params=/imagecompressor=CTSquish /streaming=1
es3=/imagecompressor=CTSquish /streaming=0
ios=/imagecompressor=CTSquish /streaming=0
; Streaming = 1 splits files.
lockSource=true
```

The example has the following characteristics.

- It declares a `recognizer` called `tif` (because `[RC tif]` is the name).
- The `pattern` specifies all files which match the regex expression `.*\.tiff?`. Note that the example escapes the backslash.
- The `params` parameter specifies the default parameters with which to invoke `rc.exe`. In the example, `.tiff` files will be compiled into the `/imagecompressor=CTSquish /streaming=1` format.
- For ES3 and iOS, streaming is turned off, overriding the default that was specified in the `params` parameter.
- `lockSource` is set to `true` to avoid conflict with external tools that create a zero byte file, pause for many seconds, and then fill it with data.

The following example specifies how `.tiff` files in the `GoldenImages` subfolder are to be processed.

```
; Feature tests use the raw .tif files for the golden image comparison.
[RC goldenimages]
```

```
pattern=.*GoldenImages\\/.*\\.tif
params=copy
```

The example has the following characteristics.

- It declares a `recognizer` called `goldenimages` which apples to any `.tiff` file in the `GoldenImages` subfolder.
- The `params` parameter specifies `copy`, so any `.tiff` file in the `GoldenImages` subfolder is copied to the cache without processing.

Notes

The two example RC sections are both in the same file. This has the following important consequences:

- The multiple rules that match the files all apply simultaneously. They are not exclusive. If you have two rules that apply to the same file, both rules are run. For example, the rules in the two examples would both apply to a file called `\dev\SamplesProject\textures\GoldenImages\myfile.tif`. The rules would produce both a `.dds` compressed version of `myfile.tif` and an uncompressed `myfile.tif` file that is copied into the cache.
- If you want to specify an exclusive subfolder rule, you must use inverse regex selectors to create exclusion patterns.

  The following example shows a set of rules that apply exclusively to `.png` files. The two rules are written so that any `.png` file matches only one of the rules.

```
;Example: Use the specified parameters to process all .png files except those in the
 libs/ui folder.
[RC png-normal]
pattern=(?!.*libs\\/ui\\/).*\\.png
params=/imagecompressor=CTSquish /streaming=0
lockSource=true
```

```
;Example: Process all .png files in the libs/ui folder using linear color space.
[RC png-ui]
pattern=(.*libs\\/ui\\/).*\\.png
params=/imagecompressor=CTSquish /streaming=0 /colorspace=linear,linear
lockSource=true
```

  For more examples, see the default `\dev\AssetProcessorPlatformConfig.ini` file.

Common Problems

When troubleshooting, be aware of the following pitfalls.

- Not escaping your regular expressions with two backslashes. Remember that one of the slashes is removed when the `.ini` file is processed.
- Duplicating a rule without changing its name. The rule `[RC png]` has a name of `png`. These names are inserted into an unordered hash. If you specify another section with the name of `png`, the second section overwrites the other in random order. This behavior is by design. For example, you can use it to allow your game version of the `.ini` file to override particular sections or specify `skip` to skip them. Otherwise, if you want to add new rules, give them a unique name. The actual name does not matter as long as each is unique. This is especially true for named sections like the `Exclude` and `ScanFolder` sections.
- Not understanding that `all` recognizers that match apply, not just the first one.

- Forgetting to prefix your regular expressions with `.*`. By default, the input files that you specify are considered absolute paths. This behavior is by design because it lets you exclude or include files based on absolute paths, if that is your intent. Use the `.*` prefix if you want to use relative paths.

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.

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 `BuildShaderPak_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. `BuildShaderPak_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 one of the following commands to do this:

- If you are using Visual Studio 2015:`lmbr_waf build_win_x64_vs2015_profile -p all --targets=CrySCompileServer`
- If you are using Visual Studio 2013:`lmbr_waf build_win_x64_vs2013_profile -p all --targets=CrySCompileServer`

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 one of the following sets of commands:

- If you are using Visual Studio 2015:

```
lmbr_waf build_win_x64_vs2015_profile -p game_and_engine
lmbr_waf build_win_x64_vs2013_profile -p shadercachegen
```

- If you are using Visual Studio 2013:

```
lmbr_waf build_win_x64_vs2013_profile -p game_and_engine
lmbr_waf build_win_x64_vs2013_profile -p shadercachegen
```

Packing the Shader Cache Using a Batch File

The `BuildShaderPak_DX11.bat` file is used to generate the `ShaderCache.pak` files, which are saved to the `dev\build\`*`patform`*`\`*`your_game`* directory. The batch file works by first calling `ShaderCacheGen.exe` and then calling `Tools\pakShaders.bat`.

Run `BuildShaderPak_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 you 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 METAL
rem if changing the above platform, also change the below directory name (D3D11, 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_version\dev\cache\your_game\platform\user\shaders\cache\D3D9\

lumberyard_version\dev\cache\your_game\platform\user\shaders\cache\D3D10\

lumberyard_version\dev\cache\your_game\platform\user\shaders\cache\D3D11\

lumberyard_version\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\
```

`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_version\dev\cache\your_game\platform\user\shaders\cache\<platform>
\lookupdata.bin -> Shadercache\<platform>\lookupdata.bin
lumberyard_version\dev\cache\your_game\platform\user\shaders\cache\<platform>\CGPShader
\FixedPipelineEmu* -> Shadercache\<platform>\CGPShader\FixedPipelineEmu*
lumberyard_version\dev\cache\your_game\platform\user\shaders\cache\<platform>\CGPShader
\Scaleform* -> Shadercache\<platform>\CGPShader\Scaleform*
lumberyard_version\dev\cache\your_game\platform\user\shaders\cache\<platform>\CGPShader
\Stereo* -> Shadercache\<platform>\CGPShader\Stereo*
lumberyard_version\dev\cache\your_game\platform\user\shaders\cache\<platform>\CGVShader
\FixedPipelineEmu* -> Shadercache\<platform>\CGVShader\FixedPipelineEmu*
lumberyard_version\dev\cache\your_game\platform\user\shaders\cache\<platform>\CGVShader
\Scaleform* -> Shadercache\<platform>\CGVShader\Scaleform*
lumberyard_version\dev\cache\your_game\platform\user\shaders\cache\<platform>\CGVShader
\Stereo* -> Shadercache\<platform>\CGVShader\Stereo*
```

```
lumberyard_version\dev\cache\your_game\platform\user\shaders\cache\<platform>
\lookupdata.bin -> Shaders\Cache\<platform>\lookupdata.bin
lumberyard_version\dev\cache\your_game\platform\user\shaders\cache\<platform>\Common.cfib -
> Shaders\Cache\<platform>\Common.cfib
lumberyard_version\dev\cache\your_game\platform\user\shaders\cache\<platform>\fallback.cfxb
 -> Shaders\Cache\<platform>\fallback.cfxb
lumberyard_version\dev\cache\your_game\platform\user\shaders\cache\<platform>
\fixedpipelineemu.cfxb -> Shaders\Cache\<platform>\fixedpipelineemu.cfxb
lumberyard_version\dev\cache\your_game\platform\user\shaders\cache\<platform>
\FXConstantDefs.cfib -> Shaders\Cache\<platform>\FXConstantDefs.cfib
lumberyard_version\dev\cache\your_game\platform\user\shaders\cache\<platform>
\FXSamplerDefs.cfib -> Shaders\Cache\<platform>\FXSamplerDefs.cfib
lumberyard_version\dev\cache\your_game\platform\user\shaders\cache\<platform>
\FXSetupEnvVars.cfib -> Shaders\Cache\<platform>\FXSetupEnvVars.cfib
lumberyard_version\dev\cache\your_game\platform\user\shaders\cache\<platform>
\FXStreamDefs.cfib -> Shaders\Cache\<platform>\FXStreamDefs.cfib
lumberyard_version\dev\cache\your_game\platform\user\shaders\cache\<platform>
\scaleform.cfxb -> Shaders\Cache\<platform>\scaleform.cfxb
lumberyard_version\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	
PC, OpenGL 4	\GL4	

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 operating systems.
   - 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 OS files, nothing else needs to be deployed to the device. Instead, they can all be accessed remotely.
4. The `system_game OS_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 **Tools**, **Console** to open the **Console** window, if it's not already open.
2. In the **Console** window text box, type `ed_MissingAssetResolver 1`.
3. In Lumberyard Editor, choose **Tools**, **Other**, **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 operating systems 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 `engineroot.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 operating systems:

```
@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 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 queueing the request for later if you are in a thread-sensitive module.

Asset Browser (Preview)

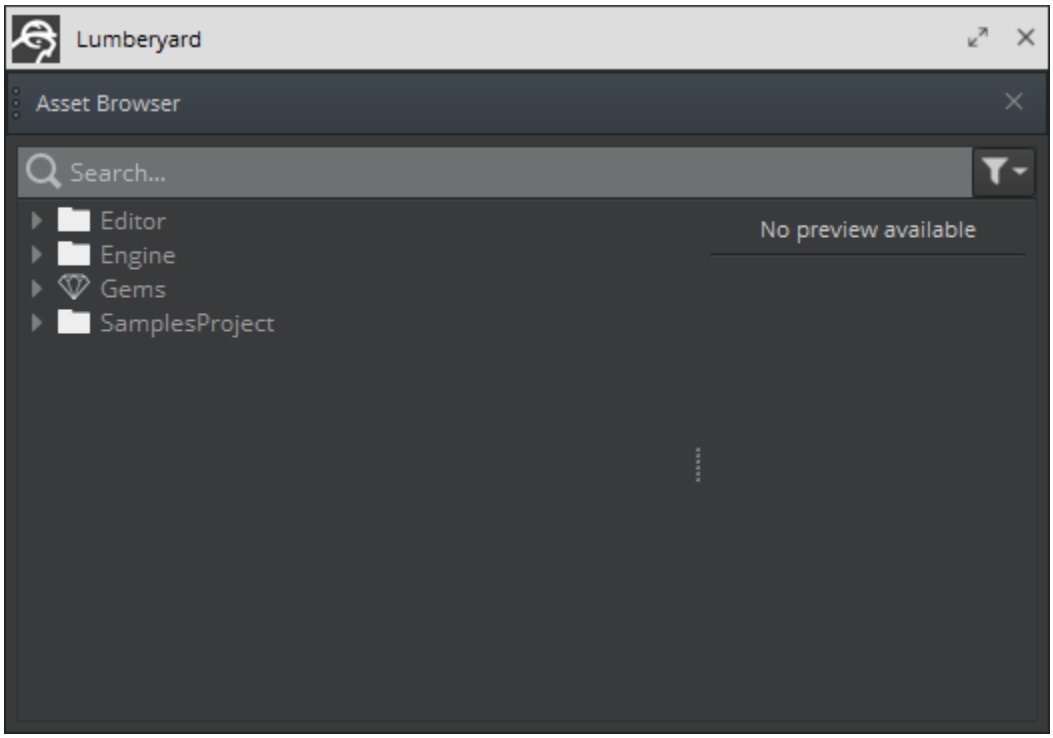
> The new Asset Browser is in preview release and is subject to change.

The Asset Browser displays all project assets in a source folder and file view to enable quick access and interaction. Source assets are also displayed with their products. For example, an `.fbx` file would appear with its meshes and animations. The Asset Browser also includes other features:

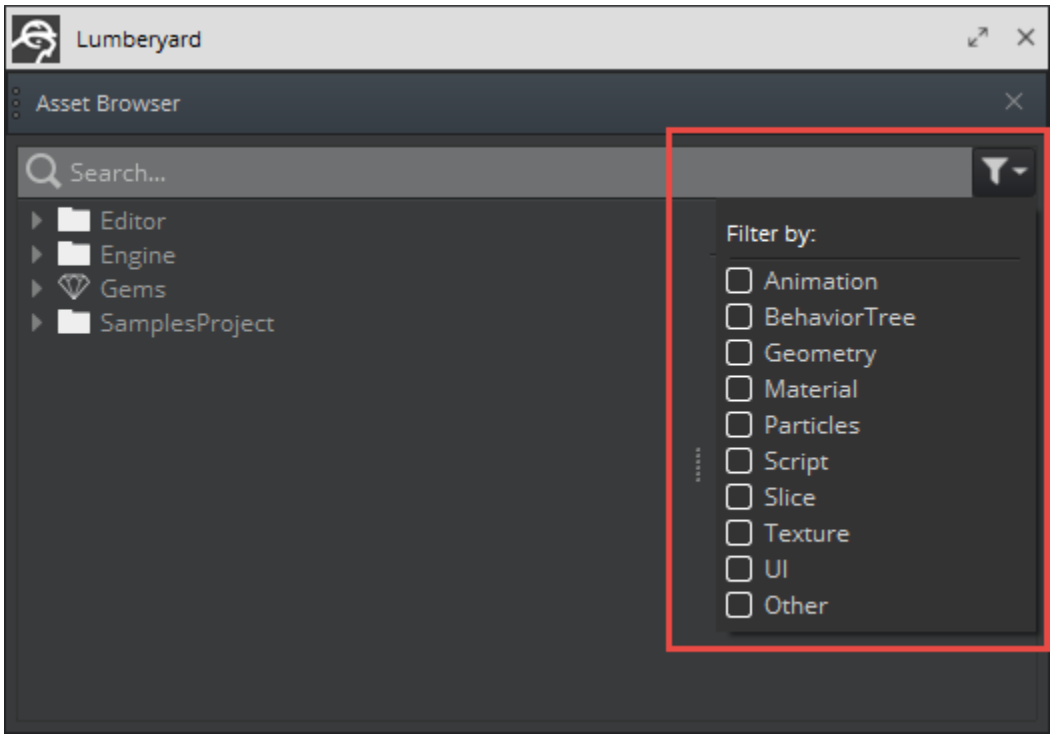
- Drag-and-drop interaction
- Right-click context menus for each asset
- Asset name filtering

You can use the Asset Browser with other editor components—such as the viewport, **Entity Outliner**, and **Entity Inspector**—to improve your development workflow. It replaces the original **File Browser**.

To open the Asset Browser, click **Tools**, **Asset Browser**. You can dock the Asset Browser window to the Lumberyard Editor.



To filter assets by type, click the filter icon and select your preferred filter types. If you aren't seeing your assets in the **Asset Browser**, clear any selections in the filter list.



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 3D audio plugins for Wwise, which enables you to create an immersive sound experience in your game. For information about how to install Wwise and 3D audio plugins, see Installing Audiokinetic Wwise and 3D Plugins (p. 182).

Lumberyard also supports a free "compact" version called Wwise LTX. The runtime SDK for it comes pre-configured with Lumberyard. For more information, see Installing Audiokinetic Wwise LTX (p. 185).

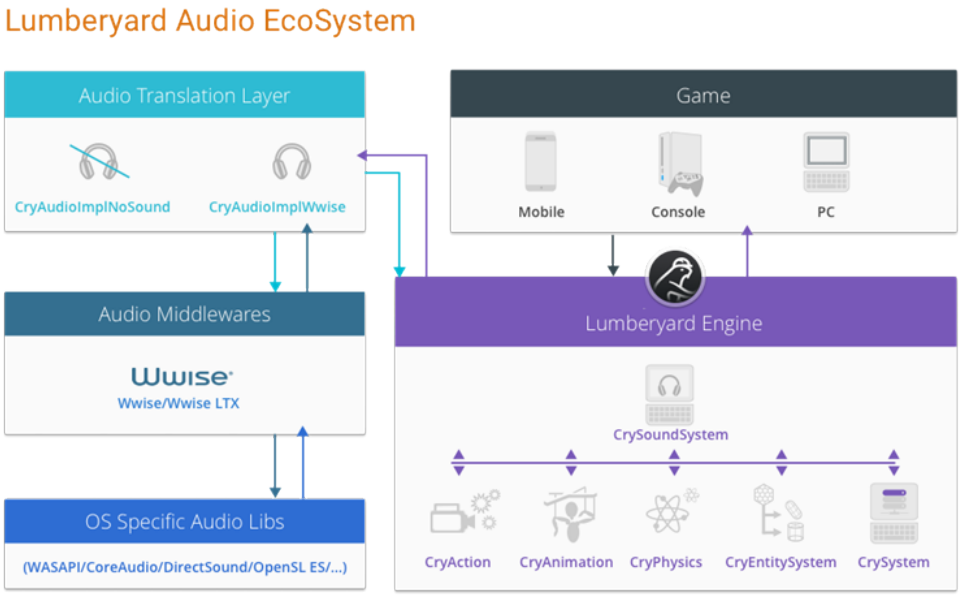
The audio system consists of the following elements:

- 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. 646).

Topics

Audio System Architecture



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 and 3D Plugins

Lumberyard supports 3D audio, or spatialized audio, with which you can create a sound environment that convincingly simulates the acoustic spatial cues in the real world. With 3D audio you can design an immersive sound experience where players can hear sounds in relation to their position in the game world. A sound produced directly in front of you can be easily distinguished from the same sound produced at a distance, or above or behind you. Advanced 3D audio systems, such as the Wwise 3D audio plugins that Lumberyard supports, can even replicate sounds that have bounced off of walls and other parts of the environment.

To set up 3D audio for Lumberyard, you need the following:

- Wwise 2016 1.1, both the SDK and the (purchased) authoring tool

- A 3D audio plugin for Wwise. Lumberyard has been tested with the following 3d audio plugins:
  - RealSpace3D (paid)
  - Oculus Spatializer (free)
- Visual Studio 2015 or 2013 Professional (to build Lumberyard)
- Lumberyard installation

Use the following procedures to prepare your Lumberyard installation for 3D audio.

- Replacing Wwise with Full Version (p. 183)
- Building Lumberyard for 3D Audio (p. 184)

Replacing Wwise with Full Version

Lumberyard comes preconfigured with the free "compact" version of Wwise, called Wwise LTX. To use 3D audio plugins, you must purchase the full version of Wwise. Then use the following procedure to replace the free version with the full version that you purchased.

To replace the free Wwise with the full version Wwise

1. Navigate to `lumberyardroot\3rdParty\Wwise\`

   You should see a directory named `LTX_2016.1.1.5823` (or similar), which contains the free version that Lumberyard ships with.
2. Copy and paste your copy of Wwise 2016 1.1 SDK into `lumberyardroot\3rdParty\Wwise\`.

   Note
   If you open the directory that you just pasted, you should see a directory named `SDK`, which contains the different builds of Wwise for various platforms. If you have Visual Studio 2015 installed, you will use the directory `x64_vs140` later in this procedure. If you have Visual Studio 2013 installed, you will use `x64_vs120`.
3. Navigate to `lumberyardroot\dev`. Use a text editor to open `SetupAssistantConfig.json`.
4. Search for **`wwise`**.

   The resulting config block that you find contains paths to the Wwise SDK.
5. Edit each `source` field to point to the full Wwise 2016 installation.

   The following example shows three `source` fields: one at the root and one for each compiler version. The Wwise version shown in the example may not be the same version as your Wwise, so verify that you are pointing to the version of your installation.

   The following picture shows the unedited block (on the left) and the edited block (on the right). The source fields are highlighted in red.

```
{
    "identifier" : "wwiseLtx",
    "name" : "wwiseLtxName",
    "version" : "LTX v2016.1.1.5823",
    "source" : "Wwise/LTX_2016.1.1.5823",
    "optional" : 0,
    "description" : "wwiseLtxDescriptionSummary",
    "detailedInstructions" : "wwiseLtxDetailedInstructions",
    "roles" : ["compilegame", "compileengine", "compileeditor", "compileandroi
    "symlinks" :
    [
        {
            "source" : "Wwise/LTX_2016.1.1.5823/SDK",
            "destination" : "$CODEFOLDERNAME$/SDKs/Wwise",
            "exampleFile" : "include/AK/AkWwiseSDKVersion.h"
        },
        {
            "compilers" : ["vc120"],
            "source" : "Wwise/LTX_2016.1.1.5823/SDK",
            "destination" : "$CODEFOLDERNAME$/SDKs/Wwise",
            "exampleFile" : "x64_vc120/Release/lib/AkSoundEngine.lib"
        },
        {
            "compilers" : ["vc140"],
            "source" : "Wwise/LTX_2016.1.1.5823/SDK",
            "destination" : "$CODEFOLDERNAME$/SDKs/Wwise",
            "exampleFile" : "x64_vc140/Release/lib/AkSoundEngine.lib"
        }
    ]
},
```

```
{
    "identifier" : "wwiseLtx",
    "name" : "wwiseLtxName",
    "version" : "2016.1.1.5823",
    "source" : "Wwise/2016.1.1.5823",
    "optional" : 0,
    "description" : "wwiseLtxDescriptionSummary",
    "detailedInstructions" : "wwiseLtxDetailedInstructions",
    "roles" : ["compilegame", "compileengine", "compileeditor", "compilean
    "symlinks" :
    [
        {
            "source" : "Wwise/2016.1.1.5823/SDK",
            "destination" : "$CODEFOLDERNAME$/SDKs/Wwise",
            "exampleFile" : "include/AK/AkWwiseSDKVersion.h"
        },
        {
            "compilers" : ["vc120"],
            "source" : "Wwise/2016.1.1.5823/SDK",
            "destination" : "$CODEFOLDERNAME$/SDKs/Wwise",
            "exampleFile" : "x64_vc120/Release/lib/AkSoundEngine.lib"
        },
        {
            "compilers" : ["vc140"],
            "source" : "Wwise/2016.1.1.5823/SDK",
            "destination" : "$CODEFOLDERNAME$/SDKs/Wwise",
            "exampleFile" : "x64_vc140/Release/lib/AkSoundEngine.lib"
        }
    ]
},
```

You are now ready to build Lumberyard with your full 2016 version of Wwise.

If you want to return to the LTX version of Wwise, change the source to point to the old `LTX` directory.

Building Lumberyard for 3D Audio

After you have installed and configured the full version of Wwise, you must build Lumberyard before you can use your new Wwise installation to create 3D audio.

To build Lumberyard after installing Wwise

1. Start Lumberyard Setup Assistant. You can do this by running `SetupAssistant.bat` from the Lumberyard root directory.
2. Step through each page of the Lumberyard Setup Assistant and ensure that all software and required SDKs are installed. When finished, close Lumberyard Setup Assistant.
3. Open a command prompt. Navigate to `lumberyardroot\dev\` and type **`lmbr_waf configure`**.

   When complete and successful, it places a VS solution in `lumberyardroot\dev\Solutions`.
4. Open the solution in Visual Studio. Find the **Editor** project. It should be located under **Sandbox** in Visual Studio's Solution Explorer.
5. Right-click the **Editor** solution and click **Set as StartUp Project**.
6. Build the editor. (p. 1847)
7. Ensure that the build completed successfully. Proceed to installing 3D audio plugins.

Lumberyard and Wwise are now configured to accept a 3D audio plugin. If the sound banks that your game loads were properly authored to take advantage of the plugin, they'll work immediately. You do not need to perform any special steps in Lumberyard in order to use 3D sound banks instead of the regular sound banks.

To load 3D audio plugins and sound banks:

- Paste the 3D audio plugin DLL (the same one that you would be using with the Wwise authoring tool) in the same directory as the editor or game executable.

  Wwise automatically loads the plugin.

For debug builds, you can place it in the `dev\Bin64` (or `Bin64vc120` or `Bin64vc140`) directory.

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. For more information, see Installing Audiokinetic Wwise and 3D Plugins (p. 182).

To access the Wwise LTX documentation once Wwise LTX is installed, press the **F1** key. You can also 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 **Install it**.
4. If prompted to sign in to your Audiokinetic account, provide the requested information and click **Sign In**, or click **Skip sign in**.
5. On the next page, you can select the desired installation components and settings for Wwise LTX or accept the default. Then click **Install**.
6. If prompted with license terms, review the end user license agreement. Then click **Accept**.
7. Once the installation has successfully completed, click the **Launch Wwise (64-bit)** button under the Wwise LTX entry to run the Authoring Tool. You can also click the wrench icon to create a desktop shortcut.
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 and under Wwise LTX, choose **Launch Wwise (64-bit)**. Alternatively, if you created a desktop shortcut earlier, you can start that.
3. The first time you run Wwise LTX, you are prompted again to review and accept the End-User License Agreement (EULA). After accepting the EULA, click **Open Other**.

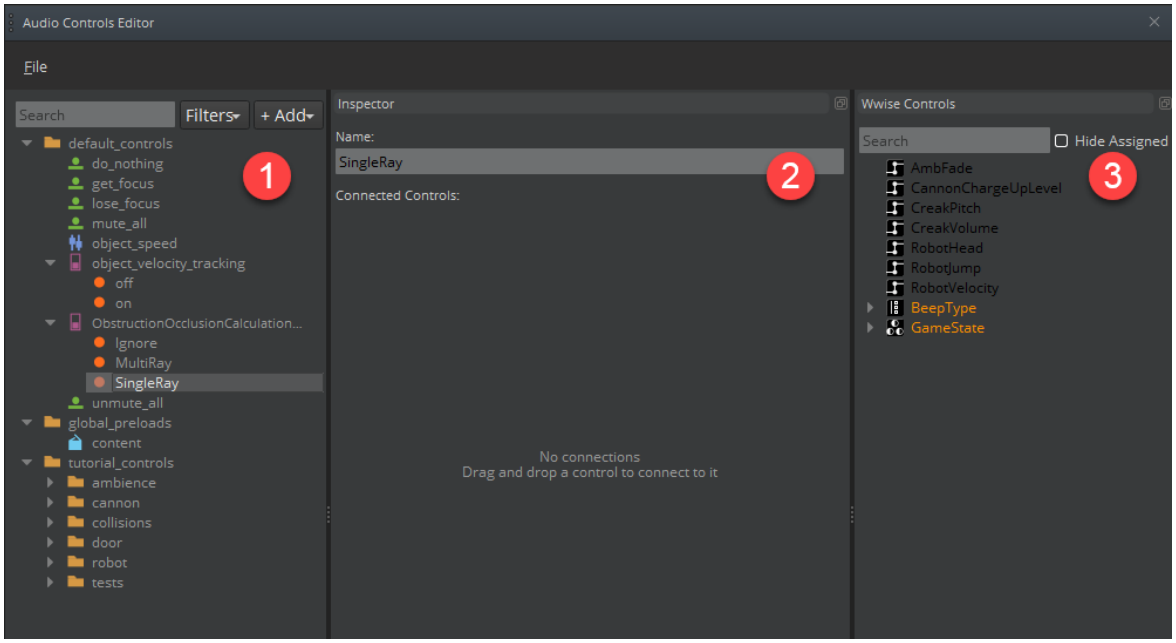
4. Navigate to and select the `.wproj` file located in `lumberyard_version\dev\SamplesProject\Sounds\wwise_project\` and then click **Open**.

   Alternatively, if you are not using SamplesProject, you may create a new Wwise LTX project by clicking **New**. For more information about setting up Wwise for your game project, see Using the Audio Controls Editor (p. 186).

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 (Wwise or Wwise LTX). 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.

To open the Audio Controls Editor, click **Tools**, **Other**, **Audio Controls Editor**. The editor consists of three panels: the ATL controls panel (1), Inspector panel (2), and middleware-specific (such as Wwise) Controls panel (3).



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	Icon	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.

Audio Control	Icon	Description
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 `Rock`, `Sand`, or `Grass`.
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, the controls exist only when that level is loaded. This setting is useful in low-memory systems because controls are loaded only in levels in which they are needed.
Auto Load	Available only for preloads. If Auto Load is selected, 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.
Preloaded Soundbanks	Available only for preloads. 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.
Platforms	Available only for preloads. Allows you to set which group of soundbanks to load for each platform. You can share a group between several platforms.
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 connects both of them.

> **Note**
> After creating a new control, in the Audio Controls Editor, click **File**, **Save All** 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: `\[game folder]\Sounds\wwise\`. These locations are hard-coded requirements and should not be changed.

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.

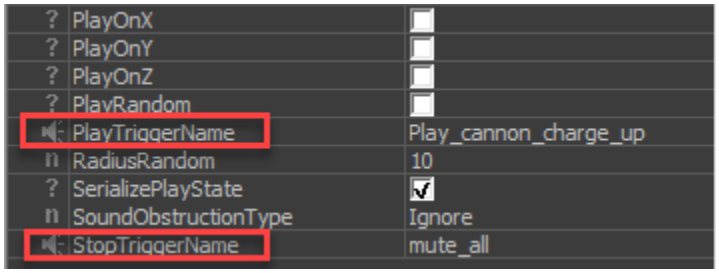
unmute_all control

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 a PlayTriggerName and StopTriggerName for audio entities in your level.

To access these properties, select the audio entity, and then view the **Properties** panel for the entity on the Rollup Bar.



Placing Triggers in Game

To hear a sound in the game, first place the audio entity that executes the audio trigger during gameplay. For each level, we recommended creating a dedicated audio layer that contains all your audio data. In the Rollup Bar, on 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**, and then drag and click to place the entity in the level.

PlayTriggerName

Upon activation of the audio trigger, the event specified in **PlayTriggerName** is executed. When that audio trigger is deactivated, the **PlayTriggerName** event is killed. Furthermore, if a **StopTriggerName** is specified, then the **StopTriggerName** event is executed when the audio trigger is deactivated.

This automatic stopping of the event can be useful, as you do not need to create any additional stop functionality inside of your audio middleware. It also assures that any looping sounds are stopped when the associated entity is disabled.

StopTriggerName

Upon deactivation of the audio trigger, the event specified in **StopTriggerName** is executed. For example, this can be playing another sound, or stopping all sounds, and so on.

You can bypass the automatic stopping of the **StartTriggerName** event by setting the `do_nothing` ATL control as the **StopTriggerName**. This can be useful, for example, when setting an audio trigger on a shooting animation of a gun, as you would want to hear the end of the gunfire even after the animation has finished. However, if you have not set an event in the **StopTriggerName** field, the **PlayTriggerName** would be terminated and cut short the gunfire sound. Setting `do_nothing` in **StopTriggerName** allows the **PlayTriggerName** to play in its entirety.

> **Note**
> If you set a looping sound as your **PlayTriggerName**, and then set set `do_nothing` as your **StopTriggerName**, the looping sound will continue to play until stopped by some other method.

If you need to stop the **PlayTriggerName** with another audio trigger that is set as **StopTriggerName**, 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 **PlayTriggerName**, create the stop functionality inside your audio middleware and set the connected ATLControl as the **StopTriggerName**.

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, obstuction 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 h` 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 calculates 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

This topic describes how to use legacy entity nodes in the **Track View** editor. These nodes will be deprecated and replaced with component entity nodes in a future release. For more information, see Component Entity Nodes (p. 395).

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. 810).

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. 195) —**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 whichs 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. Select 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 it should play, 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 the **Rollup Bar**, on the **Objects** tab, click **Audio**.
2. Under **Object Type**, click **AudioAreaAmbience**. Then click in your level to place the object.
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 begin to increase in volume.
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

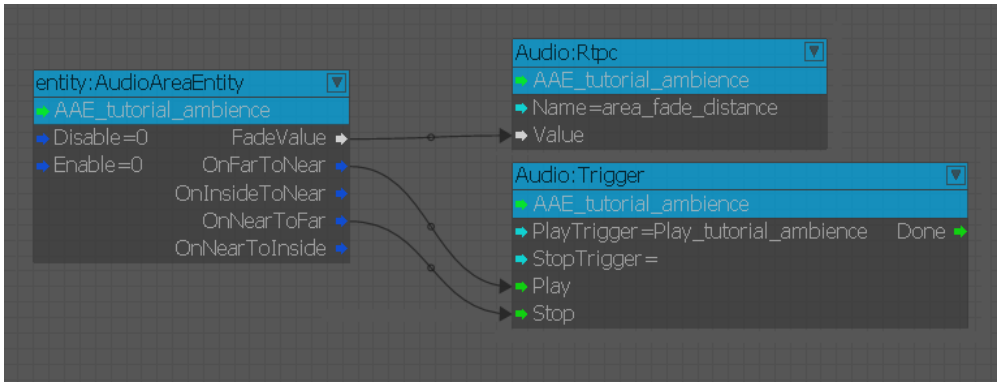
The **AudioAreaEntity** entity functions like the **AudioAreaAmbience** entity but requires manual setup in the Flow Graph editor to trigger the ATL controls. This extra step gives you access to multiple parameters and more advanced setup possibilities than the **AudioAreaAmbience** entity.

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 simplfy 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 **Tools**, **Other**, **Audio Controls Editor**.
7. In the **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 **Tools**, **Other**, **Audio Controls Editor**.
7. In the **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 similiar to that for adding ambient sounds to a level. For more information, see Adding Ambient Sounds to Levels (p. 192).

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 auxillary 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 approriate 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 **Tools**, **Track View** to open the **Track View** editor.
2. In the **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 **Tools**, **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 ProcLayer Track

Once you have determined on which scope you want to place the audio triggers, a **ProcLayer** track is first added to the scope. You can add any number of ProcLayers to a scope, which can help better organize the fragment.

To add a trigger to a ProcLayer track

- In **Mannequin Editor**, right-click on the applicable scope and click **Add Track**, **ProcLayer**.

You can also addp rocedural clips to any ProcLayer on any scope. These might, however, be saved to a different Animation Database (ADB) file, depending on your setup.

Adding a Trigger to a ProcLayer Track

To add a trigger to a ProcLayer track

1. In **Mannequin Editor**, double-click in the new ProcLayer timeline to add a procedural clip. To move the clip, you can drag its starting point.
2. Under **Procedural Clip Properties**, click **Type** and select **Audio**.
3. 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, Mac = 8192, Linux = 8192, iOS = 8192, Android = 4096

s_AudioEventPoolSize

Sets the number of preallocated audio events.

Default values: PC = 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, 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, 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.

Lumberyard provides a character animation system (Animation Editor EMotionFX) that you can use to combine skeletal-based deformation of meshes with morph-based vertex deformation to allow for complex animations. If you are using Lumberyard 1.10 or earlier, you can use the legacy animation system (Geppetto and Mannequin) to accomplish these tasks.

Topics

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

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

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. 206).

Using Character-Specific Shaders

Lumberyard provides the following shaders for use with characters:

- **Eye Shader (p. 1364)** – Renders realistic eyes that take sclera, cornea, iris, and eye moisture properties into account.
- **Hair Shader (p. 1370)** – Renders all character hair, giving the hair different color, stranding, and animation effects.
- **HumanSkin Shader (p. 1372)** – 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

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-up and y-forward (in the direction the character is facing). For more information, see Locomotion Locator Animation Best Practices (p. 324).
- 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.
- Use dual quaternion skinning in all skin-binding procedures. Other methods will create 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.

Transitioning from Legacy (Geppetto and Mannequin) to the Lumberyard Animation System (Animation Editor)

The following topic helps you understand the workflow transitions from the Geppetto and Mannequin systems to the **Animation Editor**.

Topics

> **Note**
> You can't convert your Geppetto and Mannequin assets over to the **Animation Editor**. You must recreate your content for the **Animation Editor**. For more information, see Animation Editor (EMotionFX) (p. 223).

Content Pipeline

Export and Import Content

When working with Geppetto and Mannequin, your assets are exported using the Maya or Max Exporters, or saved as FBX files and imported into Amazon Lumberyard. If you use the **FBX Settings** tool, this creates an `.assetinfo` file that stores the metadata for your FBX import settings.

The **Animation Editor** supports only the FBX workflow. All of your characters and animations must be saved as FBX files, and then loaded into Lumberyard and adjusted through the **FBX Settings** tool. **Animation Editor** uses the `.assetinfo` metadata file for the FBX import settings.

For more information, see Working with the FBX Settings (p. 212).

Content File Formats

The following are the content file formats that are required to work with Geppetto:

- `.skin` – Skinned mesh
- `.chr` – Skeleton
- `.mtl` – Material
- `.i_caf` – Intermediate animation if exported from Maya or Max
- `.caf` – Compressed animation file

The following content file formats are required to work with the **Animation Editor**. These are the only file formats that the **FBX Settings** tool creates for the **Animation Editor**.

- `.actor` – Contains the skeleton and skin information
- `.motion` – Contains the animation
- `.mtl` – Material files are created when the `.fbx` file has at least one material, which is the case for most DCC applications.

> **Note**
> To make changes to the material, go to the **Material Editor**. After you make changes to the material, the `.mtl` file is no longer a child of the `.fbx` file when you view it in the **Asset Browser** and the `.mtl` is a sibling in the source directory of the `.fbx` file. You can make other changes to the `.mtl` with a text editor or the **Material Editor**.

For more information, see Animation Editor File Types (p. 233).

Lumberyard Created Files

In Geppetto, there are additional files that are created in Lumberyard:

- `.animsettings` – Created when importing `.i_caf` file.
- `.CDF` – Character definition file, which is a combination of a `.chr` file and at least one `.skin` and `.mtl` file.
- `.chrparams` – This file works with the `.chr` file to determine where animations are located, IK setup, and so on..
- `SkeletonList.xml` – Used for importing `.i_caf` files.
- `.dba` – Animation databases or sets.
- `.animevents` – Animation events.
- `.bspace` – Blend spaces or parametric blending.
- `.comb` – Combined blend spaces.

In Mannequin, there are additional files that are created in Lumberyard:

- `Preview.xml` – Preview setup file.
- `ControllerDefs.xml` – Controller definitions file.
- `.adb` – Animation database file for Mannequin.
- `FragmentIDs.xml` – Fragment ID file.
- `Tags.xml` – Tag definition file.
- `Sequence.xml` – Animation sequences.

In the **Animation Editor**, the following files are created in Lumberyard:

- `.motionset` – Contains your motions (animations). A set of animations is called a *motion set*.
- `.animgraph` – Contains your animation networks. This is also known as an *animation graph*.
- `.emfxworkspace` – Your workspace file, which stores the character with its reference to the animation graph and motion sets.

Content Pipeline Comparison Table

Geppetto and Mannequin	Animation Editor

Content into Lumberyard
• FBX
  • Creates `.assetinfo` metadata file
• Maya Exporter
• Max Exporter
	Content into Lumberyard
	• FBX
	  • Creates `.assetinfo` metadata file

Geppetto and Mannequin	Animation Editor
Content file formats for Lumberyard • `.skin` • `.chr` • `.mtl` • `.i_caf` • `.caf`	Content file formats for Lumberyard • `.actor` • `.motion` • `.mtl`

Lumberyard created files
• Geppetto
  • `.animsettings`
  • `.chrparams`
  • `.CDF`
  • `SkeletonList.xml`
  • `.animevents`
  • `.bspace`
  • `.comb`
• Mannequin
  • `Preview.xml`
  • `ControllerDefs.xml`
  • `.adb`
  • `FragmentIDs.xml`
  • `Tags.xml`
  • `Sequence.xml`
	Lumberyard created files
	• `.motionset`
	• `.animgraph`
	• `.emfxworkspace`

Character and Animation Setup

Character Setup

In Geppetto, character setup involves creating the `.CDF` (character definition file) that references a `.chr` (skeleton) and has at least one `.skin` (skinned mesh) attached. This is the workflow that you follow to assemble the various parts of an animated character in one place in Lumberyard.

In the **Animation Editor**, your character is the `.actor` that is created from the FBX file that contains a skeleton, bind pose, and skinned mesh information. There is no additional assembly required to create a character in Lumberyard.

Animation Setup

In Geppetto, if you use the Maya or Max exporter to create your animations, you have the additional step of importing the animation to your character in Geppetto to convert the `.i_caf` files to `.caf` files. Also, you must add your character's skeleton to the `SkeletonList.xml` file, so that animations can be imported and used on that specific skeleton.

In this workflow, animations can't be shared across multiple skeletons, even if the skeletons are identical. This results in duplicate animations for multiple characters with identical skeletons. You also need to make a `.chrparams` file, so that the engine can find which folder it can import the animations for that skeleton. You can also set up a `.dba` (database of animations) to load different sets of animations for a character. When you set up animation events, you need an `.animevents` file that was also referenced

from the `.chrparams`. You can then use your character and animations in Geppetto for your animation network setup.

In the **Animation Editor**, the `.motion` file created from the FBX animation file is the base animation, (also called a *motion*) in the new animation system. A `.motion` file is not exclusively tied to a specific character, skeleton, or motion set, and can be shared across multiple `.actor` files and added to multiple motion sets.

A `.motionset` file is a library of `.motion` files that can be loaded to an `.actor` file. A motion set is like an animation database or set. Each actor should have at least one motion set. Motion events are stored in the `.assetinfo` metadata file for the FBX file in which the `.motion` file is created.

Your animation network setup is stored in an `.animgraph` file, which can be a combination of state machines and/or blend trees.

Component Entities

In the Geppetto and Mannequin workflow with the component entity system, you add the **Skinned Mesh (p. 558)** component that references the `.CDF` to your character.

You can apply animations to the entity with the **Skinned Mesh (p. 558)** component in two ways:

- Add the **Simple Animation (p. 550)** component to play animations directly.
- Add the **Mannequin (p. 513)** component to reference the `ControllerDefinition.xml` file and the **Mannequin Scope Context (p. 520)** component to reference the `.adb` file.

In the **Animation Editor** workflow with the component entity system, you add the **Actor (p. 448)** component that references the `.actor` file to your character.

You can apply animations to the entity in two ways:

- Add the **Simple Motion (p. 555)** component to play motions directly.
- Add the **AnimGraph (p. 449)** component to reference an `.animgraph` file.

For more information, see Animation Editor Components (p. 242).

Component Entities Comparison Table

Geppetto and Mannequin	Animation Editor
Component entities • **Skinned Mesh (p. 558)** • **Simple Animation (p. 550)** • **Mannequin (p. 513)** • **Mannequin Scope Context (p. 520)**	Component entities • **Actor (p. 448)** • **Simple Motion (p. 555)** • **AnimGraph (p. 449)**

For a list of components, see Component Reference (p. 446).

Lumberyard Animation System

The Lumberyard animation system (Animation Editor EMotionFX) allows you to build a character by associating one or more skinned models with an animation skeleton. You can then specify animations for the character and blend the animations to create character transitions.

Lumberyard also provides a tool that converts static `.fbx` meshes, skeletons, skins, animations, and materials into Lumberyard assets.

Topics

Working with the FBX Settings

FBX Settings is in preview release and is subject to change.

You can use the **FBX Settings** tool to convert static FBX meshes, skeletons, skins, animations, and materials into Lumberyard assets.

To open the **FBX Settings** tool, right-click on an `.fbx` file in the **Asset Browser** (p. 179). In previous versions of Lumberyard, these settings were located in the **Tools** menu.

When you export or copy `.fbx` files into your game project directory (for example, `\dev\SamplesProject\Objects`), Asset Processor detects the new or modified `.fbx` files in the directory. Lumberyard calculates default settings from the `.fbx` file, which Asset Processor uses to automatically convert the file to the appropriate mesh, skeleton, materials, or animation files.

> **Note**
> The FBX Settings tool processes `.fbx` files only when they are in the current game project directory.

If you update the FBX settings or add modifiers for the `.fbx` file, Lumberyard generates an `.assetinfo` file alongside the `.fbx` file. The `.assetinfo` file stores the configuration and modifier settings that are applied when processing the `.fbx` file. The original `.fbx` file remains intact. Lumberyard includes a sample `.fbx` file in the `\dev\SamplesProject\Objects\Tutorials\Fbx` directory.

Use the **Asset Browser** (p. 179) to access the assets that were successfully processed. To place static mesh files (`.cgf`) in your game level, drag the files from Asset Browser in Lumberyard Editor into the viewport. You can then use skins, skeletons, and animation files to create characters in Geppetto (p. 276).

Topics

Editing the FBX Settings

When you add `.fbx` files to any directory in your game project directory, Asset Processor automatically processes the files using default settings. You can edit the settings for your `.fbx` files at any time from the FBX Settings tool.

To edit the FBX settings

1. In Lumberyard Editor, in the **Asset Browser**, find the `.fbx` file whose settings you want to modify.

   > **Tip**
   > Type **`fbx`** into the Asset Browser's search bar to quickly find `.fbx` files.

Right-click the `.fbx` file and choose **Edit Settings**.

2. For your `.fbx` file, modify the settings or add modifiers as needed:
   - **Meshes** tab – Modify the settings for static meshes (`.cgf`). For information, see Meshes Tab (p. 213).
   - **Rigs** tab – Modify the settings for skeletons (`.chr`) and skins (`.skin`). This tab appears if the scene includes skeletons or skins. For information, see Rigs Tab (p. 214).
   - **Animations** tab – Modify the settings for animations (`.caf`). This tab appears if the `.fbx` file includes keyframes. For information, see Animations Tab (p. 215).
3. Click **Update** to apply your changes and the meshes in your level are automatically updated. Larger files might take longer to process.
4. Review the job status for errors or a success message. Click **OK** to return to the settings.

Meshes Tab

Mesh groups are a collection of meshes in your scene that you want to process. By default, all meshes in your scene are processed; however, you can manually exclude individual meshes within your `.fbx` file. You can also process multiple mesh groups from a single `.fbx` file.

You can do the following on the **Meshes** tab:

- **Add another mesh** – Create a mesh (`.cgf`) from the original `.fbx` file.
- **Name Mesh** – Type a name for the mesh group. This name appears in the Asset Browser for the `.cgf` file.
- **Select meshes** – Choose which meshes to process from your `.fbx` file. You should choose the meshes that you want to render. For example, if your physics proxy mesh is different than your render mesh, deselect the physics proxy mesh from the general meshes to process.
- **Add Modifier** – Add modifiers to change the way the processing of a group occurs. Add one of the following modifiers:
  - **Comment** – Add one or more informational comments for the file. For example, you can provide a comment about the change made to the `.fbx` file for tracking purposes. Comments do not affect how files are processed.
  - **Level of Detail** – Specify a level of detail (LOD). A maximum of five LODs are allowed, and these are numbered [0] to [4], with [0] being the highest level of detail. LODs are optimized meshes with a low polygon count that appears the further the camera moves from the render mesh. Click **Select nodes** to choose which mesh nodes to include for each level of detail.

    > **Note**
    > When authoring the mesh in your 3D application, you can add `_lod1`, `_lod2`, `_lod3`, `_lod4`, `_lod5` as suffixes to your mesh names in order to automatically add a **Level of Detail** modifier and assign the appropriate LOD to the level. `_lod1` is mapped to [0], `_lod2` is mapped to [1], and so on.
  - **Material** – Materials contain the surface properties of a mesh and the generated material file (`.mtl`) stores the materials of the processed mesh. Names of materials in the `.fbx` file are based on the **Name** setting for materials in the `.mtl` file. For example, a mesh processed with a file named `myfile.cgf` would have a corresponding material file named `myfile.mtl`.

    Customize materials with the following settings:
    - **Update Materials** – If enabled, updates an existing `.mtl` file to use the relevant settings from the `.fbx` file. This applies to texture map file names only.
    - **Remove Unused Materials** – If enabled, removes any material that is present in an existing `.mtl` file but not present in the `.fbx` file.
  - **Mesh (Advanced)** – Set the following advanced modifiers for processed mesh files:

  - **32-bit Vertex Precision** – If enabled, sets a higher precision in vertex data. If disabled, the vertex precision is 16-bit. Use this feature if meshes that are big in size appear with jagged artifacts.
  - **Merge Meshes** – If enabled, combines all sub-meshes into a single mesh for optimization. If disabled, treats each sub-mesh separately.
  - **Vertex Color Stream** – If enabled, supports the ability to process vertex coloring. If available, choose a vertex color stream to apply to your mesh and material settings.
- **Origin** – Change the position (translation), orientation (rotation), and scale of a mesh relative to how it was authored. Set the following values:
  - **Relative Origin Node** – Choose the transform relative to which the mesh will be processed. By default, the mesh origin is placed at the scene position 0,0,0 in the `.fbx` file.
  - **Translation** – Set the position offset of the processed mesh.
  - **Rotation** – Set the orientation offset of the processed mesh in degrees.
  - **Scale** – Set the scale offset of the processed mesh.
- **Physics Proxy** – Choose which meshes to use for physics proxies from the `.fbx` file. Physics proxies are meshes that encapsulate render geometry (for example, hit detection or physics collision) and are optimized with a low polygon count for better performance. Primitives such as a cube, sphere or capsule are best for optimal physics performance. You can override physics proxies in the **Physics Meshes** selection window.

  Note
  If your `.fbx` file includes a mesh node with the suffix `_phys`, the mesh node automatically adds a new **Physics Proxy** modifier.

You can delete a modifier or group by clicking the **X** for the entry.

Rigs Tab

Animated characters in a game require a skeleton and skins. Skeletons consist of bones that allow the character to be posed and animated. Skins (or skinned meshes) are geometry that is weighted to bones, allowing the skin to conform. A rig consists of the skeleton and skin files. If an `.fbx` file includes skinned meshes, `.chr` and `.skin` files are automatically created in the appropriate cache directory. If you delete the `.fbx` file, the associated `.chr` and `.skin` files are automatically removed from the cache directory.

Skin groups are a collection of skinned meshes in your scene that you want to process. By default, all skinned meshes in your scene are processed; however, you can manually exclude individual meshes within your `.fbx` file. You can also process multiple skin groups from a single `.fbx` file.

Skeleton groups include the skeleton in your scene that you want to process. By default, Lumberyard detects the first bone in the `.fbx` file hierarchy and sets that bone as the root bone. All child bones are included and processed.

You can do the following on the **Rigs** tab:

- **Add another skin or skeleton** – Add a skin or skeleton group from the original `.fbx` file. Rename the mesh group using the **Name Skins** field.
- Click **X** for an entry to delete a modifier or group.

Skin Group

The **Skin** group includes the following settings:

- **Name Skin** – Type a name for the skin file. This name appears in the **Asset Browser**.
- **Select skins** – Choose which skinned meshes to process from your `.fbx` file. Choose the meshes that you want to render.

- **Add Modifier** – Add modifier to the skin group for additional configuration settings. Add one of the following rules:
  - **Comment** – Add one or more informational comments for the file. For example, you can provide a comment about the change made to the `.fbx` file for tracking purposes. Comments do not affect how files are processed.
  - **BlendShapes** – Choose which blendshapes to process from your `.fbx` file. Blendshape nodes that are exported with the `.fbx` file are automatically detected and applied as a modifier for the skin group.
  - **Level of Detail** – Specify a level of detail (LOD). A maximum of five LODs are allowed, and these are numbered [0] to [4], with [0] being the highest level of detail. LODs are optimized meshes with a low polygon count that appears the further the camera moves from the render mesh. Click **Select nodes** to choose which mesh nodes to include for each level of detail.

    Note
    When authoring the mesh in your 3D application, you can add `_lod1`, `_lod2`, `_lod3`, `_lod4`, `_lod5` as suffixes to your mesh names. Doing this automatically adds a **Level of Detail** modifier and assigns the appropriate LOD to the level. `_lod1` is mapped to [0], `_lod2` is mapped to [1], and so on.
- **Material** – Materials contain the surface properties of a mesh and the generated material file (`.mtl`) stores the materials of the processed mesh. Names for materials in the `.fbx` file are based on the **Name** setting for materials in the `.mtl` file. For example, a mesh import with a file named `myfile.cgf` would have a corresponding material file named `myfile.mtl`. Customize processed materials with the following settings:
  - **Update Materials** – If enabled, updates an existing `.mtl` file to use the relevant settings from the `.fbx` file. This applies to texture map file names only.
  - **Remove Unused Materials** – If enabled, removes any material that is present in an existing `.mtl` file but not present in the `.fbx` file.
- **Skin (Advanced)** – Set the following advanced modifiers for processed skinned mesh files:
  - **32-bit Vertex Precision** – If enabled, sets a higher precision in vertex data. If disabled, the vertex precision is 16-bit. Use this feature if meshes that are big in size appear with stair-like artifacts.
  - **Vertex Color Stream** – If enabled, supports the ability to process vertex coloring. If available, choose a vertex color stream to apply to your mesh and material settings.

Skeleton Group

The **Skeleton** group includes the following settings:

- **Name Skeleton** – Type a name for the skeleton file. This name appears in the **Asset Browser** for the `.chr` file.
- **Select root bone** – Select the default root bone. The top parent of the skeleton is selected as the default bone. All the child bones of the root bone are processed.
- **Add Modifier** – Add modifiers to the skeleton group for additional configuration settings. Add one of the following modifiers:
  - **Comment** – Add one or more informational comments for the file. For example, for tracking purposes you can provide a comment about a change made to the `.fbx` file. Comments do not affect how files are processed.
  - **Skeleton Proxies** – The **FBX Settings** tool does not yet support skeleton proxies. To create joint mesh proxies, see Creating Joint Mesh Proxies (p. 362).

Animations Tab

Lumberyard generates animations from key-frame data on bones.

You can do the following on the **Animations** tab:

- **Add another animation** – Create animation skeleton key-frame data (`.caf`) from the original `.fbx` file.
- **Group name** – Type a name for the animation group. This name appears in the **Asset Browser** for the `.caf` file.
- **Select root bone** – View the default root bone, which is the first bone in the hierarchy. You can also manually choose another bone as the root of your animated skeleton. You can only pick one root bone to use per skeleton group.
- **Start frame** – Set the first key frame of the animation to process.
- **End frame** – Set the last key frame of the animation to process.
- **Add Modifier** – Add a modifier to the processed animation:
  - **Comment** – Add one or more informational comments for the file. For example, you can provide a comment about the change made to the `.fbx` file for tracking purposes. Comments do not affect how files are processed.

  To delete a modifier or group, click the **X** for the entry.

Best Practices for Creating and Exporting Meshes

To maximize the benefits of the **FBX Settings** tool, learn the recommended best practices for creating and exporting skinned meshes.

Creating Skinned Meshes Actors for the Animation Editor

Use the following best practices when you create your character for the Animation Editor. In Lumberyard a character is a skinned mesh.

Setting up the World Coordinate System and Root Joint

If you use the y-up or z-up world coordinate system in your DCC, use the following guidelines for setting up your character:

- Use a root joint for your skinned mesh. This is required to ensure that motion extraction works properly in the **Animation Editor**.
- Do not use transforms, groups, or parent nodes in the hierarchy above your root joint. The root joint must be the top parent of the skeletal hierarchy to ensure that motion extraction works properly.
- Set the root joint position at the origin: 0,0,0.
- Set the root joint rotation and orientation to 0,0,0.
- Orient your character so that the front orthographic camera view shows the front of your character.
- Ensure that your character faces the positive y direction in Lumberyard. Asset Processor automatically adds a **Coordinate system change** rule in the `.fbx` settings. The default value for **Facing Direction** is **Rotate 180 degrees around the up axis**. This enables the game entity's forward direction and character to point in the same direction.
- If your character faces the negative y direction, right-click your `.fbx` file in the **Asset Browser** and choose **Edit Settings**. For the **Coordinate system change** rule, set **Facing Direction** to **Do Nothing**.
- Use the same rule for the actor and each actor's motions. This is required to ensure that animations work properly in the **Animation Editor**.

Setting up Skin Binding

Observe the following best practices when you set up skin binding:

- Delete the geometry history on your mesh before skinning the geometry to joints.

- When skinning your mesh, limit your maximum influences per vertex to four. Lumberyard currently supports only four weight influences per vertex using the `.fbx` pipeline.
- Skin bind your mesh at the origin and in the same forward direction as the root joint.
- Check the bind pose before exporting your skinned mesh. For example, if the mesh moves after unbinding, you must reskin the mesh in order to prevent any errors in Lumberyard.
- Ensure that your skinned mesh has one bind pose in your DCC before you export to an `.fbx` file.
- Do not include any static meshes with your skinned meshes. Lumberyard cannot render unskinned meshes that are parented to bones.

Do the following to reskin your mesh:

To reskin your mesh in Maya

1. For **Menu Set**, choose **Rigging**.
2. Click **Skin**, **Unbind Skin**.
3. In the **Detach Skin Options** dialog box, for **History**, choose **Delete history**. Click **Detach** and then click **Close**.
4. Move or rotate your mesh to the appropriate position.
5. Click **Modify**, **Freeze Transformations**.
6. In the **Freeze Transformations Options** dialog box, select the **Translate**, **Rotate**, and **Scale** check boxes. Click **Freeze Transform** and then click **Close**.
7. Select your mesh.
8. Choose **Edit**, **Delete by Type**, **History**.
9. In the outliner, select the bones that you want skinned. Hold **Ctrl** and select the mesh that you want skinned.
10. Click **Skin**, **Bind Skin**.
11. In the **Bind Skin Options** dialog box, for **Bind to** choose **Selected joints**. For **Bind method**, choose your preferred bind method. Ensure that the **Skinning method** is set to **Dual quaternion**. Click **Bind Skin** and then click **Close**.

    Note
    You must export and save your deformer weights in order to import the weight maps after reskinning.

To reskin your mesh in 3ds Max

1. Select your mesh.
2. On the **Modify** tab, expand **Advanced Parameters**. Click **Save**.
3. Save the mesh envelope (skin).
4. On the **Modify** tab, select the skin. Right-click the skin and choose **Delete**.
5. On the **Utilities** tab, click **Reset XForm**.
6. Click **Reset Selected**.
7. Click **Collapse**.
8. Click **Collapse Selected**.
9. On the **Modify** tab, right-click **Editable Mesh**, and choose **Editable Poly**.
10. For **Modifier List**, choose **Skin**.
11. Under **Advanced Parameters**, click **Load** and choose the mesh envelope (skin) that you saved earlier.

    Note
    You must save your weights in order to load the weights after adding a new skin modifier.

12. In the **Load Envelopes** dialog box, click **Match by Name** to preserve the weights that you saved earlier. Close the dialog box.

If you want to add a root joint to a skinned mesh, follow the preceding steps to unbind and reskin your mesh.

Exporting Skinned Meshes Best Practices

Use the following best practices when you export your skinned meshes using the **FBX Settings** tool.

- If you use the z-up world coordinate system, use the following guidelines:
  - Ensure that your DCC scene is set to z for the up-axis world coordinate system.
  - Set the axis conversion up axis to z when exporting an `.fbx` file.
- If you use the y-up world coordinate system, use the following guidelines:
  - Ensure that your DCC scene is set to y for the up-axis world coordinate system.
  - Set the axis conversion up axis to y when exporting an `.fbx` file.
- Export your skinned mesh at the bind pose. After the `.fbx` file is processed, an actor file is created that you can use in the **Animation Editor**.
- If you use Maya or 3ds Max, do the following:
  - Select the **Animation** check box if you are exporting animations.
  - Select the **Bake Animation** check box if you are exporting animations.
  - Select the **Skins** and **Deformed Models** check boxes if you are exporting skinned meshes. Select the **Blend Shapes** check box if you are exporting blend shapes.
- Use the following guidelines for vertex count:
  - Asset Processor processes your `.fbx` file to create an `.actor` file. Occasionally the number of vertices does not match the original vertex count of the `.fbx` file. Lumberyard allows a maximum of 65,536 vertices for skinned geometry after asset processing. Although the maximum number of vertices is 65,536, the original vertex count from your DCC tool cannot be up to 65,536 vertices. As a best practice when modeling your geometry, divide 65,536 by 3 for a value of 21,845 vertices. After you export the `.fbx` file, verify that the skinned mesh was successfully processed. If Asset Processor does not successfully export the skinned mesh, reduce the number of vertices in the model until the file is successfully processed.
  - Be aware that the more vertices in an `.fbx` file, the longer Asset Processor takes to process it. View Asset Processor often and check for errors.
- Export only the skeleton and mesh. Do not use transforms, groups, or parent nodes in the hierarchy above your root joint. The root joint must be the top parent of the skeletal hierarchy to ensure that motion extraction works properly.
- Remove unused geometry, bones, vertices, materials, and nodes that are not necessary for the `.fbx` asset. This reduces the processing time and offers a better chance that the automatic processing works properly without making adjustments later. In your DCC, consider naming nodes with `_ignore` as a suffix to prevent those nodes from being processed.

Exporting Static Meshes Best Practices

Use the following best practices when you export your static meshes using the **FBX Settings** tool.

- Ensure that each object that needs a run-time collision has a physics mesh. Low-resolution physics meshes work better than high-resolution meshes. Primitives such as a cube, sphere, or capsule are best for optimal physics performance.
- The maximum number of vertices for any static geometry is 65,536. You can export a scene where the total number of vertices exceeds 65,536, but each static geometry piece cannot exceed 65,536. If the combined mesh is larger than 65,536, you must make the following modification in **FBX Settings**:

To use a combined mesh that has more than 65,536 vertices

1. In Lumberyard Editor, in the **Asset Browser**, open the **Mesh** tab.
2. Choose **Add Modifier**, **Mesh (Advanced)**.
3. Clear the **Merge Meshes** setting.

This prevents Asset Processor from merging the meshes, which allows the geometry to be processed.

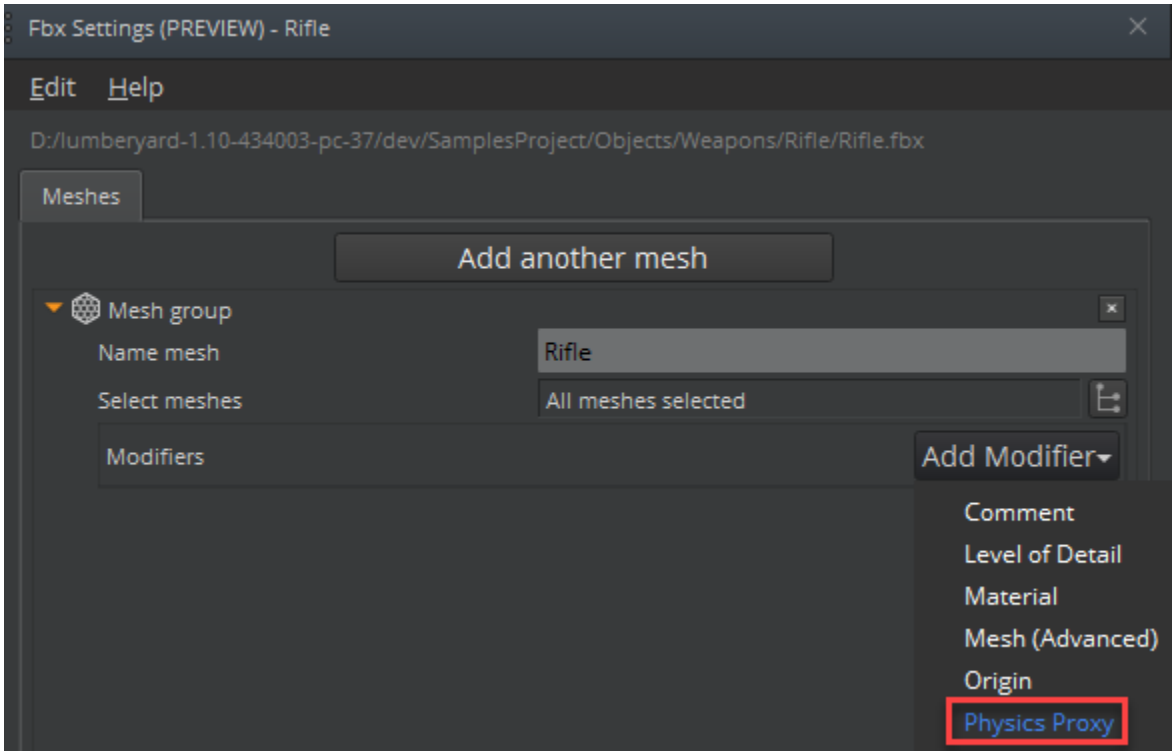
Creating Physics Proxies for Static Meshes

A physics proxy is a mesh that encapsulates physics geometry, such as hit detection and the ability to collide. Physics proxies typically use primitive meshes or meshes with a low polygon count for better performance. Primitive meshes such as cubes, spheres, or capsules work best to ensure optimal physics performance.

You have two ways to create a physics proxy for your static mesh (`.cgf` file). You can use the **FBX Settings** tool to manually create the physics proxy. Or you can use soft naming conventions, which causes Asset Processor to automatically create the physics proxy during processing. For the latter process, see Creating Physics Proxies Automatically (p. 220).

To manually create a physics proxy modifier using the FBX Settings tool

1. In the **Asset Browser**, right-click the `.fbx` file for which you want to create a physics proxy. Choose **Edit Settings**.
2. In the **FBX Settings** tool, on the **Meshes** tab, click **Add Modifier**. Choose **Physics Proxy**.



3. Click the hierarchy button  (to the right of the **Physics meshes** box). Select the meshes that you want to use for the physics proxies. Click **Select**.

4. The recommended best practice for this procedure is to prevent the selected physics mesh from also rendering as a static mesh. To do this, follow these steps:

   a. Beneath **Mesh group**, next to **Select meshes**, click the hierarchy button  .

   b. Deselect the mesh that you used for the physics proxy. Click **Select**.

      This prevents the mesh that you selected for the physics proxy from also rendering as a static mesh.

5. Click **Update**.

   When the **File progress** informs you that the job is complete, click **OK**.

Once you have assigned a physics proxy mesh to your static mesh, the static mesh is ready to display physics and collision behavior.

Creating Physics Proxies Automatically

An alternative to the manual creation of the physics proxy is to use soft naming conventions (p. 222). To do this, rename your static mesh `.fbx` file and add `_phys` as a suffix to the file name. For example, `wall_phys.fbx`.

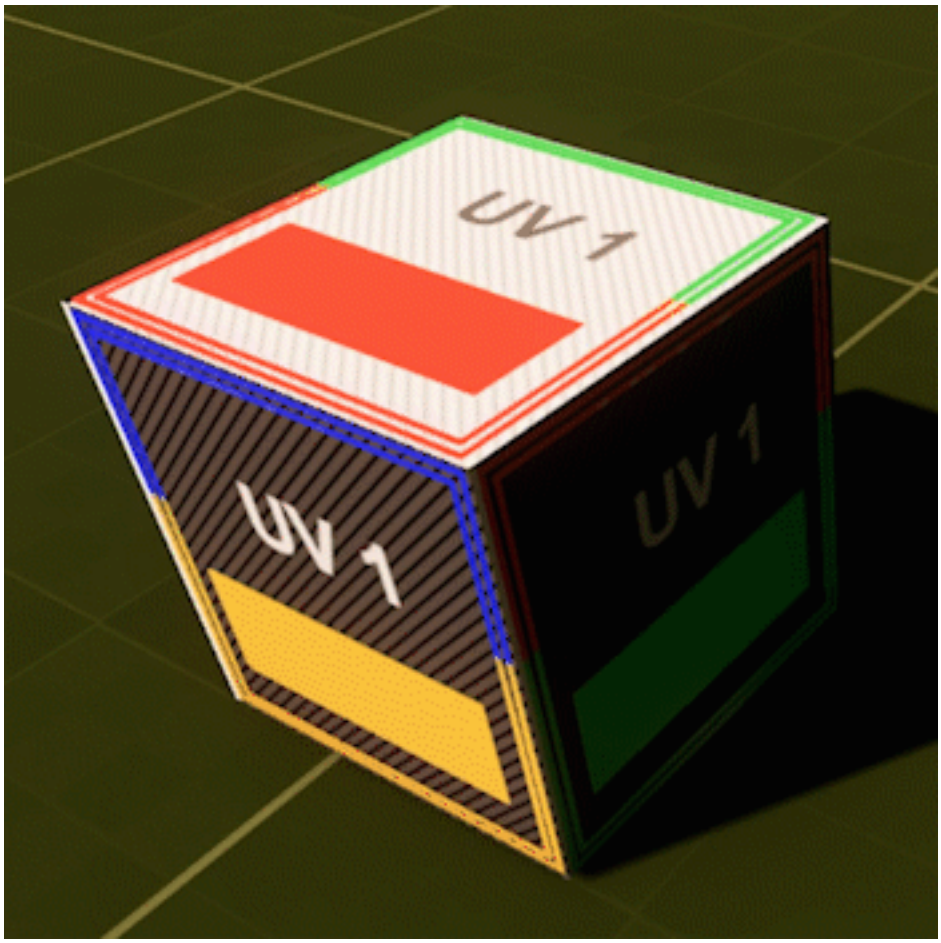
Adding the `_phys` suffix to your file name causes Asset Processor to automatically add a new **Physics Proxy** modifier. You can view this physics proxy modifier when you open the **FBX Settings** tool for an `.fbx` file with a `_phys` suffix.

Asset Processor also automatically deselects the mesh that is used as the physics proxy in the **Select Meshes** property. This prevents rendering of the physics mesh as a static mesh.

Only one physics proxy modifier is automatically created for each node that has a `_phys` suffix.

Using Multiple UV Channels

You can use the **FBX Settings** tool to import multiple UV channels. That way, you can apply to your geometry a detail or blend layer map with UV channels that are independent of the diffuse, normal, and spec channels. With multiple UV channels you can also apply an animated emittance glow that is independent of other texture maps on a mesh.



Topics

Processing Multiple UV Channels

When a new `.fbx` file is added to your project directory, **Asset Processor** automatically detects the file and imports it as a `.cgf`. If needed, you can use your DCC tool to rebuild the UV channels.

The **FBX Settings** tool follows these rules:

- If the `.fbx` file contains one UV channel on a mesh, that channel is automatically imported.
- If the `.fbx` file contains two UV channels on a mesh, both channels are automatically imported.
- If the `.fbx` file contains three or more UV channels on any of the meshes, the first two channels are automatically imported and the remaining channels are ignored.

Setting up Multiple UV Materials

By default, the **FBX Settings** tool creates materials for your imported mesh. You must use the **Shader Generation Parameters** in the **Material Editor** to specify how the material should use the second UV channel.

To apply the second UV channel

1. In Lumberyard Editor, choose **Tools**, **Material Editor**.
2. In the left pane, navigate to and select the material that you want to use.
3. In the right pane, under **Shader Generation Params**, do the following to apply the second UV channel to one of these features:
   - Blend Layer – Select **Blendlayer** and **Use uv set 2 for blendlayer maps** to apply the second UV channel to the **Second Diffuse Map**, **Specular**, **Height**, **Bump**, and **Blending Map** texture slots.
   - Detail Map – Select **Detail mapping** and **Use uv set 2 for detail map** to apply the second UV channel to the **Detail** texture slot.
   - Emittance Map – Select **Use uv set 2 for emittance map** to apply the second UV channel to the **Emittance** and **Decal** texture slots.

Using Independent Tiling

On the **Second Diffuse Map**, you can set independent values for the blend layer's texture inputs (tiling, rotation, and oscillation). Changes to these values do not affect the tiling, rotation, and oscillation values on the first **Diffuse Map**. In addition, these values are not applied to the **Blend Map**. You can use the **Blend Mask Tiling** parameter under **Shader Params** in order to tile the **Blend Map**.

You can also set independent values for tiling, rotation, and oscillation for the **Detail** and **Emittance** map features.

To set independent values for the texture inputs

1. In Lumberyard Editor, choose **Tools**, **Material Editor**.
2. In the left pane, navigate to and select the material that you want to use.
3. In the right pane, do the following:
   - Under **Texture Maps**, modify the values for **Tiling**, **Rotator**, and **Oscillator** for your diffuse maps.
   - Under **Shader Params**, modify the **Blend Mask Tiling** parameter for your blend map.

FBX Soft Naming Conventions

You can use soft naming conventions when authoring assets in your digital content creator (DCC), such as Max or Maya. Soft naming conventions are suffixes that you add to either your scene's DCC node or to the `.fbx` file name. In Lumberyard, these soft naming conventions are recognized by Asset Processor, which applies an action based on the soft naming convention used.

Lumberyard provides soft naming conventions as a convenience for content creators to automate steps that would typically be done manually in the **FBX Settings** tool. Depending on the soft naming conventions that you use, some modifiers are automatically added to the scene settings. The following table lists current soft naming conventions and the actions that are applied.

Soft Naming Conventions

Soft Naming Convention	Method	Example	Asset Processor Action
`_phys`	Apply suffix to node name	`legMesh_phys`	Nodes with the `_phys` suffix are treated as a physics proxy. A physics proxy modifier is automatically added.

Soft Naming Convention	Method	Example	Asset Processor Action

`_lod1`
`_lod2`
`_lod3`
`_lod4`
`_lod5`
	Apply suffix to node name	`jack_lod1`
		`jack_lod2`
		`jack_lod3`
		`jack_lod4`
		`jack_lod5`
			Nodes are treated as level of detail (LOD) meshes. The scene settings automatically add an LOD modifier with each suffixed node.

`_ignore`	Apply suffix to node name	`jackSkeleton_ignore`	Nodes with `_ignore` suffix are ignored and not processed by **Asset Processor**.
`_anim`	Apply suffix to filename	`jackMoves_anim.fbx`	For `.fbx` file names with the `_anim` suffix, only the animations in the `.fbx` are processed.

Animation Editor (EMotionFX)

Animation Editor is in preview release and is subject to change.

Use the **Animation Editor** (**EMotionFX**) to animate characters in Amazon Lumberyard. To build a character, you associate one or more skinned models with an animation skeleton (built in a digital content creation tool like Maya). You then import the character into the **Animation Editor** and specify the animations that you want your character to have.

You can then blend animations, so that your character transitions from one animation to another, and specify the conditions when an animation occurs for a character. For example, you can specify that your character starts in an idle position. After several seconds, the character starts to walk, run, and then slow down again until the character returns to an idle position.

In the **Animation Editor**, you can preview the animations and blends between the animations for your characters.

Topics

Animation Editor Concepts and Terms

The following concepts and terms are used in the **Animation Editor**:

Topics

Actor

A character with at least one bone is called an *actor*. An actor consists of a set of nodes in a hierarchy. Each node is a transformation (position, rotation, and scale) and can contain a mesh.

Instances of actors are called *actor instances*. For example, one soldier that is instanced 100 times can create an army of the same character. You can animate actor instances separately, so that each instance behaves differently. Each actor instance has unique transformations, but shares the same hierarchy as the actor from which it was instanced.

Actor files have the `.actor` extension (for example, `hero.actor`).

Motions

Motions are individual animation clips, such as walk loop, an idle motion, and so on. Motions contain transformation animation and/or morph target weight animation. A motion can contain animation data for the skeletal structure and the morph targets.

Motion files have the `.motion` extension (for example, `Walk.motion`).

Animation graphs

Animation networks are called animation graphs. Animation graphs contain the state machines, transitions, conditions, blend trees, and other nodes. Animation graphs are hierarchical.

Animation graph files have the `.animgraph` extension (for example, `Main.animgraph`).

Motion sets

A motion set contains a list of motions, where each motion has a unique string ID (for example, `walk`, `idle`, and so on). Nodes inside an animation graph can reference motions based on their string ID.

Motion sets can be hierarchical. A child set can override specific motions for a character, while sharing the rest of the motions with the parent set.

Motion set files have the `.motionset` extension (for example, `MainSet.motionset`).

Motion events

Markings at specific time values inside the motion files are called *motion events* (also called notifiers or notifications). A motion event has a type string (for example, "SOUND") and parameter string (for example, "Footstep").

Motion events can have a fixed time value or range. Events with a single time value are called *tick events*. Events with specified start and end times are called *ranged events*.

You can specify event presets, which are pre-setup types of events that you drag and drop into the event tracks. An *event track* is a group of events that you enable or disable. For example, you can add all sound effects to an event track specifically for sounds.

Synchronization

You can use *full clip-base sync* or *sync tracks* to synchronize motion clips to keep a character's motions in sync while blending. For example, if your character is running, synchronization helps keep the right and left feet in sync.

Full clip-based sync warps the motions so that there is a constant change in playback time of the child motion.

Sync tracks use motion events, where the events mark specific moments (for example, where the right and left feet are on the floor). This system is also known as phase matching and dynamically controls the playback speed.

Floats

Floats are numbers with decimals (for example 1.35 or 1.0.) Booleans and integers are also floats, so they can be passed as weight float inputs to blending nodes. If floats are rounded, the **Animation Editor** always rounds them down. For example, 2.99 becomes 2.

Time

All time values and durations are in seconds. For example, you can set the transition time for 0.3 or 300 ms.

About Animation Graphs

Contents

Animation graphs define the animation behavior for your game characters. Animation graphs contain the states that the character can have and define the transitions between these states. Each transition can have a set of conditions that define the logic behind the transitions.

Animation graphs contain nodes and the connections between them. These connections define how data is passed between the nodes or how transitions occur between nodes.

Animation graphs have two main node types:

- State machines
- Blend trees

Because animation graphs are hierarchical, the nodes can be nested. For example, you can have a state machine inside a state machine inside a blend tree, which contains another blend tree and state machine, and so on. The number of hierarchy levels is limitless, but as a best practice limit your hierarchy to 20 levels or fewer.

Each animation graph has one root node, which is a state machine. This root node is the default and cannot be deleted. A simple animation graph can contain one state inside this root state machine. For example, the single state can be a motion node, which outputs a pose that is applied to the character, such as an idle motion.

Before you can add nodes to an animation graph, you must create a motion set. After you create a motion set, you can create an animation graph and then assign the motion sets to the animation graph using the **Resource Management** pane in the **Anim Graph** window.

Animation Graph Nodes

In a state machine, you can add four nodes from the **Sources** category:

- **State Machine**
- **Blend Tree**
- **Motion**
- **Bind Pose**

In a blend tree, you can add other nodes from the following six categories

1. **Sources**
   - **State Machine**
   - **Blend Tree**
   - **Motion**
   - **Parameters**
   - **Bind Pose**
   - **Motion Frame**
2. **Blending**
   - **Blend Space 2D**
   - **Blend Space 1D**
   - **Blend Two**
   - **Blend N**
   - **Pose Mask**
3. **Controllers**
   - **TwoLink IK**
   - **LookAt**
   - **Transform**
   - **Accum Transform**
4. **Logic**
   - **Float Condition**
   - **Float Switch**
   - **Bool Logic**
   - **Pose Switch**
5. **Math**
   - **Float Math1**
   - **Float Math2**
   - **Vector3 Math1**
   - **Vector3 Math2**
   - **Vector2 Decompose**
   - **Vector3 Decompose**
   - **Vector4 Decompose**
   - **Vector2 Compose**
   - **Vector3 Compose**
   - **Vector4 Compose**
   - **Smoothing**
   - **Range Remapper**
   - **Direction to Weight**
6. **Misc**
   - **Mirror Pose**

About Parameters

Contents

- Adding a Parameter Node to a Blend Tree (p. 228)

When you create your animation graph, you can use parameters to control how your animations transition between different states.

Each transition can have a set of conditions applied to it. These conditions define the logical rules for the transition and how animations blend together.

Each transition condition is controlled by a set of parameters. Your Lumberyard game setup sends parameter values to the animation graph. The actor reacts to the incoming parameters. The game sends the parameter values to the animation graph, which then responds to the changes automatically. For example, you can specify parameter values such as speed, direction, weapon type, and so on.

You can set this up in a game level by adding an **Actor** and an **Animation** component to an entity with the **Entity Inspector**.

For more information, see Animation Editor Components (p. 242).

Adding Parameters to an Animation Graph

You can add parameters to an animation graph in the **Parameters** pane.

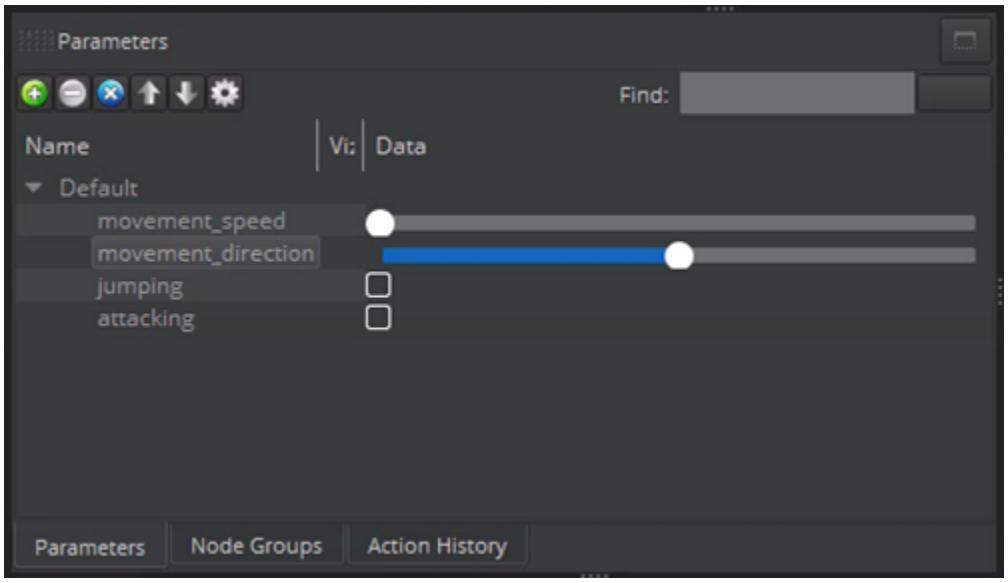
To add a parameter to an animation graph

1. In Lumberyard Editor, choose **Tools**, **Animation Editor**.
2. In the **Parameters** pane, click the green **+** icon.
3. In the **Create Parameter** dialog box, specify the parameter name, description, and the value type.

   You can specify the following value types to provide input into animation graph nodes:

   - **FloatSlider**
   - **Checkbox**
   - **FloatSpinner**
   - **IntSpinner**
   - **IntSlider**
   - **Vector2**
   - **Vector3NoGizmo**
   - **Vector4**
   - **String**
   - **Color**
   - **Vector3**
   - **Rotation**
   - **Tag**

You can name parameter types to identify the purpose of the control. For example, you can name parameters such as `movement_speed`, `movement_direction`, `jumping` and `attacking`. As an artist and game designer, you can specify the parameters that best control your animation graph.



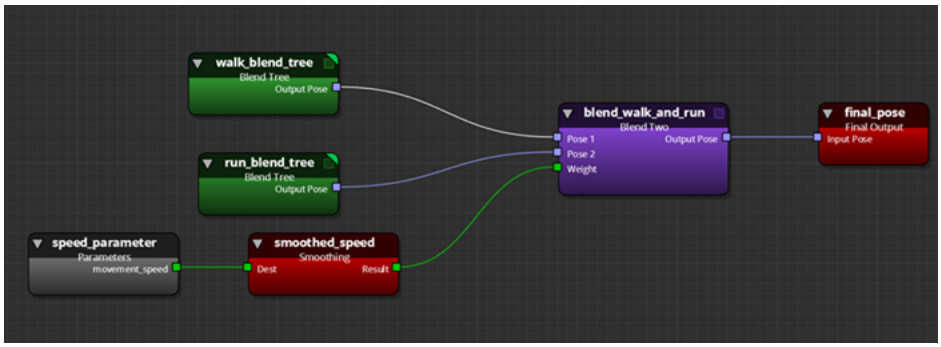
Adding a Parameter Node to a Blend Tree

After you create your parameters in the **Parameter** pane, you can add a parameter node to your blend tree.

To add a parameter node to your blend tree

1. In the **Animation Editor**, right-click the animation graph grid and choose **Create Node**, **Sources**, **Parameters**.
2. In the **Attributes** pane, click **select parameter** and specify the parameter that you want.

You can rename parameter nodes and specify them to provide input to other nodes. In the following example, the **speed_parameter** node provides input to the blend tree.



About Motion Sets

A motion set is a collection of motions, where each motion refers to a specific motion file and is identified by a string ID, such as `idle_motion1`. When you create motion nodes, you specify the string ID for the motion, not the motion file itself. You can use different motion sets in combination with the same animation graph. For example, you can create an animation graph to define animation behavior for a controllable human character and apply the same animation graph to a frog. Because frog movements differ from a human character's, you specify different motion sets for the frog. You can share animation graphs for your characters; you don't need to create unique animation graphs for each character type.

A combination of an animation graph with a specified motion set being applied to a given actor instance is called an *animation graph instance*. Each animation graph instance has a unique set of parameter values. For example, an army of 100 soldiers is controlled by 100 different animation graph instances, which allows you to animate each soldier independently.

Motion sets can also be hierarchical. Child motion sets can override certain motions from their parents. When you apply a child motion set to a character, the character uses all the motions shared by the parent except for the motions that are specified for the child motion set. For example, you can have a character that shares 90% of the same motions of the parent, but has custom motions specific to that character.

About State Machines

State machines contain a set of states that are linked together by transitions. A transition goes from one node to another node and has properties, such as the time it takes to make the transition. A blend between the outputs of both states is performed during the transition, when the animation moves from one state to another.

Transition conditions are conditions that are linked a given transition. For example, they can compare a parameter value against another value to see if the given parameter is bigger than the specified value. If the condition is met, this signals the trigger for the transition. For example, if the speed parameter is greater than 0, a character transitions from an idle to a run state. You can apply multiple conditions to a single transition. The transition occurs only when all conditions are met.

Animation Editor User Interface

Animation Editor is in preview release and is subject to change.

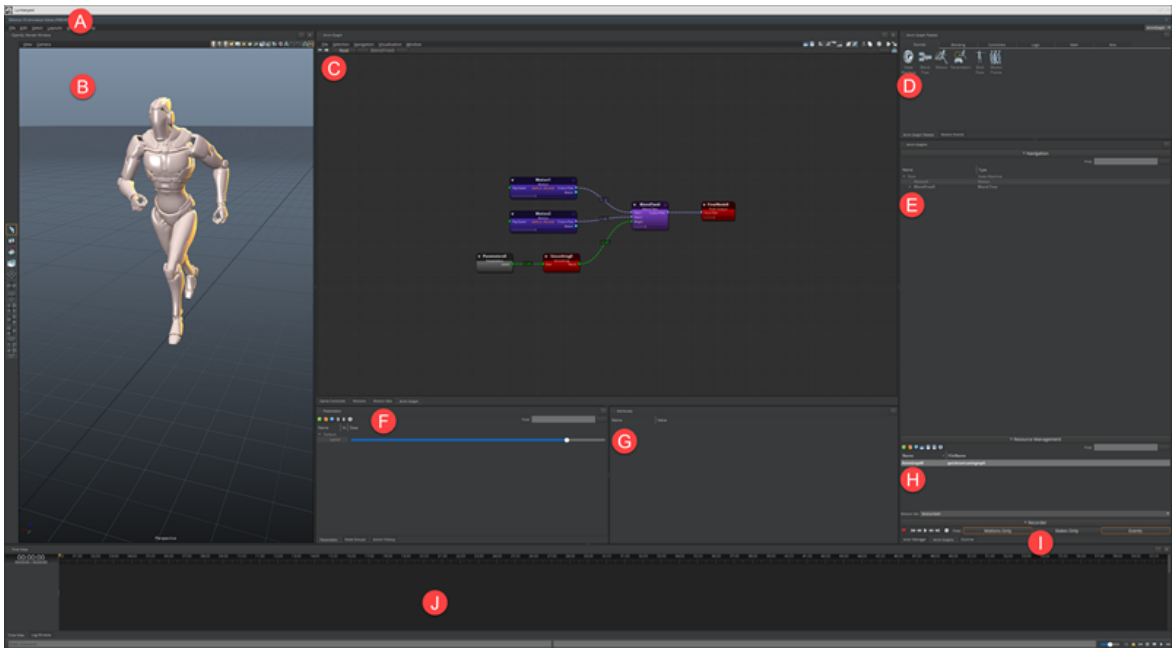
Topics

Opening Animation Editor

You can open the **Animation Editor** from Lumberyard Editor.

To open the Animation Editor

- From Lumberyard Editor, choose **Tools**, **Animation Editor**.



On the **Animation Editor**, you can do the following:

A. From the menu bar, you can do the following:
- Open actor files.
- Open, create, and save your workspaces.
- Select and deselect actor instances.
- Change the layout view of the editor.
- Display or hide menus for the **Animation Editor**.

B. View your character in the perspective window.

C. **Anim Graph** – View and edit your animation graph to define your character's behavior.

D. **Anim Graph Palette** – Drag and drop nodes into the animation graph grid.

E. **Anim Graphs Navigation** – View the hierarchy of your nodes.

F. **Parameters** – View, add, edit, or delete parameters for the animation graph.

G. **Attributes** – View, add, edit, or delete attributes for the selected node in the animation graph.

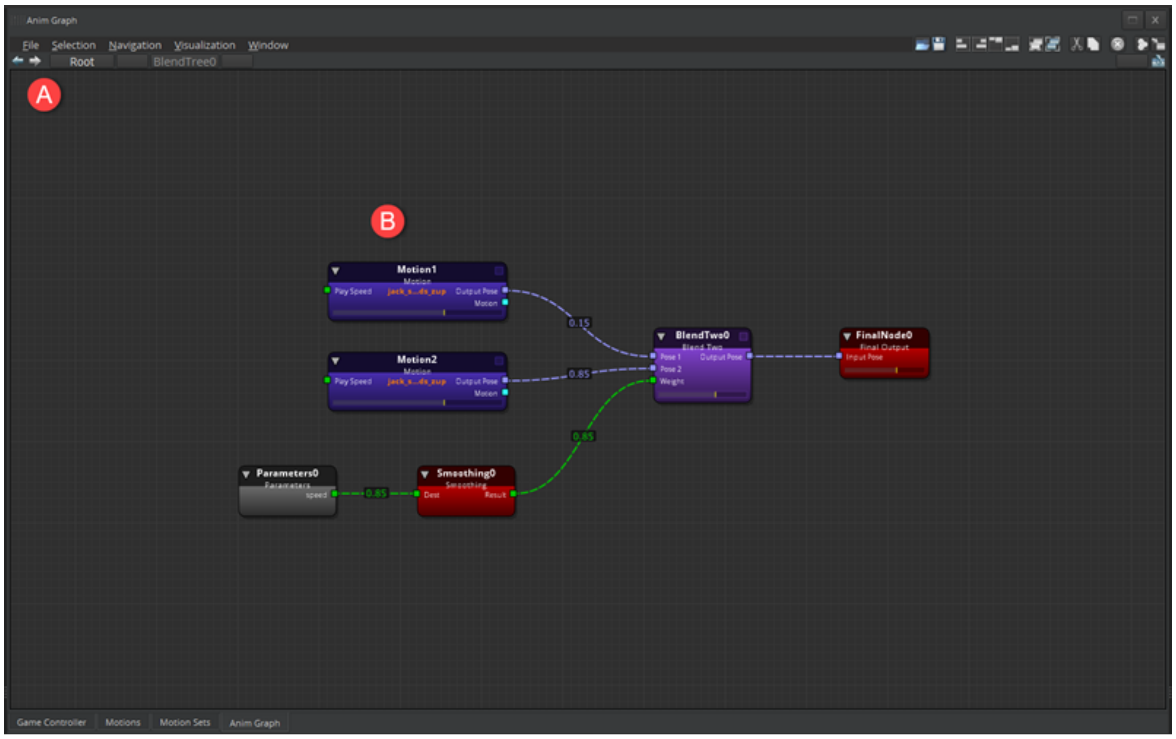
H. **Resource Management** – Add, load, save, and delete animation graphs.

I. **Recorder** – View what triggers the recording and manage the playback options.

J. **Time View** – View recorded debug information and edit event tracks of the selected motion.

Anim Graph Interface

Use the animation graph to create nodes and specify parameter and values for each node.



On the animation graph, you can do the following:

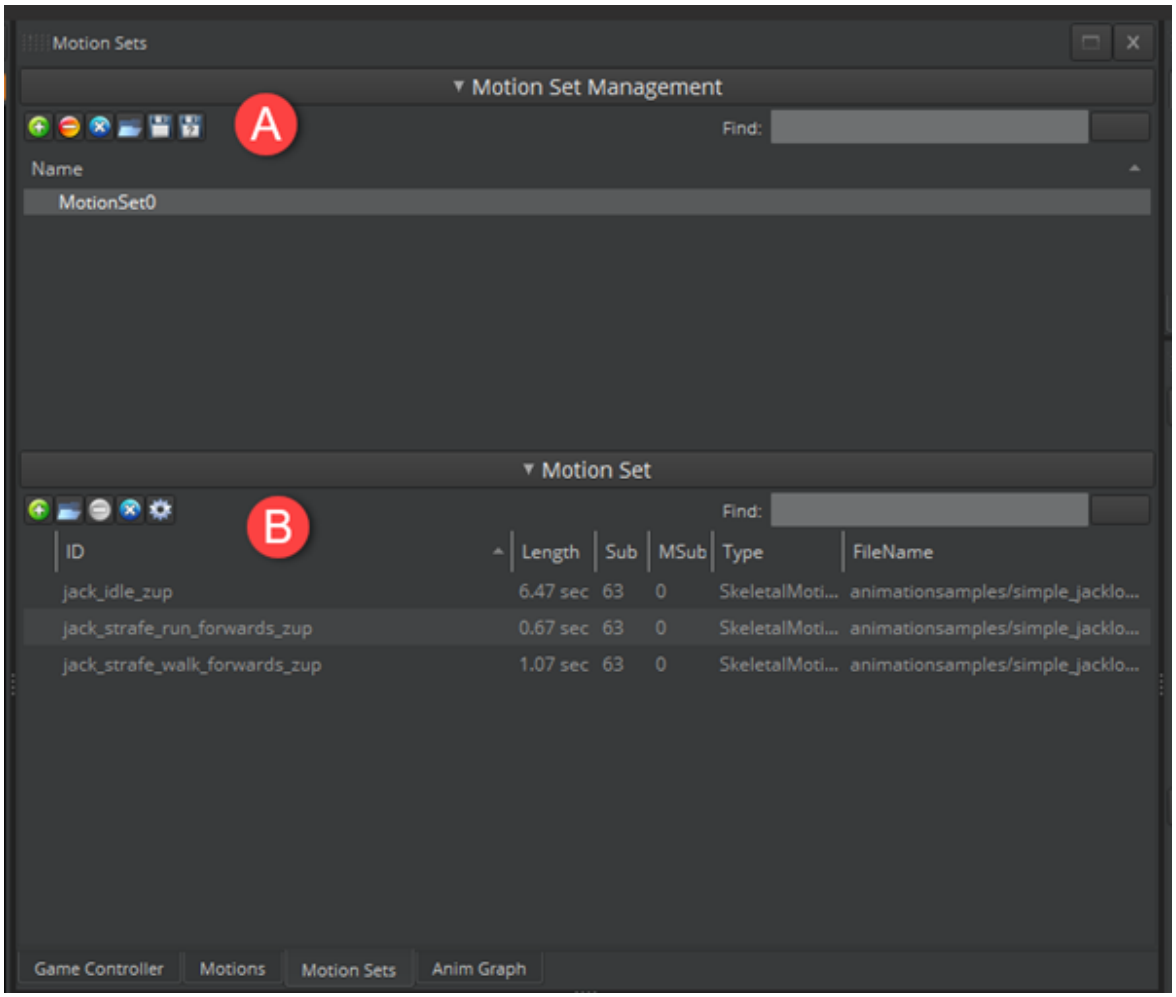
A. From the menu bar, you can do the following:

- Open and save your animation graphs.
- Select and deselect nodes.
- Change the layout view of the editor.
- Display or hide windows for animation graph.

B. In the grid, create nodes and connect them with connectors.

Motion Sets Interface

Use the **Motion Set** pane to create motion set files and add motion files to it.



In the **Motion Sets** pane, you can do the following:

A. **Motion Sets Management** – Add, open, save, or delete motion sets.
B. **Motion Sets** – Add, open, or delete the motion files in the selected motion set.

Motions Interface

Use the motion pane to add and manage motion files in a motion set.

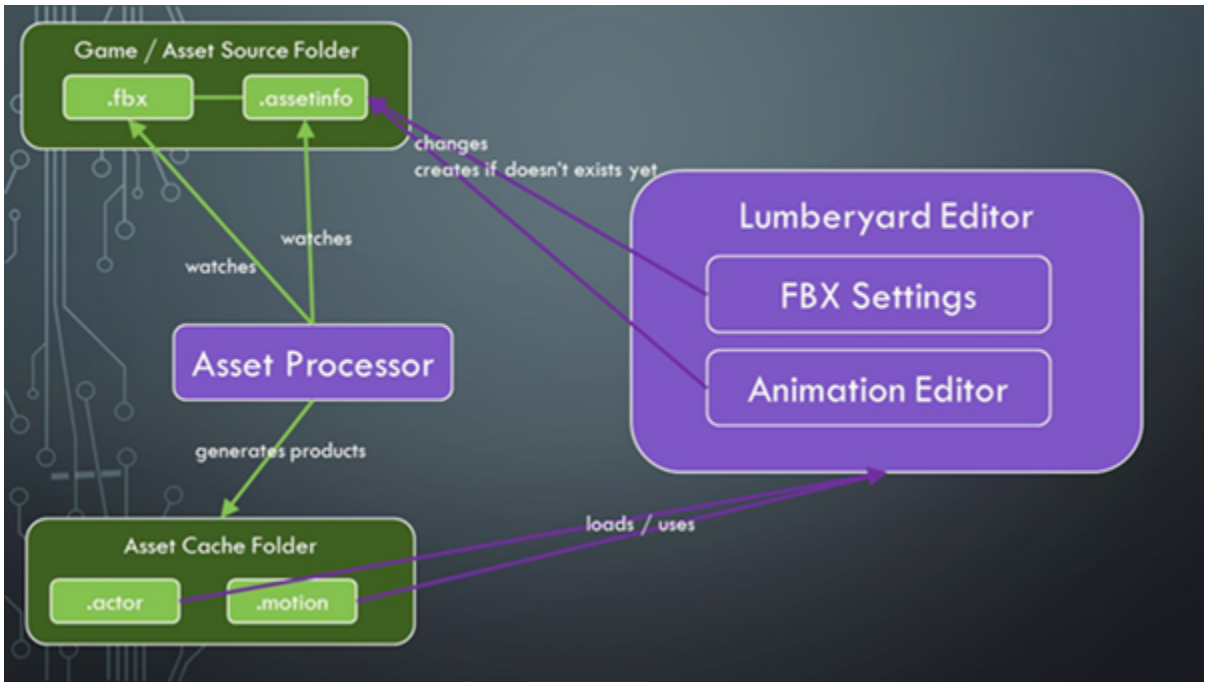
In the **Motion** pane, you can do the following:

A. Add, save, open, or delete motions.

B. **Motion Properties** – Specify how you want the motion to animate and the play speed of the motion.

Animation Editor File Types

When you import `.fbx` files from your DCC to Lumberyard Editor, Asset Processor creates files that you use in the **Animation Editor**. The following example shows how the file types are created and modified. For more information about processing files for character and animations, see Working with the FBX Settings (p. 212).



The following are file types used in the **Animation Editor**:

Files Types Created from Asset Processor

When you import `.fbx` files into Lumberyard, Asset Processor generates the following file types used for the **Animation Editor**:

- `.actor` files are created when the `.fbx` file has at least one bone. You want to use your character's skinned mesh as your `.actor` file. The `.actor` file is the character that displays the animation.
- `.motion` files are created when the `.fbx` file with a bone has at least one key frame. If your `.fbx` file has animation key frames, a `.motion` file is created. Your `.motion` files contain the animations that are added to your motion set before you build your animation graph.
- `.mtl` files are created when the `.fbx` file has at least one material, which is the case for most DCC tools. If you make changes to the material in the **Material Editor**, the `.mtl` file is no longer a child of the `.fbx` file when you view it in the **Asset Browser**. The `.mtl` file is a sibling in the source directory of the `.fbx` file. You can make other changes to the `.mtl` file with a text editor or the **Material Editor**.

When a `.motion` file is created, an `.actor` file is also created. The `.actor` file that you want to use in the **Animation Editor** is generally the skinned mesh in the bind pose that you export from your DCC. Keep track of your skinned mesh that you want to use as your `.actor` file in the **Animation Editor**. You can go to the **FBX Settings** tool and delete the actor files that you don't need.

Note
When Asset Processor creates your character and animation files from the `.fbx` file, it can also create the `.caf` and `.chr` file types. These files can only be used in Geppetto and cannot be used in the **Animation Editor**.

File Types Required for Animation Graphs

When you create your animation graph in the **Animation Editor**, the animation graph must have the following files:

- `.actor`
- `.motion`
- `.motionset`
- `.animgraph`

Saving Animation Editor Files

Saving your project in the **Animation Editor** creates a `.workspace` file. The workspace saves the actor, motion, motion set and animation graphs that you are using. When you open a workspace, the **Animation Editor** loads the files so that you can pick up where you last left off.

To save your workspace

- In the **Animation Editor**, choose **File**, and then choose one of the following:
  - **Save Workspace**
  - **Save Workspace As**

When `.actor` and `.motion` files are saved, the **Animation Editor** creates an `.assetinfo` file, alongside the source `.fbx` file. The `.assetinfo` files stores the configuration and settings for the `.actor` and `.motion` files.

Settings saved for `.actor` files include the actor name, motion extraction node, excluded bounds, collision mesh setup, and mirror setup.

Settings saved for `.motion` files include the motion extraction capture height option.

To save `.actor` and `.motion` files:

- In the **Animation Editor**, do one of the following:
  - Click **Save All** to saves any changes made to the `.actor`, `.motion`, `.motionset`, and `.animgraph` files. A dialog box prompts you to choose which files to save.
  - Click **Save Workspace** to saves your current workspace. If you don't have a workspace saved, a dialog box appears so that you can name your workspace and save it to your preferred directory.
  - Click **Save Workspace As** to saves your workspace with a different name or to another directory.
  - To save motion files individually, click the save icon in the **Motions** pane.
  - To save actor files individually, click the save icon in the **Actor Manager** pane.

Getting Started with the Animation Editor

Animation Editor is in preview release and is subject to change.

See the following procedures to get started with the **Animation Editor**.

In this procedure, you do the following:

- Import your actor file and create a motion set to specify the motions that you want for your character.
- Create a basic animation graph using nodes.
- Create a blend tree to combine the motions and use a slider to control character movement from idle to walking and then running.

You can use the Samples Project to find the files used in this procedure. For more information, see Samples Project (p. 1472).

The files for this procedure are located in the following directory:

```
lumberyard_version\dev\SamplesProject\AnimationSamples\Simple_JackLocomotion
```

> **Note**
> The **Animation Editor** can't be opened if you chose the Starter Game project in the Project Configurator. For this procedure, choose the Samples Project.

Step 1: Creating a Motion Set

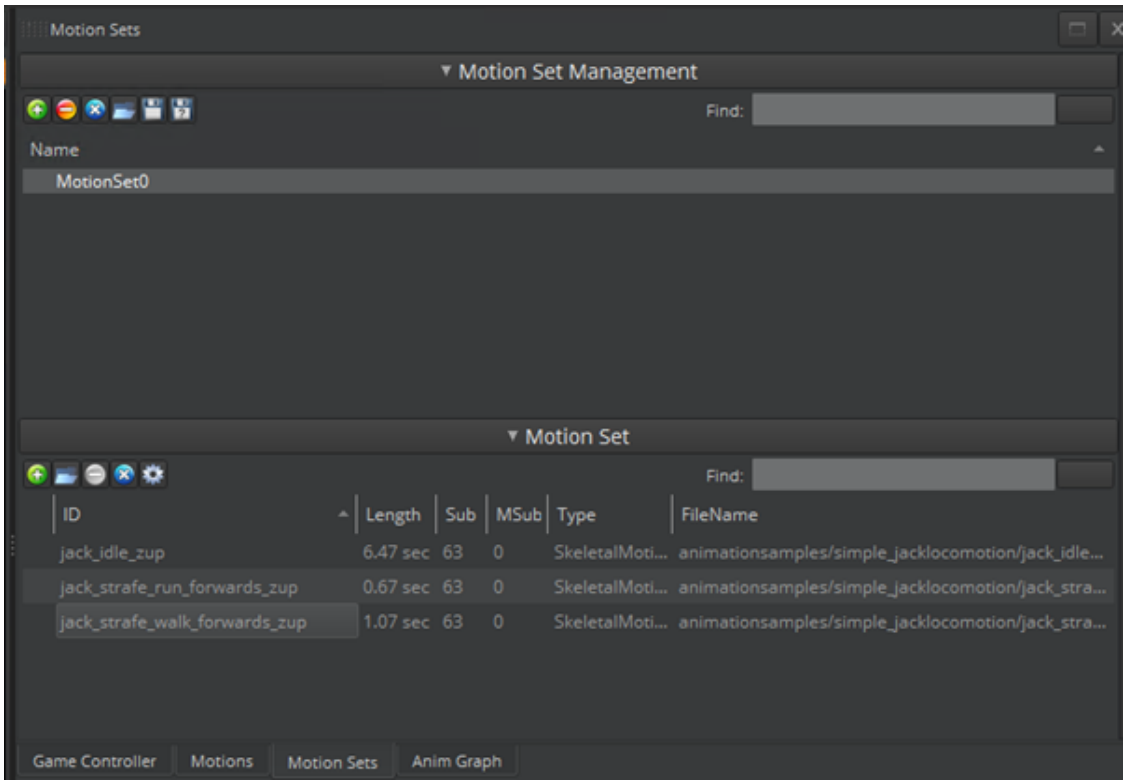
In the following procedure, you import your character, Jack the robot, select the motions that you want, and then add those motions to a motion set.

To create a motion set

1. In Lumberyard Editor, choose **Tools**, **Animation Editor**.
2. In the **Animation Editor**, choose **File**, **Open Actor** and navigate to the `AnimationSamples\Simple_JackLocomotion` directory.
3. Select the `Jack_Idle_ZUp.fbx` file and then click **OK**.

   Your character Jack appears in the **Animation Editor**.

4. Click the **Motion Sets** tab and then under the **Motion Set Management** pane, click the green **+** icon to add a motion set.
5. Under the **Motion Set** pane, click the blue folder icon to add motions.
6. Navigate to the `AnimationSamples/Simple_JackLocomotion` directory, and select the following files:
   - `Jack_Idle_ZUp.fbx`
   - `Jack_Strafe_Run_Forwards_ZUp.fbx`
   - `Jack_Strafe_Walk_Forwards_ZUp.fbx`
7. Click **OK**.
8. In the **Motion Set Management** pane, click the **Save** icon.
9. Navigate to the `/SamplesProject/AnimationSamples/Simple_JackLocomotion` directory. For the file name, type **`quickstart`** and then click the **Save** icon to save the `quickstart.motionset` file.

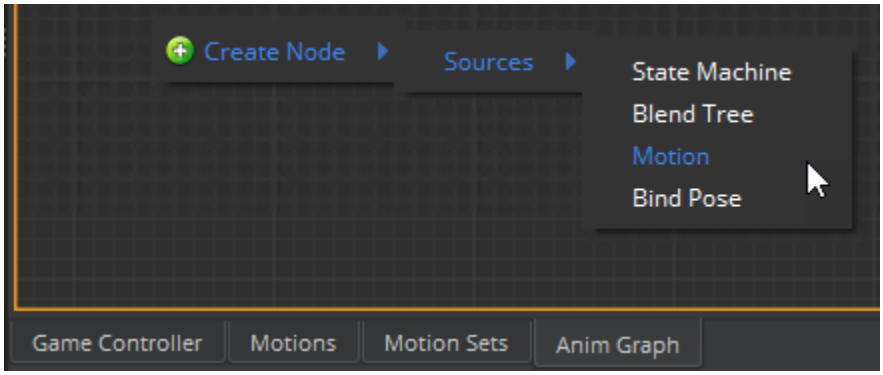


Step 2: Creating an Animation Graph

In the following procedure, create an animation graph and nodes.

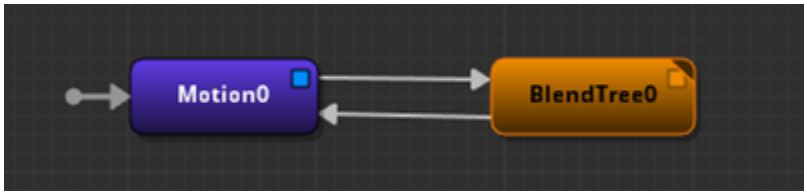
To create an animation graph

1. On the right of the **Animation Editor**, in the **Resource Management** pane, click the green **+** icon to create an animation graph.
2. Click the **Save** icon.
3. Navigate to the `/SamplesProject/AnimationSamples/Simple_JackLocomotion` directory. For the file name, type **`quickstart`** and then click **Save** to save the `quickstart.animgraph` file.
4. In the center of the **Animation Editor**, click the **Anim Graph** tab, right-click the grid, and then select **Create Node**, **Sources**, **Motion**.



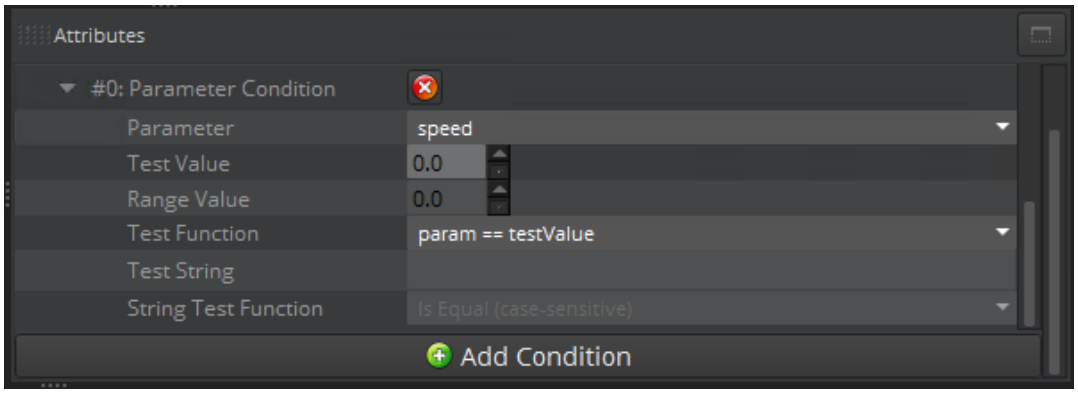
5. Select the **Motion0** node and in the **Attributes** pane, click **select motions**. In the dialog box, select `Jack_Idle_ZUp.fbx` and then choose **OK**.

6. Right-click the grid and then choose **Create Node**, **Sources**, **Blend Tree**.
7. From the **Motion0** node, click and drag a line to the **BlendTree0** node. A transition line with an arrow connects the nodes.
8. From the **BlendTree0** node, click and drag a line to the **Motion0** node.



9. In the **Parameters** pane, click the green **+** icon to create a parameter.
   a. For **Parameter Name**, rename `Parameter0` to **`speed`**.
   b. Leave the **Value Type** parameter to the default, **FloatSlider**, and then click **Create**.
10. In the **Anim Graph**, select the transition line that starts from the **Motion0** node and connects to the **BlendTree0** node.
    a. In the **Attributes** pane, click **Add Condition**.
    b. In the dialog box, select **Parameter Condition** and then click **Add Condition**.
    c. In the **Attributes** pane, go to the **Parameter Condition**, and for the **Parameter** value, select **`speed`**.
    d. For **Test Function**, leave the default value of **param > testValue**. This means that if the speed is greater than zero, the idle motion transitions to the blend tree, and the character starts to move.

11. In the **Anim Graph**, select the transition line that starts from the **BlendTree0** node and connects to the **Motion0** node.
    a. In the **Attributes** pane, click **Add Condition**.
    b. In the dialog box, select **Parameter Condition** and then click **Add Condition**.
    c. In the **Attributes** pane, go to the **Parameter Condition**, and for the **Parameter** value, select **`speed`**.
    d. For **Test Function**, specify **param == testValue**. This means that if the speed is equal to zero, the motion transitions back to idle, and the character stops moving.



Step 3: Blending the Animations

In the following procedure, you use the blend tree node to build your blend tree, which blends the walk and run animations together.

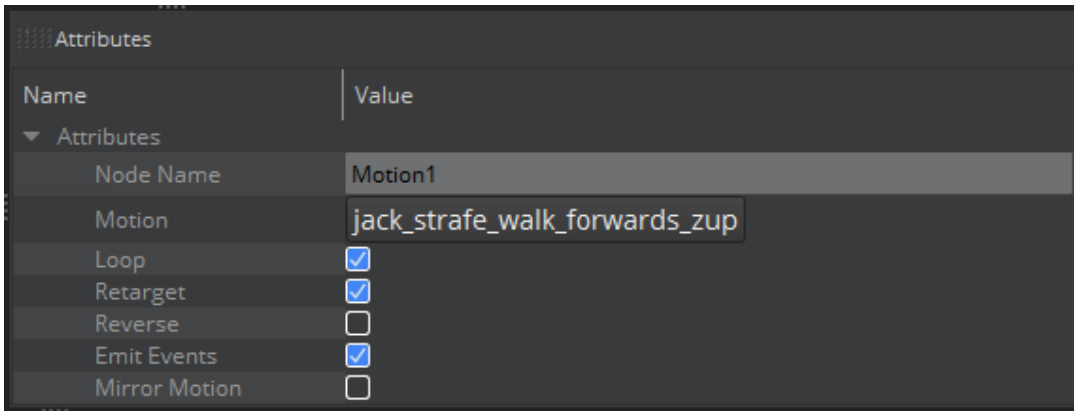
To blend the animations

1. In the **Anim Graph**, double-click the **BlendTree0** node.
2. Right-click the grid and choose **Create Node**, **Sources**, **Motion**.
3. Select the **Motion1** node.

   a. In the **Attributes** pane, choose **select motions**.

   b. In the dialog box, select `Jack_Strafe_Walk_Forwards_ZUp.fbx` and then click **OK**.

   The attributes for the **Motion1** node should look like this:

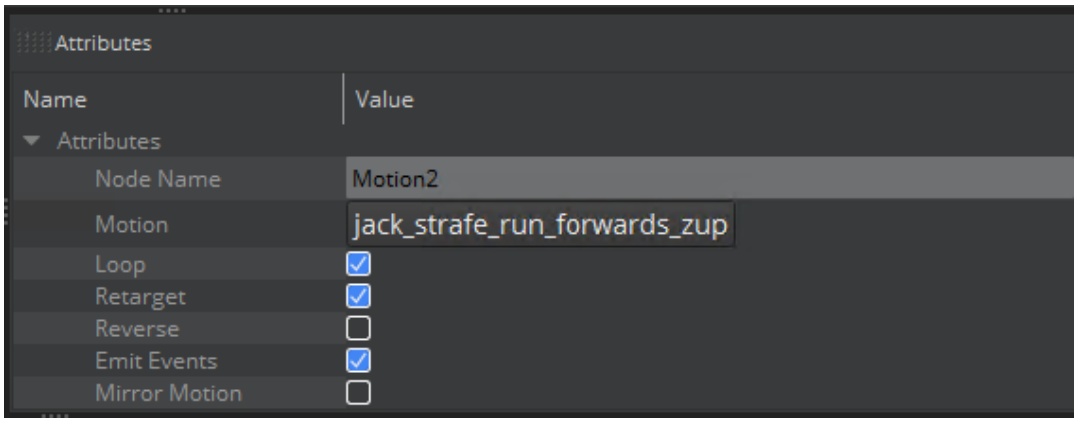


4. In the **Anim Graph**, right-click the grid and choose **Create Node**, **Sources**, **Motion**.
5. Select the **Motion2** node.

   a. In the **Attributes** pane, click **select motions**.

   b. In the dialog box, select `Jack_Strafe_Run_Forwards_ZUp.fbx` and then click **OK**.

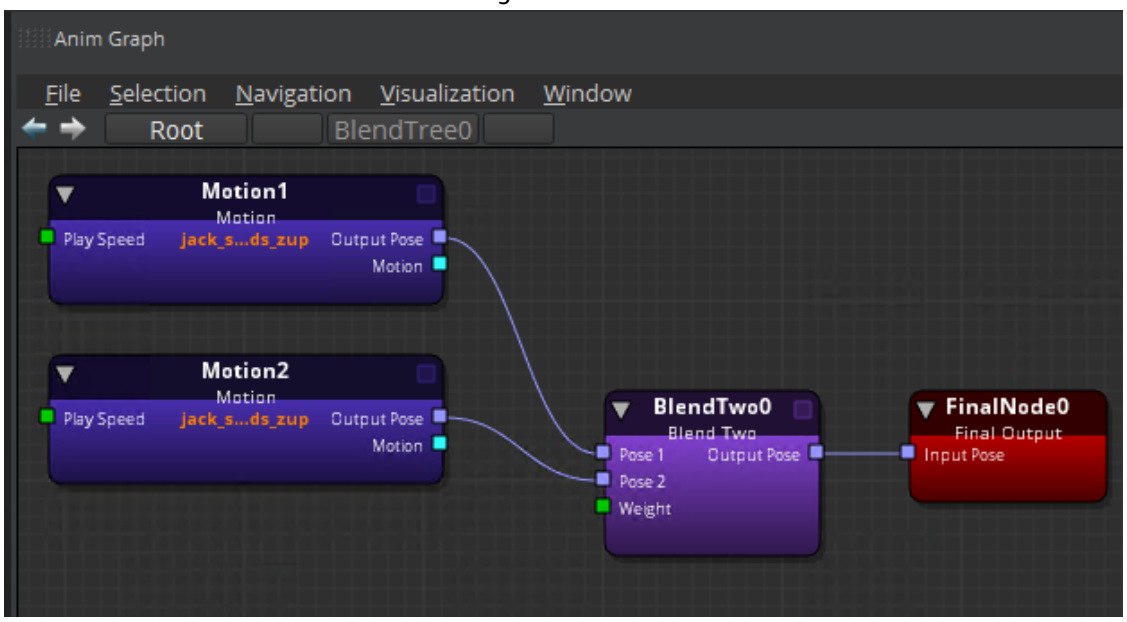
   The attributes for the **Motion2** node should look like this:



6. Right-click the grid and choose **Create Node**, **Blending**, **Blend Two**.
7. Select the **BlendTwo0** node.

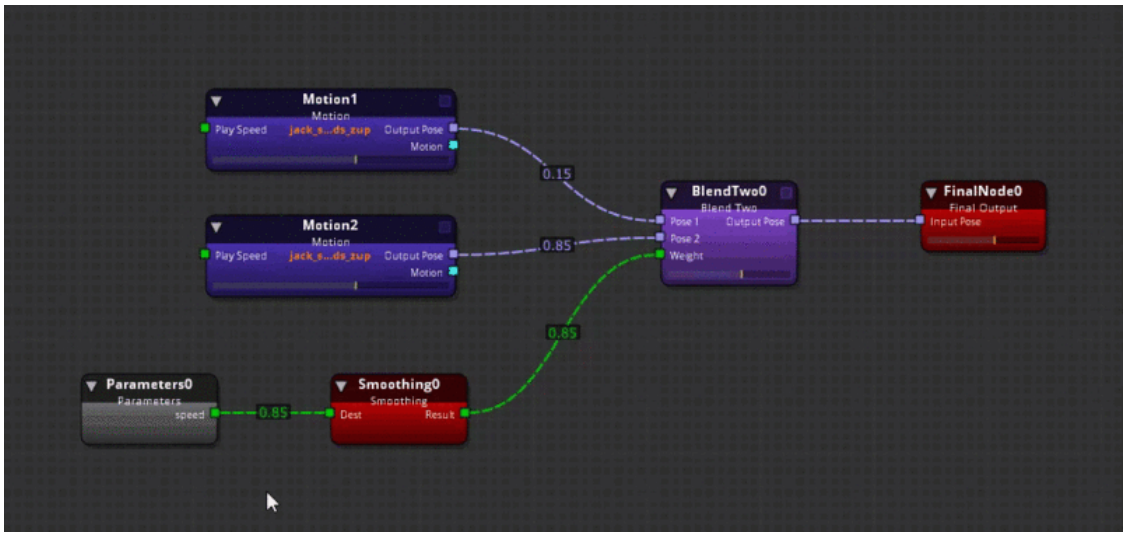
   - In the **Attributes** pane, for **Sync Mode**, select **Full Clip Based**.
8. For **Motion1** node, select the **Output Pose** box and drag the connector to the **Pose 1** input of the **BlendTwo0** node.
9. For **Motion2** node, select the **Output Pose** box and drag the connector to the **Pose 2** input of the **BlendTwo0** node.
10. For the **BlendTwo0** node, select the **Output Pose** box and drag the connector to the **Input Pose** of the **FinalNode0** node.

    Your blend tree should look like the following:

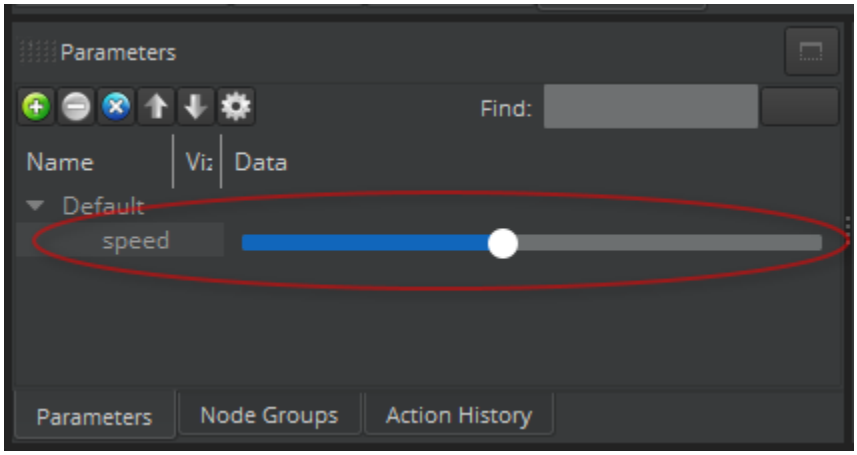


11. Right-click the grid and choose **Create Node**, **Sources**, **Parameters**.
12. Right-click the grid and choose **Create Node**, **Math**, **Smoothing**.
13. For **Parameters0** node, select the **speed** output box and drag the connector to the **Dest** input box of the **Smoothing0** node.
14. For the **Smoothing0** node, select the **Result** output box and drag the connector to the **Weight** input box of the **BlendTwo0** node.

    Your blend tree should look like the following:



15. In the **Animation Editor**, choose **File**, **Save All**. Then in the dialog box, click **OK**.
16. Navigate to the `/SamplesProject/AnimationSamples/Simple_JackLocomotion` directory. For the file name, type **`quickstart`** and then click **Save** to save the workspace.
17. In the **Resource Management** pane, double-click the **quickstart.animgraph** setting. Ensure that for the **Motion Set**, the **MotionSet0** node is specified. The character should now be animated in the idle mode.
18. Go to the **Parameters** pane. Move the **speed** slide control to the right to make Jack walk. Move the slider more to the right to make Jack run.



Animation Editor Components

Animation Editor is in preview release and is subject to change.

You can add the following **Animation Editor** components to an entity. You can then specify the actor file, animation graph file, the motion file, or attachment.

For more information, see the following components:

- **Actor (p. 448)**
- **AnimGraph (p. 449)**
- **Attachment (p. 454)**
- **Simple Motion (p. 555)**

For more information about adding a component to an entity, see Adding Components to an Entity (p. 593).

Creating and Visualizing Blend Spaces

Animation Editor is in preview release and is subject to change.

Blend spaces are a collection of sample motions that are organized spatially according to their coordinates. The visual representation is a graph with xy-axes for the coordinates. The xy-axes can represent values such as move speed, travel direction, turn angle, and so on. Each motion is represented by a point (white dot) in the graph or blend space.

When you pick a point in the blend space (interactively or with parameter controls), your character automatically plays a resulting motion that is computed based on the sample motions and appropriate blend weights.

In a 1D blend space the motions correspond to points along a line. In a 2D blend space the motions correspond to points in a 2D space.

Prerequisites

Before you can add blend space nodes to the animation graph, you must have completed the following:

- Selected an actor
- Selected a motion set
- Created an animation graph

For more information, see Getting Started with the Animation Editor (p. 234).

Creating Blend Spaces

To create a blend space, you must add a blend tree node and a blend space node, and then specify values for the attributes of the blend space node.

Blend space nodes output a pose from the **Output Pose** port. You can connect this output to the **Input Pose** port of the **Final Output** node or to the input port of any other node that accepts a pose as input.

The **Blend Space 1D** node has the following ports:

- **X** – The value for this input port indicates the current position of interest in the 1D blend space.
- **Output Pose** – The blend space node computes the blended motion that corresponds to the current position of interest and outputs the resulting motion from this port.

The **Blend Space 2D** node has the following ports:

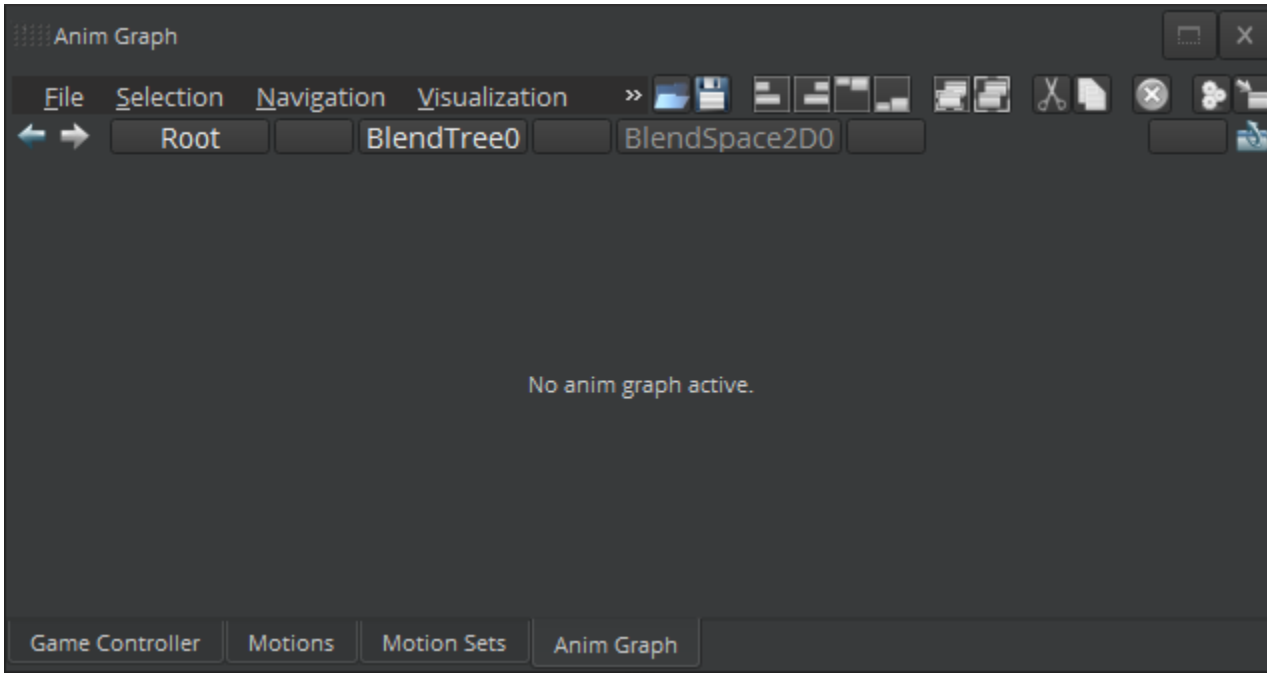
- **X** – The value for this input port is the x-coordinate of the current position of interest in the 2D blend space.
- **Y** – The value for this input port is the y-coordinate of the current position of interest in the 2D blend space.
- **Output Pose** – The blend space node computes the blended motion that corresponds to the current position of interest and outputs the resulting motion from this port.

To create a blend space and specify attributes

1. In the Animation Editor, on the **Anim Graph** tab in the top middle pane, right-click the grid and then choose **Create Node**, **Sources**, **Blend Tree**.
2. Double-click the **Blend Tree** node to go to the blend tree view.
3. Add a blend space node to the blend tree by doing one of the following:
   - On the **Anim Graph** tab, in the blend tree view, right-click the grid and then choose **Create Node**, **Blending**, **Blend Space 2D** or **Blend Space 1D**.

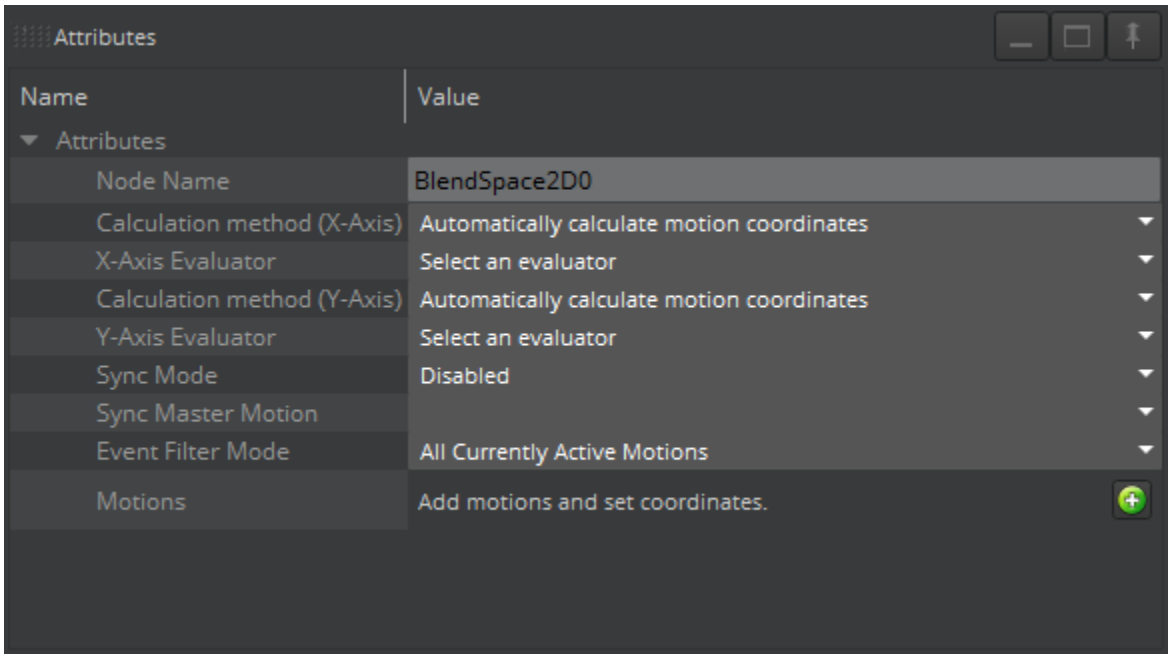
- In the **Anim Graph Palette**, on the **Blending** tab, drag the **Blend Space 2D** or **Blend Space 1D** icon into the blend tree view.

4. Double-click the blend space node to go to the blend space view. If you are using the **Blend Space 2D** node, your view should look as follows:



5. In the **Attributes** pane, specify values for the attributes of the blend space node. These values are used to set up your blend space.

   Note
   You can undock the **Attributes** pane in order to see the attributes and values without scrolling.

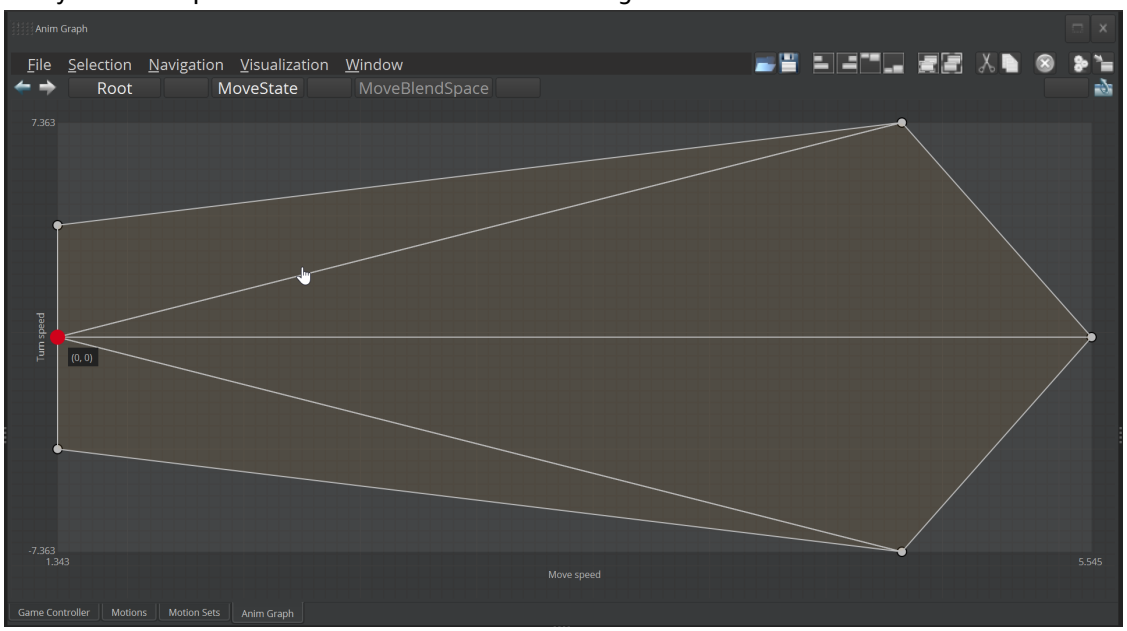


- To use provided values for the xy-axes, do the following:
  1. For **Calculation method (X-Axis)**, select **Automatically calculate motion coordinates**.
  2. For **X-Axis Evaluator**, select a common motion characteristic.
  3. For **Calculation method (Y-Axis)**, select **Automatically calculate motion coordinates**.
  4. For **Y-Axis Evaluator**, select another common motion characteristic.
- To use custom values for the xy-axes, do the following:
  1. For **Calculation method (X-Axis)**, select **Manually enter motion coordinates**.
  2. For **Calculation method (Y-Axis)**, select **Manually enter motion coordinates**.

  You can also use a combination of provided and custom values. For example, you can manually enter motion coordinates for the x-axis and automatically calculate motion coordinates for the y-axis using the **Travel distance** evaluator.

6. In the **Attributes** pane, for **Motions**, click the **+** button to add the source motion assets for your blend space.
7. In the **Motion Selection Window**, choose the motions that you want to add to the blend space, and then click **OK**.

   - The coordinate value automatically calculates if you selected **Automatically calculate motion coordinates** for **Calculation method** and if you selected a motion characteristic for the **Evaluator**.
   - You must enter coordinate values if you selected **Manually enter motion coordinates** for **Calculation method**.
8. After your motions are added to the blend space and the coordinate values are calculated, verify that your blend space view looks similar to the following:



9. In the blend space view, do the following:

   a. Drag within the blend space to change the point of interest (represented by a red dot).

   b. When the point is highlighted, the corresponding motion is automatically computed by blending the motions represented by the three vertices of the triangle. View the blend weights next to each of the motions.

   c. Notice that the motions that are closer to the point have a higher blend weight than the motions that are farther away.

10. In the **Attributes** pane, you can do the following:
    - View the coordinate values for each motion
    - Change the values to remap the animations and the blend space graph
    - Remove a motion from the blend space

Customizing State Machine Routing with Sparse Motion Sets

Animation Editor is in preview release and is subject to change.

You might run into a case where you have different characters that don't have the exact same motions as the character for which you created the animation graph. Instead of duplicating large parts or creating new animation graphs for different characters, you can share the same animation graph for all your characters.

You can allow or deny transitions and routes to motion states in a state machine, based on the existence of the given motion entry.

Topics

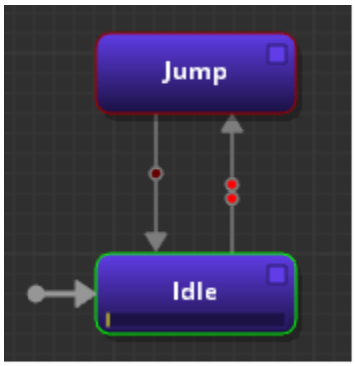
Sparse Motion Sets

A motion set contains a list of motion entries. Each motion entry has a motion ID that you can define. For example, you can name a motion ID, `Walk` or `Jump`. The motion entry maps the user-defined motion ID like `Walk` to a specific motion file, such as `/SampleProject/Animations/Human_Walk.motion`.

You can add motion entries to a motion set with the **+** icon or the folder icon in the motion set window. The **+** icon creates a new motion entry where you can specify the motion ID, but does not assign it to a motion file. You can click the folder icon to select a motion to add. Based on the motion file name, a default motion ID is generated for you.

A *sparse motion set* is when the motion entries do not have a motion file assigned to them.

When you choose an unassigned motion entry on a motion state in a state machine, the motion shows a red border around the node. If the motion state is activated, the character goes to a bind pose. In the following example, the character's animation graph does not transition from idle to jump because the **Jump** node is using a sparse motion set; no motion file is assigned. The character remains in the **Idle** node.



Hierarchical Motion Sets and Overwriting Motion Entries

A motion set can contain child motion sets that are stored in the same file. Child motion sets inherit all motion entries from the parent.

In the following example, a hierarchical motion set has the **Human** motion at the root level. The **Zombie** motion set is a child, which inherits from the parent **Human** motion set. The **Zombie** motion set does not have a motion entry for `Jump`.

Example : Hierarchical Motion Set

Human (motion set)

- + Idle – `GenericIdle.motion`
- + Walk – `GenericWalk.motion`
- + Jump – `GenericJump.motion`

Zombie (motion set)

- + Walk – `ZombieWalk.motion`

Because the parent motion set defines the `Jump` motion entry with a motion file, the child motion set inherits that motion. This means that when the **Zombie** motion set is activated, characters with that motion set play the human `Jump` motion.

If you don't want your zombie characters to jump, you can disable inheritance for that specified motion. In the child motion set, you can create a new motion entry named `Jump` and then mark it as unassigned. This way you can override the motion entry from the parent, by not assigning it for the child motion set.

Example : Child Motion Set with Unassigned Jump Motion

Human (motion set)

- + Idle – `GenericIdle.motion`
- + Walk – `GenericWalk.motion`
- + Jump – `GenericJump.motion`

Zombie (motion set)

- + Walk – `ZombieWalk.motion`
- + Jump – *Unassigned*

The **Zombie** motion set uses the `Idle` motion from the parent motion set, customizes the `Walk` motion, and disables the `Jump` motion.

For hierarchical motion sets, you can create a motion entry and unassign it to disable inheritance from the parent motion set. If you are not using hierarchical motion sets, this is the same as not having a motion entry with the specified motion ID.

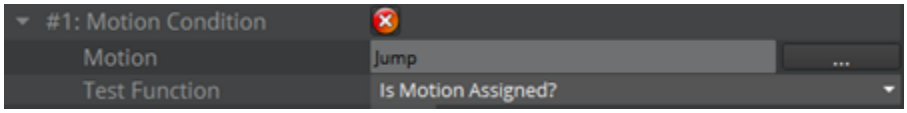
Customizing State Machines Based on Motion Sets

Animation graphs can be shared across characters. Two different characters that use the same animation graph can operate with two different motion sets. For example, you can have a human character that uses the human motion set, and a zombie character that uses the **Zombie** motion set. Both characters can use the same animation graph.

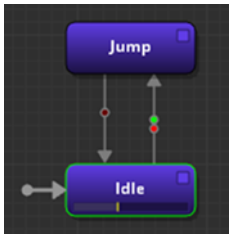
You can configure the state machine to avoid motion states that are unassigned. In this example, you don't want the zombie to go into the **Jump** state, as this motion was unassigned for the zombie motion set.

To configure the state machine to avoid motion sets

1. In the **Anim Graph**, choose the transition line between your motion nodes. For example, you can select the transition line between the **Idle** and **Jump** nodes.
2. On the **Attributes** pane, click **Add Condition** and then select **Motion Condition**.
3. For **Motion**, select the **Jump** state.
4. For **Test Function**, select **Is Motion Assigned?**.



Because the **Zombie** motion set does not have a motion file assigned for `Jump`, the character can't transition from the idle to jump state. The condition's traffic light appears red and blocks the transition. This lets you control whether a character is allowed to go to specific motion state or not.



Legacy Animation System

This section contains legacy tools and workflows.

The legacy 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

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 skeleton.

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.

Maya Export Tools

Lumberyard Tools is a plugin for Autodesk Maya 2014, 2015, 2016, and 2017 that exports geometry, animated geometry, skinned geometry, and skeletons (joint hierarchies) from Maya into Lumberyard.

To install the Lumberyard Tools plugin

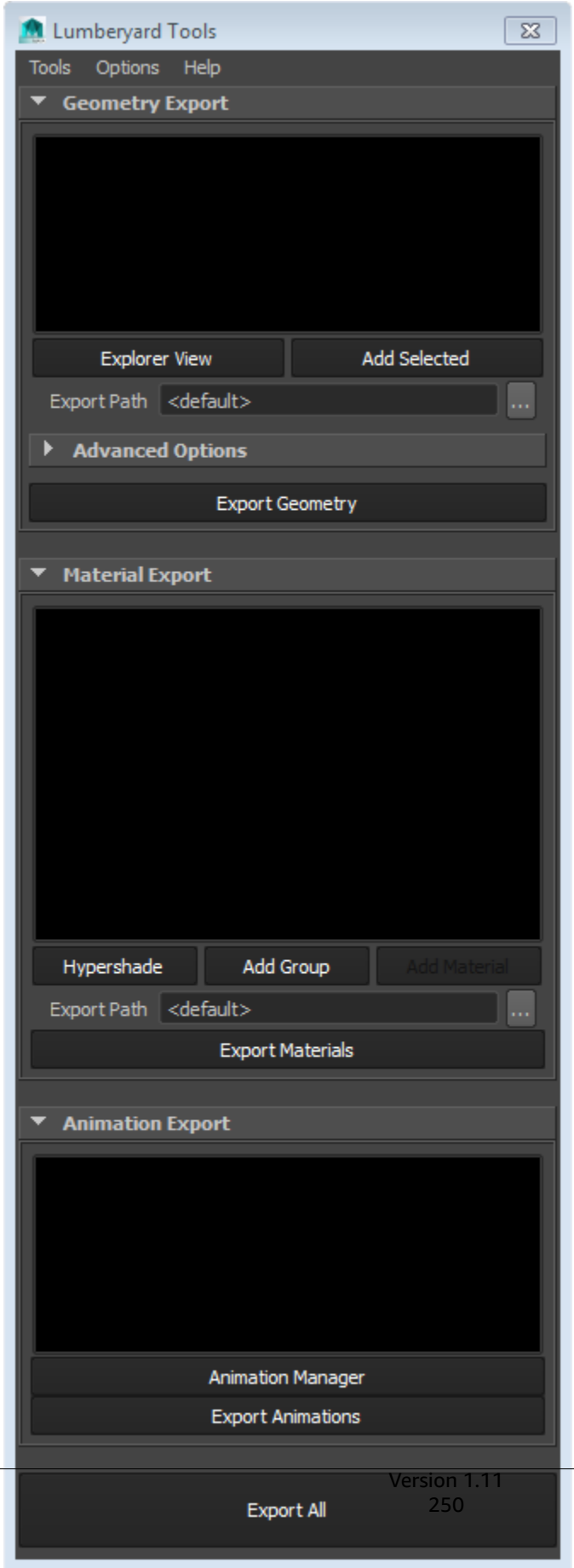
1. Navigate to the Lumberyard root directory (`\lumberyard\dev`) and run `Lumberyard Setup Assistant`.
2. On the **Install plugins** page, install **Autodesk Maya**.

Topics

Accessing Maya Export Tools

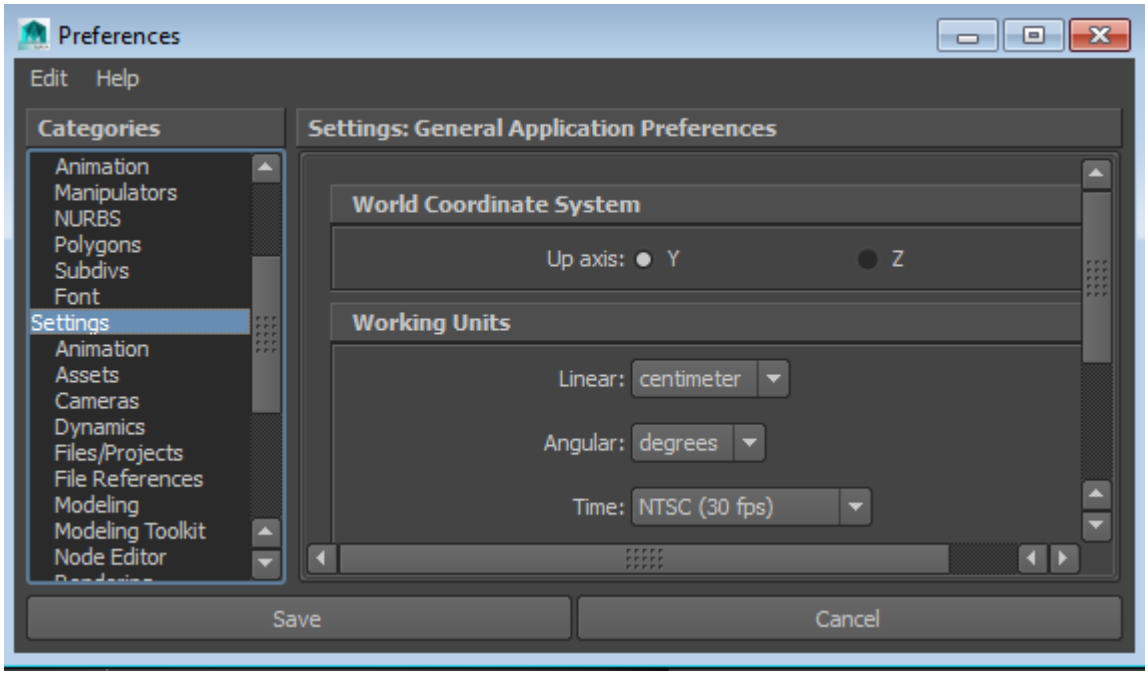
To install this plugin, run `SetupAssistant.bat`. On the **Install Plugins** page, install **Autodesk Maya**. After it is installed, the **Lumberyard** tab is available in the user interface of Maya. This tab presents a series of options, including the Lumberyard Tools beaver icon.

After the Lumberyard Tools dialog box opens, the following is shown:


Lumberyard Tools
Tools
Options
Help
Geometry Export
Explorer View
Add Selected
Export Path
<default>
Advanced Options
Export Geometry
Material Export
Hypershade
Add Group
Add Material
Export Path
<default>
Export Materials
Animation Export
Animation Manager
Export Animations
Export All

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

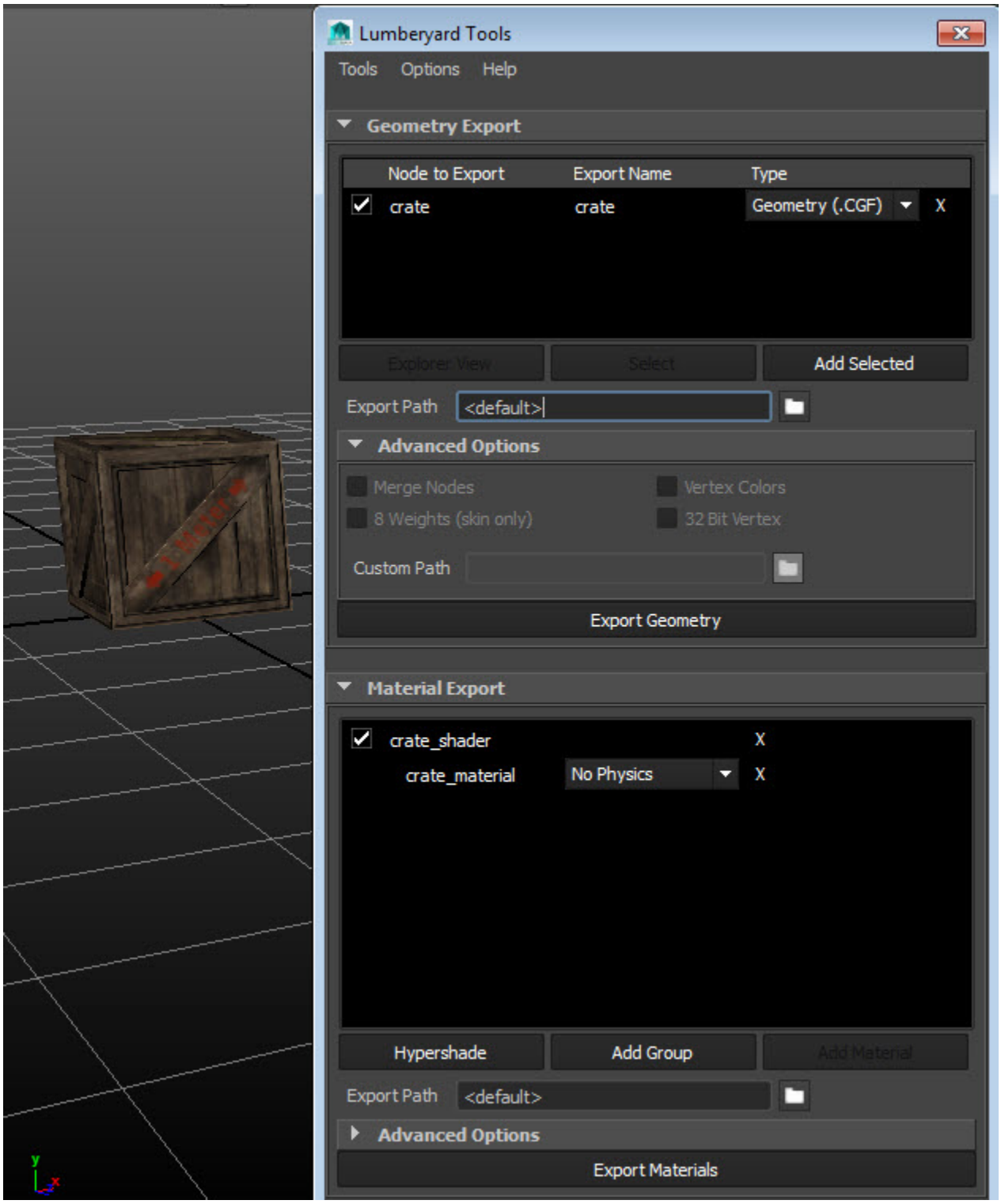
1. In Maya, choose **Window**, **Settings/Preferences**, **Preferences**.
2. In the **Preferences** dialog box, choose **Settings**.
3. Under **Working Units**, we recommend that you choose **NTSC (30fps)**, but **NTSC Field (60fps)** and **Film (240fps)** are also valid options.
4. Choose **Save**.

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.

     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 Multiple Meshes with a Collision Object

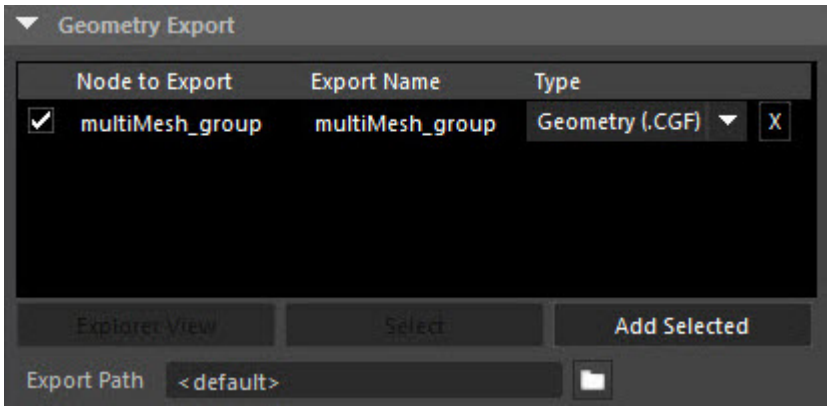
You can use the Lumberyard Tools for Maya and the **FBX Settings** tool to generate your meshes. In Maya, create your material and set the collision property so that Lumberyard can identify the mesh as a collision.

For more information about the `.fbx` format, see FBX Export in the Autodesk Maya LT documentation.

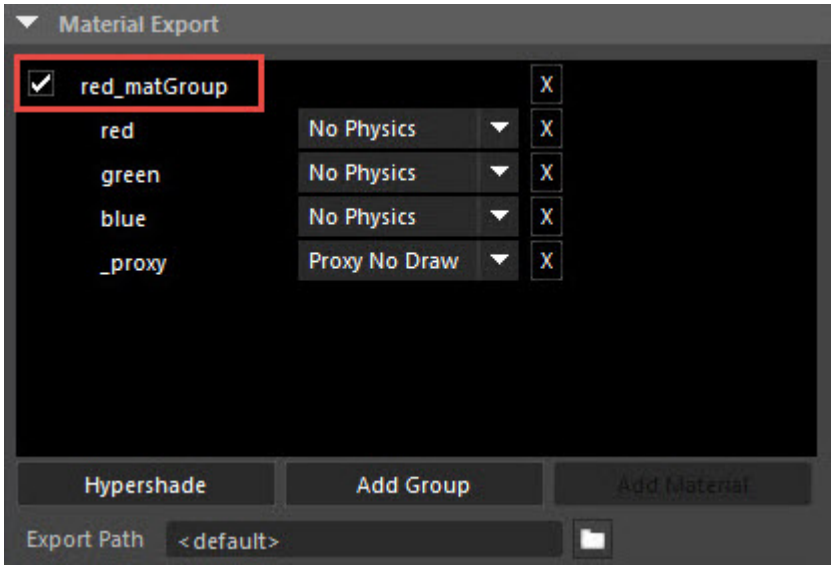
To export a mesh group with a collision object

1. In Autodesk Maya, create your objects and define your collision object.
2. On the Maya tool shelf, click **Lumberyard Tools**.
3. Click the beaver icon to open the **Lumberyard Tools** window.
4. In the **Asset Browser** pane, select your objects and press **Ctrl+G** to group the objects.
5. In the outliner or channel box, name the new group. We recommend appending the group name with `_group`.

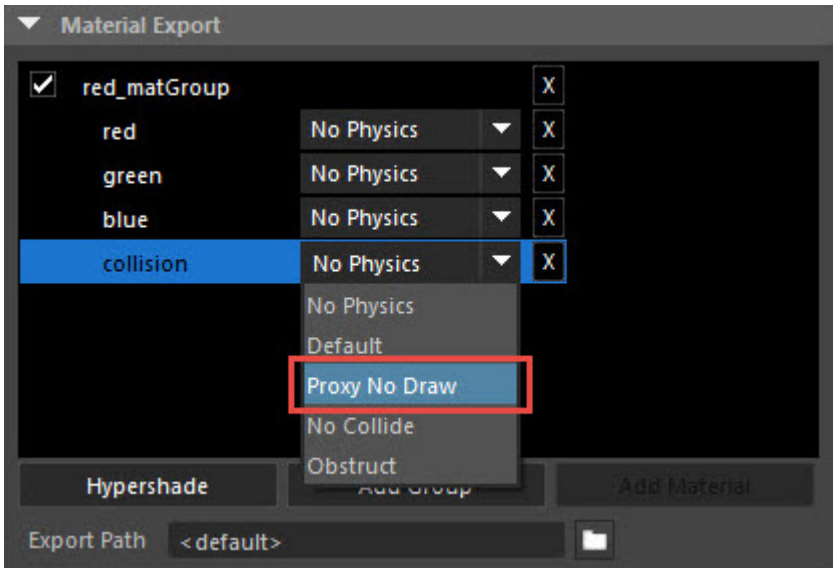
6. In the outliner or channel box, select the group name.
7. In the **Lumberyard Tools** window, under **Geometry Export**, click **Add Selected**. The group name appears under **Geometry Export**.



8. Before you can export an object, you must assign a material to the object. To do so, right-click the object in the viewport and choose **Assign New Material**.
9. In the **Assign New Material** dialog box, select a Phong shader.
10. In the viewport, select the group node with the newly assigned material.
11. In the **Lumberyard Tools** window, under **Material Export**, click **Add Group**. The default group name (from the first material) appears under **Material Export**.



12. (Optional) Under **Material Export**, rename the material for the collision object to be called collision.
13. Under **Material Export**, choose **Proxy No Draw** from the list for the **collision** shader type. You can also update this shader type later in the Lumberyard **Material Editor**.



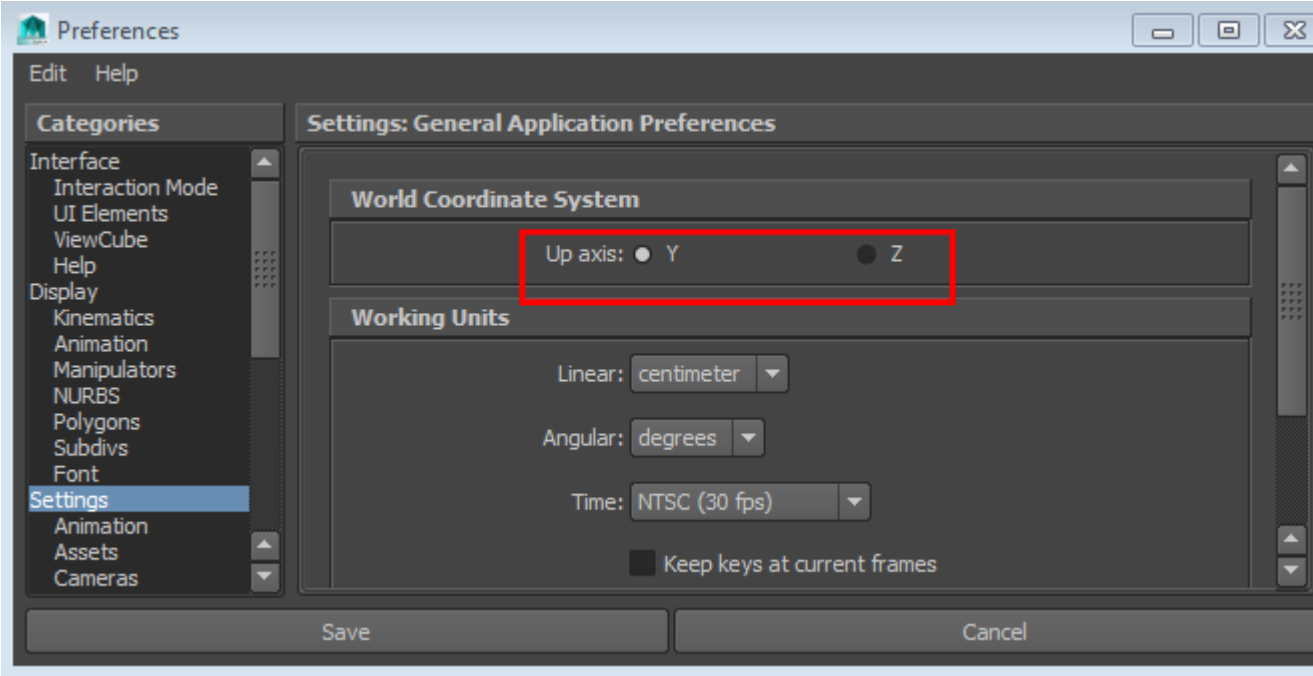
14. Save the file to your project directory.
15. Click **Export All**. Be sure to save to the directory where your Maya file types are saved.

To validate the collision box in Lumberyard Editor

1. Open Lumberyard Editor.
2. In the **Asset Browser** pane, locate the object that you created.
3. Drag the object into the viewport.
4. To frame the object, press **Z**.
5. Open the **Console Variables** window and search for `p_draw_helpers`.
6. To show the collision path, set `p_draw_helpers` to **1**. You should now see your collision box in the viewport.

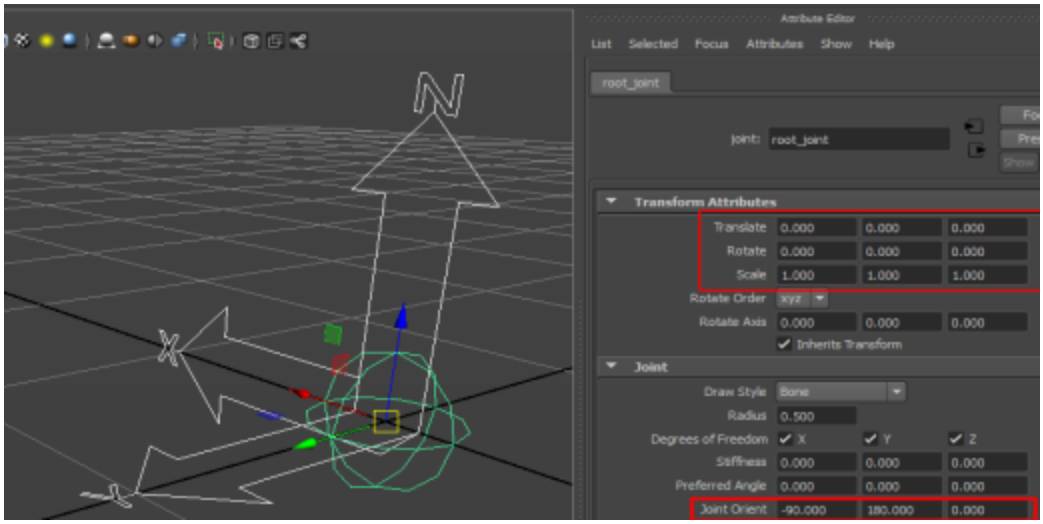
Exporting Characters

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.

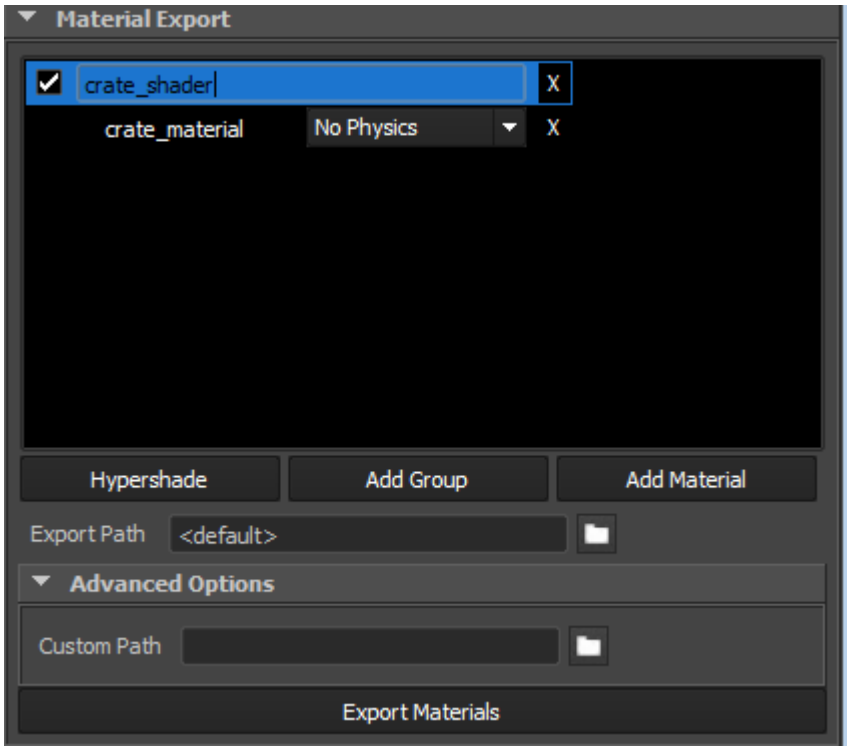
To export character geometry

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 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 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 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 a Level of Detail (LOD)

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 using up to six LODs per group node in Maya. LOD number is from 0 (highest level of detail) to 5 (lowest level of detail).

You can export the following LODs from Maya to Lumberyard:

- Static Mesh LODs

- Material LODs
- Skinned Mesh LODs

Static Mesh LODs

Use the following naming and setup to export Static Mesh LODs using the Maya Lumberyard tool.

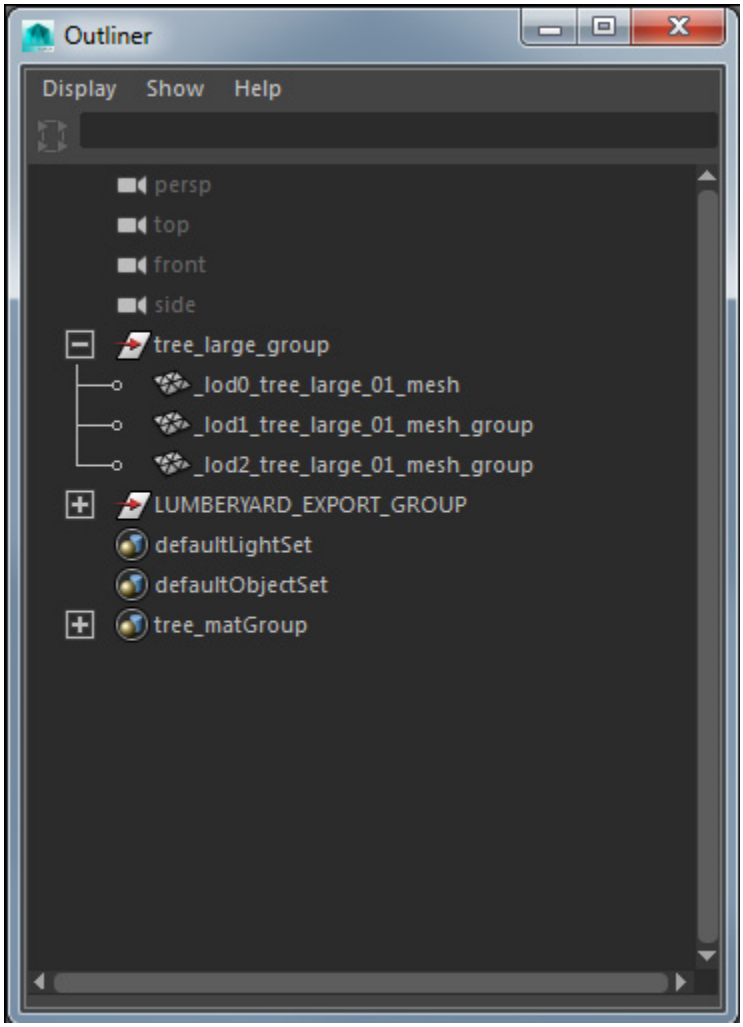
LOD Naming

Use the following naming conventions for LODs.

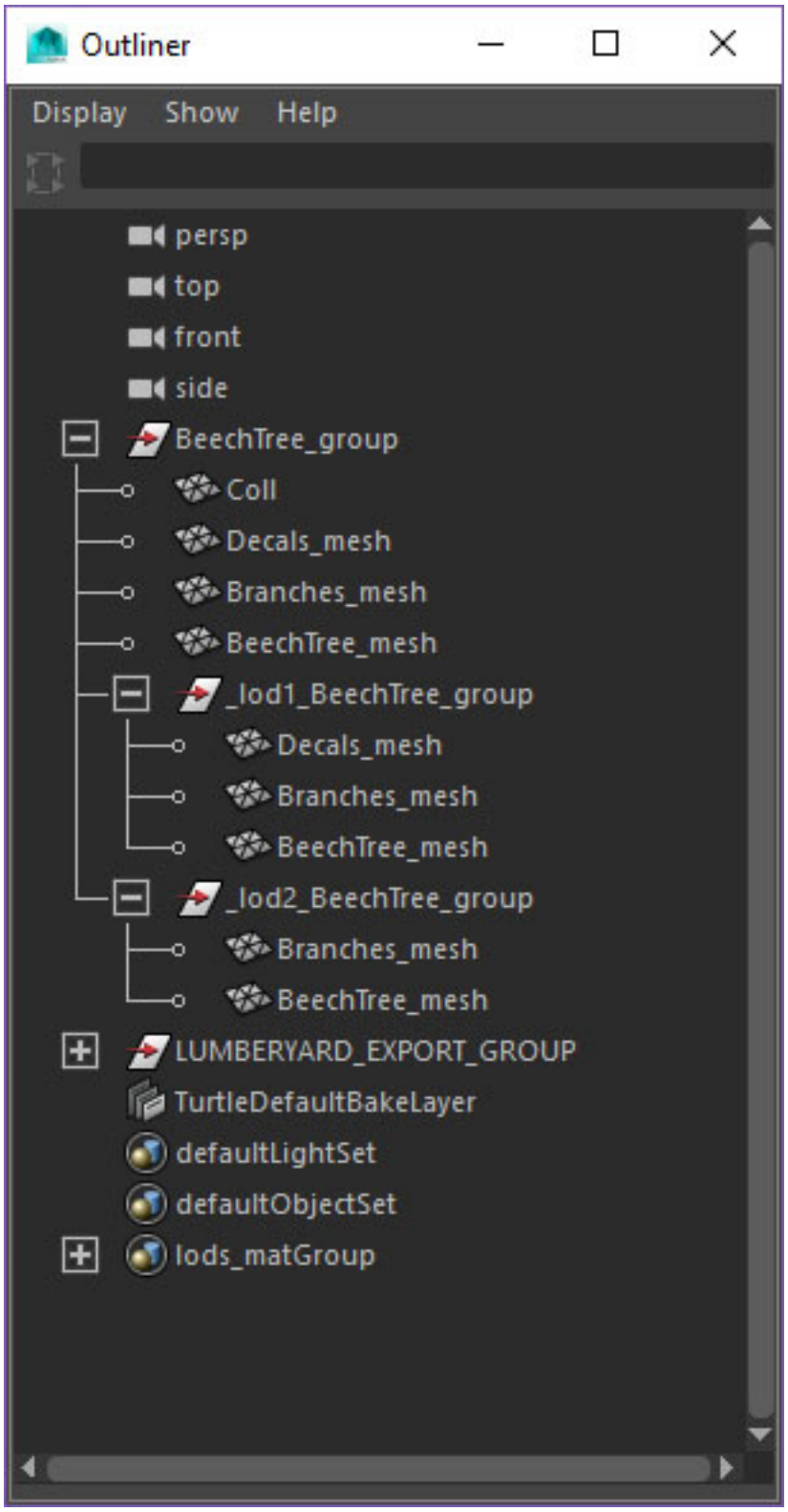
- `_lod0_` through `_lod5_` (prefix)
- `_group` (suffix)
- `_helper` (suffix)

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** check box is not selected.

Material LODs

When you set up material LODs, you create submaterials and assign them to the appropriate LOD mesh.

For example, you might have a material group named `Tree_Mats`.

Within `Tree_Mats`, you set up five submaterials. The first three submaterials you assign to LOD0, which is the highest poly model. The other two submaterials you assign to the LOD1 mesh. This means that when LOD0 is active, it uses the first three submaterials you assigned to it. When LOD1 is active, it uses only the two submaterials you assigned to it.

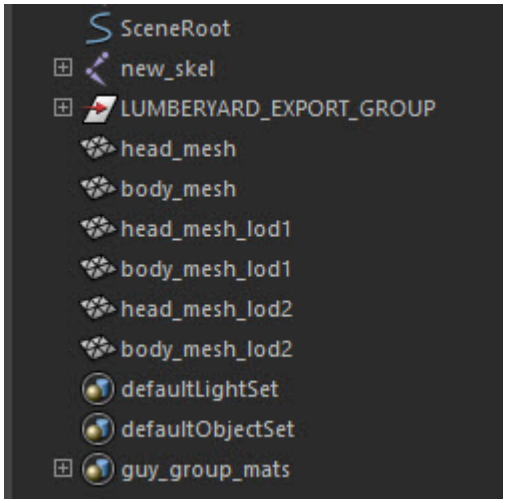
Skinned Mesh LODs

Use the following naming conventions for Skinned Mesh LODs.

- LOD0: Any name (no suffix or prefix required)
- LOD1 through LOD5: Same name as LOD0 and suffixed with `_lod1` through `_lod5`
- Optional: `_group` node suffix to organize meshes in Maya (not required for exporting to Lumberyard)

When you export your skinned mesh LODs, each skin exports as an individual `.skin` file. The following images show how a set of exported skins look in each application.

Maya Outliner:



Lumberyard Exporter in Maya:

Windows Explorer:

Name	Date modified	Type	Size
body_mesh.skin	6/14/2017 1:41 PM	ASP.NET Skin File	624 KB
body_mesh_lod1.skin	6/14/2017 1:41 PM	ASP.NET Skin File	451 KB
body_mesh_lod2.skin	6/14/2017 1:41 PM	ASP.NET Skin File	351 KB
dummyPlane_mat_group.mtl	6/13/2017 4:54 PM	MTL File	1 KB
guy_group_mats.mtl	6/13/2017 4:44 PM	MTL File	1 KB
head_mesh.skin	6/14/2017 1:41 PM	ASP.NET Skin File	72 KB
head_mesh_lod1.skin	6/14/2017 1:41 PM	ASP.NET Skin File	66 KB
head_mesh_lod2.skin	6/14/2017 1:41 PM	ASP.NET Skin File	42 KB
new_skel.chr	6/13/2017 4:54 PM	3dsMax character ...	29 KB
skin_test.cdf	6/14/2017 1:58 PM	CDF File	1 KB

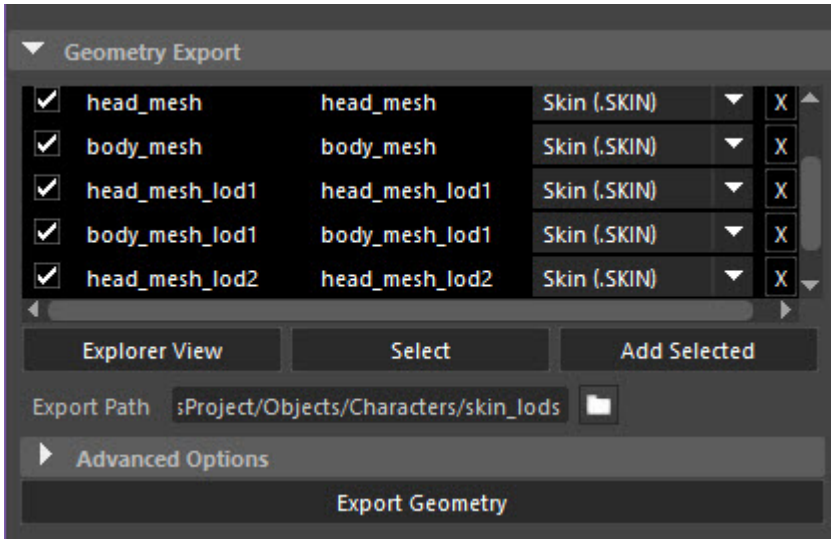
You do not need to set anything else for the LODs within the `.CDF`. This is because when you assign the LOD0 (highest poly) skin mesh, the engine automatically handles swapping LOD meshes that follow the same naming convention for skins in the same folder.

To export skinned mesh LODs

1. Ensure that your LOD0 through LOD6 meshes are named and suffixed appropriately. See the naming conventions described in the introduction.
2. Open the **Maya Lumberyard Tools**. Make sure the **Export** window is visible.
3. Select your LOD0 mesh. Click **Add Selected** to add it to the **Geometry Export** list.

   If it has skin weight data, the export type is automatically set to `.SKIN`.
4. Select your LOD1 mesh. Click **Add Selected** to add it to the **Geometry Export** list.

   If it has skin weight data, the export type is automatically set to `.SKIN`.

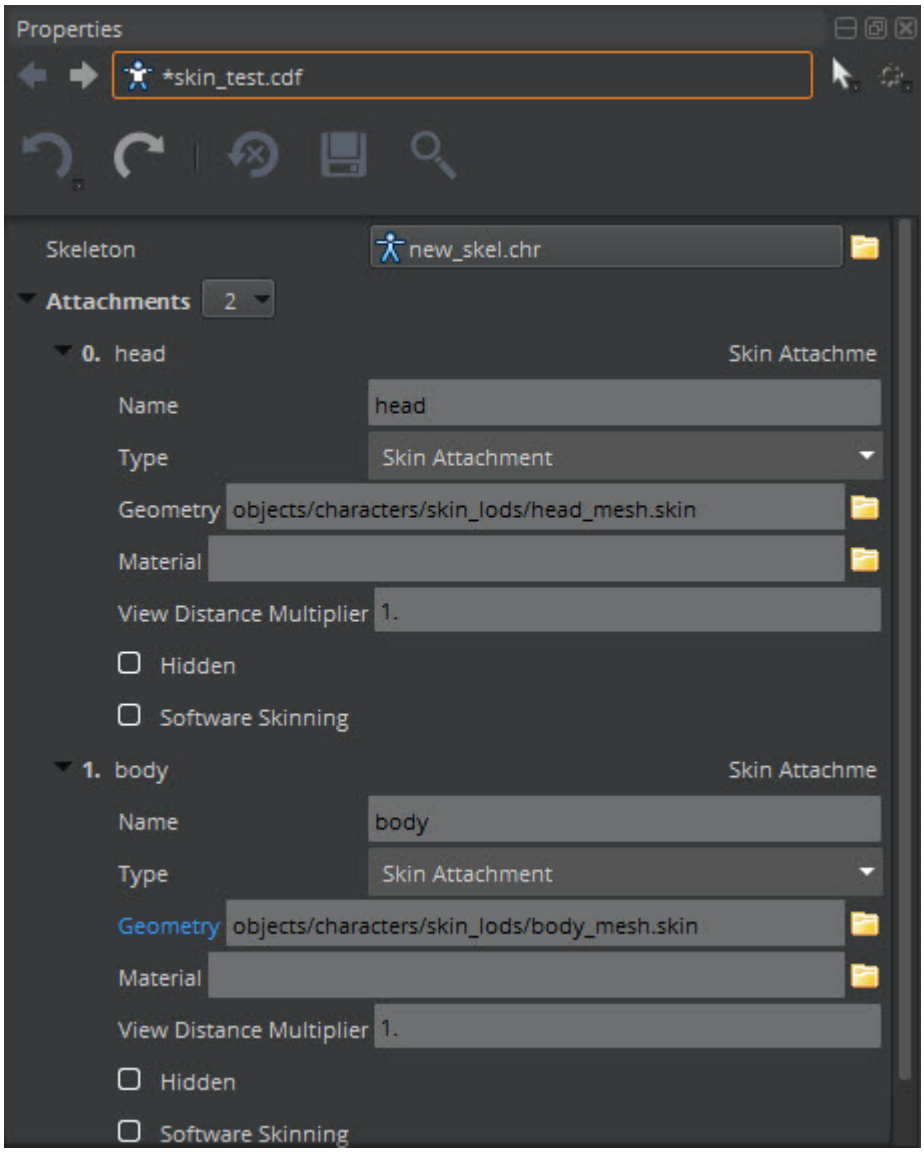


5. Repeat for the remaining LOD meshes.
6. After you add all your LOD skinned meshes, assign the appropriate export path. Then export your skeleton (`.CHR`) as you would normally.

Note
All the LOD skinned meshes must be in the same directory for the LODs to work properly in Lumberyard.

7. Open Lumberyard. Set up your `.CDF`, `.CHR` and `.SKIN` attachment(s) as normal.

   If the proper naming convention was followed, Lumberyard automatically handles the transition of the LOD meshes in-level.



To test the `.CDF` with LODs, drag the `.CDF` into the level and then move your camera back to watch the LODs transition.

Currently, there is no way to test the LODs within **Geppetto**. You can use the console variable `e_DebugDraw = 1` to display debug text information regarding LODs. However, it may difficult to read if you have multiple skin attachments that have LODs, as the text stacks on top of each other.

LOD Distance Ratio

The **LOD Distance Ratio** is a ratio derived from the **Max view distance** and **View distance multiplier** values.

To easily test the LOD distance ratio's relation to the maximum view distance, leave the distance multipler set to `1`. A higher LOD distance ratio results in faster transitions through the LODs as the camera pulls away. Lower LOD distance ratio values effect slower transitions through the LODs.

For example, LOD transitions was tested for a tree that has LOD0-LOD2 and a **Max view distance** of `100` (m) and **LOD distance ratio** of `255`. The tree transitioned from LOD0 to LOD1 around 15m away, and then from LOD1 to LOD2 around 25m away. When LOD distance ratio was set to `64`, the tree transitioned from LOD0 to LOD1 around 40m away, and then from LOD1 to LOD2 around 75m away. Experimenting with these values showed that the calculation is not linear.

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-orients 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**.

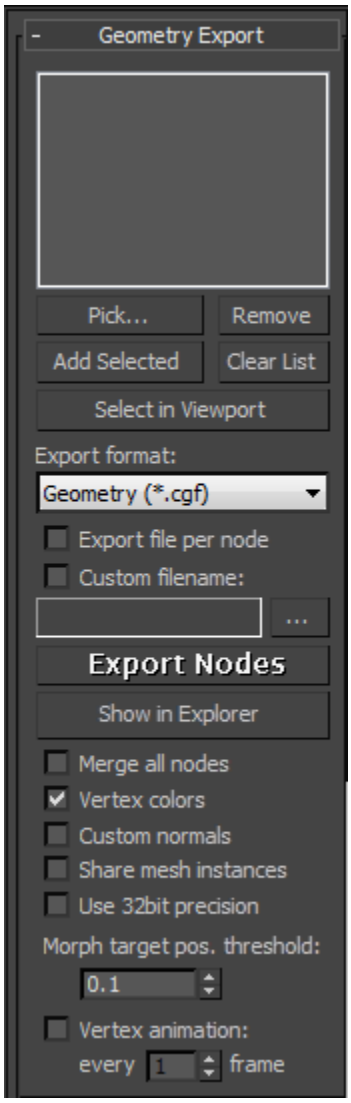
Topics

Exporting Static Meshes and Characters

Use the following procedure to export geometry and character geometry. You can specify which nodes in the scene to export, and other options about how they are exported. Child nodes are also exported.

When exporting character skin meshes, the 3ds Max exporter does not support combining multi-skin meshes into a single `.skin` file. You should export your multi-skin meshes as separate `.skin` files. To do so, add your skin meshes to the **Geometry Export** list and select the **Export file per node** check box. This will export each node in the list as a separate `.skin` file.

> **Note**
> If you are exporting multiple Proxy No Draw meshes, they must be children of a single object, such as a dummy object. This ensures 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

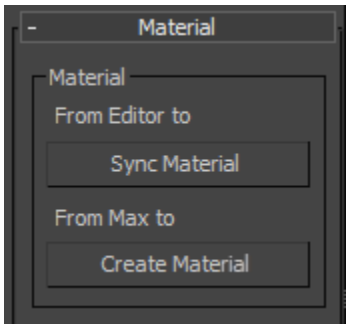
Option	Description
Export Format	Specify the file format for the exported file. Geometry export file formats include geometry (`*.cgf`), character (`*.chr`), character skeleton (`*.skel`), and character skin (`*.skin`).
Export file per node	Export each node in the export list as a separate file. The filename is generated from the node name.

Option	Description
Custom filename	Override the default export filename if **Export File per Node** is not selected.
Merge All Nodes	Compile non-skinned geometry from multiple nodes into a single node.
Vertex Colors	Export vertex colors.
Use 32-bit precision	If enabled, adds 32-bits of precision to position each vertex accurately when the mesh is located far from its pivot.
Morph target pos. threshold	Ignore vertices that don't move at least the specified distance (in meters) when the morph target is exported.
Vertex animation	Not currently supported.

Exporting Materials

Use the following procedure to export materials.

> **Note**
> When exporting a multi-material with one of the sub-materials for your proxy collision, you must assign the proxy collision sub-material to **MaterialID 1**. This allows the proxy collision mesh to process correctly in Lumberyard.



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**.

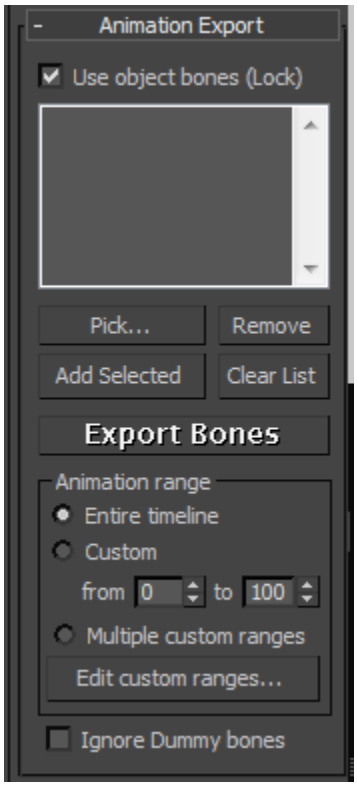
Exporting Bones and Animations

Animation Export contains the settings for exporting skeleton and animations for skinned character models. When you add a node to the **Geometry Export** list, its skeleton root bone is also added to the **Animation Export** list. When exporting your skeleton bones (`.chr`), you should have only one node in the **Geometry Export** list.

You typically don't need to configure the **Animation Export** settings if you have your skin mesh node listed in the **Geometry Export** list. However, the ability to directly edit this list may be helpful. For example, if you want 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. You can set this value in the **Time Configuration** dialog under **Frame Rate**.

Use the following procedure to export character skeleton bones. 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**.
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)	Use the bone of the geometry target listed in **Geometry Export**.
Ignore Dummy bones	Prevent exporting any dummy bones that are in the bone hierarchy.

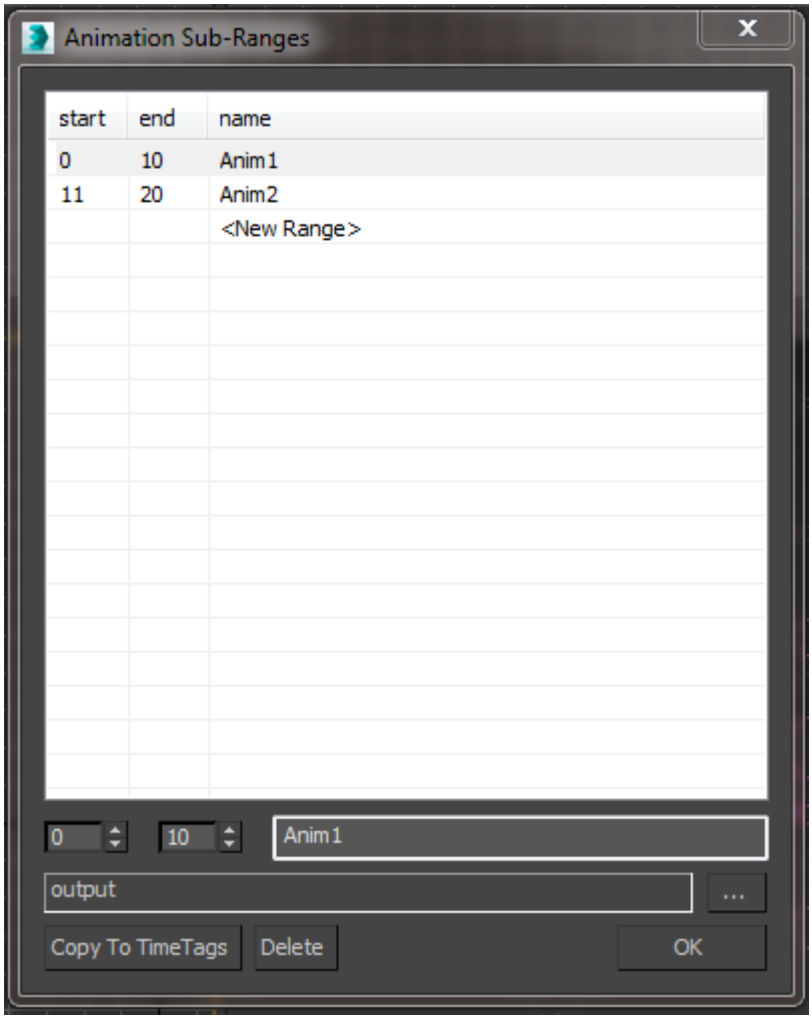
5. (Optional) In order to export animation data, within the **Animation range** you can specify the animation range for a character's skeleton. Use the various parameter options in the following table.

Animation Range parameters

Parameter	Description
Entire timeline	Use the full timeline length.
Custom	Use the customized length by specifying the start and end frames.
Multiple custom ranges	Use 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. Click the (**...**) icon and then choose an export file path for the animation range.



Exporting a Level of Detail (LOD)

Level of detail (LOD) 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 per group node in 3ds Max. The LOD numbering is from 0 (highest level of detail) to 5 (lowest level of detail).

You can export the following LODs from 3ds Max to Lumberyard:

- Static Mesh LODs
- Material LODs

Static Mesh LODs

Use the following naming and setup to export Static Mesh LODs using the 3ds Max Lumberyard tool.

LOD Naming

LOD naming conventions are very important with respect to prefixes and suffixes. You must use the following naming conventions:

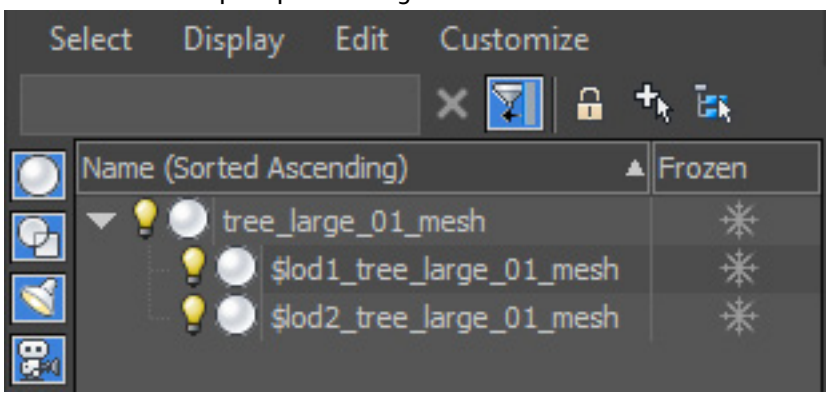
The highest LOD mesh (LOD 0) does not need a prefix.

`$lod1_` through `$lod5_` (prefix)

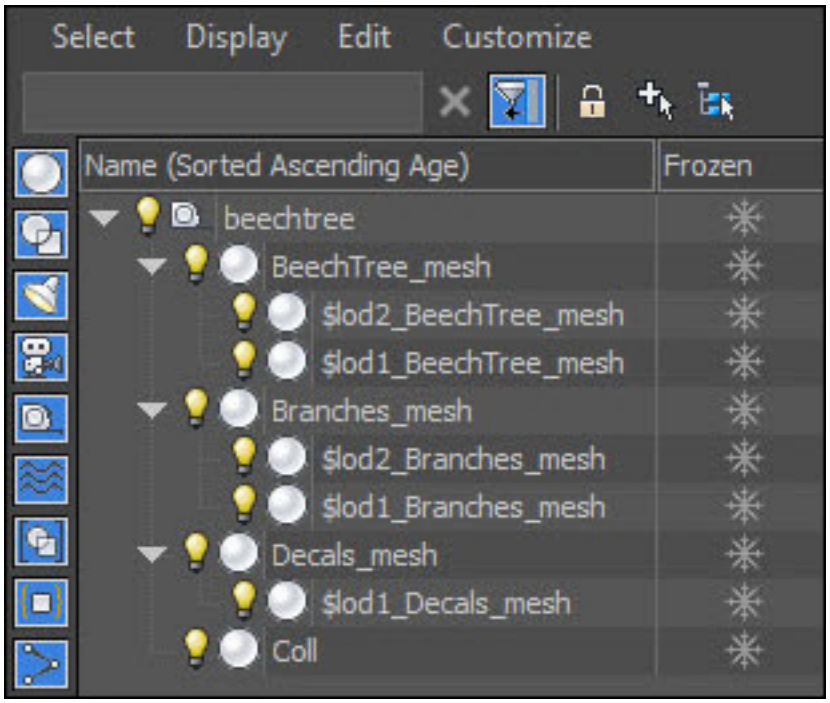
LOD Setup

You can create a basic or advanced LOD setup.

Basic LOD Setup – All LOD meshes with the appropriate prefix must 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 LOD meshes with the appropriate prefix must 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.



Material LODs

When you set up material LODs, you create submaterials and assign them to the appropriate LOD mesh.

For example, you might have a material group named `Tree_Mats`.

Within `Tree_Mats`, you set up five submaterials. The first three submaterials you assign to LOD0, which is the highest poly model. The other two submaterials you assign to the LOD1 mesh. This means that when LOD0 is active, it uses the first three submaterials you assigned to it. When LOD1 is active, it uses only the two submaterials you assigned to it.

Debugging LODs

Use the following console variables to debug 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.

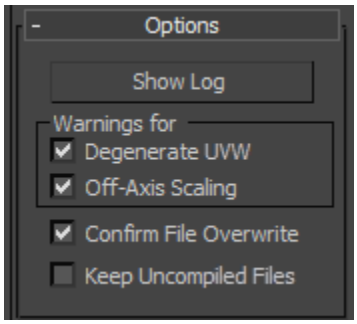
Configuring Log Options

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**.



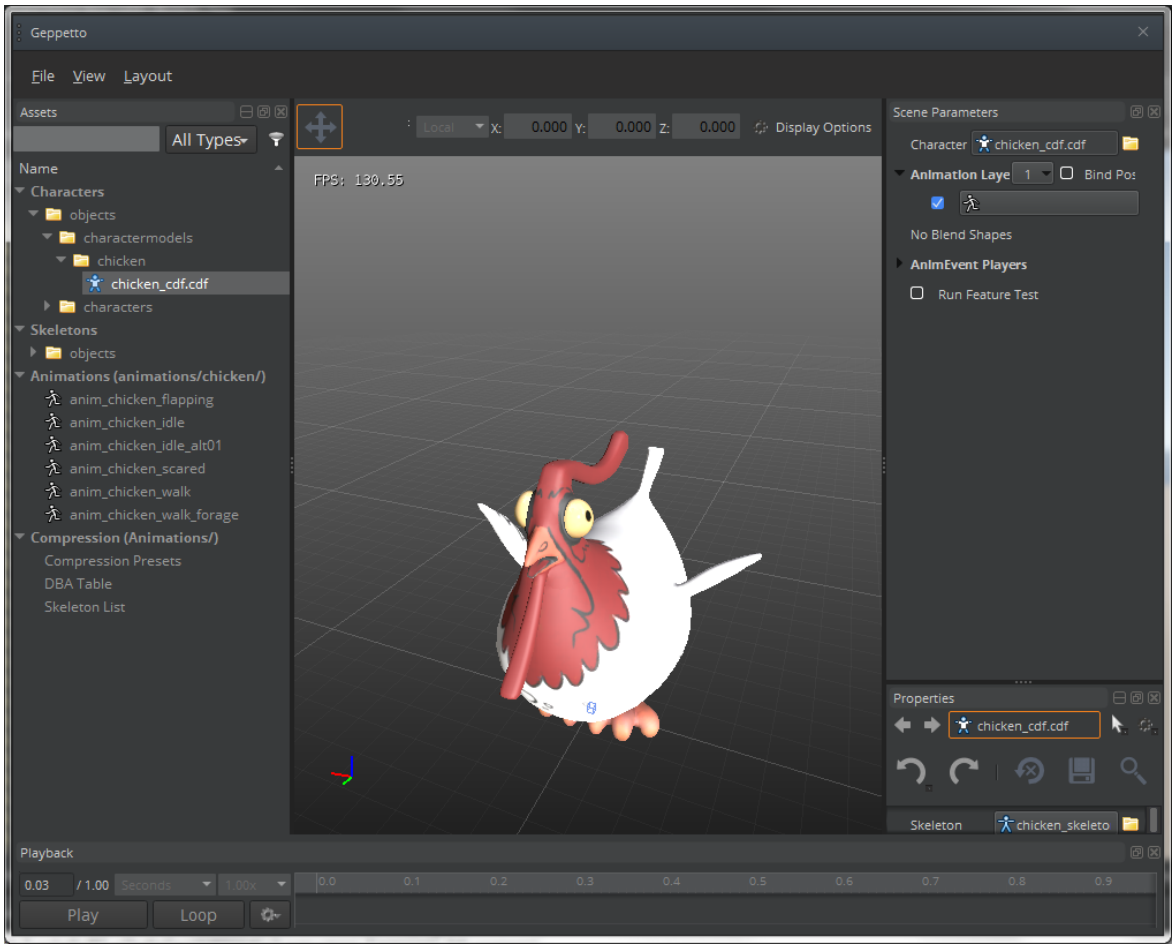
Other Options

Option	Description
Degenerate UVW	Check for degenerate texture coordinates and issue a warning if they exist; otherwise, silently export them. Degenerate coordinates arise when two vertices on a triangle have the same (or very nearly the same) UVs.
Off-axis scaling	Check whether the node is scaled along a non-primary axis. You can still export the node, but the scale won't match the object in 3ds Max.

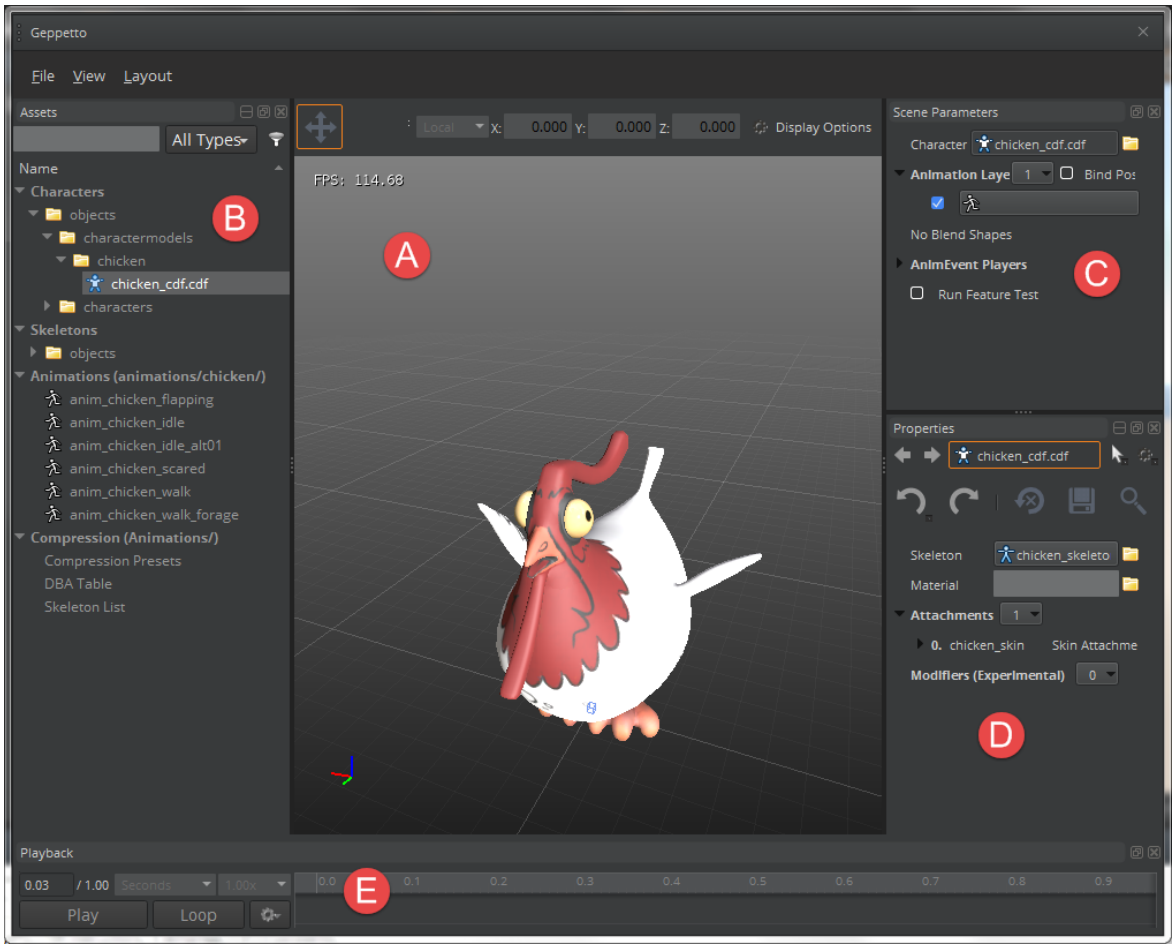
Using Geppetto

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 **Tools**, **Geppetto**. Geppetto has the following UI:



A. Viewport window

Displays the loaded character. Use the WASD keyboard keys 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

The following is a list of display option settings in Geppetto. In Geppetto, choose **Display Options** to access the various settings.

Animation

You can specify the following animation options:

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

You can specify the following rendering options:

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

You can specify the following skeleton options:

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

You can specify the following camera options:

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

You can specify the following movement smoothing options:

Position

- Slider for adjusting smoothing for the camera translation.

Rotation

Slider for adjusting smoothing for the camera rotation.

Follow Joint

You can specify the following follow joint options:

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

You can specify the following secondary animation options:

Dynamic Proxies

Enables and disables the display of dynamic proxies.

Auxiliary Proxies

Enables and disables the display of auxiliary proxies.

Physics

You can specify the following physics options:

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

You can specify the following grid options:

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

You can specify the following lighting options:

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

You can specify the following background options:

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. 284).

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. 284).

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

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. 298).

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, click 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. 298).
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, click 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.
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.

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. 298).
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

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 **PRow Attachment**.
4. For **Joint Row Name**, choose the bone icon, then open the applicable joint to place the socket on.
5. 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.

Parameter	Description
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.
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.

Parameter	Description
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:

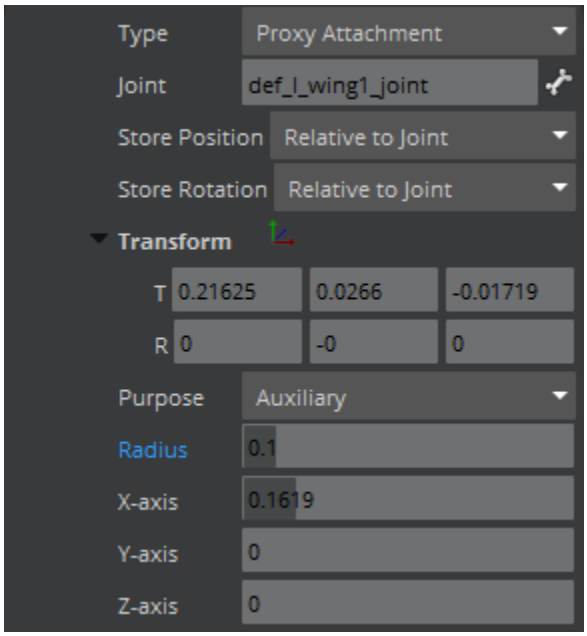
- **Auxiliary proxies (lozenges)** – Called lozenges, these are represented in gray by simple geometric objects linked to joints that move with the skeleton, and represent an approximation of a body shape. Gray proxies handle collision detection and response with the character and are normal attachments.
- **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. 293).

Skin Attachments

Skin attachments have their own skeleton, so that you can replace body parts, such as heads, hands, or upper and lower body parts. Also, these body parts are automatically animated and deformed by the base skeleton. Skeleton extensions support skinned attachments that have more joints and different joints than the base skeleton. You can also merge together different types of skeletons, even skeletons from different characters.

To create a skin attachment in Geppetto

1. In Geppetto, in the **Properties** pane, click the number next to **Attachments** and then choose **Add** or **Insert**.
2. For **Name**, type a name for the attachment.
3. For **Type**, choose **Skin Attachment**.
4. For **Geometry**, click the folder icon and then select the `.skin` file for the attachment.
5. For **Material**, click the folder icon and then select the `.mtl` file for the attachment.
6. Specify the following skin attachment parameters for the desired result.

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 selected, the mesh gets skinned on the CPU instead of the GPU. Software skinning is required for blend shapes and to have tangent frames recalculated every frame.
Linear Skinning GPU	If selected, sends the bone transforms as dual quaternions to the GPU and converts them to matrices on the GPU before blending them together for scaling. For more information, see Enabling Linear Skinning with Geppetto (p. 290).
Linear Skinning CPU	If selected, converts the bone transforms to matrices on the CPU before sending the matrices to the GPU for skinning. For more information, see Enabling Linear Skinning with Geppetto (p. 290).

Enabling Linear Skinning with Geppetto

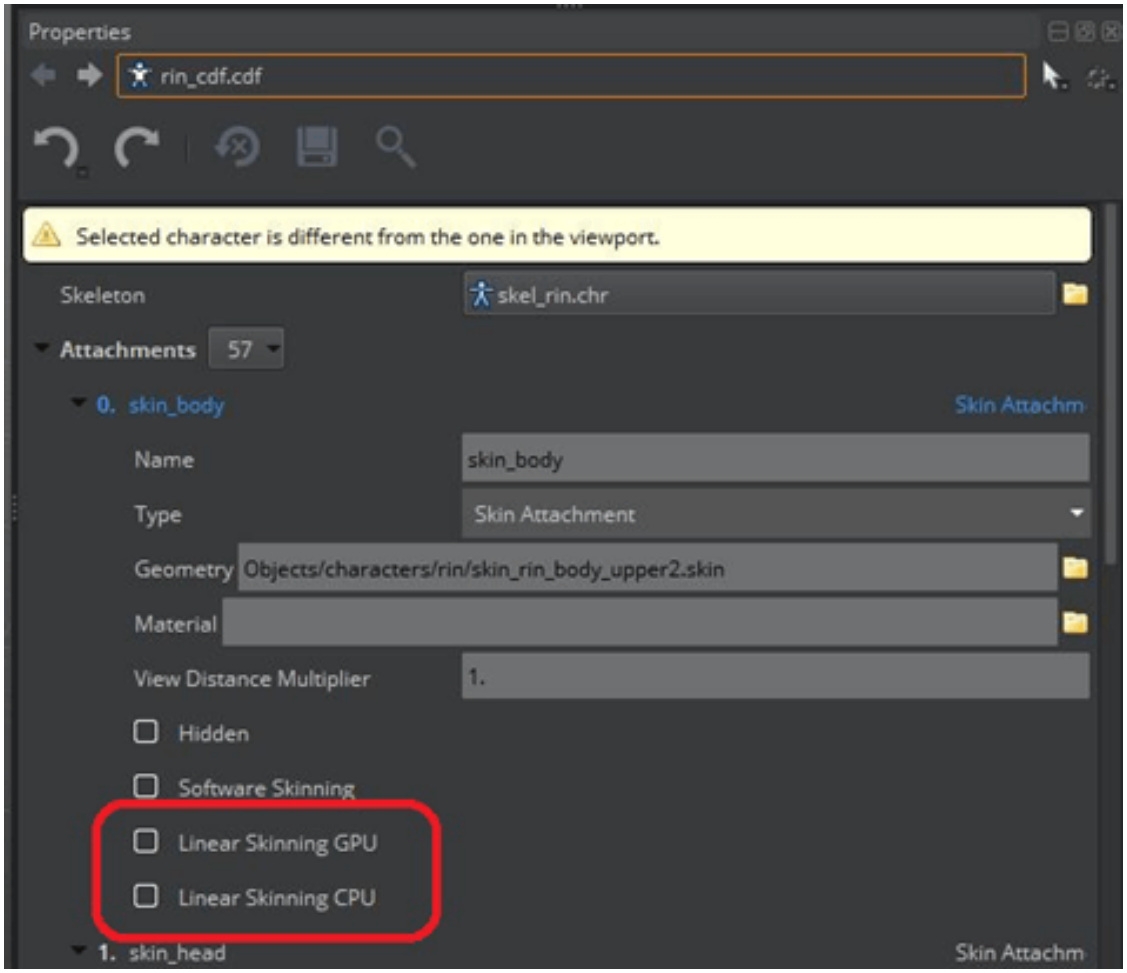
You can enable linear skinning for animated characters so character skinning uses matrices instead of dual quaternions. Linear skinning can remove some of the artifacts that appear with dual quaternions, such as bulging around joints.

> **Note**
> Skinning matrices can present their own artifacts, so consider both dual quaternion and linear skinning for your characters.

To enable linear skinning

1. In Lumberyard Editor, choose **Tools**, **Legacy/Animation**, **Geppetto Editor**.
2. In the **Assets** pane, choose a `.cdf` file with a skin attachment.
3. In the **Properties** pane, choose **Attachments** to display the skin attachment options.
4. Choose one of the following options:
   - **Linear Skinning GPU** – Sends the bone transforms as dual quaternions to the GPU and converts them to matrices on the GPU before blending them together for scaling.

- **Linear Skinning CPU** – Converts the bone transforms to matrices on the CPU before sending the matrices to the GPU for skinning.



Note
Both options display identical visual results, but as a best practice, use **Linear Skinning GPU** for better performance.
If you choose both options, **Linear Skinning CPU** overrides **Linear Skinning GPU**.

The following example uses dual quaternion skinning, which is the default if you don't enable linear skinning. Notice the bulging around the joints.

The following example enables linear skinning. Notice that the bulging around the joints is reduced.

Note
You can also enable linear skinning with the **Actor (p. 448)** component.

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. 288).

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

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.

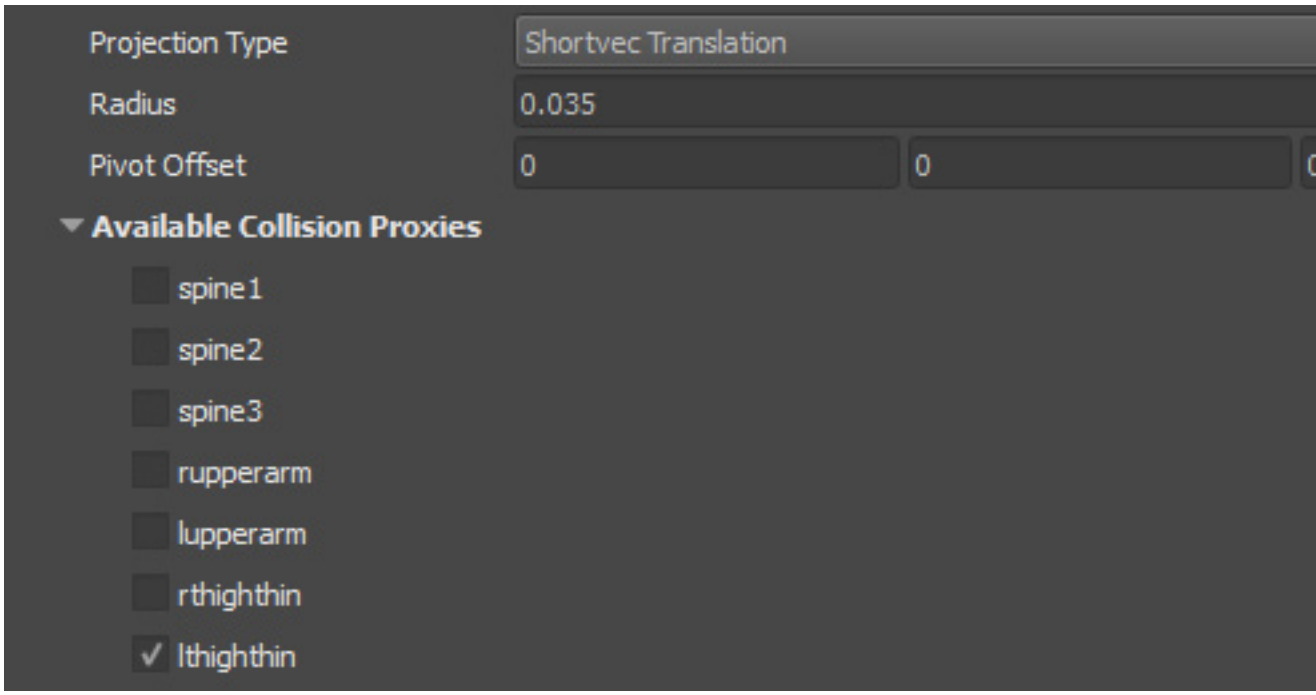
If the capsule pivot lies inside of a lozenge, collisions cannot be resolved and the proxies remain in an overlapping (collided) state. This can occur for secondary animations on characters where the simulation update is triggered after the animation update and it happens that the animation itself moves 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.

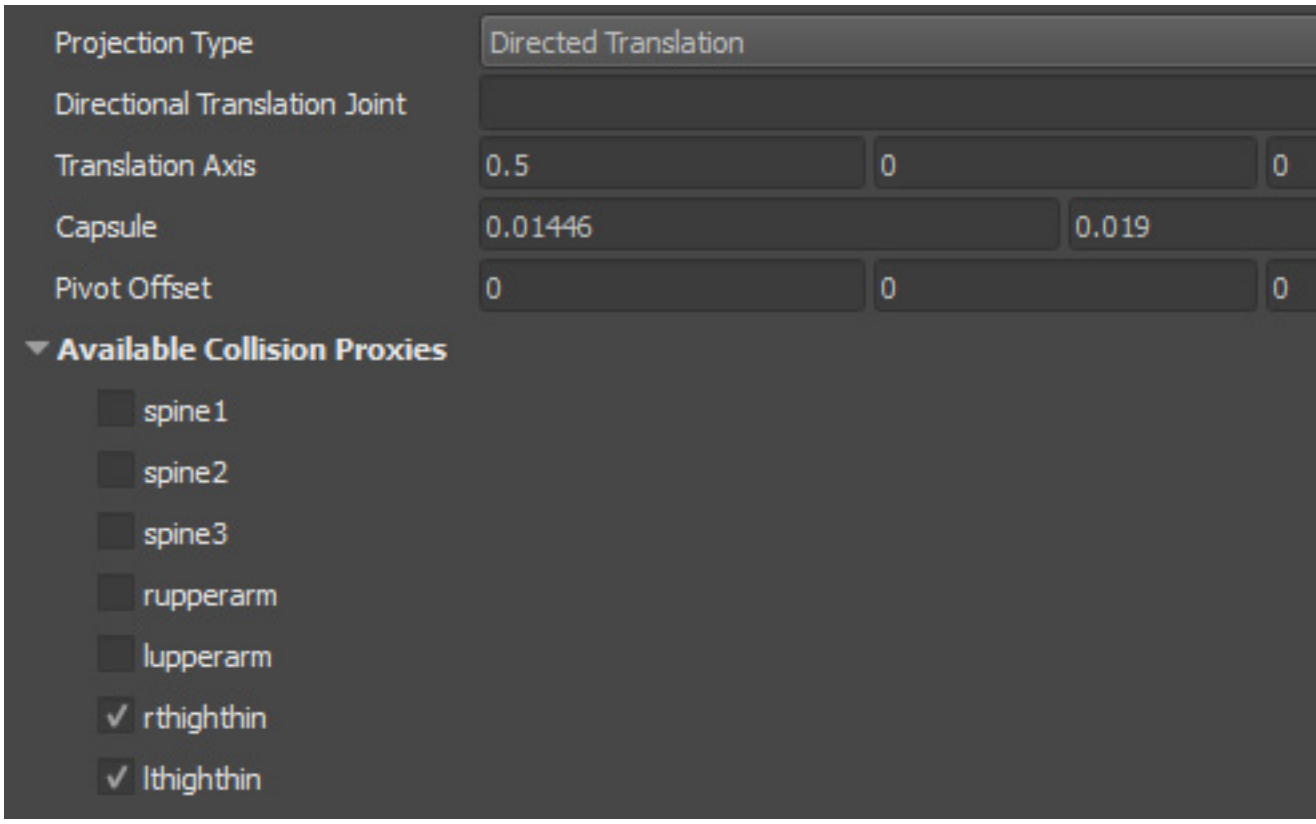
Selecting **Shortvec Translation** moves a blue proxy capsule out of a gray proxy lozenge along the shortest vector from the surface of a sphere enclosing the joint, where the radius of the sphere is specified. This type should be used in cases where there are only a few lozenges, due to potential unpredictable and undesirable movements.

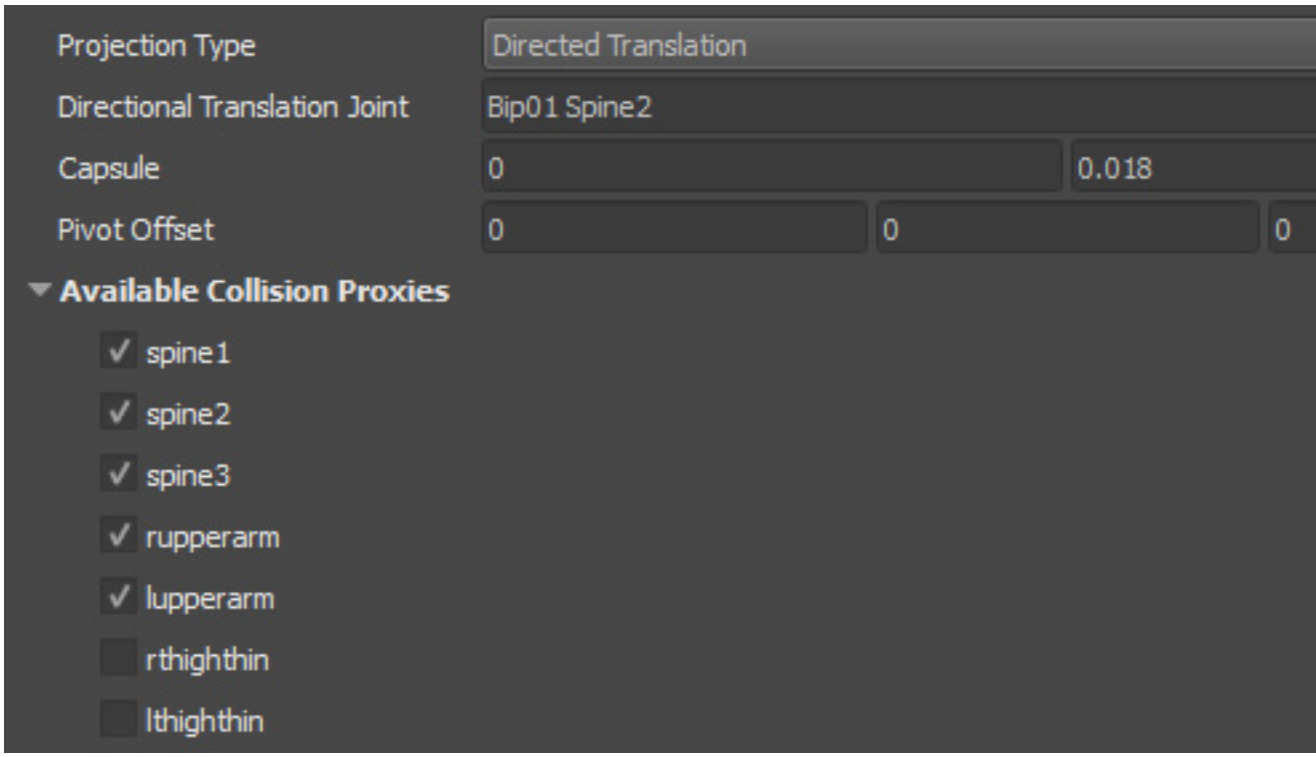
If an overlap is detected, the sphere is translated out of the lozenge along the shortest vector to the surface. This method of translation is only recommended for setups with just a few lozenges where the results are predictable. Otherwise, issues may arise where the first shortvec translation moves 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.


Projection Type
Directed Translation
Directional Translation Joint
Translation Axis
0.5
0
0
Capsule
0.01446
0.019
Pivot Offset
0
0
0
Available Collision Proxies
spine1
spine2
spine3
rupperarm
lupperarm
rthighthin
lthighthin


Projection Type
Directed Translation
Directional Translation Joint
Bip01 Spine2
Capsule
0
0.018
Pivot Offset
0
0
0
Available Collision Proxies
spine1
spine2
spine3
rupperarm
lupperarm
rthighthin
lthighthin

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-word 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.

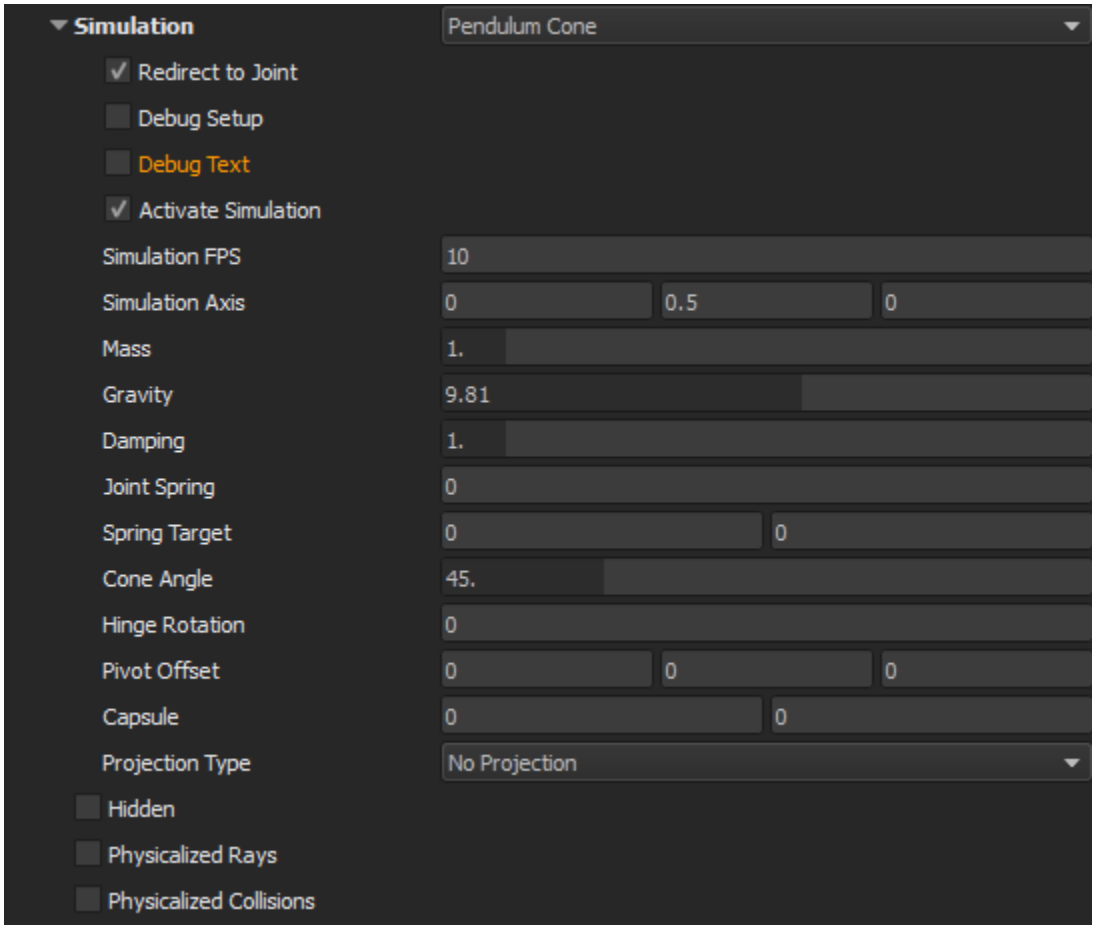
Moving springs and pendula have different motion bounding volumes that constrain the movement of objects attached to characters.

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. 288) and Collision Detection and Response (p. 293).

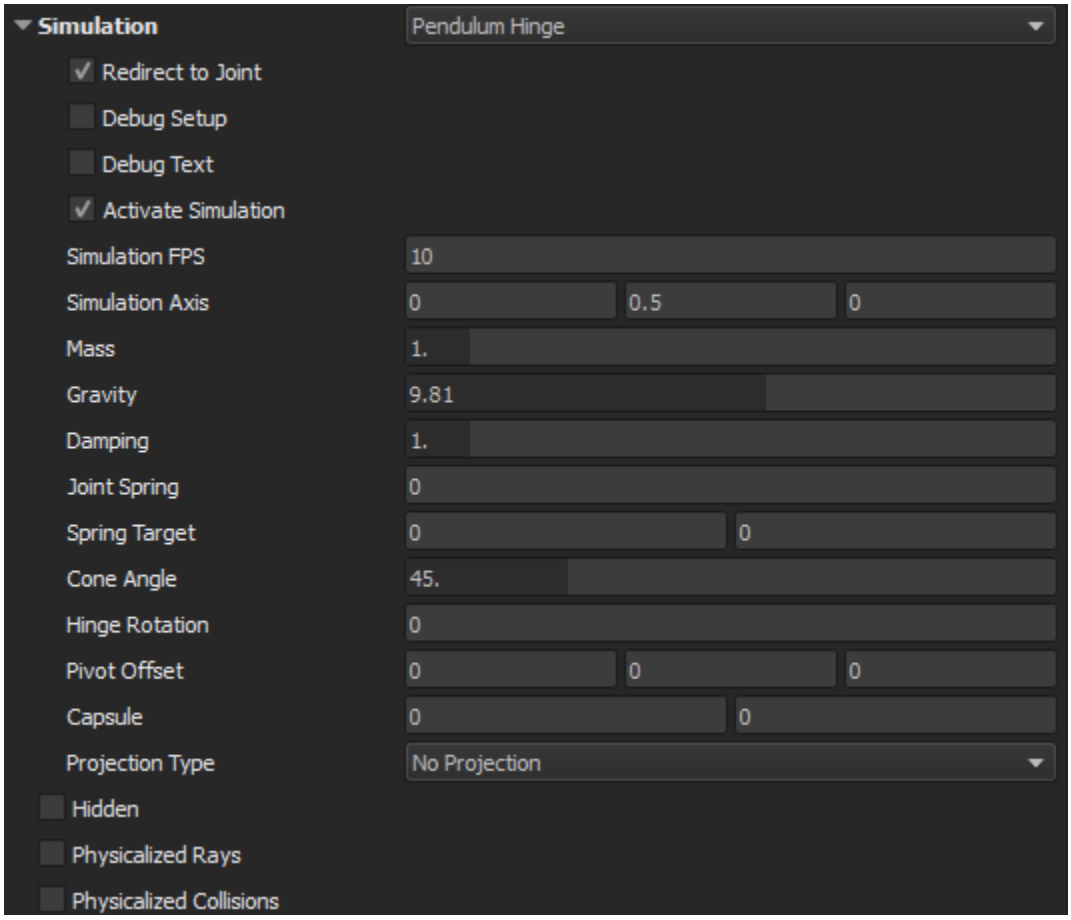
Topics

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.

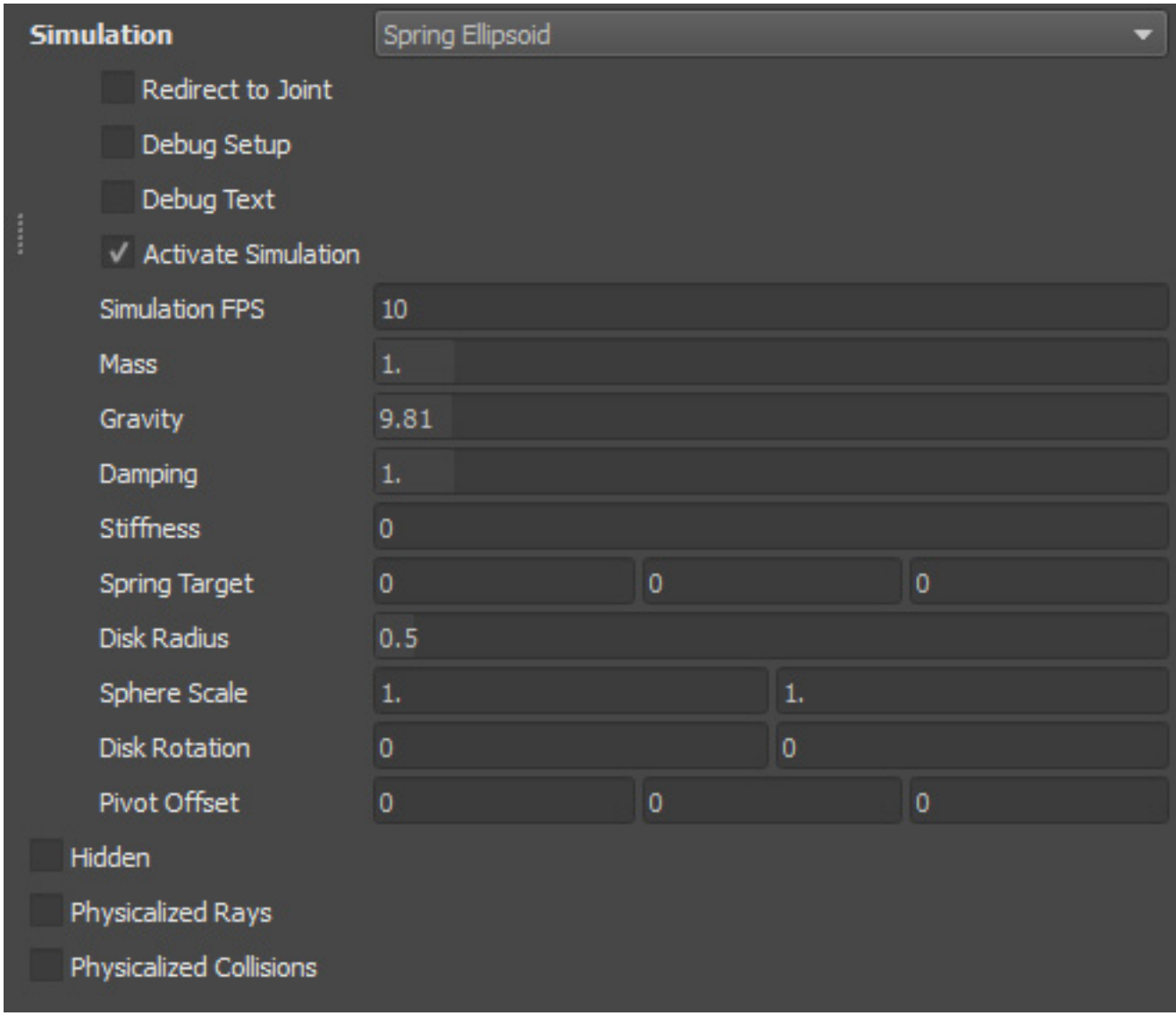
By default, it is the pivot of the model (offset = 0,0,0) and those three values are an x,y,z axes offset that translates the rendered geometry in the direction of the socket axes.

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

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. 276).
- **Mannequin Editor** – High-level system that manages animation variations, transitions, sequences, blends, layers, and procedural logic. For more information, see Using Mannequin Editor (p. 329).

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 `DbaTable.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 (DbaTable.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"/>
    <Animation Path="Animations\human\male\hits\1p
\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. .

Using Geppetto to Create a Basic Chrparams File

Geppetto allows you to set up mapping for animations, animation events, database animations, bounding box extensions, bounding box includes, joint LODs, and IK definitions. At a minimum you can use the `.chrparams` file to map animations to a specific character skeleton. You can also include other `.chrparams` settings. This topic provides details about each property and how to apply them to a `.chrparams` file.

Includes and Included Animation Set Filter

You can include another `.chrparams` Animation Sets for the currently loaded `.chrparams` file to use. Animation set data for the included `.chrparams` file is listed under the **Included Animation Set Filter** property, which only displays read-only data. This may be useful if you have characters that can

share animations because their skeletons are identical, but one particular version has a unique subset of animations.

To add an Includes entry to a `.chrparams` file using Geppetto

1. In Lumberyard Editor, click **Tools**, **Geppetto**.
2. In **Geppetto**, in the **Assets** panel, expand **Skeletons** and navigate to a character's skeleton (`*.chrparams`).
3. Select the `.chrparams` file.
4. In the **Properties** panel, click the **Includes** drop-down list and select **Add**.
5. Click the folder and assign a `.chrparams` file to include.
6. The **Included Animation Set Filter** field adds new entries based on what exists within the `.chrparams` file that you added to the **Includes** property. These properties are read-only and can only be modified by loading the original `.chrparams` file to which the properties belong.
7. Add additional `.chrparams` files as needed.
8. In the **Properties** panel, click **Save** (disk icon) to save your changes to the `.chrparams` file.

Animation Set Filter

You must assign an animation set (path) in order for a character's skeleton to recognize where animations are located to use. You can apply multiple filters if there are several animation directories. Further specify each animation set (path) filter that you have applied by setting naming filters or file extension filters.

To add an animation set to a `.chrparams` file using Geppetto

1. In Lumberyard Editor, click **Tools**, **Geppetto**.
2. In **Geppetto**, in the **Assets** panel, expand **Skeletons** and navigate to a character's skeleton (`*.chrparams`).
3. Select the `.chrparams` file.
4. In the **Properties** panel, click the **Animation Set Filter** drop-down list and select **Add**.
5. The **Animation Set Filter** field adds an empty property field with additional fields below.
6. Click the folder next to the empty field to set the directory where the animations for the character's skeleton are located. By default, there are three filters under the path to search for any named animation with the `.caf`, `.bspace`, and `.comb` extensions in the assigned directory and subdirectories within the path.

   Note
   The left field represents the alias name of the asset to use in-game. The asterisk (*) in this field represents a use-default-name or a pass-through. In the right field, the first asterisk (*) in `*/*.caf` represents including all subfolders. The second asterisk (*) includes all found `.caf` files. For example, setting **`run_loop`** in the left field and **`chicken_run.caf`** in the right field will assign the alias **`run_loop`** to the **`chicken_run.caf`**. Engine systems would then use **`run_loop`** as the animation name.
7. (Optional) Add additional filters to the assigned animation directory path by clicking the drop-down menu next to the folder icon that allowed you to assign your animation directory path.
8. In the **Properties** panel, click **Save** (disk icon) to save your changes to the `.chrparams` file.
9. You can add animation directories as needed and modify the subfields if you need to set specific filters for your animations.
10. When you load a `.cdf` file in Geppetto using the skeleton, you will see the animations from the assigned directory listed under **Animations** in the **Assets** panel.

Animation Events

You must map an `.animevents` file to the `.chrparams` file in order for animation events to be applied to the animations that are available for a character's skeleton. Only one `.animevents` file can be assigned per `.chrparams` file.

You can do either of the following:

- Manually create an `.animevents` file and assign it to the `.chrparams` file.
- Use Geppetto to automatically create the `.animevents` file and assign it to the `.chrparams` file. At least one animation must be available to play on the character's `.cdf`.

To manually add an AnimEvents path to a `.chrparams` file

1. In Lumberyard Editor, click **Tools**, **Geppetto**.
2. In **Geppetto**, in the **Assets** panel, expand **Skeletons** and navigate to a character's skeleton (`*.chrparams`).
3. Select the `.chrparams` file.
4. In the **Properties** panel, click the folder next to the **Events** field.
5. Select the directory where the `.animevents` file exists for the character's skeleton.

   Note
   Only one `.animevents` file can be assigned per `.chrparams` file.
6. In the **Properties** panel, click **Save** (disk icon) to save your changes to the `.chrparams` file.

To automatically create an `.animevents` file and assign it to a `.chrparams` file

1. In Lumberyard Editor, click **Tools**, **Geppetto**.
2. In **Geppetto**, in the **Assets** panel, expand **Characters** and double-click the character for which you want to create an `.animevents` file.
3. In the **Assets** panel, under **Animations**, select a character animation to load.
4. In the **Playback** panel, double-click anywhere in the timeline to add an AnimEvent.
5. In the **Properties** panel, under **Animation Events**, change the settings and properties of the AnimEvent.
6. In the **Properties** panel, click **Save** (disk icon) to save your changes and create the `.animevents` file.
7. In the **Assets** panel, expand **Skeletons** and select the character's skeleton (`*.chrparams`).
8. In the **Properties** panel, verify that the `.animevents` file appears in the **Events** field.

   Note
   The `.animevents` file is created in the same location as the animation with the added AnimEvent. Ensure that you check in this file if you are using source control.
9. In the **Properties** panel, click **Save** (disk icon) to save the `.animevents` file to the `.chrparams`.

Database Animations (DBA) Path

You must assign the path for the directory that contains multiple database animations (`.dba`) files to the skeleton's `.chrparams` file in order for a character's skeleton to read the animations.

To add a DBA path to a `.chrparams` file using Geppetto

1. In Lumberyard Editor, click **Tools**, **Geppetto**.
2. In **Geppetto**, in the **Assets** panel, expand **Skeletons** and navigate to a character's skeleton (`*.chrparams`).

3. Select the `.chrparams` file.
4. In the **Properties** panel, click the folder next to the **DBA Path** field.
5. Select the directory where the `.dba` files exist.
6. In the **Properties** panel, click **Save** (disk icon) to save your changes to the `.chrparams` file.

Individual Database Animations (DBAs)

You must assign the path of the database animation (`.dba`) file to the skeleton's `.chrparams` file in order for a character's skeleton to read animations from a single `.dba` file.

To add an individual DBA path to a `.chrparams` file using Geppetto

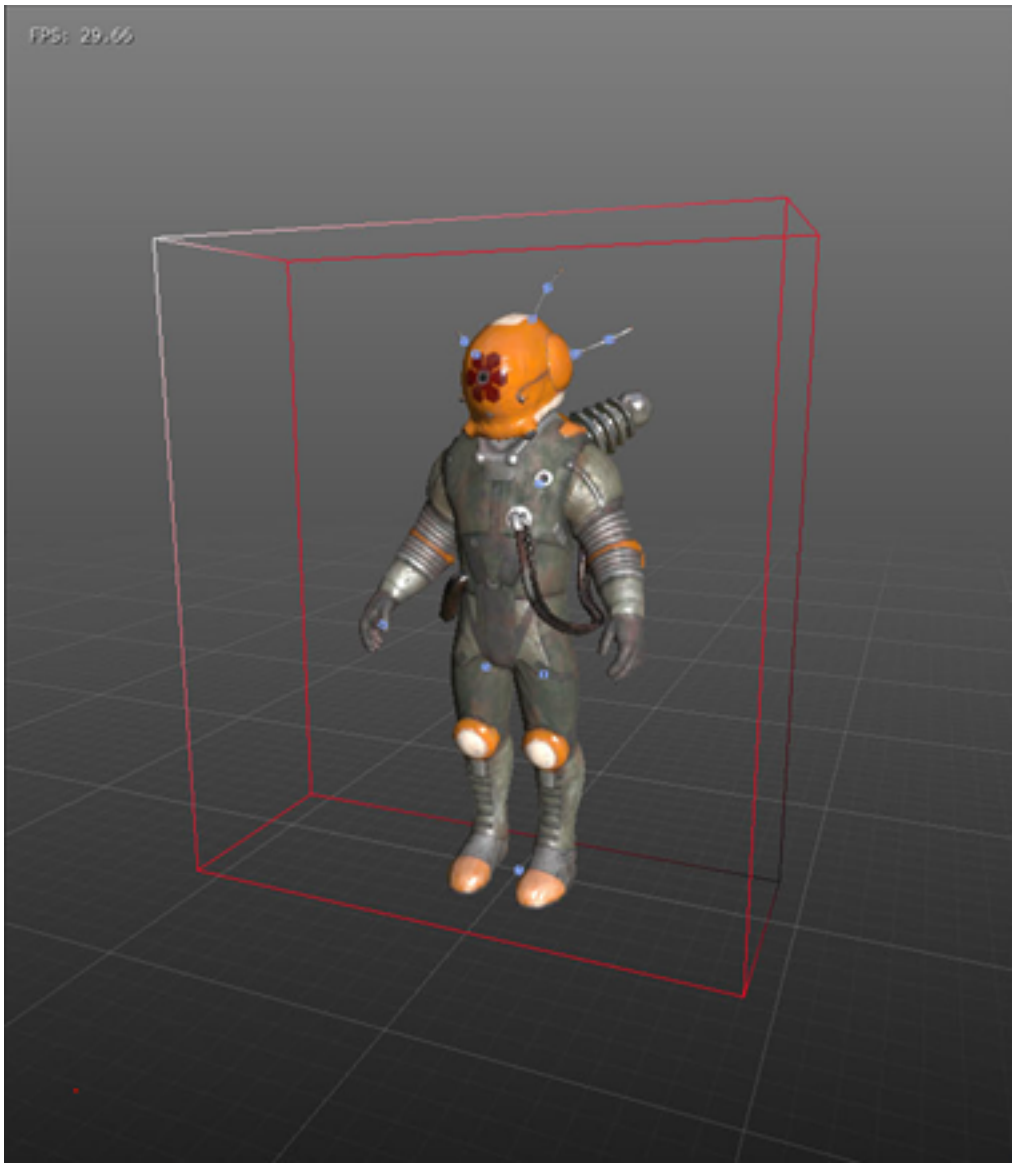
1. In Lumberyard Editor, click **Tools**, **Geppetto**.
2. In **Geppetto**, in the **Assets** panel, expand **Skeletons** and navigate to a character's skeleton (`*.chrparams`).
3. Select the `.chrparams` file.
4. In the **Properties** panel, click the **Individual DBAs** drop-down list and select **Add**.
5. Click the folder and assign an individual DBA. You can also select the **Persistent** check box to keep the DBA in memory after the character has loaded.
6. In the **Properties** panel, click **Save** (disk icon) to save your changes to the `.chrparams` file.

Bounding Box Extension

You can extend a skeleton's bounding box based on the size of the skeleton.

To add a bounding box extension to a `.chrparams` file using Geppetto

1. In Lumberyard Editor, click **Tools**, **Geppetto**.
2. In **Geppetto**, in the **Assets** panel, expand **Skeletons** and navigate to a character's skeleton (`*.chrparams`).
3. Select the `.chrparams` file.
4. In the **Properties** panel, expand **Bounding Box Extension**.
5. Set the positive (**Pos**) and negative (**Neg**) values for the XYZ coordinates to extend the bounding box of the skeleton.
6. Preview the extended bounding box by loading a `.cdf` file that uses the skeleton's `.chrparams`. Click **Display Options**, **Skeleton**, and select the **Bounding Box** check box to see your bounding box on the character.



Bounding Box Include

A skeleton's bounding box is defined by the location of the joints within the skeleton. You can select which joints in the skeleton define the size of the bounding box. By default, all joints are included.

To modify the bounding box Include in a `.chrparams` file using Geppetto

1. In Lumberyard Editor, click **Tools**, **Geppetto**.
2. In **Geppetto**, in the **Assets** panel, expand **Skeletons** and navigate to a character's skeleton (`*.chrparams`).
3. Select the `.chrparams` file.
4. In the **Properties** panel, expand **Bounding Box Include**.
5. Select the skeleton joints to include for calculating the skeleton's bounding box.
6. In the **Properties** panel, click **Save** (disk icon) to save your changes to the `.chrparams` file.

Joint LOD

You can assign a skeleton's joints with an LOD so that certain joints stop being used after a certain LOD, based on camera distance. This can help cut down performance cost of reading all joint animation data. By default, all joints are enabled and do not use LOD.

> **Note**
> Joint LODs are treated globally to the skeleton joints. You can choose to include only specific joints on an individual animation when importing the animation.

To add a joint LOD to a `.chrparams` file using Geppetto

1. In Lumberyard Editor, click **Tools**, **Geppetto**.
2. In **Geppetto**, in the **Assets** panel, expand **Skeletons** and navigate to a character's skeleton (`*.chrparams`).
3. Select the `.chrparams` file.
4. In the **Properties** panel, click the **Joint LOD** drop-down list and select **Add**.
5. For the **LOD 1** entry, select the skeleton joints to enable.
6. Add additional entries and modify the incremental joint LODs.
7. In the **Properties** panel, click **Save** (disk icon) to save your changes to the `.chrparams` file.

IK Definition

The IK definition provides access to various IK types such as Limb IK, Aim IK, Look IK, Feet Lock IK, and Recoil IK. You can enable the IK types for a character's skeleton using the `.chrparams` properties.

To add an IK definition to a `.chrparams` file using Geppetto

1. In Lumberyard Editor, click **Tools**, **Geppetto**.
2. In **Geppetto**, in the **Assets** panel, expand **Skeletons** and navigate to a character's skeleton (`*.chrparams`).
3. Select the `.chrparams` file.
4. In the **Properties** panel, expand **IK Definition**.
5. Select the check boxes for the IK definitions that you need for your character. If there are additional settings, expand the IK definition.

   > **Note**
   > For information about setting up specific IK definitions, see Using Inverse Kinematics (IK) (p. 372).

6. In the **Properties** panel, click **Save** (disk icon) 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" />
        <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" />
        <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" />
      <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 Lumberyard Editor, click **Tools**, **Geppetto**.
2. In Geppetto, in the **Assets** panel, expand **Compression (Animations)** and click **Skeleton List**.
3. 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 Lumberyard Editor, click **Tools**, **Geppetto**.
2. In Geppetto, in the **Assets** panel, expand **Animations** and select the animation to import. All unimported animations become unavailable.
3. 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 beside **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 **Tools**, **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. In Lumberyard Editor, click **Tools**, **Geppetto**.
2. In **Geppetto**, click 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 check boxes 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. 260).

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**, **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 in the blend space, as a mini version of the character model. You can control the size of the model using 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 that the animation clips appear in the `.bspace` XML file, including all pseudo examples.
- A green wireframe quad shows which assets are currently considered in the blend. In a 2D blend space there are either triangles (blends between 3 assets) or quads (blends between 4 assets).
- A red flashing cursor appears inside a triangle or quad. You can control the cursor with the blend space sliders and see which assets contribute to the final blend.
- The current dimension values correspond 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.

To create the `.animevents` file

1. Using a text editor, create a `.xml` file.
2. In the `.xml` file, add the following tags: `<anim_event_list></anim_event_list>`.
3. Name the file and include the `.animevents` extension. Save the file in the same directory as the animations to which the file will apply.

To update the `.chrparams` file

1. In Geppetto, in the **Assets** panel, expand **Skeletons**, and navigate to and select the character's skeleton (`*.chrparams`) file.
2. In the **Properties** panel, locate 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
   You can assign only one `.animevents` file 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.
- The first and last keyframe of your animation cycle should match. The locomotion locator position relative to the character on the first keyframe should also match the position relative to the character on the last keyframe.
- 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 a legacy tool.

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.

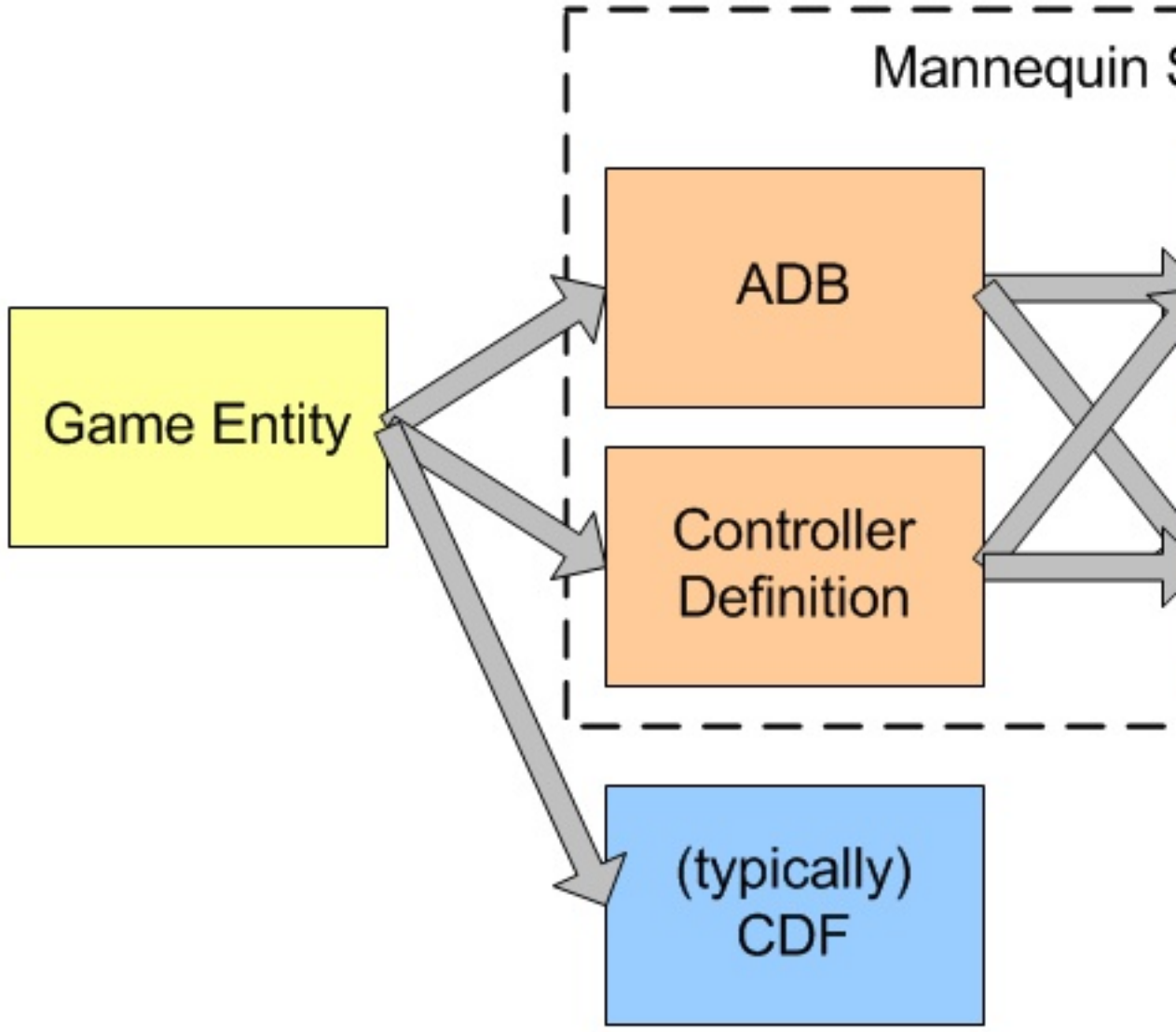
With Mannequin you can simplify complex animation code and avoid manually constructing this degree of realism. You can also author preview sequences using your fragments and transitions, reducing iteration time and allowing you to retest scenarios as your animation setup evolves. The Mannequin runtime 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

Mannequin is a legacy tool.

The Mannequin system uses a variety of file types.



With the exception of the `*.Sequence.xml` file, you must use a text editor to manually create all other `.xml` files. You must save these files in the `Animations\Mannequin` directory, and you can optionally create subfolders by character. See below for example files.

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 as 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

You must manually set up and edit some Mannequin files. Once these files are set up, you can use the Mannequin Editor to verify that the character displays in the viewport. To do so, click **File**, **Load Preview Setup**, and select the `*Preview.xml` file. You must select **File**, **Save Changes** any time a change is made.

When setting up the `*ControllerDefs.xml` file, the file name should match the character name for easier recognition. You should also reference this name in the `*Preview.xml` file.

To set up 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 `.adb`file.
7. Click **OK** in the **Edit Context** window.
8. Click **OK** in the **Context Editor** window.

Creating a Mannequin Entity

Mannequin is a legacy tool.

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, on 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 a legacy tool.

Mannequin Editor is the primary tool for creating and managing complex character animations.

To open Mannequin Editor

- In Lumberyard Editor, click **Tools**, **Animation**, **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 that are 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, email, or open 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 is a legacy tool.

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

Mannequin is a legacy tool.

Use the Mannequin EditorEditor to create, copy, and delete fragments.

To create, copy, or delete a fragment

Open Mannequin Editor (p. 329), 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 is a legacy tool.

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 is a legacy tool.

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 the 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 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. In Geppetto, select the animation from the **Animation list**, and then drag 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 draging 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 a legacy tool.

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. 345).

AISignal 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 - *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 – *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 – *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 – *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 – *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 `Param Name` 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 – *`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 `.fsq` 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 a legacy tool.

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. 346).

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 a legacy tool.

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 a legacy tool.

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 a legacy tool.

Typically, individuals 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

Mannequin is a legacy tool.

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` – Llists all subcontexts available.

- `ScopeDefs` – Defines the scopes and scope contexts used. Each element defines a scope.

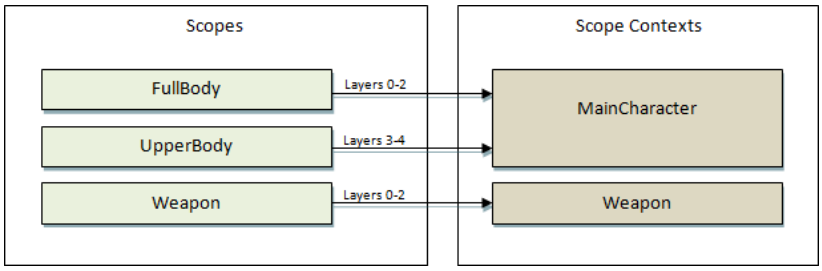
Creating and Editing Scope Contexts

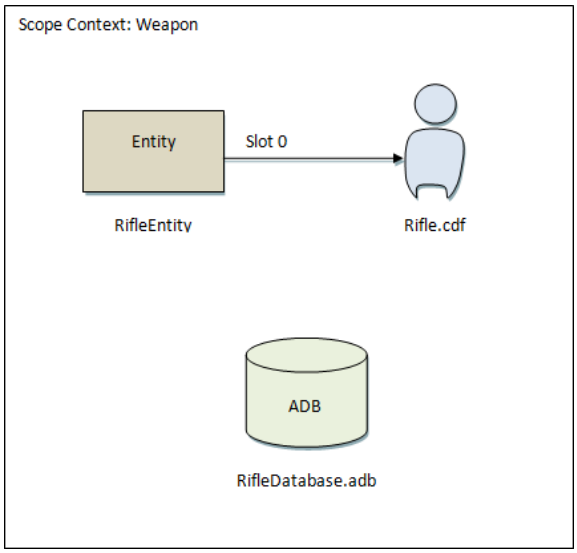
Mannequin is a legacy tool.

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 a legacy tool.

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 a legacy tool.

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 a legacy tool.

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 can 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

Mannequin is a legacy tool.

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 a legacy tool.

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> - o}` 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 a legacy tool.

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 a legacy tool.

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 a legacy tool.

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. 351)
- Setting Transition Parameters (p. 352)
- Cyclic Transitions (p. 352)

Creating and Editing Transitions

Mannequin is a legacy tool.

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, on the **Transitions** tab, click **New**.
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 a legacy tool.

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 a legacy tool.

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 a legacy tool.

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

Mannequin is a legacy tool.

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 a legacy tool.

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 queueing 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 a legacy tool.

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
IActionController* pVehicleActionController;
IAction* 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 a legacy tool.

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.

Using the New Mannequin Setup Window

Mannequin is a legacy tool.

You can use the Mannequin system to create complex, interactive character animations and define motions that are variations on a theme. For example, you can create a family of motions based on a running theme: running while injured, running while exhausted, running slowly, and so on.

Mannequin supports animation layering, blending, additive animations, and partial body animations. Each character animation requires the following Mannequin setup:

- **Controller definition file** (`*ControllerDefs.xml`) – Defines the mannequin setup.

- **Animation database file** (`.adb`) – Stores fragments and transitions. A fragment is a variation in a family of motions. A transition is used to move smoothly between families of motions.

- **Tag definition file** (`*Tags.xml`) – Stores tag definitions. You use tags to categorize fragments that you can then select during playback.

- **Fragment ID definition file** (`*FragmentIDs.xml`) – Stores fragment ID definitions. Fragments that share a fragment ID are grouped into families.

Lumberyard provides a setup tool that generates these files for a mannequin setup.

For more information about these file types and how they are used, see Mannequin System Files (p. 325).

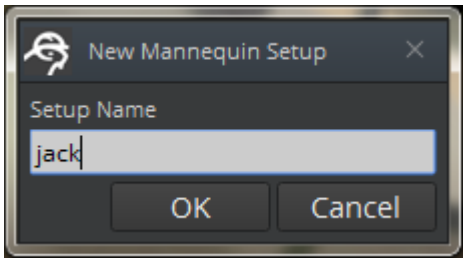
Creating Mannequin Setup Files

You can use the **New Mannequin Setup** window to create your mannequin setup and generate the necessary files.

To create mannequin setup files

1. In Lumberyard Editor, choose **Tools**, **Animation**, **Mannequin Editor**.
2. In the **Mannequin Preview** dialog box, click **Create**.
3. In the **New Mannequin Setup** dialog box, type a name and then click **OK**.

   The setup name should reflect your character or class of characters. The setup tool creates `.xml` files with the name you specify as the default file name.

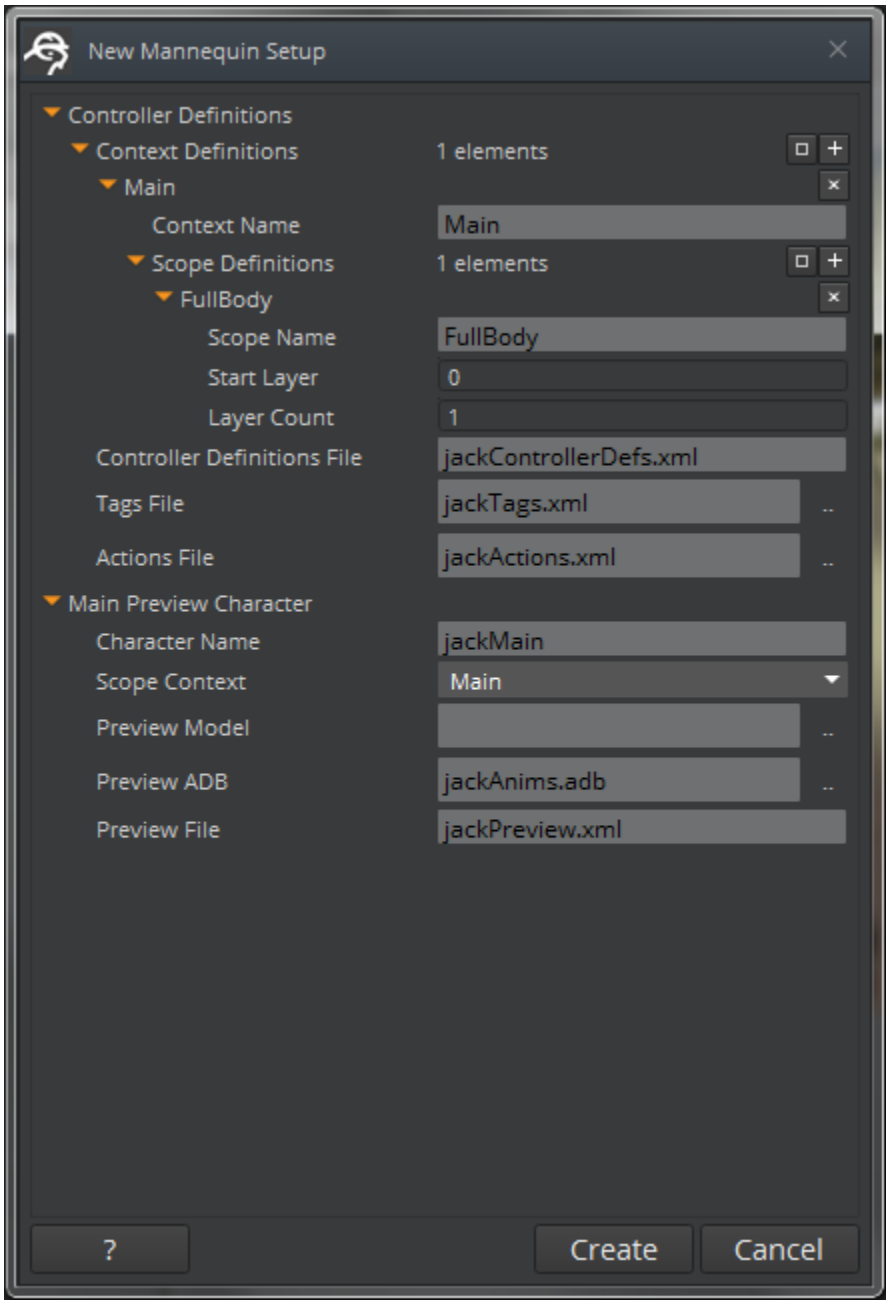
4. In the **New Mannequin Setup** window, do the following:

   a. Review the provided information for the following:

      - **Controller Definitions File**
      - **Tags File**
      - **Actions File**
      - **Preview ADB**
      - **Preview File**

      These settings represent the `.xml` files that the setup tool will create. You can modify the file names; however, we recommend using the default file names.

   b. For **Preview Model**, click the browse (**..**) button to choose your character definition file (`.cdf`). You must choose a `.cdf` file in order to generate the preview files.



5. Click **Create**.
6. In the **Output Files** dialog box, click **Yes** to confirm the file names and directory paths. The setup tool creates your mannequin files and loads the preview file in the editor.

New Mannequin Setup Properties

In a mannequin setup, you can modify the properties and attributes for controller definitions and for the main preview character.

Controller Definitions

A controller definition includes all of the data used to define a mannequin setup.

Property	Description
Context Definitions	Specifies the category of the motion set. For example, Main, Player, Alien, Enemy, etc. You can add context definitions by clicking the **+** button. Click the **X** button to delete a context definition.
Context Name	Specifies the name of the context definition.
Scope Definitions	Categorizes the parts of a character. For example, you can create categories for the full body or isolate an animation set based on the torso, arms, legs, etc.
Scope Name	Specifies the name of a scope definition entry.
Start Layer	Specifies which of the 16 animation layers that Lumberyard supports per character to include in your scope definition. This property represents the first of the range of layers that the scope uses.
Layer Count	Specifies the number of layers to assign to the scope. This property represents the maximum number of animation layers for your fragments in the Mannequin **Fragment Editor** timeline.
Controller Definitions File	Defines a mannequin setup. This file is used by the **Mannequin Editor** and your game and is typically referred to from the character Lua file and Mannequin preview file. By default, the Mannequin setup tool generates a new `.xml` file.
Tags File	Stores tagging information in an `.xml` file. You can use tags in Mannequin to group families of animations. You can edit this file at any time using the Mannequin **Tag Definition Editor**. By default, the Mannequin setup tool generates a new `.xml` file. You can reference an existing file by clicking the browse (**..**) button and navigating to an existing tags file.
Actions File	Stores fragment ID definitions. This file is also called the FragmentID Definition File and is used by the **Mannequin Editor** and your game. The controller definition and animation database files refer to this file. By default, the Mannequin setup tool generates a new `.xml` file. You can reference an existing file by clicking the browse (**..**) button and navigating to an existing actions tags file.

Main Preview Character

The mannequin preview uses the controller definition data to test your mannequin setup while authoring.

Property	Description
Character Name	Specifies the name of the character in the preview window.
Scope Context	Selects the context definition to edit in the **Mannequin Editor**. The list displays the generated context definition categories.

Property	Description
Preview Model	Selects the character definition file (`.cdf`) to display in the preview. Character definition files include a combination of skeletons, skins, and attachments. These files are authored in Geppetto.
Preview ADB (Animation Database)	Stores the data for the Mannequin **Fragment Editor** timeline. By default, the Mannequin setup tool generates a new `.xml` file. You can reference an existing file by clicking the browse (**..**) button and navigating to an existing `.adb` file. The **Preview ADB** references the **Fragment Editor** but does not reference the **Transition Editor**, which is stored in the same file.
Preview File	Stores the character name, scope context, preview model, and animation database. You can access preview files by clicking **File**, **Load Preview Setup** in the **Mannequin Editor**.

Synchronizing Multiple Characters

Mannequin is a legacy tool.

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
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 a legacy tool.

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 a legacy tool.

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

Mannequin is a legacy tool.

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.

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 more simplified than the main skeleton and often uses capsules on limb joints for more accurate simulation. The two skeleton types have similar physics mesh for joint limits and spring attributes. Any differences are used to fine-tune the ragdoll simulation.

Topics

Ragdoll Best Practices

There are a few guidelines and best practices to follow when creating physicalized ragdoll characters:

- To help with performance, use simplified geometry (for example, boxes or capsules) for your phys mesh proxies whenever possible.
- In Maya, the physics skeleton does not use the `SceneRoot` node to determine the up axis in the scene.
- The root joint and first or highest hierarchical ragdoll joint, such as the pelvis or hip, must be oriented to match z-up.
- For self-collision to work correctly, use the following naming conventions (case-sensitive) for your skeleton joints:
  - Pelvis

- Spine
- Head
- UpperArm
- Forearm
- Thigh
- Calf

Ragdoll Skeleton DCC Setup

You can set up 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 physics mesh proxies based on the joint they represent and with the appropriate suffix. Examples:
   - 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`.

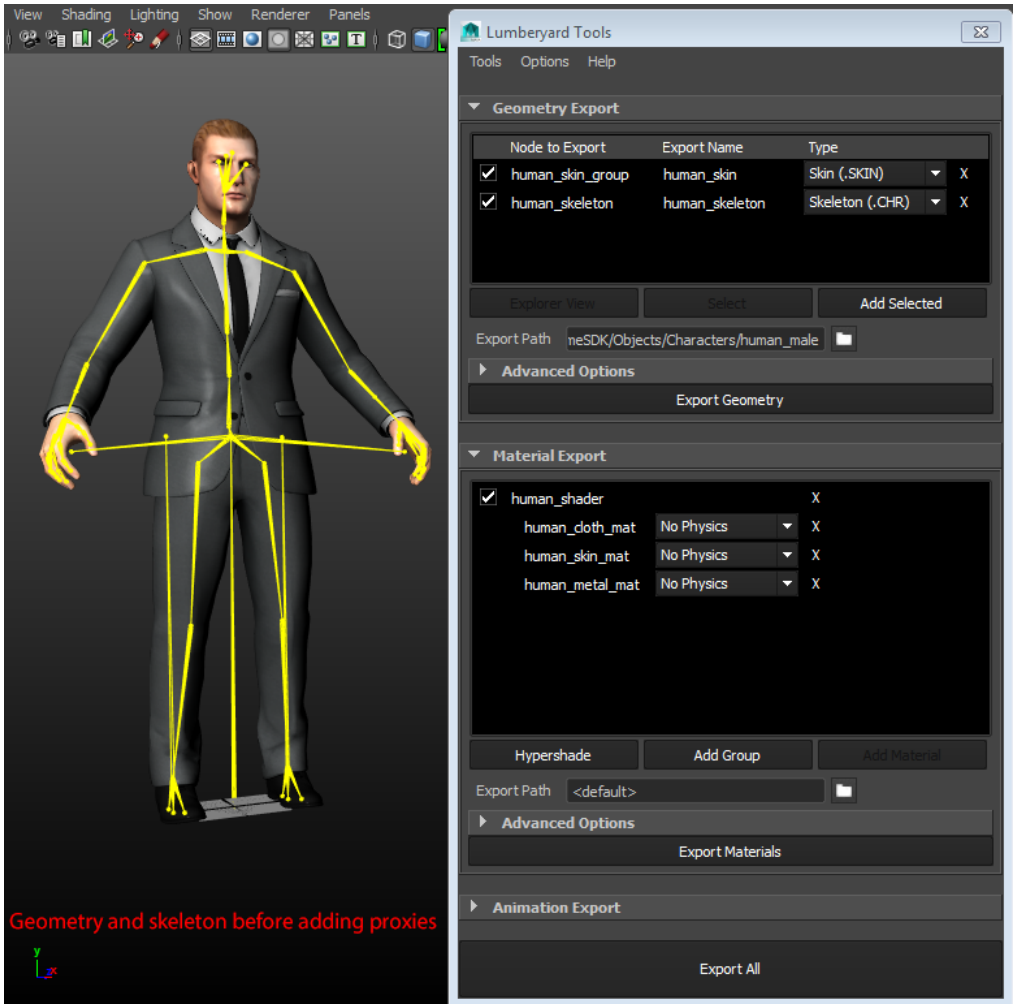
   For 3ds Max, a joint named `Bone C SpineA` would have a physParentFrame named `Bone C SpineA Phys ParentFrame`.
4. Assign each physics mesh proxy and `physParentFrame` to a parent. If using a `physParentFrame`, nest the physics mesh proxy under the `physParentFrame`, and the `physParentFrame` under the joint it belongs to. If there is only the physics mesh proxy, nest the proxy under the joint it belongs to.
5. Create and assign physics mesh materials to the appropriate physics mesh as needed. For example, a material for the left arm assigned to physics mesh proxies that are part of the left arm.
6. Add rotation limit values to skeleton joints that will be used in the ragdoll. In addition, you must add some rotation limit information to joints that do not have a physics mesh proxy but that are in the hierarchy of the ragdoll joints. For example, a clavicle needs some rotation limits even though it typically does not have a physics proxy mesh.
7. Export the skeleton's `.chr` file and the material group or multimaterial that contains the physics mesh materials.

Creating Joint Mesh Proxies

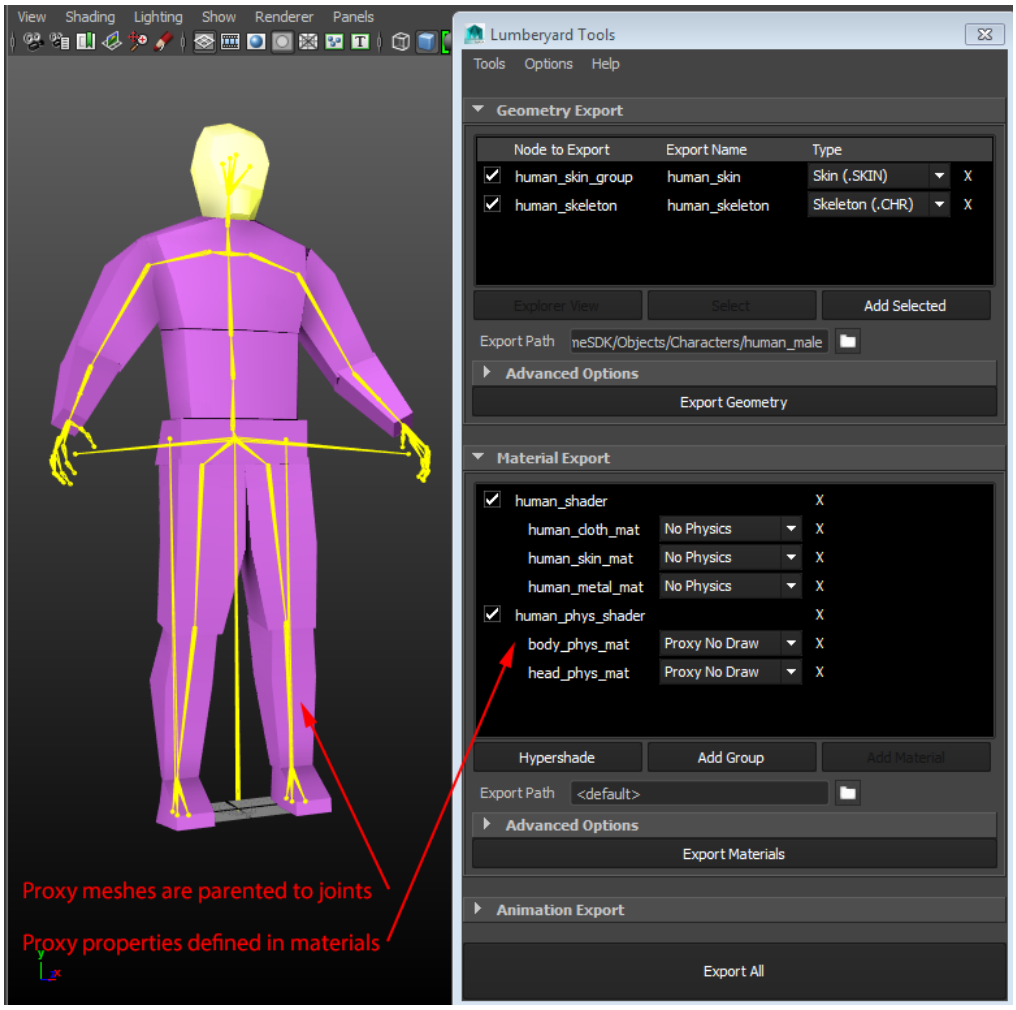
To help define body masses for character physics and collisions, you need to use a joint mesh proxy. To create a joint proxy mesh, observe the following guidelines:

- Model meshes around the geometry that needs to be detectable. Meshes with lower polygon counts perform better.

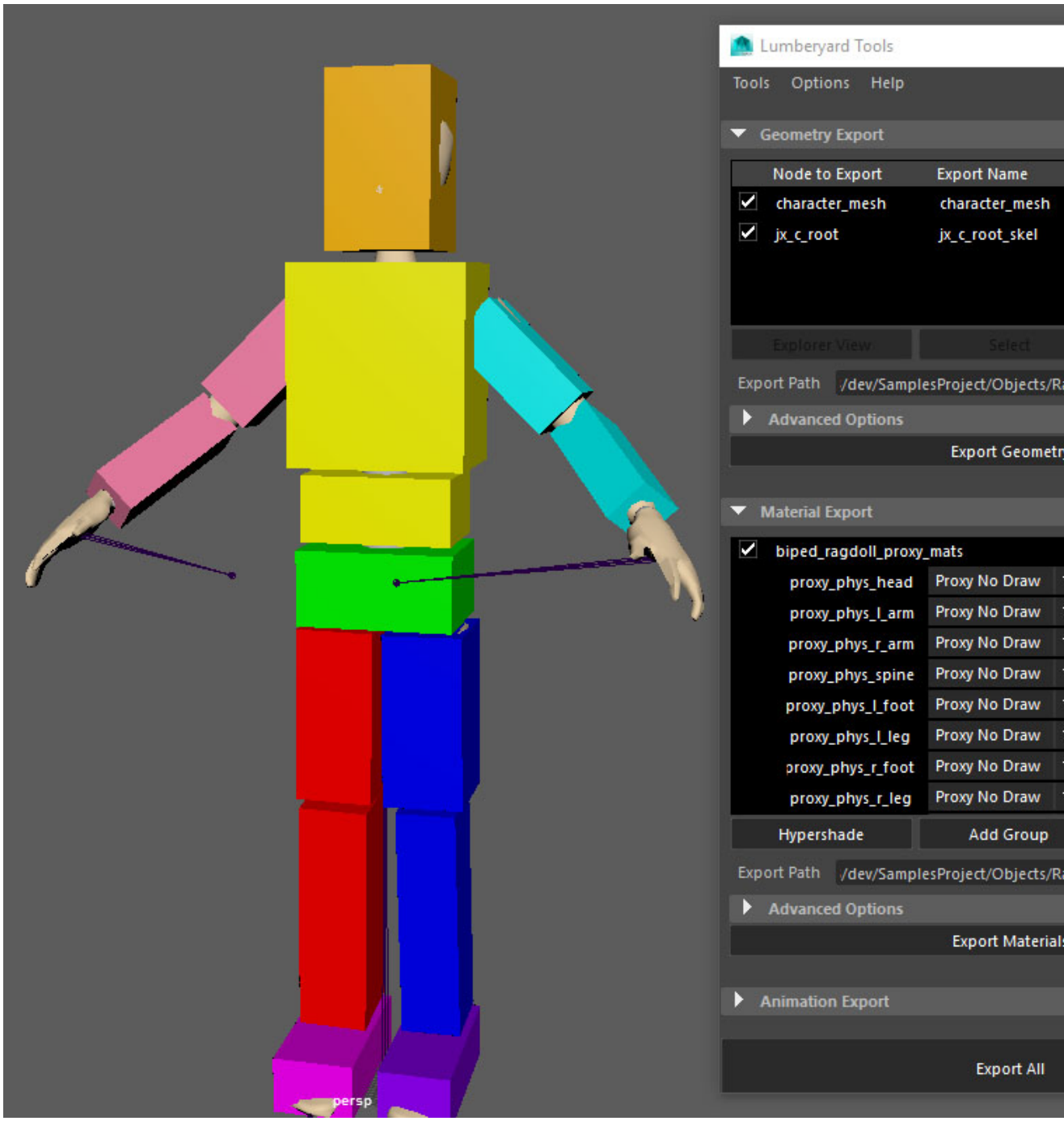
- Create meshes that use a generalized area instead of getting too granular with a physics mesh per joint to help with performance. For example, you could simplify a biped chest physics mesh proxy to cover the area of multiple spine joints instead of a proxy for each spine joint.
- Nest a proxy mesh under its corresponding joints. The proxies can then be exported with the `.chr` (skeleton) files.
- Proxy mesh naming must match the name of the joint it gets nested 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)
- To designate meshes as proxies, assign a material to them and changing the material type to **Proxy No Draw**. As a best practice, keep your ragdoll physics 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.



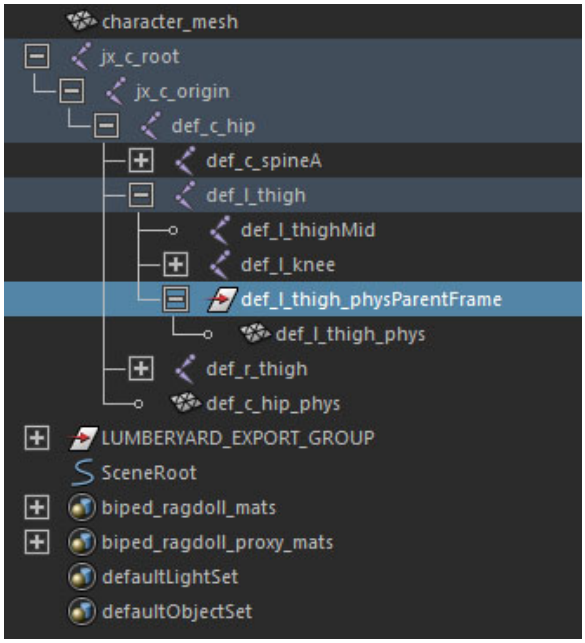
Using physParentFrames

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).



You nest the `physParentFrame` under the joint and nest the physics mesh 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. 365).

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.

The panel shows the ragdoll simulation settings that have been applied. This is true 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.

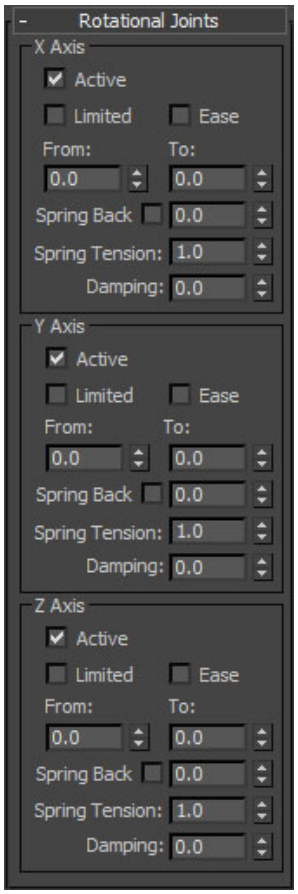
   Note
   For Maya users, there is one exception to the **Rot Limited** check boxes in the case of the pelvis/hip joint. You will want to apply some values for the **Rot Limit Min** and **Rot Limit Max** fields for the pelvis/hip joint, but keep the **Rot Limited** check boxes unchecked.

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. 370)

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** check boxes for the axes you are using and disable the **Active** check box 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. 370)

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.

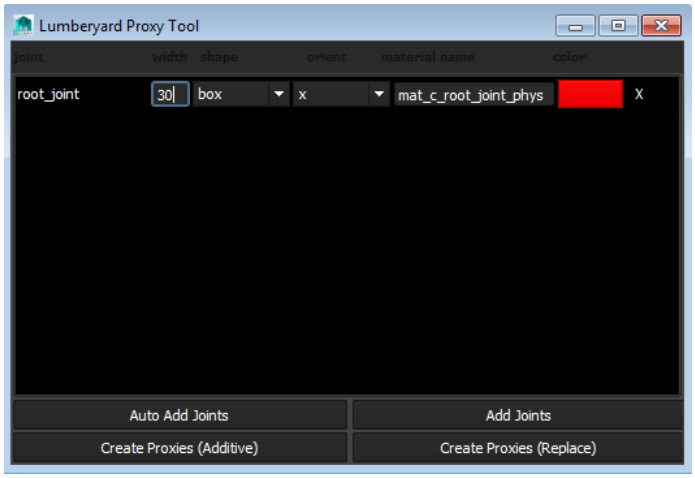
Lumberyard Proxy Tool (Experimental)

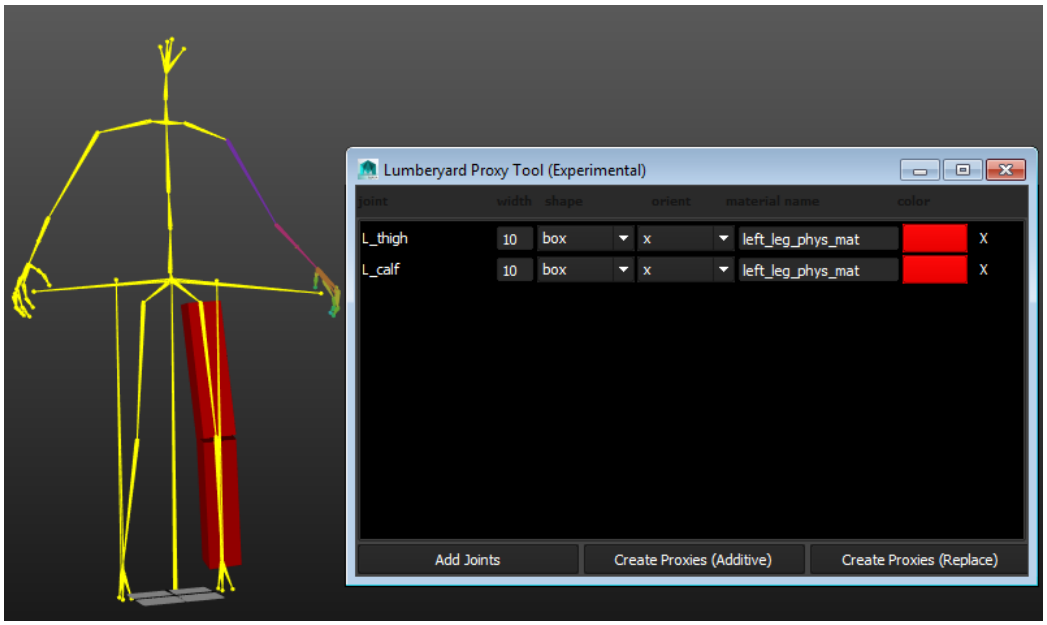
Creating individual meshes for each character body part can be time-consuming. The Lumberyard Proxy Tool automates the process of building simple joint proxy meshes and adding materials.

> **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**.
2. Use the following controls to adjust parameters as needed:
   - **Width** – Width and depth of the proxy
   - **Shape** – Shape of the proxy; options are **box**, **capsule**, and **sphere**
   - **Orient** – 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 physics mesh proxies, you assign a surface type to each submaterial. 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 whether you need to detect specific physics 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 physics 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 from your DCC with ragdoll simulation attributes on the joints and with physics mesh proxies.

To set up a Ragdoll component entity

1. In the Lumberyard Editor, right-click in your level and select **Create new component entity**.
2. Click **Tools**, **Entity Inspector** to view your component entity's settings, if the window is not open already.
3. In the **Entity Inspector** window, click **Add Component**.
4. Choose **Physics**, **Ragdoll**.

   A **Skinned Mesh** component is automatically added to the **Ragdoll** component on your entity because it is required for the ragdoll component.
5. Under the **Skinned Mesh** component, expand **Rendering**.
6. For the **Skinned asset** parameter, click the (**...**) icon 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** check box.
8. Click **AI/Physics** 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 (p. 534) for more information.

   Note
   To change the character joint rotations, you must change the simulation values on the skeleton in your DCC tool and then reexport 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 you use the **DeadBody** entity, ragdoll skeletons may collapse in unpredictable ways. To counter this, use the following settings in the **PhysParams** and **Properties** panels in **Rollup Bar**:

- `ExtraStiff` = 1 (enabled)
- `Mass` = 80
- `Stiffness` = 100

When you use the **DeadBody** entity, the ragdoll skeleton has 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 that are 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 physics 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 (that is, has the `RelinquishCharacterPhysics` value) 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 you can also 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, the animation shows falling. Otherwise, the animation is the first frame of a standup animation that corresponds 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 animation is selected from a standup animation list that best matches this pose.

You can control which type to use by `_CSkeletonPose::SetFnPAnimGroup()_methods`. At 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 requires a terminating joint to be placed precisely. All IK systems must be defined in the character's `.chrparams` file.

Lumberyard's animation system processes forward kinematics (FK) and IK tasks in the following order:

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)

When creating a game, a commonly required movement is a character aiming a weapon at a target location. This complex movement requires the weapon to point at a specific location, the hands of the character to firmly hold the weapon, and the character to look 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 with which you can create a set of poses for characters aiming in different directions. At run time, these poses are layered on top of the animation that is currently playing. This enables the character to aim toward a point in space that is requested by the game code while retaining the style present in the original authored poses as much as possible.

Characters exhibit a realistic range of motion. Note, however, that continuous, 360-degree aiming around a pivot point is not supported.

You can call aim IK from the **Flow Graph** editor, **Track View** editor, the AI system, or engine code.

Topics

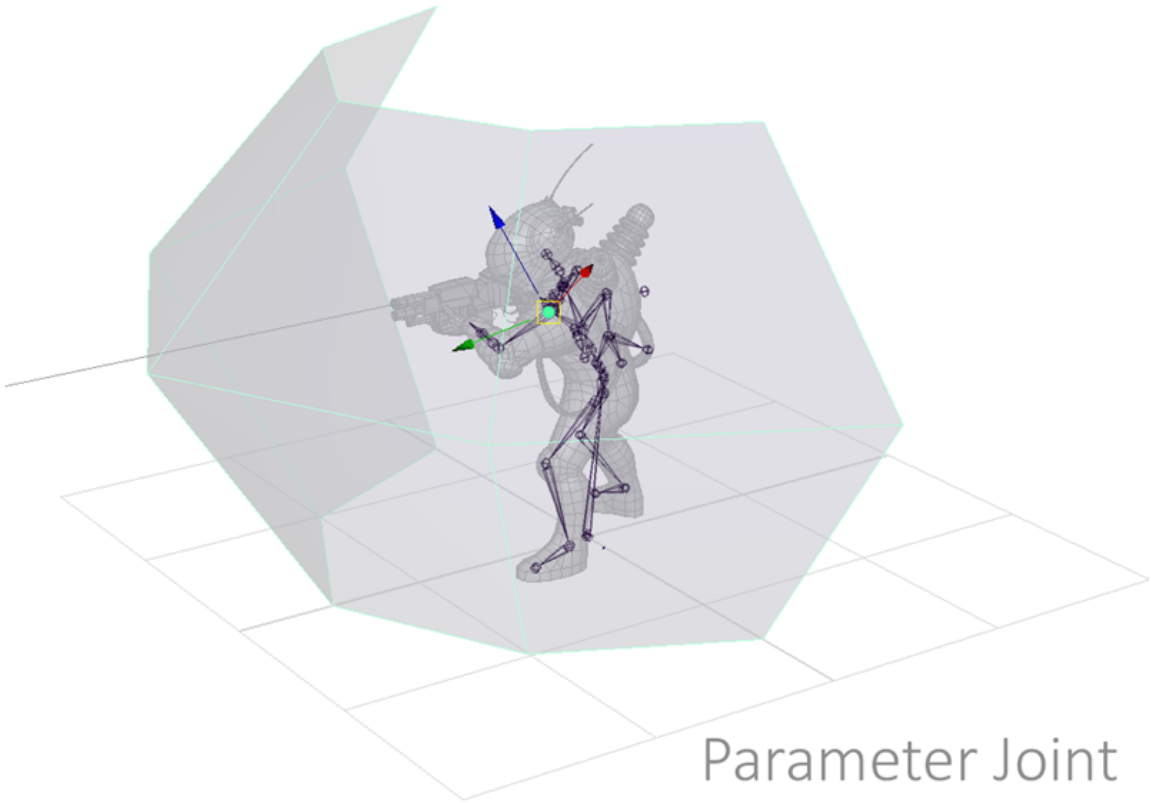
Setting up a Skeleton

The system requires certain joints to determine where a character is aiming. You can use existing joints in the skeleton or add extra joints to improve your setup.

> **Note**
> The joint names that are referenced for the attributes in the `.chrparams` file should match the names of the joints in your skeleton. The joint names don't have any specific naming requirements.

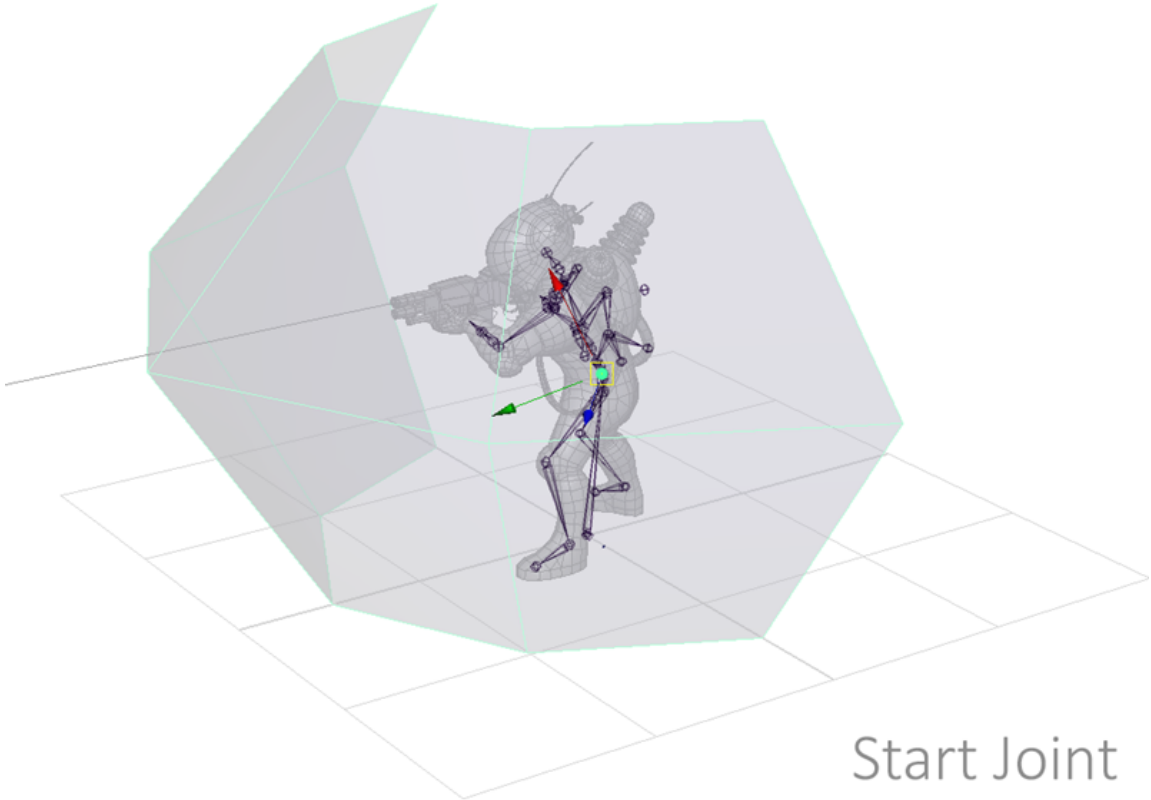
You can add the following types of joints.

- **ParameterJoint** – A joint that indicates the aim direction, with the y-axis forward.

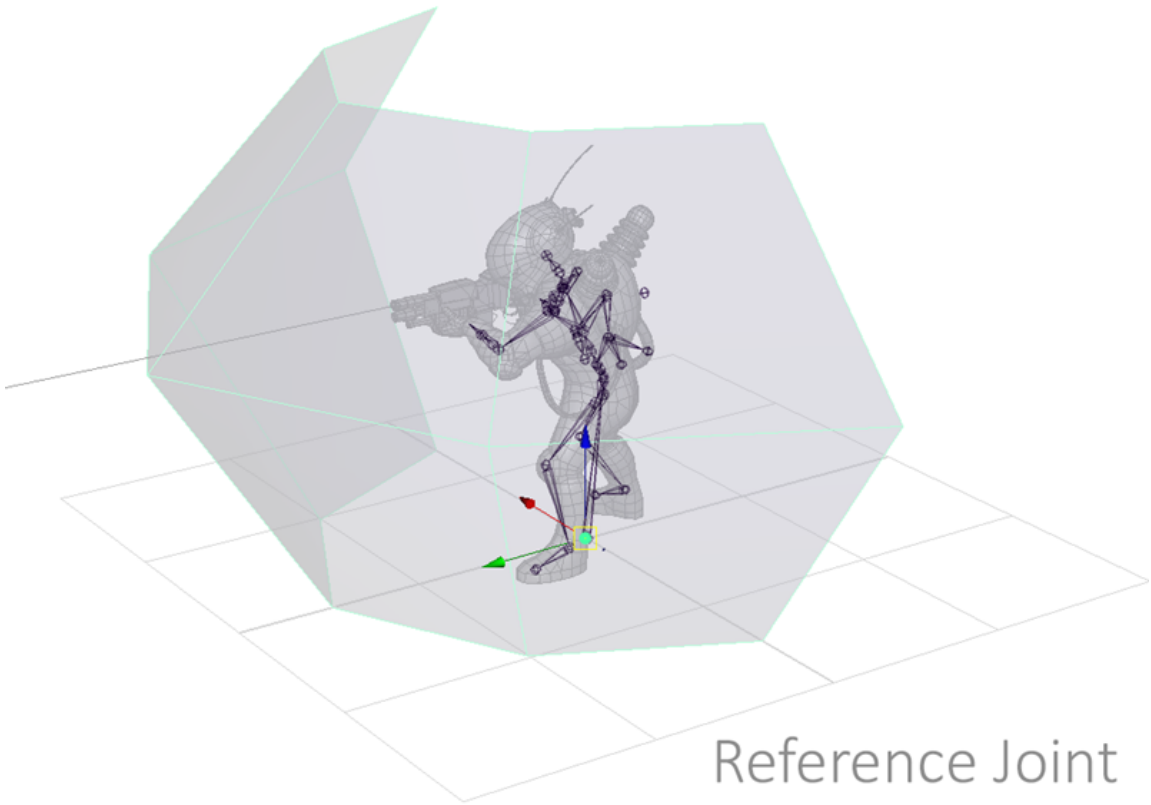


- **StartJoint** – A joint that indicates the positional center of the aim. Orientation is not important because only position information is used from this joint. For more stable results, use a less animated

joint that is less influenced by other joint animation. For example, use a joint that is connected to the root joint (the first skeleton joint, which is placed at origin 0,0,0).

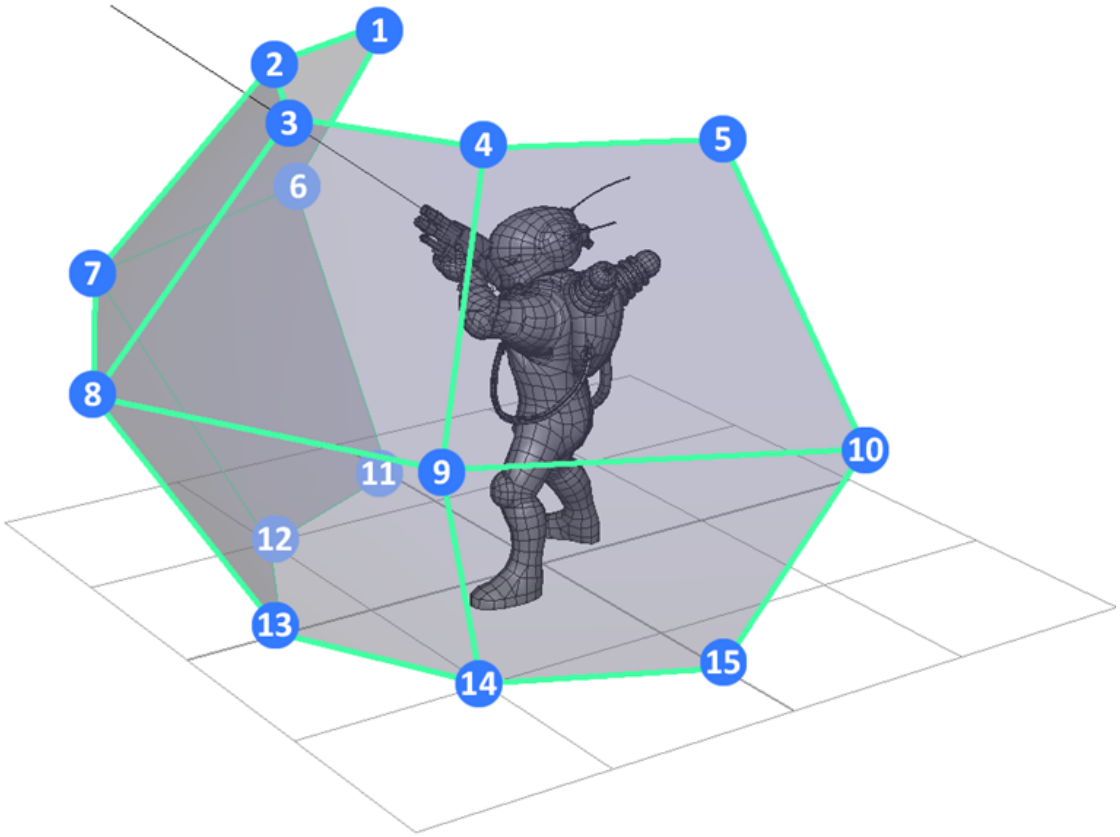


- **ReferenceJoint** – (Optional) A joint that indicates the forward direction of the character with the y-axis pointing forward. If a value is not specified, the joint at index 1 (usually the pelvis) is used. This joint is primarily used for characters in cinematics because they might have an offset on top of the root joint. In this case, don't use the root joint as the reference joint. Instead use the joint that acts as the translation or position driving joint to determine the point that is underneath the character.



Setting Up an Animation File

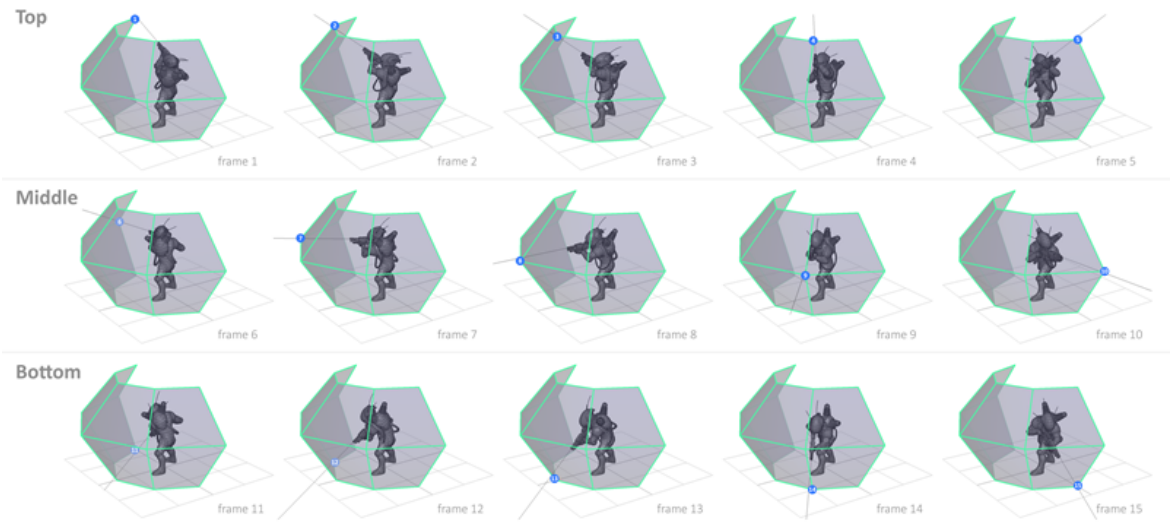
For a character aiming in several directions, the system requires a certain number of poses that can be blended together to achieve poses in any intermediate direction. For best visual results, we recommend creating 15 poses, although 9 poses might be enough in many cases. If you provide 9 poses, the system extrapolates from these poses to create 15 poses.



When you export the poses, they become a single animation file, with one pose per frame. Part of the file name should match the `AnimToken` provided in the `.chrparams` aim IK definition. For example, if the `AnimToken` parameter is set to **`AimPoses`**, the name for the animation file that contains the aim poses could be `Troop_AimPoses`. Naming this file is important because Lumberyard uses the name that is assigned in the `AnimToken` as a filter. That way Lumberyard can properly read and compile the animation data as an aim pose and not a regular animation.

When creating aim poses, you can use an underlying animation pose, such as standing idle, as a starting point. Aim poses that are created from a starting animation are applied on top of similar animations. If the underlying animation that is currently playing for a character is different enough—for example, crouching—you might need to create aim poses for that specific case to achieve better quality animations.

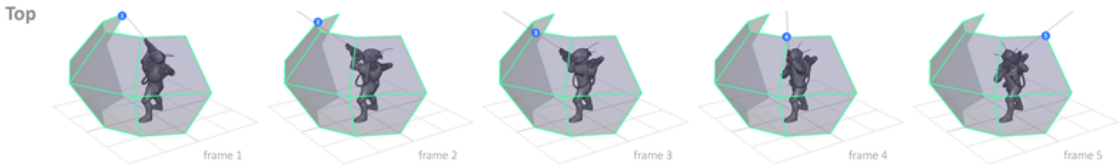
The order of the poses in the animation is also important. Reference the grid visual in the following images for the top, middle, and bottom rows.



As you create poses, keep in mind the following. This information assumes that you start on frame 1 for your aim poses animation.

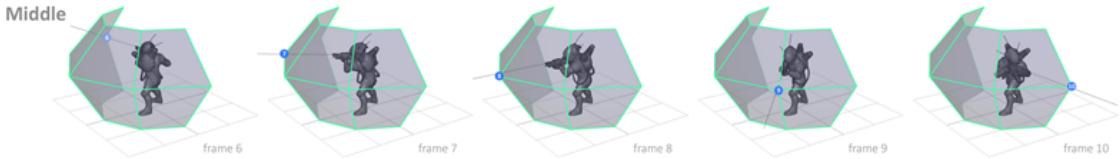
> **Note**
> Although the aim pose might appear unnatural, try to make the poses as extreme as possible. You can then set limits using the game code.

- The top row in the grid should have the character aiming upward. If you're using 15 aim poses, the top row comprises frames 1–5. If you're using 9 aim poses, the top row comprises frames 1–3.



  - The first aim pose frame in the top row (frame 1) should have the character aiming up and to the right of the character.
  - The middle aim pose frame in the top row (frame 3 of 15 or frame 2 of 9) should have the character aiming up and forward.
  - The last aim pose frame in the top row (frame 5 of 15 or frame 3 of 9) should have the character aiming up and to the left of the character.

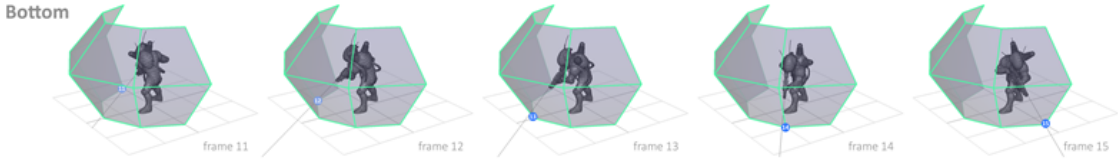
- The middle row in the grid should have the character aiming at eye level, with the weapon parallel to the ground. If you're using 15 aim poses, this row comprises frames 6–10. If you're using 9 aim poses, this row comprises frames 4–6.



  - The first aim pose frame in the middle row (frame 6 of 15 or frame 4 of 9) should have the character aiming at eye level and to the right of the character.
  - The middle aim pose frame in the middle row (frame 8 of 15 or frame 5 of 9) should have the character aiming at eye level and forward.

  - The last aim pose frame in the middle row (frame 10 of 15 or frame 6 of 9) should have the character aiming at eye level and to the left of the character.

- The bottom row in the grid should have the character aiming downward. If you're using 15 aim poses, the bottom row comprises frames 11–15. If you're using 9 aim poses, the bottom row comprises frames 7–9.



  - The first aim pose frame in the bottom row (frame 11 of 15 or frame 7 of 9) should have the character aiming down and to the right of the character.
  - The middle aim pose frame in the bottom row (frame 13 of 15 or frame 8 of 9) should have the character aiming down and forward.
  - The last aim pose frame in the bottom row (frame 15 of 15 or frame 9 of 9) should have the character aiming down and to the left of the character.

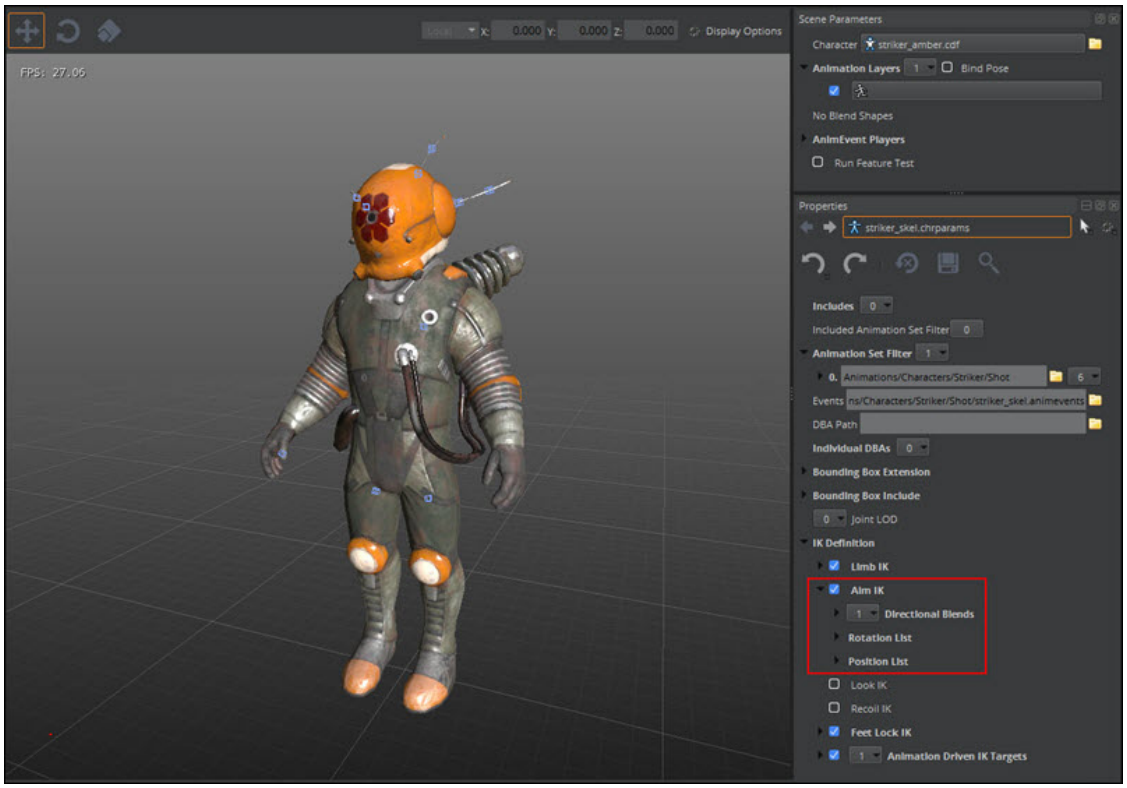
When you export your aim pose animations, ensure that you include the exact frame count for your poses. For example, if you are using frames 1–15 for your aim poses, export frames 1–15 only. If you are using frames 1–9 for a 9-aim pose setup, export frames 1–9 only. Lumberyard specifically looks for exact frame count for aim poses, which is why the order of the poses is also important.

Setting up a .Chrparams File

Use Geppetto to set up and store the aim IK definition in a character's `.chrparams` file. You can set up one aim IK definition in each `.chrparams` file.

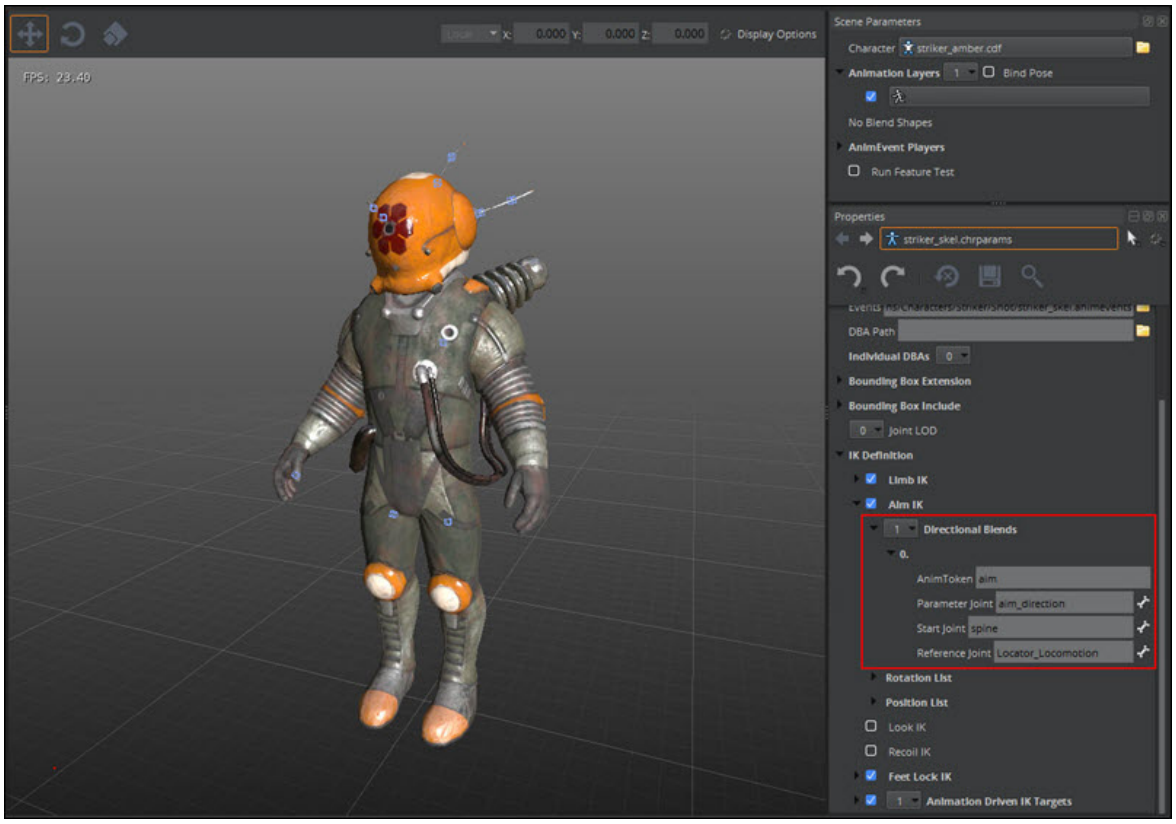
To enable the Aim IK definition in a character's .chrparams file

1. In Lumberyard Editor, choose **Tools**, **Geppetto**.
2. In **Geppetto**, in the **Assets** panel, expand **Skeletons** and navigate to the `.chrparams` file to which you want to add the aim IK definition.
3. Select the `.chrparams` file to load it in the **Properties** panel.
4. In the **Properties** panel, expand **IK Definition**.
5. Select the **Aim IK** check box.
6. Use the information in the following sections to set the **AIM IK** properties for **Directional Blends**, **Rotation List**, and **Position List**.



Directional Blends

The **Directional Blends** settings specify a combination of parameter, start, and reference joints to use for aim poses, as described in Setting up a Skeleton (p. 373). An animation is processed as an aim pose based on these settings when the `AnimToken` is found in the animation file name. For example, if a skeleton path contains the substring `aim`, the animation is considered an aim pose with `aim_direction` set as a parameter joint, `spine` as a start joint, and `Locator_Locomotion` as a reference joint. You can define these joints based on your skeleton setup, and you can specify more than one **Directional Blends**.



To set up the **Directional Blends** settings, fill in the properties for **AnimToken**, **Parameter Joint**, **Start Joint**, and **Reference Joint**.

AnimToken

A substring that must match part of the name of an animation to be processed as an aim pose. The **AnimToken** includes the current configuration for parameter, start, and reference joints.

Parameter Joint

The name of the parameter joint. For information, see Setting up a Skeleton (p. 373).

Start Joint

The name of the start joint. For information, see Setting up a Skeleton (p. 373).

Reference Joint

The name of the reference joint. For information, see Setting up a Skeleton (p. 373).

Rotation List

The **RotationList** is populated with the list of skeleton joints from the `.chr` file that is associated with the `.chrparams`. The run-time code uses the joints that are enabled in the **RotationList** to calculate and blend the orientation of joints in the aim pose.

In this list, enable the joints that are used by all aim poses and/or multiple **Directional Blends**. Verify that the list is valid for all of them. The following parameters are available for each joint in the **RotationList**.

JointName

The name of the skeleton joint. If enabled, the aim pose uses this joint's orientation data.

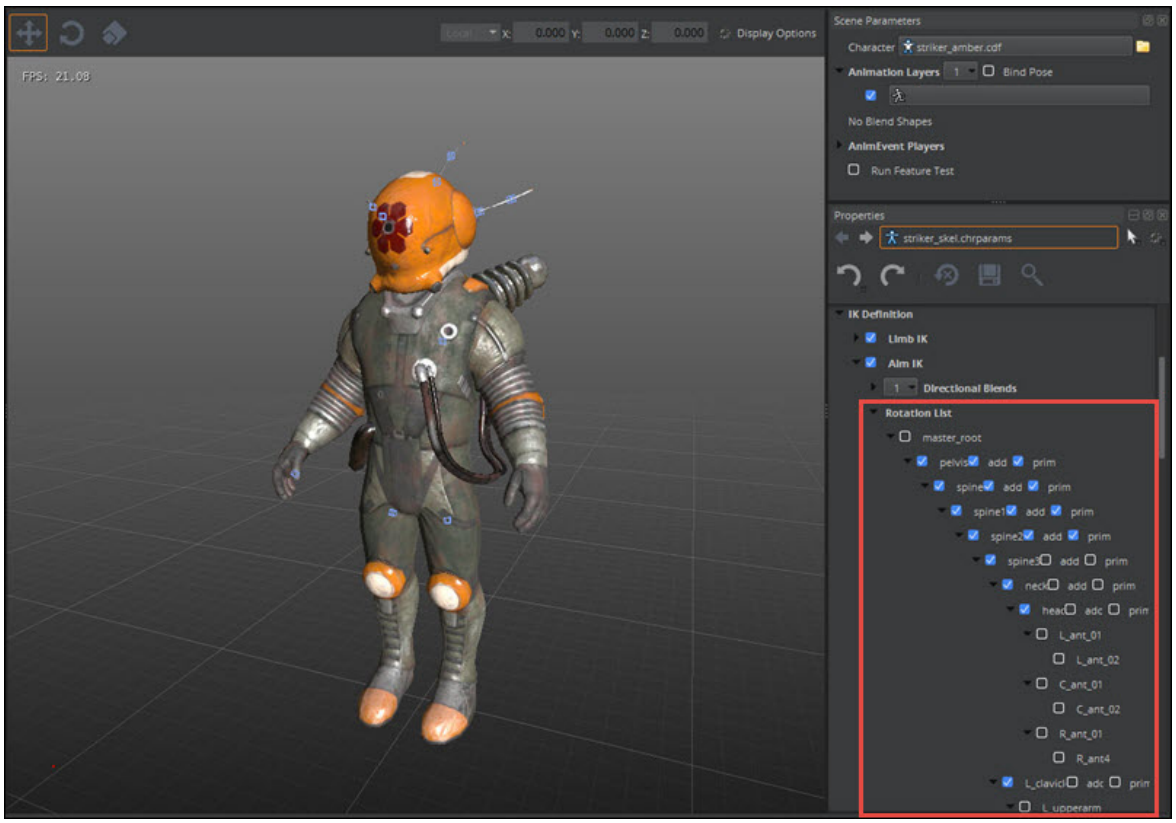
Add (Additive)

Enables additive blending for the joint's orientation. When deselected, the default is to override blending.

Prim (Primary)

Specifies whether the joint is part of the hierarchical chain from the root joint up to the parameter joint.

Primary joints that are specified in the rotation list are typically the highest joints in the hierarchy going down the chain to the parameter joint. For example, a character's arms are specifically positioned in an aim pose. The pelvis/hip joint, spine joints, and parameter joint are enabled as primary joints. But the clavicle joints through the arms are not considered primary because they are children further in the hierarchy from the parameter joint in the skeleton hierarchy.



Position List

The **PositionList** is populated with the list of skeleton joints from the `.chr` file that is associated with the `.chrparams` file. The run-time code uses the joints that are enabled in the **PositionList** to calculate and blend the position of joints in the aim pose.

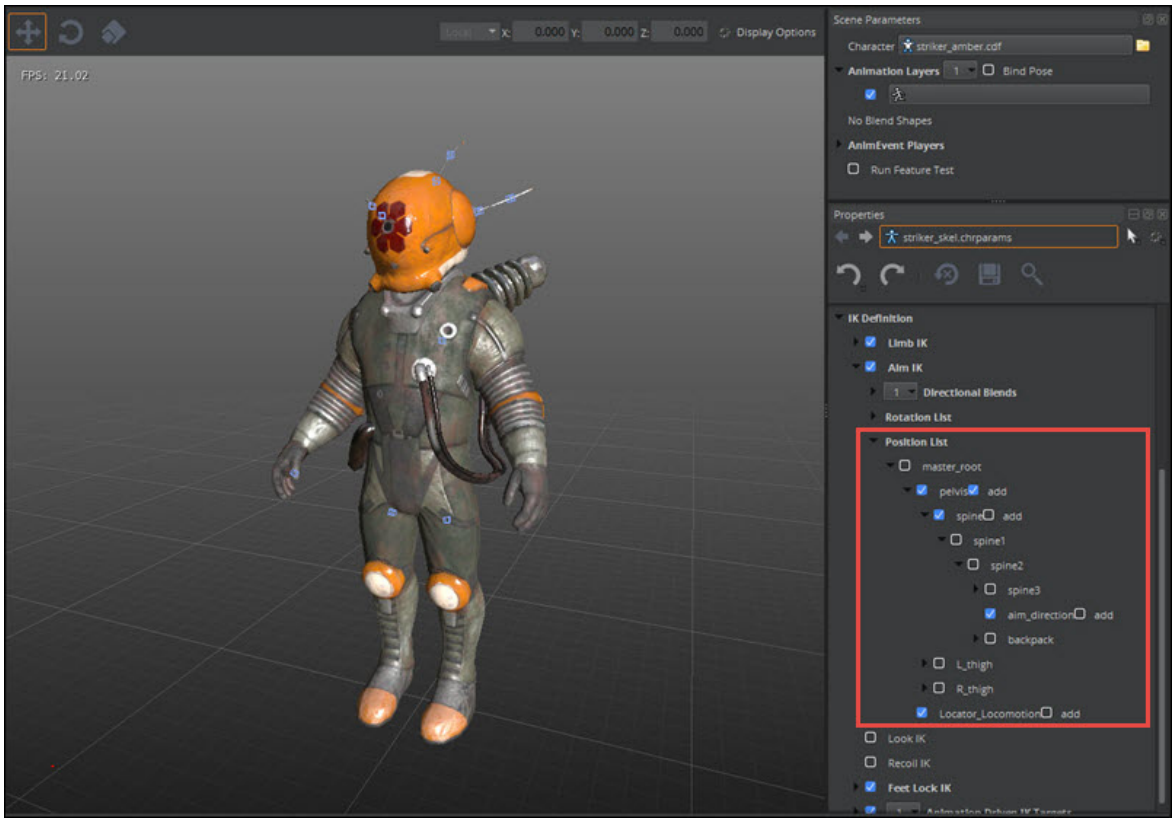
In this list, include the joints that are used by all aim poses and/or multiple **Directional Blends**. Verify that the list is valid for all of them. The following parameters are available for each joint in the **PositionList**.

JointName

The name of the skeleton joint. If enabled, the aim pose uses this joint's transform or positional data.

Add (Additive)

Enables additive blending for the joint's transform or position. When deselected, the default is to override blending.

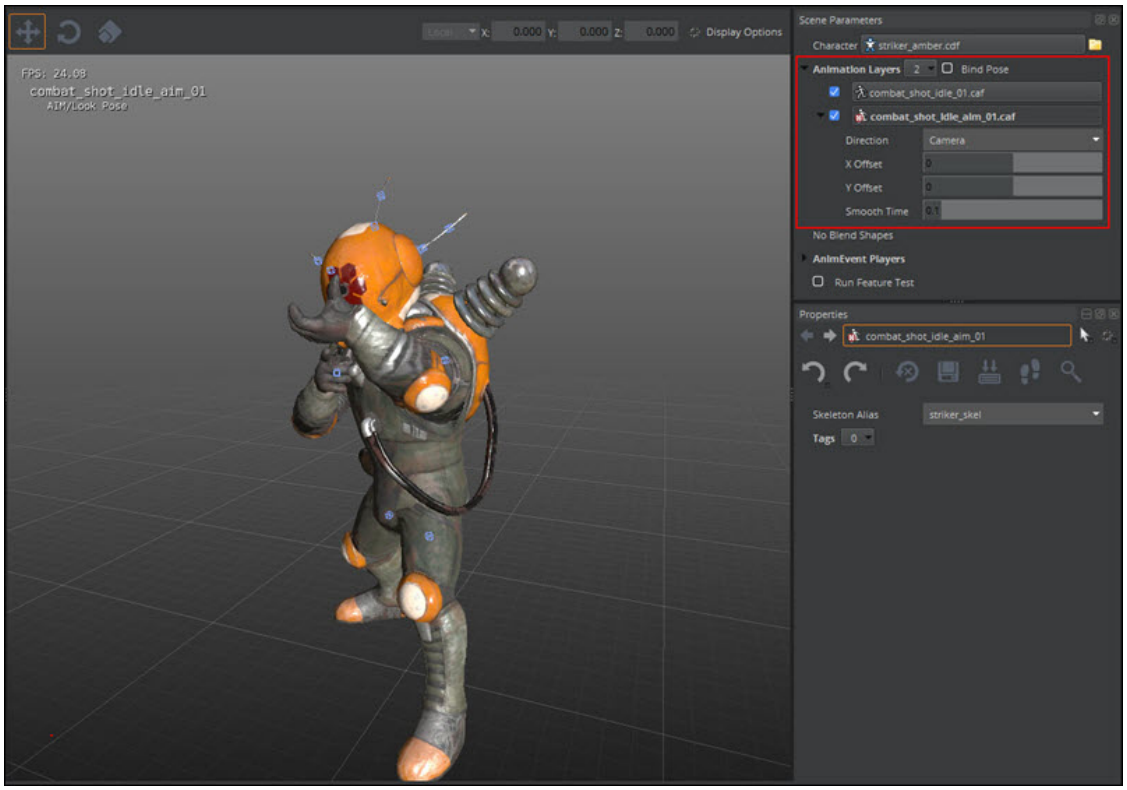


Testing and Debugging Aim IK

You can verify that aim poses are working properly by viewing them in Geppetto with animation layers.

To view aim IK through animation layers

1. In Lumberyard Editor, choose **Tools**, **Geppetto**.
2. Click **File**, **Open Character** to load your character.
3. In the **Assets** panel, under **Animations**, select an animation to assign to the base animation layer. For best results, choose an animation that the aim poses were built from.
4. In the **Scene Parameters** panel, click the menu to the right of **Animation Layers** and choose **Add**. The new animation layer becomes your active layer.
5. In the **Assets** panel, under **Animations**, select the aim pose animation to assign to the new animation layer. This layers the aim pose animation on top of the base animation.
6. In the viewport, move the camera around and observe the character aiming towards the camera.
7. Under the aim pose animation layer, you can adjust the **Direction**, **X Offset**, **Y Offset**, and **Smooth Time**.



You can adjust the following parameters for your aim IK animation layer.

Direction

Select the direction to snap the aim IK:

Camera – Provides navigation in the Geppetto viewport. The aim IK follows your camera's movements.

Forward – Points the aim IK forward. You do not have direct control of the aim IK.

Target – Exposes the **Target Position** parameter fields to set a location for the aim IK to point at.

X Offset

Applies an offset on the x-axis for the aim IK.

Y Offset

Applies an offset on the y-axis for the aim IK.

Smooth Time

Adjusts the time for the smoothness of the blend for the aim IK. The smaller the value, the more responsive the aim IK. The larger the value, the more delayed the aim IK.

Default value: 0.1.

Valid values: 0–1.

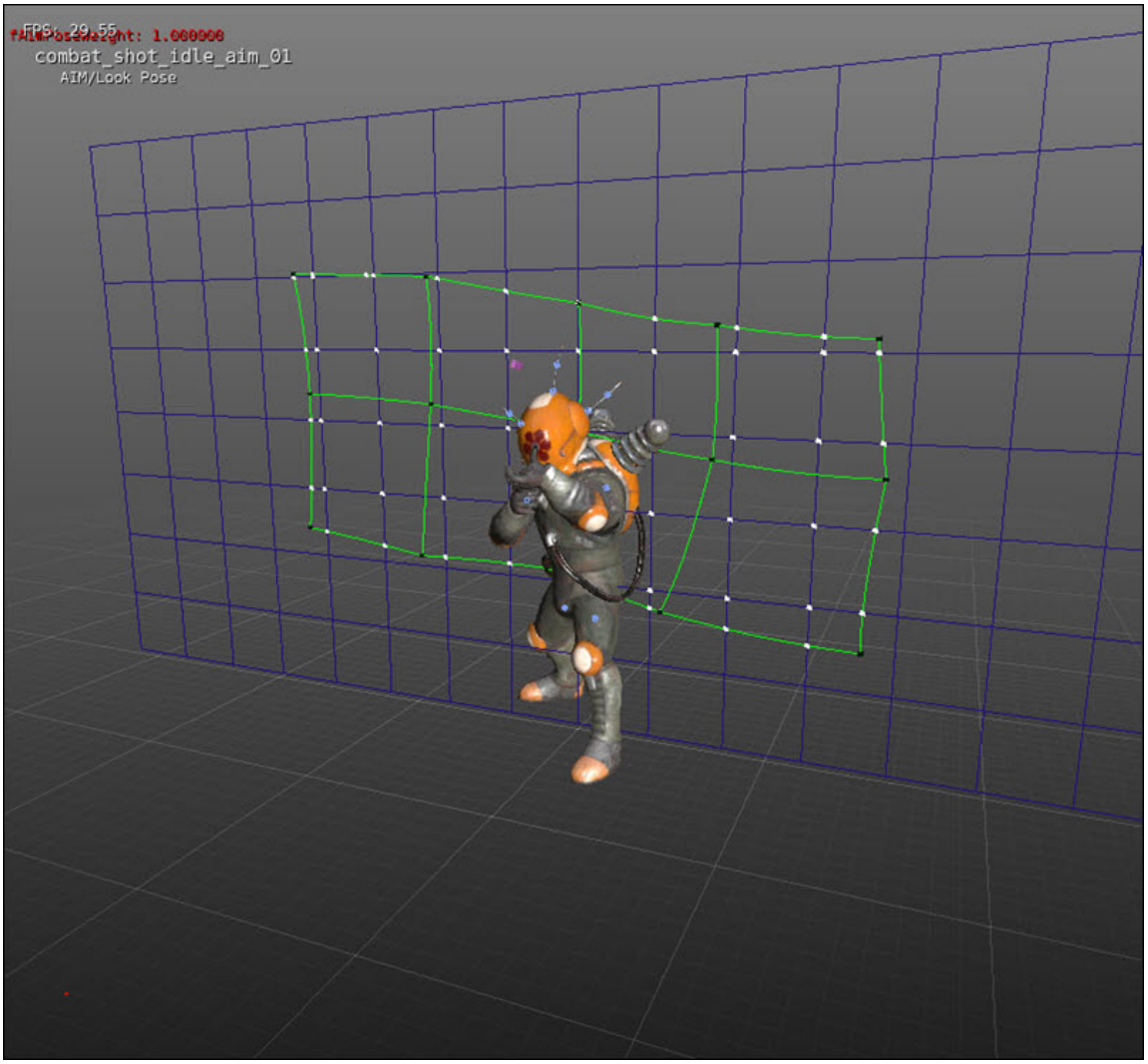
Target Position

Sets a location in xyz in the Geppetto viewport for the aim IK to point at. This parameter is only available if the **Direction** is set to **Target**.

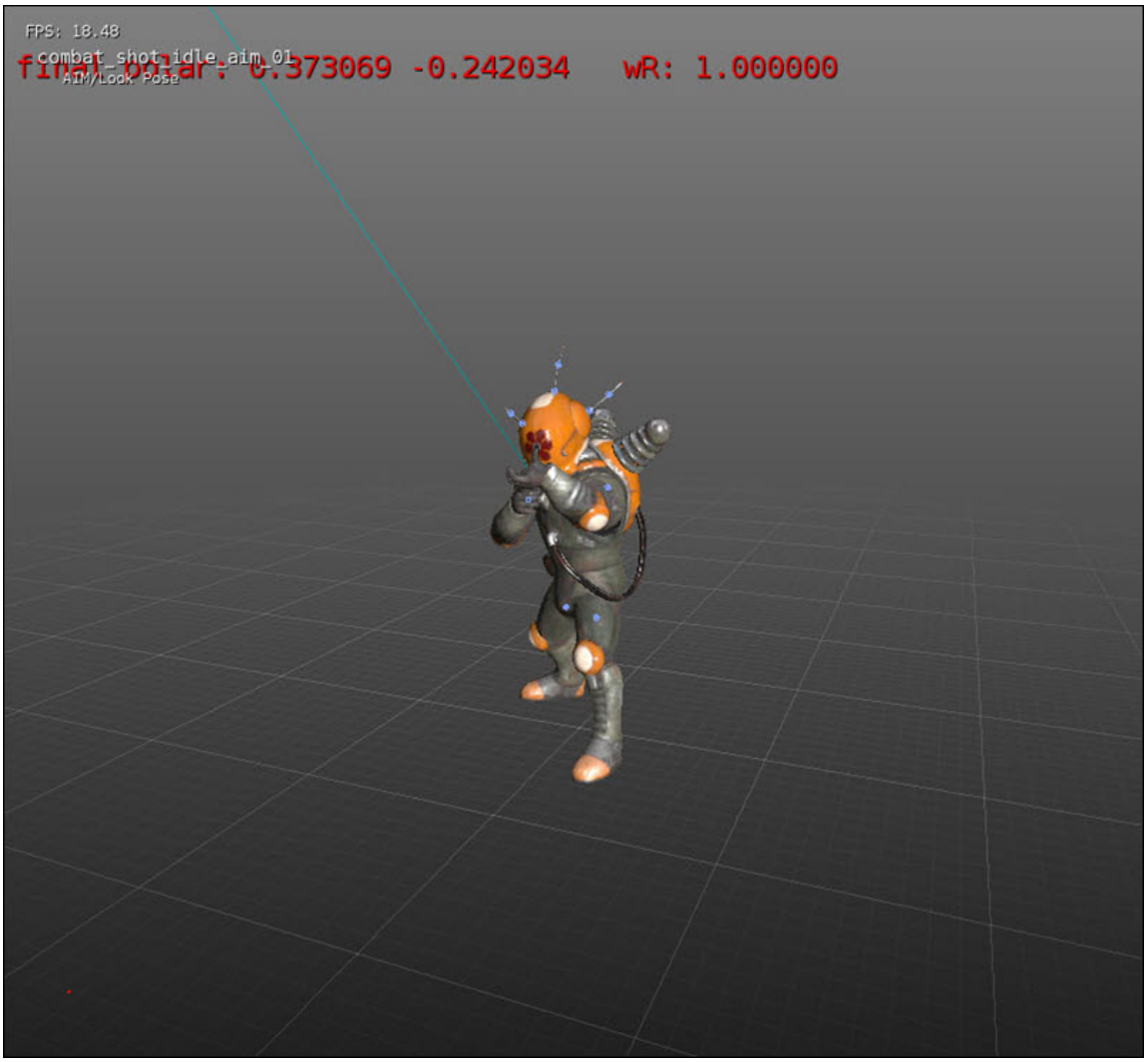
Using the Aim IK Console Variables

You can use the following console variables for debugging aim IK.

- `ca_DrawAimIKVEGrid` – To display the grid for your aim poses, set this console variable to **1**. The green rectangle shows your individual aim pose frame extremes. As you move the camera around in the Geppetto viewport, you see a pink 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 if you experience other issues, check the setup for the aim poses and the orientation of the joints in the skeleton in the `.chrparams` file.



- `ca_DrawAimPoses` – To display the aim IK's raycast and post the debug information for the coordinates on the aim IK grid, set the `ca_DrawAimPoses` console variable to **1**.



- `ca_UseAimIK` – Enables or disables aim poses on a global level for debugging.
- `es_debugAnim EntityName` – Displays the current state of a character in the animation system during gameplay debugging. This console variable contains information on all the animations that are being played, including the aim poses that are played with the base animations. The combination of the aim pose with the base animation might explain why certain aim poses are broken, for example if the combination doesn't match.

  The base layer also displays information about the blend weights and final influences of the aim IK and look IK, and whether or not the game is requesting this information.

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

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. 311).

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 automatically adjust.

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>
```

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

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.

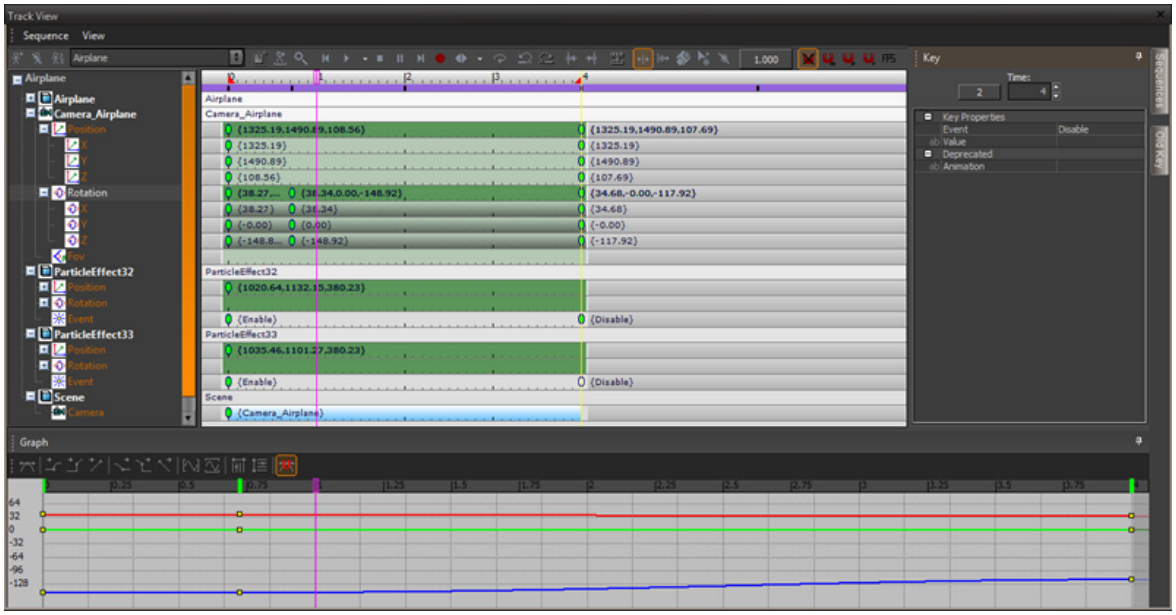
- Disable sounds and particle effects on game start.
- Use camera depth of field (focus) whenever possible as it can hide background scene imperfections. Use lower levels of detail for better performance.
- Minimize the use of simultaneous multiple effects, full-screen image, or HUD effects. Make sure 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 **Tools**, **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.



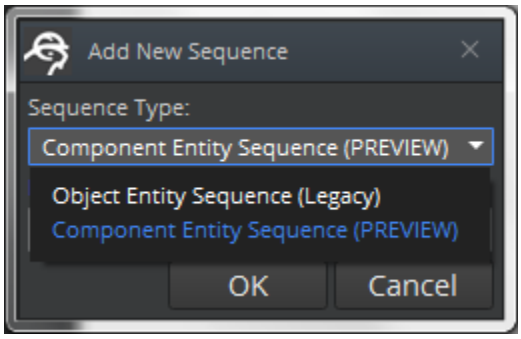
Using Cutscene Animation Curves (Curve Editor)

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.

Using Component Entity Sequences or Entity Object Sequences (Legacy)

When you create a sequence in the **Track View** editor, you must choose between the following types:

- Component Entity Sequence
- Object Entity Sequence



Object entities and component entities are incompatible and cannot exist in the same Track View sequence. Selecting your sequence type determines which nodes are allowed in the sequence. If you choose to create an object entity sequence, you can optionally create a layer to store your sequence entities.

For information about component entities, see Component Entity System (p. 437).

For information about legacy object entities, see Object and Entity System (p. 613).

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 and all other nodes are listed below the sequence.

Topics

Component Entity Nodes

In the Track View system, component entity nodes function as containers for component nodes. When you add an animation using the **Track View** editor, the animation track is applied to a component node. Therefore, component entity nodes do not directly have tracks or key properties.

When you create a component entity node, all component nodes that can be animated are nested as children. You can add tracks to any of these component nodes by right-clicking the node and selecting **Add Track**.

For more information, see Component Entity System (p. 437).

Adding or Removing Components

You can add components to or remove components from a component entity. When you add a component to a component entity in Lumberyard Editor, the component is automatically added to any component entity nodes in the **Track View** editor. When you remove a component from a component entity in Lumberyard Editor, the component and any animation data is also removed from the **Track View** editor. Use caution when removing components from component entities.

Naming Component Entities

Lumberyard uses entity IDs to identify component entities, and names to identify object entities (legacy). Because of this identification method, object entity (legacy) names must be unique. Until object entities (legacy) are deprecated in a future release, component entity names in the Track View system must also be unique.

If you attempt to add a component entity to a sequence and the name is not unique, you will see the following error message:

[Warning] Track View: '<`ENTITY_NAME`>' was already added to 'sequence', skipping...

To resolve this issue, provide a unique name for the component entity.

Component Nodes

Component nodes that can be animated are nested as children under the associated component entity node. You can add animation tracks to any of these component nodes by right-clicking the node and selecting **Add Track**.

Not all components can be animated in the **Track View** editor. For more information, see Exposing Custom Components to Track View for Animation in the *Amazon Lumberyard Developer Guide*.

Animation Components

Lumberyard provides the following components that can be animated:

- Transform
- Camera (p. 467)
- Light (Point Light (p. 529), Area Light (p. 449), Projector Light (p. 532), Environment Probe (p. 499))
- Skinned Mesh (p. 558)
- Mesh (p. 521)
- Simple Animation (p. 550)

For information about component-specific properties, see the Component Reference (p. 446).

Record Mode

When you press the record button on the playback toolbar in the **Track View** editor, the following occurs:

- Any changes to the tracks in the sequence are detected.
- A keyframe is set at the current time with the track's current value.

You can then use the property inspector panel or the translation and rotation gizmos to set the property values and keyframes in the **Track View** editor.

The record button works with all properties in the component entity sequences.

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	• **Comment** – Text string • **Duration** – Length of time the node is active • **Size** – Font size • **Color** – Font color • **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**.

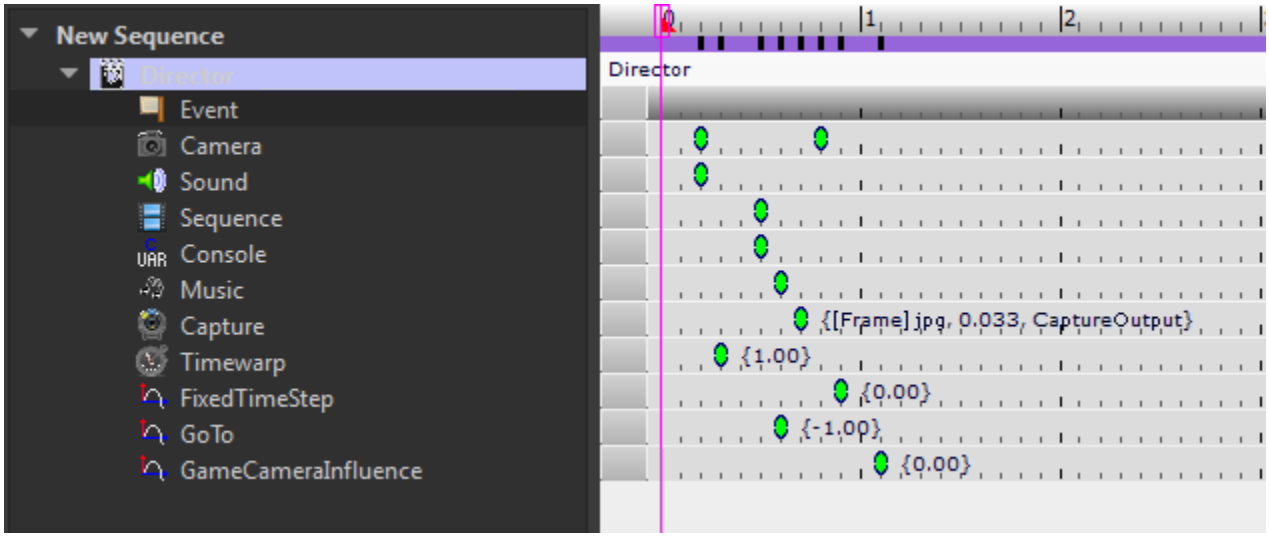
To animate a console variable

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

The **Director (Scene)** node includes a camera track that specifies the active camera for a sequence. You can add sequence-specific nodes (for example, **Depth of Field** or **Comment**) under the **Director** node in order to override the same nodes that were set at the sequence level.

You can add multiple **Director** nodes to a scene, but only one node will be active. Change the active **Director** node by right-clicking the node and selecting **Set as Active Director**. When you deactivate a **Director** node, all child node animations are also deactivated. This is useful for enabling and disabling animation for specific objects within the same shot.



To add a Director node in Track View

1. In Lumberyard Editor, click **Tools**, **Track View**.
2. In the **Track View** editor, click **Add Sequence** (first button in the toolbar).
3. In the **Add New Sequence** dialog box, type a name for your sequence and click **OK**.
4. Right-click your sequence and select **Add Director (Scene) Node**. You can add the following tracks to the **Director** node and set key properties.

Director Node Tracks and Key Properties

Track	Key Property	Description
Create Folder		Creates folder that you can use to organize director tracks.

Track	Key Property	Description
Camera	Camera	Specifies the sequence camera. You can optionally use the `mov_overrideCam` console variable to override the camera. Set the console variable to a camera name for the object entity sequences (legacy), or to an entity ID (string) of a **Camera** component entity for component entity sequences.
	Blend time	Specifies the blend time, in seconds, between sequential cameras in the track.
Capture	Duration	Sets the capture duration in seconds.
	Time Step	Forces a fixed frame rate, in seconds, by using a specified time step. Time step = 1/number of frames. 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	Specifies a prefix to apply to the image file names. For best results, use the same prefix as the sequence for clarity.
	Output Folder	Specifies the directory where the image files are stored.
	Buffer(s) to capture	Specifies the image capture format. Choose from the following options: • **Color** – RGB pixel information • **Color+Alpha** – RGB pixel data and Alpha channel data **Note** To capture Alpha channel data, you must set **Output Format** to **tga**.
Console	Command	Executes the specified console command.
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 fixed time step value of .0333 results in a game frame rate of 30 fps.
GoTo		Jumps forward or backward in a sequence. Use this track for key framing time shifts and to loop part of a sequence. This key automatically applies animation blending on animations that are currently playing in the sequence.
Music	Mood (T) or Volume (F)	Specifies if the key should change mood or volume according to the corresponding mood or time property of that key.
	Mood (if Mood)	Changes the mood.

Track	Key Property	Description
	Time (if Volume)	Changes the time.
Sequence	Sequence	Specifies the sequence to play at the specified keyframe.
	Override Start/End Times	Overrides 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 that scales the movie playback speed by the **Timewarp** value. A value of 1.0 represents normal speed; smaller values slow down time and larger values speed up time. If you set **Timewarp** to zero, playback will completely stop in your movie. **Timewarp** applies only to visuals; sounds are not slowed down.

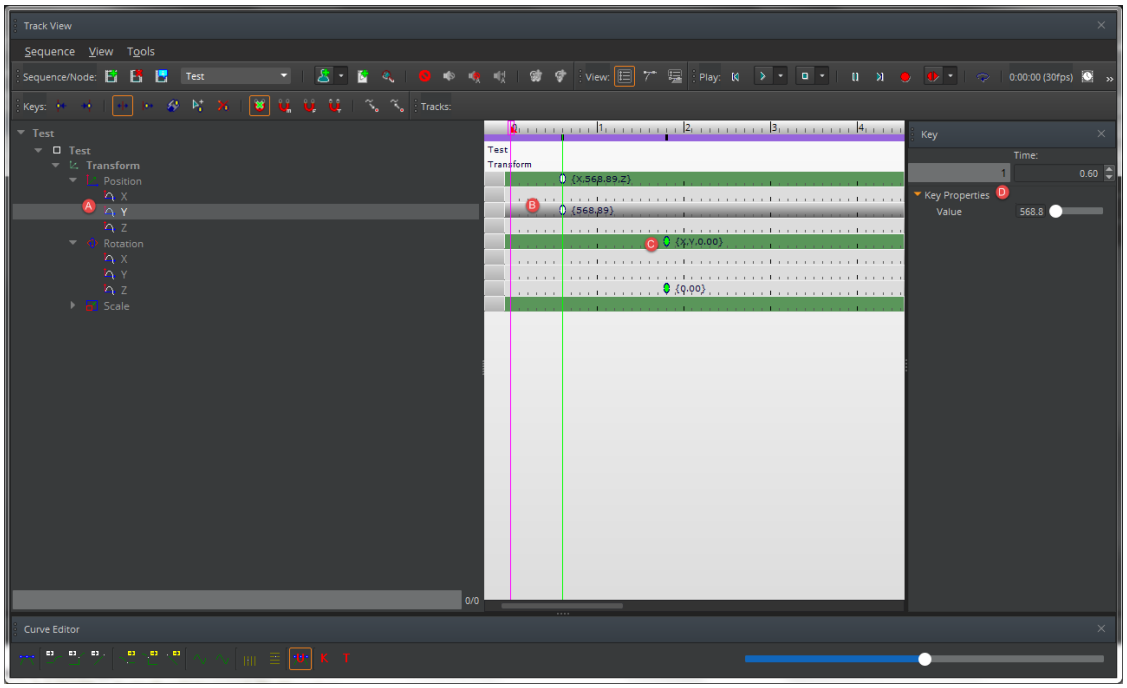
Entity Nodes (Legacy)

This topic describes how to use legacy entity nodes in the **Track View** editor. These nodes will be deprecated and replaced with component entity nodes in a future release. For more information, see Component Entity Nodes (p. 395).

Lumberyard Editor uses entity nodes to communicate with the Track View system. You can create entity nodes using the following steps.

To add a legacy entity node in the Track View editor

1. In Lumberyard Editor, select a legacy entity in the viewport and then click **Tools**, **Track View**.
2. In the **Track View** editor, create an object entity sequence (legacy) if one does not yet exist. To do so, click the **Add New Sequence** icon next to **Sequence/Node** in the toolbar. In the **Add New Sequence** dialog box, choose **Object Entity Sequence (Legacy)** from the drop-down list.
3. Right-click the sequence (top node) or **Director** node, and choose **Add Selected Entity(s)**.
4. In the left pane, under the track name, select to highlight each animation track. To add AnimObject tracks, see Animated Character Tracks in Cutscenes (p. 423).



Entity Node Tracks and Key Properties

Track	Key Property	Description
Event	Event	Provides a list of all possible events that are 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	Specifies the entity to look at.
	Target Smooth Time	Specifies the transition time, in seconds, over which to smooth the look rotational change.
Mannequin	Mannequin fragment	Specifies the mannequin fragment to play at the key frame.
	Fragment tags	Specifies the fragment tags to use with the fragment.
	Priority	Sets the fragment's priority. A higher number indicates a higher priority.
Noise		Adds noise to the position and rotation of the entity if the Position and Rotation tracks, respectively, have keys.
Physicalize		Enables and disables physics simulation on an entity.

Track	Key Property	Description
PhysicsDriven		Sets the position and rotation to be driven by physics for non-static physics entities.
Position		Sets the XYZ position of the entity.
Procedural Eyes		This track is deprecated and will be removed in a future version of Lumberyard.
Rotation		Sets the XYZ Euler rotation angles of the entity.
Scale		Sets the XYZ scale factors.
Sound	StartTrigger	Sets the audio control for starting sound. For more information, see Using the Audio Controls Editor (p. 186).
	StopTrigger	Sets the audio control for stopping sound. For more information, see Using the Audio Controls Editor (p. 186).
	Duration	Specifies the length of time, in seconds, to play the sound.
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

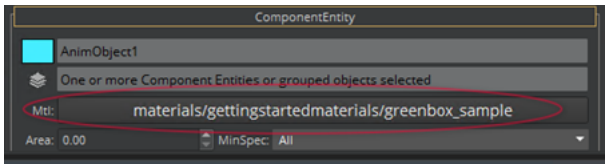
Material nodes help you animate a number of commonly used material properties that you would normally set in the **Material Editor**. You can add **Material** nodes through a sequence or from the director node context menu.

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 select the entity with the material that you want to animate. In the **Entity** pane of the **Rollup Bar**, click **Mtl** to open the **Material Editor** with the material selected.

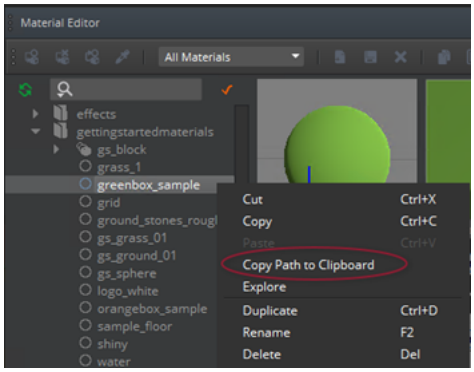
To add a Material node in the Track View editor

1. In Lumberyard Editor, select the entity with the material that you want to animate.
2. In the **Rollup Bar**, under **ComponentEntity**, click **Mtl** open the **Material Editor**.



   Note
   The **Mtl** button shows the full path of the material. If you don't see this option, make sure that you are in the **Entity** pane of the **Rollup Bar**.

3. In the **Material Editor**, right-click the selected material and choose **Copy Path 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 **Tools**, **Track View**. In the **Track View** editor, use the sequence selector on the toolbar to choose the sequence or director that you want to contain the animation.

You can also click the **Add Sequence** icon to create a new one.

5. Right-click in the node tree, and choose **Add Material Node**. In the **Material Name** dialog box, do the following:

   a. Press **Ctrl+V** to paste the full path to the material that you copied in step 3.

   b. Delete the path up to and including your project directory. For example, for the following path:

      ```
      C:\pathToGame\dev\SamplesProject\materials\GettingStartedMaterials
      \greenbox_sample.mtl
      ```

      Delete the path up to and including `SamplesProject\`. The resulting material node name is `materials\GettingStartedMaterials\greenbox_sample.mtl`.

   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, select the material that you want to animate.

   The material node should appear in colorless text. If the text is red, the **Track View** editor can't find a match for the material node name.

6. Right-click the node that you added in the previous step. Choose **Add Track, Diffuse**. Add two key frames with different colors, then scrub. Your material's diffuse color should be animated. For more information, see Using the Track Editor (p. 1622).

Material Node Tracks

Track	Key Property	Description
Diffuse		Type RGB values to specify the base color of a material.
Emission		Type RGB values to enable 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 greater than 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

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.

Property	Description
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	Selects either **FadeIn** or **FadeOut** values.
ChangeType	For this transition type select from **Cubic Square**, **Linear**, **Sinus Curve**, **Square**, or **Square Root**.
Color	Specifies the RGB value used for fading.
Duration	Specifies how long it takes to fade in or out the screen.
Texture	Specifies a texture file to use as a screen overlay. An alpha texture is commonly used for effects like dirt or blood. The texture is multiplied by the color value to allow you to animate the brightness during the fade.
Use Current Color	Ignores the **Color** property and uses 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.

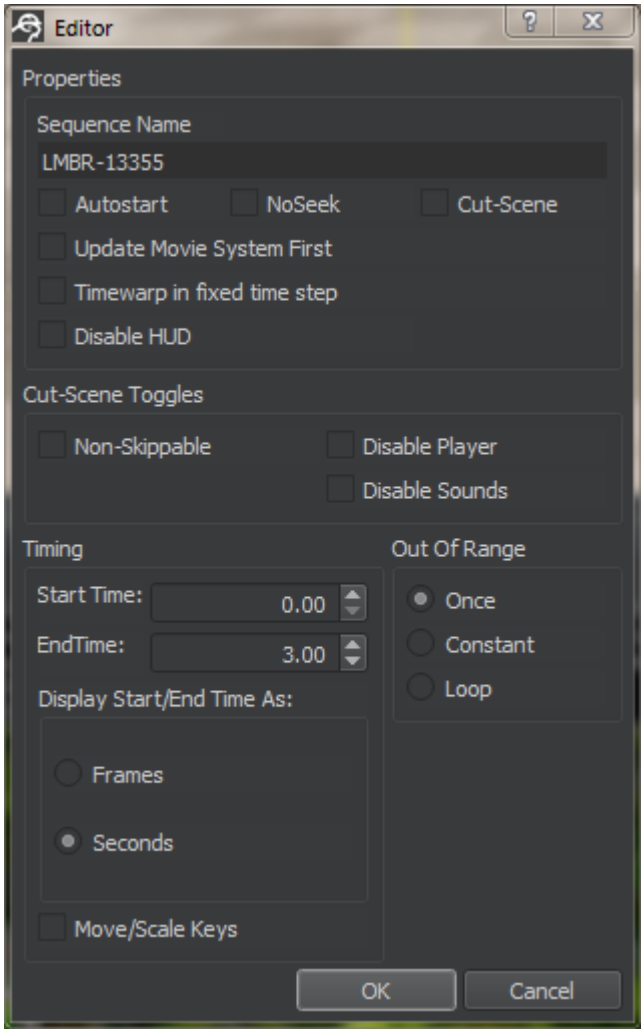
Topics

Setting Sequence Properties

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 **Edit Sequence** (third button in the **Sequence/Node** toolbar).
2. In the **Sequence Properties** dialog box, set the 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.	

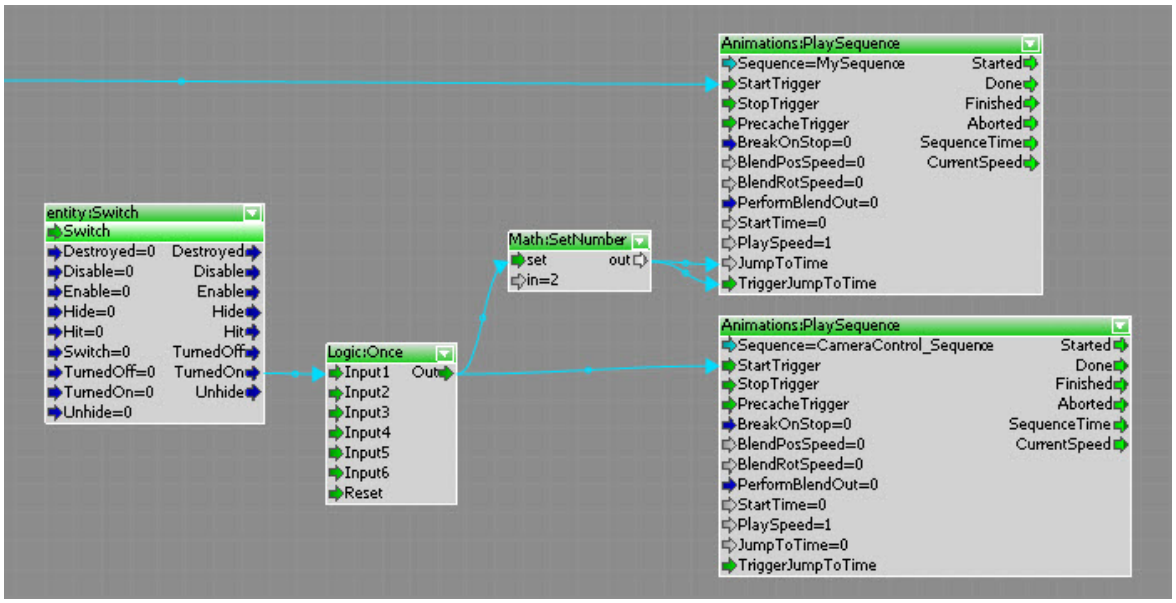
Property	Description	When to use
CutScene	Enables various scene toggles. When selected, the following options are available: • **Disable Player** – Disables the player (required for all camera-controlled scenes). • **Non-Skippable** – Sets the `IAnimSequence::eSeqFlags_NoAbort` bit flag to 1. This flag provides a UI to specify whether the sequence can be skipped. Your game code must implement the input and code required to skip a sequence. You can use the `IAnimSequence::GetFlags()` method in C++ to retrieve the set if bit flags for the sequence. • **Disable Sound** – Disables all sounds that are not in the scene.	This property is required for all scenes that are camera-controlled.
Update Movie System First	Reverses the update order. Typically, the movie system updates before the entity system updates.	Fix bone-attached entities that lag behind the parent movement. This problem typically occurs if the parent locator position is animated in Track View.
Timing	Sets the start and end times of the sequence.	
Display Start/End Time As:	Displays the start and end times in frames or seconds. This is a display-only option; the times are always stored in seconds.	
Move/Scale Keys	Scales animation curves over the timeline when the start or end times are changed.	Use when you want to lengthen or shorten a sequence, and either slow down or speed up the animations to fill the same relative percentage of the sequence timeline.
Out of Range	Changes the movie time behavior when it passes the end of the sequence: • **Once** – The movie time continues past the end of the sequence. • **Constant** – The movie time is held at the end of the sequence. • **Loop** – The movie time loops back to the beginning of the sequence when it reaches the end.	

Changing Scene Toggles Mid-Sequence

You can change the **Cut-Scene Toggles** properties 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 of 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).

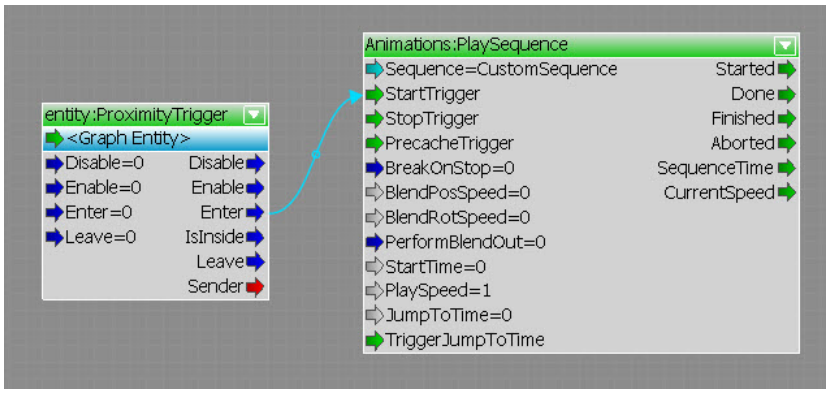


Playing a Sequence

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.

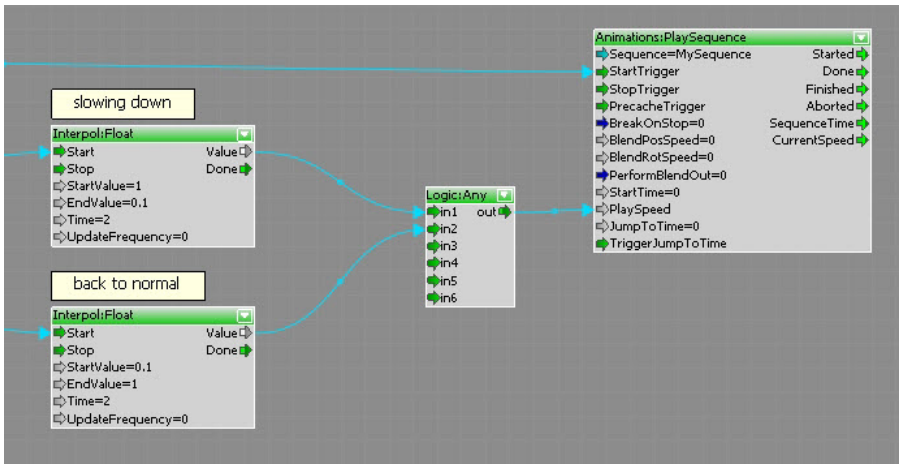
Specifically, the **entity:ProximityTrigger** node output is connected to the **StartTrigger** input of the **Animations:PlaySequence** node. When a player enters the trigger in the game, the sequence starts.



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

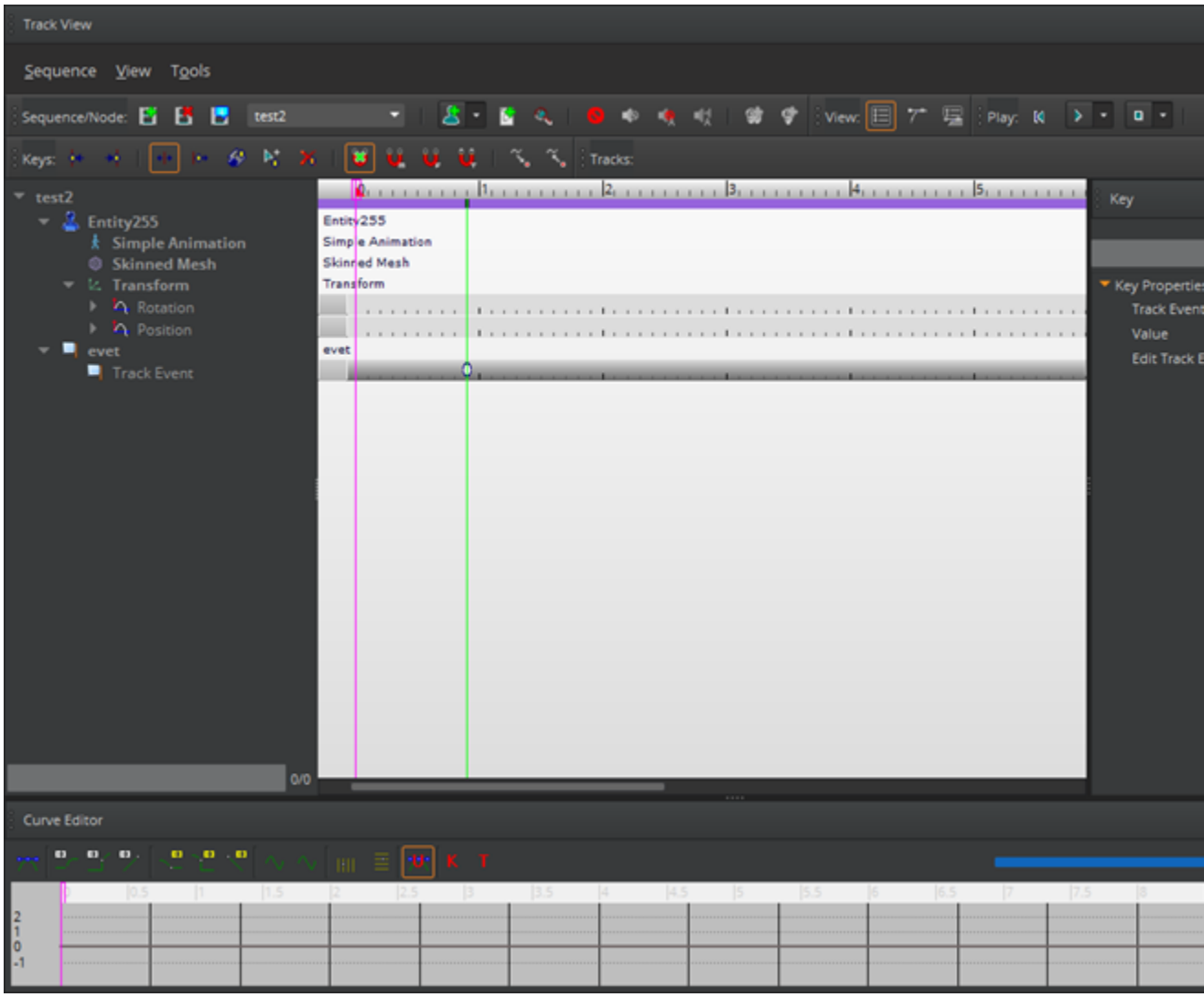
A track event is a trigger that allows you to integrate Flow Graph logic with a Track View scene. When a track event is triggered from the scene, its corresponding output in a Flow Graph node is activated. A scene can contain a number of track events that are grouped under a Track Event Node. Each track 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.

Use the following procedure to create your link between track view and flow graph after you have created an event and assigned it to the key, as described in the previous section.

To place the TrackEvent node in flow graph and link it to the event node

1. Open Flow Graph Editor.

2. Open a new or existing flow graph. Right-click in the flow graph and click **Add Track Event Node** in the context menu.
3. Double-click the **Sequence=** box. Click **..** to open the **Select Sequence** dialog.
4. Select the track view sequence that your event is coming from.

   Your **TrackEvent** node's output port is now labeled with your track view event name.

   The output port activates when the sequence key is reached. You could then wire this port to a node that will be activated at that point in the sequence.

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

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.

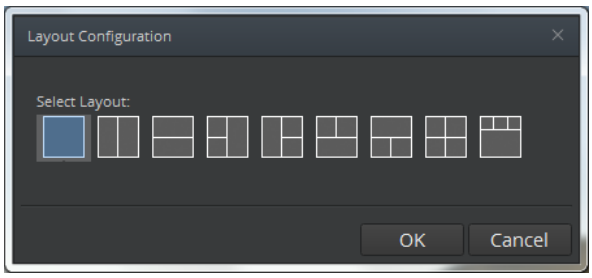
Here are some guidelines and best practices when setting up camera focus:

- 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
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.

To use the default game camera in a sequence, add a keyframe under the **Director** node on the **Camera** track. Be sure to leave the camera **Key** property set to **None**. Use this as the last keyframe on the **Director** node **Camera** track in your sequence in order to transition the last used sequence camera to the default game camera when playing the keyframe.

Blending Within a Sequence

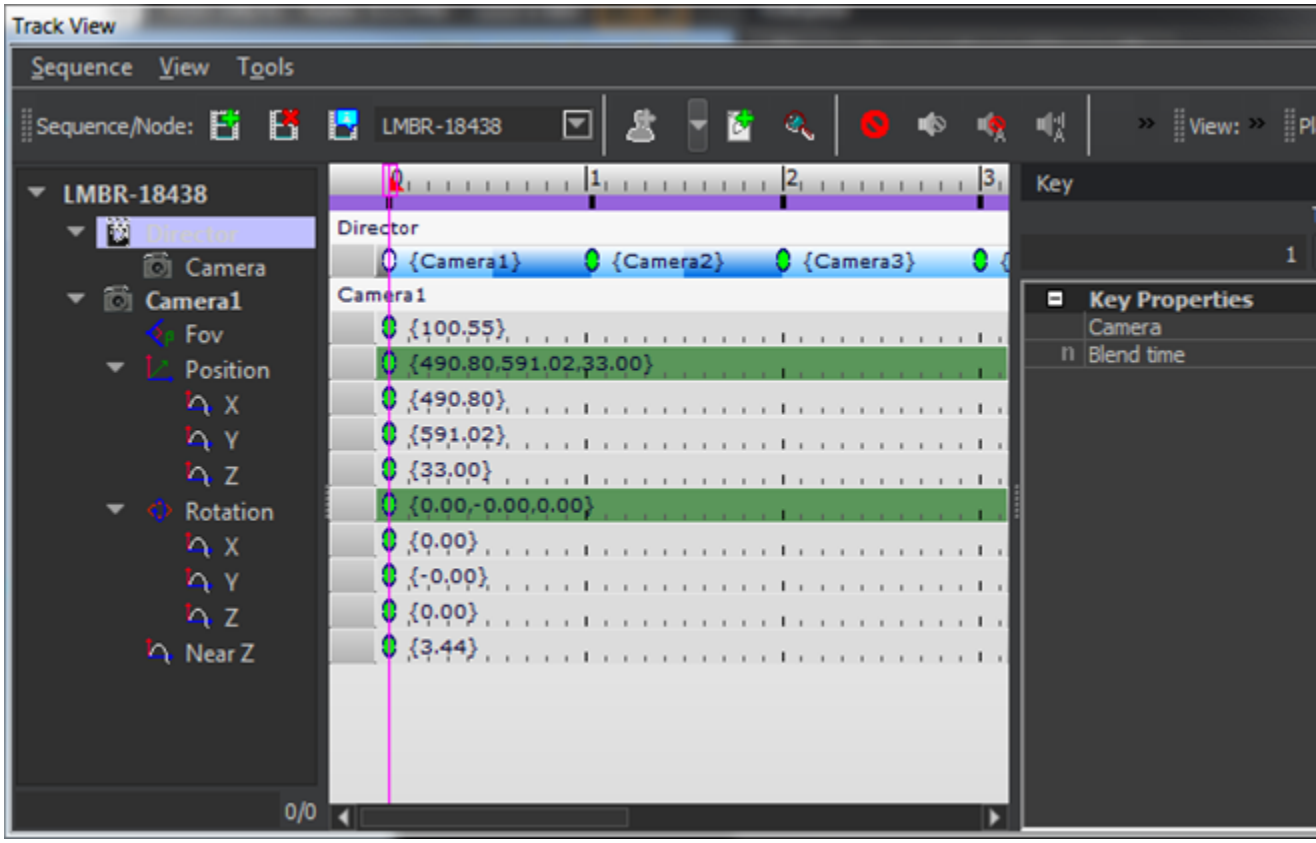
To create a series of jump cuts from one camera to another camera in a sequence, place key frames on the **Director** node **Camera** track. The **Director** node uses the **Camera** key property for each key frame to determine which camera to switch to.

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. This allows the cut to 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. Then 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**
> In this example, if you have added the first camera as a **Track View** node for **Camera1**, you must set at least one key frame for the **Position**, **Rotation**, and **FoV** when using blended sequence cameras.



Blending Into a Sequence

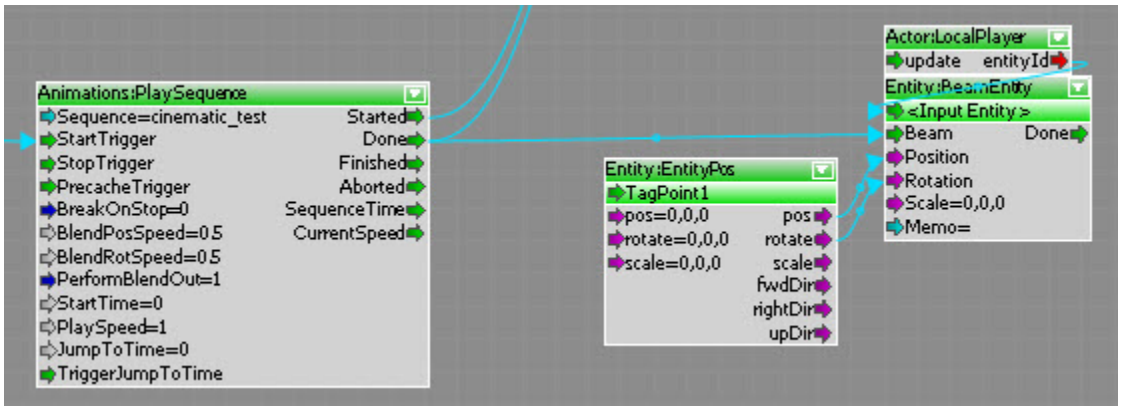
When blending into a sequence using Flow Graph, place the camera inside a large trigger that encloses the entire sequence. If you do not do this, snapping can occur when starting the sequence. For best results, set values between 0.5 and 2.0 for **BlendPosSpeed** and **BlenRotSpeed** in the **Animations:PlaySequence** node that is used to start the sequence.

Use the **Animations:PlaySequence** node to slow down player motions so that jumps, sprints, and slides transition more smoothly.

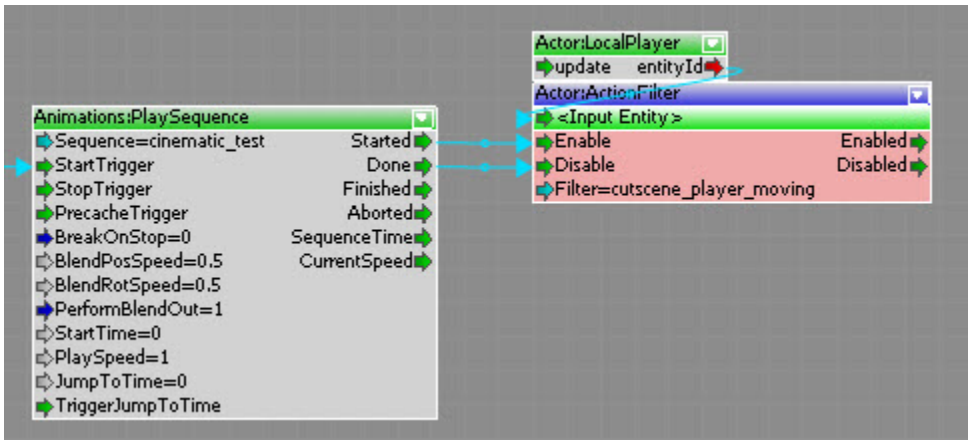
Blending Out of a Sequence (First-person Games)

When blending a camera out of a sequence, use the **Entity:BeamEntity** node to set the player to the end of the sequence. Position the **Entity:EntityPos** tag point 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** node directly to the **Beam** input of the **Entity:BeamEntity** node.

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.



Occasionally the player's last movement input is stored and remains active after the sequence, causing the player to continue to walk even though no key is pressed. To prevent this, use the **Actor:ActionFilter** 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, you can set the sequence camera by adding a **Director** node and adding keys to the **Camera** track. If a **Camera** track is set to **None**, 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° 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

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 that you want 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, and then click **Export**, **Export Selected**.
4. In **Select File to Export**, select **Autodesk (*.FBX)** for **Save as type**. 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 choose **Import FBX File**.
7. Browse to the `.fbx` file and choose **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

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

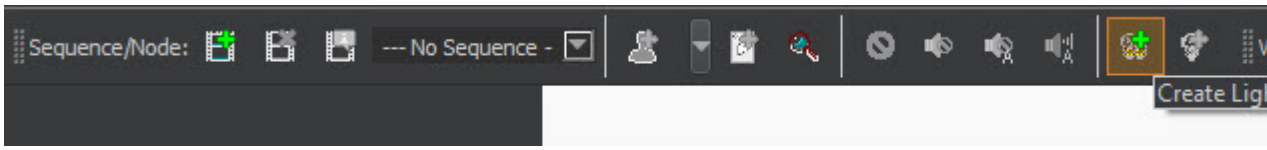
Creating lighting for cinematic scenes involves a different process than that used for creating environment lighting for a level.

Animating a Light

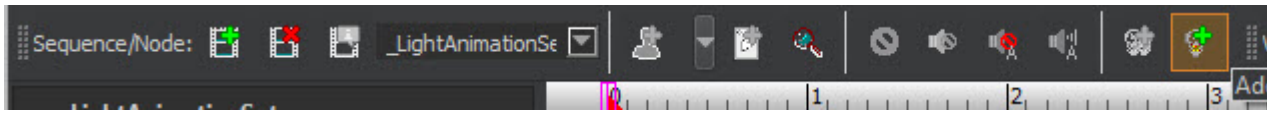
You can animate light entities in the **Track View** editor by creating a light animation set, which is a sequence that contains **Light Animation** nodes. A light entity then references these nodes with the **Light Animation** property in the Rollup Bar.

To create a new light animation set

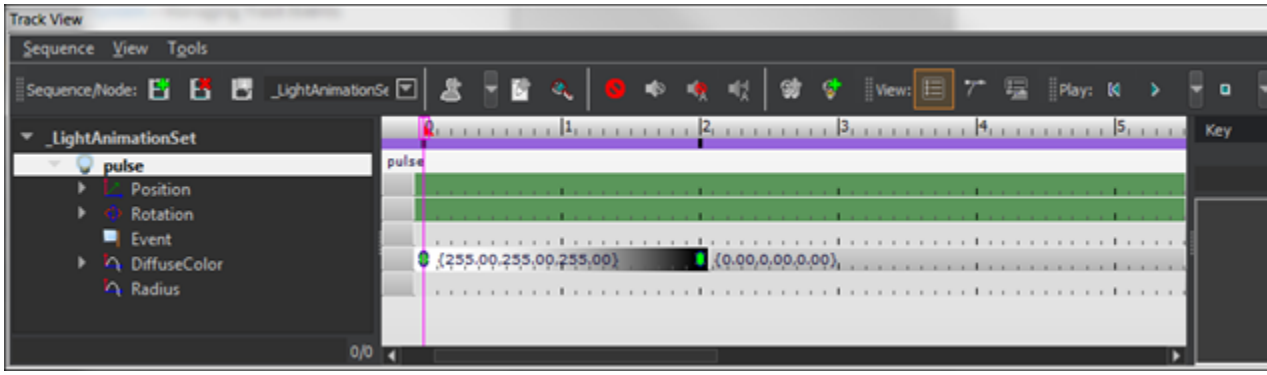
1. In the **Track View** editor, click **Create Light Animation Set**. You need to do this once per level.



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 the 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 **(...)** icon to access the **Select Light Animation** dialog.
6. Select the **Pulse** node and click **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 cinematics lighting.

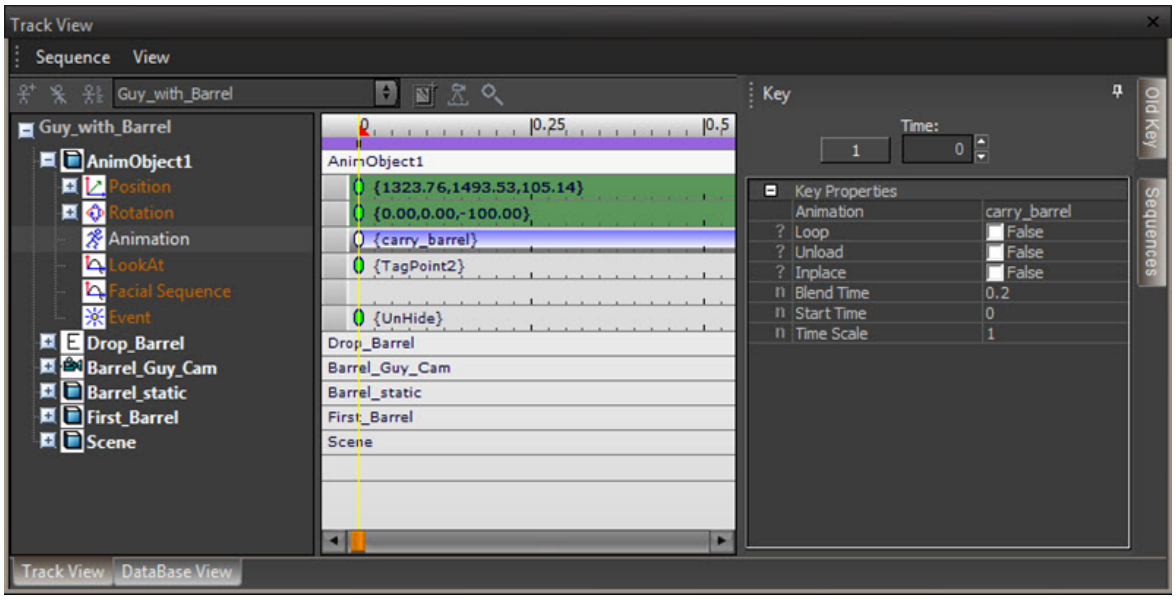
- Enable lights for a shot, then disable them when the camera cuts.
- Disable gameplay and cubemap lights as needed for shots to avoid interference.

- 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** (legacy) entity or a **Simple Animation** component.

In the case of a static asset, the **BasicEntity** (legacy) entity or **Static Mesh** component is used instead.



Topics

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

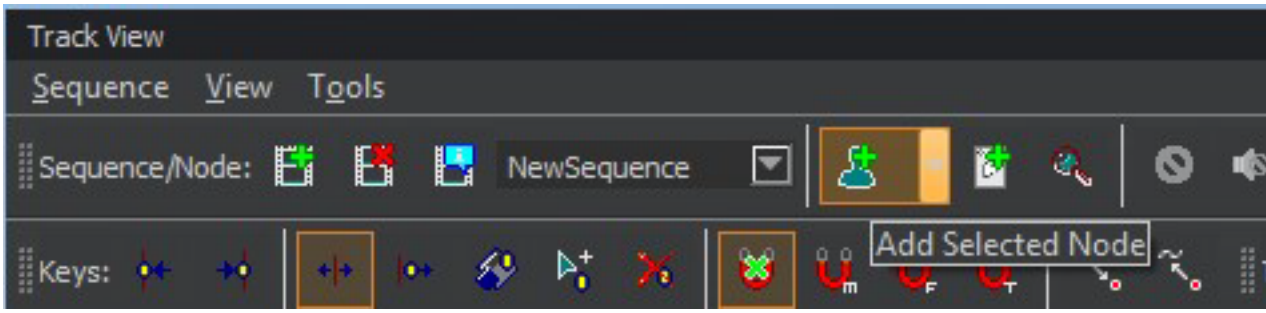
1. In Track View, right-click the node with the transform animation you wish to export and choose **Export FBX File**.
2. Choose and file name and click **Save** in the **Export Selected Nodes To FBX File**.
3. In your DCC tool, create a top-level node and name it exactly the same as the node selected 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 (legacy) entity or Simple Animation component 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. Track View triggers and blends character animation clips using an animation track.

To add an AnimObject (legacy) or Simple Animation component animation track

1. In the **Track View** editor, right-click the applicable **AnimObject** (legacy) or **Simple Animation** component node. Click **Add Track**, choose a track, and then click **Animation**.
2. Select the animation track in the tree pane, then double-click in the timeline window to place a key.
3. Click the green marker. Under **Key Properties**, adjust the values of the animation track key properties.

Animation Track Key Properties

Key Property	Description
Animation	Opens the animation browser for you to select an animation to apply to this key.
Loop	Repeats the animation until the next key.

Key Property	Description
Blend Gap	Blends the end frame of the first clip into the beginning frame of the second clip, if there is a time gap between animation clips. To use this feature, enable **Blend Gap** for the first animation.
Unload	Unloads the animation after the sequence is finished.
In Place	If set, does not change the entity's base position and orientation.
Start Time	Sets the time, in seconds, to start playing the animation within the clip. 0 indicates the start of the saved clip. The start time can never be greater than the end time.
End Time	Sets the time, in seconds, to stop playing the animation within the clip. 0 indicates the end of the saved clip. The start time can never be greater than the end time.
Time Scale	Specifies the factor to use to scale time. Values greater than 1 increase the animation speed. Values less than 1 result in a slow-motion animation.

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

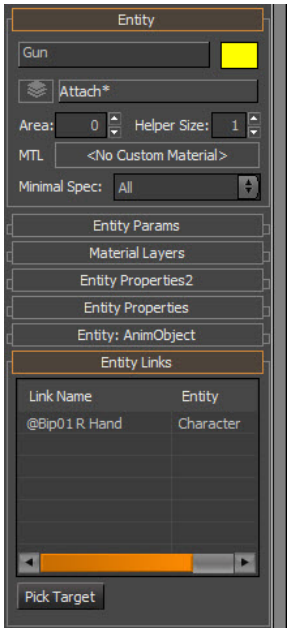
1. In the Track View editor, add the character to the desired sequence, then click the Red record button.
2. In the Lumberyard viewport, click the character, then move or rotate as desired. This automatically updates keys at the current position of the slider in the Track View sequence timeline slider.
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 identity 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.

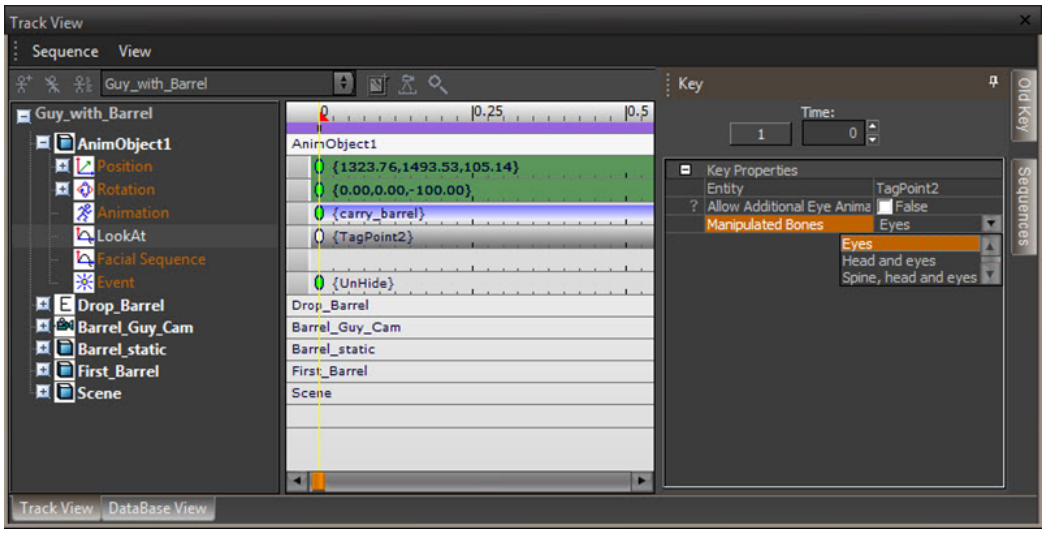
To use LookIK in a scene

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:

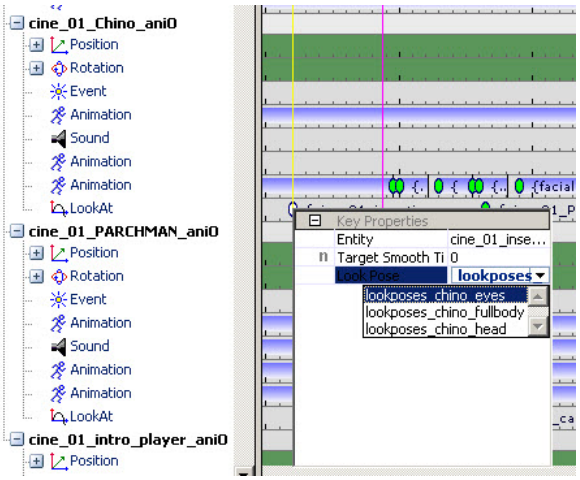
   - In **Entity**, select a target from the list.

- 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.

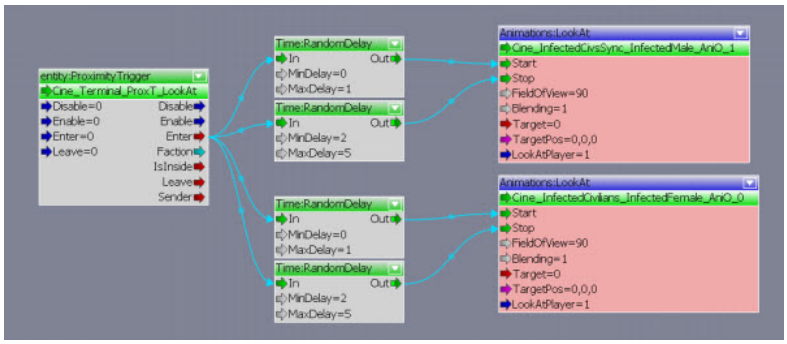


You can add multiple LookAt track keys. As soon as the timeline hits one key, the character aligns to the next key. If you want to reset LookIK, place an empty key in the timeline.



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.

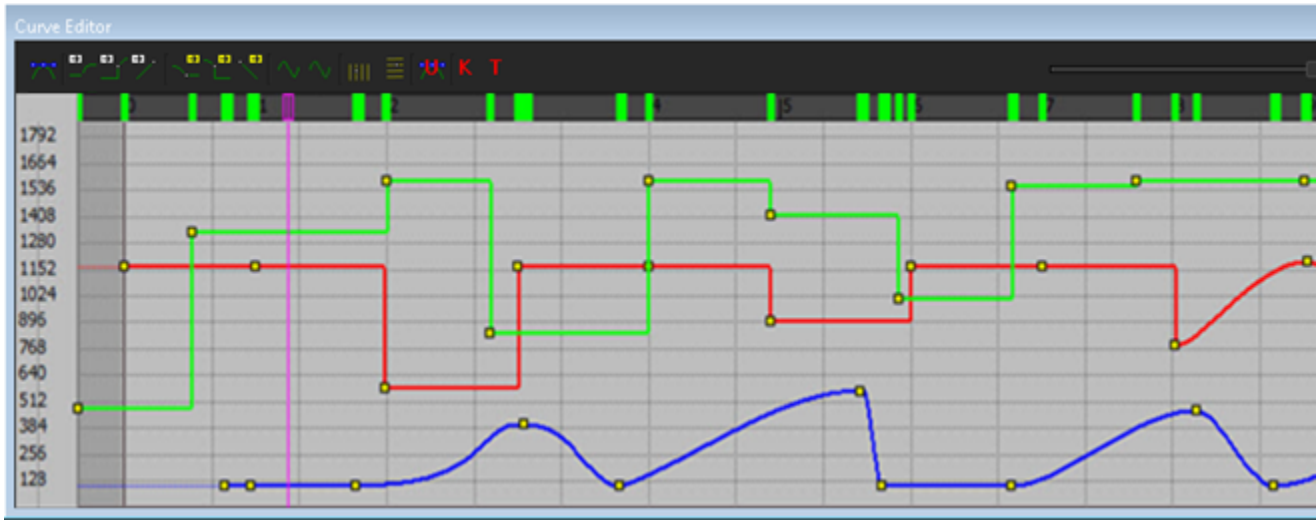
To use the Curve Editor for a scene

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.

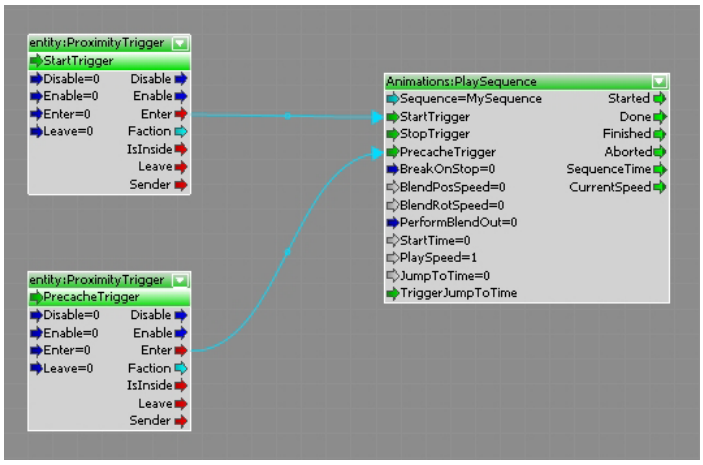


Pre-caching Cinematic Animations

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

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

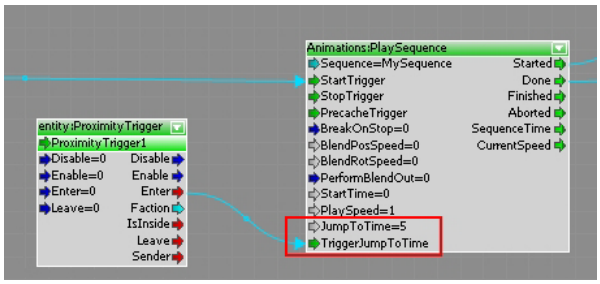
1. If applicable, in the Track View editor, right-click the top node and click **Add Director(Scene) Node**.
2. Right-click the applicable node and then click **Add Track**, **GoTo**.
3. Add a key in the GoTo track where you want the jump to occur.
4. In the timeline, right-click the key, and in **Key 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.

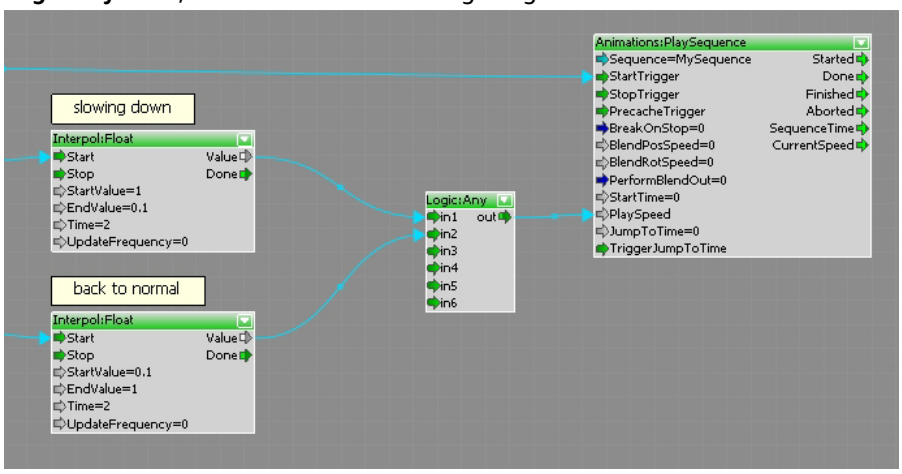
Using the Loop property instead of a GoTo track key is useful for moving mechanical parts (such as helicopter rotor blades) or when the animations are pose-matched and do not require blending. GoTo tracks could be used, but the effect does not look good.

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.



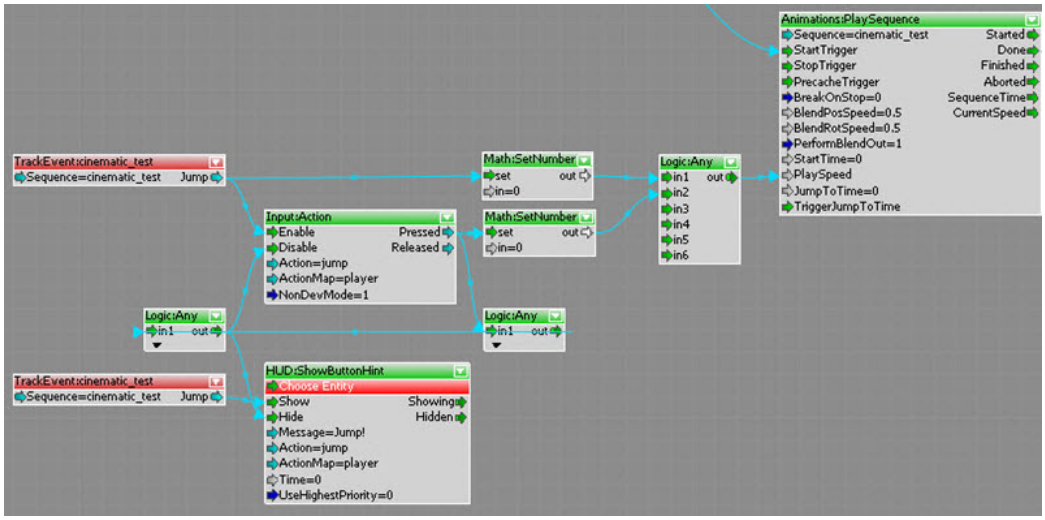
You can also set up multiple triggers that jump to different times using **Math:SetNumber** nodes and a **Logic:Any** node, as shown in the following image.



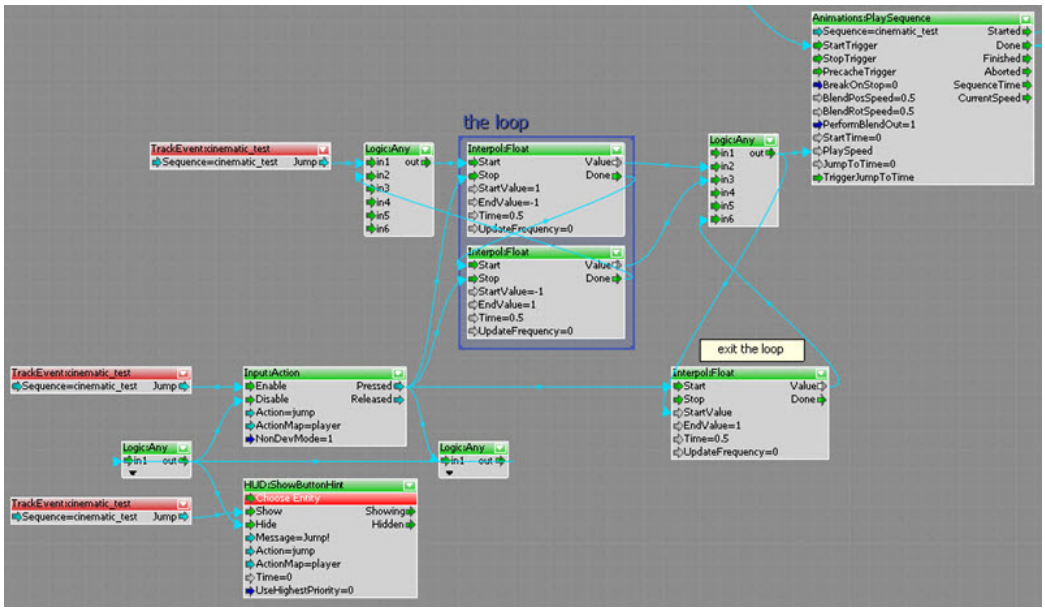
Pausing a Scene

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.

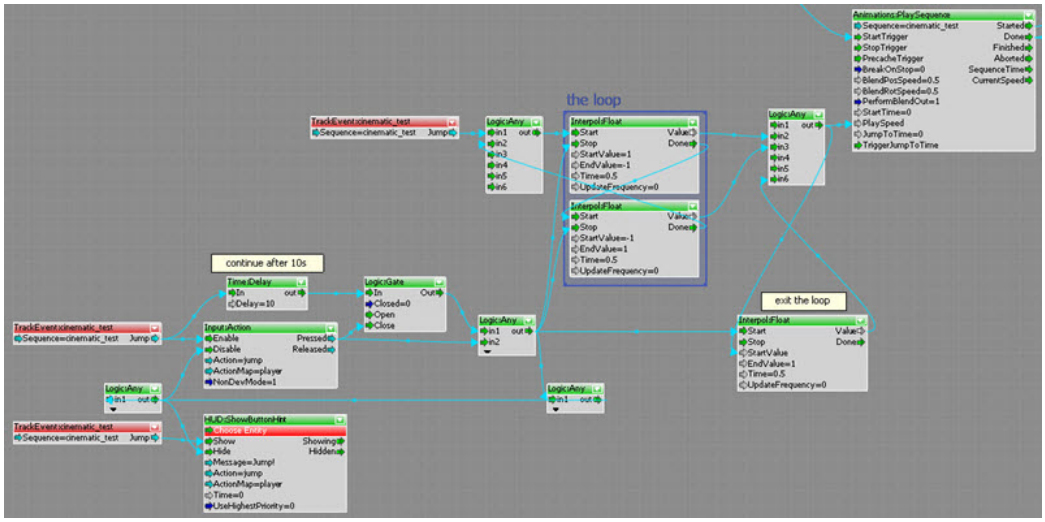
A simple implementation would be to add a track event to a sequence when the pause should happen. Then set the **PlaySpeed** input value to `0` in the **Animations:PlaySequence** Flow Graph node, and then to `1` when the player presses the required button, as shown below.



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.

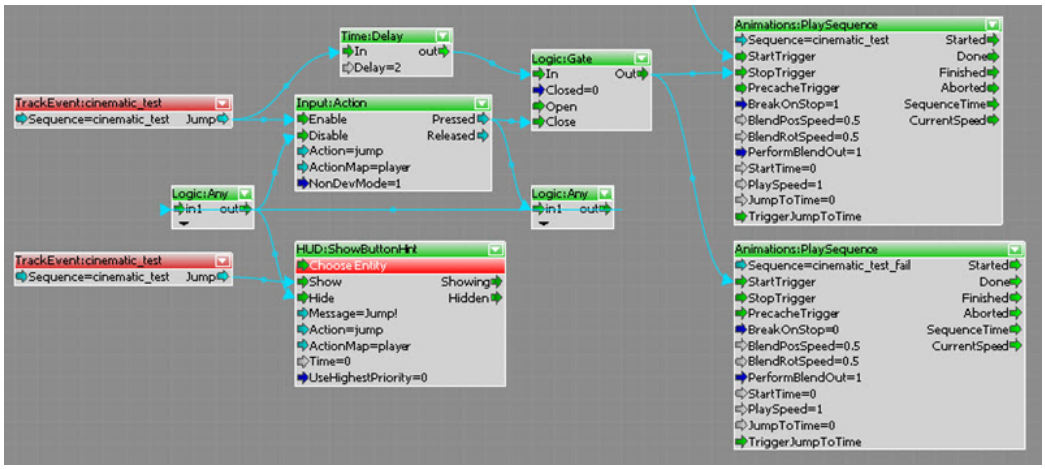


However, this would create an infinite pause of the sequence. To make it continue automatically after a certain amount of time, add a **Time:Delay** node as an optional path to the **Input:Action** node. Note the use of the **Logic:Gate** node that is used in the following example to prevent the sequence from continuing twice.

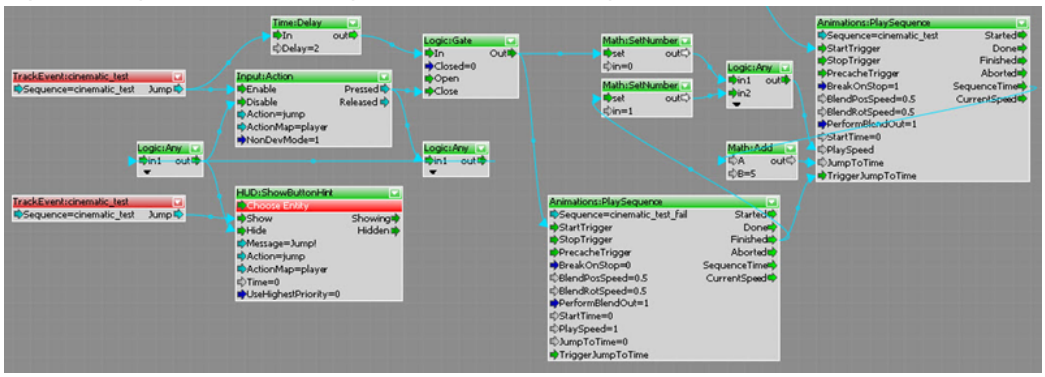


Adding a Dead-Man Switch to a Scene

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 in 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.

Capturing Image Frames

You can capture image frames using render output, a capture track, or console variables.

Capturing Image Frames using Render Output

You can use the **Render Output** tool in the **Track View** editor to capture image frames. You may need to adjust the aspect ratio for captured image frames.

To capture image frames using Render Output

1. In Lumberyard Editor, click **Tools**, **Track View**.
2. In **Track View**, click **Tools**, **Render Output**.
3. In **Render Output**, set the input and out properties as desired, and then click **Add**. You will see the capture added under **Batch**.
4. Click **Start** to start the capture.

To change the aspect ratio for image frame captures

1. In Lumberyard Editor, click **Edit**, **Editor Settings**, **Global Preferences**.
2. In **Preferences**, click **Viewports**.
3. Under **General Viewport Settings**, change the value for **Perspective View Aspect Ratio**. The default value is 1.3333.

Capturing Image Frames using a Capture Track

You can capture image frames when a sequence is played in game mode.

To capture image frames using a capture track

1. In Lumberyard Editor, click **Tools**, **Track View**.
2. In **Track View**, right-click the **Director** node and click **Add Track**, **Capture**.
3. Double-click the created track to add a capture key frame. See below for the key properties.
4. Set up a flow graph to play the sequence on game start. For information, see Playing a Sequence (p. 409).

Key Properties

Duration

Sets the capture duration in seconds.

Time Step

Forces a fixed frame rate, in seconds, by using a specified time step. Time step = 1/number of frames. 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

Specifies a prefix to apply to the image file names. For best results, use the same prefix as the sequence for clarity.

Output Folder

Specifies the directory where the image files are stored under `\directory Cache \project_name\pc\project_name\`.

Buffer(s) to capture

Specifies the image capture format. Choose from the following options:

- **Color** – RGB pixel information
- **Color+Alpha** – RGB pixel data and Alpha channel data

  Note
  To capture Alpha channel data, you must set **Output Format** to **tga**.

Just 1 Frame

Chooses between single or multi-frame image capture.

Capturing Image Frames using Console Variables

You can use console variables to capture image frames.

`fixed_time_step`

Lowers the game speed to achieve a constant frame rate throughout the sequence. For example, a time step value of `0.04` specifies a 25 fps gameplay speed.

Default: 0.0

`capture_frames`

Enables frame capture, if the value is set to 1.

`capture_file_format`

Sets the output format for the images.

Valid values: `.jpg`, `.tga`, `.tif`

`capture_file_prefix`

Sets a file name prefix to use for captured frames.

Default: `Frame`

`capture_buffer`

Sets the type of buffer to capture.

Valid values: 0 = Color (RGB pixels) | 1 = Color with Alpha (RGBA pixels where the alpha channel is set to 255 where geometry exists)

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. It employs reflection, serialization, messaging using the event bus (EBus), fully cascading slices, and the ability to drag-and-drop and edit entities and their components in Lumberyard Editor.

The topics that follow describe how to add components to entities and customize their properties in Lumberyard Editor. For information on programmatically creating your own custom components, see Component Entity System in the *Amazon Lumberyard Developer Guide*.

Use the following Lumberyard Editor tools to improve the workflow for the component entity system.

- Entity Outliner (p. 437)
- Entity Inspector (p. 441)
- Asset Browser (p. 179)

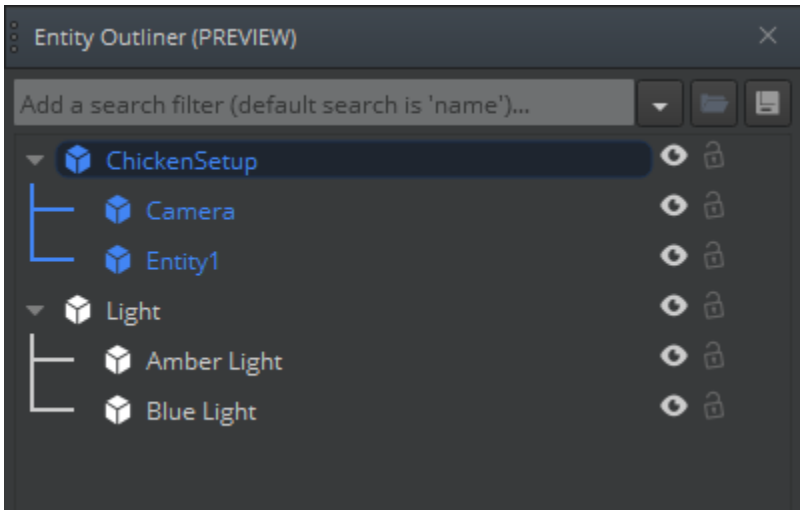
> **Note**
> The component entity system replaces the (now legacy) Object and Entity System (p. 613) in Lumberyard.

Topics

Entity Outliner

Component entity system is in preview release and is subject to change.

The **Entity Outliner** shows all the entities and slices in the level. You can view the parenting hierarchies, lock selections, show and hide entities in the viewport, and create and save search filters.



From the **Entity Outliner**, you can do the following:

- Create a new entity (p. 593)
- Reorder slices and entities (p. 438)
- Hide and show parent and child entities (p. 440)
- Lock parent entities or individual child entities (p. 440)
- Perform searches on all entities (p. 441)
- Load and save search filters (p. 441)

Entities in blue are part of a slice (p. 597).

Entities in white represent freestanding entities that are not part of a slice.

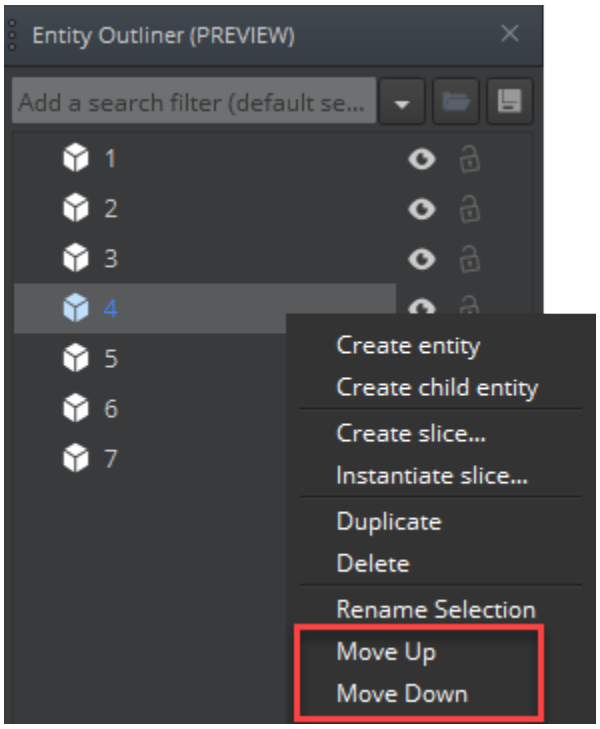
To open the Entity Outliner

- In Lumberyard Editor, choose **Tools**, **Entity Outliner**.

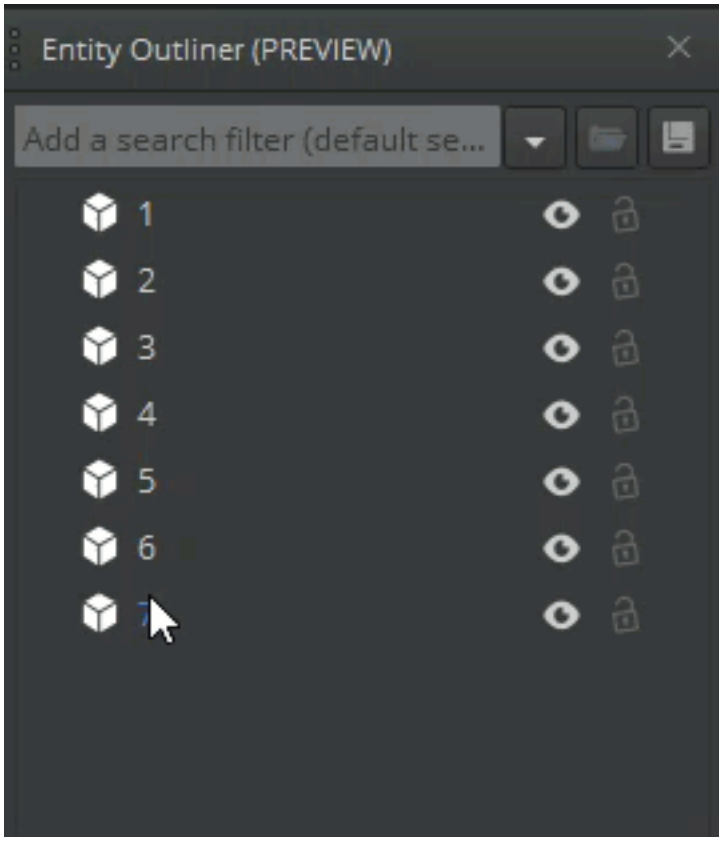
Reordering Entities

When you create new entities or instantiate slices, they appear at the bottom of the list in the **Entity Outliner**.

To move an entity up or down, right-click on an entity and then choose **Move Up** or **Move Down**.

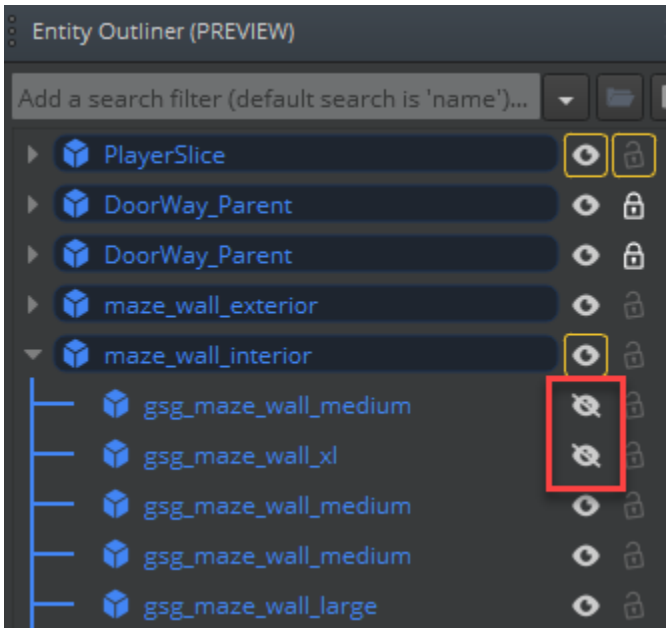


You can also reorder entities by dragging and dropping one or more entities into the preferred order. To reorder, drag until the white line appears in your preferred location. The purple square indicates parenting rather than reordering.



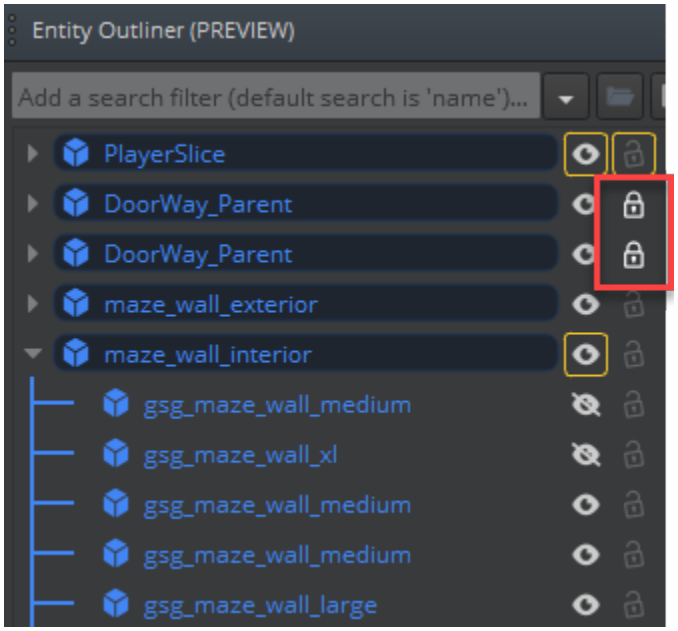
Hiding and Showing Entities

To hide a parent entity and all its children in the viewport, click on the eye icon next to the parent. The crossed-out eye icon denotes any entities currently hidden in the viewport. You can also hide child entities within a parent entity.



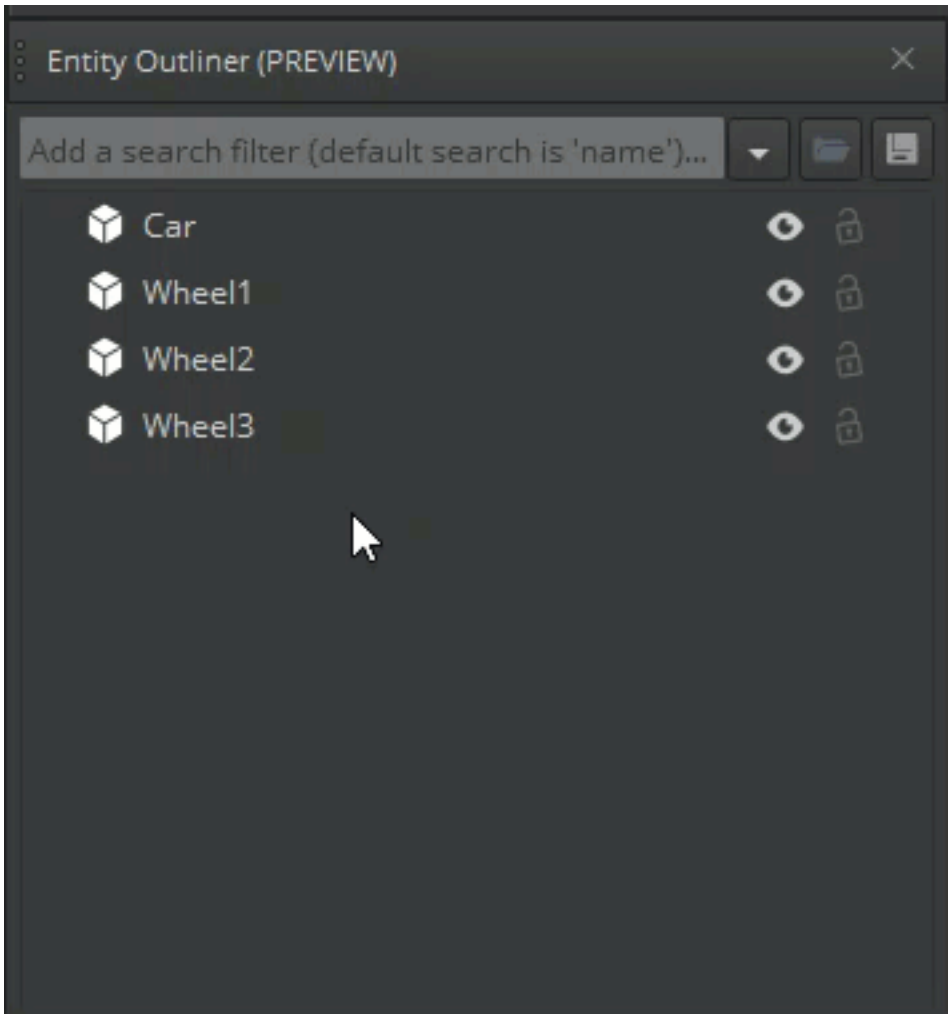
Locking Entities

To lock a parent entity and all its children in the viewport, click the lock icon next to the parent. This prevents selection of the parent or its child entities in the viewport. You can also lock child entities within a parent entity.



Parenting

To make an entity the child of another entity, drag and drop the entity name onto its intended parent.



Filtering

Type text in the search filter box to find specific entities. Any entity whose name does not match is hidden.

Delete all text to resume showing all entities.

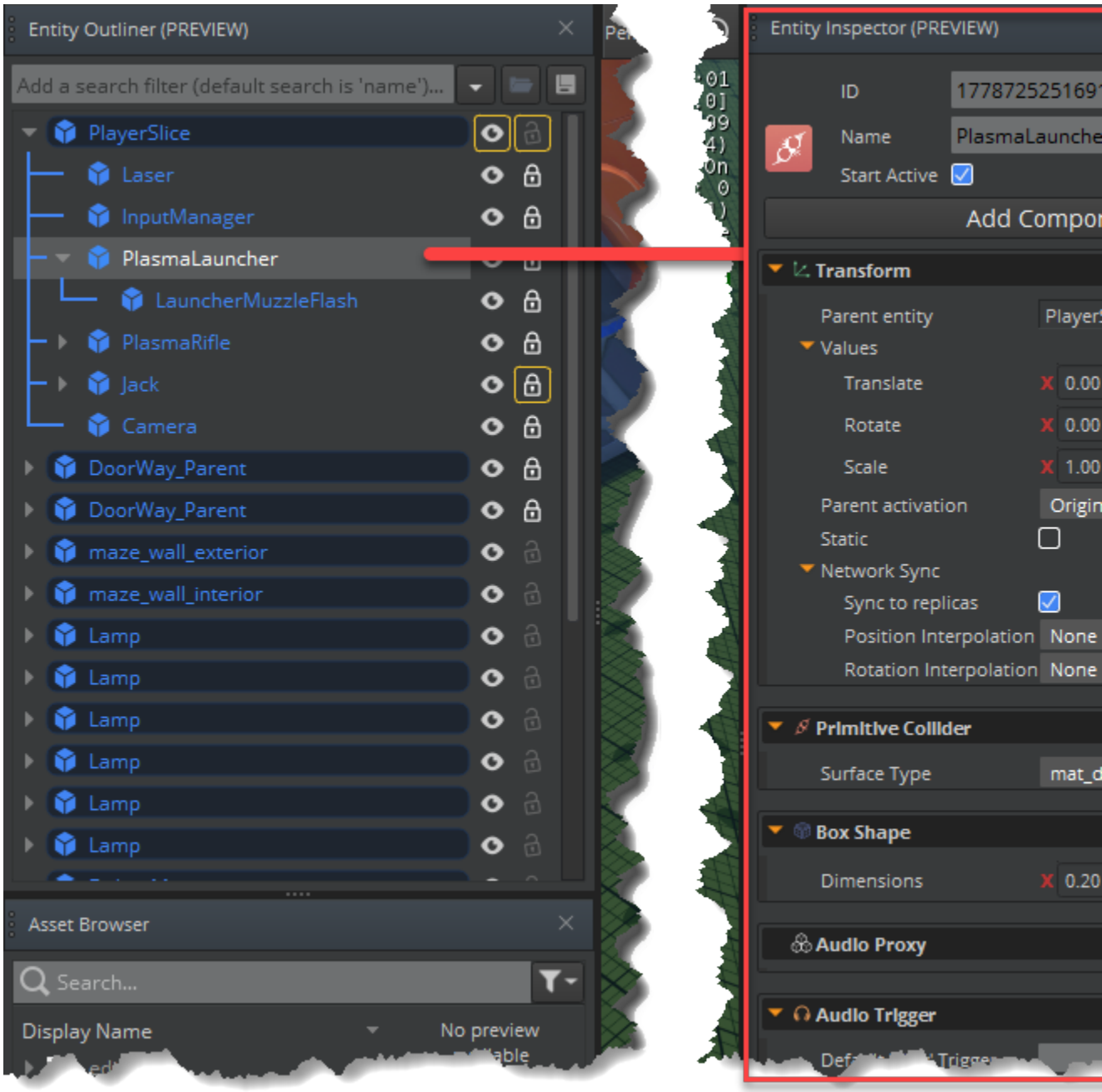
To save a search filter, click the save icon. To load a saved search filter, click the folder icon.

Entity Inspector

Component entity system is in preview release and is subject to change.

The **Entity Inspector** manages all the components for each entity. Select an entity in the **Entity Outliner** to see and perform actions on the components in the **Entity Inspector**. At the top of the **Entity Inspector**, the following appears:

- **ID** – Unique ID number generated by Lumberyard; cannot be modified.
- **Name** – Name of the entity. Type in the box to rename the entity.
- **Entity Icon** – Customizable icon to help you recognize entities in the viewport.



Use the **Entity Inspector** to do the following:

- Add components to entities (p. 593) and modify their properties (p. 596)

- Customize the entity icon (p. 443)
- Reorder components (p. 594)
- Remove, copy, cut, and paste components (p. 594)
- Modify component properties (p. 596)
- Start an entity as inactive (p. 444)
- Copy and paste asset references between components (p. 445)
- Create your own help topic for your custom component (p. 445)

For a list and descriptions of available components, see Component Reference (p. 446). You can also click the **Help** icon on the title header of each component to open the *Amazon Lumberyard User Guide* to that component's help topic.

To open the Entity Inspector

- In Lumberyard Editor, choose **Tools**, **Entity Inspector**.

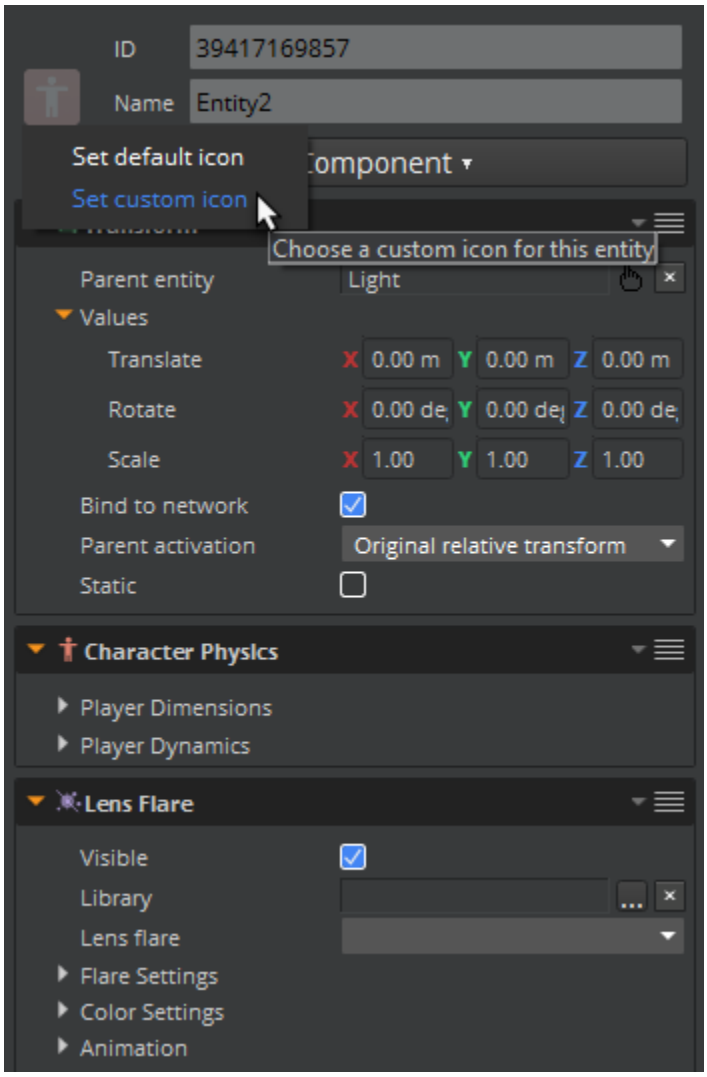
Customizing the Entity Icon

The default icon for an entity without any added components is the **Transform** ( ) component's icon. When you add another component, the icon changes to the first component that you add to that entity.

You can also specify your own icon.

To customize an entity icon

1. Click the image of the icon at the top of the **Entity Inspector** (p. 441).
2. Choose **Set custom icon**.



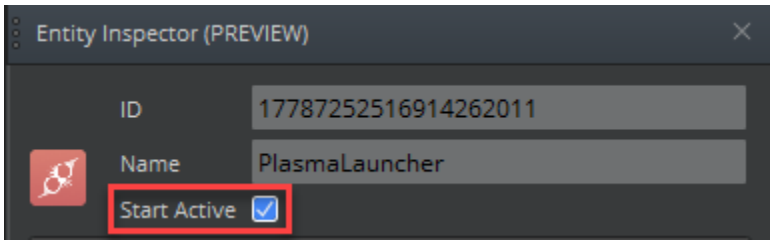
3. Select an icon from your game project directory.

Starting an Entity as Inactive

By default, an entity starts as active in a level. When creating a game, you may prefer that an entity remain inactive until activated through some mechanism (such as a script or player action).

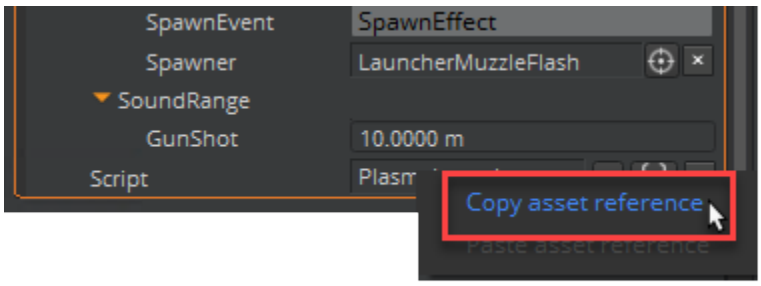
To start an entity as inactive

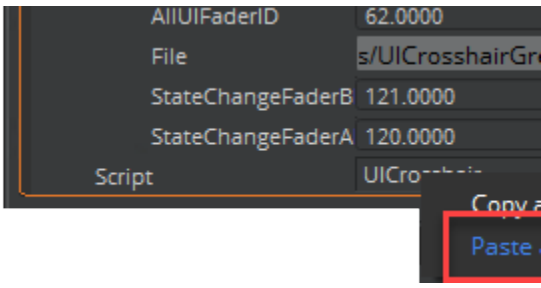
1. In the **Entity Outliner**, select an entity.
2. In the **Entity Inspector**, clear the check box for **Start Active**.



Copying and Pasting Asset References

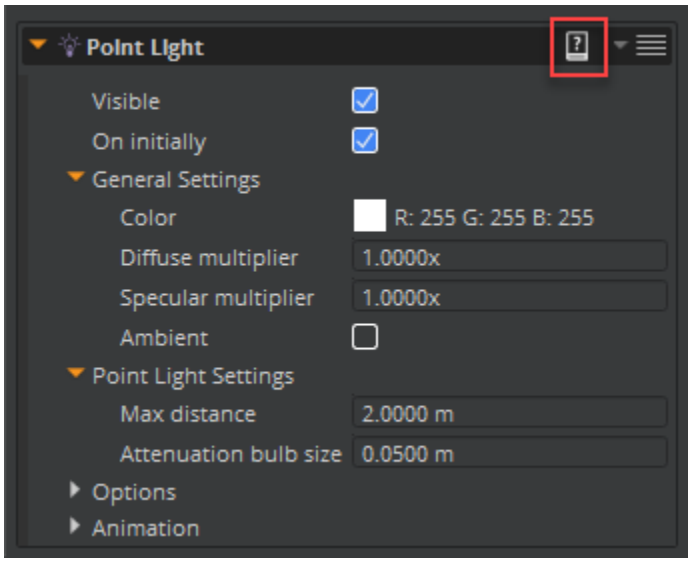
In the **Entity Inspector**, you can copy an asset reference from one component and paste it into another component. You can copy such assets as scripts, mesh assets, particle effect libraries, cubemap assets, and so on.





Creating Custom Component Help Topics

If you have created your own components, you can point the component header's help icon to your own documentation.



To do this, add the `HelpPageURL` attribute to your component reflection.

For example:

```
Attribute(AZ::Edit::Attributes::HelpPageURL, "https://docs.aws.amazon.com/lumberyard/
latest/userguide/component-comment.html")
```

Component Reference

Component entity system is in preview release and is subject to change.

The following components are available to add to entities using the Entity Inspector:

AI

- the section called “Behavior Tree” (p. 465)
- the section called “Navigation” (p. 523)

Animation

- the section called “Actor” (p. 448)
- the section called “AnimGraph” (p. 449)
- the section called “Attachment” (p. 454)
- the section called “Simple Motion” (p. 555)

Animation (Legacy)

- the section called “Attachment (Legacy)” (p. 454)
- the section called “Mannequin” (p. 513)
- the section called “Mannequin Scope Context” (p. 520)
- the section called “Motion Parameter Smoothing” (p. 522)
- the section called “Simple Animation” (p. 550)

Audio

- the section called “Audio Area Environment” (p. 456)
- the section called “Audio Environment” (p. 457)
- the section called “Audio Listener” (p. 458)
- the section called “Audio Proxy” (p. 460)
- the section called “Audio Rtpc” (p. 460)
- the section called “Audio Switch” (p. 461)
- the section called “Audio Trigger” (p. 462)

Camera

- the section called “Camera” (p. 467)
- the section called “Camera Rig” (p. 468)

Editor

- the section called “Comment” (p. 482)

Gameplay

Network

Physics

Rendering

Scripting

Shape

- Sphere Shape (p. 546)

UI

- the section called “UI Canvas Asset Ref” (p. 581)
- the section called “UI Canvas Proxy Ref” (p. 582)
- the section called “UI Canvas on Mesh” (p. 581)

VR

- the section called “VR Preview” (p. 592)

Miscellaneous

- Transform (p. 564)

Actor

Component entity system is in preview release and is subject to change.

You can use the **Actor** component to create characters for your game. After you import your character files from your DCC tool into Lumberyard, you can create an entity and add the **Actor** component to it. For example, you must use an **Actor** component to create a controllable character for your game.

For the **Actor** component to work properly, you must also add one of the following:

- **Simple Motion (p. 555)** component – Uses a single motion for your actor.
- **AnimGraph (p. 449)** component – Uses an animation graph to control your actor's behavior.

Actor Component Properties

The **Actor** component has the following properties:

Actor Asset

Lets you select the actor that you want to add to your entity.

LOD Materials

Lets you select the material that is linked to your actor asset.

Attachment Type

The **Actor** component has the following attachment types:

Actor Attachment

Attaches the current actor to the target entity.

Skin Attachment

Attaches a skin attachment to a target entity.

Options

The **Actor** component has the following options:.

Draw skeleton

Determines whether character joints are visible.

Draw character

Determines whether character mesh is visible.

Skinning method

Lets you select the skinning method to use for the actor. You can choose the following options:

- **Linear Skinning GPU** – Sends the bone transforms as dual quaternions to the GPU and converts them to matrices on the GPU before blending them together for scaling.
- **Linear Skinning CPU** – Converts the bone transforms to matrices on the CPU before sending the matrices to the GPU for skinning.

You can also enable linear skinning in Geppetto. For more information, see Enabling Linear Skinning with Geppetto (p. 290).

AnimGraph

Component entity system is in preview release and is subject to change.

You can use the **AnimGraph** component to add an animation graph and motion set to your character. Add this component to the Actor (p. 448) component to control character behavior from an animation graph. For single motions, see the Simple Motion (p. 555) component.

AnimGraph Component Properties

The **AnimGraph** component has the following properties:

Anim graph

Lets you select the animation graph that was created in the Animation Editor.

Motion set

Lets you select the motion set that was created in the Animation Editor.

Parameters

Displays the parameters that were created with the Animation Graph.

Area Light

Component entity system is in preview release and is subject to change.

Use the **Area Light** component on an entity to light an area.

The **Area Light** component has the following settings:

Visible

Shows the light.

On initially

When created, the light is on by default.

General Settings

See the following general settings:

Color

The color of the light.

Default value: 0xFFFFFFFF

Diffuse multiplier

Sets the strength of the diffuse color.

Default value: 1

Specular multiplier

Sets the strength of the specular brightness.

Default value: 1

Ambient

Light acts as a multiplier for cubemap values.

Area Light Settings

See the following area light settings:

Area width

Width of the area light in meters.

Default value: 5

Area Height

Height of the area light in meters.

Default value: 5

Max Distance

Maximum distance in meters that the area light extends.

Default value: 2

Options

See the following options:

View distance multiplier

Adjusts the maximum view distance. For example, **`1.0`** uses the default and **`1.1`** is 10% farther than the default.

Default value: 1

Minimum spec

Minimum specular value at which the light is enabled.

Default value: Low

Cast shadow spec

The minimum specular at which shadows are cast.

Default value: Never

Voxel GI mode

Mode for light interaction with voxel global illumination (GI).

Default value: None

Ignore vis areas

Light ignores vis areas.

Indoor only

Light is only rendered indoors.

Affects this area only

Light affects only the immediate area.

Volumetric fog only

Light affects only volumetric fog.

Volumetric fog

Light affects volumetric fog and surrounding area.

Animation

See the following animation settings:

Style

Type a number to specify a preset light animation curve to play as defined in the `Light.cfx` file. Valid values are **`0`** to **`48`**. You can also use values **`40`** to **`48`** for testing and debugging.

Default value: 0

Speed

Multiple of the base animation rate. For example, a value of **`2.0`** makes an animation play twice as fast.

Default value: 1

Phase

Animation start offset from **`0`** to **`1`**. A value of **`0.1`** is 10% into the animation. For example, you can use this setting, for to prevent lights in the same scene, with the same animation, from being animated in unison.

Default value: 0

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).

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 off the light.

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) in the *Amazon Lumberyard Developer Guide*.

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()
    LightComponentRequestBus.Event.TurnOnLight(self.entityId)
    LightComponentRequestBus.Event.TurnOffLight(self.entityId)
    LightComponentRequestBus.Event.ToggleLight(self.entityId)
end
```

Attachment

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.

Attachment Component Properties

The **Attachment** component has the following properties:

Target Entity

Specify the character entity that you want to attach.

Joint Name

Specify the joint that you want to attach.

Position Offset

Allows x, y, and z local positional offsets from the target bone.

Rotation Offset

Allows x, y, and z local rotational offsets from the target bone.

Attached Initially

The attachment attaches to the target entity automatically.

Scaling

The **Attachment** component has the following scaling options:

Use world scale

Use the scale of the world.

Use target entity scale

Use the scale of the target entity.

Use target bone scale

Use the scale of the target bone.

Topics

- Attachment (Legacy) (p. 454)

Attachment (Legacy)

Note
The following component works with the legacy Geppetto system.

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) in the *Amazon Lumberyard Developer Guide*.

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

`offsetTransform` – Attachment's offset from target.

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) in the *Amazon Lumberyard Developer Guide*.

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 Area Environment

Component entity system is in preview release and is subject to change.

The **Audio Area Environment** component enables entities that are moving around and throughout a shape to have environment effects applied to any sounds that they trigger. You must also add a shape component to use the audio area environment component.

Audio Area Environment Properties

The **Audio Area Environment** component has the following properties:

Broad-phase Trigger Area

Link this property to an entity that includes a trigger area and shape. This trigger area is used for broad-phase checks. The **Audio Area Environment** component tracks any entity that moves inside the trigger area.

Default: None

Environment name

The name of the Audio Translation Layer (ATL) (p. 188) environment to apply to entities in the area.

Default: None

Environment fade distance

The distance around a shape where the environment amounts will fade based on an entity's distance from the shape. Only positive, non-zero values are valid.

Default: 1.0

Using the Audio Area Environment Component

Setting up the **Audio Area Environment** component requires two entities. The second entity is linked to the **Audio Area Environment** component in the first entity, acting as a broad-phase trigger area. When these entities are configured properly, any entity that passes near or through the inner shape of the first entity will have an environment amount applied to any triggered sounds.

To set up the Audio Area Environment component

1. In Lumberyard Editor, right-click the viewport in your level, and click **Create new component entity**.
2. Click **Tools**, **Entity Inspector**. Be sure that your new component entity is selected in the viewport.
3. In the **Entity Inspector**, click **Add Component**, **Shape**, and then select one of the shape options.
4. Click **Add Component**, **Audio**, **Audio Area Environment**.
5. Repeat steps 1 – 4 to add another entity with a **Shape** (any) component and a **Trigger Area** component (located under **Scripting**).
6. Place and size the two entities so that the following conditions are met:
   - The second entity's shape completely encompasses the first entity's shape.
   - The second entity's shape is larger than the first entity's shape by at least the value for the **Environment fade distance**. This allows the **Audio Area Environment** component to track an entity's distance from the inner shape when the entity enters the outer trigger area.
7. (Optional) In the **Entity Inspector**, for the **Trigger Area** component, use the **Tag Filters** to filter entities that you don't want the **Audio Area Environment** component to process.

Audio Environment

Component entity system is in preview release and is subject to change.

The **Audio Environment** component provides access to features of the Audio Translation Layer (ATL) (p. 188) 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) in the *Amazon Lumberyard Developer Guide*.

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 Listener

Component entity system is in preview release and is subject to change.

The **Audio Listener** component allows a virtual microphone to be placed in the environment. An audio listener acts as a sink for sound sources in the virtual world, and 3D audio rendering is processed with respect to the listener's world transform. You can specify the audio listener's position and rotation independently.

Audio Listener Properties

The **Audio Listener** component has the following properties:

Rotation Entity

Link this property to an entity where the audio listener adopts the rotational part of the transform. The current entity is used if a value is not specified.

Position Entity

Link this property to an entity where the audio listener adopts the positional part of the transform. The current entity is used if a value is not specified.

Using the Audio Listener Component

Only one audio listener is supported in a game. You can add the **Audio Listener** component to an entity that contains the game camera.

To set up the Audio Listener component

1. In Lumberyard Editor, right-click the viewport in your level, and click **Create new component entity**.
2. Click **Tools**, **Entity Inspector**. Be sure that your new component entity is selected in the viewport.
3. In the **Entity Inspector**, click **Add Component**, **Audio**, **Audio Listener**.

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) in the *Amazon Lumberyard Developer Guide*.

SetRotationEntity

Specify the entity with the rotational part of the transform that the audio listener will adopt.

Parameters

`entityId` – Entity to use for the rotational part of the transform

Return

None

Scriptable

Yes

SetPositionEntity

Specify the entity with the positional part of the transform that the audio listener will adopt.

Parameters

`entityId` – Entity to use for the positional part of the transform

Return

None

Scriptable

Yes

SetFullTransformEntity

Specify the entity with the full transform that the audio listener will adopt.

Parameters

`entityId` – Entity to use for the transform

Return

None

Scriptable

No

Audio Proxy

Component entity system is in preview release and is subject to change.

The Audio Proxy component is a required dependency if you add multiple audio components to an entity. It acts as a proxy audio object wrapped in a component. For example, if you have an audio trigger component and an audio rtpc component on the same entity, they communicate to the same audio object using this audio proxy component.

Audio Rtpc

Component entity system is in preview release and is subject to change.

The **Audio RTPC** component provides basic Real-Time Parameter Control (RTPC) (p. 188) 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) in the *Amazon Lumberyard Developer Guide*.

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 entity system is in preview release and is subject to change.

The **Audio Switch** component provides basic Audio Translation Layer (ATL) (p. 188) 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. 186) 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) in the *Amazon Lumberyard Developer Guide*.

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 entity system is in preview release and is subject to change.

The **Audio Trigger** component provides basic play and stop features so that you can set up Audio Translation Layer (ATL) (p. 188) 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) in the *Amazon Lumberyard Developer Guide*.

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 the 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) in the *Amazon Lumberyard Developer Guide*.

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 entity system is in preview release and is subject to change.

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) in the *Amazon Lumberyard Developer Guide*.

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**.

```
local behaviortreescript =
```

```
{
    Properties =
    {
        Target = EntityId(),
    },
}

function behaviortreescript:OnActivate()
    BehaviorTreeComponentRequestBus.Event.StartBehaviorTree(self.entityId)
    BehaviorTreeComponentRequestBus.Event.SetVariableValue(self.entityId,
 Crc32("HasTarget"), self.Properties.Target:IsValid())
end

return behaviortreescript
```

Camera

Component entity system is in preview release and is subject to change.

The camera component allows an entity to be used as a camera. To use the camera component, you must first add a Camera Framework Gem to your project. For information, see the section called "Camera Framework Gem" (p. 1075).

Camera Component Properties

The **Camera** component has the following properties:

Field of View

Vertical field of view in degrees.

Valid values: 0 – 180

Default value: 75

Near Clip Plane Distance

Distance to the near clip plane of the view frustum in meters.

Default value: 0.2

Far Clip Plane Distance

Distance to the near far plane of the view frustum in meters.

Default value: 1024

EBus Request Bus Interface

Use the following request functions with the event bus (EBus) interface, `CameraRequestBus`, to communicate with other components of your game.

For more information about using the EBus interface, see Event Bus (EBus) in the *Amazon Lumberyard Developer Guide*.

GetFov

Gets the current field of view.

SetFov

Sets the current field of view.

GetNearClipDistance

Gets the current near clip distance.

SetNearClipDistance

Sets the current near clip distance.

GetFarClipDistance

Gets the current far clip distance.

SetFarClipDistance

Sets the current far clip distance.

The following is an example of script using the **Request Bus Interface**.

```
local camerasample =
{
    Properties =
    {
    }
}

function camerasample:OnActivate()
    CameraRequestBus.Event.SetFov(self.entityId, 85)
    local nearClip = CameraRequestBus.Event.GetNearClipDistance(self.entityId)
    CameraRequestBus.Event.SetFarClipDistance(self.entityId, nearClip + 1024)
end

return camerasample
```

Creating Camera Entity from View

You can create a static camera view from a specific entity by right-clicking an entity in the viewport and choosing **Create camera entity from view**. This places a new entity with a camera component at the same point. You can adjust the view of the camera by modifying its transform component.

Camera Rig

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. To use the camera component, you must first add the the section called “Camera Framework Gem” (p. 1075) to your project.

Camera Rig Component Properties

The **Camera Rig** component has the following properties:

Target acquirers (p. 469)

Array of behaviors that define how a camera selects a target. The rig tries each acquirer in the order listed until one successfully finds a target.

Look-at behaviors (p. 470)

Array of behaviors that modify the look-at target transform. The rig runs each in order to generate a final target transform.

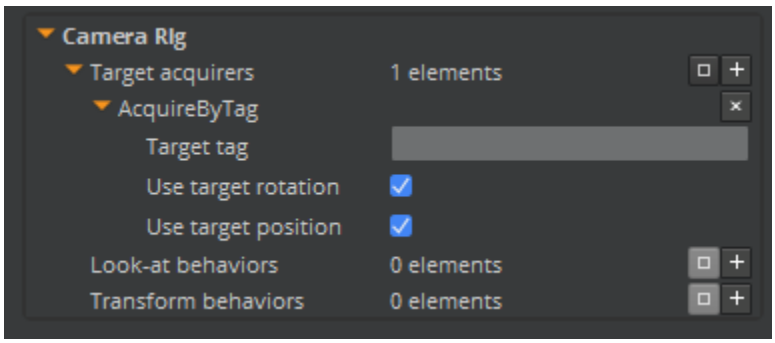
Transform behaviors (p. 472)

Array of behaviors that modify the camera transform based on the look-at target transform. The rig runs each in order before setting the camera component's transform.

Target Acquirers

Target Acquirers identify valid targets and acquire their transforms for use in other rig behaviors.

Acquire By Tag



AcquireByTag has the following properties:

Target tag

Find a target by tag. If multiple entities are found, it uses the first to respond.

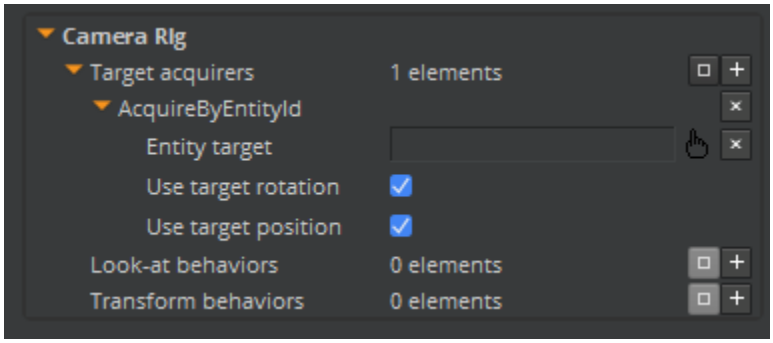
Use Target Rotation

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.

Acquire By Entity Id



AcquireByEntityId has the following properties:

Entity Target

Select a specific entity to use as the camera target

Use Target Rotation

If selected, uses the target's rotation when determining camera behavior.

Use Target Position

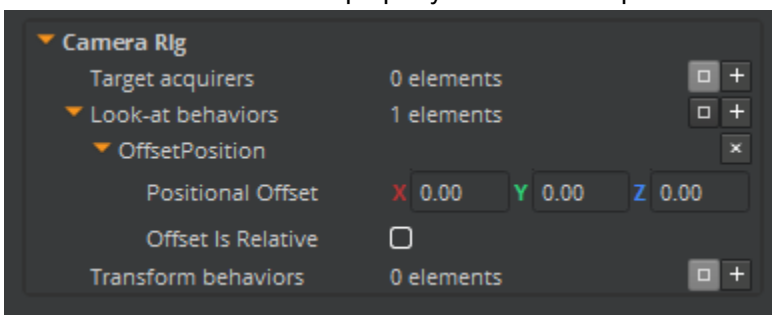
If selected, uses the target's position when determining camera behavior.

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.

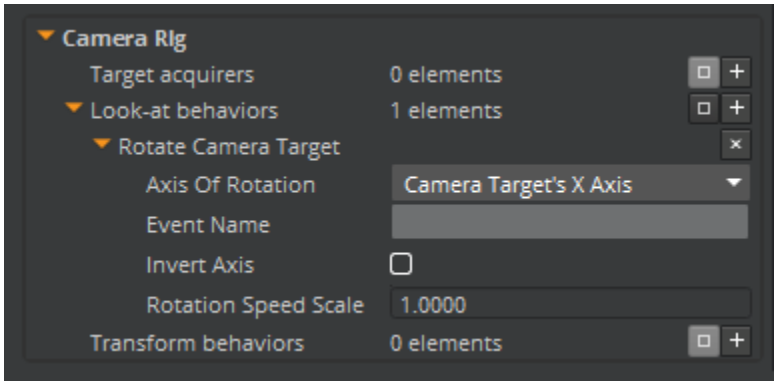
Offset Is Relative

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

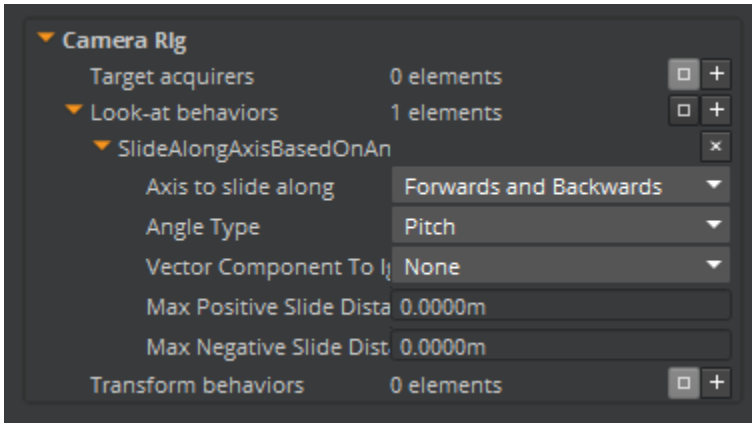
If selected, inverts the axis of rotation.

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

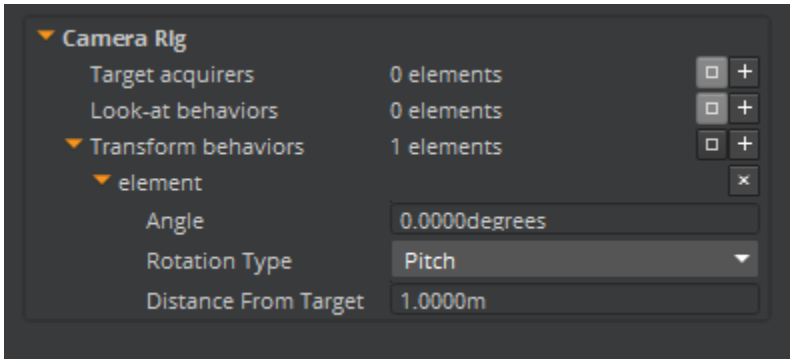
Transform Behaviors are a critical component of how the camera responds to the target. For example, you can set the camera to face the target, follow from a distance, or follow the target at a specific angle.

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.

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

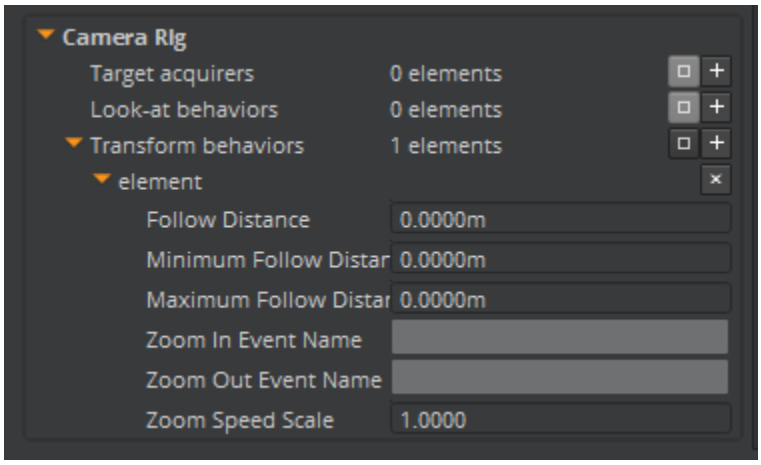
Rotation type of the angle for following the target: yaw, pitch, or roll.

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

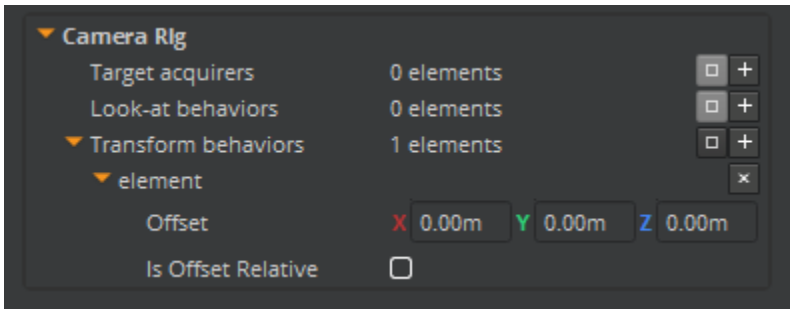
Scale amount for the incoming zoom value.

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

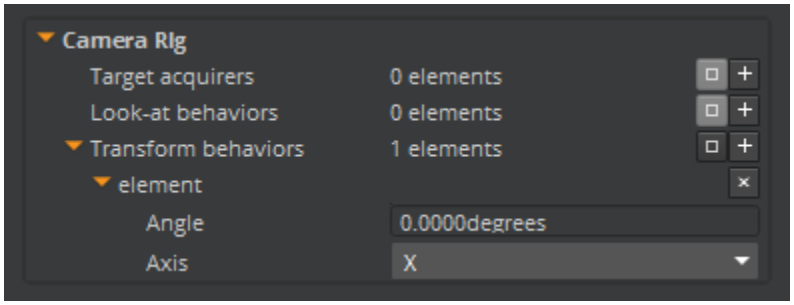
The vector offset in meters from the target.

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.

Character Physics

Component entity system is in preview release and is subject to change.

The **Character Physics** component adds physical behavior to and configures simulation characteristics for character entities, such as players and enemies.

Character Physics Component Properties

The **Character Physics** component has the following properties.

Player Dimensions

Player Dimensions determine the character's physical properties.

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

Player Dynamics settings control the physics of the entity.

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

You can set the following limits for the **Character Physics** component.

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

The **Collides with type** settings affect what collisions the character entity can experience.

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) in the *Amazon Lumberyard Developer Guide*.

EnablePhysics

Makes the entity a participant in the physics simulation.

Parameters

None

Return

None

Scriptable

Yes

DisablePhysics

Stops the entity from participating in the physics simulation.

Parameters

None

Return

None

Scriptable

Yes

IsPhysicsEnabled

Queries if the entity's physics component has been enabled.

Parameters

None

Return

Bool – whether physics is enabled

Scriptable

Yes

AddImpulse

Adds an impulse to the entity.

Parameters

Vector3 – an impulse in world space

Return

None

Scriptable

Yes

GetVelocity

Gets the velocity of the entity.

Parameters

None

Return

Vector3 - velocity of the entity to set

Scriptable

Yes

SetVelocity

Sets the velocity of the entity.

Parameters

Vector3 - velocity of the entity

Return

None

Scriptable

Yes

GetAcceleration

Gets the acceleration of the entity.

Parameters

None

Return

Vector3 – acceleration of the entity

Scriptable

Yes

GetAngularVelocity

Gets the angular velocity of the entity.

Parameters

None

Return

Vector3 – angular velocity of the entity

Scriptable

Yes

GetMass

Gets the total mass (kg) of the entity.

Parameters

None

Return

float – the total mass of the entity

Scriptable

Yes

SetMass

Sets the total mass (kg) of the entity.

Parameters

Float – the total mass to set

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) in the *Amazon Lumberyard Developer Guide*.

GetPhysicsParameters

Passes in any class that inherits from `pe_params` to retrieve them, such as `pe_params_pos`.

Parameters

(output) `pe_params&`

Return

None

Scriptable

No

SetPhysicsParameters

Passes in any class that inherits from `pe_params` to set them, such as `pe_params_pos`.

Parameters

const `pe_params&`

Return

None

Scriptable

No

GetPhysicsStatus

Passes in any class that inherits from `pe_status` to retrieve them, such as `pe_status_pos`.

Parameters

(output) `pe_status&`

Return

None

Scriptable

No

ApplyPhysicsAction

Passes in any class that inherits from `pe_action` to set them, such as `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) in the *Amazon Lumberyard Developer Guide*.

Move

Requests movement from a 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**.

```
local ExampleComponent = {
    Properties = {
        MoveSpeed = 1.0
    }
}

function ExampleComponent:OnActivate()
    -- connect to the TickBus so we receive OnTick events
    self.handler = TickBus.Connect(self, 0)
end

function ExampleComponent:OnTick(deltaTime, scriptTime)
    -- Jump modes
    -- 0 not jumping
    -- 1 jumping, instant velocity change
    -- 2 jumping, add to current velocity
    local jumpMode = 0

    -- get this entity's transform
    local transform = TransformBus.Event.GetWorldTM(self.entityId)
```

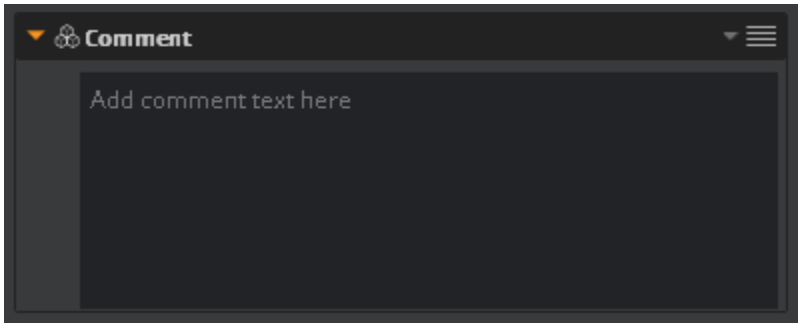
```
    -- we want to apply velocity in the forward direction
    local velocity = transform:GetColumn(1) * deltaTime * self.Properties.MoveSpeed

    -- request velocity
    CryCharacterPhysicsRequestBus.Event.RequestVelocity(self.entityId, velocity, jumpMode)
end

function ExampleComponent:OnDeactivate()
    -- disconnect from the TickBus
    self.handler:Disconnect()
    self.handler = nil
end

return ExampleComponent
```

Comment

Component entity system is in preview release and is subject to change.

The **Comment** component allows you to add long-form text comments for component entities. When enabled, the **Comment** component displays a dialog box that expands based on the size of the comment that you enter. The following examples demonstrate how you can use the comment text box:

- Explain how the scripts or components on an entity interact with other scripts or components
- Describe how everything in a level ties together
- Send descriptions, instructions, or notes to team members



Comment Properties

The **Comment** component has the following property:

Comment text box

Stores the user comment for the component entity.

Default: None

Using the Comment Component

You can use this feature by adding the component to an entity in your level.

To use the Comment component

1. In Lumberyard Editor, right-click the viewport in your level, and click **Create entity**.
2. In the **Entity Inspector**, click **Add Component**.
3. Under **Editor**, click **Comment**.
4. In the **Entity Inspector**, under **Comment**, add comments for the component entity in the text box.

Constraint

Component entity system is in preview release and is subject to change.

The **Constraint** component creates a physical limitation or restriction between an entity and its target, such as a door rotating about a set of hinges, a sliding glass door moving along a track, or a bell clapper swinging within its bell housing.

A working **Constraint** component setup has three distinct pieces: the **Constrained entity**, the **Constraint target**, and the **Constraint pivot**, which is the entity with the constraint component on it. To use the previous example, the door would be the constrained entity, the wall with the hinges would be the constraint target, and then a third entity, placed between them, would be the constraint pivot. The constraint pivot defines the constraint behavior, connects the constrained entity to the constraint target, and provides the point about which the constrained entity moves.

The **Constrained entity** is always a movable object. For example, the rotating door, the sliding glass door, and the bell clapper are all movable objects.

The **Constraint target** is an anchor to which the constraint pivot is attached. The constraint target can be movable or unmovable; for example, a wall with door hinges or a sliding glass door track aren't typically movable, but a bell is.

The **Constraint pivot** is typically a third entity that contains only the constraint component, and is placed between the contrained entity and the constraint target. The constraint pivot defines the constraint behavior type, such as ball, hinge, slider, plane, magnet, fixed, or free. There are other ways to use the constraint pivot as well, such as the following:

- Add the constraint pivot directly to the constrained entity, and then set the **Owning entity** to **Self**. This causes the constrained entity to move in the defined constraint behavior type about its own axis.
- Set the constraint pivot's **Target type** to **World space**. This causes the constraint pivot to be anchored to its position in world space rather than to another object. The **Constraint target** field is not available when you select world space.

The following procedure provides an example for placing a constrained entity, a constraint target, and a constraint pivot.

To place a constrained entity, a constraint target, and a constraint pivot

1. To place the constrained entity, and then add necessary mesh and physics components to it, follow these steps:

   a. Create an entity. (p. 593)

   b. In the Entity Inspector (p. 441), click **Add Component**, **Rendering**, **Static Mesh** (p. 521) or **Skinned Mesh** (p. 558). Next to **Static asset** or **Skinned asset**, click the ellipsis button to select an asset.

This defines the shape and appearance of your object.

c. In the Entity Inspector (p. 441), click **Add Component**, **Physics**, **Rigid Body Physics** (p. 537). Set rigid body properties as appropriate.

   This defines the entity as a movable object.

d. Click **Add Component**, **Physics**, **Primitive Collider** (p. 534) or **Mesh Collider** (p. 522).

   This allows other objects to collide with this object.

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 Physics** (p. 561) (instead of rigid body).
3. To place the constraint pivot, do the following:

   a. Create an entity. (p. 593)
   b. In the **Entity Inspector** (p. 441), click **Add Component**, **Physics**, **Constraint**.
   c. For **Owning entity**, select **Other entity**.

      You will select the owning entity (constrained entity) in a later step.

      In this example procedure, you are placing the constraint pivot between the constrained entity and the constraint target. The other option, **Self**, is used when you place the constraint component directly on the constrained entity.

   d. For **Target type**, select **Entity**.

      The other option, **World space**, causes the constrained entity to move about the world space position of the constraint pivot.

   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. 484) as appropriate.

Constraint Component Properties

The primary **Constraint** component property is the **Constraint type**. This property defines how the constrained entity moves in relation to the constraint target.

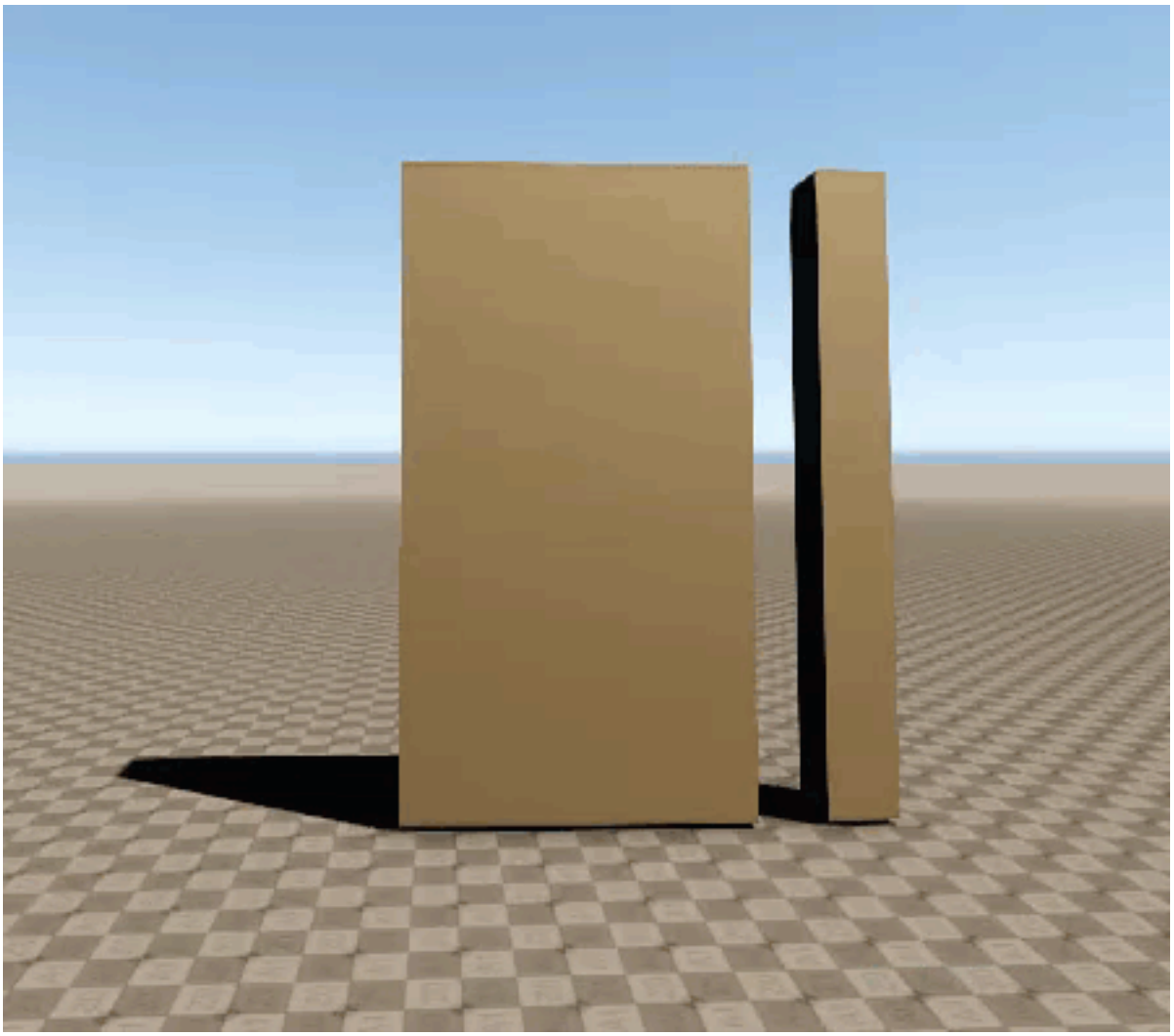
Constraint Type

When setting constraint properties, select one of the following **Constraint types**.

Hinge

The constrained entity rotates around the selected **Axis** as if attached by a hinge. This constraint type provides one degree of freedom (X, Y, or Z) between the constrained entity and the constraint target.

An example of a hinge is a door, which rotates around the Z axis.

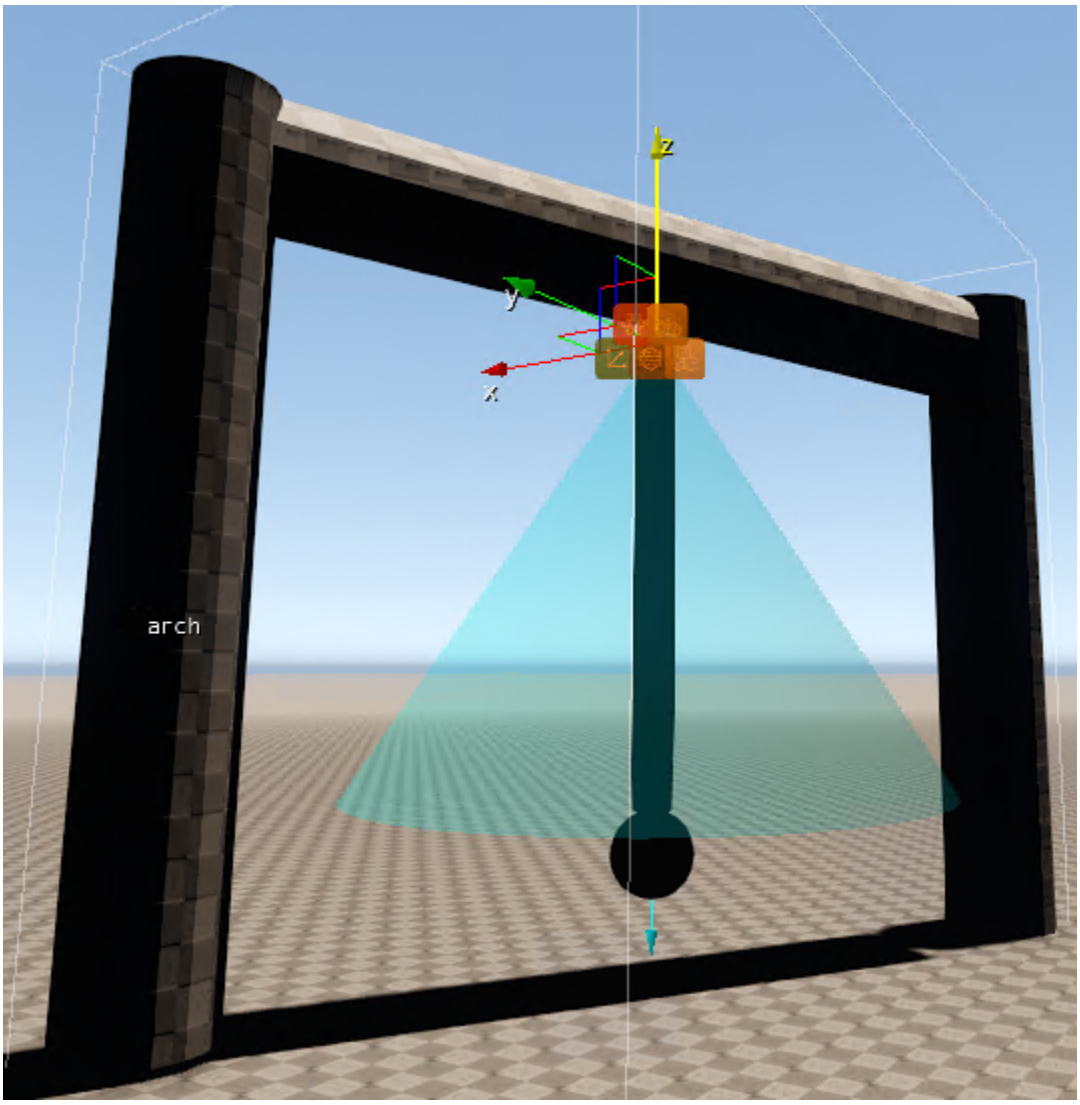


Ball

The constrained entity rotates around the z-axis as if attached at the constraint pivot 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).

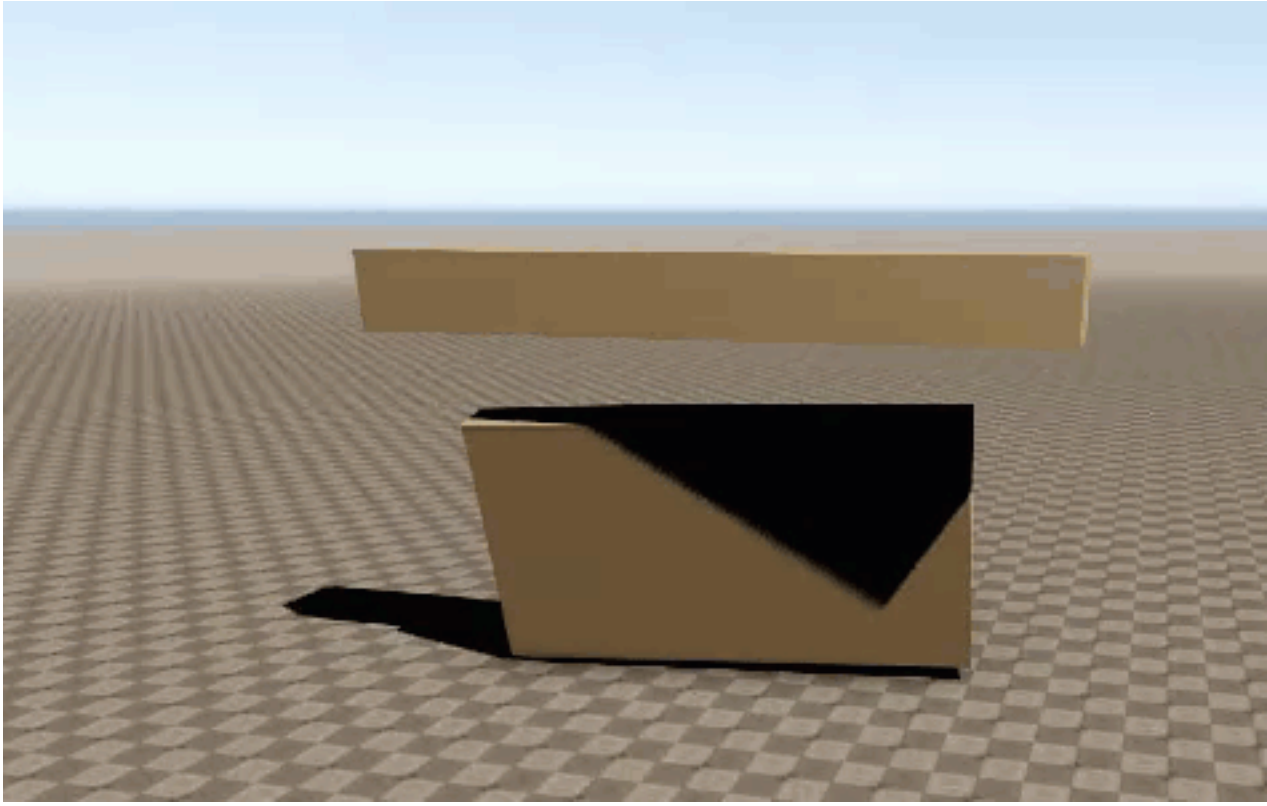
This constraint type always moves about the z-axis. If you want to use this constraint type, but need it to move in a different direction (for example, attaching at the bottom and moving at the top), you must change the orientation of the z-axis. To do this, rotate the constraint pivot.



Slider

The constrained entity slides along one axis. If rotation is also enabled, then the constrained entity acts like a movable hinge around the selected axis.

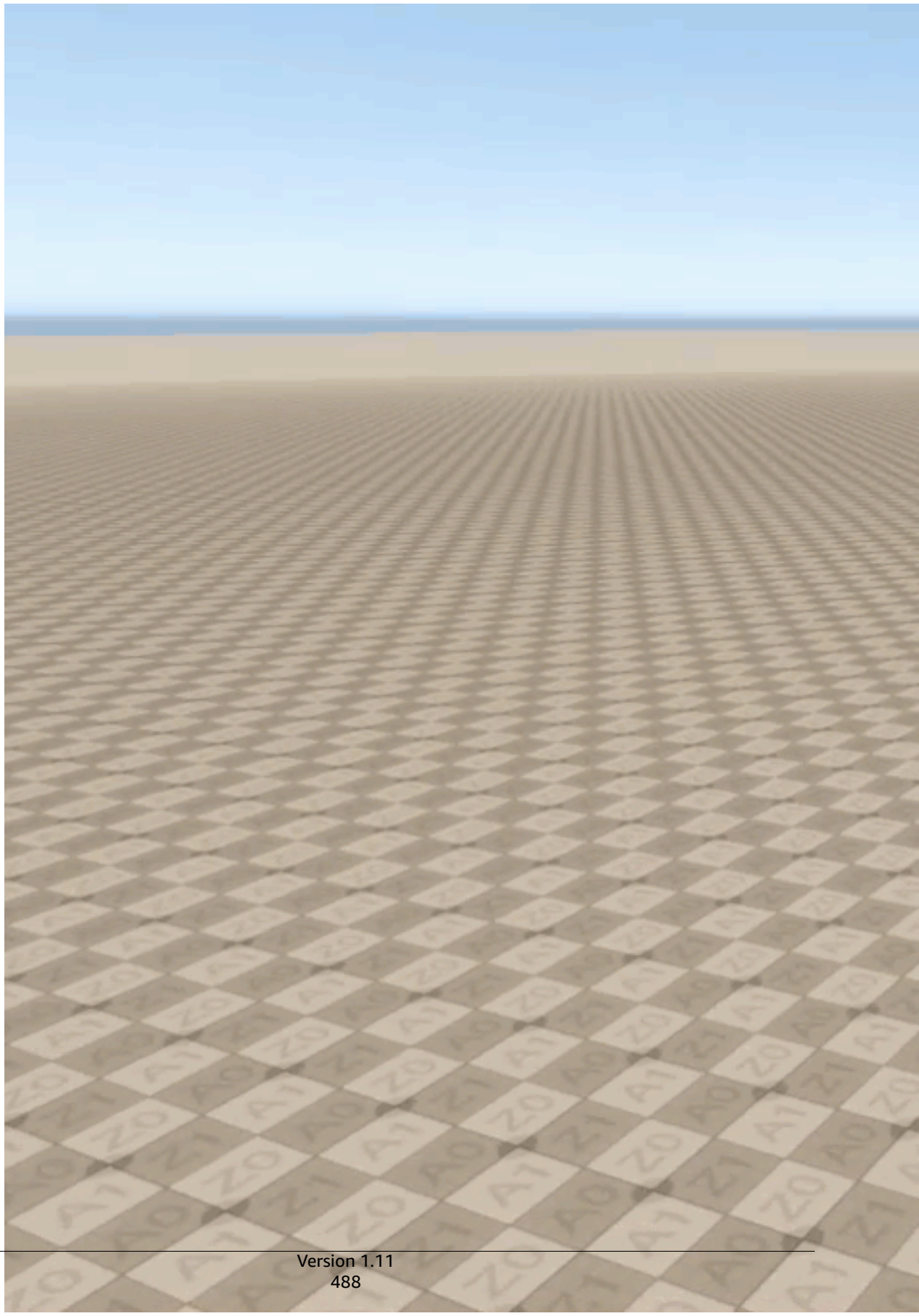
An example of a slider constraint is a sliding door.



Plane

The constrained entity 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.



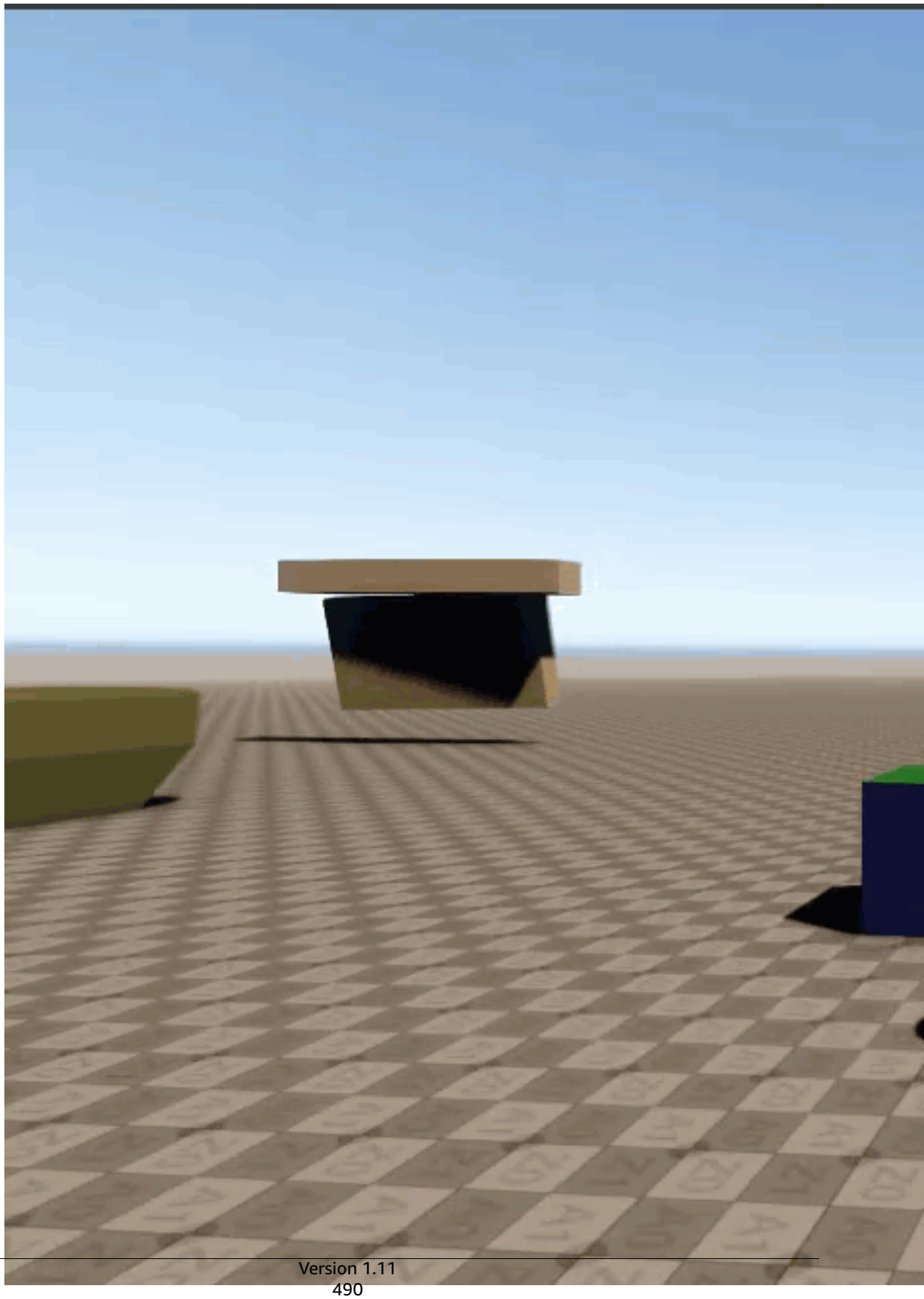
Magnet

The constrained entity and the constraint target move toward each other.

Fixed

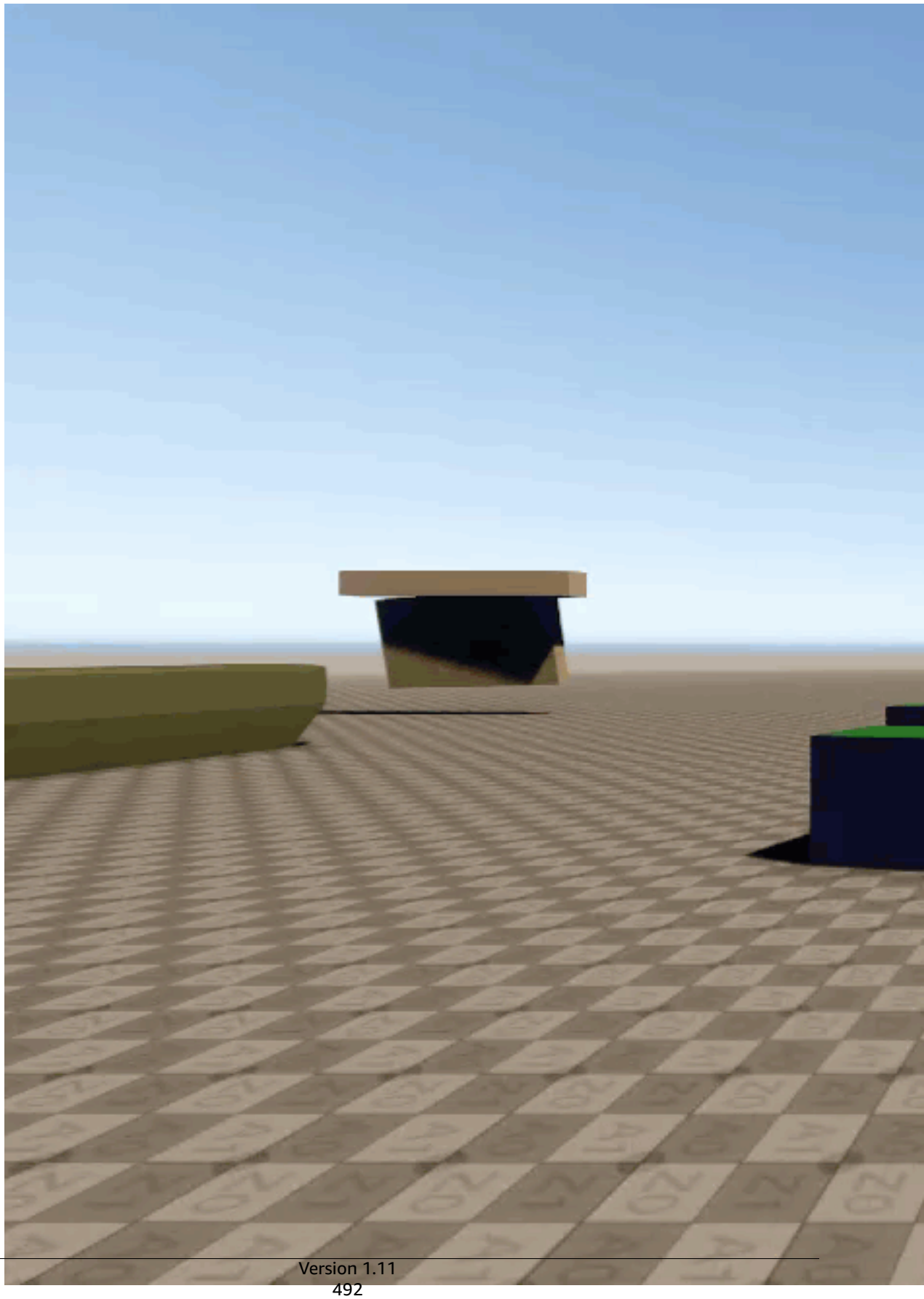
The **Fixed** constraint type must be applied to two rigid body (movable) entities. This constraint type constrains the two entities with no rotation relative to one another (although they can rotate together). The distance and orientation between the constrained entity and the constriant target are preserved.

An example of a fixed constraint is the two sides of a dumbell—with an invisible and intangible bar fixing the two sides together.



Free

The constrained entity 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. For example, if the constrained entity and the constraint target were both statues, they could be moved any distance apart and in any direction, but if one is knocked over, the other mirrors that orientation, wherever it is.



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.

Default value: Hinge

Owning entity

Select either **Self** or **Other entity**.

In the typical setup, select **Other entity**, then select the **Constrained entity**. If you select **Self**, the constrained entity moves about its own axis.

Default value: Self

Target type

Select either **Entity** or **World space**.

Select **Entity** if the constraint target is a rigid body (movable object), and you want the constraint pivot anchored to it. Select **World space** if the constrained entity is to move about the constraint pivot's position in world space.

Default value: Entity

Constrained entity

Click the picker (hand icon) and, in the viewport, select the entity that is to be the constraint owner. This field is not available if you set **Owning entity** to **Self**.

Constraint target

Click the picker (hand icon) and, in the viewport, select the entity that is to be the constraint target. This field is not available if you set **Target type** to **World space**.

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 (such as 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.

Default value: z

Enable on activate

If selected, the constraint is enabled upon activation of the level.

Part ids

If selected, you can specify the following:

- **Owner part id** – Part ID on the constrained entity, such as a weapon, that is to be constrained.

- **Target part id** – Part ID on the constraint target, such as a character's hand, that is to be the constraint target.

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 counterclockwise. 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. 1855) 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) in the *Amazon Lumberyard Developer Guide*.

SetConstraintEntities

Sets which entities are affected by this constraint.

Parameters

AZ::EntityId `owningEntity` – The entity that owns the constraint (constrained entity).

AZ::EntityId `targetEntity` – The target entity. If the target type is world space, this can be `Invalid EntityId`.

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) in the *Amazon Lumberyard Developer Guide*.

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. If the constraint target is world space, this can be `Invalid EntityId`.

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.

```
local 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.tickBusHandler = TickBus.Connect(self)
    self.constraintHandler = ConstraintComponentNotificationBus.Connect(self,
 self.entityId)
    Debug.Log("ConstraintComponent activated for entity: " .. tostring(self.entityId))
end

function constraintexample:OnTick(deltaTime, timePoint)
    self.TransitionCountDown = self.TransitionCountDown - deltaTime
    if (self.TransitionCountDown < 0.0) then
        ConstraintComponentRequestBus.Event.SetConstraintEntities(self.entityId,
 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

return constraintexample
```

Decal

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`**.

Default value: Planar

Material

The decal's material file.

Sort Priority

Sort priority relative to other decals in the system.

Valid values: 0 – 255

Default value: 16

Depth

Projection depth for deferred decals.

Valid values: 0 – 10

Default value: 1

Offset

Allows offsetting the decal relative to the entity's position.

Default value: 0,0,0

Opacity

Degree of transparency for the decal (only available for deferred decals).

Deferred

Shows **No** for **`Planar`** and **`On Terrain Projection Type`**. Shows **Yes** for **`On Terrain and Static Objects`**.

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) in the *Amazon Lumberyard Developer Guide*.

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 decalexample:OnActivate()
    DecalComponentRequestBus.Event.Hide(self.entityId)
    DecalComponentRequestBus.Event.Show(self.entityId)
    DecalComponentRequestBus.Event.SetVisibility(self.entityId, false)
end
```

Environment Probe

Component entity system is in preview release and is subject to change.

Use the **Environment Probe** component to achieve the right visual quality for a space. Environment probes help to determine proper reflections, ambient diffuse values, particle diffuse values, and shadow colors.

The **Environment Probe** component has the following settings:

Visible

Shows the light.

On initially

Specify if the light is on when created.

General Settings

See the following general settings:

Color

The color of the light.

Default value: 0xFFFFFFFF

Diffuse multiplier

Sets the strength of the diffuse color.

Default value: 1

Specular multiplier

Sets the strength of the specular brightness.

Default value: 1

Environment Probe Settings

See the following environment probe settings:

Area dimensions

The width, height, and depth in meters of the environment probe's effective area.

Sort priority

Priority number for probe rendering. Lower priority numbers (for example, **0** or **1**) are rendered on top of the higher priority numbers (for example, **100**).

Default value: 0

Resolution

Cubemap resolution in pixels that is generated by this probe.

Default value: 256

Box projected

Enables box projection. Does not require regeneration of the cubemap.

Box height

Height of the box projection.

Box length

Length of the box projection.

Box width

Width of the box projection.

Attenuation falloff

The value that the light begins to falloff. For example, a value or 1.0 means the light starts to fall off at 1.0 meter.

Specify a value from **`0.0`** to **`1.0`**.

Options

See the following options:

View distance multiplier

Adjusts the maximum view distance. For example, **`1.0`** uses the default and **`1.1`** is 10% farther than the default.

Default value: 1

Minimum spec

Minimum specular at which the light is enabled.

Default value: Low

Voxel GI mode

Mode for light interaction with voxel global illumination (GI).

Default value: None

Ignore vis areas

Light ignores vis areas.

Indoor only

Light is only rendered indoors.

Affects this area only

Light affects only the immediate area.

Volumetric fog only

Light affects only volumetric fog.

Volumetric fog

Light affects volumetric fog and surrounding area.

Animation

See the following animation settings:

Style

Type a number to specify a preset light animation curve to play as defined in the `Light.cfx` file. Valid values are **`0`** to **`48`**. You can also use values **`40`** to **`48`** for testing and debugging.

Default value: 0

Speed

Multiple of the base animation rate. For example, a value of **`2.0`** makes an animation play twice as fast.

Default value: 1

Phase

Animation start offset from **`0`** to **`1`**. A value of **`0.1`** is 10% into the animation. For example, you can use this setting to prevent lights in the same scene, with the same animation, from being animated in unison.

Default value: 0

Cubemap Generation

See the following cubemap generation settings:

Cubemap

Click **Generate** to generate or regenerate a cubemap for this probe. The button is enabled only when **Use customized cubemap** is deselected.

View cubemap

Displays a preview of the cubemap in the viewport. You can only select this option when a cubemap exists.

Use customized cubemap

If selected, you can specify a custom cubemap for **Cubemap asset**. If deselected, a cubemap is generated.

Cubemap asset

File path for the cubemap asset.

Additional Resources

For more information about the **Environment Probe** component, see the following:

- Getting Started: Using Lighting Tutorial (text)
- Getting Started: Using Lighting Tutorial (video)

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).

All light components share common Ebus functions. For more information, see Light Components EBus Request Bus Interface (p. 452).

Flow Graph

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 entitiy'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.

A few things to note about component entities and 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.

To add a flow graph component to an 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.

High Quality Shadow

Component entity system is in preview release and is subject to change.

You can use the **High Quality Shadow** component to give an entity its own shadow map. This can be enabled or disabled in scripts with the **HighQualityShadowComponentRequestBus**. For more information, see Shadow Nodes (p. 746).

High Quality Shadow Component Properties

The **High Quality Shadow** component has the following properties:

Enabled

Lets you enable the shadow map

Const Bias

Entity avoids the self shadow artifacts

Slope Bias

Can be used to avoid self shadow artifacts

Jitter

Filter kernel size, directly affects shadow softness

Bounding Box Scale

Scale factor for bounding box of the selected entity. This is useful in case the engine bounding box is too small or too large.

Shadow Map Size

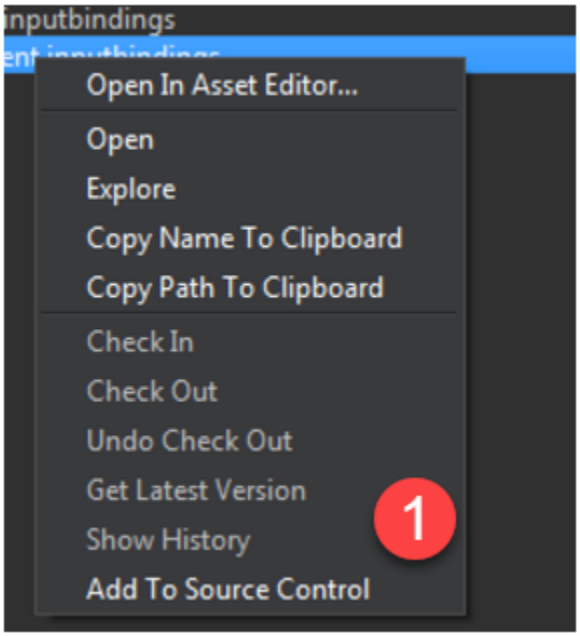
Size of the custom shadow map. Value is automatically rounded to the next power of two.

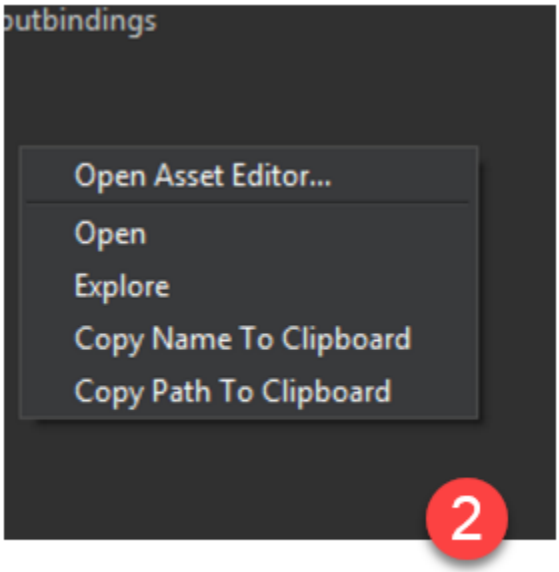
Input

Component entity system is in preview release and is subject to change.

The **Input** 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 Asset Browser (p. 179). Select an option from the asset context menu (1). The asset context menu (1) presents a different set of options than the general context menu (2).





Input Properties

The **Input** 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 browse (**...**) icon to select an `.inputbindings` file.

Input contexts

Type a string to enter valid context(s) for this input binding. The default context is an empty string.

> **Note**
> "" is the default context. It is active whenever explicitly pushed to the top of the input context stack, or when the input context stack is empty. Note that input ebus messages are not generated while the console is open.

Creating an Input to Event Binding Asset

Using the **File Browser**, you can create a new asset by clicking **File**, **Create New Asset**, **Input to Event Bindings 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 **Tools**, **File Browser**.
2. Right-click on an asset, and then click **Open Asset Editor**.

   You can also open the **Asset Editor** from the **Entity Inspector** (p. 441) while viewing an **Input** component by clicking the edit icon  .
3. Click **File**, **Create New Asset**, **Input to Event Binding Asset**.

Creating Input Event Groups

An **Input Event Bindings** file can have zero (0) or more **Input Event Groups**.

To add an input event group

1. In the **Asset Editor**, after creating a new input event binding (p. 505), click the plus sign (**+**) next to **Input Event Groups**.
2. In the **Event Name** box, type a name for your event.

Creating 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.

Input sub components are objects that have the same lifetime as components and must override Activate and Deactivate.

```
//////////////////////////////////////////////////////////////////////////////
// IInputSubComponent
void Activate(const AZ::InputEventNotificationId& channelId) override;
void Deactivate(const AZ::InputEventNotificationId& channelId) override;
```

`GameplayNotificationBus` handlers

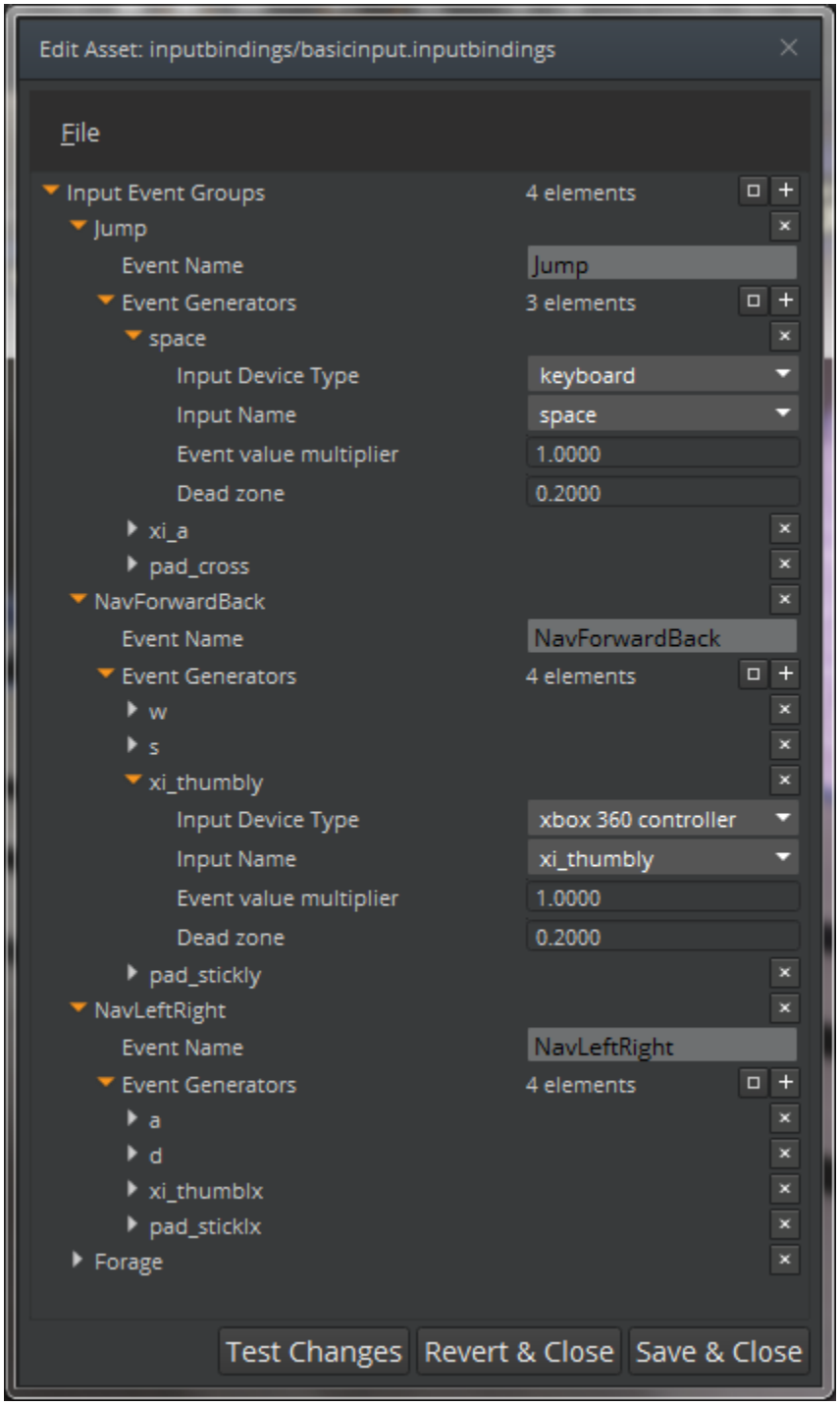
These handlers aggregate one more `GameplayActionEvents` into a single output `GameplayActionEvent`. They do not listen for raw input like the `InputNotificationBus` handlers. The following events are `ActionNotificationBus` handlers:

- **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.



To edit the `.inputbindings` file

1. Open the **File Browser** by clicking **Tools**, **File Browser**.
2. Right-click the `.inputbindings` file you want to edit. From the context menu, click **Edit**.

Single Event to Action Properties

You can use the **Held**, **Pressed**, and **Released** scripts to transform processed input events into game play events.

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.
- **Dead zone** – An input is considered pressed outside of this threshold and released within it. 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.

The event generators **Held** and **Analog** also have the following unique properties:

Held

- **Duration to hold** – The duration of seconds that the input must be held.
- **Success pulse interval** – The interval between successful messages after holding for the duration specified in the `Duration to hold` setting.
- **Invoke type** – Select when you want the events to be invoked:
  - Every duration
  - Once per release
  - Every frame after duration

Analog

- **Send continuous updates** – If selected, updates are sent continuously. If deselected, sends a message only when the analog value has changed.

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

- **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.
- **Dead zone length** – A threshold for vector length below which an event is not generated.

Event Notification Bus

Use the following notification functions with the event noficiation bus interface to communicate with other components of your game.

For more information about using the event bus (EBus) interface, see Event Bus (EBus) in the *Amazon Lumberyard Developer Guide*.

InputEventNotificationBus

`OnPressed`

type: float

Event sent when an input surpasses the threshold.

`OnHeld`

type: float

Event sent when an input continues to surpass the threshold.

`OnReleased`

type: float

Event sent when an input no longer surpasses the threshold.

The following is an example script.

```
local held =
{
    Properties =
    {
        IncomingInputEventName = "",
        OutgoingGameplayEventName = "",
    },
}

function held:OnActivate()
    local inputBusId = InputEventNotificationId(self.Properties.IncomingInputEventName)
    self.inputBus = InputEventNotificationBus.Connect(self, inputBusId)
end


function held:OnHeld(floatValue)
    GameplayNotificationBus.Event.OnEventUpdating(GameplayNotificationId(self.entityId,
 self.Properties.OutgoingGameplayEventName), floatValue)
end

function held:OnReleased(floatValue)
    GameplayNotificationBus.Event.OnEventEnd(GameplayNotificationId(self.entityId,
 self.Properties.OutgoingGameplayEventName), floatValue)
end

function held:OnDeactivate()
    self.inputBus:Disconnect()
end

return held
```

Input Request Bus

Use the following functions with the input request bus interface to communicate with other components of your game.

For more information about using the event bus (EBus) interface, see Event Bus (EBus) in the *Amazon Lumberyard Developer Guide*.

InputRequestBus

`PushContext`

Pushes a new context onto the stack, which then becomes the active context.

`PopContext`

Removes the top context from the input context stack.

`PopAllContexts`

Clears the context stack, and the active context becomes "".

`GetCurrentContext`

Returns the context at the top of the stack. If the stack is empty, returns "".

The following is an example script.

```
local foo
{
    Properties =
    {
        Context {default = "", description = "A context to push onto the input stack. Like
 console, or UI, or turret, under water, etc."},
    }
}

function foo:OnActivate()
    -- by default the context is blank ""
    InputRequestBus.Broadcast.PushContext(self.Properties.Context) -- context stack is now
 1)user defined property
    InputRequestBus.Broadcast.PushContext("under water") -- context stack is now 1)user
 defined property, 2) "under water"
    Debug.Log(InputRequestBus.Broadcast.GetCurrentContext()) -- prints "under water"
    InputRequestBus.Broadcast.PushContext("for fun") -- context stack is now 1)use defined
 property, 2) "under water", 3) "for fun"
    InputRequestBus.Broadcast.PopContext() -- context stack is now 1)user defined property,
 2) "under water"
    InputRequestBus.Broadcast.PopAllContexts() context stack is now empty
end

return foo
```

Lens Flare

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.

Library

Select a lens flare library that has been authored by the **Lens Flare** editor.

Lens flare

Select a lens flare from the available flares in the lens flare library.

Flare Settings

Minimum spec

The minimum spec at which this lens flare is enabled.

Default: Low

FOV

The field of view (FOV) in degrees around the lens flare. Use `360` degrees to make the lens flare visible from all angles.

Default: 360

Size

The size of the lens flare.

Attach to sun

If selected, attaches the lens flare to the sun (as opposed to attaching to the entity).

Ignore vis areas

If selected, lens flare ignores vis areas.

Indoor only

If selected, lens flare is rendered indoors only.

On initially

If selected, the lens flare is on when the scene starts.

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.

Tint [alpha]

Alpha value that sets the flare's transparency.

Brightness

Brightness of the lens flare.

Animation

Sync with light

If selected, uses the animation settings of the provided light. Select the light entity in the **Light** setting.

Light

This setting appears when the **Sync with light** setting is selected. Use the picker (hand icon) to select the light component you want to sync animation settings with.

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.

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) in the *Amazon Lumberyard Developer Guide*.

- **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) in the *Amazon Lumberyard Developer Guide*.

- **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()
    LensFlareComponentRequestBus.Event.SetLensFlareState(self.entityId,
 LensFlareComponentState.Off)
    LensFlareComponentRequestBus.Event.TurnOnLensFlare(self.entityId)
```

```
    LensFlareComponentRequestBus.Event.TurnOffLensFlare(self.entityId)
    LensFlareComponentRequestBus.Event.ToggleLensFlare(self.entityId)
end
```

Lua Script

You can use the **Lua Script** component to add arbitrary Lua logic to an entity in the form of a Lua script. For information on the **Lua Script** component and writing Lua scripts, see Writing Lua Scripts for the Component Entity System in the Amazon Lumberyard Developer Guide.

Mannequin

> Component entity system is in preview release and is subject to change.

The **Mannequin** component animates a component entity using the Mannequin System (p. 325). This component works in conjunction with the **Mannequin Scope Context** (p. 520) 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.

To see how Lua scripting works with the Mannequin component, see Controllable Chicken sample project located at `dev\SamplesProject\Levels\Component_Tests\Controllable_Chicken`.

Mannequin Component Properties

The **Mannequin** component has the following property:

Controller Definition

Path to the controller definition file (p. 325) 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) in the *Amazon Lumberyard Developer Guide*.

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 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 entity system is in preview release and is subject to change.

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. 513) 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.

Mesh

Component entity system is in preview release and is subject to change.

The **Mesh** component is the primary tool for adding visual geometry to entities. This component also features key controls and options for using the engine's basic rendering features. The supported geometry type is meshes (`.cgf`).

Mesh Component Properties

The **Mesh** component has the following properties:

Visible

The entity is visible.

Mesh asset

Asset file for the mesh entity.

Material override

Specifies an override material.

Options

The **Mesh** has the following options.

Opacity

The entity's degree of transparency.

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. For example, `1.1` extends the default by 10%.

LOD distance ratio

Sets the level of detail (LOD) ratio over distance.

Cast shadows

Casts shadow maps.

Use VisAreas

Allows VisAreas to control the component's visibility.

Advanced

The **Mesh** component has the following advanced options.

> **Note**
> A mesh is static with the static transform enabled. A static mesh can move or deform only when the **Receive Wind** or **Deformable Mesh** options are enabled.

A static mesh displays more options than a dynamic mesh. Note that dynamic meshes do not affect nav meshes. You can use static objects for more optimal paths during rendering and processing. We recommend that you create static meshes whenever possible. For more information about the static transform, see Transform Component Properties (p. 564).

Rain occluder

The entity blocks or stops dynamic raindrops.

Affect dynamic water

The entity generates ripples in dynamic water.

Receive wind

The entity is affected by wind.

Accept decals

The entity can receive decals.

Affect navmesh

The entity affects navmesh generation.

Visibility occluder

The entity can block visibility of other objects.

Deformable mesh

If selected, allows static mesh assets to deform that have specific dynamic data. For example, you can have a mesh deform asset that is stationary.

> **Note**
> If you select the **Receive wind** or **Deformable mesh** options, the entity transform remains static, but the mesh is considered dynamic.

Mesh Collider

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. 521) 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.

Motion Parameter Smoothing

With the **Motion Parameter Smoothing** component, you can configure animation blend parameter behavior (for blend spaces) for a specified character instance. If you add this component to an animated entity, the animation system automatically applies the settings and requires no additional interaction or setup.

The **Motion Parameter Smoothing** component has the following properties:

Ground Angle Time

Time in seconds over which the applied ground angle blend parameter converges on the physically detected ground angle.

Travel Angle Time

Time over which the applied travel angle parameter converges on the actual physical value.

Travel Distance Time

Time over which the applied travel distance-per-frame parameter converges on the actual physical value.

Travel Speed Time

Time over which the applied travel speed parameter converges on the actual physical value.

Turn Angle Time

Time over which the applied turn angle-per-frame parameter converges on the actual physical value.

Turn Speed Time

Time over which the applied turn speed parameter converges on the actual physical value.

Navigation

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 movement requests for each frame 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, all interested parties are informed. The requester (or other interested parties) must then tell the component to commit to that path and move its entity.

This component is not responsible for assessing the tactical viability of any given pathfinding or pathfollowing request. 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 that includes the potential for some additional validation before the path is actually traversed. Consider this to be more of a screening opportunity than a true tactical decision-making point; its main value is 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

Set the entity type for navigation purposes. Use this type to select which navigation mesh (p. 1857) the entity follows in a scenario where there are different navigation meshes for larger vehicles and smaller humanoid bots.

Default value: MediumSizedCharacters

AgentRadius

Set the entity radius for navigation purposes. Independent of physics or any other collision concerns, the pathfinder uses this value for moving around an area with obstacles while cutting corners.

Default value: 4

LookAheadDistance

Set the distance between the points that an entity walks over while following a given path.

Default value: 0.1

Arrival Distance Threshold

Set the minimum distance from an end point when an entity's movement stops and is considered complete.

Default value: 0.25

Repath Threshold

Set the minimum distance from the previously known location before an entity's new path is calculated.

Default value: 1

Move Physically

Apply movement through physics. You must also place a **Character Physics** component on the entity in order for this functionality to work. We recommend this method to apply movement to a character during pathfollowing because it handles uneven terrain and character limitations properly. If deselected, movement will apply directly to the entity's transform.

Default value: true

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) in the *Amazon Lumberyard Developer Guide*.

FindPath

Find 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

Create a pathfinding request to navigate toward the specified entity.

Parameters

`EntityId` – ID of the entity toward which you want to navigate.

Return

A unique identifier for the pathfinding request.

Scriptable

Yes

Stop

Stop all pathfinding operations for the provided `requestId`. Use the ID 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) in the *Amazon Lumberyard Developer Guide*.

OnSearchingForPath

Indicate that the pathfinding request has been submitted to the navigation system.

Parameters

`requestId` – ID of the path search request.

Return

None

Scriptable

Yes

OnPathFound

Indicate 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 or not to traverse this path.

Scriptable

No

OnTraversalStarted

Indicate that traversal for the indicated request has started.

Parameters

`requestId` – ID of the request for which traversal has started.

Return

None

Scriptable

Yes

OnTraversalInProgress

Indicate 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

Indicate that traversal for the indicated request has completed successfully.

Parameters

`requestId` – ID of the request for which traversal has finished.

`distanceRemaining` – The remaining distance in the path.

Return

None

Scriptable

Yes

OnTraversalCancelled

Indicate 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

The following is an example of Lua script that moves an entity toward another entity.

```
local navigationmoversimplified =
{
    -- Adds properties to the Entity Inspector for easy setup of initial values.
    Properties =
    {
        -- This is the target toward which you'll move
        MoveToEntity = {default=EntityId(), description="Entity to move to."},
    }
}

function navigationmoversimplified:OnActivate()
    -- Move!
    self.requestId = NavigationComponentRequestBus.Event.FindPathToEntity(self.entityId,
 self.Properties.MoveToEntity)
end

return navigationmoversimplified
```

Network Binding

Component entity system is in preview release and is subject to change.

The **Network Binding** component marks an entity as able to be replicated across the network. It interfaces with the NetBindable interface in order to call the functions specified there.

Particle

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.

The particle component uses effects that come from an XML **Particle effect library**. If no emitters are listed, you can open the **Particle Editor** (p. 1272) to create emitters.

Particle Component Properties

The Particle component has the following properties:

Source

Visible

If selected, renders the emitter.

Enable

If selected, activates the particle effect.

Particle effect library

Select the particle effect library.

Emitters

Select an emitter from the list after specifying a particle effect library.

Spawn Settings

Color tint

Select the color of the effect using the color swatch.

Pre-Roll

If selected, sets the emitter to behave as though it has been running indefinitely.

Count scale

Set the multiplier for the particle count.

Valid values: 0 – 1000

Default value: 1

Time scale

Set the multiplier for the emitter time evolution.

Valid values: 0 – 1000

Default value: 1

Pulse period

Set the frequency at which to restart the emitter.

Default value: 1

Global size scale

Set the multiplier for all effects sizes.

Valid values: 1 – 100

Default value: 1

Particle size scale X

Set the multiplier for the particle size on the x-axis.

Valid values: 1 – 100

Particle size scale Y

Set the multiplier for the particle size on the y-axis.

Valid values: 1 – 100

Speed scale

Set the multiplier for the particle emission speed.

Valid values: 1 – 1000

Lifetime strength

Set the parameter strength over the emitter lifetime. A negative value applies the strength over the emitter lifetime curve. A zero value removes the curve effect. A positive value applies the effect uniformly, based on the strength over the lifetime curve.

Valid values: -1 – 1

Ignore rotation

If selected, ignores the entity's rotation.

Not attached

If selected, ignores the entity's position. The emitter does not follow its entity.

Register by bounding box

If selected, uses the bounding box instead of position to register in visarea.

Use LOD

If cleared, ignores the emitter level of detail (LOD).

Audio Settings

Enable audio

If selected, enables audio.

Audio RTPC

Select the audio RTPC that the 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) in the *Amazon Lumberyard Developer Guide*.

- **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()
    ParticleComponentRequestBus.Event.Show(self.entityId)
    ParticleComponentRequestBus.Event.Hide(self.entityId)
    ParticleComponentRequestBus.Event.SetVisibility(self.entityId, false)
end
```

Point Light

Component entity system is in preview release and is subject to change.

Use the **Point Light** component on an entity to create a point of light.

The **Point Light** component has the following settings:

Visible

Shows the light.

On initially

Specify if the light is on when created.

General Settings

See the following general settings:

Color

The color of the light.

Default value: 0xFFFFFFFF

Diffuse multiplier

Sets the strength of the diffuse color.

Default value: 1

Specular multiplier

Sets the strength of the specular brightness.

Default value: 1

Ambient

Light acts as a multiplier for cubemap values.

Point Light Settings

See the following point light settings:

Max Distance

Maximum distance, in a radius, from which this light is visible.

Default value: 2

Attenuation bulb size

Radius in meters before light falloff begins.

Default value: 0

Options

See the following options:

View distance multiplier

Adjusts the maximum view distance. For example, **`1.0`** uses the default and **`1.1`** is 10% farther than the default.

Default value: 1

Minimum spec

The minimum specular at which this light is enabled.

Default value: Low

Cast shadow spec

The minimum specular at which shadows are cast.

Default value: Never

Voxel GI mode

Mode for light interaction with voxel global illumination (GI).

Ignore vis areas

Light ignores vis areas.

Indoor only

Light is only rendered indoors.

Affects this area only

Light affects only the immediate area.

Volumetric fog only

Light affects only volumetric fog.

Volumetric fog

Light affects volumetric fog and surrounding area.

Animation

See the following animation settings:

Style

Type a number to specify a preset light animation curve to play as defined in the `Light.cfx` file. Valid values are **`0`** to **`48`**. You can also use values **`40`** to **`48`** for testing and debugging.

Default value: 0

Speed

Multiple of the base animation rate. For example, a value of **`2.0`** makes an animation play twice as fast.

Default value: 1

Phase

Animation start offset from **`0`** to **`1`**. A value of **`0.1`** is 10% into the animation. For example, you can use this setting to prevent lights in the same scene, with the same animation, from being animated in unison.

Default value: 0

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).

All light components share common Ebus functions. For more information, see Light Components EBus Request Bus Interface (p. 452).

Projector Light

Component entity system is in preview release and is subject to change.

Use the **Projector Light** component on an entity to project a light.

The **Projector Light** component has the following settings:

Visible

Shows the light.

On initially

Specify if the light is on when created.

General Settings

See the following general settings:

Color

The color of the light.

Default value: 0xFFFFFFFF

Diffuse multiplier

Sets the strength of the diffuse color.

Default value: 1

Specular multiplier

Sets the strength of the specular brightness.

Default value: 1

Ambient

If selected, light acts as a multiplier for cubemap values.

Project Light Settings

See the following point light settings:

Max Distance

Maximum distance that the projector light extends, in meters.

Default value: 5

Attenuation bulb size

Radius before light begins to fade, in meters.

Default value: 0

FOV

Projector light's field of view (FOV), in degrees.

Default value: 90

Near plane

Distance of the near plane to the projector light, in meters. Objects behind this plane are not affected by the projector light.

Texture

Projector light's texture file. Without a texture, light does not shine.

Material

Projector light's material file.

Options

See the following options:

View distance multiplier

Adjusts the maximum view distance. For example, **`1.0`** uses the default and **`1.1`** is 10% farther than the default.

Default value: 1

Minimum spec

The minimum specular at which this light is enabled.

Default value: Low

Cast shadow spec

The minimum specular at which shadows are cast.

Default value: Never

Voxel GI mode

Mode for light interaction with voxel global illumination (GI).

Ignore vis areas

Light ignores vis areas.

Indoor only

Light is only rendered indoors.

Affects this area only

Light affects only the immediate area.

Volumetric fog only

Light affects only volumetric fog.

Volumetric fog

Light affects volumetric fog and surrounding area.

Animation

See the following animation settings:

Style

Type a number to specify a preset light animation curve to play as defined in the `Light.cfx` file. Valid values are **`0`** to **`48`**. You can also use values **`40`** to **`48`** for testing and debugging.

Default value: 0

Speed

Multiple of the base animation rate. For example, a value of **`2.0`** causes an animation to play twice as fast.

Default value: 1

Phase

Animation start offset from **`0`** to **`1`**. A value of **`0.1`** is 10% into the animation.

For example, you can use this setting to prevent lights in the same scene, with the same animation, from being animated in unison.

Default value: 0

Additional Resources

- For more information about the **Projector Light** component, see the following:
  - Getting Started: Using Lighting Tutorial (text)
  - Getting Started: Using Lighting Tutorial (video)

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).

All light components share common Ebus functions. For more information, see Light Components EBus Request Bus Interface (p. 452).

Primitive Collider

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 entity system is in preview release and is subject to change.

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, starts the entity 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, collides the entity 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) in the *Amazon Lumberyard Developer Guide*.

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**.

```
RagdollPhysicsRequestBus.Event.EnterRagdoll(self.entityId)
RagdollPhysicsRequestBus.Event.ExitRagdoll(self.entityId)
```

Rigid Body Physics

Use the **Rigid Body Physics** component to represent solid objects that move realistically when touched, such as a ball. You can use a rigid body physics component with a primitive collider (p. 534) or mesh collider (p. 522) component to provide geometry for a physical entity.

The **Rigid Body Physics** component has the following properties:

Enabled Initially

If selected, the entity is initially enabled in the physics simulation

Specify mass or density

Whether total mass is specified, or calculated at spawn time based on density and volume.

Density (kg/cubic meter)

Mass in kg per cubic meter of the mesh's volume. Total mass of entity is calculated at spawn. Water's density is 1000.

This property is only available when: **Specify mass or density** is set to **`Density`**.

At rest initially

If selected, entity remains at rest until it is agitated. If deselected, entity falls after it spawns.

Collision response

If selected, the entity's simulation is affected normally as a result of collisions with other bodies. If deselected, collision events are reported, but the entity's simulation is not affected by simulations.

Interacts with triggers

Whether the entity will interact with the **Trigger Area** (p. 577) component.

Record collisions

Whether `OnCollision` events will be sent out on the entity's `PhysicsComponentNotificationBus`.

Number of collisions

Maximum number of collisions to record and report per frame.

Simulation

Damping

Uniform damping value applied to object's movement.

Minimum energy

The energy threshold under which the object will go to sleep.

Buoyancy

Water damping

Uniform damping value applied while in water.

Water density

Water density strength.

Water resistance

Water resistance strength.

Ebus Request Bus Interface

Use the following request functions with the `PhysicsComponentRequestBus` interface to communicate with other components of your game.

For more information about using the Event Bus (EBus) interface, see Event Bus (EBus) in the *Amazon Lumberyard Developer Guide*.

EnablePhysics

Makes the entity a participant in the physics simulation.

Parameters

None

Return

None

Scriptable

Yes

DisablePhysics

Stops the entity from participating in the physics simulation.

Parameters

None

Return

None

Scriptable

Yes

IsPhysicsEnabled

Queries whether the entity's physics component has been enabled.

Parameters

None

Return

Bool – whether physics is enabled

Scriptable

Yes

AddImpulse

Adds an impulse to the entity.

Parameters

Vector3 – an impulse in world space

Return

None

Scriptable

Yes

AddImpulseAtPoint

Adds an impulse to the entity at a specified point.

Parameters

Vector3 – an impulse in world space

Vector3 – world space point at which impulse is applied. Specify a zero Vector3 to apply at center of mass.

Return

None

Scriptable

Yes

AddAngularImpulse

Adds an angular impulse to the entity.

Parameters

Vector3 – angular impulse to add

Return

None

Scriptable

Yes

AddAngularImpulseAtPoint

Adds an angular impulse to the entity at a specified point.

Parameters

Vector3 – an angular impulse in world space

Vector3 – world space pivot around which to apply the impulse. Specify a zero Vector3 to apply around center of mass.

Return

None

Scriptable

Yes

GetVelocity

Gets the velocity of the entity.

Parameters

None

Return

Vector3 – velocity of the entity to set

Scriptable

Yes

SetVelocity

Sets the velocity of the entity.

Parameters

Vector3 – world space velocity to set directly on the entity

Return

None

Scriptable

Yes

GetAcceleration

Gets the acceleration of the entity.

Parameters

None

Return

Vector3 – acceleration of the entity

Scriptable

Yes

GetAngularVelocity

Gets the angular velocity of the entity.

Parameters

None

Return

Vector3 – angular velocity of the entity

Scriptable

Yes

SetAngularVelocity

Sets the angular velocity of the entity.

Parameters

Vector3 – world-space angular velocity to directly set on the entity

Return

None

Scriptable

Yes

GetAngularAcceleration

Gets the angular acceleration of the entity.

Parameters

None

Return

Vector3 – world space angular velocity of the entity

Scriptable

Yes

GetMass

Gets the total mass (kg) of the entity.

Parameters

None

Return

float – the total mass of the entity

Scriptable

Yes

SetMass

Sets the total mass (kg) of the entity.

Parameters

float – the total mass to set

Return

None

Scriptable

Yes

GetDensity

Gets the density of the entity, in kg/m3, (Volume / Mass). Water 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).

Parameters

None

Return

float – density of the entity

Scriptable

Yes

SetDensity

Sets the density of the entity, in kg/m3.

Parameters

float – density of the entity to set

Return

None

Scriptable

Yes

GetDamping

Gets the simulation damping coefficient of the entity.

Parameters

None

Return

float – damping coefficient

Scriptable

Yes

SetDamping

Sets entity's damping coefficient for simulation.

Parameters

float - entity's damping coefficient for simulation

Return

None

Scriptable

Yes

GetMinEnergy

Gets the minimum kinetic energy below which the entity should fall asleep.

Parameters

None

Return

float – minimum kinetic energy

Scriptable

Yes

SetMinEnergy

Set the entity's minimum kinetic energy below which the entity should fall asleep

Parameters

float – minimum kinetic energy to set

Return

float – minimum kinetic energy

Scriptable

Yes

GetWaterDamping

Get the entity's uniform damping coefficient while submerged in water.

Parameters

float – minimum kinetic energy to set

Return

None

Scriptable

Yes

SetWaterDamping

Sets the entity's uniform damping coefficient while submerged in water. A cheaper alternative/addition to water resistance (applies uniform damping when in water). 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. 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.

Parameters

float – damping cofficient to set

Return

None

Scriptable

Yes

GetWaterDensity

Gets the entity's density applied when it interacts with water.

Parameters

None

Return

float – entity's density while in water

Scriptable

Yes

SetWaterDensity

Sets the entity's density applied when it interacts with water. This 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 will sink easier. (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 will be less than submerged_volume 1000 (with 1000 being the actual density underwater). Decreasing per object effective water density will allow such objects to sink (as they would in reality) while still having large-volume physical geometry. Important note : If you are changing the default value of 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.

Parameters

float – entity's density while in water

Return

None

Scriptable

Yes

GetWaterResistance

Gets the entity's medium resistance while it's in water.

Parameters

None

Return

float – entity's medium resistance while in water

Scriptable

Yes

SetWaterResistance

Sets the entity's medium resistance while it's in water. 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). (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. Although water resistance is not too visible on a general object, setting it to a suitable value will prevent very light objects from jumping in the water, and water flow will affect things more realistically. 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.

Parameters

float – entity's medium resistance while in water

Return

None

Scriptable

Yes

Script Canvas

Component entity system is in preview release and is subject to change.

Use the **Script Canvas** component to add a script to an entity.

Script Canvas Component Properties

The **Script Canvas** component has the following properties:

Script Canvas Asset

Specify the script file that you want to add to the component.

Shapes: Cylinder, Capsule, Box, Sphere, Compound

Component entity system is in preview release and is subject to change.

The **Shape** components provide generic shape facilities for components that use shapes. The shape component helps to define the following:

- Trigger (p. 577) Volumes

  Use shapes as volumes to specify triggering bounds.
- Collision (p. 522) Volumes

  Use shapes as volumes to specify collider bounds.
- Audio Area Ambiances (p. 457)

  Uses shapes as volumes in which a reverb is applied.
- Audio Areas (p. 462)

  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.
- **Compound Shape** – Use picker (hand icon) to select two or more shapes to combine. With the compound shape component, you can combine shapes to create a complex object that you can use to generate a physics collider or a trigger shape, or any other application of shapes. The individual shapes may or may not be children of the entity with the compound shape component. To move the shapes independently of the entity with compound shape, do not set them as child entities. To move them together with the compound shape, set them as child entities. Compound Shapes service the full shape component bus; however, note that each individual shape added increases the cost of requests such as `IsPointInside`.

ShapeComponentRequestsBus

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) in the *Amazon Lumberyard Developer Guide*.

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 AABB 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

DistanceFromPoint

Returns the minimum distance between a specified point and the shape.

Parameters

point `Vector3` – Point from which to calculate distance

Return

type: float

Distance from point to shape

Scriptable

Yes

DistanceSquaredFromPoint

Returns the minimum squared distance between a specified point and the shape.

Parameters

point `Vector3` – Point from which to calculate square distance

Return

type: float

Square distance from point to shape

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

CompoundShapeComponentRequestsBus

Fetches the configuration of the CompoundShape.

Name

GetCompoundShapeConfiguration

Parameters

None

Return

`CompoundShapeConfiguration` object which provides access to the compound shape configuration

Scriptable

Yes

ShapeComponentNotificationsBus

OnShapeChanged

Informs listeners that the shape component has been updated, meaning that the shape was modified.

Name

OnShapeChanged

Parameters

`ShapeChangeReasons` Indicates whether the shape has to be updated due to either a transform change or a shape parameters change.

Return

void

Scriptable

Yes

Simple Animation

Component entity system is in preview release and is subject to change.

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.

Animate root

Enables animation-driven root motion during playback of this animation.

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) in the *Amazon Lumberyard Developer Guide*.

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.

```
local startanimation =
{
    Properties =
    {
```

```
    }
}

function startanimation:OnActivate()
    -- 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)
    SimpleAnimationComponentRequestBus.Event.StartAnimation(self.entityId, animInfo)
end

return startanimatio
```

The following is an example of script using the **Request Bus Interface**.

```
local 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." },
        Test1 = { Vector3(1,2,3), Vector3(1,2,3) },
    },
}

function chickenanimcontroller:OnActivate()

    self.FlapCountdown = 0.0;

    -- For handling tick events.
    self.tickBusHandler = TickBus.Connect(self);

    -- 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);
    SimpleAnimationComponentRequestBus.Event.StartAnimation(self.entityId, animInfo);

end

function chickenanimcontroller:OnTick(deltaTime, timePoint)

    -- Get current transform
    local tm = TransformBus.Event.GetWorldTM(self.entityId);

    -- 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);
        SimpleAnimationComponentRequestBus.Event.StartAnimation(self.entityId, animInfo);
        self.FlapCountdown = self.Properties.FlapInterval;
        --Debug.Log("Played the flap");
    end

    -- 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.
    TransformBus.Event.SetWorldTM(self.entityId, tm);

end

function chickenanimcontroller:OnDeactivate()
    self.tickBusHandler:Disconnect();
end

return chickenanimcontroller;
```

Simple Motion

Component entity system is in preview release and is subject to change.

You can use the **Simple Motion** component to play a motion without using an animation graph. Add this component to the Actor (p. 448) component to use a single motion for your actor. For complex motions, see the AnimGraph (p. 449) component.

Simple Motion Component Properties

The **Simple Motion** component has the following properties:

Motion

Lets you select the single motion that you want the actor to play.

Loop motion

Causes the animation to run continuously.

Retarget motion

Allows motion that was created with an actor that was configured with specific bone lengths to be played on another actor with different bone lengths. When applied, the motion does not affect bone lengths. The skeleton must follow the same hierarchy and the bone names must be identical to work properly.

Reverse motion

Causes the animation to run in reverse.

Mirror motion

Causes the animation of the character's body parts to be mirrored. For example, if the actor kicks with the right leg while the left leg is planted, the mirror effect causes the left leg to kick while the right leg is planted.

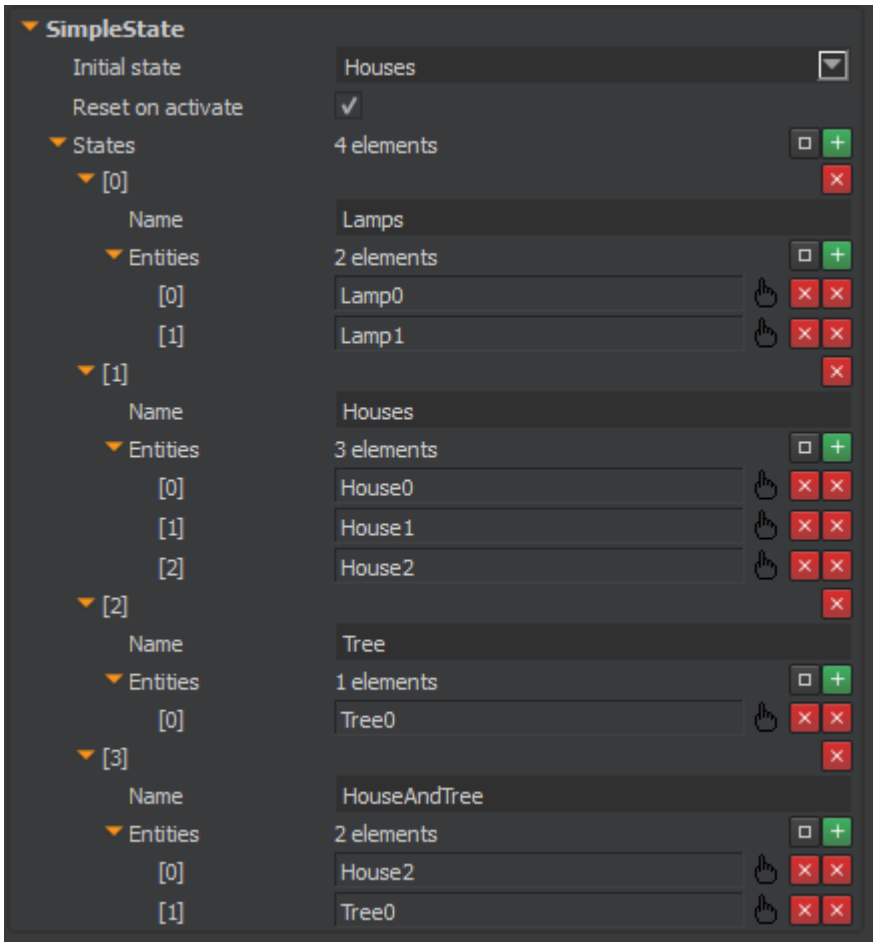
Play speed

Changes the rate at which the motion is played.

Simple State

Component entity system is in preview release and is subject to change.

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.

Enitities

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) in the *Amazon Lumberyard Developer Guide*.

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) in the *Amazon Lumberyard Developer Guide*.

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**.

```
local simplestateexample =
{
    Properties =
    {
        TransitionInterval = 1.0,
        States = {"Houses", "Nope", "Lamps", "Tree", "HouseAndTree", "NoState"},
    }
}

function simplestateexample:OnActivate()
```

```
    self.TransitionCountDown = self.Properties.TransitionInterval
    self.StateIdx = 0
    self.tickBusHandler = TickBus.Connect(self)
    self.stateChangedHandler = SimpleStateComponentNotificationBus.Connect(self,
 self.entityId)

    Debug.Log("SimpleStateComponent activated for entity: " .. tostring(self.entityId.id))
end

function simplestateexample:OnDeactivate()
    self.tickBusHandler:Disconnect()
    self.stateChangedHandler:Disconnect()
end

function simplestateexample:OnTick(deltaTime, timePoint)
    self.TransitionCountDown = self.TransitionCountDown - deltaTime
    if (self.TransitionCountDown < 0.0) then
        SimpleStateComponentRequestBus.Event.SetState(self.entityId,
 self.Properties.States[self.StateIdx + 1])
        self.StateIdx = (self.StateIdx + 1) % table.getn(self.Properties.States)
        self.TransitionCountDown = self.Properties.TransitionInterval
    end
end

function simplestateexample:OnStateChanged(oldState, newState)
    Debug.Log("Old State: " .. (oldState or "NullState")  .. " => New State: " .. (newState
 or "NullState"))
end

return simplestateexample
```

Skinned Mesh

Component entity system is in preview release and is subject to change.

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.

Character definition

Asset file for the skinned mesh entity.

Options

Skinned Mesh component properties have the following 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

Rain occluder

When selected, the entity blocks or stops dynamic raindrops.

Affect dynamic water

When selected, the entity generates ripples in dynamic water.

Receive wind

When selected, the entity is affected by wind.

Accept decals

When selected, the entity can receive decals.

Affect navmesh

When selected, the entity affects navmesh generation.

Visibility occluder

When selected, the entity can block visibility of other objects.

Spawner

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) in the *Amazon Lumberyard Developer Guide*.

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) in the *Amazon Lumberyard Developer Guide*.

OnSpawned

Notifies that a spawn has occurred.

Parameters

`spawnedEntities`

Scriptable

`Yes`

Static Physics

Use the **Static Physics** component to represent unmovable objects that other physical entities can collide with, such as a wall. You can use a static physics component with a primitive collider (p. 534) or mesh collider (p. 522) component to provide geometry for a physical entity.

The **Static Physics** component has one property, **Enabled initially**. If selected, the entity is initially enabled in the physics simulation.

Tag

> Component entity system is in preview release and is subject to change.

Use the **Tag** component to apply one or more labels, or tags, to an entity such as **`burning`** or **`player`**. 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) in the *Amazon Lumberyard Developer Guide*.

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) in the *Amazon Lumberyard Developer Guide*.

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**.

```
local enemies = TagGlobalRequestBus.Event.RequestTaggedEntities(Crc32("Enemy"));

local burning = TagComponentRequestBus.Event.HasTag(self.entityId, 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

Transform

Component entity system is in preview release and is subject to change.

The **Transform** component controls the translation, rotation, and scale information of an entity in the 3D world. When you create an entity in Lumberyard Editor, the **Transform** component is automatically added. The translation is the coordinate location (x, y, and z axes) of the entity. The rotation is the degree in which the entity is rotated around its center. The scale is the dimension of the entity in comparison to its original size.

World space refers to the entity's absolute translation, rotation, and scale in the level. If a child is attached to a parent entity, *local space* refers to the entity's translation, rotation, and scale relative to its parent entity.

Transform Component Properties

The **Transform** component has the following properties:

Parent entity

The entity assigned as the parent. If a parent entity is specified, the **Transform** component follows the parent entity.

Values

The **Transform** component has the following values:

Translate

The local position (relative to the parent) in meters.

Rotate

The local rotation (relative to the parent) in degrees.

Scale

The local scale.

Parent activation

Configures transform behavior when the parent entity activates.

Static

Entities that can't be moved at run time. Some systems in Lumberyard treat static entities differently than movable entities (for example, the renderer can optimize static entities, making them less resource intensive to draw).

Network Sync

The **Transform** component has following network sync options:

Sync to replicas

Transform component syncs in a networked game. If you enable this option, you must also add the Network Binding (p. 527) component. Otherwise, the entity is not replicated and the interpolation options don't apply.

Position Interpolation

The smoothing of position between network updates and interruptions. This is useful if your objects change location and you notice visual jitter or sudden changes in orientation due to network conditions.

Rotation Interpolation

The smoothing of rotation between network updates and interruptions. This is useful if your objects rotate and you notice visual jitter or sudden changes in orientation due to network conditions.

> **Note**
> Scale interpolation is not supported in the **Transform** component.

EBus Request Bus Interface

TransformBus is the request bus for the **Transform** component. An entity's transform is the translation, rotation, and scale information.

For more information about using the event bus (EBus) interface, see Event Bus (EBus).

Use the following request functions with the EBus interface to communicate with other components of your game.

GetLocalTM

Returns the entity's local transform. Doesn't include the parent entity's transform.

Parameters

None

Return

Entity's local transform.

SetLocalTM

Sets the entity's local transform, relative to its parent entity, and notifies all listeners.

Parameters

Entity's local transform.

Return

None

GetWorldTM

Returns the entity's world transform, including the parent entity's transform.

Parameters

None

Return

Entity's world transform.

SetWorldTM

Sets the world transform and notifies all listeners.

Parameters

Entity's world transform.

Return

None

GetLocalAndWorld

Retrieves the entity's local and world transforms.

Parameters

Transform [out] – Local transform, relative to parent entity.

Transform [out] – World transform.

Return

None

SetWorldTranslation

Sets the entity's world space translation.

Parameters

New world space location, in x, y, and z coordinates.

Type: Vector3

Return

None

SetLocalTranslation

Sets the entity's local space translation, which is relative to its parent entity.

Parameters

New local space location, in x, y, and z coordinates.

Type: Vector3

Return

None

GetWorldTranslation

Gets the entity's world space translation.

Parameters

None

Return

Entity's world space, in x, y, and z coordinates.

Type: Vector3

GetLocalTranslation

Gets the entity's local space translation, which is relative to its parent entity.

Parameters

None

Return

Entity's local space, in x, y, and z coordinates.

Type: Vector3

MoveEntity

Moves the entity within world space.

Parameters

Offset in world space, in x, y, and z coordinates.

Type: Vector3

Return

None

SetWorldX

Sets the entity's translation x-axis coordinate in world space.

Parameters

X-axis coordinate in world space.

Type: Float

Return

None

SetWorldY

Sets the entity's translation y-axis coordinate in world space.

Parameters

Y-axis coordinate in world space.

Type: Float

Return

None

SetWorldZ

Sets the entity's translation z-axis coordinate in world space.

Parameters

Z-axis coordinate in world space.

Type: Float

Return

None

GetWorldX

Gets the entity's translation x-axis coordinate in world space.

Parameters

None

Return

X-axis coordinate in world space.

Type: Float

GetWorldY

Gets the entity's translation y-axis coordinate in world space.

Parameters

None

Return

Y-axis coordinate in world space.

Type: Float

GetWorldZ

Sets the entity's translation z-axis coordinate in world space.

Parameters

None

Return

Z-axis coordinate in world space.

Type: Float

SetLocalX

Sets the entity's translation x-axis coordinate in local space.

Parameters

X-axis coordinate in local space.

Type: Float

Return

None

SetLocalY

Sets the entity's translation y-axis coordinate in local space.

Parameters

Y-axis coordinate in local space.

Type: Float

Return

None

SetLocalZ

Sets the entity's translation z-axis coordinate in local space.

Parameters

Z-axis coordinate in local space.

Type: Float

Return

None

GetLocalX

Gets the entity's translation x-axis coordinate in local space.

Parameters

None

Return

X-axis coordinate in local space.

Type: Float

GetLocalY

Gets the entity's y-axis coordinate in local space.

Parameters

None

Return

Y-axis coordinate in local space.

Type: Float

GetLocalZ

Gets the entity's z-axis coordinate in local space.

Parameters

None

Return

Z-axis coordinate in local space.

Type: Float

GetWorldRotation

Gets the angles in radians for each principle axis around which the world transform is rotated in the following order: z-axis, y-axis, x-axis.

Parameters

None

Return

The Euler angles in radians, which indicate the degree of rotation around each principle axis.

Type: Vector3

GetWorldRotationQuaternion

Gets the quaternion that represents the world rotation.

Parameters

None

Return

The quaternion that represents the world rotation.

Type: Quaternion

SetLocalRotation

Sets the local rotation around each principle axes in the following order: z-axis, y-axis, x-axis.

Parameters

The Vector3 denoting radian angles of the rotations around each principle axis.

Type: Vector3

Return

None

SetLocalRotationQuaternion

Sets the local rotation matrix using a quaternion.

Parameters

The local rotation matrix.

Type: Quaternion

Return

None

RotateAroundLocalX

Rotates around the local x-axis for a radian angle.

Parameters

The radian angle to rotate around the local x-axis.

Type: Float

Return

None

RotateAroundLocalY

Rotates around the local y-axis for a radian angle.

Parameters

The radian angle to rotate around the local y-axis.

Type: Float

Return

None

RotateAroundLocalZ

Rotates around the local z-axis for a radian angle.

Parameters

The radian angle to rotate around the local z-axis.

Type: Float

Return

None

GetLocalRotation

Gets angles in radian for each principle axis around which the local transform is rotated in the following order: z-axis, y-axis, x-axis.

Parameters

None

Return

Indicates how much in radian is rotated around each principle axis.

Type: Vector3

GetLocalRotationQuaternion

Gets the quaternion representing the local rotation.

Parameters

None

Return

The quaternion that represents the local rotation.

Type: Quaternion

SetLocalScale

Sets local scale of the transform.

Parameters

Local scale of the transform, in x, y, and z coordinates.

Type: Vector3

Return

None

SetLocalScaleX

Sets local scale of the transform on the x-axis.

Parameters

X-axis coordinate for the local scale.

Type: Float

Return

None

SetLocalScaleY

Sets local scale of the transform on the y-axis.

Parameters

Y-axis coordinate for the local scale.

Type: Float

Return

None

SetLocalScaleZ

Sets local scale of the transform on the z-axis.

Parameters

Z-axis coordinate for the local scale.

Type: Float

Return

None

GetLocalScale

Gets the scale value on each axis in local space.

Parameters

None

Return

Scale value for each axis in local space.

Type: Vector3

GetWorldScale

Gets the scale value on each axis in world space. Note that the transform is skewed when it is rotated and has a parent transform scale in which the returned world scale from this function is inaccurate.

Parameters

None

Return

Scale values for each axis in world space.

Type: Vector3

GetParentId

Returns the parent entity's ID. If the entity does not have a parent, the entity ID is invalid.

Parameters

None

Return

EntityID of the parent

Type: Int

SetParent

Sets the entity's parent entity and notifies all listeners. The entity's local transform is moved into the parent entity's space to preserve the entity's world transform.

Parameters

EntityId – Parent entity ID

Type: Int

Return

None

SetParentRelative

Sets the entity's parent entity, moves the transform relative to the parent entity, and notifies all listeners. This function uses the world transform as a local transform and moves the transform relative to the parent entity.

Parameters

EntityId – Parent entity ID

Type: Int

Return

None

GetChildren

Returns the entity IDs of the entity's immediate children.

Parameters

None

Return

Vector of EntityIds

GetAllDescendants

Returns the entity IDs of all descendants of the entity. The descendants are the entity's children, the children's children, and so on. The entity IDs are ordered breadth first.

Parameters

None

Return

Vector of EntityIds

GetEntityAndAllDescendants

Returns the entity ID of the entity and all its descendants. The descendants are the entity's children, the children's children, and so on. The entity IDs are ordered breadth first, and this entity's ID is the first in the list.

Parameters

None

Return

Vector of EntityIds

IsStaticTransform

Returns whether the transform is static. A static transform doesn't move and doesn't respond to requests to move it.

Parameters

None

Return

Boolean

EBus Notification Bus Interface

TransformNotificationBus is the notification bus for the **Transform** component. 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).

OnTransformChanged

Signals that the local or world transform of the entity changed.

Parameters

Transform – The new local transform of the entity

Transform – The new world transform of the entity

OnParentChanged

Signals that the parent of the entity changed.

Parameters

EntityId – The entity ID of the previous parent. The entity ID is invalid if there was no previous parent.

EntityId – The entity ID of the new parent. The entity ID is invalid if there is no new parent.

OnChildAdded

Signals that a child was added to the entity.

Parameters

EntityId – The entity ID of the added child

OnChildRemoved

Signals that a child was removed from the entity.

Parameters

EntityId – The entity ID of the removed child

Trigger Area

Component entity system is in preview release and is subject to change.

The **Trigger Area** component provides generic triggering services by using Shape (p. 546) components as its bounds.

To add a trigger component

1. Create a new entity, or add the **Trigger Area** component to an existing entity.
2. In the Entity Inspector, click **Add Component**, then select **Trigger Area**.

   Entity Inspector displays a message, "This component is missing a required component service and has been disabled.
3. Click **Add Required Component**, then click one of the shape components that appears.

   The shape component you select defines the boundaries for the trigger.
4. If you want to change the boundaries of your shape component, simply remove the existing shape component, and then add a different shape component.

Trigger Area Component Properties

The **Trigger Area** component has the following properties:

Network Bindable

Network bindable 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) in the *Amazon Lumberyard Developer Guide*.

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) in the *Amazon Lumberyard Developer Guide*.

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 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

The following is an example of script using the EBus interface.

```
local triggerexample =
{
    Properties =
    {
    }
}

function triggerexample:OnActivate()
    self.triggerHandler = TriggerAreaEntityNotificationBus.Connect(self, self.entityId)
end

function triggerexample:OnDeactivate()
    self.triggerHandler:Disconnect()
end

function triggerexample:OnEntityEnteredTriggerArea(entityId)
    Debug.Log("------- TRIGGERED.")
end

return triggerexample
```

UI Canvas Asset Ref

Component entity system is in preview release and is subject to change.

With the **UI Canvas Asset Ref** component, you can associate a UI canvas with a component entity in a level.

When you set up a UI canvas asset ref component, you can:

- Select whether to automatically load the UI canvas when the level loads
- Use a flow graph that is associated with the UI canvas to reference it using the **UIe** flow graph nodes.

Use this component in conjunction with the **UI Canvas on Mesh** (p. 581) component if you want to place a UI canvas on a 3D mesh that a player can interact with.

For more information about how to use the **UI Canvas Asset Ref** component, see Placing UI Canvases in the 3D World (p. 1550).

UI Canvas Asset Ref Component Properties

The **UI Canvas Asset Ref** component has the following properties:

Canvas pathname

The relative pathname of the UI canvas asset file.

Load automatically

If selected, the canvas is automatically loaded when this component entity is loaded, typically when the level is loaded.

UI Canvas on Mesh

Component entity system is in preview release and is subject to change.

With the **UI Canvas on Mesh** component, you can place a UI canvas on a component entity in the 3D world that a player can interact with via ray casts. Use this component in conjunction with the **UI Canvas Asset Ref** (p. 581) component.

For more information about how to use the **UI Canvas Asset Ref** component, see Placing UI Canvases in the 3D World (p. 1550).

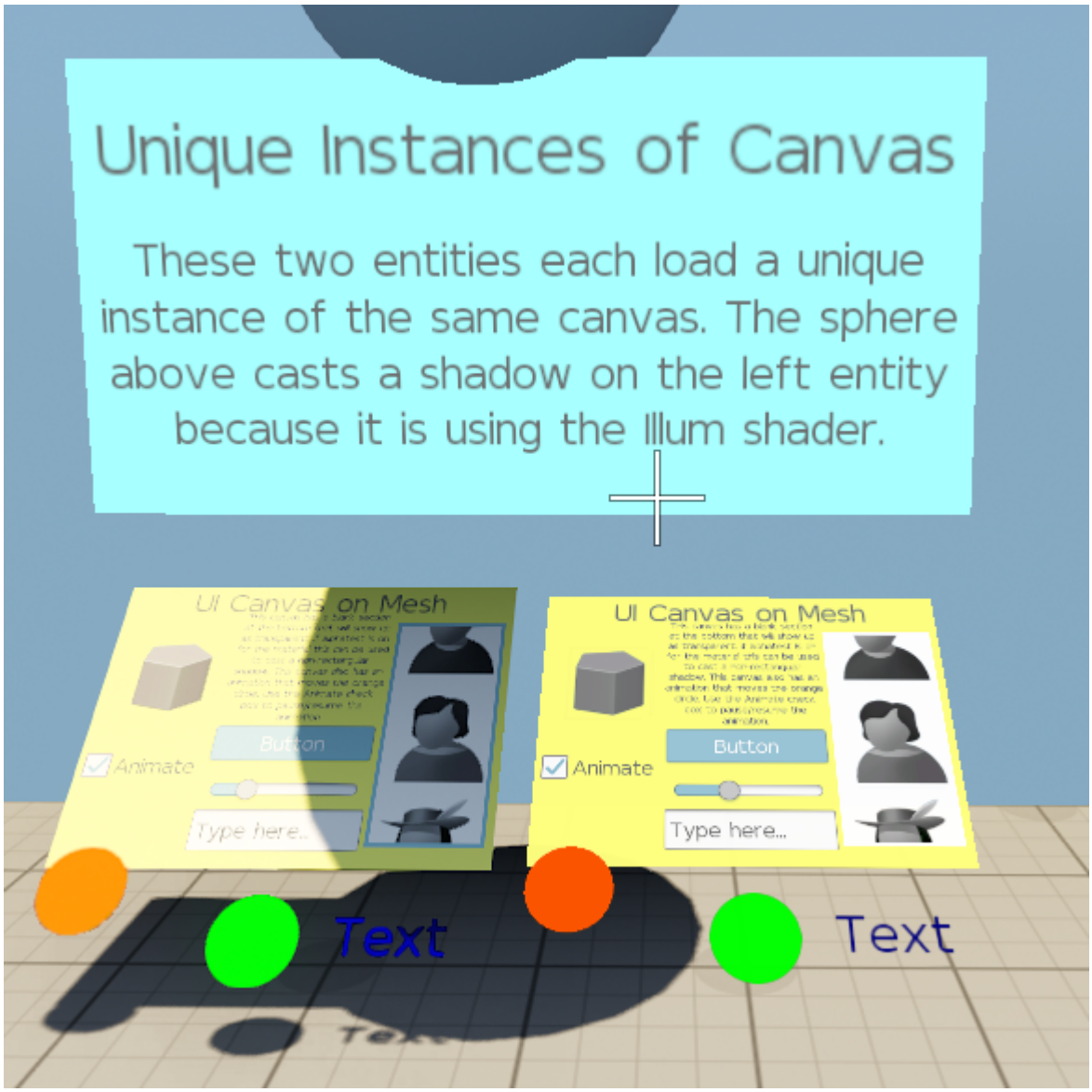
UI Canvas on Mesh Component Properties

The UI Canvas on Mesh component has the following properties:

Render target override

For simple cases, you can leave this property blank. The UI canvas specifies a render target, and that render target can be used as a texture name for the material on the 3D mesh.

You can use the **Render target override** property when you want to load two unique instances of the same UI canvas that the user can set to different states. This example case is demonstrated in the **UiIn3DWorld** level in the **Samples Project**.



UI Canvas Proxy Ref

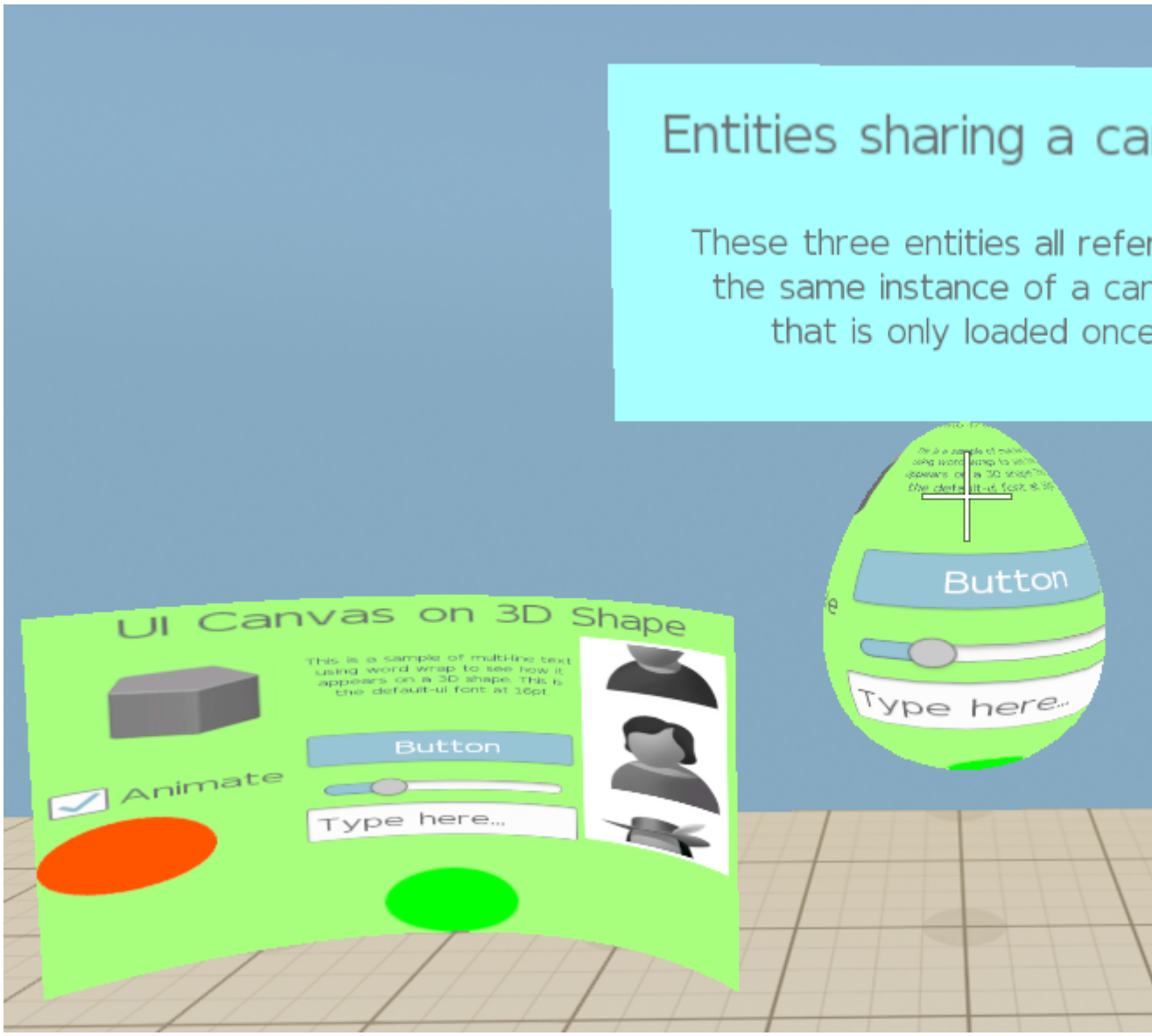
Component entity system is in preview release and is subject to change.

With the UI Canvas Proxy Ref component, you can associate a component entity in a level with another component entity that is managing a UI canvas. Use this component in conjunction with the **UI Canvas on Mesh** (p. 581) component if you want to place a UI canvas on a 3D mesh that a player can interact with in several places in the 3D world.

Use of this component is often a special case, as it supports displaying the same UI canvas on multiple entities in the 3D world. The **UI Canvas Proxy Ref** component allows the component entity that it is on it to act as if it had a **UI Canvas Asset Ref** (p. 581) component but without having to load another copy of the UI canvas. This means that, as the user interacts with one UI canvas on a 3D object, the other 3D object shows the same changes.

To see an example of using the **UI Canvas Proxy Ref** component, open the **UiIn3DWorld** level in the **Samples Project**.

The following picture shows three entities that share the same loaded canvas. The curved plane entity has a **UI Canvas Asset Ref** component and the egg and the sphere both have **UI Canvas Proxy Ref** components:



UI Canvas Proxy Ref Component Properties

The UI Canvas Proxy Ref component has the following properties:

Canvas Asset Ref entity

Click the picker (hand icon) to select the component entity that you want to mirror. The picked component must have the **UI Canvas Asset Ref** component on it.

Video Playback Component

Component entity system is in preview release and is subject to change.

You can use the **Video Playback** component to play a video file on an object in your Lumberyard level. For example, you can use a flat or plane object to simulate a movie screen. You then add the video playback component to it and specify a video file you want to display. You can use flow graphs or Lua scripting to trigger the video to play, pause, or stop, depending on player actions.

To use the video playback component, you must perform an initial setup. This involves installing software (either FFmpeg or LibAV), adding the video playback gem to your project, and then rebuilding your project to enable the gem.

You can also set your video to play in visual stereo (not audio stereo). To test this feature, you must use a virtual reality head mounted display (HMD).

Audio is not currently supported with the video playback component. You can, however, trigger audio playback separately if you want to play audio along with your video.

Setting up Video Playback

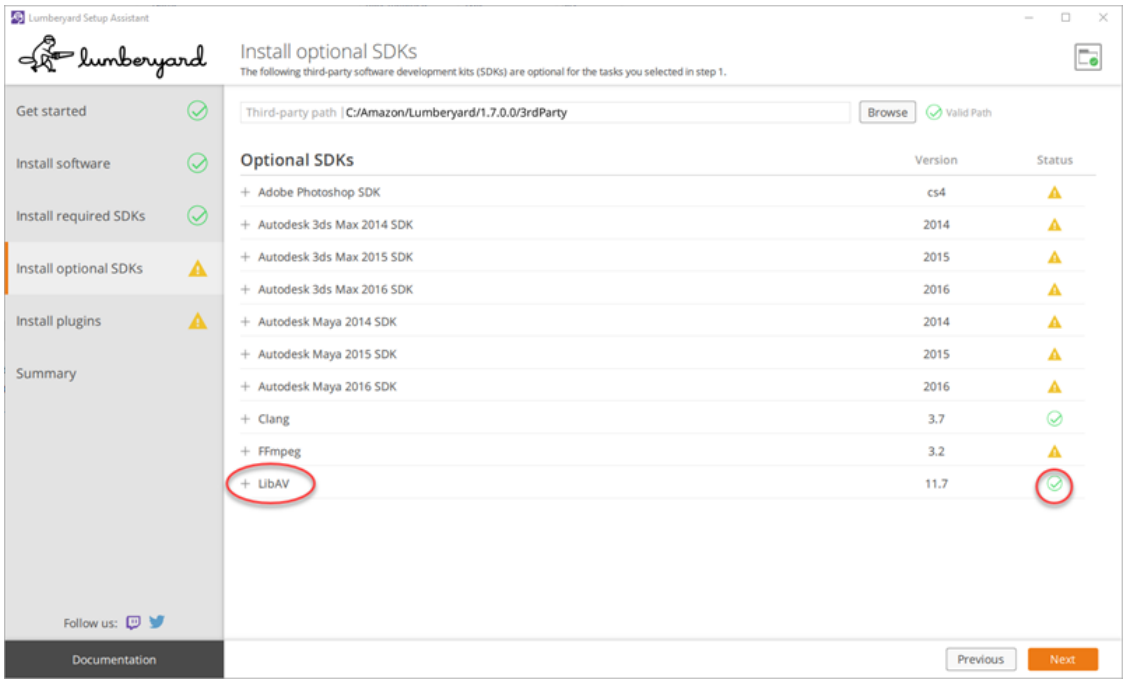
To set up video playback in Lumberyard, you must install either FFmpeg or LibAV. If both are installed, Lumberyard uses FFmpeg. To use FFmpeg, follow the instructions in the Lumberyard Setup Assistant in the **Install software** page under **Optional software**. If you want to use LibAV, remove FFmpeg; then download LibAV and carry out the following instructions.

To install LibAV

1. Download LibAV from http://builds.libav.org/windows/. Select the release-lgpl build. As of this writing, version 11.7 is the latest build.
2. Extract the `.7z` file to a directory called `libav`. To open and extract `.7z` files, you must use a 7z application, such as 7-Zip.

   Within the `libav` directory, you should have the following:

   - Directory named `usr`
   - `config.log`
   - `md5sum`
3. Move the `libav` directory to Lumberyard's `3rdParty` directory.
4. Run Lumberyard Setup Assistant and view the **Install optional SDKs** page to verify that Lumberyard detects LibAV.



To use the video playback component, you must also install the **Video Playback** gem. Follow the Gems (p. 1060) instructions to install the **Video Playback** gem and rebuild your project.

Using the Video Playback Component

The video playback component is available to use in Lumberyard after you have installed the required software, added the **Video Playback** gem to your project, and then rebuilt your project.

Video playback supports the following container formats:

- `.mp4`
- `.mkv` (recommended)
- `.webm` (recommended)

Video playback supports the following codecs:

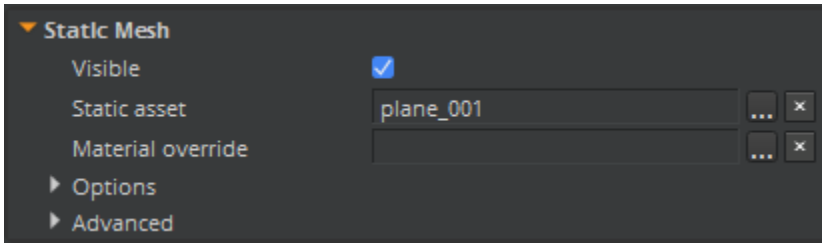
- `h.264`
- `h.265`
- `VP8` (recommended)
- `VP9` (recommended)

The basic setup for the video playback component involves placing a camera, adding a static mesh and video playback component, and configuring material.

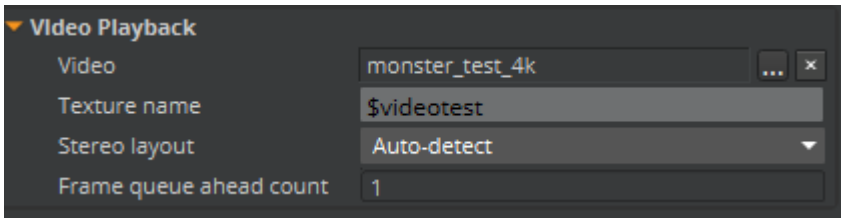
To use the video playback component

1. If you do not yet have a camera in your scene, place a camera component (p. 467) near where your video playback is to be placed.

   You can use the camera to view your video playback. Ensure that the camera is facing the direction where you place your video playback component.
2. Create a new component entity by right-clicking in your scene and clicking **Create new component entity**.
3. Use the **Entity Inspector** (p. 441) to add a static mesh component (p. 521) to your new component entity.
4. Select a **Static asset** for your static mesh component. This is the object that your video renders onto. A cube or plane is a good test mesh.

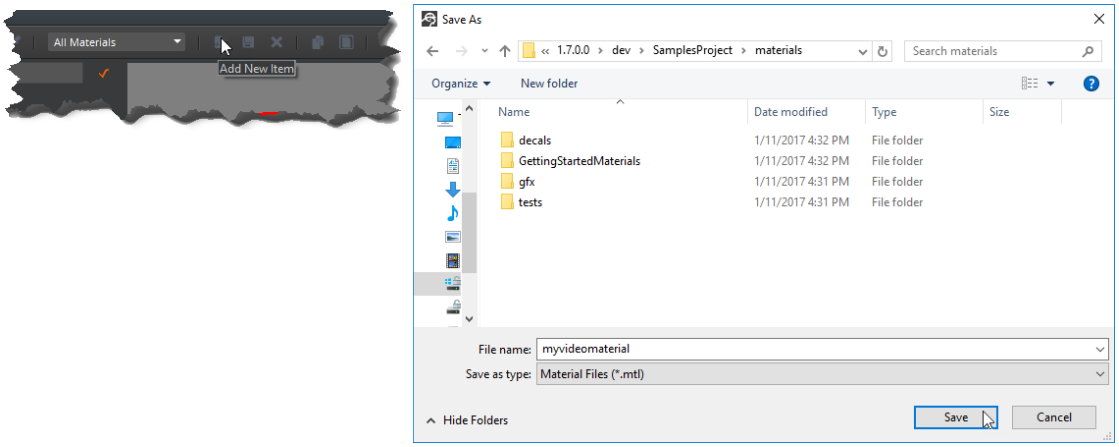
   

5. Add the **Video Playback** component to the same entity. To do this, click **Add Component**, **Rendering**, **Video Playback**.
6. In the video playback component's **Video** setting, select the video that you want to display.
7. For **Texture name**, type a name for your texture, preceded by a dollar sign ($). This is a user-defined field, so it can be anything you want, but it must begin with a $ character to indicate that it is a render target. For example, **`$videotest`** is a valid name but **`videotest`** is not.

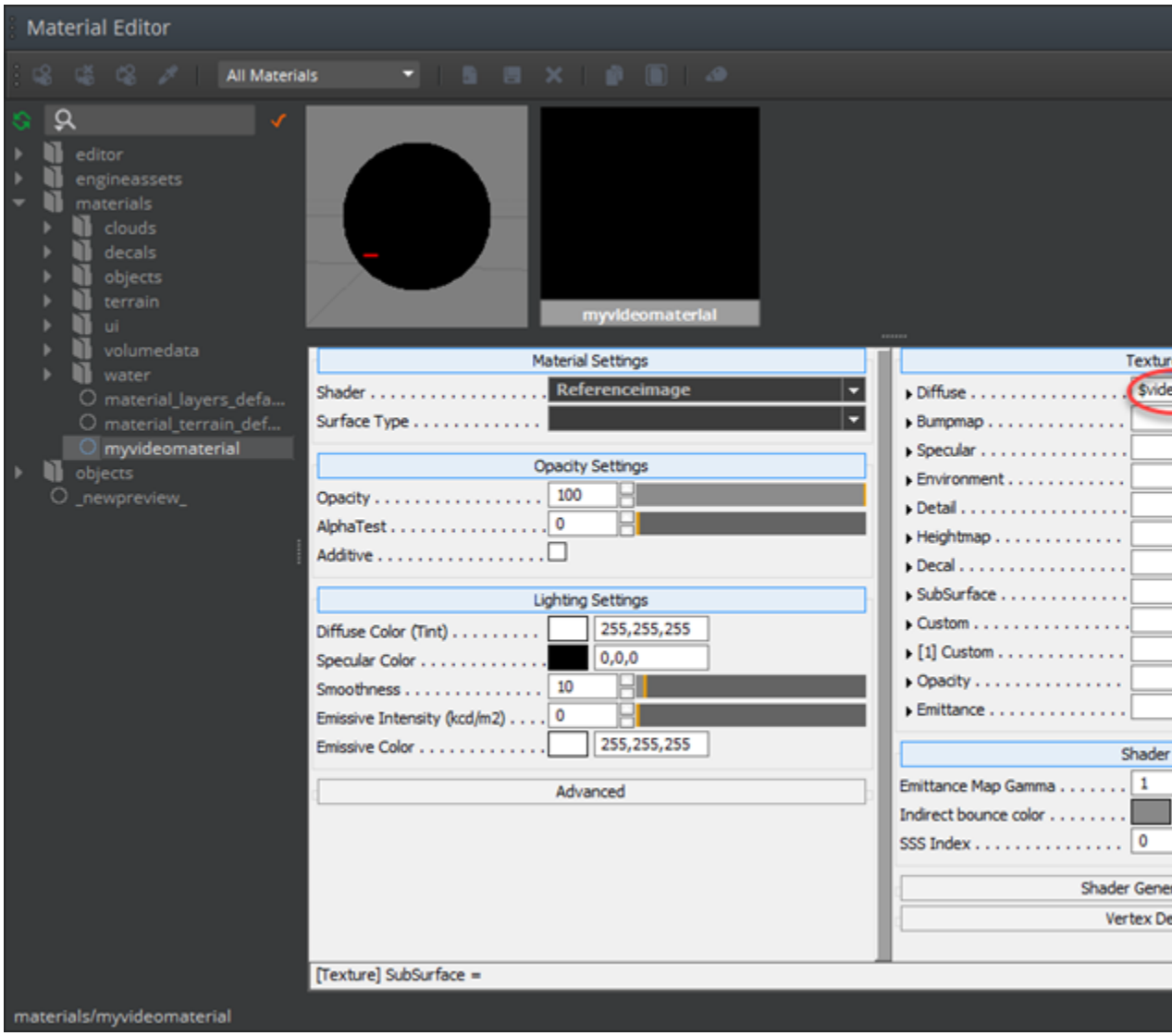
   

8. Use the **Frame queue ahead count** setting to set the number of frames to buffer. Leave this value at **`1`**, an acceptable value. A value of **`2`** or **`3`** is typically safe as well.

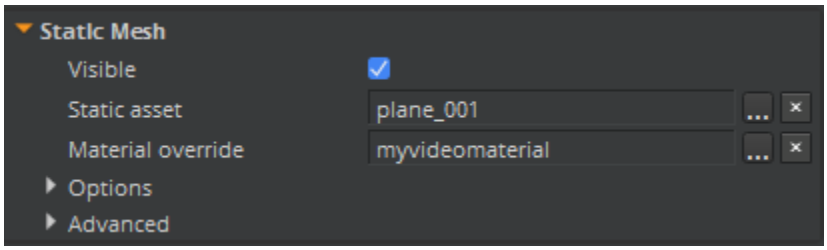
   Queueing too many frames to buffer (for example, a value of **`100`** frames) can use too much memory and cause performance issues.
9. Open the Material Editor (p. 1350) (use the keyboard shortcut **M** to open it quickly).
10. Create a new material by clicking **Add New Item**, as shown in the following picture. Give the material a descriptive name, such as **`myvideomaterial`**.



11. Under **Texture Maps**, on the **Diffuse** line, type the name of your video component's **Texture name** field. Be sure to include the $ character.



12. Close the **Material Editor** and return to the **Entity Inspector** (p. 441). In the static mesh component, for the **Material override** setting, select the material you just created.

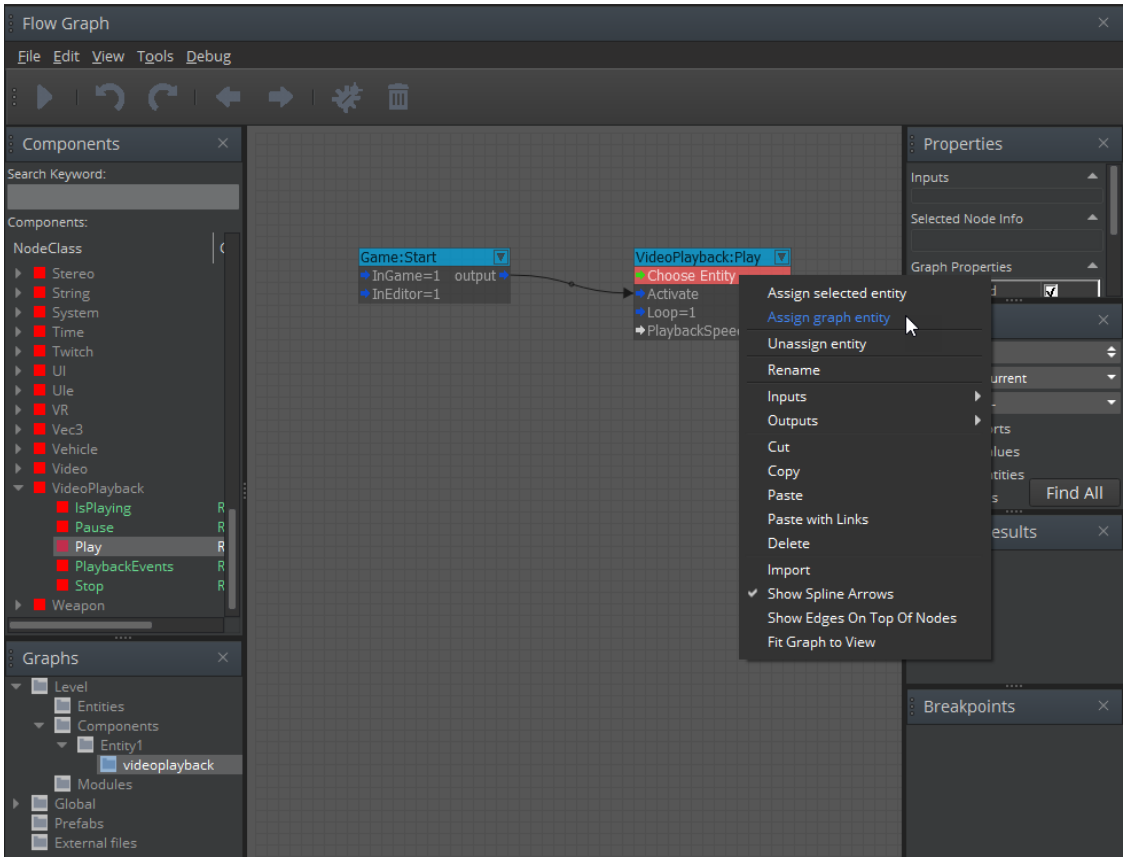


You can trigger the video to play at the start of your game using either flow graphs or Lua scripting. The following procedure shows you how to create a simple flow graph to start playing the video when your game starts.

To set up a flow graph to start your video

1. In the viewport, right-click on the static mesh/video playback entity. Then click **Flowgraph**, **Add**. Type a name for your new flow graph.
2. Drag a **Game:Start** node and a **VideoPlayback:Play** node onto your flow graph. Connect the **output** port of the **Game:Start** node to the Activate port of the **VideoPlayback:Play** node.

   Right-click **Choose Entity** in the **VideoPlayback:Play** node and click **Assign graph entity**.



3. To play the game and test your video playback, press **Ctrl G**.

Note
Audio playback is not supported with this component. You can trigger audio playback separately.

Setting Up Stereo Video Playback

Before setting up stereo video playback, ensure that you have completed the setup instructions in Setting up Video Playback (p. 584).

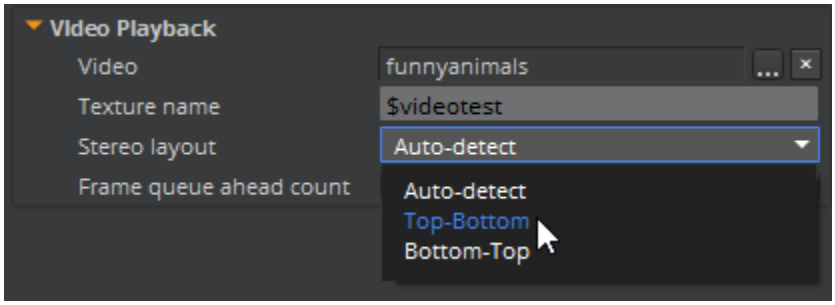
Stereo video playback means that the video presents a slightly different image for each eye, creating a 3D feel. *Stereo* in this case refers to visual stereo only; audio is not supported in the video playback component. While audio is not supported and must be synced externally, it is possible to play back spatialized audio. To do so, you must use the full, commercial version of Wwise and a 3D spatializer plugin such as Oculus Spatializer or RealSpace 3D.

To set up stereo video playback, your source video file must be laid out for stereo. Lumberyard supports videos in a top-bottom or bottom-top layout. You must have a VR headset to verify that the video is playing in stereo.

To set up stereo video playback, follow the instructions in the section called "Using the Video Playback Component" (p. 585). The only differences in the setup are the following:

- You must use a source video file with 3D or stereo information
- Set the **Stereo Layout** property initially to **Auto-Detect**. If it fails to auto-detect, manually set it to **Top-Bottom** or **Bottom-Top**.

  All supported video files should have their stereo layout written into their metadata. This, however, is not a requirement and may not have been inserted by your encoder. If you would like to inject stereo metadata into your video, see https://support.google.com/jump/answer/7044297?hl=en.



When you enter game mode (using **Ctrl G**), you should see the left eye of your video play. If you do not see this, try changing your **Stereo layout** setting.

To verify that your video is playing in stereo, you must enter VR mode. You can enter VR mode by clicking **VR Preview** at the bottom right corner of the viewport. Then press **Ctrl G** to enter game mode. If your **VR Preview** button isn't enabled, or you can't get into VR preview mode, ensure that your VR headset is working outside of Lumberyard and then restart the Lumberyard editor.

Playing stereo video is resource intensive. Because the video is often close in proximity to the player, it becomes easy to detect inconsistencies and artifacts in the video. To prevent that, use higher resolution videos whenever possible. To conserve resources, do not play more than one or two high resolution stereo videos at a time.

Lua Bindings for Video Playback

You can use Lua bindings to interact programmatically with video playback components that you've placed in your scene. Lua provides a way to establish complex logic for playing, pausing, and stopping videos.

Global Functions

The following functions provide programming interfaces for the video playback systems.

VideoPlaybackRequestBusSender

Parameters

`EntityID`

Return

Returns the `VideoPlaybackRequestBusSender` object that is connected to the specified entity. For more information, see VideoPlaybackRequestBusSender Object (p. 590).

VideoPlaybackNotificationBusHandler

Exposes callbacks to your Lua script that are triggered by events during video playback.

For more information, see VideoPlaybackNotificationBusHandler Object (p. 591).

Parameters

`Table` – The Lua table to which you want to expose the callback functions. Pass `self` to expose the callbacks to the current Lua script.

`EntityId` –

Return

Returns the `VideoPlaybackRequestBusSender` object that is connected to the specified entity. For more information, see VideoPlaybackRequestBusSender Object (p. 590).

VideoPlaybackRequestBusSender Object

The `VideoPlaybackRequestBusSender` object contains functions with which you can send requests to the video playback component.

`Bool IsPlaying()`

Returns `true` if the video is playing. If the video is paused or stopped, returns `false`.

`Void Play()`

Plays the video. If no video is selected or the video is already playing, this has no effect.

`Void Pause()`

Pauses the video. If the video is already paused, this has no effect.

`Void Stop()`

Stops the video and remains on the last frame. When the video plays again, it begins at the first frame of the video. If the video is already stopped, this has no effect. If the video is playing or paused, the video stops.

`Void EnableLooping(Bool)`

Sets whether this video automatically restarts from the beginning once the end of the video is reached. Pass `true` to enable looping or `false` to disable looping. Looping is disabled by default.

`Void SetPlaybackSpeed(Float)`

Sets how fast the video plays. For example, **`1.0`** is normal speed, **`0.5`** is half speed, **`2.0`** is double speed, and so on.

Caution is advised when setting the video speed. Setting a speed that is too high can result in choppy playback.

VideoPlaybackNotificationBusHandler Object

The `VideoPlaybackNotificationBusHandler` object exposes callback functions to your Lua script that are triggered by events that happen during video playback.

`Void OnPlaybackStarted()`

Called when video playback begins.

`Void OnPlaybackPaused()`

Called when video playback pauses. Not called when video stops.

`Void OnPlaybackStopped()`

Called when video playback is stopped by the user. If the video reaches the end and is not set to loop, this function is not called.

`Void OnPlaybackFinished()`

Called when all frames in the video are played. This is not called if the user manually stops video playback. If looping is enabled, this function is called every time the video loops.

Setting Up Video Playback with Flow Graph

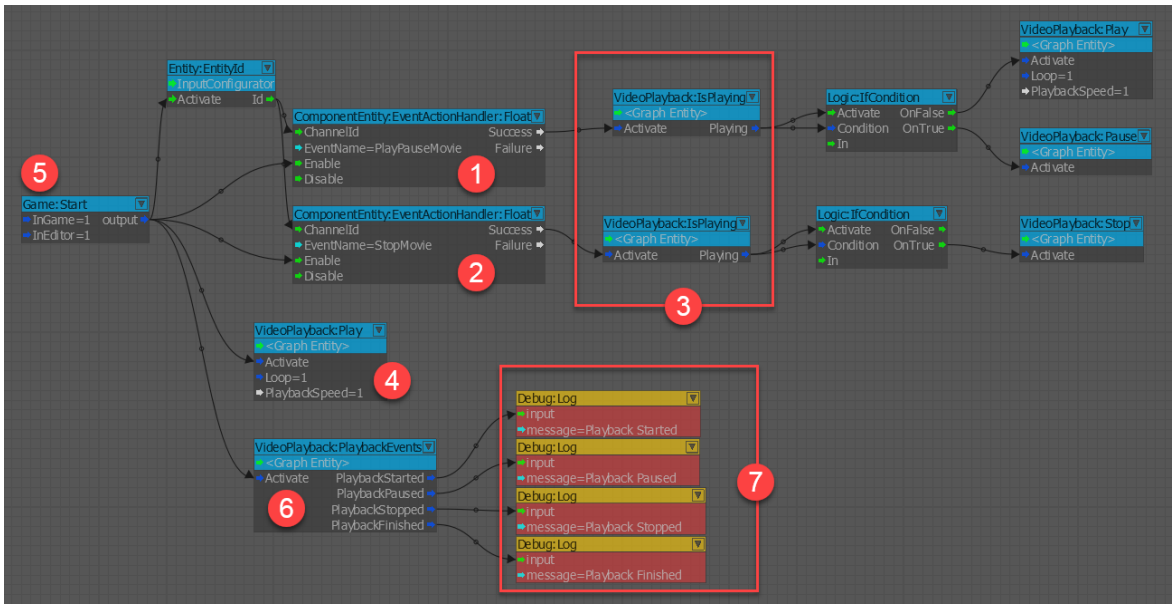
Using the Video Playback flow graph nodes (p. 1031), you can set up video playback actions that trigger based on certain actions.

In the following flow graph example, the **PlayPauseMovie** input event (1) plays or pauses the video on the attached graph entity. The **StopMovie** input event (2) stops the movie.

The **PlayPauseMovie** and **StopMovie** events each connect to an **IsPlaying** node (3), which checks whether the video should be played, paused, or stopped.

The **Play** node (4) sets **PlaybackSpeed** to **`1`** (normal speed) and enables the **Loop** setting (Boolean value of **`1`**), indicating that video looping is enabled. The **Play** node (4) also triggers the video to play on **Game:Start** (5).

The **PlaybackEvents** node (6) triggers the **Debug** nodes (7) to print to the console whenever the video plays, pauses, stops, and finishes.



VR Preview

The **VR Preview** component, when added to an entity or spawned from a slice, creates a user-editable navigation mesh (p. 523). The navigation mesh is used to define valid areas that users can teleport to. The **VR Preview** component sets up necessary dependencies for exploring or navigating a scene in virtual reality. This component contains no editable properties.

To use the VR Preview component

1. Create a new entity (p. 593).
2. Open the **Entity Inspector** (p. 441). Add the **VR Preview** component to the entity by clicking **Add Component**, **VR**, **VR Preview**.

   A navigation mesh of 50x50x50 size is generated around the entity.
3. Save the entity to a slice (p. 601).

   Now you can drag the saved slice to another part of the level, and the navigation mesh will be generated around the entity.

The navigation mesh that is generated is a separate entity from the original entity. Lumberyard names the new nav mesh entity by appending `_NavMesh` to the original entity name.

If you delete the original entity, the nav mesh entity continues to exist. However, if you delete the nav mesh entity, it won't be generated again.

Working with Entities and Components

Component entity system is in preview release and is subject to change.

You can use the component entity system tools to create entities and add components to them. See the following topics:

Topics

Creating an Entity

Entities are objects with which the player interacts. For more information about entities, see Component Entity System (p. 437).

To create a new entity

Do one of the following:

- In the Lumberyard Editor viewport, right-click and choose **Create entity**. This creates a new entity at the cursor location with a basic transform component, which gives it a 3D location in the level.
- In the **Entity Outliner** (p. 437), right-click and choose **Create entity**. This creates a new entity in the center of your Lumberyard viewport.
- Use the **Asset Browser** (p. 179) to create new entities that already have the preferred configuration, depending on the particular asset.

  For example, if you drag a `.cgf` mesh asset from **Asset Browser** into the Lumberyard Editor viewport, Lumberyard 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.

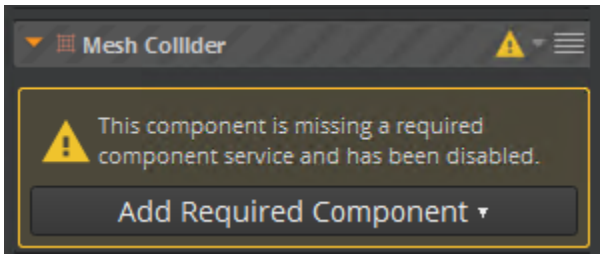
Adding Components to an Entity

After you create an entity, you can add components to it.

To add a component to an entity

1. In the **Entity Inspector** (p. 441), click **Add Component** or right-click and choose **Add Component**.
2. Components are grouped by category, such as **Animation**, **Camera**, and **Gameplay**. To find a specific component quickly, type the name into the search bar at the top of the component list.

   Note
   You can pause on the component name to see a description of that component. For example, for the **Ragdoll** component, the description reads: "The Ragdoll component uses physics to drive characters and is ideal for simulating environmental effects upon unconscious characters."
3. Select the component.
4. Specify your settings for the component. For example, for the **Ragdoll** component, you can specify the **Mass**, **Damping**, and **Buoyancy**. For more information, see Rag Doll (p. 534).
5. If you add a component that requires another component to function, the **Entity Inspector** displays the following message:



For example, the **Ragdoll** component also requires the **Skinned Mesh** component. Click **Add Required Component** and then select the required component.

> **Note**
> Components must have the required dependencies to appear in game mode. The **Entity Inspector** displays a warning message for the following scenarios:
>
> - Incompatibilities exist between components.
> - A component is missing a required component.
> - A component is a duplicate.
>
> You must resolve the issue before the component can be used for the entity. Otherwise, the component is disabled.

Managing Components

You can also use the context menu in the **Entity Inspector** to remove, copy, and reorder components.

To manage components from the context menu

1. Select the entity in the **Entity Outliner** (p. 437) or in your viewport.
2. In the **Entity Inspector** (p. 441), right-click on a component, or click the menu button in the header, and then select one of the following:
   - **Add Component** – Add a component to the entity
   - **Remove Component** – Remove a component from an entity
   - **Cut Component** – Cut a component from one entity and paste it onto a different entity
   - **Copy Component** – Copy a component and then paste it onto the same entity, which creates a duplicate component that you can then customize
   - **Paste Component** – Copy a component and then paste it onto a different entity
   - **Enable Component** – Make a component available
   - **Disable Component** – Make a component unavailable
   - **Move Component Up** – Move a component up in the list
   - **Move Component Down** – Move a component down in the list

   You can also reorder components by dragging individual components to their preferred order.

   Organize the components for an entity in any order that makes sense to you. The custom order for your components is saved to the level. The component order has no significance to its function.

   > **Note**
   > Some actions are disabled, depending on the context. For example, you can't paste a component if you have not copied one.

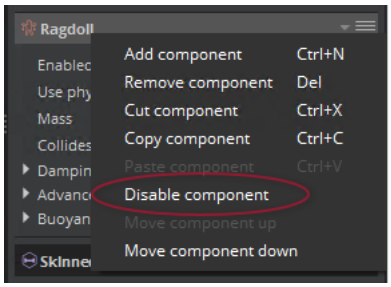
Disabling and Enabling Components on Entities

After you add components to entities, you can choose to disable and enable them as needed. This can help you see how different components interact with the entity. For example, if you create an entity and add the **Area Light**, **Environment Probe**, and **Point Light** components to it, you can see how the different light components interact with the entity.

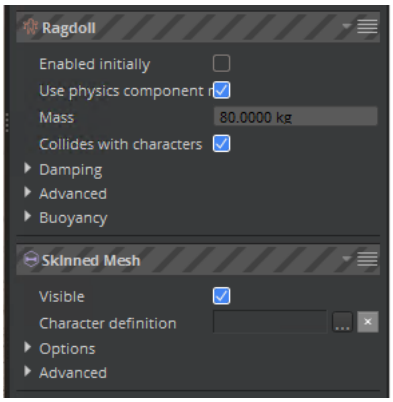
You can then disable the components that you don't want and reenable them later. Disabled components are read only and do not activate, generate warnings, or export with game data.

To disable and enable components

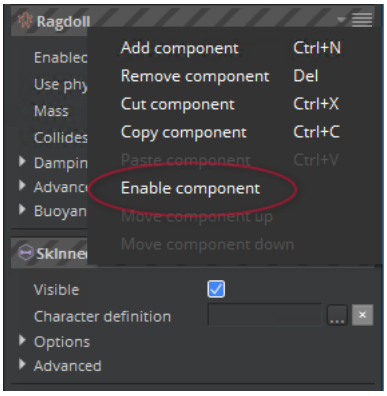
1. In the **Entity Outliner** (p. 437), select the entity that you want.
2. In the **Entity Outliner**, under **Add Component**, right-click the component that is associated with the entity.
3. Click **Disable component** to disable the component.



Disabled components are dimmed with stripes.



4. To reenable a disabled component, right-click the component and choose **Enable component**.

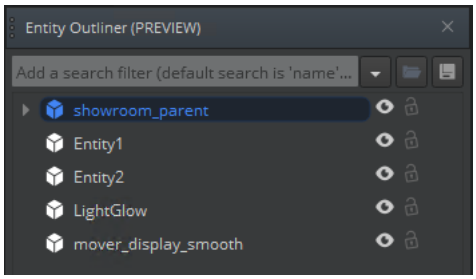


Finding an Entity

The **Entity Outliner** shows all the entities and their child entities in your level.

To find an entity in your level

1. Use the **Entity Outliner** (p. 437) to see the entities in your level.
2. To find a specific component quickly, type any part of its name into the search box. Entities with names that match the filter's entry appear, along with their child or parent entities.



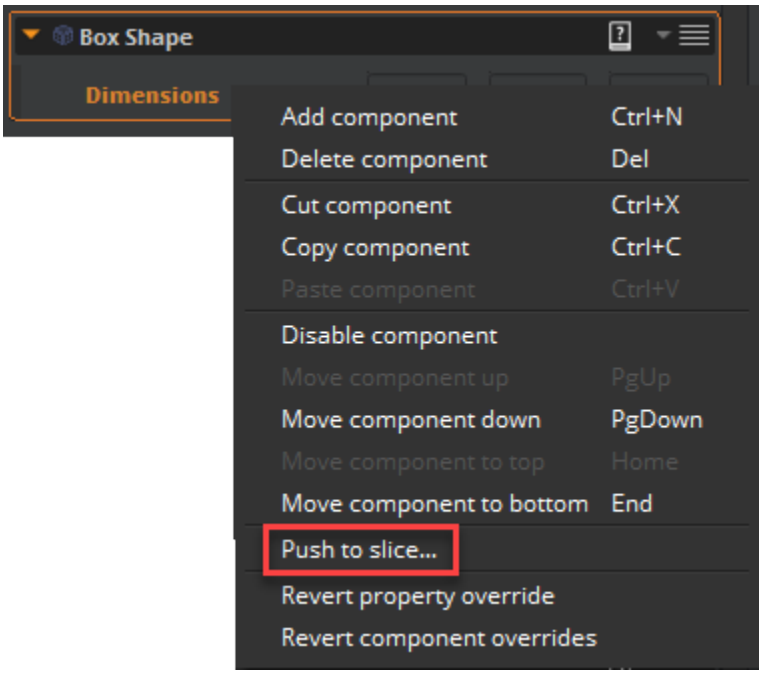
Editing Component Properties

You can edit properties for your components. For example, you can change the mass or enable the **Collides with characters** property for the **Ragdoll** component.

To edit component properties

1. Select the entity in the viewport or the **Entity Outliner** (p. 437).
2. Edit the component properties in the **Entity Inspector** (p. 441).
3. To undo a change that you made to a property, press **Ctrl+Z**. To redo the change, press **Ctrl+Shift+Z**.
4. To edit multiple entities at the same time, select the entities that you want and make your changes. The changes that you make to the first entity propagate to all selected entities.

   Note
   If your entity is part of a slice instance, any settings that you modify from the source slice asset appear in orange within the **Entity Inspector**. Right-click the setting and choose **Revert property override** to reset to the original value for the property. Choose **Revert component overrides** to reset the original value for all properties on the component. Or choose **Push to slice** to save your change to the slice asset, which affects all other instances of the target slice asset.
   For more information about slices, see Working with Slices (p. 597).



Working with Slices

Component entity system is in preview release and is subject to change.

When you build a level, you place certain items repeatedly, such as a prop, furniture, or a piece of landscaping. To create multiples of an item, you could copy and paste it throughout your level. Doing this would result in each item having its own independent properties. If you wanted to change one property, such as making all the motorbikes blue instead of red, you would have to modify each one. This is time consuming and inefficient.

With Lumberyard's slices, you can modify one instance of the item and then save that change to all the other instances of your item within the game project. Saving changes to the other instances is not an automatic process. This means that you can modify a slice instance and then not save the changes. This makes that instance unique. You can also detach a slice instance so that it does not inherit saved modifications. That detached slice instance becomes a regular entity with individual properties.

Slices contain entities, including their components and properties, and may also contain instances of other slices. This ability to nest slices without flattening the hierarchy is a feature unique to Lumberyard.

Slices are saved as `.slice` files within your game project directory.

Topics

The sections in this chapter use the following terminology to describe working with slices.

Source Slice

The source slice is the `.slice` asset that is saved in the game project directory when you create a slice.

Owning Slice

Entities in a slice instance originate from an owning slice. For example, a tire entity from an instance of `wheel.slice` is owned by `wheel.slice`. Slices automatically pass down changes to any slice instance entities that they own. For example, say that you update the material reference on a tire entity that is owned by a `wheel.slice` instance. When you push that change to the source `wheel.slice`, Lumberyard updates all other tire entities of `wheel.slice`.

Entities can have more than one owning slice. To continue the example, `motorcycle.slice` owns two instances of `wheel.slice`. The tire entity within the two `wheel.slice` instances, which is within `motorcycle.slice`, inherits from both the wheel slice and the motorcycle slice.

Slice Instance

A slice instance is a distinct instantiation of a source slice. The slice instance inherits entities and properties from its source slice (the `.slice` file) and is updated when the source slice changes.

You can modify one slice instance. To apply those modifications to all instances of the source slice, you can save the changes to the source slice asset.

When you work with one instance of a slice, those changes are exclusive to that slice until you push the changes to the source slice.

Slice Entity

Entities that are owned by a slice are called slice entities. A slice entity inherits from its source slice. The **Entity Outliner** distinguishes slice entities from nonslice entities by color. Slice entities are blue, and nonslice entities are white.

Saving Slice Overrides

Saving slice overrides means to commit entity modifications from an instance to a source slice. When you use the **Save slice overrides** command, Lumberyard prompts you to choose which modifications or overrides to save. For a nested slice, you can also choose which slice in the hierarchy receives the overrides.

After you save overrides, Lumberyard automatically updates all instances of that slice. If any of those slice instances have unsaved overrides or modifications, it does not accept updates to those overrides. You can revert a slice's entities and properties to resync to the source slice.

> **Note**
> Lumberyard does not save the **Translate** value in the **Transform** component to the slice. This value determines an entity's position in the level and as such is excluded from affecting other slice instances.

Overriding

Changing any part of a slice, such as adding a component or modifying a property, results in an override or modification. This means that the slice instance is different from the source slice. Overrides to component properties are marked in a bold orange font in the Entity Inspector to distinguish it from properties that haven't changed.

If you save those overrides, they are saved to all the slice instances. This means they are no longer overrides and the font returns to normal. Overrides that you don't save to the slice are maintained for that instance even when other changes are received from the source slice.

You can also force an override for a property that you haven't yet modified, as well as revert overrides to an entity, component, or property. For more information, see Reverting and Forcing Overrides (p. 602).

Detaching

You can detach an entity within a slice. This removes it from that slice instance, and that entity no longer receives updates when changes are saved to the source slice.

You can also detach any slice instance. This means that it no longer receives any changes saved to the slice.

For more information, see Detaching Entities and Slice Instances (p. 601).

Nesting

When a slice instance is nested, that means that it is a child of another slice instance.

A nested slice instance shares a transform hierarchy with its parent. This means that you can move, rotate, and scale the topmost parent in any particular slice instance, and all its children move, rotate, or scale concurrently. You can, however, independently move, rotate, or scale its children without affecting the parent.

If you push a nested slice instance to its parent's source slice, then the nested slice instance inherits from both its original source slice and any source slices that own its parents.

Inheriting

Slice instances inherit data from all source slices in their hierarchy. When a slice instance inherits from multiple source slices, inheritance priority occurs from top to bottom in the outliner hierarchy.

A slice instance can lose its inheritance. If you modify a slice instance and then don't save those changes to the source slice, that slice instance loses its inheritance. To restore the inheritance, you must either reset the slice instance or save the changes.

Transform Hierarchy

A transform hierarchy defines movement, rotation, and scaling of entities in both the editor and at run time. A child entity shares the transform data of its parent. This means that you can move, rotate, or scale a parent, and its children always follow. You can, however, move, rotate, and scale a child independent of its parent.

Slice Hierarchy

A slice hierarchy defines a slice instance's relationship to its source slices. An entity inside of a nested slice can potentially inherit from multiple source slices. When this happens, slice data at the top of the slice hierarchy overrides slice data at the bottom of that hierarchy.

Creating a Slice

A slice can contain any number of entities that have their own transform hierarchy. The slice, however, must have a single transform root. This means that you can change the size and position of child entities without affecting the parent entity. But if you change the size or position of the parent entity, the child entities automatically adjust as well.

To create a slice

1. In the viewport or in the **Entity Outliner**, select one or more entities to include in the slice.

2. Right-click in the viewport or in the **Entity Outliner** and choose **Create slice**.
3. Save the slice file to the preferred location.

   You can create slices for many different purposes. Name your slices meaningfully and organize them into directories and subdirectories.

To save multiple entities in a slice, they must have a single transform root. If they don't, then a "Cannot Create Slice" message appears. To fix this issue, assign all entities in the slice to one parent entity.

To assign entities to a new parent entity

1. Open **Entity Outliner** (p. 437). Or use the viewport for this procedure.
2. Right-click in the viewport or in **Entity Outliner** and then choose **Create entity**.

   This creates a new, empty entity which will serve as the parent entity.
3. Do one of the following:

   - Within **Entity Outliner**, drag the child entities and drop them on top of the new parent entity.
   - Select one or more entities that are to be the child entities. In the **Entity Inspector**, within the **Transform** component, beside **Parent Entity**, click the picker (hand icon). Then click the parent entity in the viewport.
   - Select one or more entities that are to be the child entities. From the **Entity Outliner**, drag the parent entity into the **Entity Inspector** and drop it beneath the **Transform** component into the **Parent Entity** box.

Instantiating a Slice

Loading an instance of a slice in your level is called instantiating a slice.

To instantiate a slice

Do one of the following:

- In the viewport, right-click and choose **Instantiate slice**.
- Drag a slice asset from the **Asset Browser** (p. 179) into the viewport. The slice is instantiated at the drop location.

Creating Nested Slices

A nested slice contains instances of other slices. Nested slices can store their own overrides for component and entity properties. They can also contain their own entities.

For example, suppose that you have a slice called `wheel.slice`, composed of one wheel. You have another slice called `body.slice`, composed of an engine, seat, and frame. You create a new slice called `motorcycle.slice` and add two instances of the wheel slice and one instance of the body slice, and then save the changes. The body and wheel slices are now nested in the motorcycle slice.

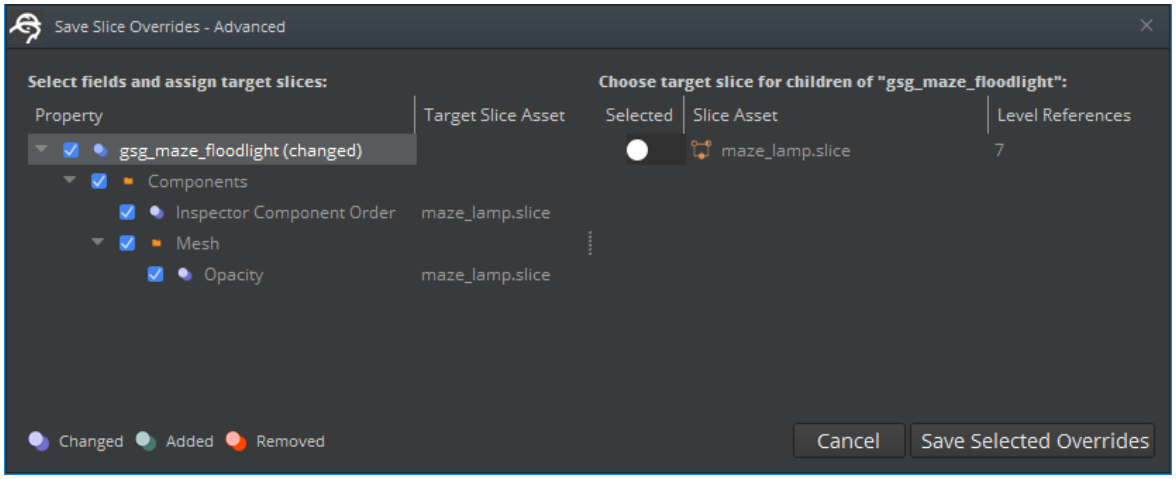
To create a nested slice

1. In the **Entity Outliner**, drag one or more slices onto another slice. The slices become child entities of the slice you dropped them on.
2. Right-click on the parent entity and choose **Create slice**. Save the slice with a new name.

Modifying a Slice and Saving Changes

If the currently selected entity is part of a slice instance, the **Entity Inspector** panel highlights in bold orange any properties that are different between the selected entity and its owning slice.

You can save overrides to the slice from the **Entity Inspector**, the **Entity Outliner**, or from the viewport. To do this, right-click the changed property or the entity itself, and choose **Save slice override**. To save multiple changes, select multiple entities in the **Entity Outliner** or in the viewport. From the **Slave Slice Overrides** dialog box, you can select the changes you want to save, and the target slices to save them to.



Because slices can be nested, you can save modifications to one or more levels of the nested slice hierarchy.

The **Entity Inspector** displays in orange text any component or property value that differs from the source slice.

Detaching Entities and Slice Instances

You can detach an entity that is inside of a slice. Detaching stops the entity from inheriting any changes from its owning slice.

To detach entities within a slice instance

1. In the **Entity Outliner**, select one or more entities within a slice instance.
2. Right-click the selection, and then choose **Detach slice entities**.

   Lumberyard displays a message that the entity will be converted into a nonslice entity and that the action cannot be undone.
3. Click **Detach**.

You can detach a slice instance. This stops that slice instance from inheriting any changes from the source slice.

To detach a slice instance

1. In the **Entity Outliner**, select any part of a slice.
2. Right-click the selection and then choose **Detach slice instance**.

   A message notifies you that the detached instance will no longer inherit any changes from its slice and that all entities in the slice instance will be converted into nonslice entities.

3. Click **Detach**.

Reverting and Forcing Overrides

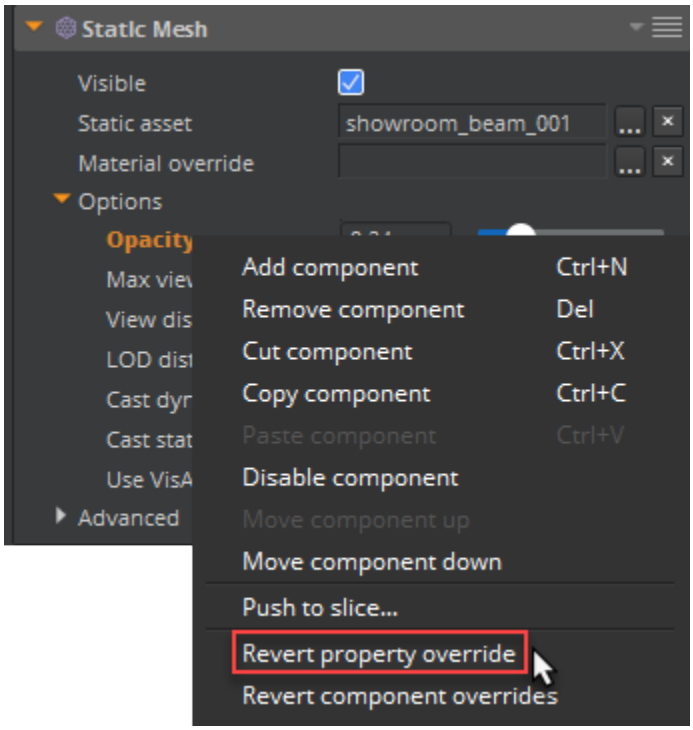
When you change any part of a slice instance, such as adding a component, removing a component, or modifying a component property, you create an override. Overrides to component properties are highlighted with bold orange text in the **Entity Inspector**. If you do not save the override, that property retains its override and does not inherit changes from the source slice.

Reverting Property Overrides

You can discard your modification so that the component property once again inherits changes from the source slice.

To revert property overrides

1. Right-click the property that you modified.
2. Choose **Revert property override**. The component property inherits changes from the source slice.



Reverting All Overrides to a Component

If you make multiple property modifications to a component in a slice, you can revert all overrides at once.

To revert all overrides on a component in a slice

1. In the **Entity Inspector**, right-click the component header.
2. Choose **Revert component overrides**.

Reverting All Overrides to an Entity

You can also revert all overrides to an entity in a slice. This is useful if you want to revert all modifications that you made to an entity, including adding or removing components and changing component properties.

To revert all overrides to an entity in a slice

1. In the **Entity Outliner**, right-click the entity header.
2. Choose **Revert overrides**.

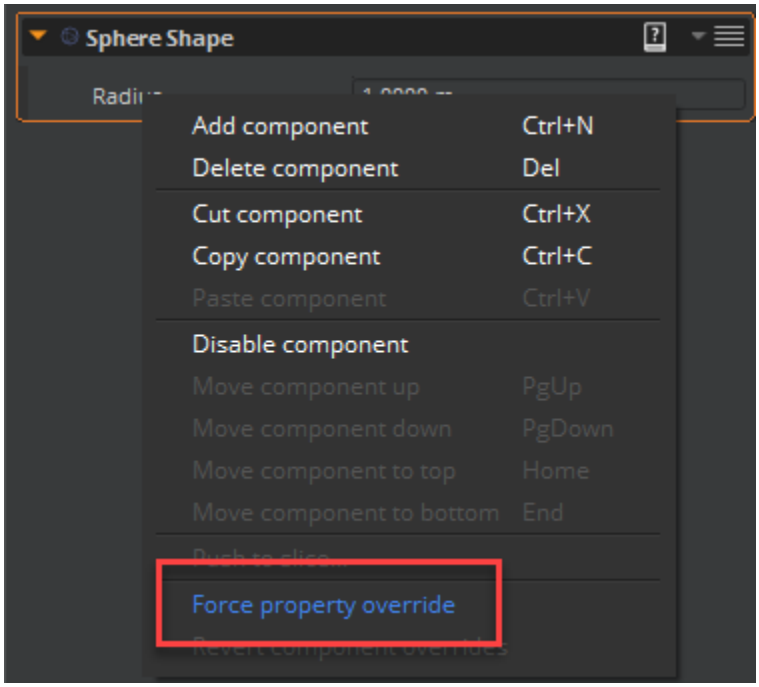
> **Note**
> Reverting overrides to a parent entity does not revert overrides to the child entities; it only affects modifications that you made directly to the parent entity.

Forcing a Property Override

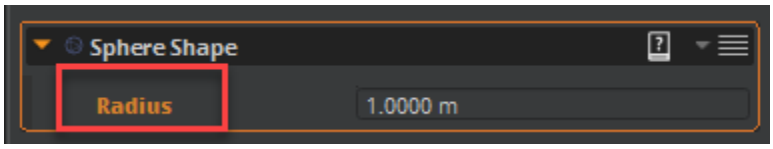
Unmodified properties inherit changes made to the source slice. If you don't want a property to inherit changes from the source slice, you can create an override to the property.

To force a property override

1. In the **Entity Inspector**, right-click an unchanged component property.
2. Choose **Force property override**.



The property override appears orange to indicate that it no longer inherits modifications from the slice.



Slice Reloading

Slices support run-time reloading. If a slice has changed for any reason—such as a data push operation, retrieval from source control, or manual edit—Lumberyard Editor reloads the slice asset. Lumberyard Editor also recalculates any slice instances that were affected by the change.

Converting Entities with the Legacy Converter

Component entity system is in preview release and is subject to change.

The Legacy Converter converts your legacy entities (CryEntities) to the new component entity system. Legacy features will eventually be removed from Lumberyard Editor. When you use the Legacy Converter, it tries to convert all legacy entities in your level. You cannot select which entities to convert or skip. Once an entity is converted, it cannot be converted back to a legacy entity.

Topics

Converting Your Legacy Entities

If your level has legacy entities, Lumberyard Editor detects them and prompts you to convert them.

To convert your legacy entities

1. Do any of the following to open the Legacy Converter:

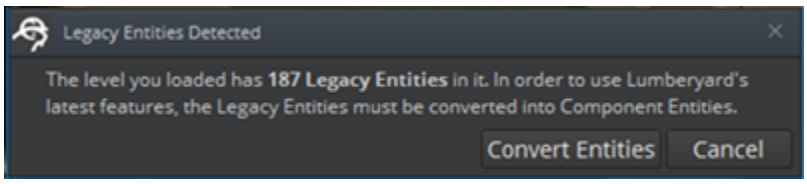
   - If the CryEntity Removal gem is enabled, and you load a level that contains legacy entities, the Legacy Converter opens by default.

   - Choose **File**, **Upgrade Legacy Entities**.

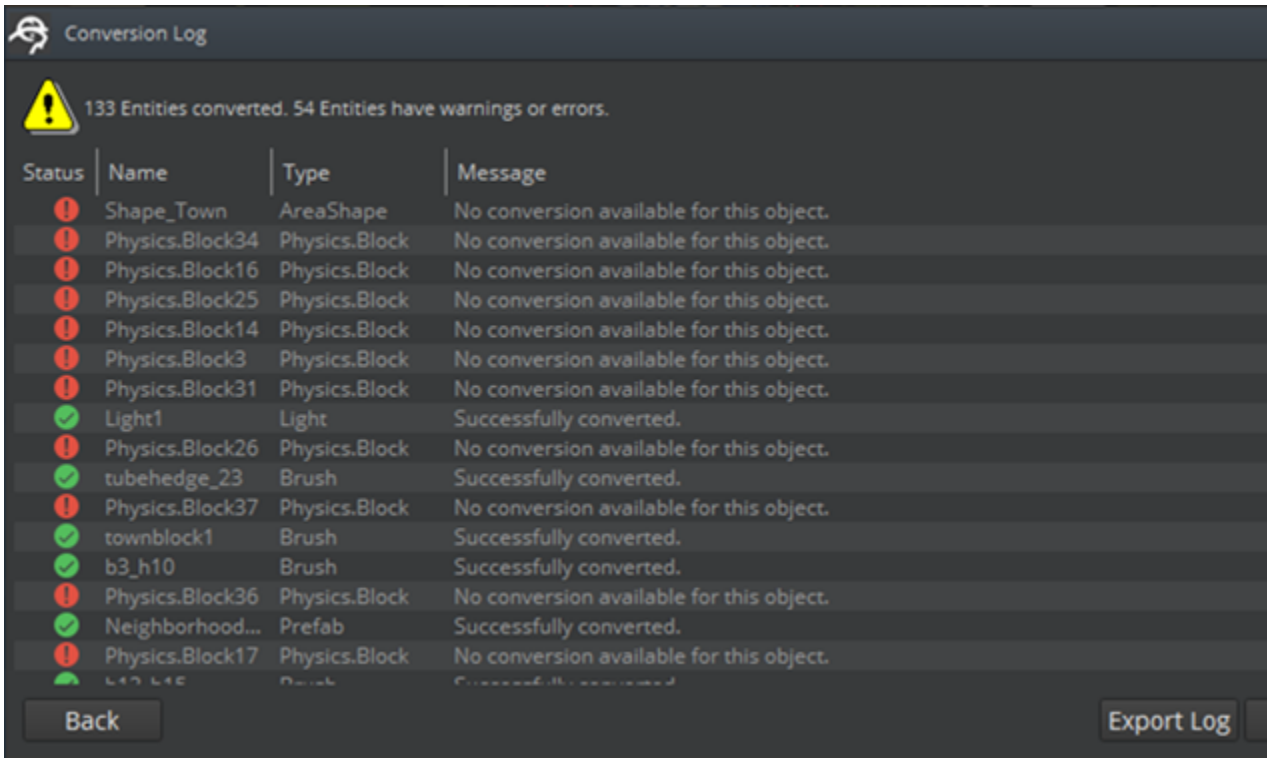
     Note
     The CryEntity Removal gem disables all legacy features in Lumberyard Editor. For more information, see the CryEntity Removal Gem (p. 1076).

2. The dialog box shows the number of legacy entities in your level. Choose **Convert Entities**.

   A progress bar appears that shows the number of entities that the Legacy Converter is processing. The conversion takes several seconds to complete.

3. After the conversion completes, the dialog box shows the number of converted and unconverted entities. See the **Status** and **Message** columns for more information. Choose **OK**.



Note
If the Legacy Converter can't convert an entity, a **Conversion Error** dialog box appears that shows the number of entities that were not converted. If the CryEntity Removal Gem is enabled, unconverted entities remain in your level but cannot be edited. For more information, choose **View Log**. You can also choose **Export Log** to save the conversion log as a `.csv` file.

List of Legacy Entity Conversions

The Legacy Converter converts each legacy entity to its corresponding component entity. For a list of legacy entities, see Entity Reference (p. 634).

Converted entities keep the same position in the viewport. For more information, see Using the Viewport (p. 85).

Note
Some legacy entities cannot be converted at this time.

See the following tables for the converted component entity:

Contents

Area Box Entity

An area box entity is converted to a component entity that is attached with a Box Shape (p. 546) component.

Settings in Area Box Entity	Converted Settings in Box Shape Component
Width	Dimensions : x
Length	Dimensions : y
Height	Dimensions : z

Area Sphere Entity

An area sphere entity is converted to a component entity that is attached with a Sphere Shape (p. 546) component.

Settings in Area Sphere Entity	Converted Settings in Sphere Shape Component
Radius	Radius

Brush, Geom, and Simple Entities

The brush, geom, and simple entities are converted to a component entity that is attached with a Mesh (p. 521) component or Skinned Mesh (p. 558) component, depending on the asset type.

Settings in Brush, Geom, and Simple Entities	Converted Settings in Mesh or Skinned Mesh Components
Geometry (set to `.cdf` files)	Character Definition (Skinned Mesh component)
Geometry (set to non `.cdf` file)	Static asset (Mesh component)

Camera Entity

A camera entity is converted to a component entity that is attached with a Camera (p. 467) component.

Settings in Camera Entity	Converted Settings in Camera Component
FOV	Field of View
NearZ	Near Clip Distance

Settings in Camera Entity	Converted Settings in Camera Component
FarZ	Far Clip Distance

Camera Target Entity

The camera target entity is converted to an empty component entity.

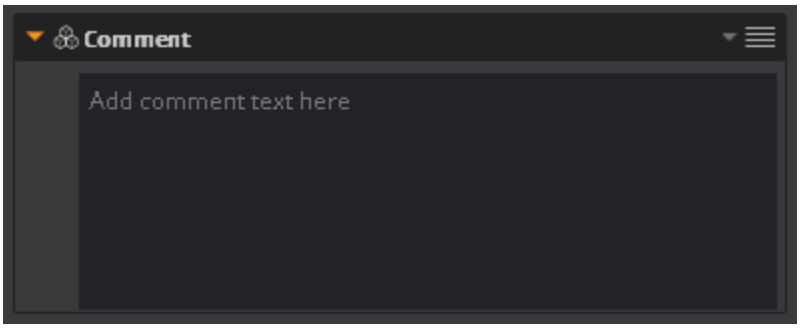
Comment Entity

A Comment entity is converted to a component entity that is attached with a

Component entity system is in preview release and is subject to change.

The **Comment** component allows you to add long-form text comments for component entities. When enabled, the **Comment** component displays a dialog box that expands based on the size of the comment that you enter. The following examples demonstrate how you can use the comment text box:

- Explain how the scripts or components on an entity interact with other scripts or components
- Describe how everything in a level ties together
- Send descriptions, instructions, or notes to team members



Comment Properties

The **Comment** component has the following property:

Comment text box

Stores the user comment for the component entity.

Default: None

Using the Comment Component

You can use this feature by adding the component to an entity in your level.

To use the Comment component

1. In Lumberyard Editor, right-click the viewport in your level, and click **Create entity**.

2. In the **Entity Inspector**, click **Add Component**.

3. Under **Editor**, click **Comment**.
4. In the **Entity Inspector**, under **Comment**, add comments for the component entity in the text box.

(p. 467) component.

Settings in Comment Entity	Converted Settings in Comment Component
Text	Comment

Environment Probe Entity

The environment probe entity is converted to a component entity that is attached with an Environment Probe (p. 499) component.

Settings in Environment Probe Entity	Converted Settings in Environment Probe Component
EnvironmentProbe Params : cubemap_resolution	Environment Probe Settings : Resolution
EnvironmentProbe Params : Outdoor Only	Options : Indoor only
EnvironmentProbe Params : ViewDistanceMultiplier	Options : View distance multiplier
EnvironmentProbe Properties : Active	On initially
EnvironmentProbe Properties : BoxSizeX	Environment Probe Settings : Area dimensions : X
EnvironmentProbe Properties : BoxSizeY	Environment Probe Settings : Area dimensions : Y
EnvironmentProbe Properties : BoxSizeZ	Environment Probe Settings : Area dimensions : Z
Color : Diffuse	General Settings : Color
Color : DiffuseMultiplier	General Settings : Diffuse multiplier
Color : SpecularMultiplier	General Settings : Specular multiplier
Options : AffectVolumetricFogOnly	Options : Volumetric fog only
Options : AttenuatioinFallofMax	Environment Probe Settings : Attenuation fall off
Options : IgnoresVisAreas	Options : Ignore vis areas
Options : Sort Priority	Environment Probe Settings : Sort priority
Options : VolumetricFog	Options : Volumetric fog
OptionsAdvanced : deferred_cubemap	Cubemap generation: Cubemap asset
Projection : BoxHeight	Environment Probe Settings : Box height
Projection : BoxLength	Environment Probe Settings : Box length
Projection : BoxProject	Environment Probe Settings : Box projected

Settings in Environment Probe Entity	Converted Settings in Environment Probe Component
Projection : BoxWidth	Environment Probe Settings : Box width

Decal Entity

A decal entity is converted to a component entity that is attached with a Decal (p. 498) component.

Settings in Decal Entity	Converted Settings in Decal Component
Entity : Mtl	Decal Settings : Material
Entity : MinSpec	Decal Settings : Minimum spec
Entity Params : ProjectType	Decal Settings : Projection type
Entity Params : Deferred	Decal Settings : Deferred
Entity Params : View Distance Multiplier	Options : View distance multiplier
Entity Params : SortPriority	Decal Settings : View distance multiplier
Entity Params : Projection Depth (Deferred)	Decal Settings : Depth

Groups and Layers

Groups and layers are converted to empty component entities with the same names.

Legacy entities that are nested under legacy groups or layers appear nested under the newly created component entities, which keep the same hierarchy.

If a legacy entity belongs to a layer and a group, the converted component entity appears under the converted group component entity. It also appears as a child of the converted layer component entity.

Example

1. You have the following legacy entities in your level:
   - LegacyEntityA, LegacyEntityB and LegacyEntityC are grouped under LegacyGroupA, so the hierarchy appears as: LegacyGroupA [LegacyEntityA, LegacyEntityB, LegacyEntityC]
   - LegacyEntityA belongs to LegacyLayerA, so the hierarchy appears as: LegacyLayerA [LegacyEntityA]
2. You use the Legacy Converter to convert your entities.
3. The converted entities have the following hierarchy:

   ComponentEntityLayerA [ComponentEntityGroupA[ComponentEntityA, ComponentEntityB, ComponentEntityC]]

Light Entity

A light entity with the **Planar Light** setting specified is converted to a component entity that is attached with a Area Light (p. 449) component.

A light entity with the **Projector** setting specified to **Texture** is converted to a component entity attached with a Projector Light (p. 532) component.

A light entity with neither the **Planar Light** or **Projector** setting specified to **Texture** is converted to a component entity attached with a Point Light (p. 529) component

Settings in Light Entity	Converted Settings Common in Area Light, Projector Light, and Point Light Components
Entity Params : Outdoor Only	Options : Indoor Only
Entity Params : ViewDistanceMultiplier	Options : View distance multiplier
Entity Params : HiddenInGame	Visible
Entity Properties : Active	On initially
Entity Properties : Color : Diffuse	General Settings : Color
Entity Properties : Color : DiffuseMultiplier	General Settings : Diffuse multiplier
Entity Properties : Color : SpecularMultiplier	General Settings : Specular multiplier
Entity Properties : Options : AffectsThisAreaOnly	Options : Affects this area only
Entity Properties : Options : AffectsVolumetricFogOnly	Options : Volumetric fog only
Entity Properties : Options : Ambient	General Settings : Ambient
Entity Properties : Options : IgnoresVisArea	Options : Ignore vis areas
Entity Properties : Options : VolumetricFog	Options : Volumetric fog
Entity Properties : Style : LightStyle	Animation : Style
Entity Properties : Style : AnimationSpeed	Animation : Speed
Entity Properties : Style : AnimationPhase	Animation : Phase
Entity Properties : Shadows : CastShadows (spec)	Options : Cast shadow spec
The following settings appear only in the **Entity Inspector** when the setting **Cast shadow spec** has any value, expect **Never**.	
Entity Properties : Shadows : ShadowBias	Shadow Settings : Shadow bias
Entity Properties : Shadows : ShadowResolutionScale	Shadow Settings : Shadow resolution scale
Entity Properties : Shadows : ShadowSlopeBias	Shadow Settings : Shadow slope bias
Entity Properties : Shadows : ShadowUpdateMinRadius	Shadow Settings : Shadow update radius
Entity Properties : Shadows : ShadowUpdateRatio	Shadow Settings : Shadow update ratio

Settings in Light Entity	Converted Settings in Projector Light Component
AttenuationBulbSize	Projector Light Settings : Attenuation bulb size

Settings in Light Entity	Converted Settings in Projector Light Component
Projector : ProjectNearPlane	Projector Light Settings : Near plane
Projector : ProjectorFov	Projector Light Settings : FOV
Projector : Texture	Projector Light Settings : Texture
Radius	Projector Light Settings : Max distance

Settings in Light Entity	Converted Settings in Area Light Component
Radius	Area Light Settings : Max distance
Shape : SourceDiameter	Area Light Settings : Area height
Shape : SourceWidth	Area Light Settings : Area width

Settings in Light Entity	Converted Settings in Point Light Component
AttenuatioinBulbSize	Point Light Settings : Attenuation bulb size
Radius	Point Light Settings : Max distance

Light Entity with Lens Flare

A light entity with a lens flare is converted to a component entity that is attached with a Lens Flare (p. 510) component and a light component.

The lens flare asset that is specified for a light entity is converted to the **Lens Flare** component settings: **Library** and **Lens flare**.

Settings in Light Entity	Converted Settings in Point Light Component
Entity Properties : Style : FlareFOV	Flare Settings : FOV

Particle Effect Entity

The particle effect entity is converted to a component entity that is attached with a Particle (p. 527) component.

Settings in Particle Effect Entity	Settings in Particle Component
ParticleEntity Properties : Prime	Spawn Settings : Pre-Roll
ParticleEntity Properties : CountScale	Spawn Settings : Count scale
ParticleEntity Properties : TimeScale	Spawn Settings : Time scale
ParticleEntity Properties : PulsePeriod	Spawn Settings : Pulse period

Settings in Particle Effect Entity	Settings in Particle Component
ParticleEntity Properties : Scale	Spawn Settings : Speed scale
ParticleEntity Properties : Strength	Spawn Settings : Strength curve time
Audio : EnableAudio	Audio Settings : Enable audio
Audio : Rtpc	Audio Settings : Audio RTPC

Proximity Trigger Entity

A proximity trigger entity is converted to a component entity attached with a Box Shape (p. 546) component, and a Trigger Area (p. 577) component.

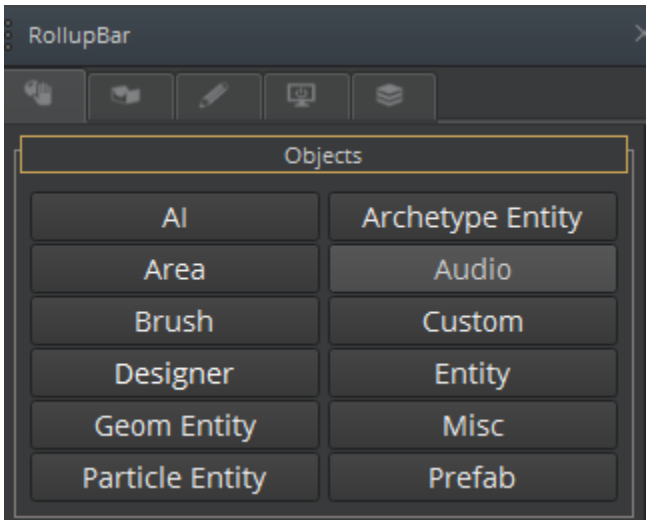
Settings in Proximity Trigger Entity	Converted Settings in Box Shape Component
Entity Properties : DimX	Dimensions : x
Entity Properties : DimY	Dimensions : y
Entity Properties : DimZ	Dimensions : z

Tag Point Entity

A tag point entity is converted to an empty component entity.

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**.

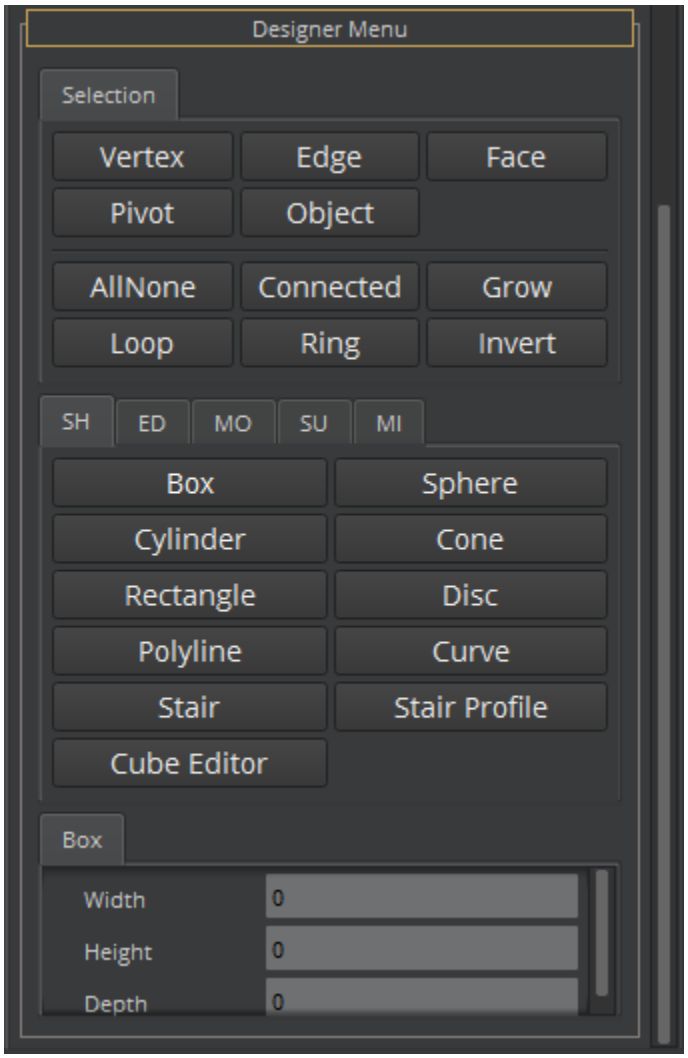


Note
The Component Entity System (p. 437) replaces the existing Entity system in Lumberyard at a future date.

Topics

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

The following parameter groups are available under the **Settings** panel, on the **Objects** tab, in the **Rollup Bar**.

CD Settings

The following parameters are available on the **CD** tab under 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 under the **Settings** panel.

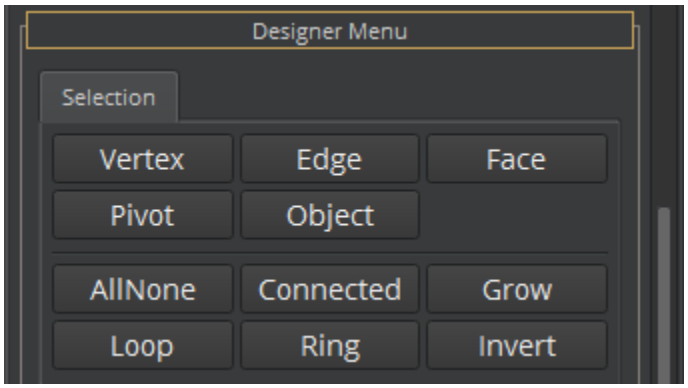
Object Parameters

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.

Property	Description
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

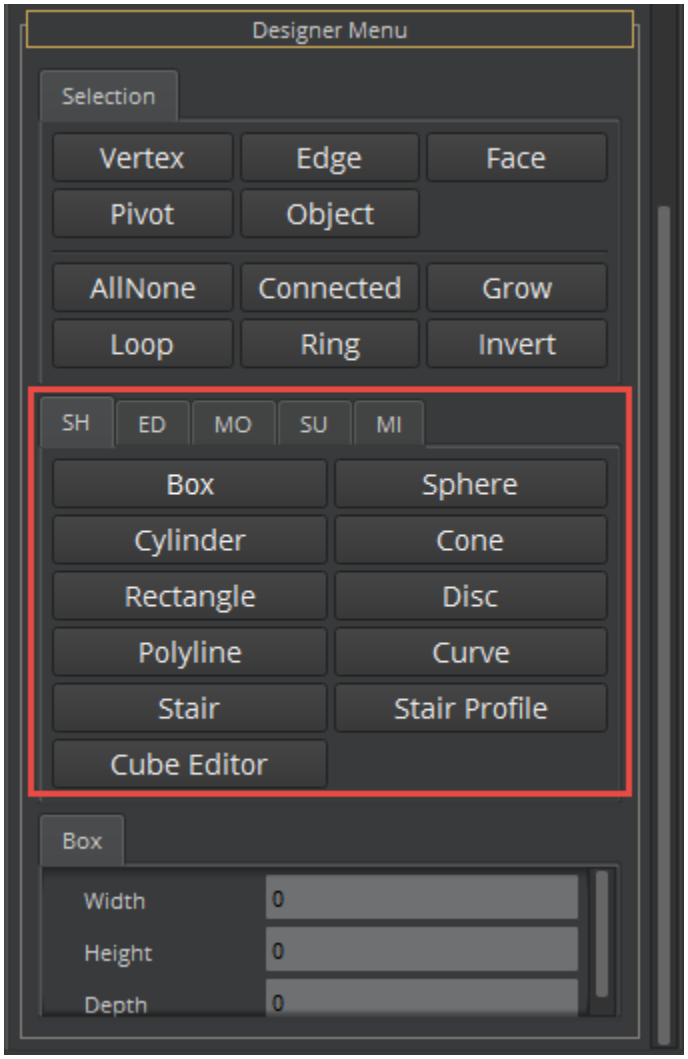
Use to select sequence edges that are not connected but on the opposite side to each other. You can also select serial-connected quad faces in a direction that is perpendicular to the direction that the selected two faces set.

Vertex, Edge, Face

Use to select and move vertices, edges, and faces. You can select muliple 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

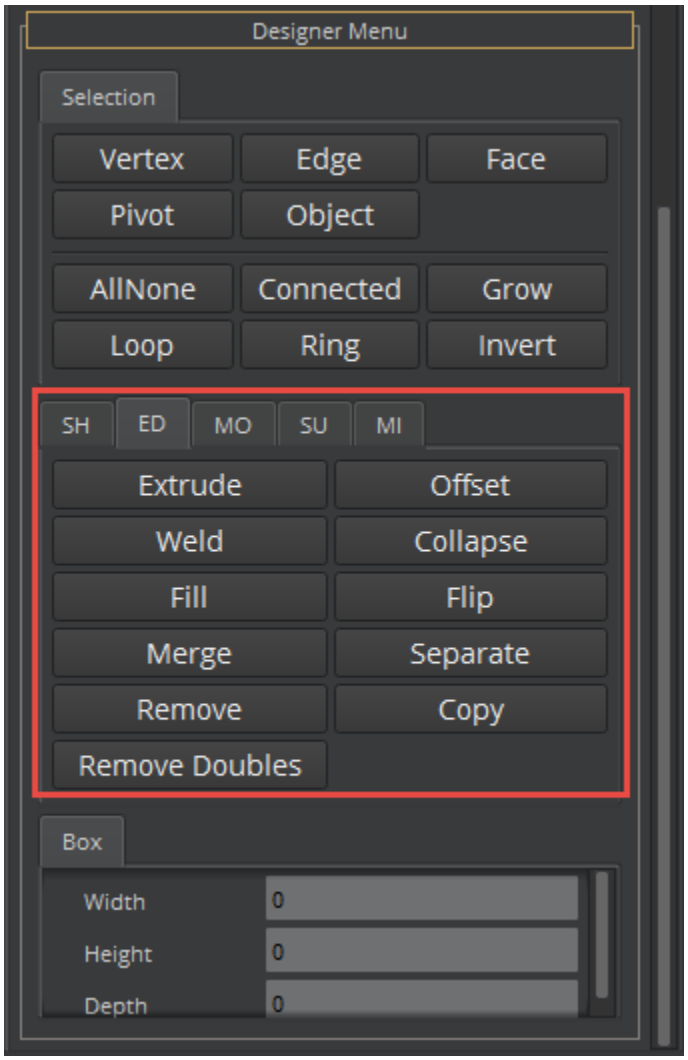
Used to draw a stair profile on a surface, which can be pulled using the **Extrude** function to be a stair. You can adjust the **Step Rise** value.

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

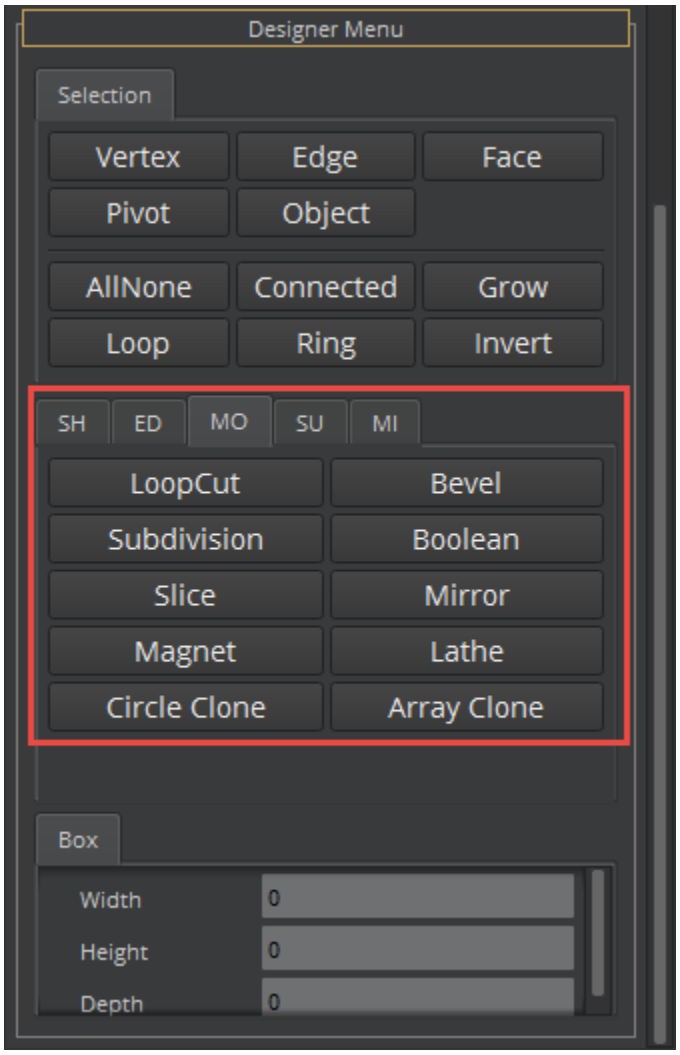
Used to separate two or more objects.

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:

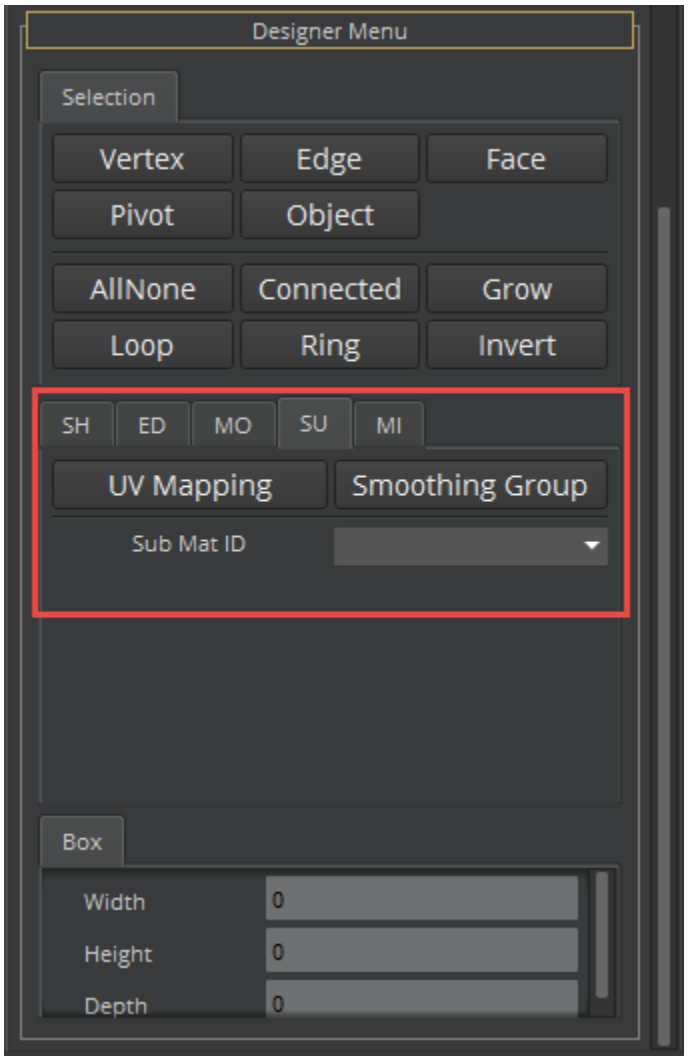
- **Apply** – Splits a mesh by a mirror plane and copies the half part to the other part and then starts the mirror editing.
- **Invert** – Invert a direction of the mirror plane.
- **Center Pivot** – Moves the pivot position to the center of the bounding box.
- **Align X**, **Align Y**, and **Align 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.

Property	Description
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 lees than the threshold angle.
Threshold Angle	Used to set the angle in degrees

UV Mapping

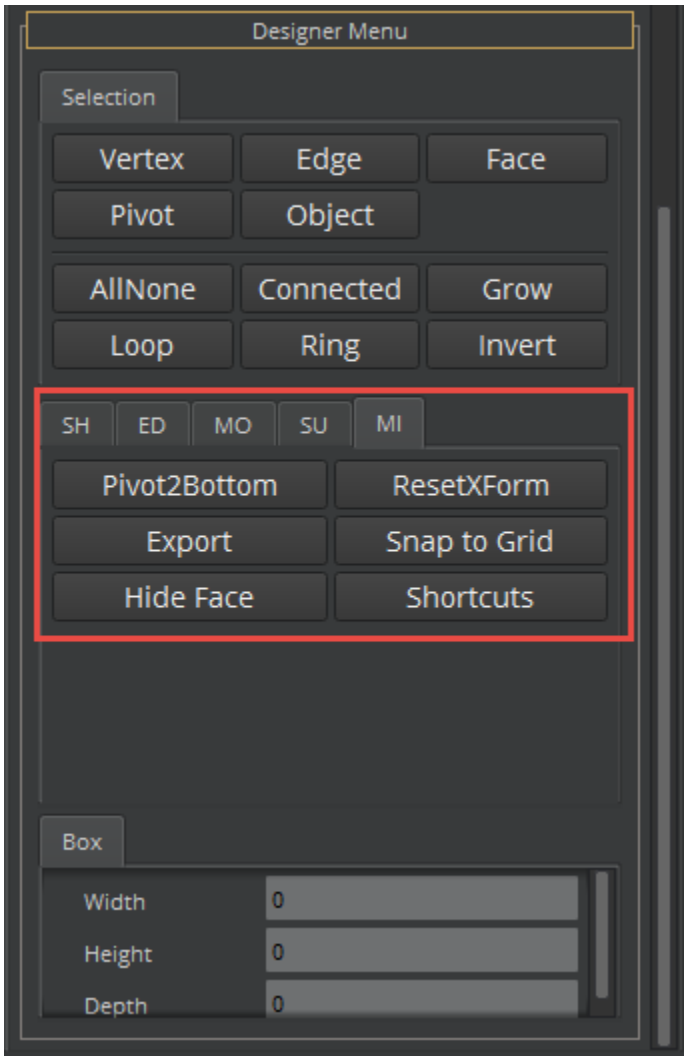
Materials can be assigned to each face differently and you can manipulate the UV coordinates using this tool.

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.
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 check boxes are selected.

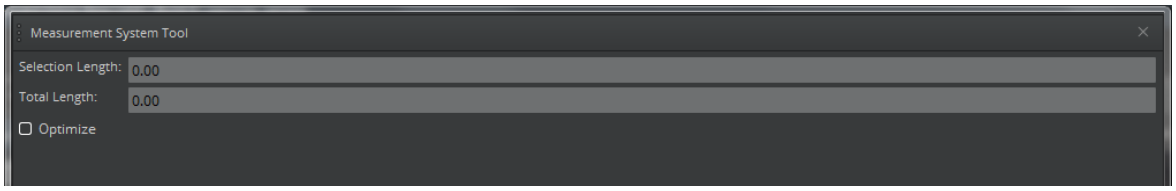
Shortcuts

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 **Tools**, **Other**, **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. 1178) that are selectable, visible, and not frozen.

To open Object Selector

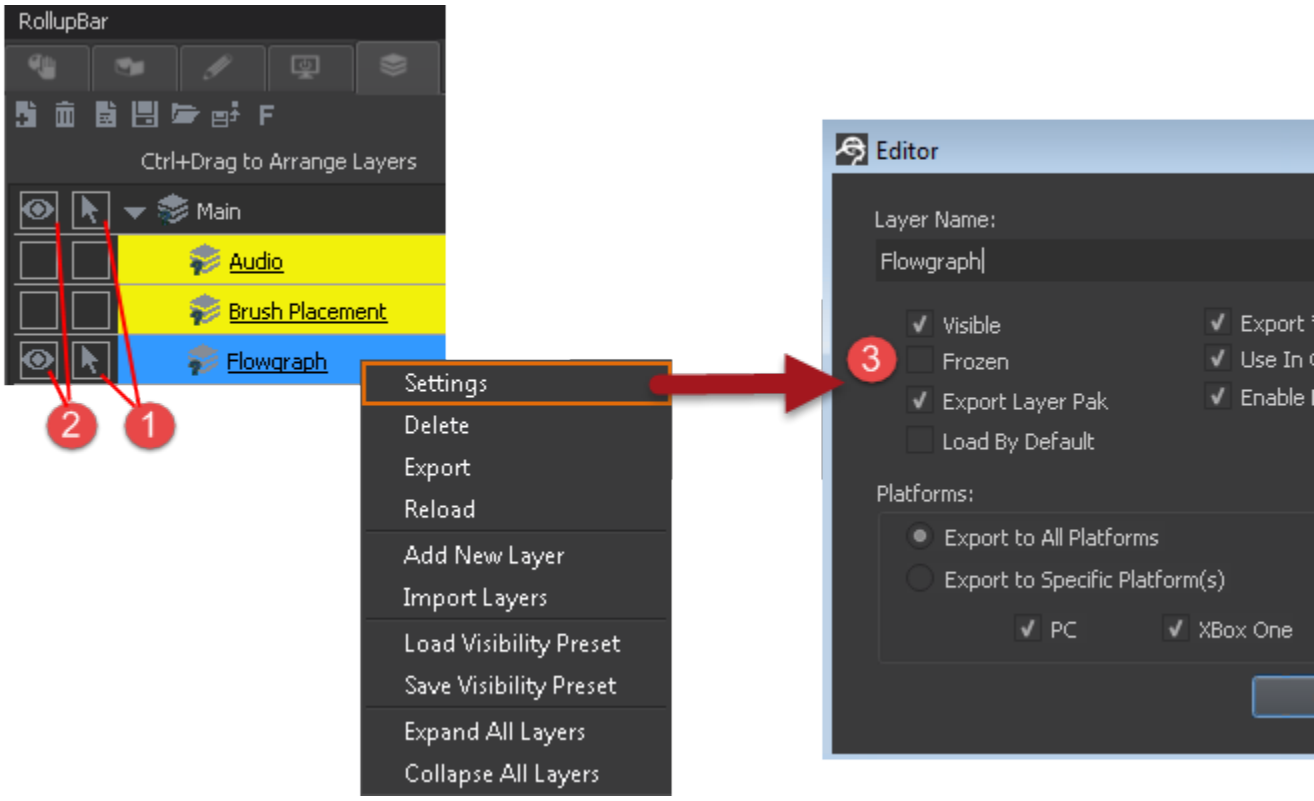
Do one of the following:

- On the main menu, click **Tools**, **Object Selector**.
- Press **Ctrl+T**.
- In 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. 1178) 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. 625) 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.
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. 1856) 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	• Ship • Shield • Asteroid

Parameter	Description
Ignore	• Ship • Shield • Asteroid
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.
NoAmnbShadowCaster	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 enablng **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.
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

Use the **Entity** pane to modify basic entity properties, such as the name of your object or the currently selected layer. You can type a new name for your object in the first text box in the pane.

Depending on their type, certain entities will have color schemes applied by default. Click the color next to the text box to open the color editor.

Click the layers icon to open the layer window and place your object in the appropriate layer. The layer text box displays the name of the layer that is currently selected.

Standard Parameters

Property	Description
Area	Increases or decreases the radius of the onscreen object placement helper.

Property	Description
Mtl button	Opens the material window for you to pick a material to apply to the object. The custom material path displays in the Mtl text box.
MinSpec	Sets the value at which the selected object appears in game detail settings.

Entity Parameters

Use the **Entity Params** panel to modify common entity parameters. These parameters enable effects that are added to an object, and toggle options such as hiding the object in-game.

Entity Parameters

Property	Description
Outdoor Only	Prevents an object from rendering when inside a VisArea.
Cast Shadow MinSpec	Casts a shadow on the selected quality setting and above.
LodRatio	Defines the distance from the current camera position when different level of detail models for the object are used.
ViewDistRatio	Defines the distance from the current camera position when the object will be rendered.
HiddenInGame	Prevents an object from displaying in game mode. This is useful for debugging or prototyping.
Receive Wind	Influences an object by any wind that is set up in your level.
RenderNearest	Eliminates z-buffer artifacts when rendering in first person view.
NoStaticDecals	When set to true, prevents decals from rendering on the object.
Created Through Pool	Used for memory optimization for AI entities.

Scripting and Flow Graph Entity Parameters

Use this pane to modify the parameters that are 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. Clicking the **>** button displays more options for this file.
Reload Script	Implements any changes made to the script. This is useful for reviewing particle effects because reloading reactivates the changes.
Entity Archetype	Displays the entity name on the button, if the entity is an archetype entity. Opens the archetype in the Database View tool.

Property	Description
Create	Creates a new flow graph.
List	Lists the flow graphs that are associated with the selected entity.
Remove	Removes a flow graph that was created for the entity.
Sequence	Displays the name of the sequence, if the entity is used in a Track View sequence. Opens the sequence in the Track View window.
Save State	Saves the physical state of a selected entity (when AI/Physics in enabled). This is useful for placing physical props realistically around your level without manually rotating and aligning their positions.
Clear State	Clears any saved physical state.

Entity Links

Use this pane to view the entities that are linked to the main entity. Each entity can link to multiple entities. To create an entity link, you create a dynamic link that can be referenced in Lua script.

To create a link, click **Pick Target** and select the desired entity. 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 to open a menu with additional commands.

Entity Links Parameters

Property	Description
Change Target Entity	Changes the entity associated with a link.
Rename Link	Renames the selected link.
Delete Link	Deletes the selected link.
Pick New Target	Provides the same functionality as the **Pick Target** button.

Entity Events

Use this pane to edit and run the script behind objects. When **AI/Physics** is enabled, you can test the effect of any changes you have made to the entity script.

Enable **AI/Physics** 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

Property	Description
Send	After choosing an input or output event, click **Send** to test the effect. For example, an input event called OnKill might kill an entity and OnSpawn might spawn an entity back to life.
Methods	Displays a list of executable methods.
Run	Displays a list of executable methods.
Goto	Deprecated
Add	Deprecated

Attached Entities

Use this pane to create links to other objects in the perspective viewport. This pane is visible for certain entities.

Attached Entity Parameters

Property	Description
Pick	Links two selected objects. You will see the link in the viewport and the object name in the target window.
Remove	Removes a link between two objects.
Select	Selects an object from the target window. You can also double-click the object name in the target window to select the object.

Shape Parameters

Use this pane to edit the effect area for a shape and create links to other objects in the viewport. This pane is 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 reverses the AI path. The arrow onscreen points 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	Flattens the shape and all other points to the height of the selected point.

Property	Description
Pick	Links a shape to an object. You will see the link in the viewport and 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. You can also double-click the object name in the target window to select the object.

Entity Reference

The following is a complete list of the entities in the Entity system.

Topics

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:

- AI Anchor
- 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

AI Anchor

An AI Anchor 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).
AreaId	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 rendered grid cells.
Agent_width	When Render_voxel_grid is enabled this determines the height along the x-axis of the rendered grid cells.
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: • Flight • Free 2D • Road • Smart Object • Triangular • Unset • Volume • Waypoint 3D Surface • Waypoint Human
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).
AreaId	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 rendered grid cells.
Agent_width	When Render_voxel_grid is enabled this determines the height along the x-axis of the rendered grid cells.
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	• Waypoint • Hide • Sec Hide • Entry/Exit • Exit-only
Nav Type	• 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 perminant 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.
Height	Specifies how high the shape area should be (0 means infinite height).
AreaId	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 rendered grid cells.
Agent_width	When Render_voxel_grid is enabled this determines the height along the x-axis of the rendered grid cells.
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.

Parameter	Description
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 Navigation (p. 523).

Parameters

Parameter	Description
Height	The height of the navigation area.
DisplayFilled	Just for visibility in the editor this option defines if the area should be rendered as filled or not.

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.
ViewDistanceMultiplier	Defines how far from the current camera position, 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
AreaId	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.

Parameter	Description
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.
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
AreaId	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.
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).
AreaId	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.

Parameter	Description
Agent_height	When Render_voxel_grid is enabled this determines the height along the y-axis of the rendered grid cells.
Agent_width	When Render_voxel_grid is enabled this determines the height along the x-axis of the rendered grid cells.
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.

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, click **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. 1394).

Parameter Table

Parameter	Description
Width	Specifies how wide the entity is.
Height	Specifies how high the shape area should be (0 means infinite height).
AreaId	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 rendered grid cells.
Agent_width	When Render_voxel_grid is enabled this determines the height along the x-axis of the rendered grid cells.
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.

Parameter	Description
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.
Advanced	
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

This topic describes how to use legacy audio entities in the Rollup Bar. These audio entities will be deprecated and replaced with audio component entities in a future release. For more information, see Component Reference (p. 446).

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

This topic describes how to use the legacy audio trigger spot in the Rollup Bar. This audio entity will be deprecated and replaced with audio trigger component entity in a future release. For more information, see Audio Trigger (p. 462).

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.
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	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.

Properties	Description
	• **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

This topic describes how to use the legacy audio area entity in the Rollup Bar. This audio entity will be deprecated and replaced with audio area environment component entity in a future release. For more information, see Audio Area Environment (p. 456).

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 flow graph node is starting to output values.
SoundObstructionType	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. • **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

This topic describes how to use the legacy audio area ambience entity in the Rollup Bar. This audio entity will be deprecated and replaced with audio environment component entity in a future release. For more information, see Audio Environment (p. 457).

Audio Area Ambience entities are used to set up ambiences 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	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. • **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

This topic describes how to use the legacy audio area random entity in the Rollup Bar. This audio entity will be deprecated and replaced with audio area environment component entity in a future release. For more information, see Audio Area Environment (p. 456).

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	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. • **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.

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: • 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.
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.
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 Entity

You can use the (legacy) **Camera** entity to place a camera in your level. In addition to placing a camera, you can also place a 'LookAt' target, which automatically determines where the camera faces. For more information about this legacy camera entity, see Rendering Cameras (p. 1450).

To use the newer **Camera** components, see Camera (p. 467) and Camera Rig (p. 468).

To add a render camera to your level

1. In the **Rollup Bar**, on the **Objects** tab, click **Misc**, **Camera**.
2. Hover your pointer in your level, and then click to position the camera.
3. To create a ‘LookAt’ target (determines what the camera looks at), when placing the camera in your level, hover your pointer to determine its position, then click and drag to the 'LookAt' target, or the view that you want your camera to see. A 'LookAt' target is created where the drag ended. This object is named with your original camera name, with **Target** appended.
4. Adjust the values of the following parameters as needed.

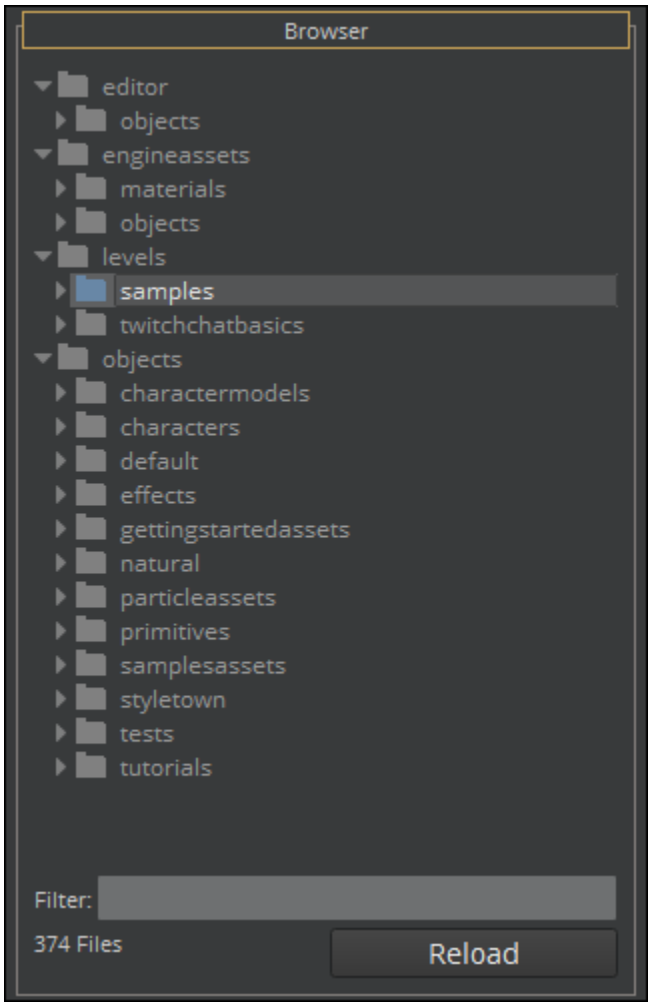
Parameters

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
Random Seed	Apply some random variation to the noise

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 Attentuation 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.

Property	Description
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.
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 flow graph, track view, etc., so you shouldn't need to use this option. It's 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. For example, High won't work on Low/Medium. To ensure shadows are always cast, set the this to 'Low Spec'. This setting is often confused as a quality setting for the shadows; however, 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.

Property	Description
ShadowSlopeBias	Allows you to adjust the gradient (slope-based) bias used to compute the shadow bias.
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.
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.

Parameter	Description
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.

Lightning Arc Entity

You can use the Lightning Arc entity to creates 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 lighting 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.

Property	Description
maxNumStrikes	Hard limit on the number of beam segments that can be generated regardless of previous parameters.
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 strikeNumSegments* 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 strikeNumSegments*strikeNumPoint*2. Because maxNumStrikes 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 strikeNumSegments*strikeNumPoint*2*maxNumStrike. However, remember that every time the LightningArc 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 a 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	exture 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 beamTexTiling 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.

Port	Description
Strike	Allows to manually trigger a spark even when the entity is disabled. This allows synchronization of the spark effect with other level events.
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.

Parameter	Description
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.
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.

Property	Description
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 multiples 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.
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 a Strength Over Emitter Life 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	

Property	Description
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.
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 mass = density x

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.
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 g_ec_removeThreshold 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 GameRulesClientServer.cpp 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 @use_object.
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.

Property	Description
PushableByPlayers	It true, the player pushes the object by walking/running into it.
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	

Property	Description
x_max	If set greater than x_min, the constraint only rotates the object along its x-axis within the defined angle.
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)

Property	Description
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.
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	A cheaper alternative/addition to water resistance (applies uniform damping when in water). 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. 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.
water_density	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. (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). Decreasing per-object effective water density allows such objects to sink while still having large-volume physical geometry. 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).

Property	Description
water_resistance	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). (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. 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. 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.

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 omni-directionally 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.
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. 1394).

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. 1394).

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.
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.

Parameter	Description
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

Parameter	Description
	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.
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.

Parameter	Description
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.
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.

Property	Description
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.

Property	Description
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.
MultiplayerOptions	
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).
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.

Property	Description
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.

Getting Started with Script Canvas

Script Canvas is in preview release and is subject to change.

Script Canvas is a new visual scripting environment for Amazon Lumberyard that you can use to create game logic and behaviors. Integrated with Lumberyard's component entity system and built on AzCore, **Script Canvas** is designed to use the serialization, reflection, modularization, and EBus messaging systems. **Script Canvas** offers an approachable and easy-to-read environment to author behaviors using the same framework as Lua and C++. With its visual scripting options, you can use **Script Canvas** to create game logic without needing to know how to code.

For an example level with **Script Canvas**, see Script Canvas Basic Sample (p. 1482).

For information on programmatically working with nodes, see Script Canvas Programmer's Guide in the *Amazon Lumberyard Developer Guide*.

Topics

Script Canvas Concepts and Terms

The following concepts and terms are used in **Script Canvas**:

Script

A script is a collection of nodes, node properties, and node connections that, when combined, create a visual script.

Node

Nodes represent the data, events, and actions that you use to create logic and behaviors in **Script Canvas**.

See the following node types:

Event nodes

Event nodes subscribe to event bus (EBus) handlers to listen for events to occur. Examples include entering a trigger area, colliding with an object, turning off a light, and when the game ticks.

For more information about using the EBus interface, see Event Bus (EBus) in the *Amazon Lumberyard Developer Guide*.

Action nodes

Action nodes are used to get or send data across an EBus. Examples of action nodes include getting the mass of an entity, turning on a light, setting the text of a UI element, and playing an animation.

For more information about using the EBus interface, see Event Bus (EBus) in the *Amazon Lumberyard Developer Guide*.

Variable and data nodes

Variable and data nodes represents the custom data that can be required to build game logic. You can use these nodes to make counters, store entity references, specify a direction, define a color, and so on. Variable nodes are added to a script to declare and initialize them. Use get and set nodes to retrieve or set the variable's value.

The following are the commonly used data types in **Script Canvas**:

- Boolean
- Color
- Entity
- Number
- String
- Transform
- Vector 2/3/4

Logic

Logic nodes include comparison and timing operations. You can use logic nodes to check whether two values are equal, control the execution of nodes, delay the execution of a node for a specific amount of time, and more.

Math

Math nodes enable math operations, such as arithmetic, geometry, algebra, calculus and so on.

Debugging

Debugging nodes verify whether a script is functioning as expected. You can use debugging nodes to print data to the console or viewport and check for errors. These nodes pass logic flow, but do not execute in release builds.

User defined

You can build your own nodes for your project's specific needs.

Node Palette

The **Node Palette** contains a searchable list of nodes. By default, the palette is docked to the left of the **Script Canvas** editor.

To display the Node Palette

1. From Lumberyard Editor, choose **Tools**, **Script Canvas**.
2. In the **Script Canvas** window, do one of the following:
   - Right-click the canvas.
   - Choose **View**, **Node Palette**.
   - Press **Ctrl+Shift+L**.

Node Inspector

The **Node Inspector** shows the properties of the node. You can edit each property in the inspector or directly in the node. By default, this window doesn't appear in the editor.

To display the Node Inspector

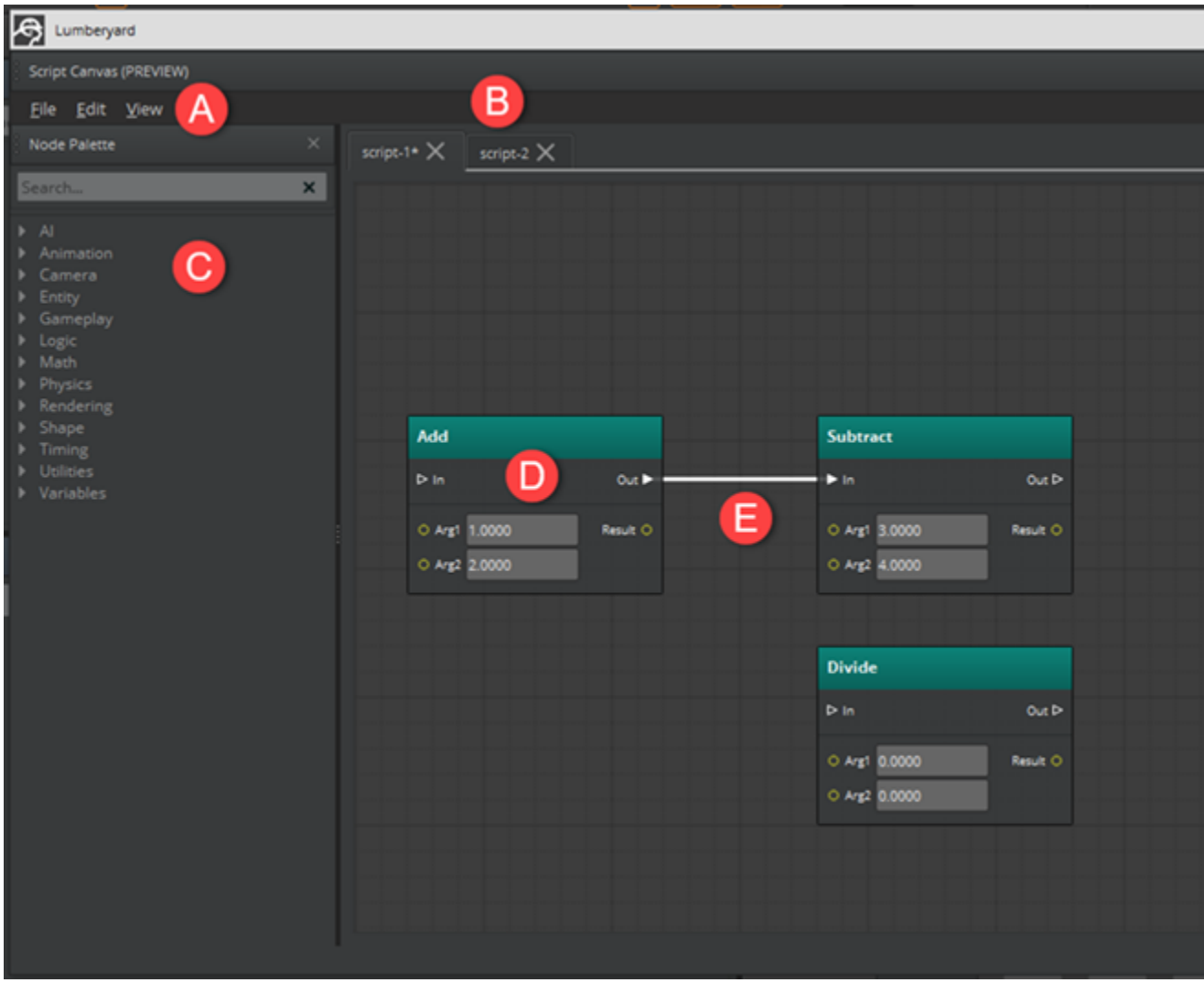
1. Open **Script Canvas** (p. 684).
2. Do one of the following:
   - Click **View**, **Node Inspector**.
   - Press **Ctrl+Shift+I**.

Script Canvas Editor Interface

You can open the **Script Canvas** editor from Lumberyard Editor.

To open the Script Canvas editor

1. In Lumberyard Editor, choose **Tools**, **Script Canvas**.
2. Choose **File**, **New Script** or drag a node from the **Node Palette** and drop it on the canvas.



In the **Script Canvas** editor, you can do the following:

A. Use the menu bar to do the following:
   - Create, save, and open your scripts.
   - Cut, copy, or undo actions.
   - Change the **Script Canvas** editor view.

B. Use the tabs to switch between scripts.

C. In the **Node Palette**, you can search for nodes.

D. In the **Script Canvas** canvas, you can drag a node from the **Node Palette**.

E. On a node, you can specify values for the arguments.

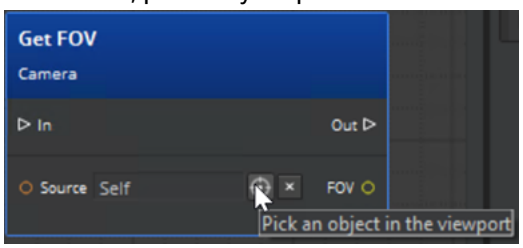
F. Drag to connect the input pin of a node to an output pin of another node. This line creates a connection between the nodes.

Referencing Entities for Nodes

Nodes can contain an entity property. These properties tell the node which entity to affect. By default, many nodes reference **Self**, the entity to which the **Script Canvas** component attaches the current script. You can reference a specific entity other than self.

To reference entities for a node

1. From the **Node Palette**, locate the node that you want to add to your script and drag it to the canvas..
2. In the node, position your pointer over the entity property and then click the target icon.



3. In the Lumberyard Editor viewport or the **Entity Outliner**, select the entity that you want to reference.
4. To clear the entity, position your pointer over the entity property and then click the **x** icon.
5. To reset a property back to self, right-double-click the entity property and choose **Set to Self**.

Pins and Connections

Connections define the cause and effect from one node to the next. Some connections determine the order of execution while others pass data from one node to the next.

Contents

Making Connections

You can make connections only between pins of the same type. For example, logic connections can only be made between logic pins, and data pins can only be made between data pins of the same type. You can't create connections between incompatible pins, such as a logic and a data pin.

To make a connection

1. In the **Script Canvas** canvas, drag from the input pin of one node to an output pin of another node. This creates a connection line between the two pins.
2. To move a connection from one pin to another, drag the end of a line from one pin and drop it on another pin.

   To delete a connection, right-click and choose **Delete**. You can also hold **Alt** and then click the connection to delete it.

Pin and Connection Types

Lumberyard has two main pin and connection types:

Logic pins and connections

Logic connections determine the order of execution. A **Script Canvas** script runs when the entity that is attached to the script is activated. Nodes are connected from the left side. After they finish running, they activate nodes that are connected to the right side of their logic flow.

An outgoing logic pin with multiple connections executes a logic branch in sequence. The execution sequence is determined by the order the connections were made, from earliest to newest. If a specific execution order is required, we recommend that you specify the sequence order through a single logic flow.

An incoming logic pin with multiple connections runs each time that the logic flow triggers the node. If a node is triggered by three different nodes in a script, the node runs three times.

Data pins and connections

Data connections enable scripts to read and write data between nodes. Data is read from the right side of one node and then set on the left side of another node.

Adding Scripts to Entities

Like Lua scripts, you can add a script to a level or slice is by adding a **Script Canvas** component to an entity. The **Script Canvas** component adds Script Canvas capabilities to the entity and provides a field to specify a script assignment. You can specify a script with the **Script Canvas** component.

For more information, see the Script Canvas (p. 545) component.

For more information about adding a component to an entity, see Adding Components to an Entity (p. 593).

Other Script Canvas Menus and Tools

Script Canvas editor has additional menus and tools:

File menu

The **File** menu offers basic script management. From the **File** menu, you can create a script, open a script, open a recently edited script, save a script, and close the **Script Canvas** editor.

You can open multiple scripts at once. Each opened script is tabbed to the script panel. From this tab, you can rearrange the tab order and close an opened tab.

Node Outliner

View and search for nodes in the script. You can select a node in the outliner, which then selects the node in the script. Double-click the node in the **Node Outliner** centers the view to that specific node.

Node Inspector

Shows the properties for the selected node. You can specify the entities and values for the node.

Note
The preview release of **Script Canvas** doesn't include a debugger tool.

Script Canvas Tutorials

Script Canvas is in preview release and is subject to change.

Use the following **Script Canvas** tutorials to learn about the Lumberyard visual scripting system.

These tutorials assume that Samples Project is set as the default project. For more information about changing your default project, see Launching a Game Project (p. 32).

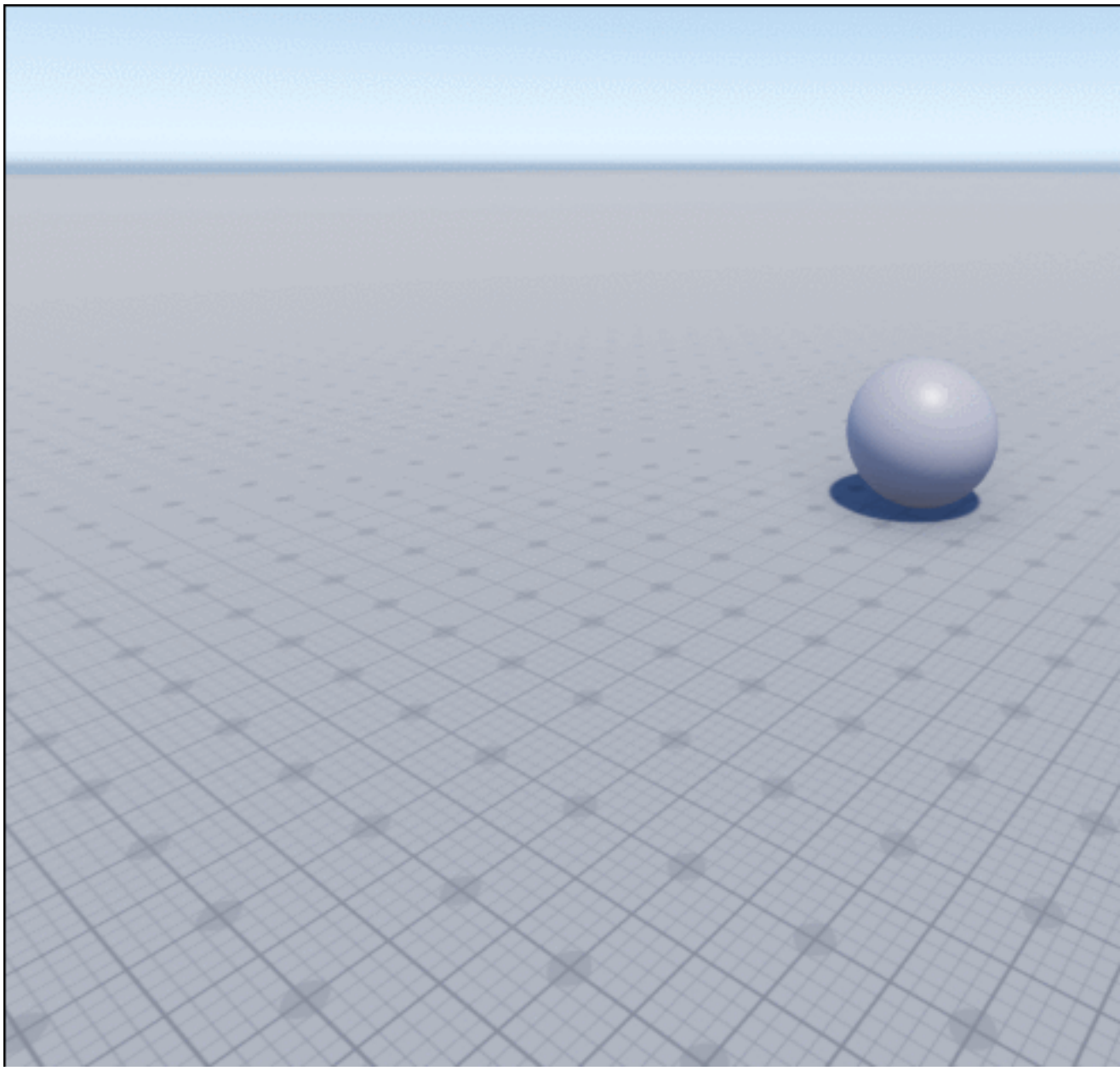
Topics

Script Canvas Tutorial: Creating a Controllable Entity

Script Canvas is in preview release and is subject to change.

This tutorial walks you through the steps to create a simple sphere that you can control with keyboard input. In addition to node basics, input, movement, and logging, you learn the following key concepts:

- Adding nodes
- Creating execution and data connections
- Adding event nodes
- Adding action nodes



Step 1: Create a Level

Before you create a sphere that you can control with keyboard input, you need to create a level.

To create a level

1. In Lumberyard Editor, choose **File**, **New**.
2. In the **New Level** dialog box, type **`firstscriptcanvas`** and then click **OK**.
3. In the **Generate Terrain Texture** dialog box, choose **512x512** and then click **OK**.

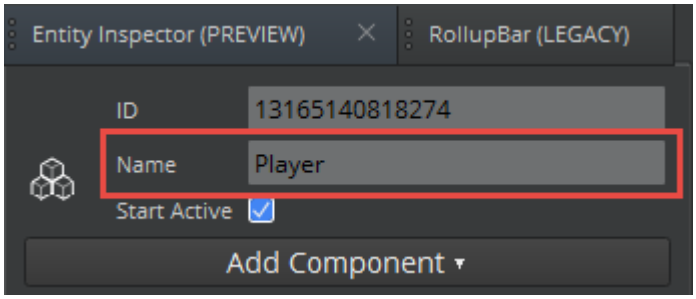
Step 2: Create an Entity and Add Components to Create a Controllable Entity

To create a sphere that you can control with keyboard input, you create an entity and add the **Mesh**, **Rigid Body Physics**, **Mesh Collider**, and **Input** components. You also create an input mapping that converts WASD keyboard input into movement in the x and y directions.

To create a controllable entity

1. Create an entity by right-clicking anywhere in the **Perspective** viewport and choosing **Create entity**.
2. In the **Entity Inspector**, do the following:

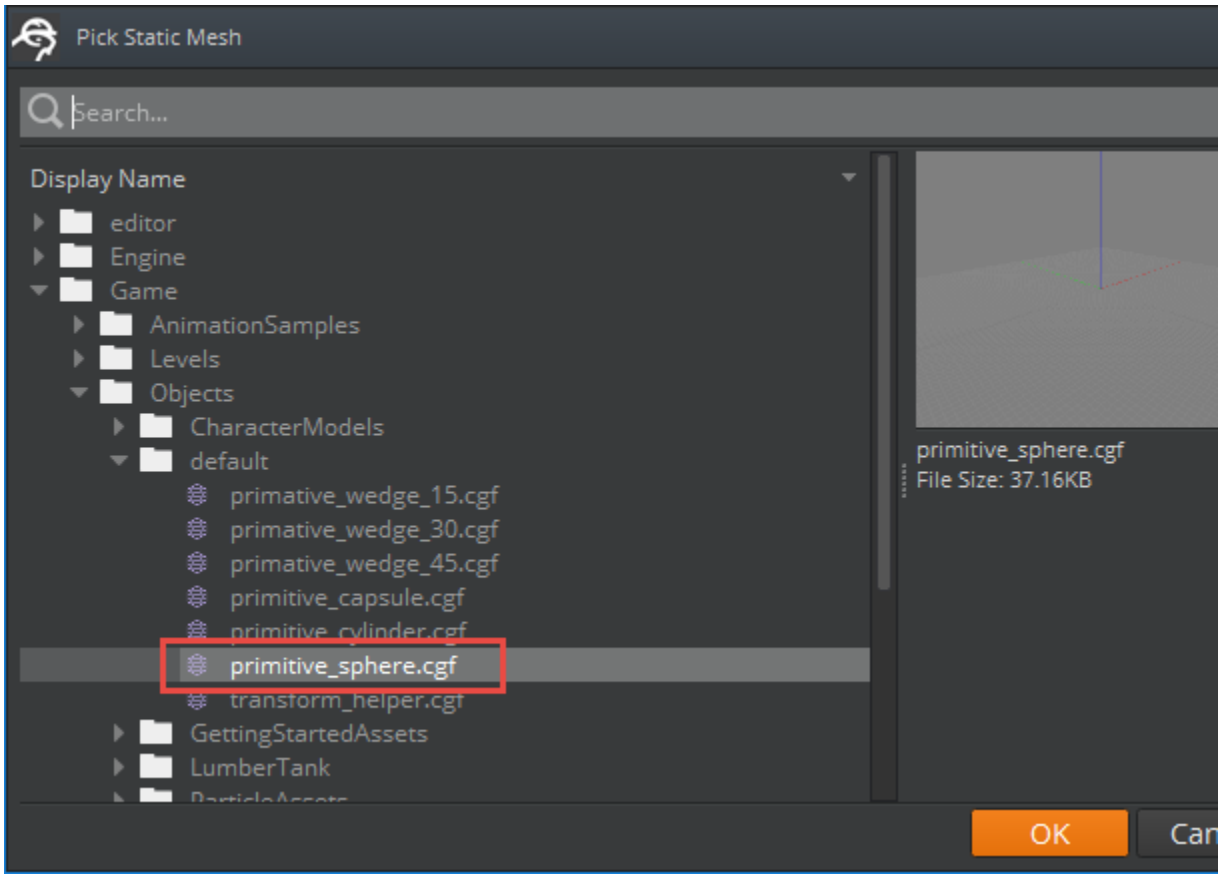
   a. For **Name**, type **`Player`**.

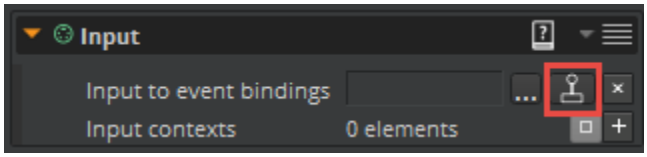
   b. Click **Add Component**.

   c. Under **Rendering**, choose **Mesh**.

   d. Under **Mesh**, for **Mesh asset**, click the browse (**...**) button and navigate to the `\Game\Objects\default` directory. Select `primitive_sphere.cgf` and then click **OK**.



3. In the viewport, use the **Move** tool (p. 79) to select the z-axis and move the entity off the ground.
4. In the **Entity Inspector**, click **Add Component**. Under **Physics**, click **Rigid Body Physics**.
5. Click **Add Component** again. Under **Physics**, click **Mesh Collider**.
6. Click **Add Component** again. Under **Gameplay**, choose **Input**.
7. In the **Entity Inspector**, under **Input**, click the **Input Bindings Editor** icon.



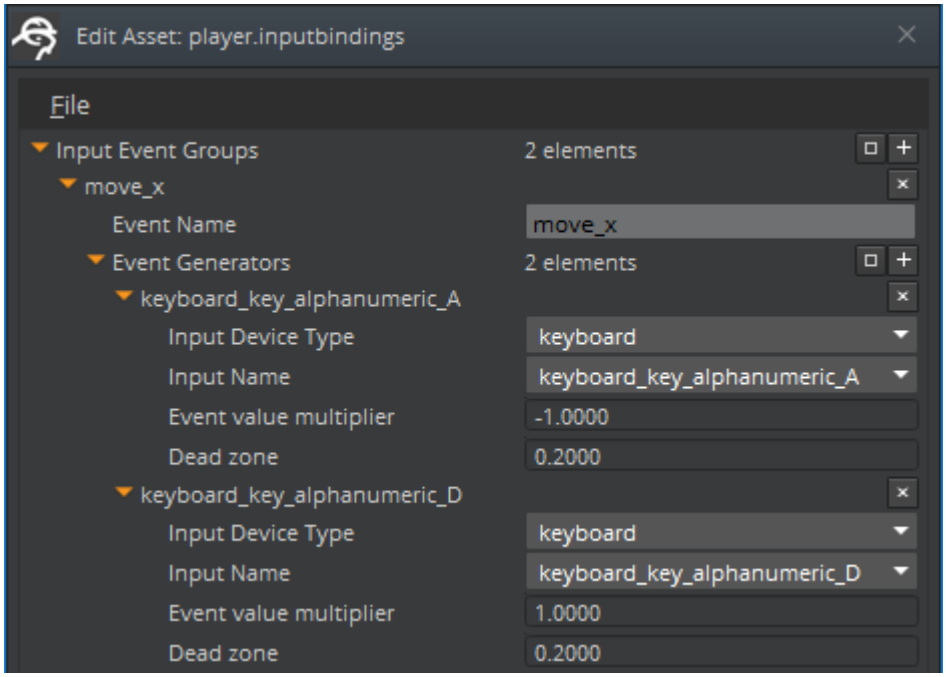
8. In the **Input Bindings Editor**, choose **File**, **Create New Asset**, **Input to Event Bindings Asset**. For **File name**, type `player` and then click **Save**.
9. In the **Asset Browser**, navigate to the `player.inputbindings` file. Right-click the file and choose **Open**.
10. In the **Edit Asset** window, do the following:

    a. For **Input Event Groups**, click **+** to add a new input event group.
    b. Expand the input event group. For **Event Name**, type `move_x`.
    c. For **Event Generators**, click **+** to add an event generator.
    d. In the **Class to create** dialog box, click **OK** to add an input class.
    e. Navigate to **move_x**, **Event Generators**, **gamepad_button_a**. For **Input Device Type**, select **keyboard**.
    f. For **Input Name**, choose **keyboard_key_alphanumeric_A**.

g. For **Event value multiplier**, type `-1`.

h. Repeat steps C – E. Then, for **Input Name**, select **keyboard_key_alphanumeric_D**. For **Event value multiplier**, use the default value of 1.

i. Verify that your settings appear as shown in the following image:



11. In the **Edit Asset** window, do the following:

a. For **Input Event Groups**, click **+** to add a new input event group.

b. Expand the input event group. For **Event Name**, type `move_y`.

c. For **Event Generators**, click **+** to add an event generator.

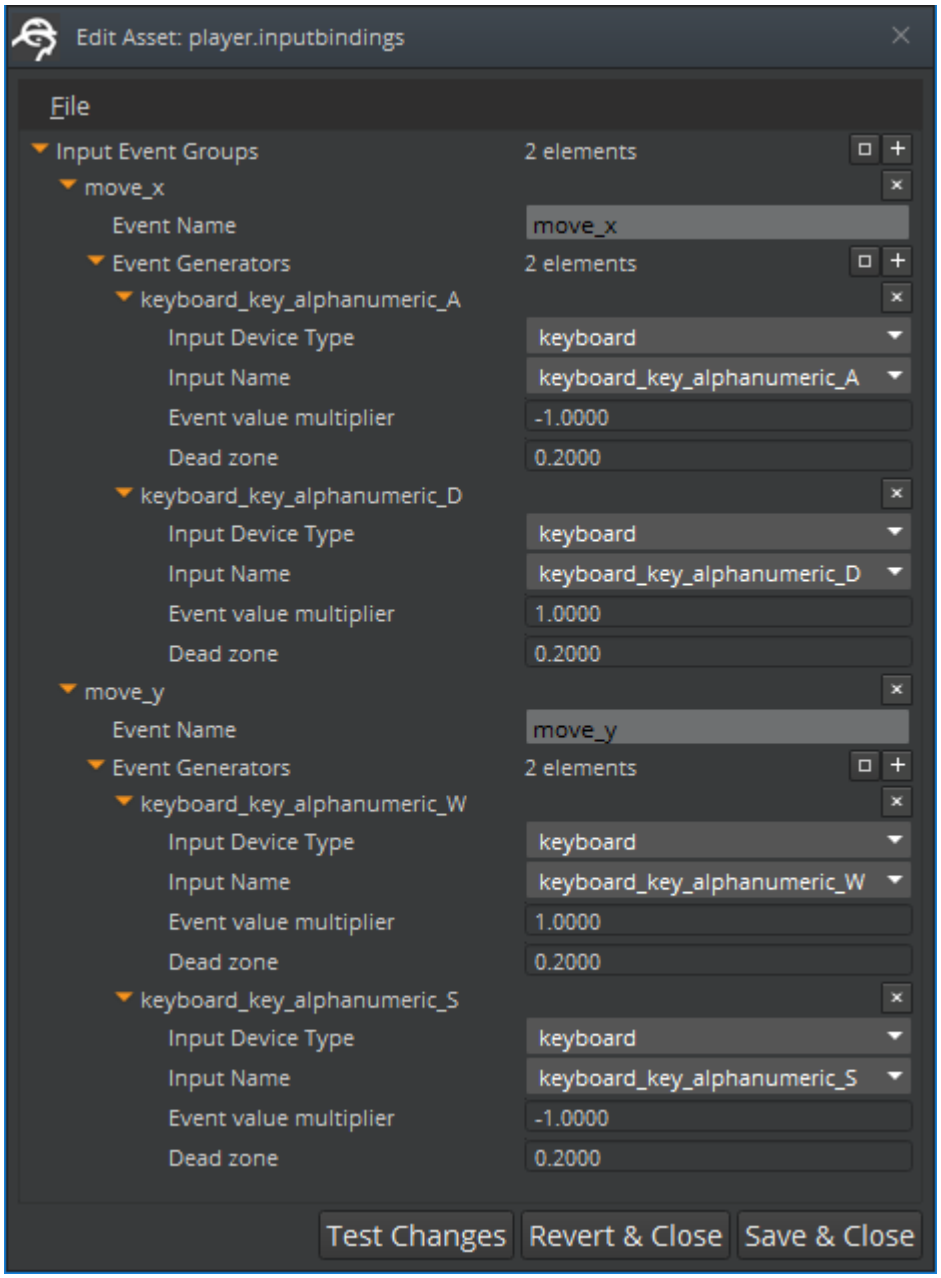
d. In the **Class to create** dialog box, click **OK** to add an input class.

e. Navigate to **move_y**, **Event Generators**, **gamepad_button_a**. For **Input Device Type**, select **keyboard**.

f. For **Input Name**, select **keyboard_key_alphanumeric_W**. For **Event value multiplier**, use the default value of 1.

g. Repeat steps C – E. Then, for **Input Name**, select **keyboard_key_alphanumeric_S**. For **Event value multiplier**, type `-1`.

h. Verify that your settings appear as shown in the following image:



   i. Click **Save & Close**.

12. In the **Entity Inspector**, under **Input**, click the browse (**...**) button and navigate to the `player.inputbindings` file. Select `player.inputbindings` and then click **OK**.

Step 3: Create an Input Script Using Script Canvas

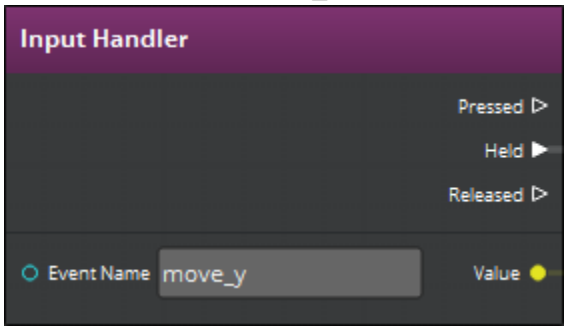
Now that you've set up a sphere with physics and input mapping, you can create your first script using **Script Canvas**.

To create an input script

1. In Lumberyard Editor, choose **Tools**, **Script Canvas**.

2. In the **Script Canvas** editor, choose **File**, **New Script**.
3. After the new canvas loads, choose **File**, **Save As**.
4. In the **Save As** dialog box, for **File name**, type **`player.scriptcanvas`** and then click **Save**.
5. In the **Node Palette**, type **`input`** in the search box.

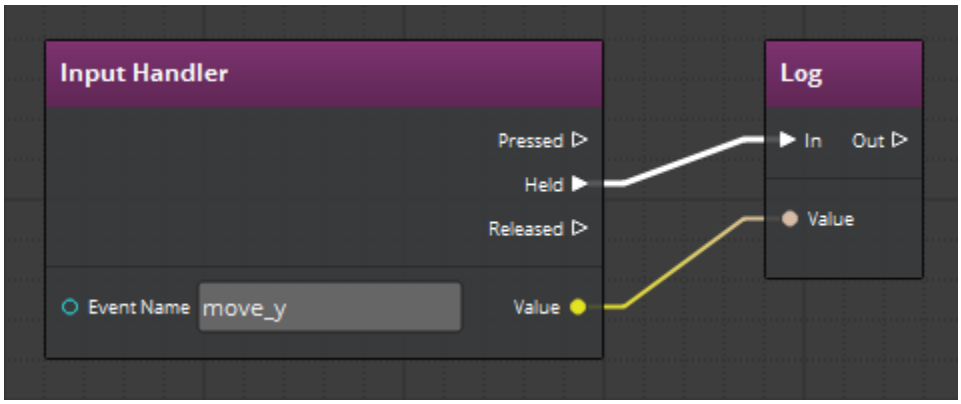
   Note
   You can also access the list of nodes by right-clicking in an empty area on the canvas.
6. Drag **Input Handler** from the **Node Palette** to the canvas. **Input Handler** is an event node. When an event occurs, the event node sends a message to the script.
7. For **Event Name**, type **`move_y`**. This tells the node to listen for the input event.



8. In the **Node Palette**, type **`log`** in the search box.
9. Drag **Log** from the **Node Palette** to the canvas. **Log** is an action node. When you execute an action node, it completes actions such as request data, set data, manipulate data, and trigger functions. Action nodes also print data to the editor console, allowing you to check values as the script is being executed.
10. Select the **Held** pin (white arrow) for **Input Handler** and drag to connect it to the **In** pin for **Log**. This connection tells the **Log** node to execute after the input handler receives a held event.

    Note
    Node execution always flows from the left side to the right side of a node. For more information, see Pins and Connections (p. 686).
11. Select the **Value** pin (yellow circle) for **Input Handler** and drag to connect it to the **Value** pin for **Log**. This connection tells the **Input Handler** node to pass the input event value from the input handler to the first argument on the **Log** node.

    Note
    Data is always input on the left side of a node and output from the right side of the node. For more information, see Pins and Connections (p. 686).
12. Verify that your script looks like the following image:



Step 4: Assign the Script to Your Entity and Test the Script

The script that you made outputs the value of the input event. Now you can assign the script to your entity and test the script.

To assign and test your script

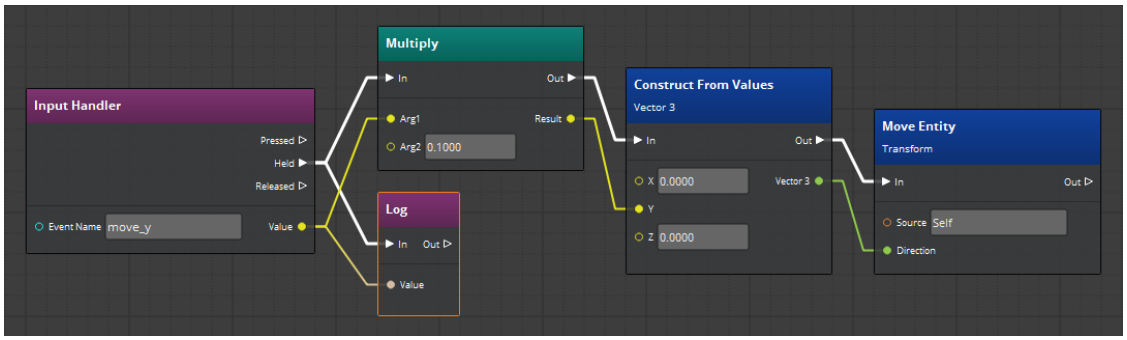
1. In the **Script Canvas** editor, choose **File**, **Save**. You can also press **Ctrl+S**.
2. In Lumberyard Editor, select the entity that you created in Step 2: Create an Entity and Add Components to Create a Controllable Entity (p. 690).
3. In the **Entity Inspector**, click **Add Component**.
4. Under **Scripting**, choose **Script Canvas**.
5. Under **Script Canvas**, click the browse (**...**) button and navigate to the `player.scriptcanvas` file. Choose `player.scriptcanvas` and then click **OK**.
6. Press **Ctrl+G** to enter game mode.
7. To trigger your input event, press **W** and then press **S**. The **Console** pane in Lumberyard Editor outputs the value of the input event when the `move_y` event is triggered. In this case, the **Console** pane outputs 1 when you press **W** and outputs -1 when you press **S**.
8. When you are done testing your script, press **Esc**.

Step 5: Create a Script to Move Your Sphere

Now that you've successfully created your first script, you can add nodes to the script to move your sphere. You also modify the input event value to control the movement speed.

To move your sphere

1. In the **Script Canvas** editor, in the **Node Palette**, type **`multiply`** in the search box.
2. Drag **Multiply** from the **Node Palette** to the canvas.
3. Select the **Held** pin for **Input Handler** and drag to connect it to the **In** pin for **Multiply**. This connection executes the **Multiply** node after the input handler receives a held event.
4. Select the **Value** pin for **Input Handler** and drag to connect it to the **Arg1** pin for **Multiply**.
5. In the **Multiply** node, for **Arg2**, type **`0.1`**. This smaller value for the input event throttles the movement speed.
6. In the **Node Palette**, type **`move`** in the search box.
7. Drag **Move Entity** from the **Node Palette** to the canvas.
8. In the **Node Palette**, type **`vector 3`** in the search box.
9. Drag **Construct from Values** from the **Node Palette** to the canvas. You use this node to build a vector 3 from a number.
10. Select the **Out** pin for **Multiply** and drag to connect it to the **In** pin for **Construct from Values**.
11. Select the **Result** pin for **Multiply** and drag to connect it to the **Y** pin for **Construct from Values**. You only need a y direction, so use the default value (0) for **X** and **Z**.
12. Select the **Out** pin for **Construct from Values** and drag to connect it to the **In** pin for **Move Entity**.
13. Select the **Vector 3** pin for **Construct from Values** and drag to connect it to the **Direction** pin for **Move Entity**.
14. Verify that your script looks like the following image:



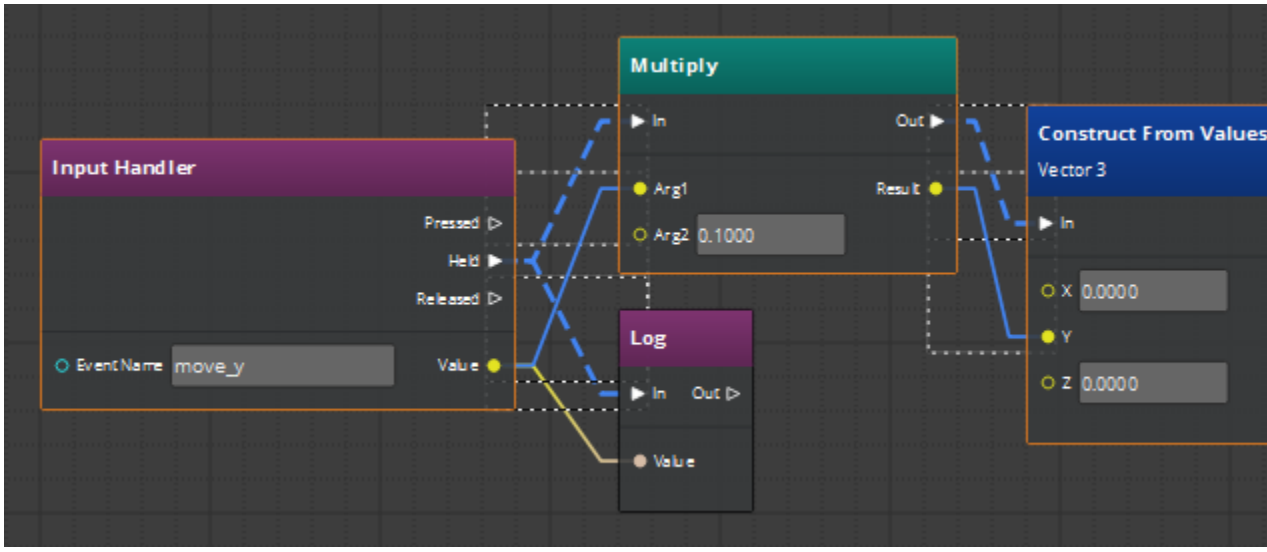
15. Choose **File**, **Save**. You can also press **Ctrl+S**.
16. In Lumberyard Editor, press **Ctrl+G** to enter game mode and test your script.
17. To move the sphere forward, press **W**. To move the sphere backwards, press **S**.
18. When you are done testing your script, press **Esc**.

Step 6: Add Movement on the X-Axis for Your Sphere

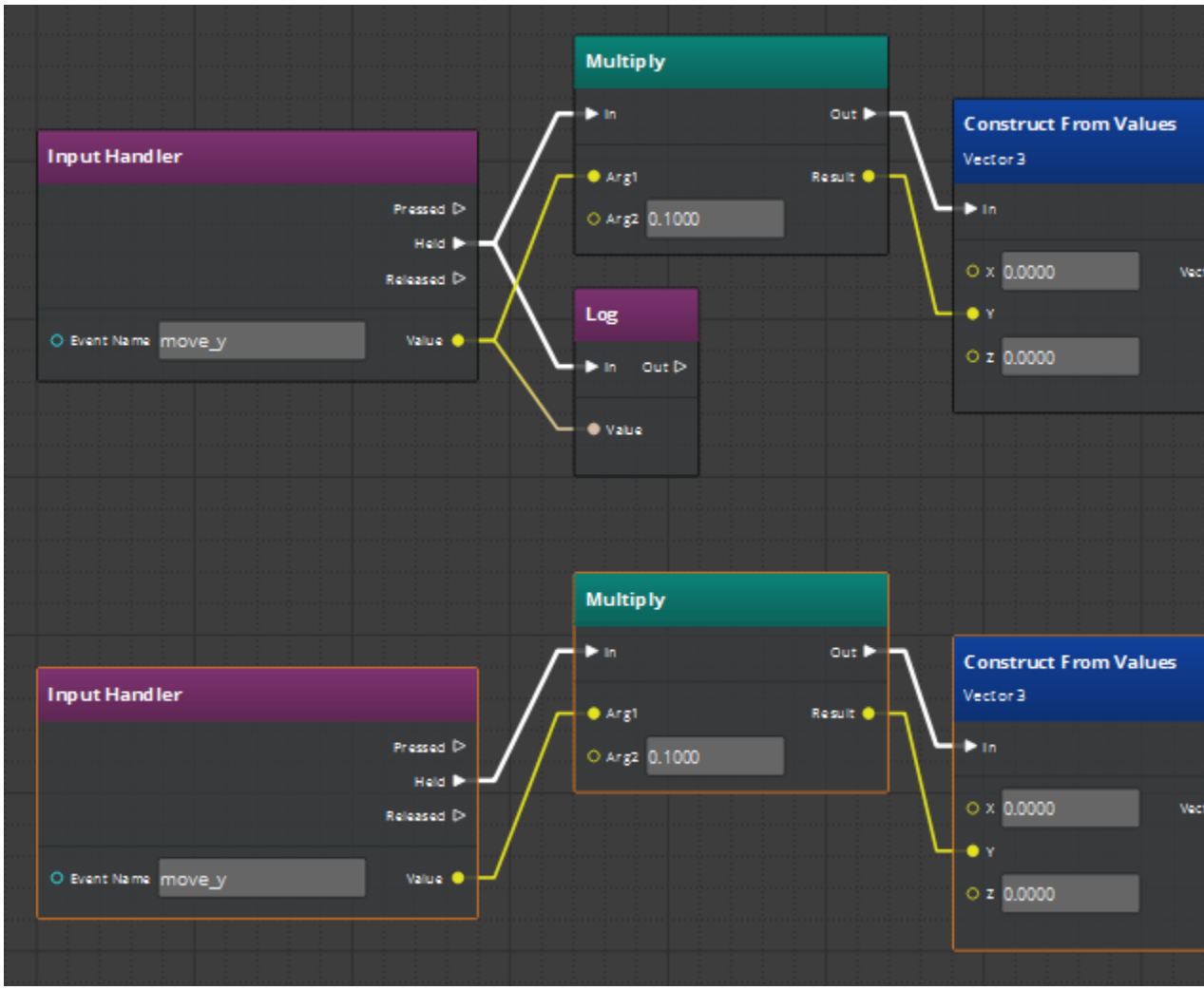
Now that you've converted your input event value to a direction that moves the entity on the y-axis, you can add movement on the x-axis.

To add movement on the x-axis

1. In the **Script Canvas** editor, drag on the canvas to select the **Input Handler**, **Multiply**, **Construct from Values**, and **Move Entity** nodes. An orange outline appears around the selected nodes.

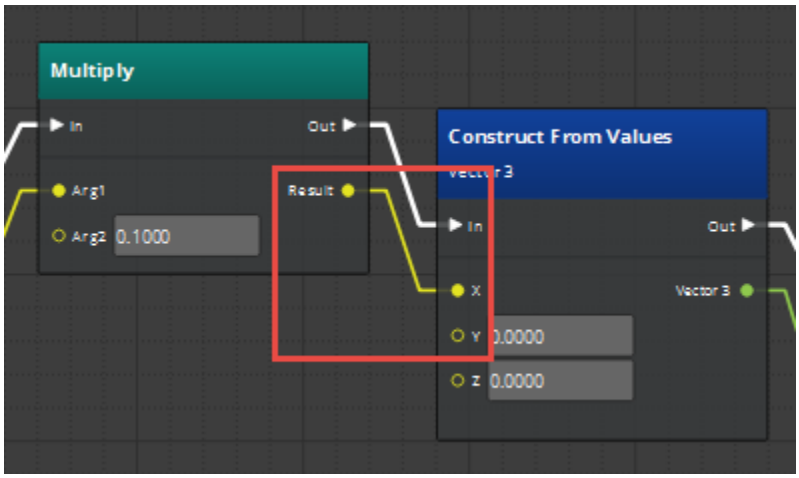


2. Press **Ctrl+C** to copy the selected nodes.
3. Press **Ctrl+V** to paste the copied nodes and their connections. This allows you to clone the existing script and modify the cloned version with the appropriate settings for movement on the x-axis.
4. While still selected, move the nodes so they don't overlap the copied nodes.

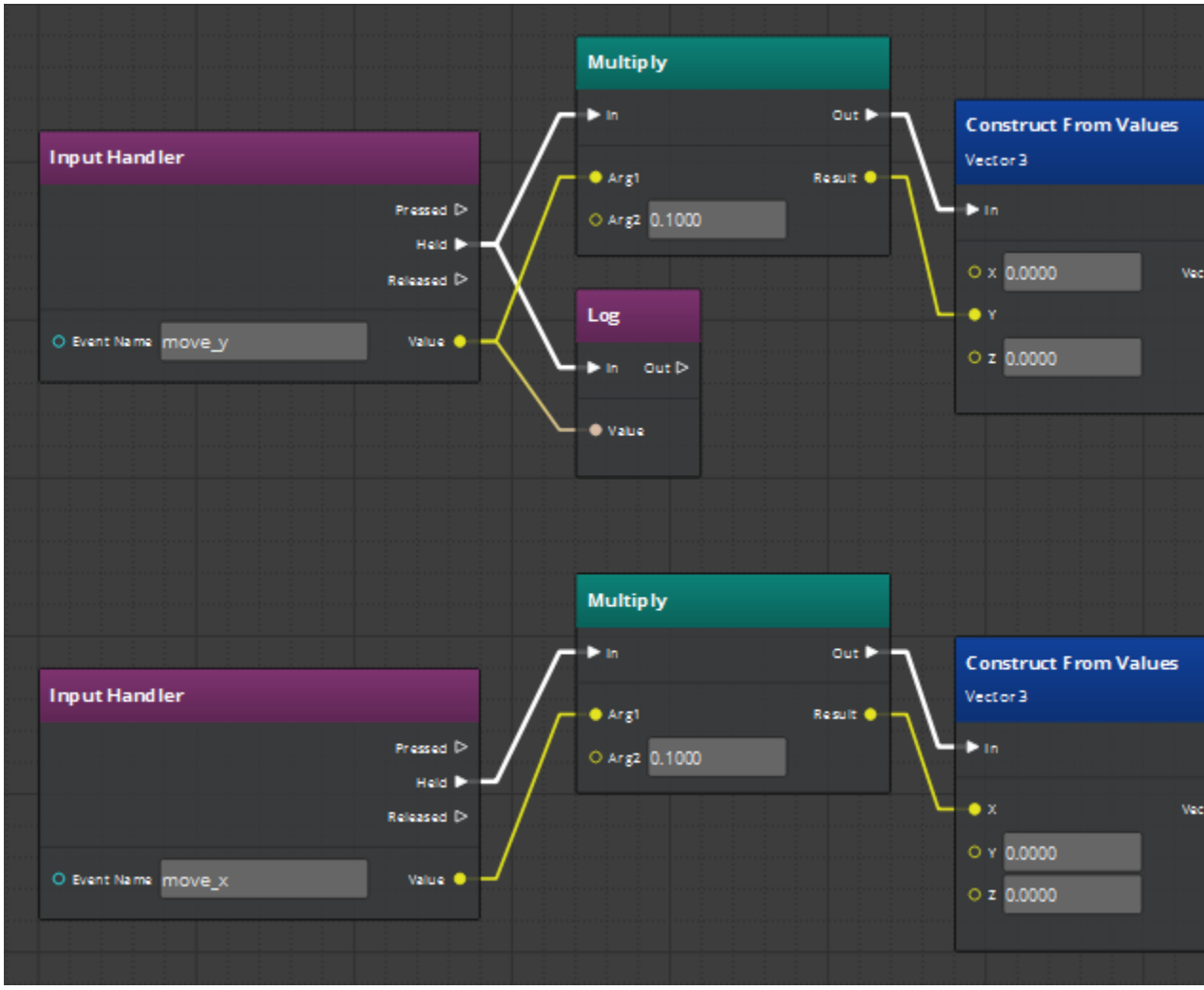


5. In the duplicate **Input Handler** node, for **Event Name**, type `move_x`.
6. Find the yellow line that connects the **Result** pin on the duplicate **Multiply** node to the **Y** pin on the duplicate **Construct from Values** node. Click the connection near the **Y** pin on the **Construct from Values** node and move to the **X** pin.

   Note
   To disconnect a connection, drag it to an empty part of the canvas. To delete a connection, press **Alt** and click the connection.



7. Verify that your script looks like the following image:



8. Choose **File**, **Save**. You can also press **Ctrl+S**.
9. In Lumberyard Editor, press **Ctrl+G** to enter game mode and test your script.
10. Do the following:

   a. To move the sphere forward, press **W**.
   b. To move the sphere backwards, press **S**.
   c. To move the sphere from side to side, press **A** and **D**.
11. When you are done testing your script, press **Esc**.

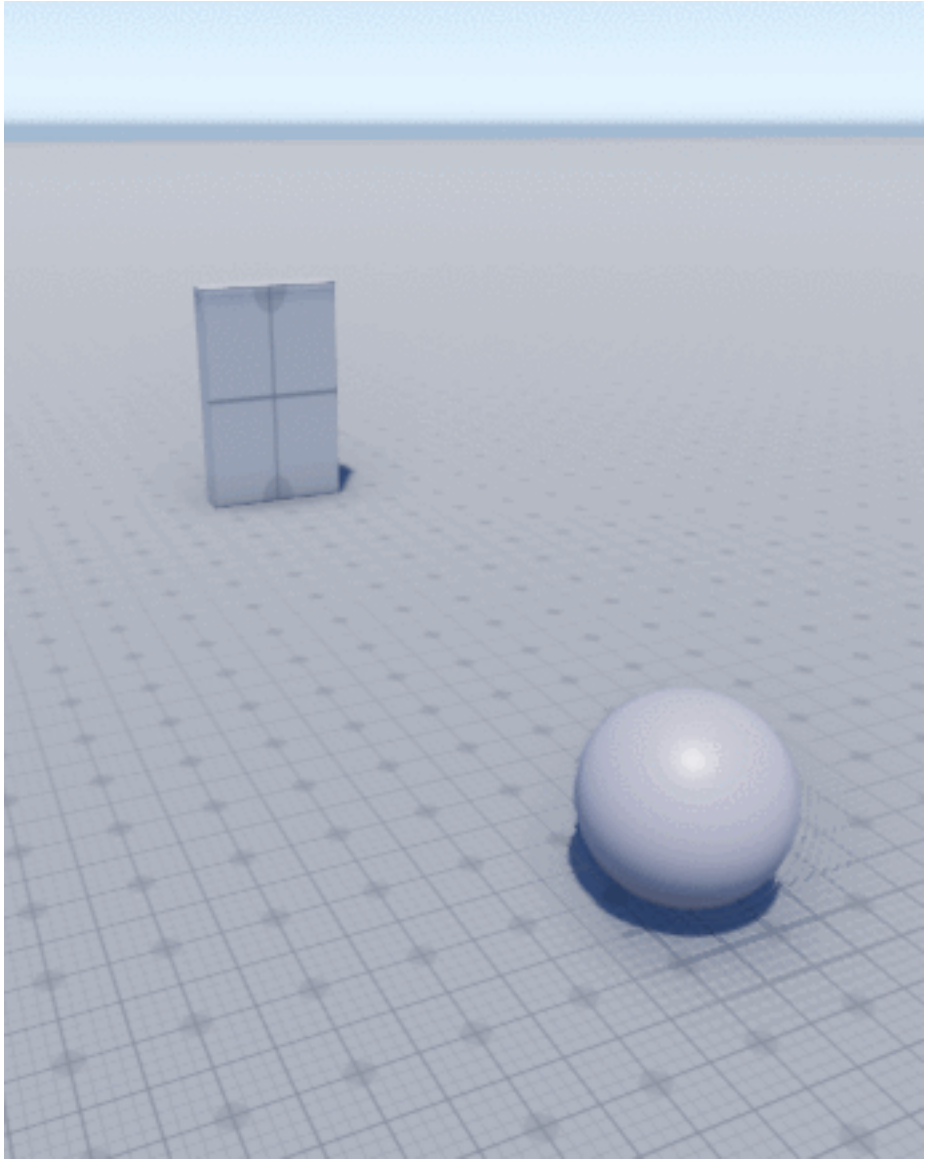
Script Canvas Tutorial: Opening and Closing a Door with Trigger Areas and Variables

Script Canvas is in preview release and is subject to change.

This tutorial builds on what you learned in the Script Canvas Tutorial: Creating a Controllable Entity (p. 688). In the following tutorial you create a door that opens and closes when your controllable sphere enters and exits a trigger area. To do this, you:

- Create a trigger area
- Add event nodes
- Add variables

Before you begin, you must complete the Script Canvas Tutorial: Creating a Controllable Entity (p. 688).

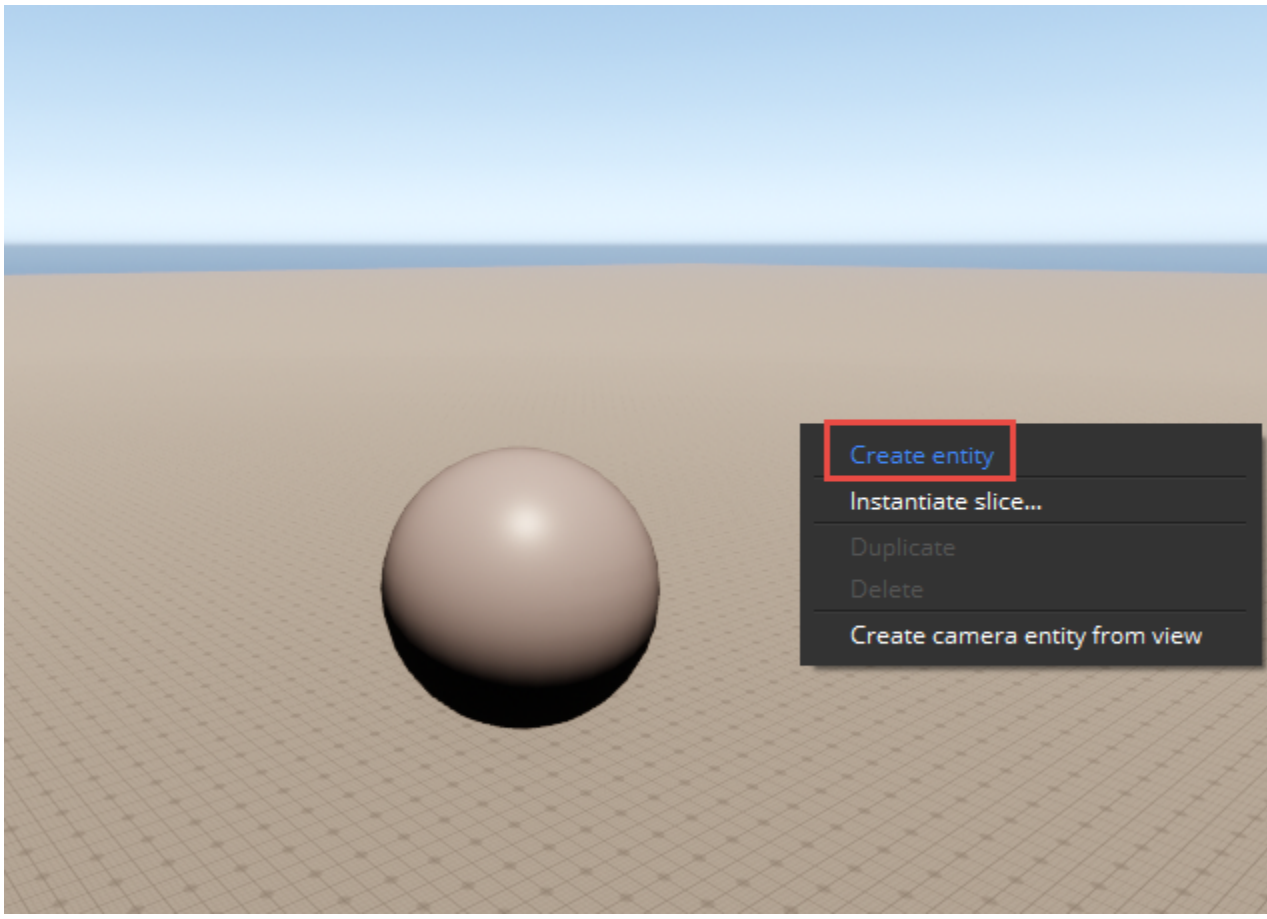


Step 1: Create a Door and Trigger Area

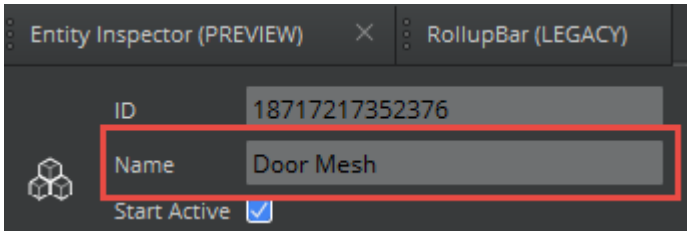
To create a door and trigger area, you create an entity with child entities and add components to the child entities.

To create a door and trigger area

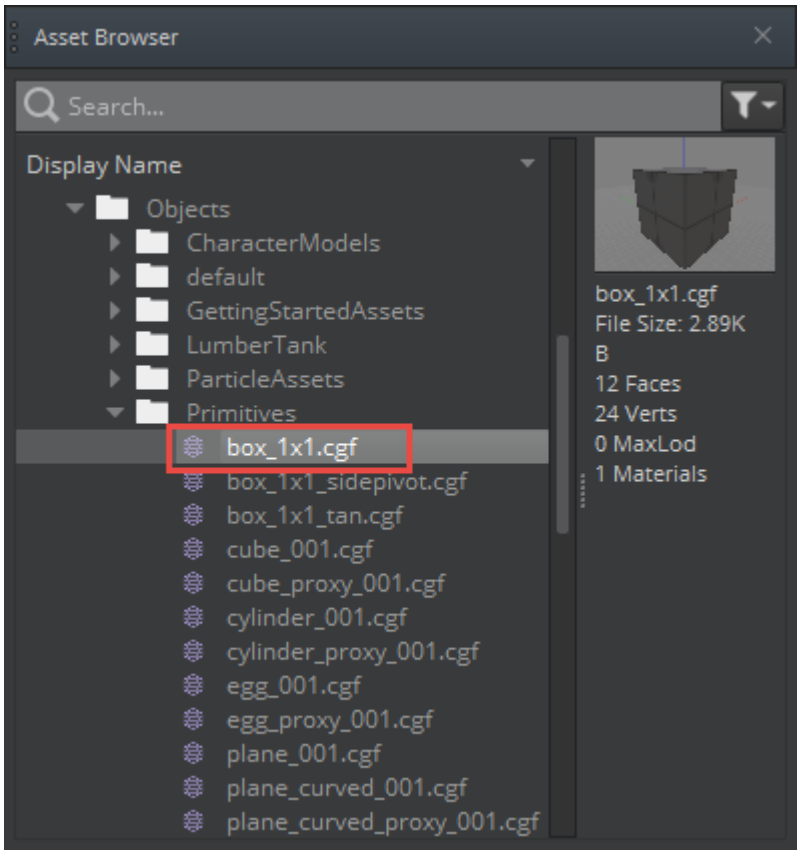
1. In Lumberyard Editor, create an entity by right-clicking in the **Perspective** viewport near your controllable sphere and choosing **Create entity**.



2. In the **Entity Inspector**, for **Name**, type **`Door Group`**.
3. Do the following to create a door:
   a. In the **Entity Outliner**, right-click **Door Group** and choose **Create child entity**. This child entity is your door.
   b. In the **Entity Inspector**, for **Name**, type **`Door Mesh`**.



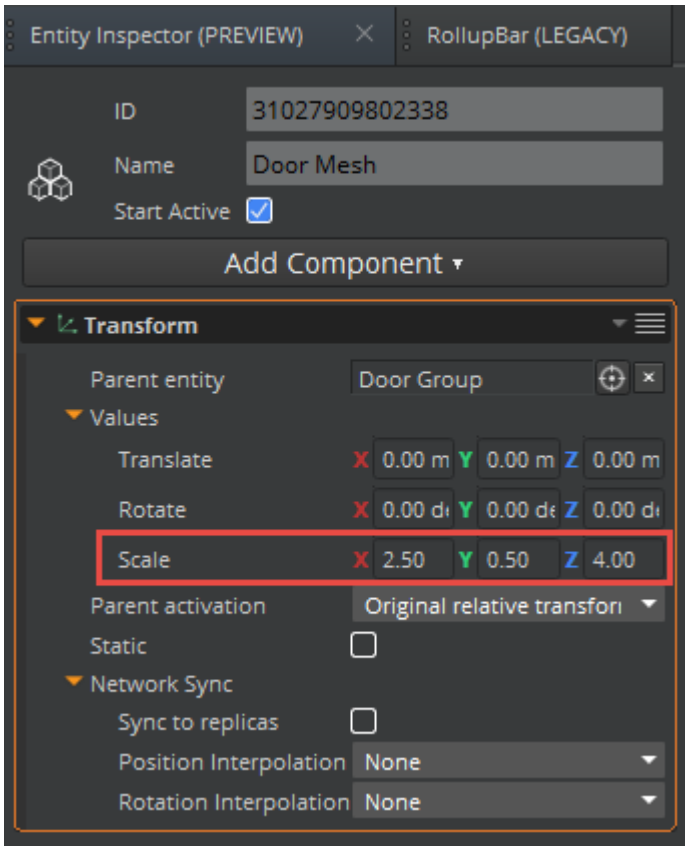
   c. In the **Asset Browser**, navigate to `Game\Objects\Primitives`.
   d. With the **Door Mesh** entity selected in the viewport, drag `box_1x1.cgf` from the **Asset Browser** to the **Entity Inspector** to create a **Mesh** component.



e. In the **Entity Inspector**, click **Add Component**. Under **Physics**, click **Static Physics**. This adds a **Static Physics** component to the **Door Mesh** entity.

f. Click **Add Component** again. Under **Physics**, click **Mesh Collider**. This component defines the collision shape for the **Door Mesh** entity.

g. In the **Entity Inspector**, under **Transform**, for **Scale**, set **X** to `2.5`, **Y** to `0.5`, and **Z** to `4.0`.



4. Do the following to create a trigger area:

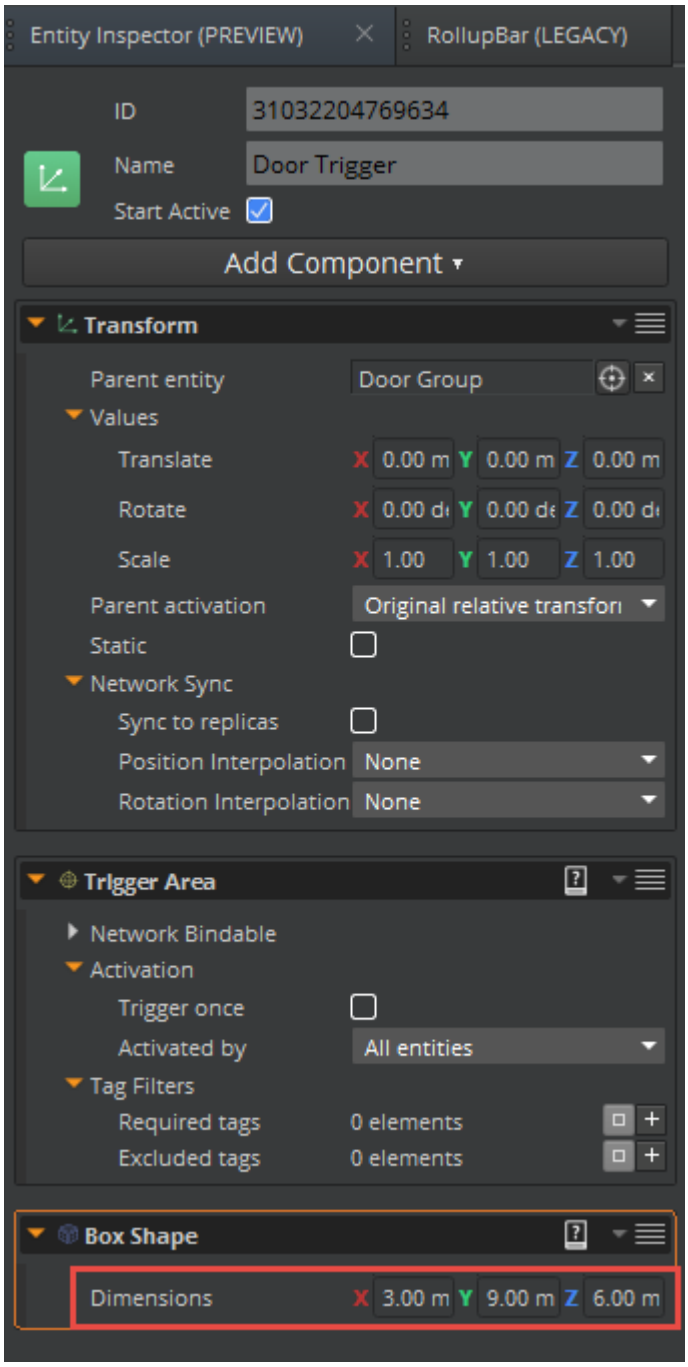
   a. In the **Entity Outliner**, right-click **Door Group** and choose **Create child entity**. This child entity is your trigger area.

   b. In the **Entity Inspector**, for **Name**, type `Door Trigger`.

   c. Click **Add Component**. Under **Scripting**, click **Trigger Area**.

   d. Under **Trigger Area**, click **Add Required Component**. Under **Shape**, click **Box Shape**.

   e. Under **Box Shape**, for **Dimensions**, set **X** to `3.0`, **Y** to `9.0`, and **Z** to `6.0`.



f. Click **Add Component** again. Under **Scripting**, click **Script Canvas**.

g. In Lumberyard Editor, choose **Tools**, **Script Canvas**.

h. In the **Script Canvas** editor, choose **File**, **New Script**.

i. After the new canvas loads, choose **File**, **Save As**.

j. For **File name**, type **`door`** and then click **Save**. Note the directory path for your saved file, which you will need later.

k. In Lumberyard Editor, in the **Entity Inspector**, under **Script Canvas**, click the browse (**...**) icon and navigate to the `door.scriptcanvas` file. Choose `door.scriptcanvas` and then click **OK**.

Step 2: Create a Script to Open and Close the Door

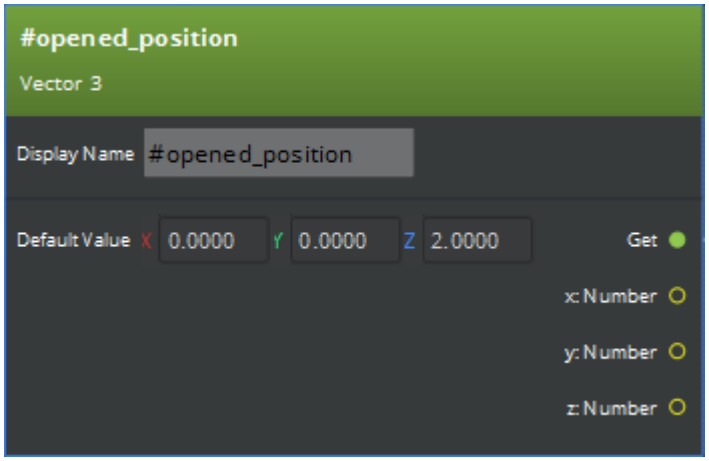
Now that you've set up your door and trigger area, you can create a script that opens and closes the door when another entity enters or leaves the trigger area.

To create a script that opens and closes the door

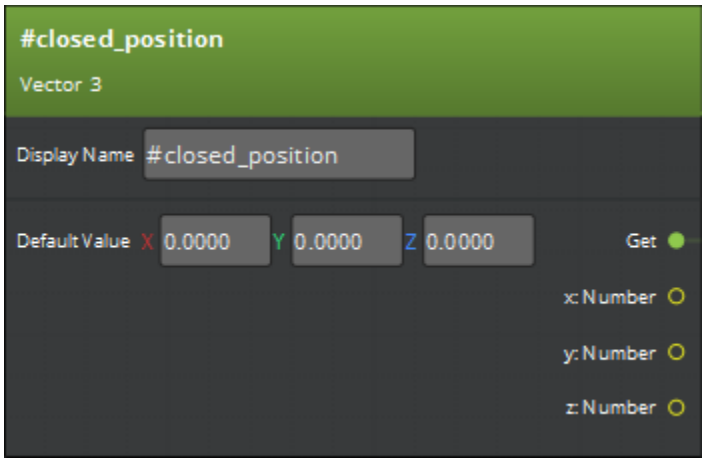
1. In the **Script Canvas** editor, open the `door.scriptcanvas` file, if it's not still open from Step 1: Create a Door and Trigger Area (p. 700).
2. In the **Node Palette**, type **`vector 3`** in the search box.
3. Under **Variables**, **Create Variable**, drag **Vector 3** from the **Node Palette** to the canvas. You can use variables to store and modify persistent values in your script.
4. In the **#Variable 1** node, do the following:

   a. For **Display Name**, type **`opened_position`**.

   b. For **Default Value**, set **Z** to **`2`**. A positive value for the z-axis slides the door up. Because the door entity is a child of **Door Group**, you can use local relative positions to control the open and closed positions of the door.

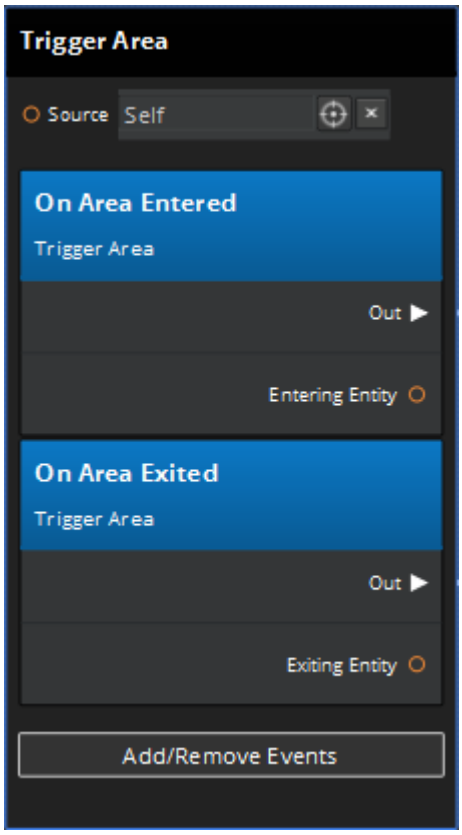


5. Drag another **Vector 3** from the **Node Palette** to the canvas.
6. In the **#Variable 2** node, for **Display Name**, type **`closed_position`**. The closed position can use the default value of 0,0,0.



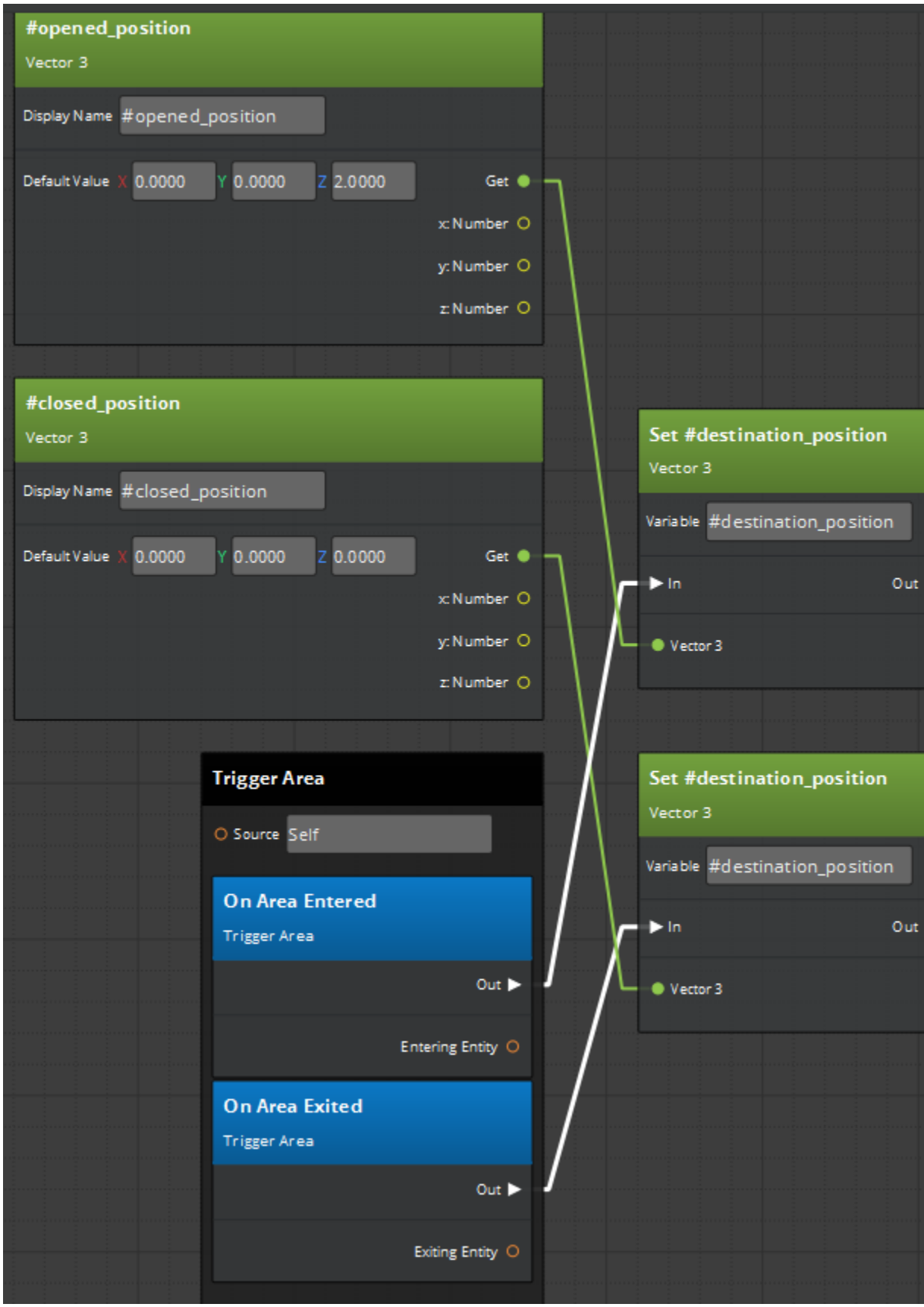
7. Drag another **Vector 3** from the **Node Palette** to the canvas.
8. In the **#Variable 3** node, for **Display Name**, type **`destination_position`**. This variable sets the door's destination position when the entity enters and exits the trigger area.

9. In the **Node Palette**, type `trigger` in the search box.
10. Under **Gameplay**, **Trigger Area**, drag **On Area Entered** from the **Node Palette** to the canvas. With this event node you can easily use the Lumberyard EBus messaging system.
11. In the **Trigger Area** node, click **Add/Remove Events** and then select the **On Area Exited** check box. This exposes the entered and exited events from the **Trigger Area** EBus.



12. In the **Node Palette**, type `position` in the search box.
13. Drag two **Set #destination_position** nodes from the **Node Palette** to the canvas. Get and Set nodes are available in the **Node Palette** after you create a variable. These nodes enable you to work with dynamic variable values. The core variable node initializes the variable.
14. In the canvas, make the following connections:

a. Select the **Out** pin for **On Area Entered** and drag to connect it to the **In** pin for a **Set #destination_position** node.
b. Select the **Out** pin for **On Area Exited** and drag to connect it to the **In** pin of the other **Set #destination_position** node.
c. Select the **Get** pin for **#opened_position** and drag to connect it to the **Vector 3** pin for the **Set #destination_position** node that is connected to **On Area Entered**.
d. Select the **Get** pin for **#closed_position** and drag to connect it to the **Vector 3** pin for the **Set #destination_position** node that is connected to **On Area Exited**.
e. Verify that your script looks like the following image:

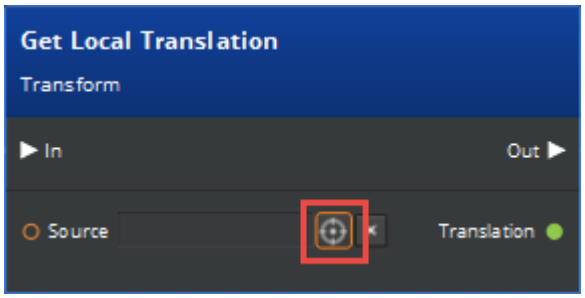

#opened_position
Vector 3
Display Name #opened_position
Default Value X 0.0000 Y 0.0000 Z 2.0000
Get
x: Number
y: Number
z: Number
#closed_position
Vector 3
Display Name #closed_position
Default Value X 0.0000 Y 0.0000 Z 0.0000
Get
x: Number
y: Number
z: Number
Set #destination_position
Vector 3
Variable #destination_position
In
Out
Vector 3
Set #destination_position
Vector 3
Variable #destination_position
In
Out
Vector 3
Trigger Area
Source Self
On Area Entered
Trigger Area
Out
Entering Entity
On Area Exited
Trigger Area
Out
Exiting Entity

15. Do the following to get the **Door Mesh** entity's position and interpolate to the destination:

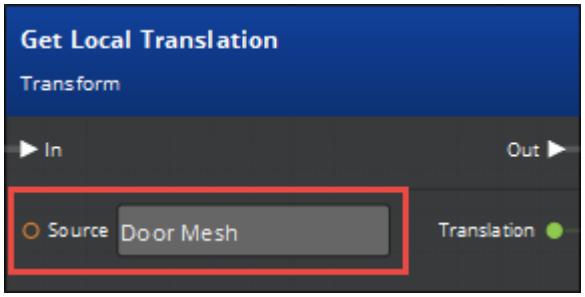
    a. In the **Node Palette**, type `get local` in the search box.

    b. Drag **Get Local Translation** from the **Node Palette** to the canvas. You can use this node to get the current position of the **Door Mesh** entity and interpolate to the destination. A local translation applies to the translation of the entity relative to its parent.

    c. In the **Get Local Translation** node, pause on the **Source** text box and click the target button. When selected, the target button has an orange outline.

    d. In the **Entity Outliner**, select **Door Mesh** to assign the **Door Mesh** entity to the **Source** property in the **Get Local Translation** node.

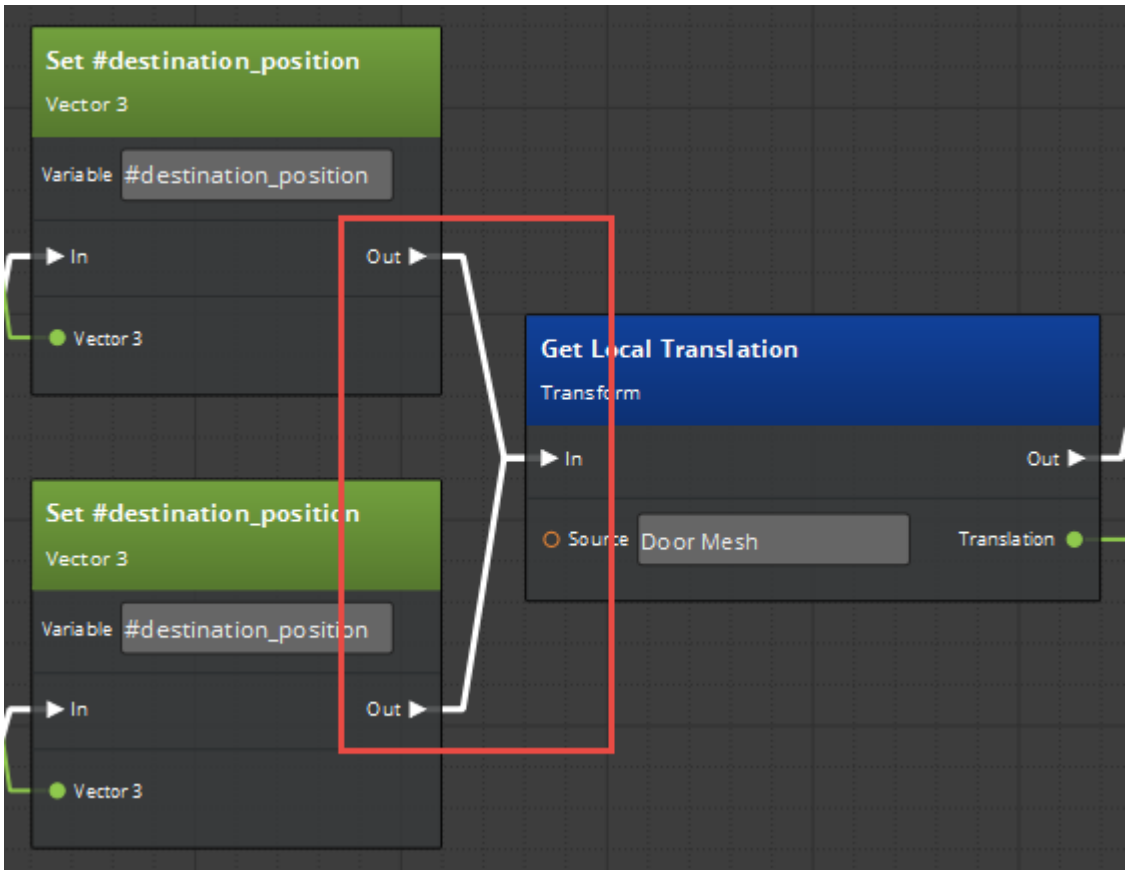
    

    Note
    To reset an entity reference, right-click twice on the **Source** text box and choose **Set to Self**.

    e. In the **Script Canvas** editor, make the following connections:

        i. Select the **Out** pin for a **Set #destination_position** node and drag to connect it to the **In** pin for **Get Local Translation**.

        ii. Select the **Out** pin for the other **Set #destination_position** node and drag to connect it to the **In** pin for **Get Local Translation**.



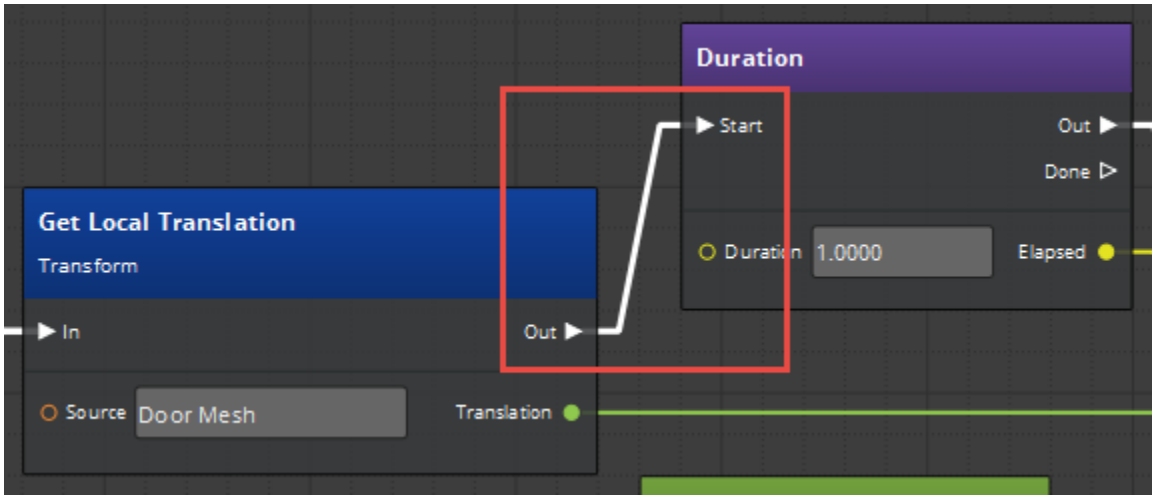
When multiple connections enter a single logic pin, the node is executed each time either execution is triggered. The node is executed more than once in the same game tick if multiple executions are triggered simultaneously.

16. Do the following to execute nodes for a specified amount of time, in seconds:

    a. In the **Node Palette**, type `duration` in the search box.

    b. Drag **Duration** from the **Node Palette** to the canvas.

    c. Select the **Out** pin for **Get Local Translation** and drag to connect it to the **Start** pin for **Duration**. Triggering the **Duration** node resets the time.



d. In the **Duration** node, for **Duration**, type `1.0` (seconds).

17. Do the following to set up interpolation between the current position and the destination:

    a. In the **Node Palette**, type `lerp` in the search box.

    b. Under **Math**, **Vector 3**, drag **Lerp** from the **Node Palette** to the canvas. This node blends two values based on the **Percentage** property.

    c. Select the **Out** pin for **Duration** and drag to connect it to the **In** pin for **Lerp**.

    d. Select the **Elapsed** pin for **Duration** and drag to connect it to the **Percentage** pin for **Lerp**.

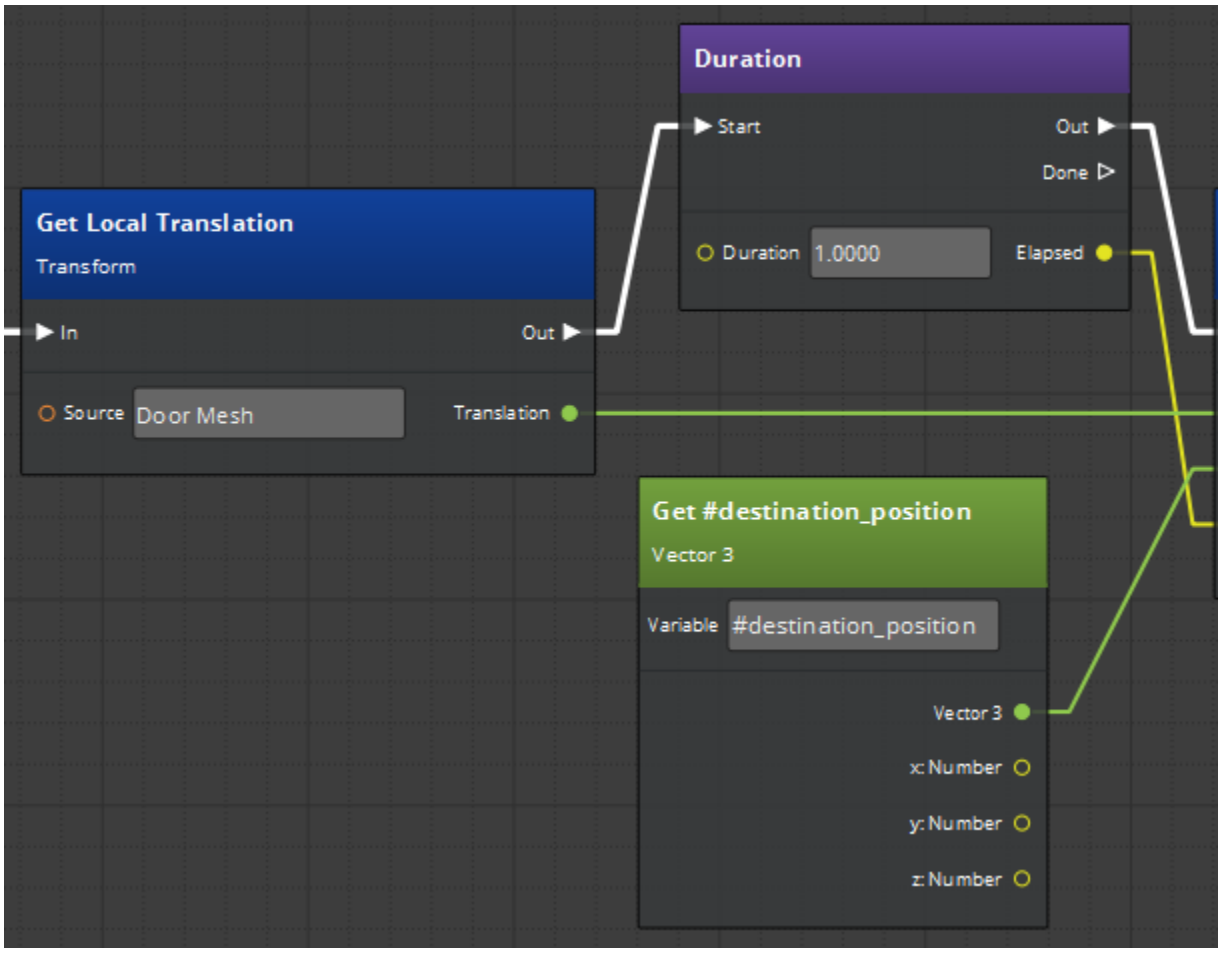
    e. Select the **Translation** pin for **Get Local Translation** and drag to connect it to the **Start** pin for **Lerp**.

    f. In the **Node Palette**, type `destination` in the search box.

    g. Drag **Get #destination_position** from the **Node Palette** to the canvas.

    h. Select the **Vector 3** pin for **Get #destination_position** and drag to connect it to the **End** pin for **Lerp**.

    i. Verify that your script looks like the following image:



18. Do the following to set the position of the door when the **Duration** node blends between the current and destination positions:

    a. In the **Node Palette**, type `set local translation` in the search box.

    b. Drag **Set Local Translation** from the **Node Palette** to the canvas.

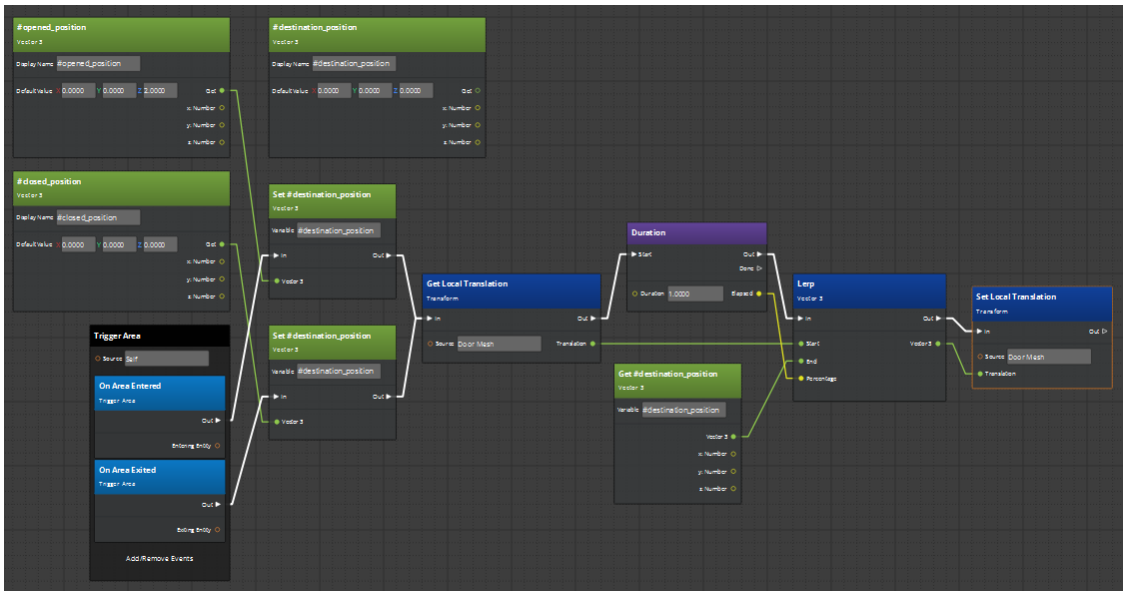
    c. Select the **Out** pin for **Lerp** and drag to connect it to the **In** pin for **Set Local Translation**.

d. Select the **Vector 3** pin for **Lerp** and drag to connect it to the **Translation** pin for **Set Local Translation**.

e. In the **Set Local Translation** node, pause on the **Source** text box and click the target button. When selected, the target button has an orange outline.

f. In the **Entity Outliner**, select **Door Mesh** to assign the **Door Mesh** entity to the **Source** property in the **Set Local Translation** node.

19. Verify that your script looks like the following image:



20. In the **Script Canvas** editor, choose **File**, **Save**. You can also press **Ctrl+S**.
21. In Lumberyard Editor, press **Ctrl+G** to enter game mode and test your script.
22. To move the sphere forward into the door trigger area and slide open the door, press the **W**, **A**, **D** keys.
23. To move the sphere backwards out of the trigger area and slide the door closed, press **S**.
24. When you are done testing your script, press **Esc**.

Script Canvas Node Reference

Script Canvas is in preview release and is subject to change.

Following are some of the most commonly used nodes in the **Script Canvas** editor. This reference includes the various types of nodes, their uses, and their input and output ports.

Topics

- Finding the Texture Name (p. 753)

> **Tip**
> On the **Node Palette** or on the node canvas, you can pause on the node name and the node parameters for more information.

Some nodes are hidden from the default **Script Canvas** editor view.

To show hidden nodes in Script Canvas

1. In Lumberyard Editor, choose **Tools**, **Script Canvas**.
2. In the **Script Canvas** editor, choose **Edit**, **Settings**, **Global Preferences**.
3. In the **Global Preferences** window, choose **Show nodes excluded from preview**.

Environment Nodes

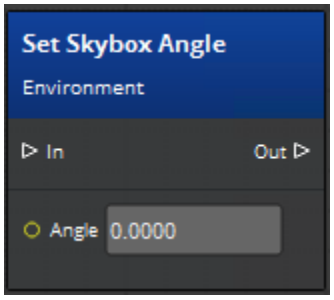
You can use environment nodes in the **Script Canvas** to configure sky effects.

Topics

Set Skybox Angle

Rotates the skybox around the z-axis.

Contents



Inputs

Port	Type	Description
In	Event	Triggers the node.
Angle	Number	Rotation angles in degrees.

Outputs

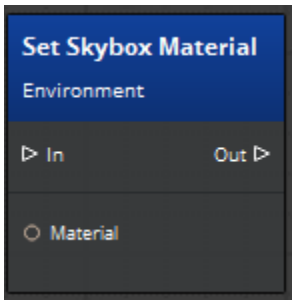
Port	Type	Description
Out	Event	Sends when the node is finished.

Note
This node corresponds to the SkyMaterialSwitch node (p. 865) in Flow Graph

Set Skybox Material

Apply a specified material to the skybox to give the sky a different look.

Contents



Input

Port	Type	Description
In	Event	Triggers the node.
Material	Material	The material to set.

Outputs

Port	Type	Description
Out	Event	Sends when the node is finished.

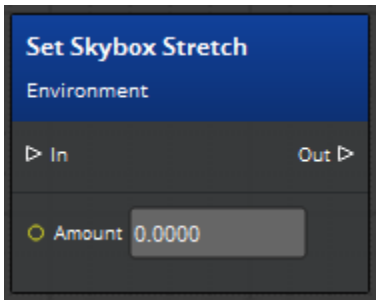
Note
This node corresponds to the SkyMaterialSwitch node (p. 865) in Flow Graph.

Set Skybox Stretch

Stretches the skybox vertically.

Contents

- Outputs (p. 715)



Inputs

Port	Type	Description
In	Event	Triggers the node.
Amount	Number	Vertical stretch/scale factor. The default value is `0`, which means no stretch.

Outputs

Port	Type	Description
Out	Event	Sends when the node is finished.

> **Note**
> This node corresponds to the SkyMaterialSwitch node (p. 865) in Flow Graph.

Material Nodes

Scripting offers two ways to interact with materials:

1. You can access and manipulate materials directly, which affects all entities to which the material is applied. For more information, see Direct Material Scripting (p. 715).
2. You can manipulate materials on a specific entity, which automatically clones the material first, so that changes apply only to the specific entity. For more information, see Entity Material Scripting (p. 725).

Topics

Direct Material Scripting

You can work with material objects directly through scripts. For example, you can load materials, get and set parameters, and apply the material to entities. Changes made to a material affect all entities that use that material.

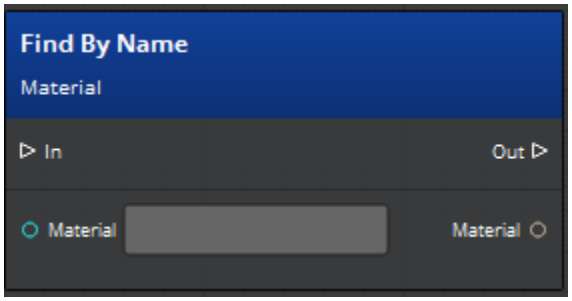
You cannot use scripts to create a material object directly, but you can access materials through nodes like **Get Material (p. 728)** or **Load by Name (p. 717)**.

Topics

Find by Name

Finds and returns the material with the specified name. The material must be loaded into memory in order to be found.

Contents



Inputs

Port	Type	Description
In	Event	Triggers the node.
Material	String	The path name of a material to find. For more information, see Finding the Texture Name (p. 753).

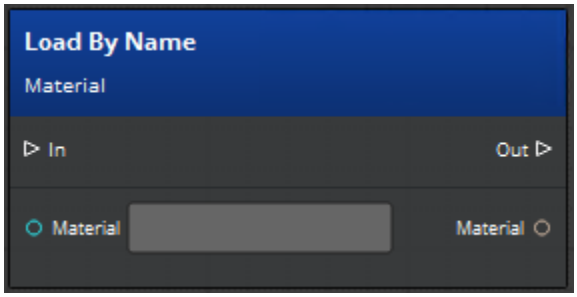
Outputs

Port	Type	Description
Out	Event	Sends when the node is finished.
Material	Material	The material that was found. Returns `Null` if the material is not found.

Load by Name

Finds and returns the material with the specified name. Loads the material if it is not already loaded.

Contents



Inputs

Port	Type	Description
In	Event	Triggers the node.
Material	String	The path name of a material to find and load. For more information, see Finding the Material Name (p. 749).

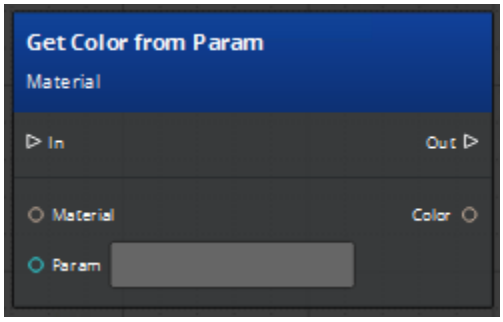
Outputs

Port	Type	Description
Out	Event	Sends when the node is finished.
Material	Material	The material that was found. Returns `Null` if the material is not found.

Get Color from Param

Gets a material parameter as a color value.

Contents



Inputs

Port	Type	Description
In	Event	Triggers the node.
Material	Material	The material to query.
Param	String	The name of the material parameter to return. For more information, see Finding the Material Parameter Name (p. 750).

Outputs

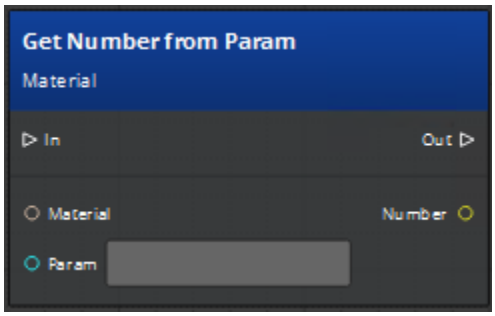
Port	Type	Description
Out	Event	Sends when the node is finished.
Color	Color	The queried value. If the parameter is not found, the value is (0, 0, 0, 0).

> **Note**
> This node corresponds to the MaterialParams node (p. 937) in Flow Graph.

Get Number from Param

Gets a material parameter as a numerical value.

Contents



Inputs

Port	Type	Description
In	Event	Triggers the node.
Material	Material	The material to query.
Param	String	The name of the material parameter to return. For more information, see Finding the Material Parameter Name (p. 750).

Outputs

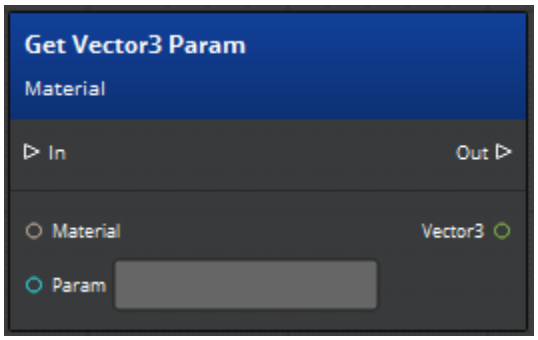
Port	Type	Description
Out	Event	Sends when the node is finished.
Number	Number	The queried value. If the parameter is not found, the value is `0`.

> **Note**
> This node corresponds to the MaterialParams node (p. 937) in Flow Graph.

Get Vector3 Param

Gets a material parameter as a Vector3 value.

Contents



Inputs

Port	Type	Description
In	Event	Triggers the node.
Material	Material	The material to query.
Param	String	The name of the material parameter to return. For more information, see Finding the Material Parameter Name (p. 750).

Outputs

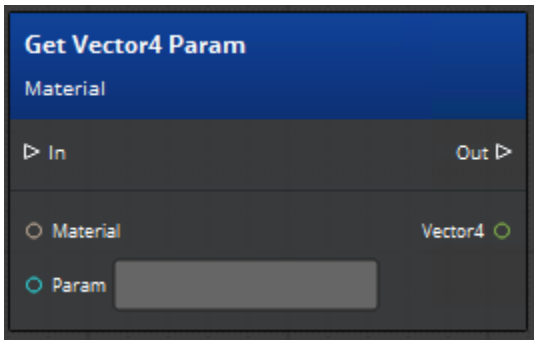
Port	Type	Description
Out	Event	Sends when the node is finished.
Vector3	Vector3	The queried value. If the parameter is not found, the value is (0, 0, 0).

Note
This node corresponds to the MaterialParams node (p. 937) in Flow Graph.

Get Vector4 Param

Gets a material parameter as a Vector4 value.

Contents



Inputs

Port	Type	Description
In	Event	Triggers the node.
Material	Material	The material to query.
Param	String	The name of the material parameter to return. For more information, see Finding the Material Parameter Name (p. 750).

Outputs

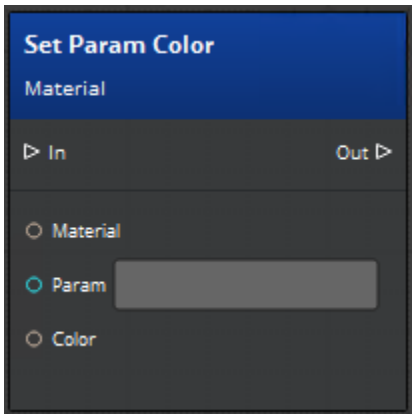
Port	Type	Description
Out	Event	Sends when the node is finished.
Vector4	Vector4	The queried value. If the parameter is not found, the value is (0, 0, 0, 0).

Note
This node corresponds to the MaterialParams node (p. 937) in Flow Graph.

Set Param Color

Sets a material parameter with a color value.

Contents



Inputs

Port	Type	Description
In	Event	Triggers the node.
Material	Material	The material to modify.
Param	String	The name of the material parameter to return. For more information, see Finding the Material Parameter Name (p. 750).
Color	Color	The new value to apply.

Outputs

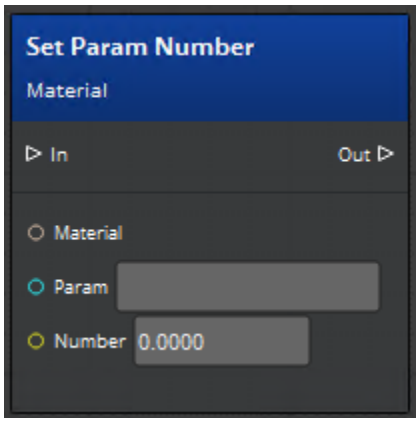
Port	Type	Description
Out	Event	Sends when the node is finished.

> **Note**
> This node corresponds to the MaterialParams node (p. 937) in Flow Graph.

Set Param Number

Sets a material parameter with a number value.

Contents



Inputs

Port	Type	Description
In	Event	Triggers the node.
Material	Material	The material to modify.
Param	String	The name of the material parameter to set. For more information, see Finding the Material Parameter Name (p. 750).
Number	Number	The new value to apply.

Outputs

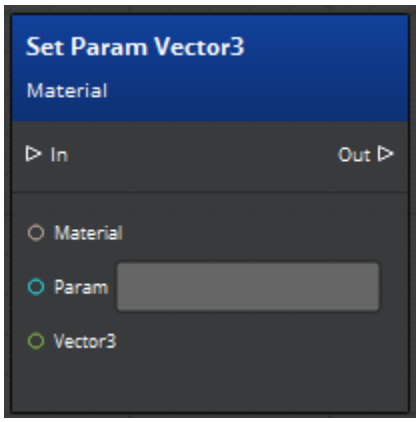
Port	Type	Description
Out	Event	Sends when the node is finished

> **Note**
> This node corresponds to the MaterialParams node (p. 937) in Flow Graph.

Set Param Vector3

Sets a material parameter with a Vector3 value.

Contents



Inputs

Port	Type	Description
In	Event	Triggers the node.
Material	Material	The material to modify.
Param	String	The name of the material parameter to set. For more information, see Finding the Material Parameter Name (p. 750).
Vector3	Vector3	The new value to apply.

Outputs

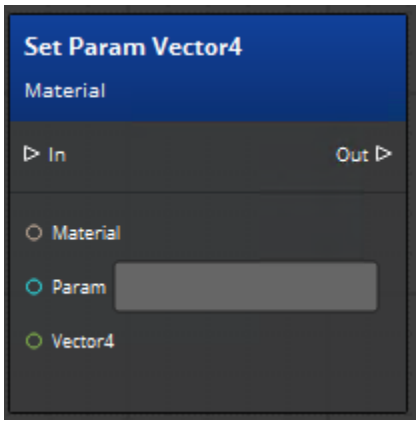
Port	Type	Description
Out	Event	Sends when the node is finished.

> **Note**
> This node corresponds to the MaterialParams node (p. 937) in Flow Graph.

Set Param Vector4

Sets a material parameter with a Vector4 value.

Contents



Inputs

Port	Type	Description
In	Event	Triggers the node.
Material	Material	The material to modify.
Param	String	The name of the material parameter to set. For more information, see Finding the Material Parameter Name (p. 750).
Vector4	Vector4	The new value to apply.

Outputs

Port	Type	Description
Out	Event	Sends when the node is finished.

> **Note**
> This node corresponds to the MaterialParams node (p. 937) in Flow Graph.

Entity Material Scripting

You can use scripting to get and set parameters for an entity's material. Changes made to material parameters affect only the target entity; the entity's material is automatically cloned before any changes are made.

> **Important**
> Depending on the material owner type, the material asset may not be available when the script is first activated (`OnActivate` in Lua or the **On Graph Start** node in **Script Canvas**). This lack of availability occurs because assets are still loading. Use the **OnMaterialOwnerReady (p. 726)** node instead of the **On Graph Start** node.

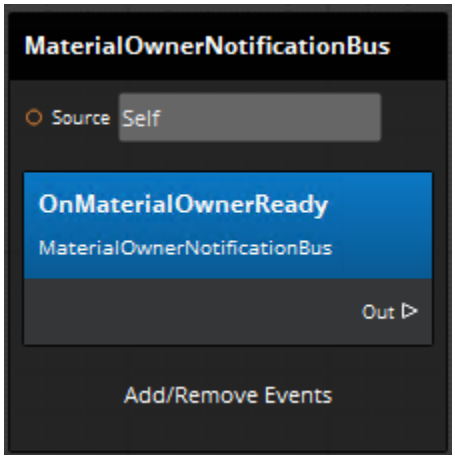
Topics

- OnMaterialOwnerReady (p. 726)

OnMaterialOwnerReady

Triggered when the material owner finishes loading its assets. We recommend that you wait for this event before you access the `MaterialOwnerRequestBus`. Otherwise the material asset might not be available when the script is first activated. For example, a mesh component usually does not have its mesh loaded until at least one frame after the script starts.

Contents



Inputs

Port	Type	Description
Source	EntityID	References a specific entity from which events are generated. By default, it references **Self**, the entity to which the **Script Canvas (p. 545)** component attaches the current script. You can also specify another entity. For more information, see Referencing Entities for Nodes (p. 686).

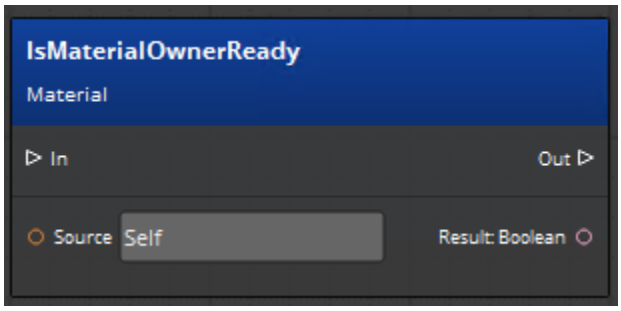
Outputs

Port	Type	Description
Out	Event	Sends when the node is finished.

IsMaterialOwnerReady

Indicates whether the material owner finished loading its assets and the material is ready.

Contents



Inputs

Port	Type	Description
In	Event	Triggers the node.
Source	EntityID	References a specific entity from which events are generated. By default, it references **Self**, the entity to which the **Script Canvas (p. 545)** component attaches the current script. You can also specify another entity. For more information, see Referencing Entities for Nodes (p. 686).

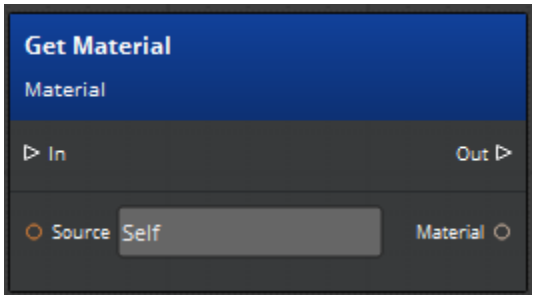
Outputs

Port	Type	Description
Out	Event	Sends when the node is finished.
Result	Boolean	Indicates whether the material owner is ready so that request functions may be called.

Get Material

Returns an entity's current material.

Contents



> **Note**
> If this node is called after **Set Material (p. 729)** with an invalid material, the node returns the entity's default material rather than `Null`.

Inputs

Port	Type	Description
In	Event	Triggers the node.
Source	EntityID	References a specific entity from which events are generated. By default, it references **Self**, the entity to which the **Script Canvas (p. 545)** component attaches the current script. You can also select another entity. For more information, see Referencing Entities for Nodes (p. 686).

Outputs

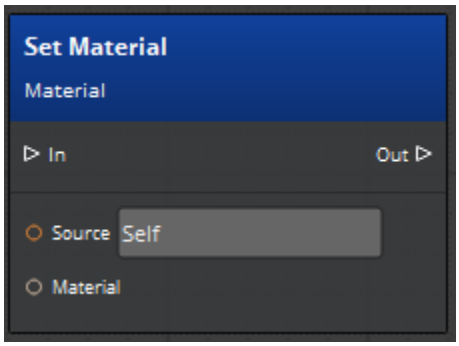
Port	Type	Description
Out	Event	Sends when the node is finished.
Material	Material	The entity's current material.

> **Note**
> This node corresponds to the EntityMaterialChange node (p. 936) in Flow Graph.

Set Material

Sets an entity's material. If the material is `Invalid`, this node removes the entity's material. The entity uses its default material if it has one.

Contents



Inputs

Port	Type	Description
In	Event	Triggers the node.
Source	EntityID	References a specific entity from which events are generated. By default, it references **Self**, the entity to which the **Script Canvas (p. 545)** component attaches the current script. You can also select another entity. For more information, see Referencing Entities for Nodes (p. 686).
Material	Material	The new material.

Outputs

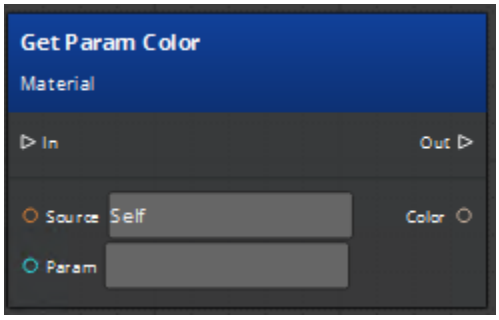
Port	Type	Description
Out	Event	Sends when the node is finished.

> **Note**
> This node corresponds to the EntityMaterialChange node (p. 936) in Flow Graph.

Get Param Color

Returns a material's color parameter value for the specified entity.

Contents



Inputs

Port	Type	Description
In	Event	Triggers the node.
Source	EntityID	References a specific entity from which events are generated. By default, it references **Self**, the entity to which the **Script Canvas (p. 545)** component attaches the current script. You can also select another entity. For more information, see Referencing Entities for Nodes (p. 686).
Param	String	The name of the material parameter to query. For more information, see Finding the Material Parameter Name (p. 750).

Outputs

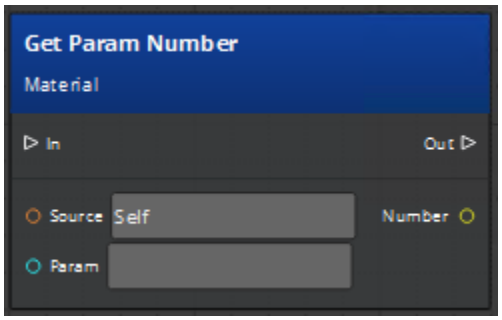
Port	Type	Description
Out	Event	Sends when the node is finished.
Color	Color	The queried value. If the parameter is not found, the value is (0, 0, 0, 0).

Note
This node corresponds to the EntityMaterialChange node (p. 936) in Flow Graph.

Get Param Number

Returns a material's numerical parameter value for the specified entity.

Contents



Inputs

Port	Type	Description
In	Event	Triggers the node.
Source	EntityID	References a specific entity from which events are generated. By default, it references **Self**, the entity to which the **Script Canvas (p. 545)** component attaches the current script. You can also select another entity. For more information, see Referencing Entities for Nodes (p. 686).
Param	String	The name of the material parameter to query. For more information, see Finding the Material Parameter Name (p. 750).

Outputs

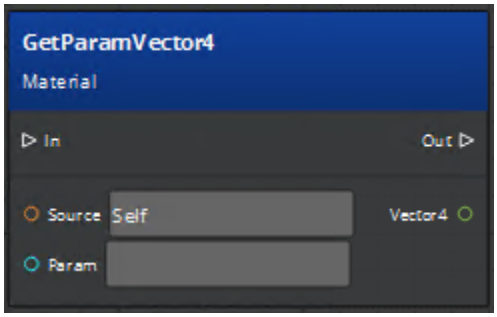
Port	Type	Description
Out	Event	Sends when the node is finished.
Number	Number	The queried value. If the parameter is not found, the value is `0`.

Note
This node corresponds to the EntityMaterialChange node (p. 936) in Flow Graph.

GetParamVector4

Returns a material's Vector4 parameter value for the specified entity.

Contents



Inputs

Port	Type	Description
In	Event	Triggers the node.
Source	EntityID	References a specific entity from which events are generated. By default, it references **Self**, the entity to which the **Script Canvas (p. 545)** component attaches the current script. You can also select another entity. For more information, see Referencing Entities for Nodes (p. 686).
Param	String	The name of the material parameter to query. For more information, see Finding the Material Parameter Name (p. 750).

Outputs

Port	Type	Description
Out	Event	Sends when the node is finished.

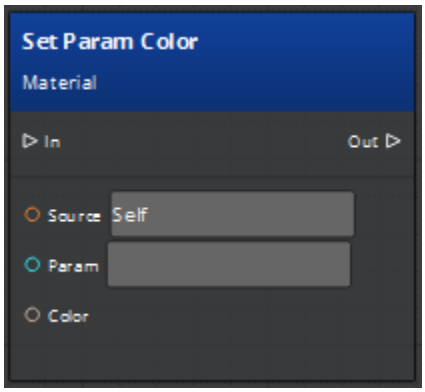
Port	Type	Description
Vector4	Vector4	The queried value. If the parameter is not found, the value is (0, 0, 0, 0).

Note
This node corresponds to the EntityMaterialChange node (p. 936) in Flow Graph.

Set Param Color

Sets a material's color parameter value for the specified entity.

Contents



Inputs

Port	Type	Description
In	Event	Triggers the node.
Source	EntityID	References a specific entity from which events are generated. By default, it references **Self**, the entity to which the **Script Canvas (p. 545)** component attaches the current script. You can also select another entity. For more information, see Referencing Entities for Nodes (p. 686).
Param	String	The name of the material parameter to query. For more information, see Finding the Material Parameter Name (p. 750).

Port	Type	Description
Color	Color	The new value to apply.

Outputs

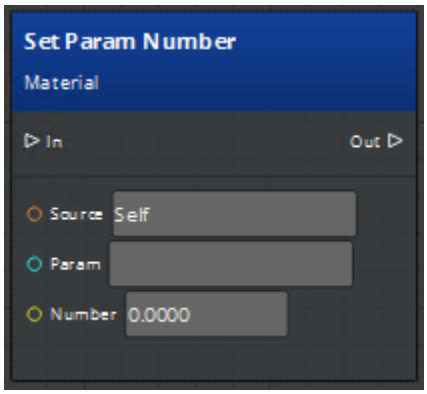
Port	Type	Description
Out	Event	Sends when the node is finished.

> **Note**
> This node corresponds to the EntityMaterialChange node (p. 936) in Flow Graph.

Set Param Number

Sets a material's numerical parameter for the specified entity.

Contents



Inputs

Port	Type	Description
In	Event	Triggers the node.
Source	EntityID	References a specific entity from which events are generated. By default, it references **Self**, the entity to which the **Script Canvas (p. 545)** component attaches the current script. You can also select another entity. For more information, see Referencing Entities for Nodes (p. 686).

Port	Type	Description
Param	String	The name of the material parameter to query. For more information, see Finding the Material Parameter Name (p. 750).
Number	Number	The new value to apply.

Outputs

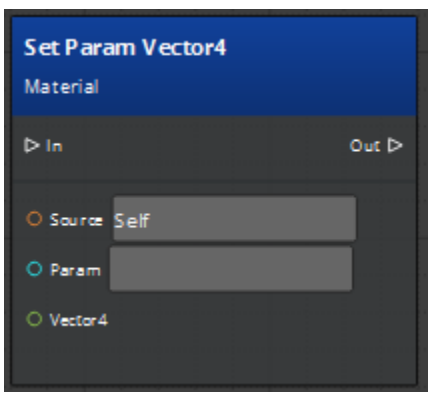
Port	Type	Description
Out	Event	Sends when the node is finished.

> **Note**
> This node corresponds to the EntityMaterialChange node (p. 936) in Flow Graph.

Set Param Vector4

Sets a material's Vector4 parameter value for the specified entity.

Contents



Inputs

Port	Type	Description
In	Event	Triggers the node.
Source	EntityID	References a specific entity from which events are generated. By default, it references **Self**, the entity to which the **Script**

Port	Type	Description
		Canvas (p. 545) component attaches the current script. You can also select another entity. For more information, see Referencing Entities for Nodes (p. 686).
Param	String	The name of the material parameter to query. For more information, see Finding the Material Parameter Name (p. 750).
Vector4	Vector4	The new value to apply.

Outputs

Port	Type	Description
Out	Event	Sends when the node is finished.

> **Note**
> This node corresponds to the EntityMaterialChange node (p. 936) in Flow Graph.

Post Effects Nodes

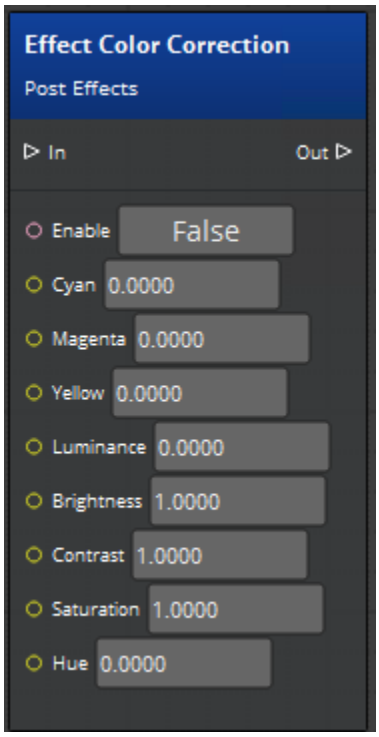
You can configure the following post effect nodes in the Amazon Lumberyard **Script Canvas** editor. Post effects are useful if you want to enhance a scene after it renders, such as color, blur, or water effects.

Topics

Effect Color Correction

Sets color correction parameters. You can use this node to specify the CMYK, brightness, contrast, saturation, and hue in a scene. Most color correction properties don't update smoothly, so we recommend that you hide stronger color correction changes by cuts or fading between scenes.

Contents



Inputs

Port	Type	Description
In	Event	Triggers the node.
Enable	Boolean	Enables color correction.
Cyan	Number	Adjusts cyan to enhance the color of the scene.
Magenta	Number	Adjusts magenta to enhance color of the scene.
Yellow	Number	Adjusts yellow to enhance color of the scene.
Luminance	Number	Adjusts luminance to enhance the color of the scene.
Brightness	Number	Adjusts brightness to enhance light and darkness of the scene.
Contrast	Number	Adjusts contrast to enhance the bias of highlights and shadows of the scene.
Saturation	Number	Adjusts saturation to enhance the color intensity of the scene.
Hue	Number	Adjusts hue to enhance the color globally.

Outputs

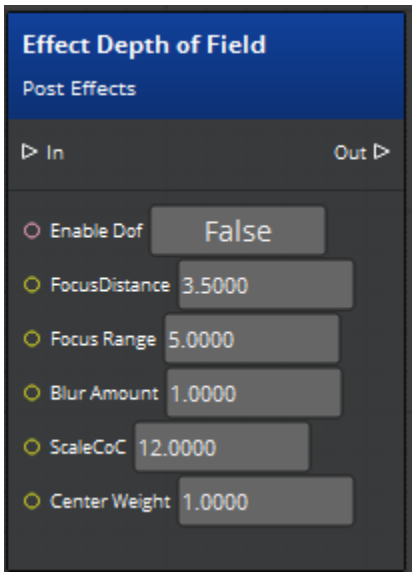
Port	Type	Description
Out	Event	Sends when the node is finished.

Note
This node corresponds to the ColorCorrection node (p. 882) in Flow Graph.

Effect Depth of Field

Configures the depth of field effect. You can use the node to add realism to scenes by simulating the way a real-world camera works. You can use a broad depth of field (DOF) 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.

Contents



Inputs

Port	Type	Description
In	Event	Triggers the node.
EnableDof	Boolean	Enables or disables depth of field effect.
FocusDistance	Number	Distance the focus is from the camera. Positive values are in front of the camera while negative values are behind the camera.

Port	Type	Description
FocusRange	Number	Distance toward and away from the camera until maximum blurriness is reached. By default, this value is twice the **FocusDistance** value.
BlurAmount	Number	Maximum blurriness value.
ScaleCoC	Number	Sets the circle of confusion scale.
CenterWeight	Number	Sets the central samples weight.

Outputs

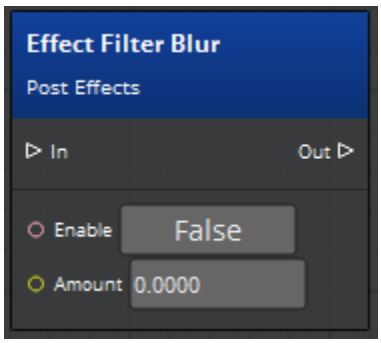
Port	Type	Description
Out	Event	Sends when the node is finished.

Note
This node corresponds to the EffectDepthOfField node (p. 884) in Flow Graph.

Effect Filter Blur

Sets the blur filter, which uses Gaussian blur.

Contents



Inputs

Port	Type	Description
In	Event	Triggers the node.
Enable	Boolean	Enables filter blur effect.

Port	Type	Description
Amount	Number	Amount of blurring.

Outputs

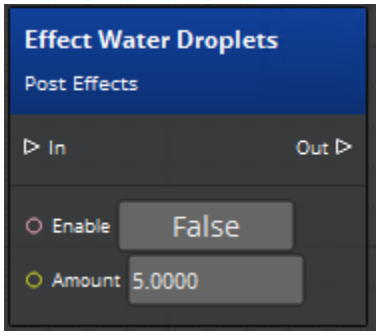
Port	Type	Description
Out	Event	Sends when the node is finished.

Note
This node corresponds to the FilterBlur node (p. 887) in Flow Graph.

Effect Water Droplets

Enable water droplets effects. For example, you can use this node when the camera goes out of the water.

Contents



Inputs

Port	Type	Description
In	Event	Triggers the node.
Enable	Boolean	Enables water droplets effect.
Amount	Number	Amount of water.

Outputs

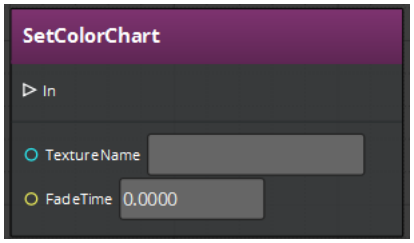
Port	Type	Description
Out	Event	Sends when the effect starts.

Note
This node corresponds to the EffectWaterDroplets node (p. 886) in Flow Graph.

SetColorChart

Applies a color chart texture for color grading.

Contents



Note
This node's functionality is not available in Lua at this time.

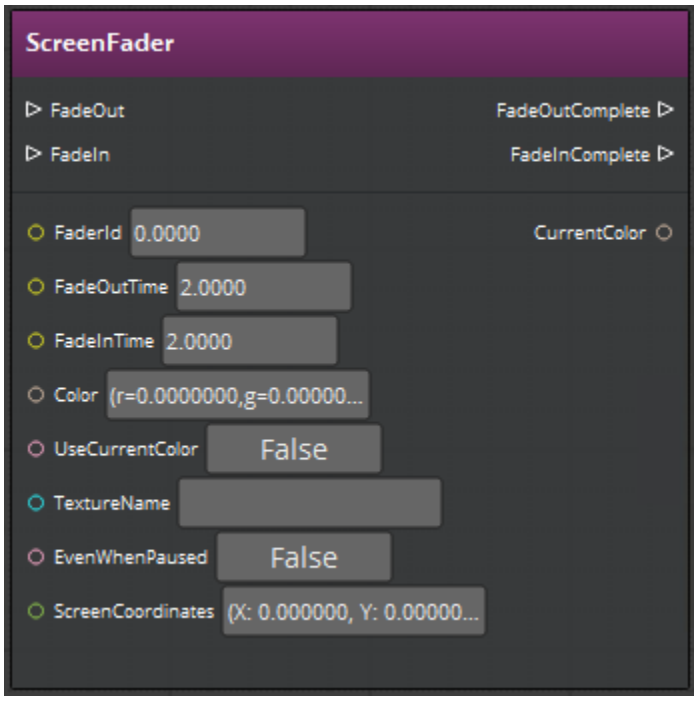
Inputs

Port	Type	Description
In	Event	Triggers the node.
TextureName	String	The name of a color chart texture. For more information, see Finding the Texture Name (p. 753).
FadeTime	Number	Number of seconds to fade into the color grading.

ScreenFader

Controls fading the screen to a color and/or texture.

Contents



Inputs

Port	Type	Description
FadeOut	Event	Triggers fading out to a color or texture.
FadeIn	Event	Triggers fading back in from a color or texture.
FaderId	Number	Specifies which fader to use. This lets you maintain separate settings and/or layer fades on top of each other.
FadeInTime	Number	Number of seconds when fading in.
FadeOutTime	Number	Number of seconds when fading out.
Color	Color	The color to fade to and from. The alpha channel is ignored. The use of this property depends on the value of **UseCurrentColor**, and whether you are fading in or fading out.
UseCurrentColor	Boolean	Continue to use the existing color for the scene. The use of the **Color** property depends on the value of

Port	Type	Description
		UseCurrentColor, and whether you are fading in or fading out. **CurrentColor** is the color that is rendered by the fader, including the alpha channel. This is generally whatever color is left over from prior fading activity. See the following Color Property Behavior (p. 743) table that shows the actual color values that are blended to cause the fading transition.
TextureName	String	The name of a texture to fade to and from. For more information, see Finding the Texture Name (p. 753).
EvenWhenPaused	Boolean	Continues fading even when the game is paused.
ScreenCoordinates	Vector4	Sets the screen coordinates where the fade mask is drawn (left, top, right, bottom). The default is full screen (`0`, `0`, `1`, `1`).

The following table shows the actual color values that are blended to cause the fading transition.

Color Property Behavior

Fading	UseCurrentColor	Start Color	Final Color
In	true	CurrentColor	CurrentColor with Alpha=0
In	false	Color with Alpha=1	Color with Alpha=0
Out	true	CurrentColor	Current with Alpha=1
Out	false	Color with Alpha=0	Color with Alpha=1

Outputs

Port	Type	Description
FadeOutComplete	Event	Sends when fade out is complete.

Port	Type	Description
FadeInComplete	Event	Sends when fade in is complete.
CurrentColor	Color	Outputs the current value for the fade mask color.

Note

- Most **Script Canvas** graphics features are available in Lua. However, **ScreenFader** is a single node in **Script Canvas**, and an EBus in Lua. For more information, see the ScreenFader Ebuses (p. 744).
- This node corresponds to the ScreenFader node (p. 891) in Flow Graph.

Topics

ScreenFader Ebuses

Provides the Lua scripting equivalent to the **ScreenFader (p. 741)** node in **Script Canvas**.

For more information about using the event bus (EBus) interface, see Event Bus (EBus) in the *Amazon Lumberyard Developer Guide*.

Contents

ScreenFaderRequestBus FadeOut

Triggers fading out to a solid color.

Parameter	Type	Description
color	Color	Color to fade out to.
duration	Float	Number of seconds for the fade to last.
useCurrentColor	Boolean	Sets the starting color of the transition. If `true` the transition begins from the current color left over from prior fading. Otherwise, the transition begins fully transparent (the current rendered output).

Parameter	Type	Description
updateAlways	Boolean	Continues fading even when the game is paused.

ScreenFaderRequestBus FadeIn

Triggers fading in to a solid color.

Parameter	Type	Description
color	Color	Color to fade through. This parameter is ignored if **useCurrentColor** is `true`.
duration	Float	Number of seconds for the fade to last.
useCurrentColor	Boolean	Sets the starting color of the transition If `true`, the transition begins from the current color left over from any prior fading. Otherwise, the transition begins from **color**.
updateAlways	Boolean	Continues fading even when the game is paused.

ScreenFaderRequestBus SetTexture

Sets a texture for the fade mask. Specify an empty string to clear the texture.

Parameter	Type	Description
textureName	String	The name of a texture to and from. For more information, see Finding the Texture Name (p. 753).

ScreenFaderRequestBus SetScreenCoordinates

Sets the screen coordinates where the fade mask is drawn.

Parameter	Type	Description
screenCoordinates	Vector4	A set of four values (left, top, right, and bottom) that define the area of the screen where the fade mask is drawn. The default is full screen (`0`, `0`, `1`, `1`).

ScreenFaderRequestBus GetCurrentColor

Returns the current color of the fade mask.

ScreenFaderNotificationBus OnFadeOutComplete

A callback function that is called when fading out is complete.

ScreenFaderNotificationBus OnFadeInComplete

A callback function that is called when fading in is complete.

ScreenFaderManagementRequestBus GetNumFaderIDs

Returns the number of available fader IDs.

Shadow Nodes

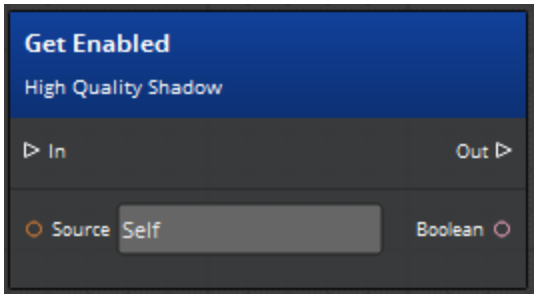
You use can use the following shadow nodes in the **Script Canvas** editor to configure shadow effects.

Topics

Get Enabled

Returns whether an entity's **High Quality Shadow (p. 503)** component is enabled.

Contents



Inputs

Port	Type	Description
In	Event	Triggers the node.
Source	EntityID	References a specific entity from which events are generated. By default, it references **Self**, the entity to which the **Script Canvas (p. 545)** component attaches the current script.

Port	Type	Description
		You can also select another entity. For more information, see Referencing Entities for Nodes (p. 686).

Outputs

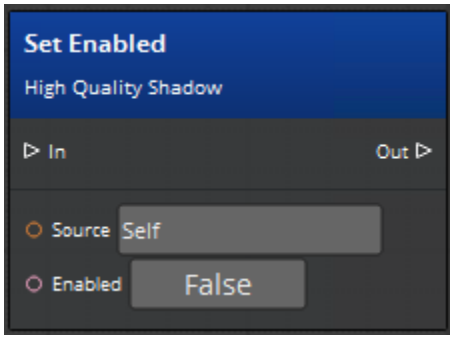
Port	Type	Description
Out	Event	Sends when the node is finished.

> **Note**
> This node corresponds to the PerEntityShadows node (p. 864) in Flow Graph.

Set Enabled

Sets whether an entity's **High Quality Shadow (p. 503)** component is enabled.

Contents



Inputs

Port	Type	Description
In	Event	Triggers the node.
Source	EntityID	References a specific entity from which events are generated. By default, it references **Self**, the entity to which the **Script Canvas (p. 545)** component attaches the current script. You can also select another entity. For more information,

Port	Type	Description
		see Referencing Entities for Nodes (p. 686).
Enabled	Boolean	Indicates whether the **Script Canvas (p. 545)** component is enabled.

Outputs

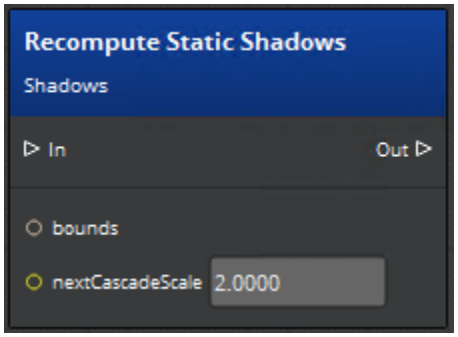
Port	Type	Description
Out	Event	Sends when the node is finished.

Note
This node corresponds to the PerEntityShadows node (p. 864) in Flow Graph.

Recompute Static Shadows

Triggers recalculation of cached shadow maps.

Contents



Inputs

Port	Type	Description
In	Event	Triggers the node.
bounds	AABB	Specifies the area where shadow maps should be recomputed.
nextCascadeScale	Number	Multiplier for scaling the bounding boxes for subsequent cached cascades. The bounding boxes are scaled versions of the preceding cascades.

Outputs

Port	Type	Description
Out	Event	Sent when the node is finished.

Note
This node corresponds to the RecomputeStaticShadows node (p. 865) in Flow Graph.

Finding the Material Name

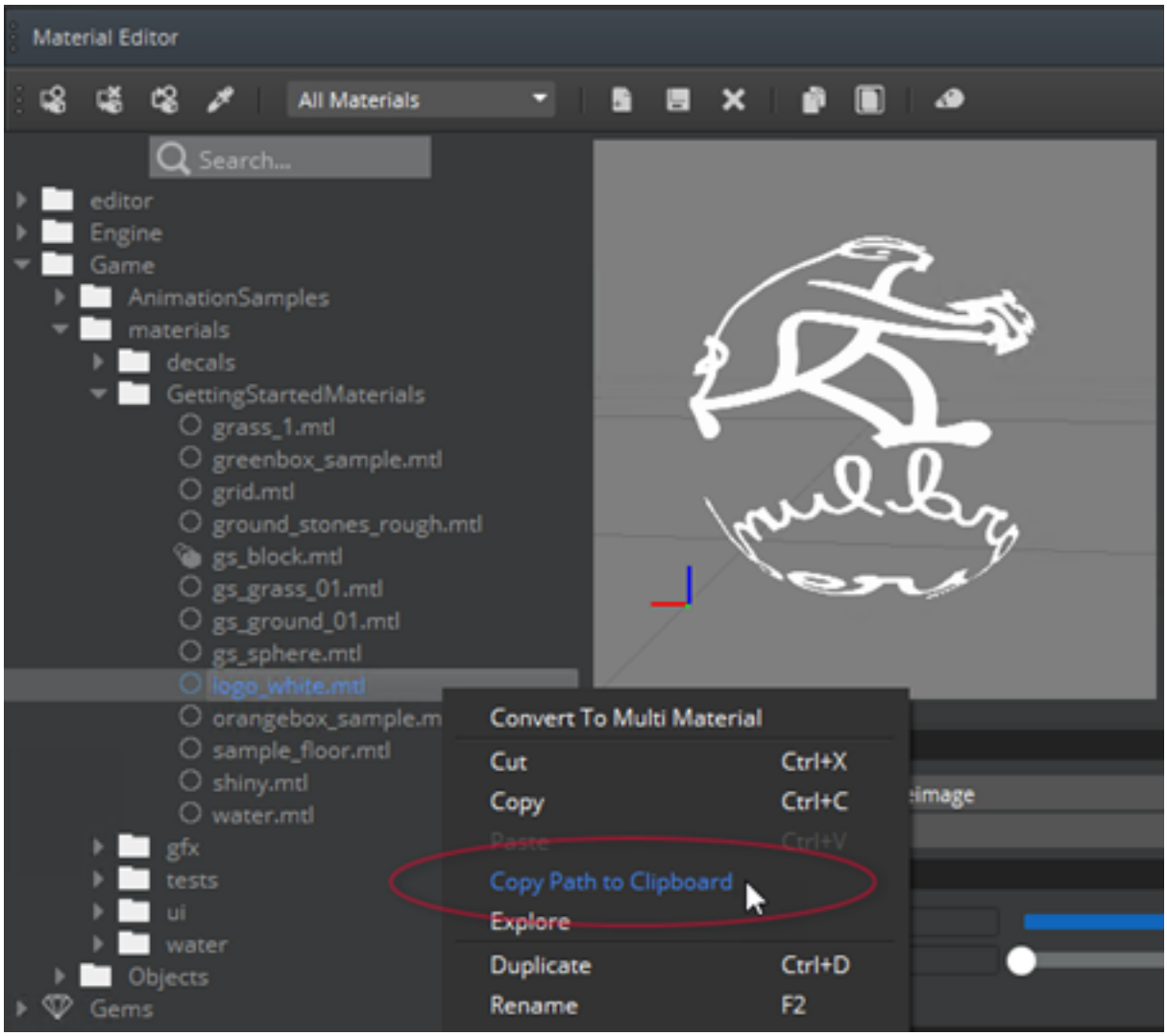
You can specify material names to reference for nodes in the **Script Canvas** editor. The material name is generally the path that starts at the top material directory but without the `.mtl` file name extension.

To find the material name

1. In Lumberyard Editor, choose **Tools**, **Material Editor**.
2. In the **Material Editor**, navigate to the material file.
3. Right-click the material file and choose **Copy Path to Clipboard**. This copies the path to the file.

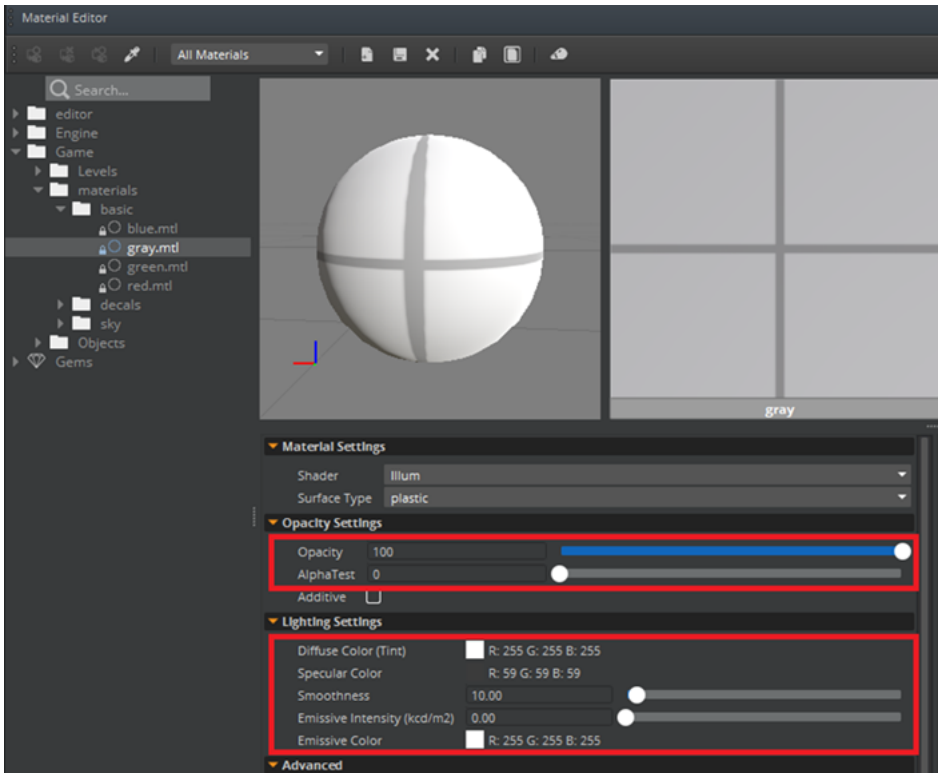
   Example

   In the Samples Project, the material name for `logo_white.mtl` is `materials/gettingstartedmaterials/logo_white`.



Finding the Material Parameter Name

You can specify material parameter names for **Script Canvas** editor nodes. In the **Material Editor**, you can find the material parameter names in the **Opacity Settings** and **Lighting Settings**.

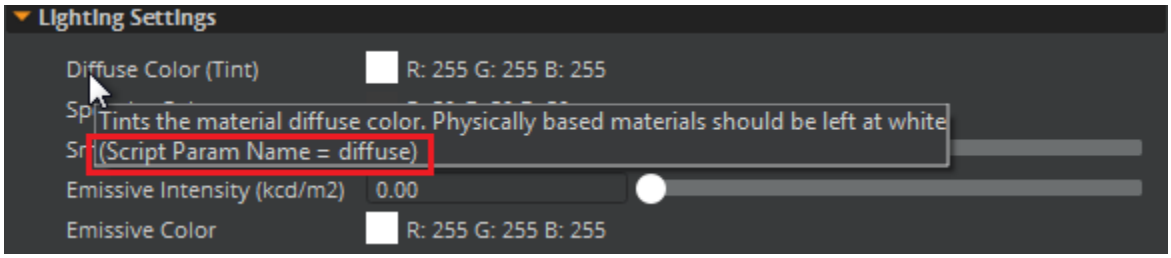


To find the material parameter name

1. In Lumberyard Editor, choose **Tools**, **Material Editor**.
2. In the **Material Editor**, navigate to the material file.
3. In the **Opacity Settings** and **Lighting Settings**, pause on a parameter to find the script parameter name.

   #### Example

   For **Diffuse Color (Tint)**, the script parameter name is **diffuse**.



The following parameter names are built in all materials and are commonly available.

Material Param Names

Param	Type	Display Name	Description	
opacity	Float	Opacity	Sets the transparency amount. Uses	

Param	Type	Display Name	Description	
			`0-99` to set **Alpha Blend** and `100` for **Opaque** and **Alpha Test**.	
alpha	Float	AlphaTest	Uses the alpha mask and refines the transparent edge. Uses `0-50` to bias toward white or `50-100` to bias toward black.	
diffuse	Color	Diffuse Color (Tint)	Tints the material diffuse color. Physically based materials should be left at white.	
specular	Color	Specular Color	Shininess and color of reflective highlights.	
shininess	Float	Smoothness	Smoothness or glossiness simulating how light bounces off the surface.	
emissive_intensity	Float	Emissive Intensity (kcd/m2)	Brightness simulating light emitting from the surface, making an object glow.	
emissive_color	Color	Emissive Color	Tints the emissive color.	

Finding Custom Parameter Names

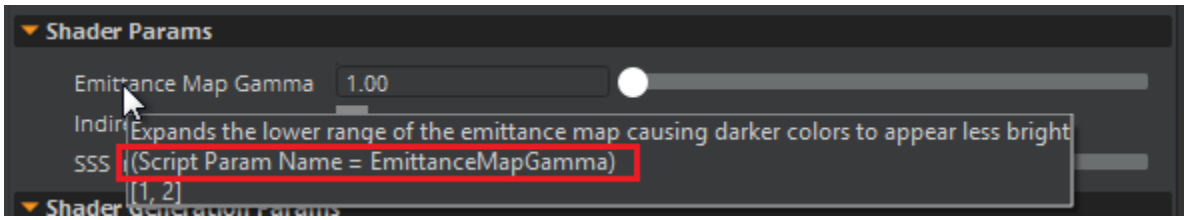
Custom parameters may also be available, as defined by each shader.

To find custom parameter names

1. In Lumberyard Editor, choose **Tools**, **Material Editor**.
2. In the **Material Editor**, navigate to the material file.
3. For **Shader Params**, pause on a parameter name to find the script parameter name.

   Example

   For **Emittance Map Gamma**, the script parameter name is **EmmitanceMapGamma**.



Finding the Texture Name

You can specify texture names for nodes in the **Script Canvas** editor. The texture name is generally the path that starts at the top asset directory.

To find the texture name

1. In Lumberyard Editor, choose **Tools**, **Asset Browser**.
2. From the **Asset Browser**, navigate to the texture file.
3. Right-click the texture file and choose **Copy Path to Clipboard**. This copies the path to the file.
4. Delete parts of the path to get the texture name.

Examples

Asset Location	Copied Path	Path to Delete	Texture Name to Reference
Project	*lumberyard_version*\dev\SamplesProject\textures\defaults\white.tif	*lumberyard_version*\dev\SamplesProject\	textures\defaults\white.tif
Gem	*lumberyard_version*\dev\Gems\Rain\Assets\Textures\Rain\rainfall.tif	*lumberyard_version*\dev\Gems\Rain\Assets\	Textures\Rain\rainfall.tif
Editor	*lumberyard_version*\dev\Editor\Plugins\ParticleEditorPlugin\defaults\feather01.tif	*lumberyard_version*\dev\	Editor\Plugins\ParticleEditorPlugin\defaults\feather01.tif
Engine	*lumberyard_version*\dev\Engine\textures\skys\night\half_moon.tif	*lumberyard_version*\dev\Engine\	textures\skys\night\half_moon.tif

Flow Graph System

Flow Graph is deprecated and will be removed in a future version of Lumberyard. Learn about Script Canvas (p. 682), Lumberyard's new visual scripting environment.

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

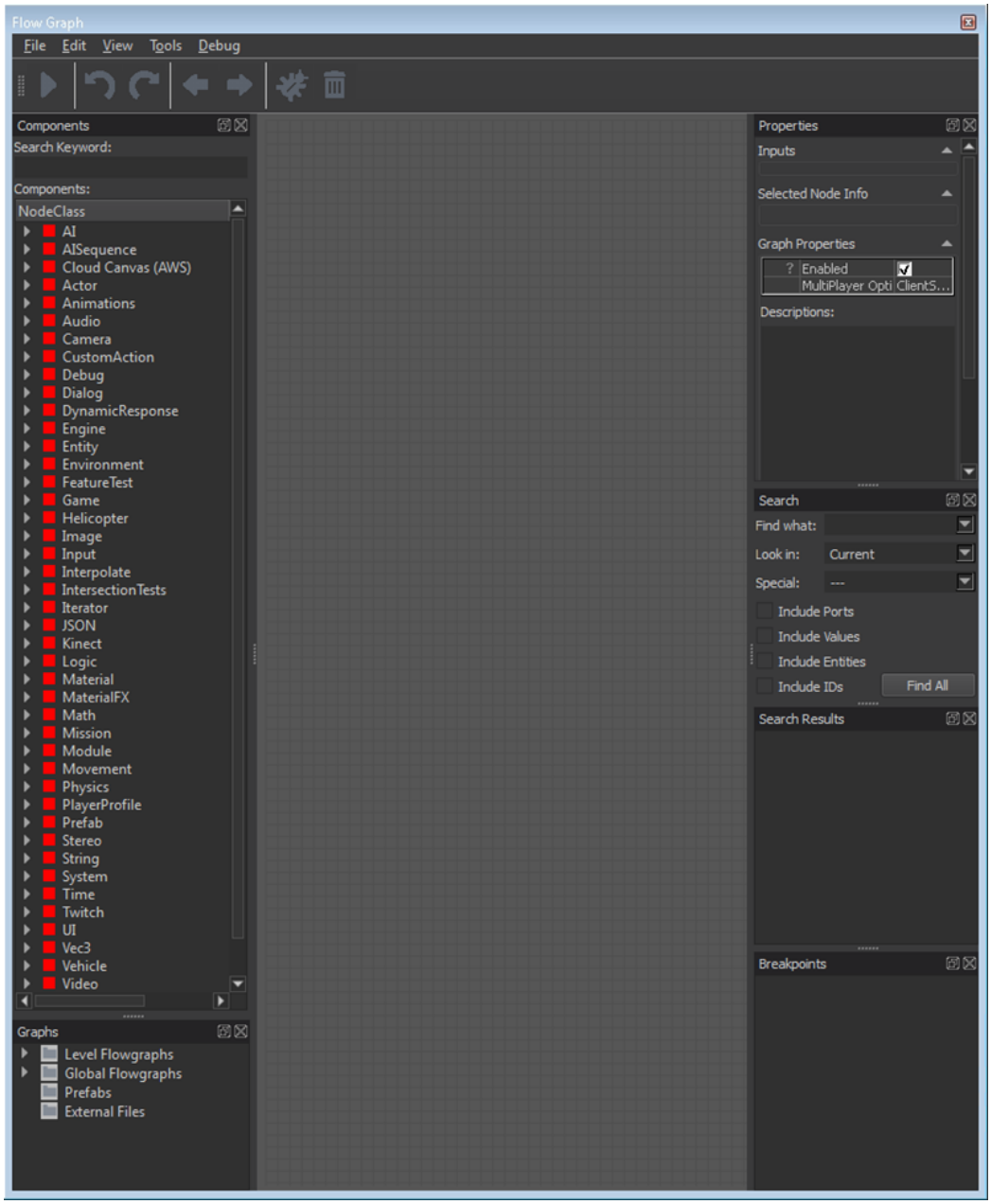
Using Flow Graph Editor

Flow Graph is deprecated and will be removed in a future version of Lumberyard. Learn about Script Canvas (p. 682), Lumberyard's new visual scripting environment.

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 is deprecated and will be removed in a future version of Lumberyard. Learn about Script Canvas (p. 682), Lumberyard's new visual scripting environment.

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

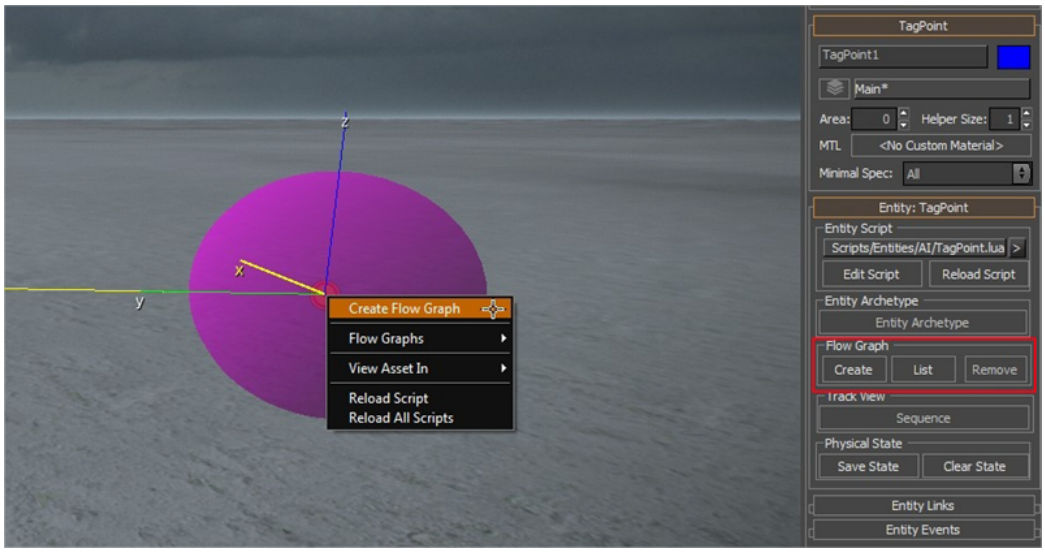
These are Flow Graph scripts that have been imported.

Managing Flow Graphs

> Flow Graph is deprecated and will be removed in a future version of Lumberyard. Learn about Script Canvas (p. 682), Lumberyard's new visual scripting environment.

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 it's 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

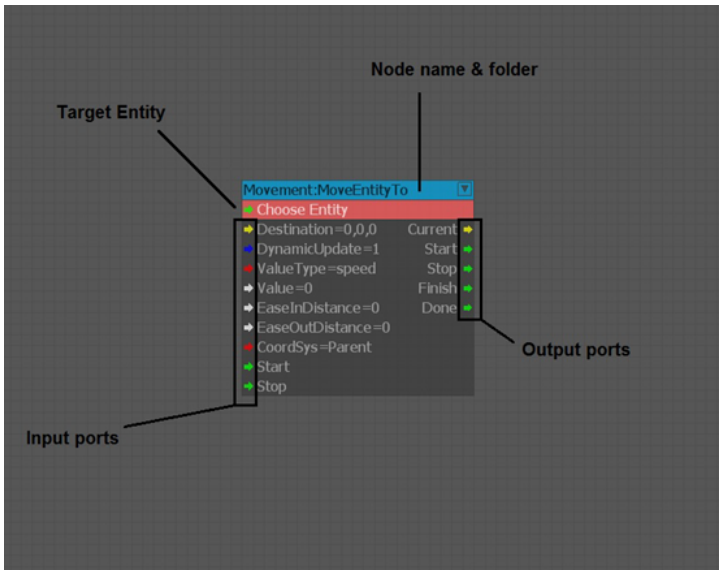
- Open the target flow graph you want to import to add the exported flow graph nodes to, right-click anywhere in the graph pane, click **Import**, then enter the name of file you want to import..

The imported flow graph is positioned relative to the old flow graph.

Using Flow Graph Nodes

Flow Graph is deprecated and will be removed in a future version of Lumberyard. Learn about Script Canvas (p. 682), Lumberyard's new visual scripting environment.

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 (keyboard 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

Flow Graph is deprecated and will be removed in a future version of Lumberyard. Learn about Script Canvas (p. 682), Lumberyard's new visual scripting environment.

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_your_flow_node_name : public CFlowBaseNode<eNCT_Instanced>
{
public:
  CFlowNode_your_flow_node_name(SActivationInfo* pActInfo)
  {
  };

  virtual IFlowNodePtr Clone(SActivationInfo *pActInfo)
  {
    return new CFlowNode_your_flow_node_name(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 comptability 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_your_flow_node_name : public CFlowBaseNode<eNCT_Instanced>
{
public:
  // ...

  virtual void GetConfiguration( SFlowNodeConfig& config )
  {
    static const SInputPortConfig in_config[] = {
      InputPortConfig<int>("your_input", _HELP("your_help_text")),
      {0}
    };
    static const SOutputPortConfig out_config[] = {
      {0}
    };
    config.sDescription = _HELP( "your_flow_node_tooltip_description" );
    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 comptability 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 Drop-down List

There are several types of enums that you can use to display a drop-down 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	
aianchor	IVariable::DT_AI_ANCHOR	
aibehavior	IVariable::DT_AI_BEHAVIOR	
aicharacter	IVariable::DT_AI_CHARACTER	
aipfpropertieslist	IVariable::DT_AI_PFPROPERTIESLIST	
aientityclasses	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	
soevent	IVariable::DT_SOEVENT	
customaction	IVariable::DT_CUSTOMACTION	
gametoken	IVariable::DT_GAMETOKEN	
mat	IVariable::DT_MATERIAL	

Editor Name	Editor Type	
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	
actionFilter	IVariable::DT_USERITEMCB	
actionMaps	IVariable::DT_USERITEMCB	
actionMapActions	IVariable::DT_USERITEMCB	
geomcache	IVariable::DT_GEOM_CACHE	

Editor Name	Editor Type	
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

Flow Graph is deprecated and will be removed in a future version of Lumberyard. Learn about Script Canvas (p. 682), Lumberyard's new visual scripting environment.

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. 1626), Setting Up Virtual Reality with Flow Graph (p. 1791), 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

Flow Graph is deprecated and will be removed in a future version of Lumberyard. Learn about Script Canvas (p. 682), Lumberyard's new visual scripting environment.

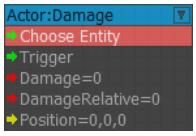
You can use the following flow graph nodes to configure various actor behaviors and settings.

> **Note**
> These nodes will only work with the Legacy Game Sample (CryEngine GameSDK), which is available at Lumberyard Downloads.

Topics

Damage node

Used to damage the chosen entity using the **Damage** input value when the **Trigger** input is activated.

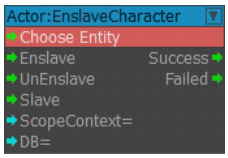


Inputs

Port	Type	Description
Trigger	Any	Activates the node
Damage	Integer	Type of damage to inflict
DamageRelative	Integer	Level of relative damage
Position	Vec3	Location where the damage occurs

EnslaveCharacter node

Used to enslave one character to another character.



Inputs

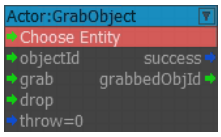
Port	Type	Description
Enslave	Any	Enslaves the character
Unenslave	Any	Frees the character
Slave	Any	Character to enslave
ScopeContext	String	Context of the scope
DB	String	Optional database name

Outputs

Port	Type	Description
Success	Any	Tigger for successful enslaving
Failed	Any	Trigger for failed enslaving

GrabObject node

Used by the chosen entity to grab an object, then to drop or throw the object.



Inputs

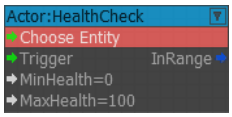
Port	Type	Description
objectId	Any	ID of the object to grab
grab	Any	Grabs the object
drop	Any	Drops the object
throw	Boolean	Throws the object

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

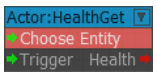
Port	Type	Description
Trigger	Any	Activates the port
MinHealth	Float	Lower limit of health range
MaxHealth	Float	Upper limit of health range

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

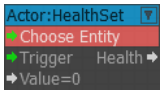
Port	Type	Description
Trigger	Any	Activate this port to get the current health of the chosen entity

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
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

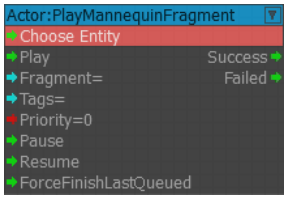
Port	Type	Description
update	Any	Updates the ID of the local player entity; required for multiplayer games

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

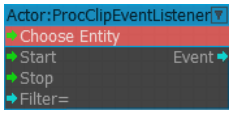
Port	Type	Description
Priority	Integer	Priority number
Pause	Any	Pauses the entity actionController
Resume	Any	Resumes the entity actionController
ForceFinishLastQueued	Any	Finishes the last queued action

Outputs

Port	Type	Description
Success	Any	Trigger for successful fragment command
Failed	Any	Trigger for failed fragment command

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

Flow Graph is deprecated and will be removed in a future version of Lumberyard. Learn about Script Canvas (p. 682), Lumberyard's new visual scripting environment.

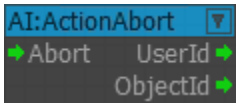
You can use these flow graph nodes to configure AI agent behaviors and settings.

> **Note**
> These nodes will only work with the Legacy Game Sample (CryEngine GameSDK), which is available at Lumberyard Downloads.

Topics

ActionAbort node

Used to define a "clean-up" procedure that is run when an AI action is aborted.



Inputs

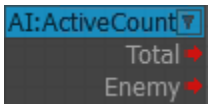
Port	Type	Description
Abort	Any	Cancels execution of AI action

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

ActiveCount node

Used to count how many AI agents are active.

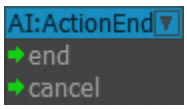


Outputs

Port	Type	Description
Total	Integer	Number of active agents
Enemy	Integer	Number of enemies

ActionEnd node

Used to end an AI action.

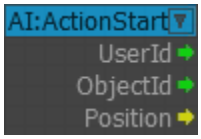


Inputs

Port	Type	Description
End	Any	Ends the AI action
Cancel	Any	Cancels the action

ActionStart node

Used to start an AI action.

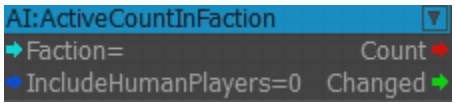


Outputs

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
Position	Vec3	Position of the object

ActiveCountInFaction node

Used to count how many AI factions are active.



Inputs

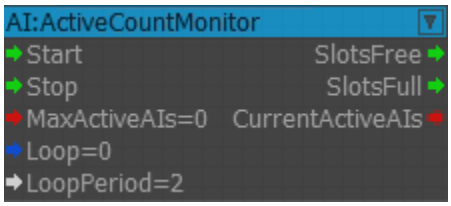
Port	Type	Description
Faction	String	Factions to be counted
IncludedHumanPlayers	Boolean	Include human players when counting active AI agents in the faction

Outputs

Port	Type	Description
Count	Integer	Number of active agents in the faction
Changed	Any	Trigger for changes in the number of active agents

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

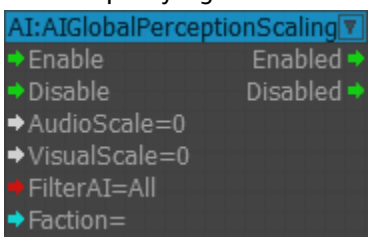
Port	Type	Description
Stops	Any	Stops monitoring
MaxActiveAIs	Integer	Maximum number of active AIs
Loop	Boolean	Enables loop monitoring
LoopPeriod	Float	Period of time between checks if Loop is enabled

Outputs

Port	Type	Description
SlotsFree	Any	Triggers when the number of active agents drops below MaxActiveAIs
SlotsFull	Any	Triggers when the number of active agents is equal to or above MaxActiveAIs
CurrentActiveAIs	Integer	Current number of active AI agents

AIGlobalPerceptionScaling node

Used to specify a global scale for AI perception.



Inputs

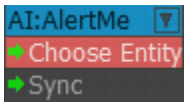
Port	Type	Description
Enable	Any	Enables perception scaling
Disable	Any	Disables perception scaling
AudioScale	Float	Auditory perception scaling factor
VisualScale	Float	Visual perception scaling factor
FilterAI	Integer	Filter which AI agents are used
Faction	String	Faction

Outputs

Port	Type	Description
Enabled	Any	Triggers when node is enabled
Disabled	Any	Triggers when node is disabled

AlertMe node

A generic AI signal.

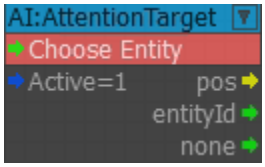


Inputs

Port	Type	Description
Sync	Any	Generic AI signal

AttentionTarget node

Used to output an AI agent's attention target.



Inputs

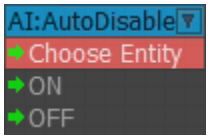
Port	Type	Description
Active	Boolean	Activates the node

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.

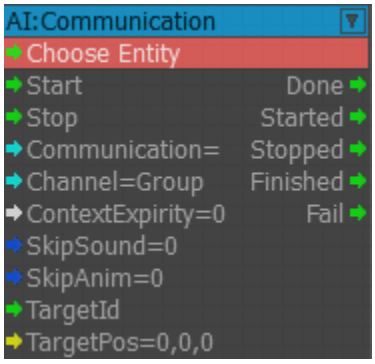


Inputs

Port	Type	Description
On	Any	Enables autodisabling
Off	Any	Disables autodisabling

Communication node

Used to specify the communication that an AI agent plays.



Inputs

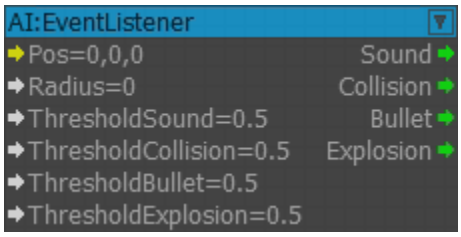
Port	Type	Description
Start	Any	Starts communication
Stop	Any	Stops communication
Communication	String	Name of communication to play
Channel	String	Name of channel to play the communications in
ContextExpirity	Float	Time that must elapse before communiction can be played again
SkipSound	Boolean	Skips 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 has started
Stopped	Any	Triggers if communication has 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

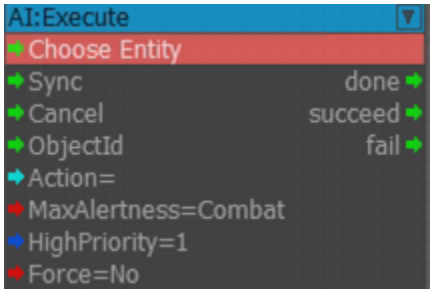
Port	Type	Description
Pos	Vec3	Position of the listener
Radius	Float	Listening radius of the listener
ThresholdSound	Float	Sensitivity of the sound output
ThresholdCollision	Float	Sensitivity of the collision output
ThresholdBullet	Float	Sensitivity of the bullet output
ThresholdExplosion	Float	Sensitivity of the explosion output

Outputs

Port	Type	Description
Sound	Any	Trigger for a sound event
Collision	Any	Trigger for a collision event
Bullet	Any	Trigger for a bullet event
Explosion	Any	Trigger 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

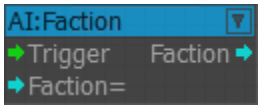
Port	Type	Description
ObjectId	Any	ID of the entity that receives the action that is executed
Action	String	Action to be executed
MaxAlertness	Integer	Maximum alertness that allows execution
HighPriority	Boolean	Action priority level
Force	Integer	Force execution method

Outputs

Port	Type	Description
Done	Any	Triggers when the action has been executed
Succeed	Any	Triggers if the action has been executed
Fail	Any	Triggers if the action has not been executed

Faction node

Used to trigger an AI faction.



Inputs

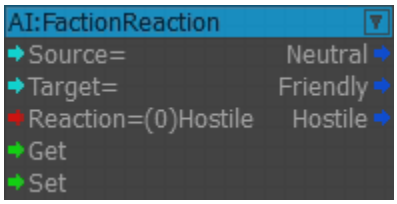
Port	Type	Description
Trigger	Any	Tiggers the output
Faction	String	Name of faction to trigger

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

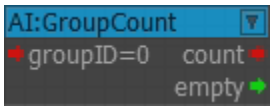
Port	Type	Description
GroupId	Integer	ID of group to set alertness level for

Outputs

Port	Type	Description
Alertness	Integer	Alertness level of the group
Green	Any	Trigger for green alertness level
Orange	Any	Trigger for orange alertness level
Red	Any	Trigger for red alertness level
PlayerSighted	Any	Triggers if the player has been sighted

GroupCount node

Used to output the AI agent count in a group.



Inputs

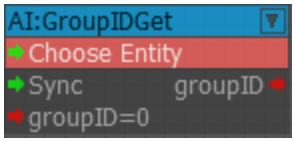
Port	Type	Description
GroupId	Integer	Agent group ID

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

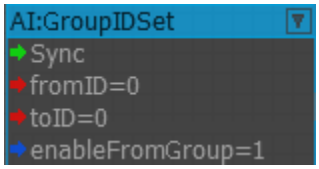
Port	Type	Description
Sync	Any	Triggers the output
GroupId	Integer	Group ID

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

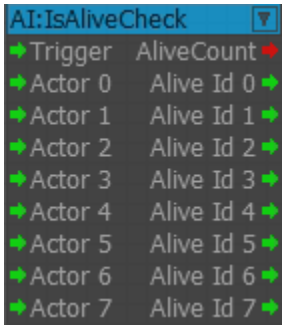
Used to make an AI agent ignore enemies or to be ignored.

Inputs

Port	Type	Description
Hostile	Any	Activates the node
Ignore	Any	Agent ignores enemies
ResetPerception	Any	Resets Ignore state

IsAliveCheck node

Used to check which AI actors of a group are active.



Inputs

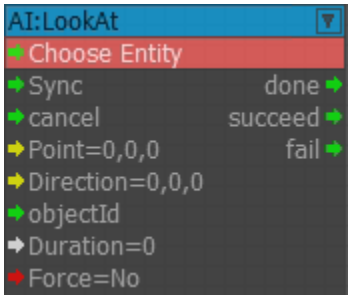
Port	Type	Description
Trigger	Any	Triggers the node
Actor 0 - 7	Any	Specific actors to check

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

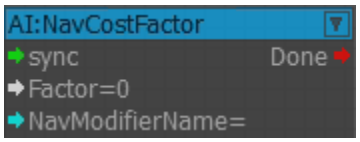
Port	Type	Description
Sync	Any	Activates the node
Cancel	Any	Cancels the operation
Point	Vec3	Point for agent to look at
Direction	Vec3	Direction for agent to look along
ObjectId	Any	ID of object for agent to look at
Duration	Float	Time in seconds for agent to look
Force	Integer	Force execution method

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

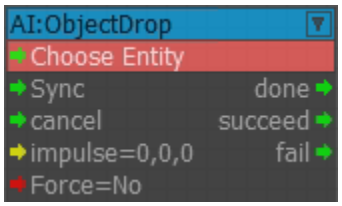
Port	Type	Description
Sync	Any	Activates the node
Factor	Float	Navigation cost factor
NavModifierName	String	Name of the cost factor navigation modifier

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

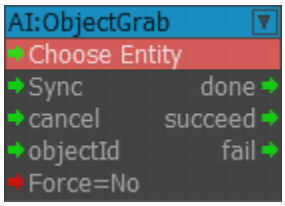
Port	Type	Description
Sync	Any	Activates the node
Cancel	Any	Cancels the operation
Impulse	Vec3	Impulse strength for dropping object
Force	Integer	Force execution method

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

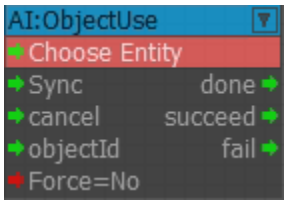
Port	Type	Description
Sync	Any	Activates the node
Cancel	Any	Cancels the operation
ObjectId	Any	Object to be grabbed
Force	Integer	Force execution method

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

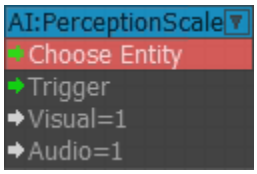
Port	Type	Description
Sync	Any	Activates the node
Cancel	Any	Cancels the operation
ObjectId	Any	Object to be used
Force	Integer	Force execution method

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.

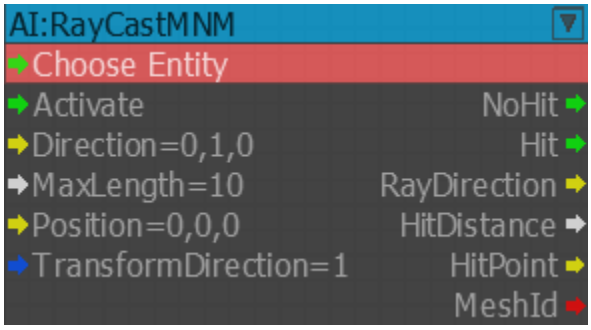


Inputs

Port	Type	Description
Trigger	Any	Triggers the node
Visual	Float	Visual perception scale factor
Audio	Float	Auditory perception scale factor

RayCastMNM node

Performs a raycast to the AI multilayer navigation mesh relative to an entity.



Inputs

Port	Type	Description
Choose Entity	Any	Changes the attached entity dynamically
Activate	Any	Triggers the node
Direction	Vec3	The direction of the raycast
MaxLength	Float	The maximum length of the raycast

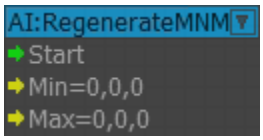
Port	Type	Description
Position	Vec3	The ray start position, relative to the entity
TransformDirection	Boolean	Whether the direction is transformed by entity orientation

Outputs

Port	Type	Description
NoHit	Any	Activated if no object was hit by the raycast
Hit	Any	Activated if an object was hit by the raycast
RayDirection	Vector	Actual direction of the cast ray, possibly transformed by entity rotation (assumes Hit)
HitDistance	Float	Distance to the hit object (assumes Hit)
HitPoint	Vector	Position of the hit (assumes Hit)
MeshId	Integer	The mesh ID of the navigation mesh hit

RegenerateMNM node

Used to regenerate the AI multi-navigation mesh.

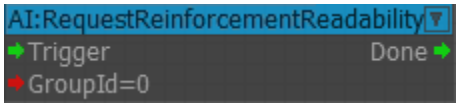


Inputs

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

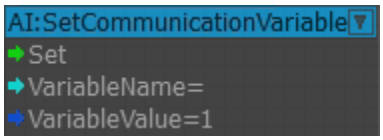
Port	Type	Description
Trigger	Any	Triggers the node
GroupId	Integer	ID of the group that is notified

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.

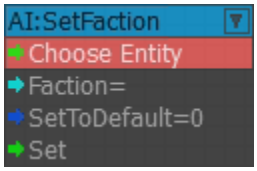


Inputs

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.

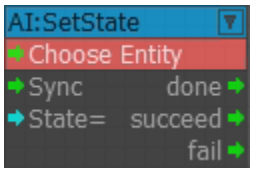


Inputs

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

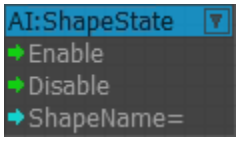
Port	Type	Description
Sync	Any	Activates the node
State	String	Smart object state to be set

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.

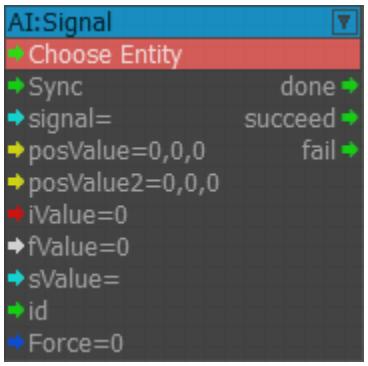


Inputs

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

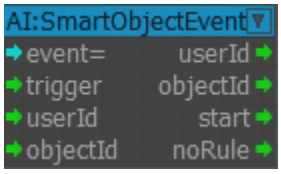
Port	Type	Description
Sync	Any	Activates the node
Signal	String	Name of the signal to be sent
PosValue	Vec3	Position value 1 of fhe signal
PosValue2	Vec3	Position value 2 of the signal
IValue	Integer	Integer value of the signal
FValiue	Float	Floating point value of the signal
SValue	String	String value 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
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

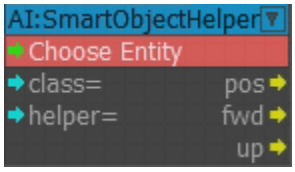
Port	Type	Description
Event	String	Smart object event to be triggered
Trigger	Any	Triggers the event
UserId	Any	Limits event to specific user ID
ObjectId	Any	Limits event to specific object

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

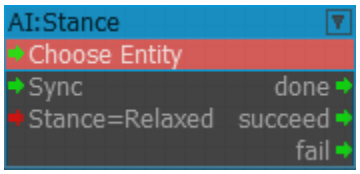
Port	Type	Description
Class	String	Class of smart object helper
Helper	String	Name of smart object helper

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

Flow Graph is deprecated and will be removed in a future version of Lumberyard. Learn about Script Canvas (p. 682), Lumberyard's new visual scripting environment.

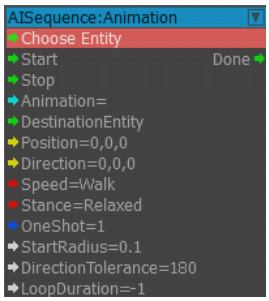
You can use these flow graph nodes to configure AI sequence behaviors and settings. All AI sequence nodes must be executed with the **AISequence:Start** node.

> **Note**
> These nodes will only work with the Legacy Game Sample (CryEngine GameSDK), which is available at Lumberyard Downloads.

Topics

Animation node

Used to make an AI agent move to a specific location and play an animation.



Inputs

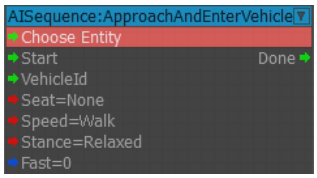
Port	Type	Description
Start	Any	Starts the animation
Stop	Any	Stops the animation
Animation	String	Name of the animation
DestinationEntity	Any	Destination to move to
Position	Vec3	Position to move to
Direction	Vec3	Direction to move along
Speed	Integer	Speed of movement
Stance	Integer	Stance of the agent 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

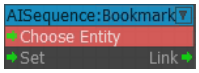
Port	Type	Description
Start	Any	Allows the AI agent to move to, and enter the specified vehicle
VehicleID	Any	Vehicle to be entered
Seat	Integer	Seat to be entered
Speed	Integer	Speed at which the AI agent approaches the vehicle
Stance	Integer	Stance of the agent when approaching the vehicle
Fast	Boolean	Skips the approaching animation

Outputs

Port	Type	Description
Done	Any	Output 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

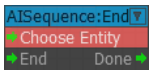
Port	Type	Description
Set	Any	Sets a bookmark for the AI sequence from which to resume from

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

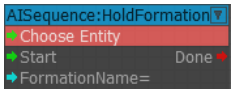
Port	Type	Description
End	Any	Triggers when the AI sequence ends

Outputs

Port	Type	Description
Done	Any	Output 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	Triggers when the formation is complete

JoinFormation node

Use to have an AI agent join a formation.

Inputs

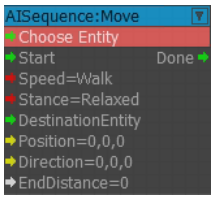
Port	Type	Description
Start	Any	Start formation join
LeaderId	Any	ID of the leader

Outputs

Port	Type	Description
Done	Any	Triggers when formation join is complete

Move node

Use to command an AI agent to move to a location.



Inputs

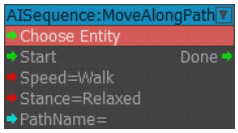
Port	Type	Description
Start	Any	Start movement
Speed	Integer	Movement speed
Stance	Integer	Stance of the agent while moving
DestinationEntity	Any	Destination entity to move to
Position	Vec3	Position to move to
Direction	Vec3	Direction to move along
EndDistance	Float	End distance to move to

Outputs

Port	Type	Description
Done	Any	Triggers 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

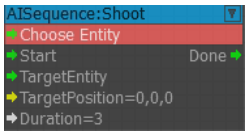
Port	Type	Description
Speed	Integer	Speed of the agent
Stance	Integer	Stance of the agent while following the path
PathName	String	Name of the path the agent follows

Outputs

Port	Type	Description
Done	Any	Triggers 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

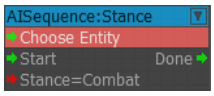
Port	Type	Description
Start	Any	Starts the AI agent shooting
TargetEntity	Any	Entity the agent shoots at
TargetPosition	Vec3	Position the agent shoots at
Duration	Float	Length of shooting time

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 AI agent stance

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

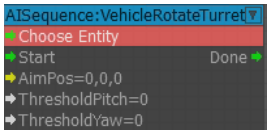
Port	Type	Description
Start	Any	Starts the AI sequence
Interruptible	Boolean	Automatically stops the agent when not in the Idle state.
ResumeAfterInterruption	Boolean	When the Idle state ends, AI sequence automatically resumes from the start or from the agent's bookmark ID

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 that the turret aims a
ThresholdPitch	Float	Pitch angle threshold at which the output port is triggered; must be used with the ThresholdYaw port

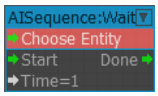
Port	Type	Description
ThresholdYaw	Float	Yaw angle threshold at which the output port is triggered; must be used with the ThresholdPitch port

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	Triggers when wait is finished

Animations Nodes

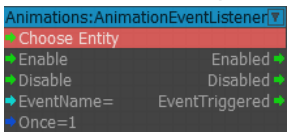
Flow Graph is deprecated and will be removed in a future version of Lumberyard. Learn about Script Canvas (p. 682), Lumberyard's new visual scripting environment.

You can use these flow graph nodes to configure animation-related settings.

Topics

AnimationEventListener node

Use to listen for a specific animation event and trigger the output.



Inputs

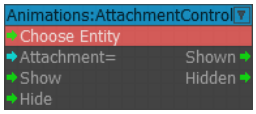
Port	Type	Description
Enable	Any	Starts listening for animation events
Disable	Any	Stops listening for animation events
EventName	String	Name of the animation event to listen for
Once	Boolean	If set to 1 (true), the node is disabled after the event has been received.

Outputs

Port	Type	Description
Enabled	Any	Triggers when listening has started
Disabled	Any	Triggers when listening has stopped
EventTriggered	Any	Triggers 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	Shows the attachment

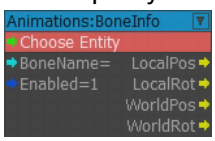
Port	Type	Description
Hide	Any	Hides the attachment

Outputs

Port	Type	Description
Shown	Any	Triggers when the attachment is shown
Hidden	Any	Triggers when the attachment is hidden

BoneInfo node

Use to specify and output character bones to create attachments or link objects.



Inputs

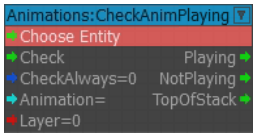
Port	Type	Description
BoneName	String	Name of the bone to get information for
Enabled	Boolean	Enables and disables the node

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

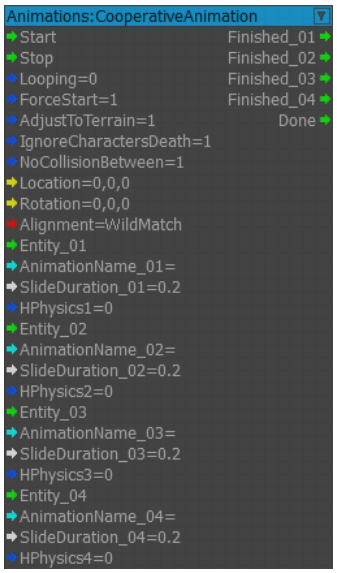
Port	Type	Description
CheckAlways	Boolean	Checks each frame whether the animation is playing
Animation	String	Name of the animation
Layer	Integer	Specifies which layer should play the animation

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

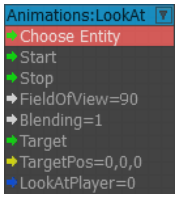
Port	Type	Description
IgnoreCharactersDeath		If false and any of the actors die, stops the animation for all the actors
NoCollisionBetween		If true, the first actor won't collide with the other actors
Location		Starts the animation at a specific location
Rotation		Starts the animation at a specific rotation
Alignment		Alignment type: • **WildMatch**: Moves both characters the least amount • **FirstActor**: Allows first actor to be rotated but not moved • **FirstActorNoRot**: Prevents first actor from being 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 - SlideDuraction_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	Triggers when the specific actor is done
Done	Any	Triggers when all actors are done

LookAt node

Use to make a character look at a position.

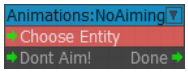


Inputs

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	Target for the character to look at
TargetPos	Vec3	Target look position
LookAtPlayer	Boolean	Character looks at player

NoAiming node

Use to suppress aiming for a character.



Inputs

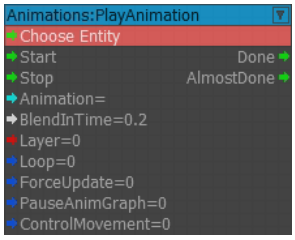
Port	Type	Description
Dont Aim!	Any	Suppresses aiming for a character

Outputs

Port	Type	Description
Done	Any	Triggers 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

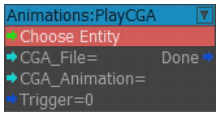
Port	Type	Description
Stop	Any	Stops the animation
Animation	String	Name of the animation to play
BlendInTime	Float	Blend-in time in seconds
Layer	Integer	Layer on which to play the animation
Loop	Boolean	Loops the animation indefinitely
ForceUpdate	Boolean	Plays animation even if not visible
PauseAnimGraph	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

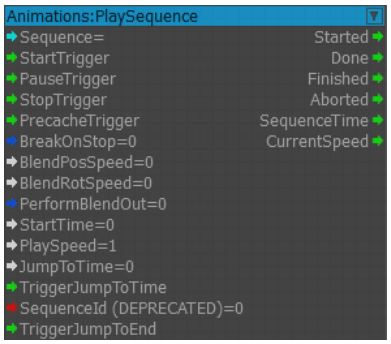
Port	Type	Description
CGA_File	String	File name of the animation
CGA_Animation	String	Name of the animation
Trigger	Boolean	Starts the animation

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	Precaches 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 end of the cut scene dissolves to the new view
StartTime	Float	Time at which the sequence starts playing
PlaySpeed	Float	Speed at which the sequence plays
JumpToTime	Float	Jumps to a specific time in the sequence
TriggerJumpToTime	Any	Triggers the animation to jump to the specified sequence time
TriggerJumpToEnd	Any	Triggers the animation to jump to the end of the sequence.

Outputs

Port	Type	Description
Started	Any	Triggers when the animation starts
Done	Any	Triggers when the animation has stopped or is aborted

Port	Type	Description
Finished	Any	Triggers when the animation has stopped
Aborted	Any	Triggers 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

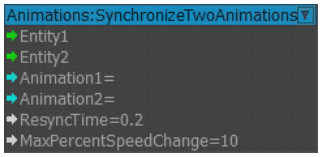
Port	Type	Description
Stop!	Any	Stops the animation

Outputs

Port	Type	Description
Done	Any	Triggers when the animation has stopped

SynchronizeTwoAnimations node

Use to synchronize two animations for two entities.

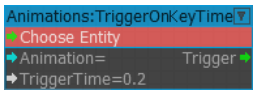


Inputs

Port	Type	Description
Entity1	Any	First entity to synchronize
Entity2	Any	Second entity to synchronize
Animation1	Srring	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

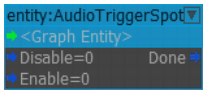
Flow Graph is deprecated and will be removed in a future version of Lumberyard. Learn about Script Canvas (p. 682), Lumberyard's new visual scripting environment.

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

Used to enable and disable the associated entity.



Inputs

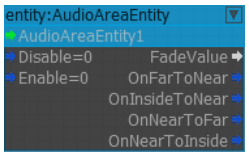
Port	Type	Description
Disable	Boolean	Stops the sound. If available, triggers the event set in the **StopTriggerName** property.
Enable	Boolean	Starts the sound. Triggers the event set in the **PlayTriggerName** property.

Outputs

Port	Type	Description
Done	Boolean	Triggers 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

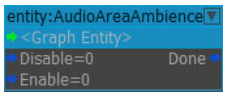
Port	Type	Description
Disable	Boolean	Stops the sound. If available, triggers the event set in the **StopTriggerName** property.
Enable	Boolean	Starts the sound. Triggers the event set in the **PlayTriggerName** property.

Outputs

Port	Type	Description
FadeValue	Float	Normalized value from 0 to 1 of the **FadeDistance** when the player approaches the shape
OnFarToNear	Boolean	Triggers when player enters the fade distance
OnInsideToNear	Boolean	Triggers when player leaves the shape
OnNearToFar	Boolean	Triggers when player leaves the fade distance
OnNearToInside	Boolean	Triggers when player enters the shape

entity:AudioAreaAmbience node

Used to enable and disable the associated entity.



Inputs

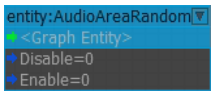
Port	Type	Description
Disable	Boolean	Disables the audio entity
Enable	Boolean	Enables the audio entity

Outputs

Port	Type	Description
Done	Boolean	Triggers 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

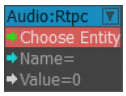
Port	Type	Description
Preload Request	String	Defines the preload requests that should be loaded or unloaded
Load	Any	Loads the preload requests

Port	Type	Description
Unload	Any	Unloads 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.



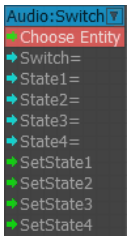
Inputs

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	Triggers when the event is complete

Camera Nodes

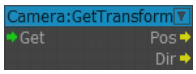
Flow Graph is deprecated and will be removed in a future version of Lumberyard. Learn about Script Canvas (p. 682), Lumberyard's new visual scripting environment.

You can use the following flow graph nodes to configure player camera settings.

Topics

GetTransform node

Used to get and output the position and direction of the player camera.



Inputs

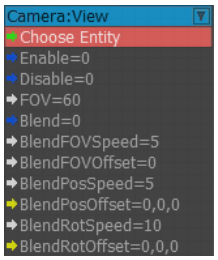
Port	Type	Description
Get	Any	Triggers the retrieval of the currently active camera position and direction

Outputs

Port	Type	Description
Pos	Vec3	Outputs camera position
Dir	Vec3	Outputs camera direction

View node

Used to create a custom view linked to the chosen entity.

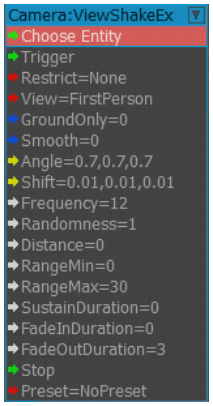


Inputs

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	Applies shake only when the player is standing on the ground
Smooth	Boolean	Anys 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

Flow Graph is deprecated and will be removed in a future version of Lumberyard. Learn about Script Canvas (p. 682), Lumberyard's new visual scripting environment.

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.

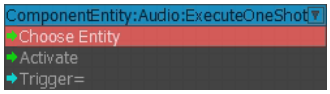
In addition, with the following exceptions, flow graph nodes that have an **Entity ID** input port do not work with the component entity system nodes:

- **Physics:ActionImpulse** node
- **Physics:Dynamics** node
- **Movement:RotateEntity** node
- **Entity:EntityID** node

Topics

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
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

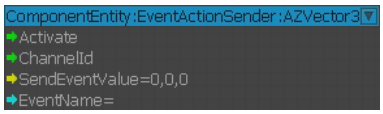
Port		Description
ChannelId	Any	Entity channel ID
EventName	String	Name of the event action handler
Enable	Any	Enables the event action handler
Disable	Any	Disables the event action handler

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.

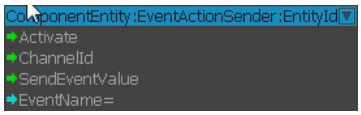


Inputs

Port		Description
Activate	Any	Activates the node
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

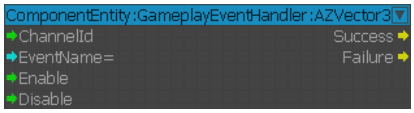
Used for the entity event action sender.

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

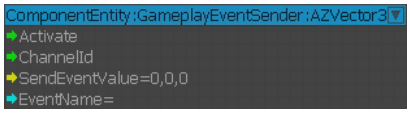
Port		Description
ChannelId	Any	Entity channel ID
EventName	String	Name of the gameplay event handler
Enable	Any	Enables the gameplay event handler
Disable	Any	Disables the gameplay event handler

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.

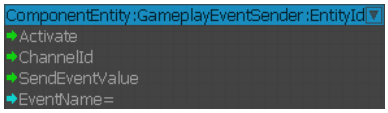


Inputs

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.

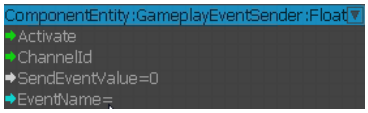


Inputs

Port		Description
Activate	Any	Activates the node
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

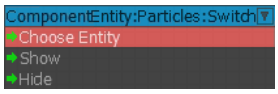
Used to turn the light entity on or off.

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.

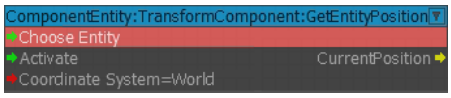


Inputs

Port		Description
Show	Any	Displays the particle
Hide	Any	Hides the particle

TransformComponent:GetEntityPosition node

Used to get the entity position.



Inputs

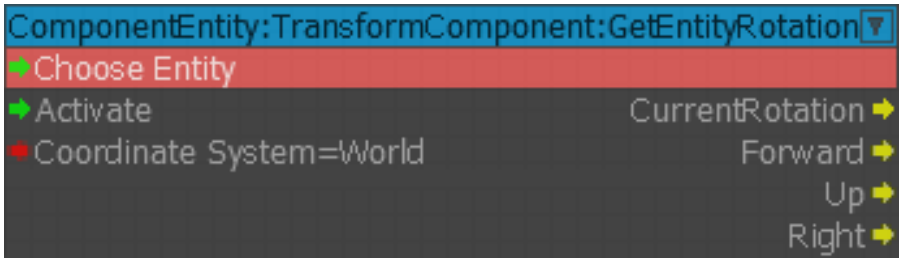
Port		Description
Activate	Any	Activates the node
Coordinate System	Integer	Coordinate system used

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
Coordinate System	Integer	Coordinate system used

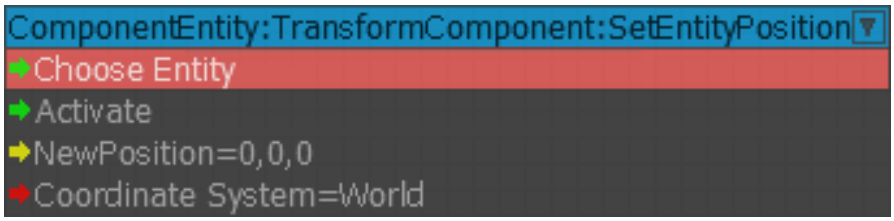
Outputs

Port		Description
CurrentRotation	Vec3	Current entity rotation

Port		Description
Forward	Vec3	Entity forward position
Up	Vec3	Entity up position
Right	Vec3	Entity right position

TransformComponent:SetEntityPosition node

Used to set entity position.

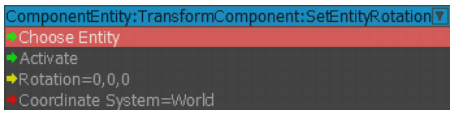


Inputs

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.

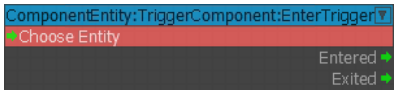


Inputs

Port		Description
Activate	Any	Activates the node
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

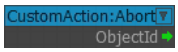
Flow Graph is deprecated and will be removed in a future version of Lumberyard. Learn about Script Canvas (p. 682), Lumberyard's new visual scripting environment.

You can use the following flow graph nodes to control custom actions that entities take.

Topics

Abort node

Used to start the abort path of a custom action.

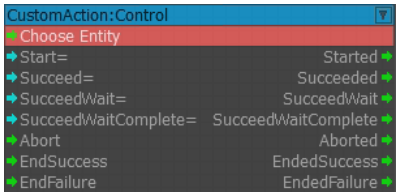


Outputs

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

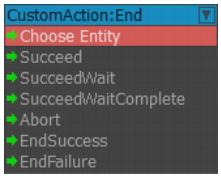
Port	Type	Description
Start	String	Entity is entering the start path
Succeed	String	Entity is entering the succeed path
SucceedWait	String	Entity is entering the succeed wait path
SucceedWaitComplete	String	Entity is entering the succeed wait complete path
Abort	Any	Entity is entering the abort path
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.

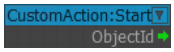


Inputs

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.

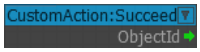


Outputs

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.

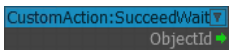


Outputs

Port	Type	Description
ObjectId	Any	Entity ID of the object on which the custom action is executing on

SucceedWait node

Used ro indicate that a custom action wait succeeded.

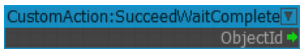


Outputs

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

Flow Graph is deprecated and will be removed in a future version of Lumberyard. Learn about Script Canvas (p. 682), Lumberyard's new visual scripting environment.

You can use the following flow graph nodes to configure various settings used for debugging purposes.

Topics

CSVDumper node

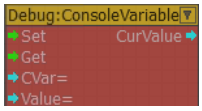
Used to store the cell values of the specified `.csv` file.

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

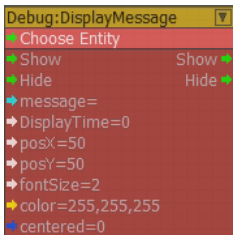
Port	Type	Description
Set	Any	Set console variable value
Get	Any	Get console variable value
CVar	String	Name of console variable
Value	String	Value of console variable to set

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 messasge
Hide	Any	Hide messasge
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 posirion
fontSize	Float	Input font size
color	Vec3	Color of the message text
centered	Boolean	Centers the text around the coordinates

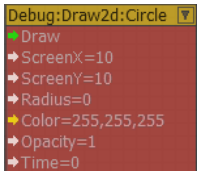
Outputs

Port	Type	Description
Show	Any	Displays the message
Hide	Any	Hides the message

Draw2d nodes

Draw2d:Circle node

Used to draw a circle.



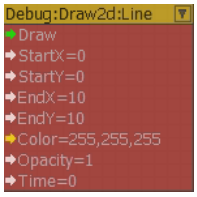
Inputs

Port	Type	Description
Draw	Any	Draws a 2D circle

Port	Type	Description
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.

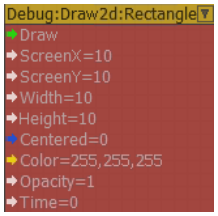


Inputs

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
Time	Float	Number of seconds the line will be visible for

Draw2d:Rectangle node

Used to draw a rectangle.

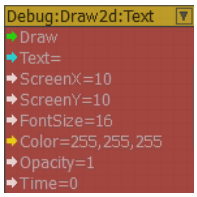


Inputs

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.



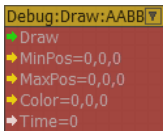
Inputs

Port	Type	Description
Draw	Any	Displays text
Text	String	Text to display
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.

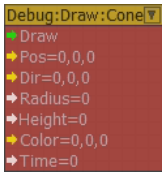


Inputs

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 a draw a cone.

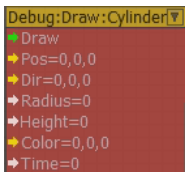


Inputs

Port	Type	Description
Draw	Any	Draws a cone
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.

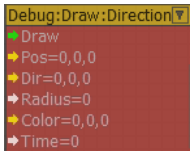


Inputs

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.

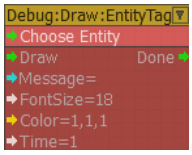


Inputs

Port	Type	Description
Draw	Any	Draws an arrow
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

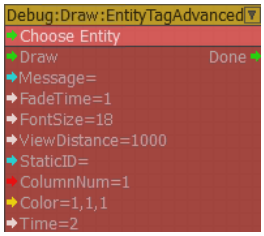
Port	Type	Description
Message	String	Text message
FontSize	Float	Text message font size
Color	Vec3	Text message color
Time	Float	Number of seconds the message will be visible for

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

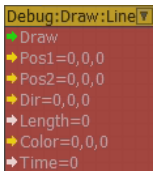
Port	Type	Description
Draw	Any	Displays a text message above an entity
Message	String	Message to be displayed
FadeTime	Float	Number of seconds for text message to fade outr
FontSize	Float	Font size of the text message
ViewDistance	Float	Distance from camera the entity must be within for message to be 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.

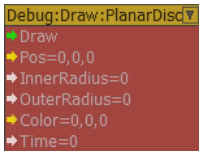


Inputs

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
Time	Float	Number of seconds the circle will be visible for

Draw:PlanarDisc node

Used to draw a disc.

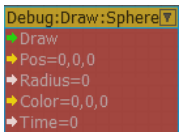


Inputs

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.

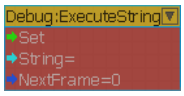


Inputs

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.

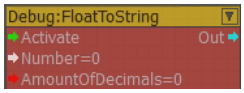


Inputs

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

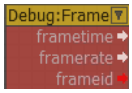
Port	Type	Description
Activate	Any	Activates the node
Number	Float	Floating point number to convert
AmountOfDecimals	Integer	Nimber of decimal places for the floating point

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.

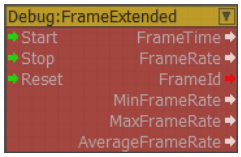


Outputs

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

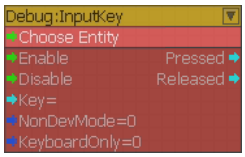
Outputs

Port	Type	Description
FrameTime	Float	Current frame time
FrameRate	Float	Current frame rate
FrameId	Integer	Frame ID
MinFrameRate	Float	Minimum frame rate

Port	Type	Description
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

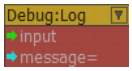
Port	Type	Description
Enable	Any	Activates the node
Disable	Any	Deactivates the node
Key	String	Key name
NonDevMode	String	Can be used in non-dev mode if set to true
Keyboard only	Boolean	Ignores non-keyword data if set to true

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.

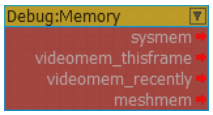


Inputs

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
sysmem	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

Flow Graph is deprecated and will be removed in a future version of Lumberyard. Learn about Script Canvas (p. 682), Lumberyard's new visual scripting environment.

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. 841)

PlayDialog node

Used to play a dialog.

Inputs

Port	Type	Description
Play	Any	Plays the dialog

Port	Type	Description
Stop	Any	Stops the dialog
Dialog	String	Name of the dialog to play
StartLine	Integer	Line to start the dialog from
AIInterrupt	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

Port	Type	Description
LastLine	Integer	Last line played when the dialog was aborted
CurLine	Integer	Current line; triggered whenever a line starts

Dynamic Response Nodes

Flow Graph is deprecated and will be removed in a future version of Lumberyard. Learn about Script Canvas (p. 682), Lumberyard's new visual scripting environment.

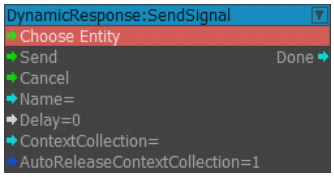
You can use the following flow graph nodes to configure settings for the Dynamic Response system.

> **Note**
> These nodes will only work with the Legacy Game Sample (CryEngine GameSDK), which is available at Lumberyard Downloads.

Topics

SendSignal node

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

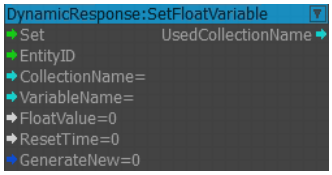
Port	Type	Description
ContextCollection	String	Name of the variable collection sent along with the signal as a context
AutoReleaseContextCollection	Boolean	Controls whether the variable collection is released after processing the signal.

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

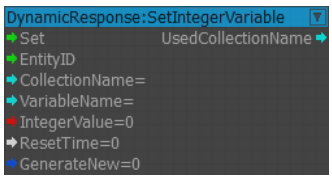
Port	Type	Description
Set	Any	Set the given value to the specified variable
EntityID	Any	The ID of the entity to fetch the collection from
CollectionName	String	The name of the collection
VariableName	String	The name of the variable to set
FloatValue	Float	The value of the variable
ResetTime	Float	The 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

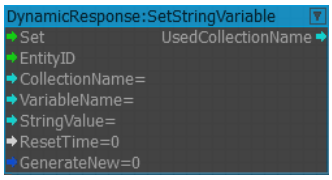
Port	Type	Description
Set	Any	Set the given value to the specified variable
EntityID	Voif	The ID of the entity to fetch the collection from
CollectionName	String	The name of the collection
VariableName	String	The name of the variable to set
FloatValue	Float	The value of the variable
ResetTime	Float	The 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

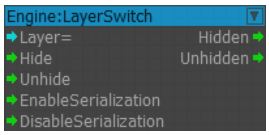
Flow Graph is deprecated and will be removed in a future version of Lumberyard. Learn about Script Canvas (p. 682), Lumberyard's new visual scripting environment.

You can use the followint flow graph nodes to configure various Lumberyard engine settings.

Topics

LayerSwitch node

Used to activate and deactivate objects in a layer, as well as streaming in data to a layer.



Inputs

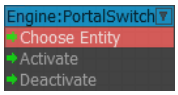
Port	Type	Description
Layer	String	Name of the layer
Hide	Any	Hides objects in the layer
Unhide	Any	Shows objects in the layer
EnableSerialization	Any	Enables objects in the layer
DisableSerialization	Any	Disables objects in the layer

Outputs

Port	Type	Description
Hidden	Any	Triggered if hidden
Unhidden	Any	Triggered if visible

PortalSwitch node

Used to switch the portal on or off.

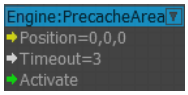


Inputs

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
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

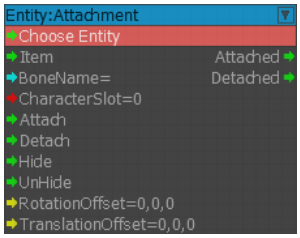
Flow Graph is deprecated and will be removed in a future version of Lumberyard. Learn about Script Canvas (p. 682), Lumberyard's new visual scripting environment.

You can use the following flow graph nodes to control entity behavior and configure related settings.

Topics

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

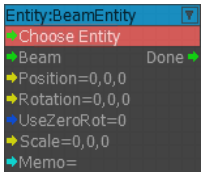
Port	Type	Description
RotationOffset	Vec3	Rotation offset
TranslationOffset	Vec3	Translation offset

Outputs

Port	Type	Description
Attached	Any	Triggers when entity is attached
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

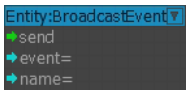
Port	Type	Description
Beam	Any	Trigger to beam the entity
Position	Vec3	Destination location to beam to
Rotation	Vec3	Rotation to apply to entity
UseZeroRot	Boolean	Applies rotation even if it is 0
Scale	Vec3	Vector scale value
Memo	String	Memo to log when position is 0

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.

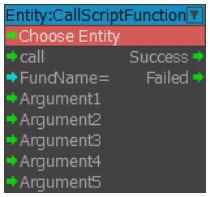


Inputs

Port	Type	Description
send	Any	Trigger to send an event
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

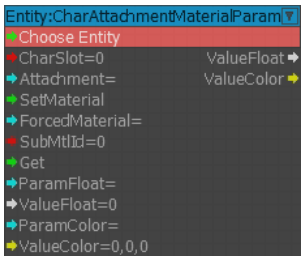
Outputs

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.

Set Material is the trigger, **ForcedMaterial** is the full file path to the material (for example: `materials/references/basecolors/grey.mtl`) and **SubMtId** is the number of the sub-material.



Inputs

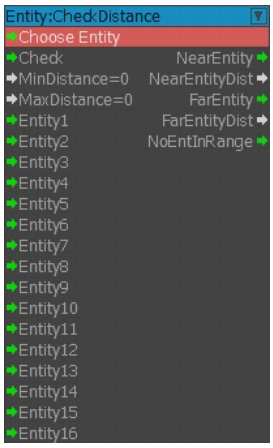
Port	Type	Description
CharSlot	Integer	Character slot within the entity
Attachment	String	Attachment
SetMaterial	Any	Sets the material
ForcedMaterial	String	Forcefully set the material
SubMtlId	Integer	Submaterial ID
Get	Any	Trigger to get current value
ParamFloat	String	Float parameter to get or be 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

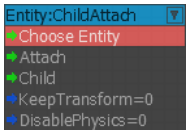
Port	Type	Description
Check	Any	Trigger to check distance
MinDistance	Float	An entity that is nearer this distance will be ignored
MaxDistance	Float	An entity that is further than this distance will be ignored
Entity1 - Entity16	Any	Entity ID values

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.



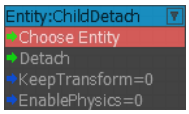
Inputs

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.

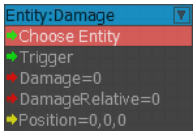


Inputs

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.

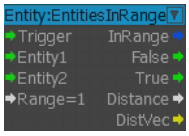


Inputs

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

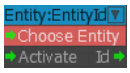
Port	Type	Description
Trigger	Any	Triggers the node
Entity1	Any	Entity 1
Entity2	Any	Entity 2
Range	Float	Distance range to check

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

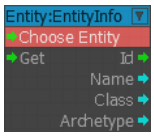
Port	Type	Description
Activate	Any	Entity ID

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

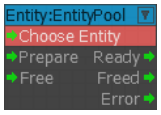
Port	Type	Description
Get	Any	Gets entity information

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

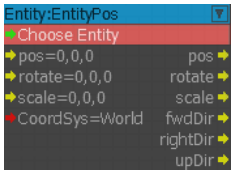
Outputs

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

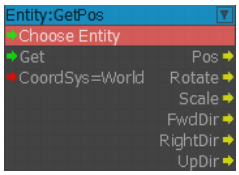
Port	Type	Description
Get	Any	Gets entity existence status
EntityId	Any	Entity ID

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

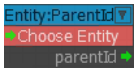
Port	Type	Description
Get	Any	Gets entity position
CoordSys	Integer	Coordinate system used

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.

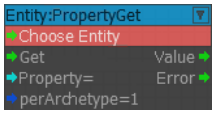


Outputs

Port	Type	Description
Parent Id	Any	Parent entity ID

PropertyGet node

Used to retrieve an entity property value.



Inputs

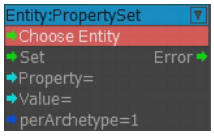
Port	Type	Description
Get	Any	Trigger to get entity property value
Property	String	Name of property to get
PerArchetype	Boolean	True if a per archetype property; false if a per instance property

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

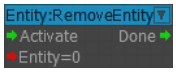
Port	Type	Description
Set	Any	Sets property value
Property	String	Name of property to set
Value	String	Property value to be set
PerArchetype	Boolean	True if a per archetype property; false if a per instance property

Outputs

Port	Type	Description
Error	Any	Any

RemoveEntity node

Used to remove an entity.



Inputs

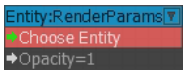
Port	Type	Description
Activate	Any	Triggers the node
Entity	Integer	Entity to remove

Outputs

Port	Type	Description
Done	Any	Triggers when entity has been removed

RenderParams node

Used to set rendering parameters.

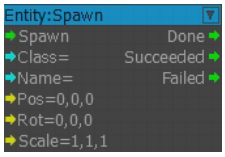


Inputs

Port	Type	Description
Opacity	Float	Sets entity transparency value

Spawn node

Used to spawn an entity with the specified properties.



Inputs

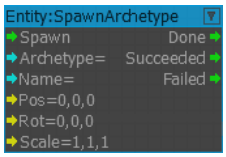
Port	Type	Description
Spawn	Any	Spawns an entity
Class	String	Entity class
Name	String	Entity class
Pos	Vec3	Entity position
Rot	Vec3	Entity rotation
Scale	Vec3	Entity scale

Outputs

Port	Type	Description
Done	Any	Triggers when entity has completed spawning
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

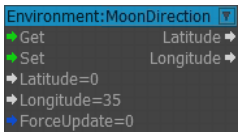
Flow Graph is deprecated and will be removed in a future version of Lumberyard. Learn about Script Canvas (p. 682), Lumberyard's new visual scripting environment.

You can use the following flow graph nodes to configure environment settings.

Topics

MoonDirection node

Used to set the moon's position in the sky.



Inputs

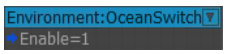
Port	Type	Description
Get	Any	Get current latitude and longitude
Set	Any	Set latitude and longitude
Latitude	Float	Latitude to be set
Longitude	Float	Longitude to be set
ForceUpdate	Boolean	Force immediate update of the sky

Outputs

Port	Type	Description
Latitude	Float	Output current latitude
Longitude	Float	Output current longitude

OceanSwitch node

Used to enable ocean rendering.

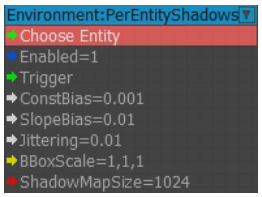


Inputs

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

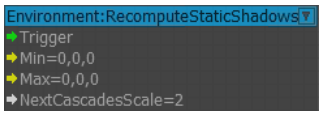
Used to get and output rain properties.

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.

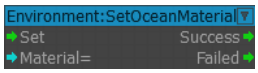


Input

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

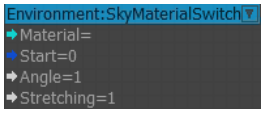
Port	Type	Description
Set	Any	Set material on for the ocean
Material	String	Material to be set for the ocean

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.

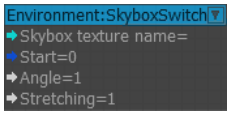


Inputs

Port	Type	Description
Material	String	Material to use for the sky
Start	Boolean	Start material switch
Angle	Float	Starting angle
Stretching	Float	If stretching is performed or not

SkyboxSwitch node

Used to enable asynchronous sky box switching.

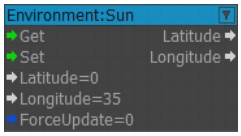


Inputs

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
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

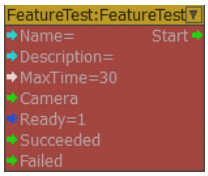
Flow Graph is deprecated and will be removed in a future version of Lumberyard. Learn about Script Canvas (p. 682), Lumberyard's new visual scripting environment.

You can use the following flow graph nodes to configure feature test settings.

Topics

FeatureTest node

Used to control automated feature tests.



Inputs

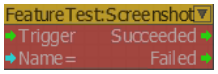
Port	Type	Description
Name	String	Name of the feature test
Description	String	Description of the feature test
MaxTime	Float	How long in game time the test is allowed to run before it fails
Camera	Any	(Optional) Camera entity used for the test
Ready	Boolean	Indicates whether all dependencies have been 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

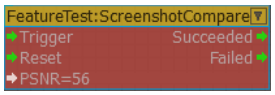
Port	Type	Description
Name	String	Name of the screenshot

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

Force Feedback Nodes

> Flow Graph is deprecated and will be removed in a future version of Lumberyard. Learn about Script Canvas (p. 682), Lumberyard's new visual scripting environment.

Lumberyard uses the force feedback system to manage haptic feedback for motion controllers. You can use the force feedback nodes to configure haptic feedback settings.

Topics

- Game:ForceFeedbackSetDeviceIndex (p. 870)

Game:ForceFeedbackSetDeviceIndex

Sets the device ID for force feedback effects.

Node Inputs

DeviceIndex

Enables or disables the force feedback functionality.

Valid values: `0` = Enable | `-1` = Disable

Game:ForceFeedback

Controls the force feedback effect with an effect that is specified in the `ForceFeedbackEffects.xml` file (located in the `\libs\GameForceFeedback` directory).

The following is an example `ForceFeedbackEffects.xml` file with `right_shoot` and `left_shoot` effects added:

```
<ForceFeedback>
    <Patterns>
        <!-- Pattern definition accepts from 1 to 16 samples -->
        <Pattern name="100" samples="1" />
        <Pattern name="75" samples="0.75" />
        <Pattern name="50" samples="0.5" />
        <Pattern name="30" samples="0.3" />
        <Pattern name="20" samples="0.2" />
        <Pattern name="12" samples="0.12" />
    </Patterns>
    <Envelopes>
        <!-- Envelope definition accepts from 1 to 8 samples -->
        <Envelope name="linearFalloff" samples="1,0" />
        <Envelope name="cosineFalloff" samples="1,0.98,0.92,0.83,0.69,0.54,0.25,0" />
        <Envelope name="constant" samples="1" />
    </Envelopes>
    <Effects>
        <Effect name="small_rumble" time="1.0" >
            <MotorAB frequency="1" pattern="100" envelope="linearFalloff" />
        </Effect>
        <Effect name="small_pulse" time="0.05" >
            <MotorA frequency="1" pattern="20" envelope="cosineFalloff" />
            <MotorB frequency="1" pattern="20" envelope="cosineFalloff" />
        </Effect>
        <Effect name="medium_pulse" time="0.12" >
            <MotorA frequency="1" pattern="20" envelope="cosineFalloff" />
            <MotorB frequency="1" pattern="50" envelope="cosineFalloff" />
        </Effect>
        <Effect name="heavy_recoil" time="0.2" >
            <MotorA frequency="1" pattern="50" envelope="cosineFalloff" />
            <MotorB frequency="1" pattern="50" envelope="cosineFalloff" />
        </Effect>
        <Effect name="big_recoil" time="0.5" >
            <MotorA frequency="1" pattern="75" envelope="linearFalloff" />
            <MotorB frequency="1" pattern="75" envelope="linearFalloff" />
```

```
        </Effect>
        <Effect name="weak" time="0.5" >
            <MotorAB frequency="1" pattern="12" envelope="linearFalloff" />
        </Effect>
        <Effect name="medium" time="0.5" >
            <MotorB frequency="1" pattern="50" envelope="linearFalloff" />
        </Effect>
        <Effect name="strong" time="0.5" >
            <MotorAB frequency="1" pattern="100" envelope="linearFalloff" />
        </Effect>
    <Effect name="right_shoot" time="0.1" >
            <MotorA frequency="1" pattern="50" envelope="cosineFalloff" />
        </Effect>
    <Effect name="left_shoot" time="0.5" >
            <MotorB frequency="1" pattern="50" envelope="cosineFalloff" />
        </Effect>
    </Effects>
</ForceFeedback>
```

Node Inputs

Effect Name

Select the effect to use from the **Effect Name** drop-down list.

Play

Plays the force feedback effect.

Intensity

Applies the intensity factor to the effect being played.

Default value: `1`

Delay

Time delay to start the effect.

Default value: `0`

Stop

Stops the force feedback effect.

StopAll

Stops all force feedback effects.

Input Device Type

Specifies the input device to which to send force feedback.

Default value: `Gamepad`

Valid values: `Gamepad` | `Joystick` | `Keyboard` | `MotionController` | `MotionSensor` | `Mouse` | `TouchScreen`

> **Note**
> MotionController uses Left, LowPass, and MotorA terminology for the Left Controller. It uses Right, HighPass, and MotorB terminology for the Right Controller.

Game:ForceFeedbackTriggerTweaker

Controls the force feedback effect on the left and right triggers.

Node Inputs

LeftTouchToActivate

The left trigger's gain is modulated by how much the trigger is pressed.

Default value: `1`

LeftGain

Gain sent to the left trigger's motor.

Default value: `1`

LeftEnvelope

Envelope sent to the left trigger's motor.

Default value: `1`

Valid values: `0 – 2000` (in multiples of 4)

RightTouchToActivate

The right trigger's gain is modulated by how much the trigger is pressed.

Default value: `1`

RightGain

Gain sent to the right trigger's motor.

Default value: `1`

RightEnvelope

Envelope sent to the right trigger's motor.

Default value: `1`

Valid values: `0 – 2000` (in multiples of 4)

Activate

Activates the effect on the left and right triggers.

Deactivate

Deactivates the effect on the left and right triggers.

Input Device Type

Specifies the input device to which to send force feedback.

Default value: `Gamepad`

Valid values: `Gamepad` | `Joystick` | `Keyboard` | `MotionController` | `MotionSensor` | `Mouse` | `TouchScreen`

> **Note**
> MotionController uses Left, LowPass, and MotorA terminology for the Left Controller. It uses Right, HighPass, and MotorB terminology for the Right Controller.

Game:ForceFeedbackTweaker

Controls individual high or low frequency force feedback effects.

Node Inputs

LowPass

Applies the multiplier to the low frequency effect being played.

Default value: `1`

HighPass

Applies the multiplier to the high frequency effect being played.

Default value: `1`

Activate

Activates the high or low frequency force feedback effect.

Deactivate

Deactivates the high or low frequency force feedback effect.

Input Device Type

Specifies the input device to which to send force feedback.

Default value: `Gamepad`

Valid values: `Gamepad` | `Joystick` | `Keyboard` | `MotionController` | `MotionSensor` | `Mouse` | `TouchScreen`

> **Note**
> MotionController uses Left, LowPass, and MotorA terminology for the Left Controller. It uses Right, HighPass, and MotorB terminology for the Right Controller.

Game Nodes

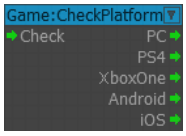
Flow Graph is deprecated and will be removed in a future version of Lumberyard. Learn about Script Canvas (p. 682), Lumberyard's new visual scripting environment.

You can use the following flow graph nodes to check and to configure various game settings.

Topics

CheckPlatform node

Used to change game events depending on what platform you are running on.



Inputs

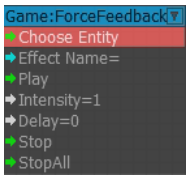
Port	Type	Description
Check	Any	Triggers a check of the current platform

Outputs

Port	Type	Description
PC	Any	Triggers if the platform is PC
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.

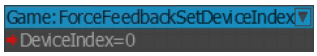


Inputs

Port	Type	Description
Effect Name	String	Name of the force feedback effect
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.

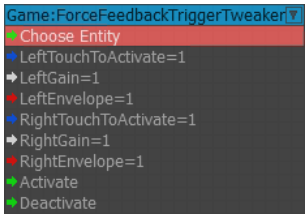


Inputs

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.

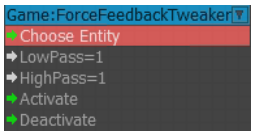


Inputs

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.

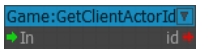


Inputs

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

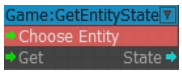
Port	Type	Description
In	Any	Gets client actor ID

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

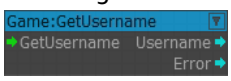
Port	Type	Description
Get	Any	Gets the game rules
Mapname	String	Map name

Outputs

Port	Type	Description
GameRules	String	Outputs the game rules

GetUsername node

Used to get the user name.



Inputs

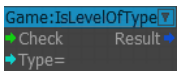
Port	Type	Description
GetUsername	Any	Gets the user name

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

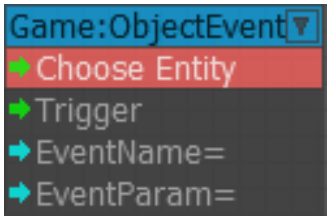
Port	Type	Description
Check	Any	Checks if a level is of a given type
Type	String	Level type to check against

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.

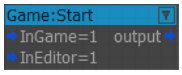


Inputs

Port	Type	Description
Trigger	Any	Triggers the node
EventName	String	Game object event name
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

Flow Graph is deprecated and will be removed in a future version of Lumberyard. Learn about Script Canvas (p. 682), Lumberyard's new visual scripting environment.

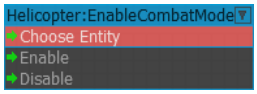
You can use the follosing flow graph nodes to configure flying vehicle and flight AI-related settings.

> **Note**
> These nodes will only work with the Legacy Game Sample (CryEngine GameSDK), which is available at Lumberyard Downloads.

Topics

EnableCombatMode node

Used to alter the path the flight AI should follow so as to find the best position from which to engage the target.

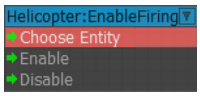


Inputs

Port	Type	Description
Enable	Any	Enables combat mode
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.

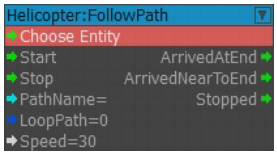


Inputs

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

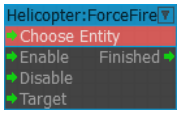
Port	Type	Description
Start	Any	Start following the path
Stop	Any	Stop following the path
PathName	String	Name of the path to follow
LoopPath	Boolean	How many times to loop around the path
Speed	Float	Speed of the flight AI

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

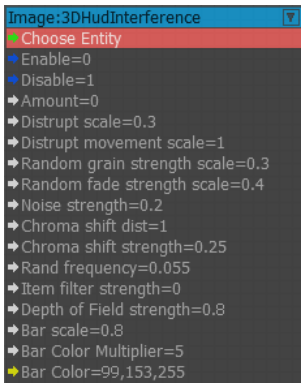
Flow Graph is deprecated and will be removed in a future version of Lumberyard. Learn about Script Canvas (p. 682), Lumberyard's new visual scripting environment.

You can use the following flow graph nodes to configure various visual effects and image settings.

Topics

3DHudInterference node

Used to add distortion effects to the HUD.



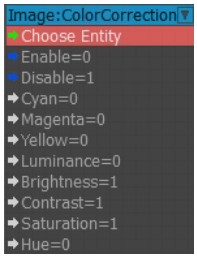
Inputs

Port	Type	Description
Enable	Boolean	Enables the effect
Disable	Boolean	Disables the effect
Amount	Float	Interference amount

Port	Type	Description
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	Stength of the depth of field
Bar scale	Float	Bar scale
Bar color multiplier	Float	Bar color multiplier
Bar color	Vec3	Bar color

ColorCorrection node

Used to control basic image settings such as saturation, contrast, brightness, and color.



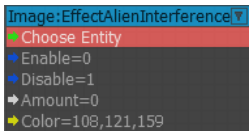
Inputs

Port	Type	Description
Enable	Boolean	Enables color correction
Disable	Boolean	Disables color correction
Cyan	Any	Cyan increase or decrease
Magenta	Any	Magneta increase or decrease
Yellow	Any	Yellow increase or decrease
Luminance	Any	Luminance increase or decrease

Port	Type	Description
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.

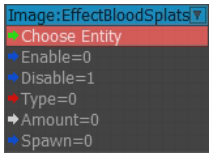


Inputs

Port	Type	Description
Enable	Boolean	Enables the effect
Disable	Boolean	Disables the effect
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.

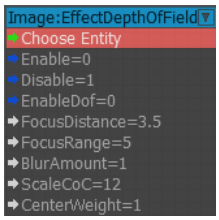


Inputs

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.

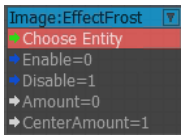


Inputs

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
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.

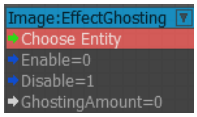


Inputs

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.

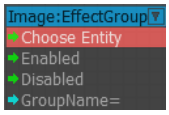


Inputs

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.

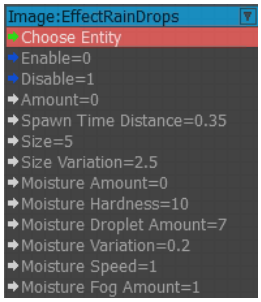


Inputs

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.



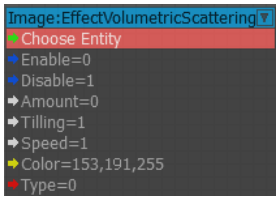
Inputs

Port	Type	Description
Enable	Boolean	Enables the node

Port	Type	Description
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.

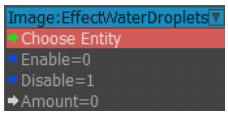


Inputs

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.

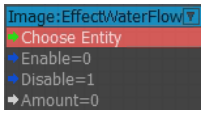


Inputs

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.

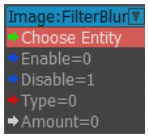


Inputs

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.

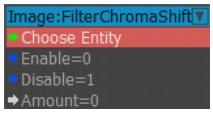


Inputs

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.

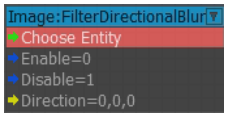


Inputs

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.

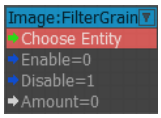


Inputs

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.

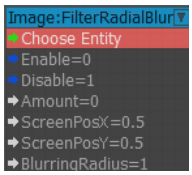


Inputs

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.

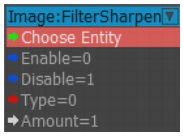


Inputs

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
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.

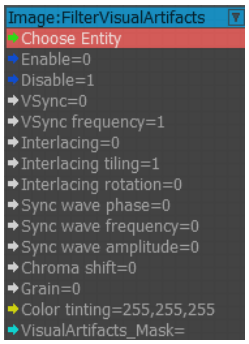


Inputs

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.

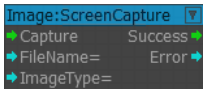


Inputs

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
Interlacing rotation	Float	Interlacing rotation
Sync wave phase	Float	Sync wave phase
Sync wave frequence	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

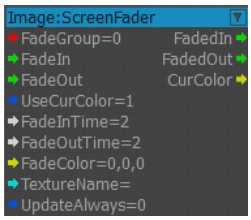
Port	Type	Description
ImageType	Any	File type to use

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

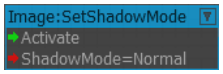
Outputs

Port	Type	Description
FadedIn	Any	Triggered when the screen completed faded in
FadedOut	Any	Triggered when the screen completed faded out

Port	Type	Description
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

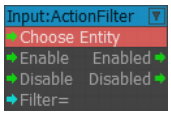
Flow Graph is deprecated and will be removed in a future version of Lumberyard. Learn about Script Canvas (p. 682), Lumberyard's new visual scripting environment.

You can use the following flow graph nodes to capture input events and configure input settings.

Topics

ActionFilter node

Used to catch key inputs. Should only be used for debugging purposes however.



Inputs

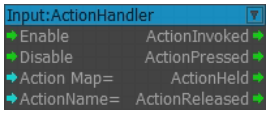
Port	Type	Description
Enable	Any	Enables the node
Disable	Any	Disables the node
Filter	String	Name of the action filter

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

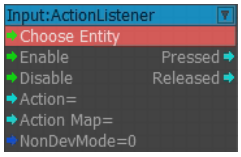
Port	Type	Description
Enable	Any	Enables listening to the action map
Disable	Any	Disables listening to the action map
Action Map	String	Name of the action map
ActionName	String	Name of the action to listen for

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

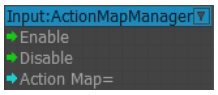
Port	Type	Description
Enable	Any	Enables the node
Disable	Any	Disables the node
Action	String	Action to trigger
Action Map	String	Action map to use
NonDevMode	Boolean	When set to true, can be used in non dev mode as well

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

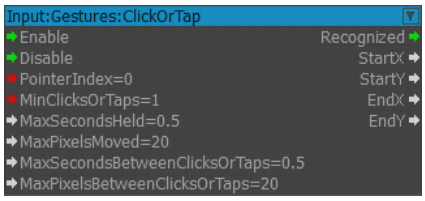
Port	Type	Description
Enable	Any	Enables the node
Disable	Any	Disables the node
Action Map	String	Name of the action map to use

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

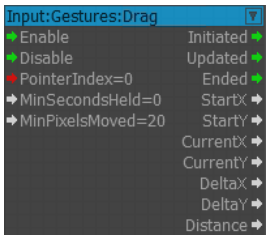
Port	Type	Description
Enable	Any	Enables the node
Disable	Any	Disables the node
PointerIndex	Integer	Pointer (button or finger) index to track
MinClicksOrTaps	Integer	Minimum number of clicks or taps required for the gesture to be recognized
MaxSecondsHeld	Float	Maximum number of seconds allowed while 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
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

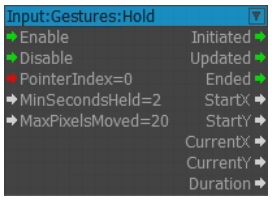
Port	Type	Description
Enable	Any	Enables the node
Disable	Any	Disables the node
PointerIndex	Integer	Pointer (button or finger) index to track
MinSecondsHeld	Float	Mimimum number of seconds after the initial press before a drag is recognized
MinPixelsMoved	Float	Mimimum distance in pixels before a drag 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
Ended	Any	Activated when a continuous drag gesture has ended
StartX	Float	X-axis screen position of the drag start
StartY	Float	X-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

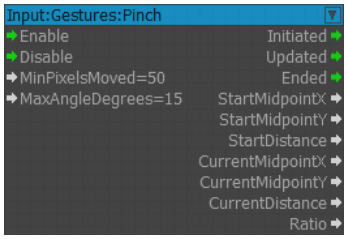
Port	Type	Description
Enable	Any	Enables the node
Disable	Any	Disables the node
PointerIndex	Integer	The button or finger index to track
MinSecondsHeld	Float	Mimimum number of seconds before a hold is recognized
MaxPixelsMoved	Float	Mimimum distance in pixels before a hold is recognized

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
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

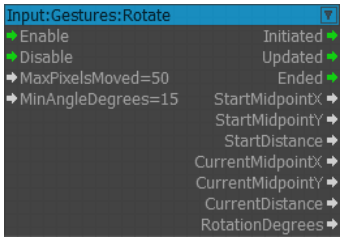
Port	Type	Description
Enable	Any	Enables the node
Disable	Any	Disables the node
MinPixelsMoved	Float	Mimimum distance in pixels before a pinch is recognized
MaxAngleDegrees	Float	Maximum angle in degrees that pinch can deviate before it is recognized

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
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

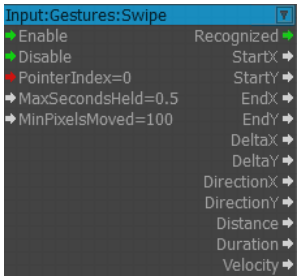
Port	Type	Description
Disable	Any	Disables the node
MaxPixelsMoved	Float	Maximum distance in pixels before a rotation is recognized
MinAngleDegrees	Float	Minimum angle in degrees before a rotation is recognized

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
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

Port	Type	Description
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	Mimimum 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
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

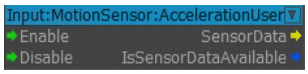
Port	Type	Description
Enable	Any	Enables the node
Disable	Any	Disables the node

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

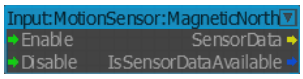
Port	Type	Description
Enable	Any	Enables the node
Disable	Any	Disables the node

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

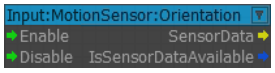
Port	Type	Description
Enable	Any	Enables the node
Disable	Any	Disables the node

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

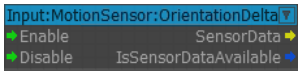
Port	Type	Description
Enable	Any	Enables the node
Disable	Any	Disables the node

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

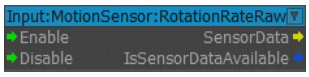
Port	Type	Description
Enable	Any	Enables the node
Disable	Any	Disables the node

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

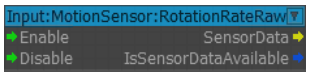
Port	Type	Description
Enable	Any	Enables the node
Disable	Any	Disbles the node

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

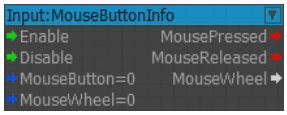
Outputs

Port	Type	Description
SensorData	Vec3	Outputs an unbiased rotation rate in degrees per second

Port	Type	Description
IsSensorDataAvailable	Boolean	Outputs true or false when the node is activated

MouseButtonInfo node

Used to output mouse button state information.



Inputs

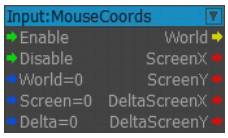
Port	Type	Description
Enable	Any	Enables the node
Disable	Any	Disables the node
MouseButton	Boolean	Mouse button state information
MouseWheel	Boolean	Mouse wheel state information

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

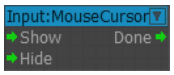
Port	Type	Description
Delta	Boolean	Shows the number of screen pixels the mouse cursor has moved

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

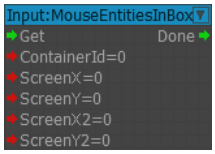
Port	Type	Description
Show	Any	Shows the mouse cursor
Hide	Any	Hides the mouse cursor

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

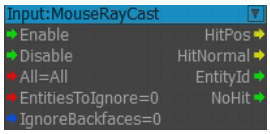
Port	Type	Description
ContainerId	Integer	ID of the container that stores the entities
ScreenX	Integer	X-axis screen position of the mouse cursor
ScreenY	Integer	Y-axis screen position of the mouse cursor
ScreenX2	Integer	X-axis screen position 2 of the mouse cursor
ScreenY2	Integer	Y-axis screen position 2 of the mouse cursor

Outputs

Port	Type	Description
Done	Any	Triggers when completed

MouseRayCast node

Used to output the mouse raycast information.



Inputs

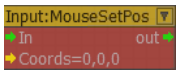
Port	Type	Description
Enable	Any	Enables the node
Disable	Any	Disables the node
All	Integer	Raycast filter type
EntitiesToIgnore	Integer	Entities to ignore during raycast
IgnoreBackFaces	Boolean	Ignore backfaces of geometry during raycast

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

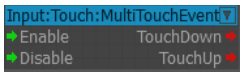
Port	Type	Description
In	Any	Activates the node
Coords	Vec3	Coordinates to set the mouse at

Outputs

Port	Type	Description
Out	Any	Triggers when the new mouse position is set

Touch:MultiTouchEvent node

Used to output finger touch location.



Inputs

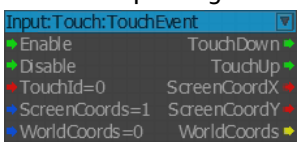
Port	Type	Description
Enable	Any	Enables the node
Disable	Any	Disables the node

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

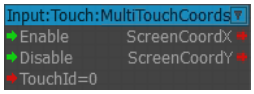
Port	Type	Description
Enable	Any	Enables the node
Disable	Any	Disables the node
TouchId	Integer	Touch (finger) ID for which events will be sent from
ScreenCoords	Boolean	Output screen coordinates
WorldCoords	Boolean	Output world coordinates

Outputs

Port	Type	Description
TouchDown	Any	Finger (touch) ID that was pressed
TouchUp	Any	Finger (touch) ID that was released
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

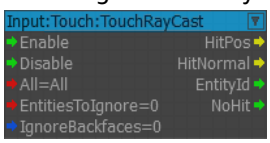
Port	Type	Description
Enable	Any	Enables the node
Disable	Any	Disables the node
TouchId	Integer	Finger (touch) ID for which the coordinates will obtained

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

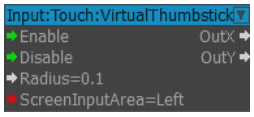
Port	Type	Description
Enable	Any	Enables the node
Disable	Any	Disables the node
All	Integer	Raycast filter type
EntitiesToIgnore	Integer	Entities to ignore during raycast
IgnoreBackFaces	Boolean	Ignore backfaces of geometry during raycast

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

Port	Type	Description
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

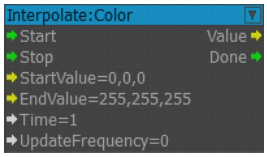
Flow Graph is deprecated and will be removed in a future version of Lumberyard. Learn about Script Canvas (p. 682), Lumberyard's new visual scripting environment.

You can use these flow graph nodes to configure interpolate-related settings.

Topics

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

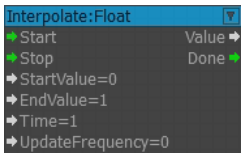
Port	Type	Description
EndValue	Vec3	Ending value for color
Time	Float	Interpolation duration in seconds
UpdateFrequency	Float	Interpolation update frequency in seconds. 0 = every frame

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

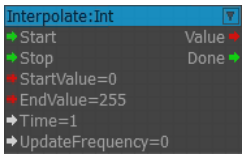
Port	Type	Description
Start	Any	Starts interpolation
Stop	Any	Stops interpolation
StartValue	Float	Starting value for floating point
EndValue	Float	Ending value for floating point
Time	Float	Interpolation duration in seconds
UpdateFrequency	Float	Interpolation update frequency in seconds. 0 = every frame

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

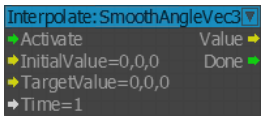
Port	Type	Description
Start	Any	Starts interpolation
Stop	Any	Stops interpolation
StartValue	Integer	Starting value for integer
EndValue	Integer	Ending value for integer
Time	Float	Interpolation duration in seconds
UpdateFrequency	Float	Interpolation update frequency in seconds. 0 = every frame

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

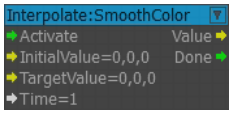
Outputs

Port	Type	Description
Value	Vec3	Current value

Port	Type	Description
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

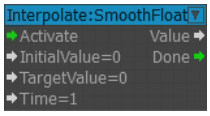
Port	Type	Description
Activate	Any	Triggers the node
InitialValue	Vec3	Initial interpolation value for color
TargetValue	Vec3	Target interpolation value for color
Time	Float	Interpolation duration in seconds

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

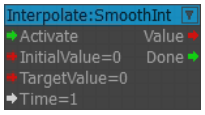
Port	Type	Description
Activate	Any	Triggers the node
InitialValue	Float	Initial interpolation value for floating point
TargetValue	Float	Target interpolation value for floating point
Time	Float	Interpolation duration in seconds

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

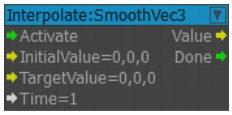
Port	Type	Description
Activate	Any	Triggers the node
InitialValue	Integer	Initial interpolation value for integer
TargetValue	Integer	Target interpolation value for integer
Time	Float	Interpolation duration in seconds

Outputs

Port	Type	Description
Value	Integer	Current value
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

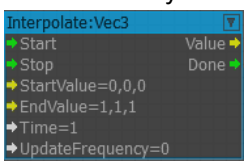
Port	Type	Description
Time	Float	Interpolation duration in seconds

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
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

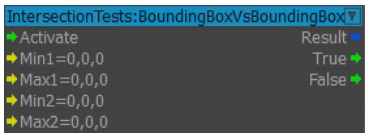
Flow Graph is deprecated and will be removed in a future version of Lumberyard. Learn about Script Canvas (p. 682), Lumberyard's new visual scripting environment.

You can use the following flow graph nodes to configure intersection tests.

Topics

BoundingBoxVsBoundingBox node

Used to test two bounding boxes to see if they intersect.



Inputs

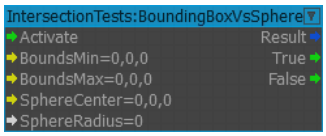
Port	Type	Description
Activate	Any	Triggers the node
Min1	Vec3	Minimum point for the first bounding box
Max1	Vec3	Maximum point for the first bounding box
Min2	Vec3	Minimum point for the second bounding box
Max2	Vec3	Maximum point for the second bounding box

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

Port	Type	Description
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

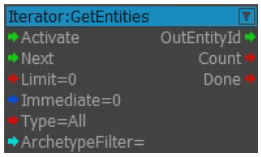
Flow Graph is deprecated and will be removed in a future version of Lumberyard. Learn about Script Canvas (p. 682), Lumberyard's new visual scripting environment.

You can use the following flow graph nodes to configure iterator-related settings.

Topics

GetEntities node

Used to find and return all entities in the world.



Inputs

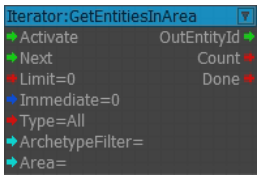
Port	Type	Description
Activate	Any	Triggers the node
Next	Any	Gets the next entity found
Limit	Integer	Limits how many entities are returned
Immediate	Boolean	Iterates immediately through the results
Type	Integer	Type of entity to iterate
ArchetypeFilter	Any	Returns archetype entities

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

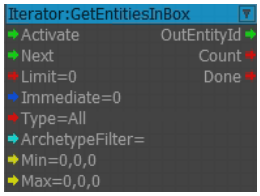
Port	Type	Description
Activate	Any	Triggers the node
Next	Any	Gets the next entity found
Limit	Integer	Limits how many entities are returned
Immediate	Boolean	Iterates immediately through the results
Type	Integer	Type of entity to iterate
ArchetypeFilter	String	Returns archetype entities
Area	String	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

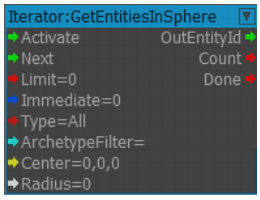
Port	Type	Description
Activate	Any	Triggers the node
Next	Any	Gets the next entity found
Limit	Integer	Limits how many entities are returned
Immediate	Any	Iterates immediately through the results
Type	Integer	Type of entity to iterate
ArchetypeFilter	String	Returns archetype entities
Min	Vec3	Minimum 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

Port	Type	Description
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

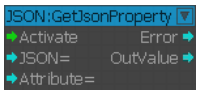
Flow Graph is deprecated and will be removed in a future version of Lumberyard. Learn about Script Canvas (p. 682), Lumberyard's new visual scripting environment.

You can use these flow graph nodes to configure JSON settings.

Topics

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

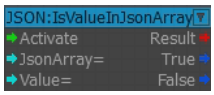
Port	Type	Description
Attribute	String	The attribute to get the value of

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

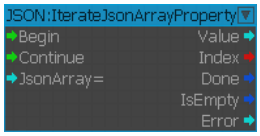
Port	Type	Description
Activate	Any	Triggers the node
JsonArray	String	The JSON array to search on
Value	String	The JSON value to search for

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

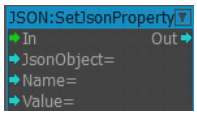
Port	Type	Description
Continue	Any	Continues iterating over the supplied JSON array
JsonArray	String	The JSON array to iterated over

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

Flow Graph is deprecated and will be removed in a future version of Lumberyard. Learn about Script Canvas (p. 682), Lumberyard's new visual scripting environment.

You can use these flow graph nodes to configure Kinect settings.

Topics

Alignment node

Used to get the default Kinect skeleton joint lengths when a new closest tracked skeleton is detected.

Inputs

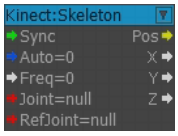
Port	Type	Description
Enable	Any	Enables the skeleton alignment watcher
Disable	Any	Disables the skeleton alignment watcher
ForceAlign	Any	Forces the beginning of the new skeleton alignment phase
AlignTime	Float	The time spent each time the skeleton alignment is started

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.

Port	Type	Description
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

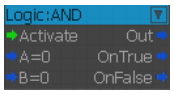
Flow Graph is deprecated and will be removed in a future version of Lumberyard. Learn about Script Canvas (p. 682), Lumberyard's new visual scripting environment.

You can use the following flow graph nodes to define logic operations.

Topics

AND node

Used to perform a logical AND operation on the input ports. Output is true if both inputs are true.



Inputs

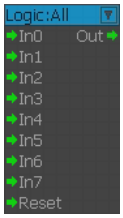
Port	Type	Description
Activate	Any	Triggers the node
A	Boolean	First input
B	Boolean	Second input

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 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

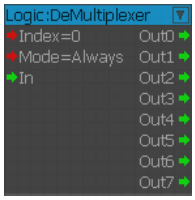
Outputs

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

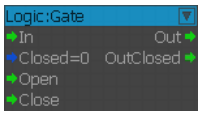
Port	Type	Description
Index	Integer	Determines which output receives the input (In)
Mode	Integer	Determines when the outputs are activated
In	Any	Input value

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

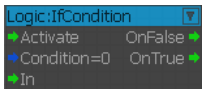
Port	Type	Description
In	Any	Input value
Closed	Boolean	If true, blocks the input from passing to the output
Open	Any	Sets Closed to false
Close	Any	Sets Closed to true

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

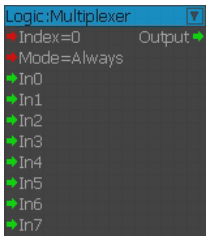
Outputs

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

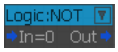
Port	Type	Description
Index	Integer	Determines which input is passed to the output
Mode	Integer	Determineswhich inputs activate the output
In0 - In7	Any	Input values

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

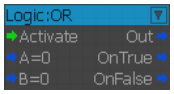
Port	Type	Description
In	Boolean	Input value

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

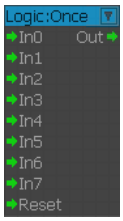
Port	Type	Description
In	Boolean	Input value

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

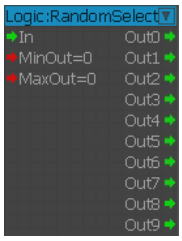
Port	Type	Description
Reset	Any	Resets the input and allows new activation to occur

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

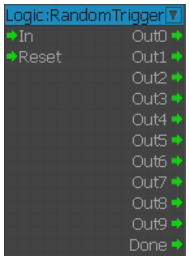
Port	Type	Description
In	Any	Input value
MinOut	Integer	Minimum number of outputs to trigger
MaxOut	Integer	Maximum number of outputs to trigger

Outputs

Port	Type	Description
Out0 - Out9	Any	Receives active input values

RandomTrigger node

Used to trigger one of the outputs in random order.



Inputs

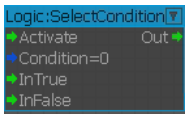
Port	Type	Description
In	Any	Input value
Reset	Any	Resets the activations to 0

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

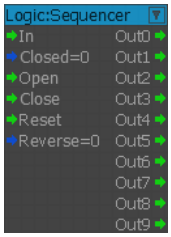
Port	Type	Description
Activate	Any	Triggers the node
Condition	Boolean	Condition value
InTrue	Any	Value sent to Out when Condition is true
InFalse	Any	Value sent to Out when Condition is false

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

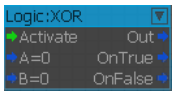
Port	Type	Description
In	Any	Input value
Closed	Boolean	If true, blocks all inputs
Open	Any	Sets Closed to false
Close	Any	Sets Closed to true
Reset	Any	Forces active output to Out0
Reverse	Boolean	If true, the order of ouput activation is reversed

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

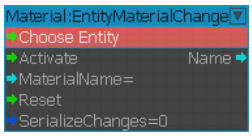
Flow Graph is deprecated and will be removed in a future version of Lumberyard. Learn about Script Canvas (p. 682), Lumberyard's new visual scripting environment.

You can use the following flow graph nodes to define material settings.

Topics

EntityMaterialChange node

Used to apply the specified material to an entity.



Inputs

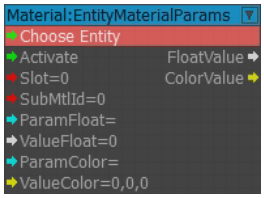
Port	Type	Description
Activate	Any	Activates the node
MaterialName	String	Name of material to apply
Reset	Any	Reset to the original material
SerializeChanges	Boolean	Serialize the change

Outputs

Port	Type	Description
Name	String	Outputs the name of the material

EntityMaterialParams node

Used to get the entity's material parameters.



Inputs

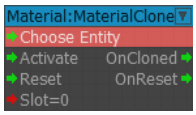
Port	Type	Description
Activate	Any	Activates the node
Slot	Integer	Material slot
SubMtlId	Integer	Submaterial ID
ParamFloat	String	Float parameter to be set
ValueFloat	Float	Sets float parameter value
ParamColor	String	Color parameter to be set
ValueColor	Vec3	Color value to be set

Outputs

Port	Type	Description
FloatValue	Float	Current float value
ColorValue	Vec3	Current color value

MaterialClone node

Used to clone an entity's material or reset it back to the original.



Inputs

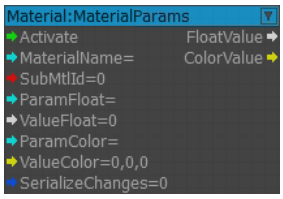
Port	Type	Description
Activate	Any	Activates the node
Reset	Any	Resets to the original material
Slot	Integer	Material slot

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

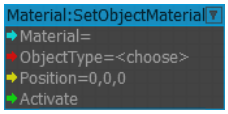
Port	Type	Description
Activate	Any	Activates the node
MaterialName	String	Material name
SubMtlId	Integer	Submaterial name
ParamFloat	String	Float parameter to be set
ValueFloat	Float	Value of the float parameter
ParamColor	String	Color parametetr to be set
ValueColor	Vec3	Value of the color parameter
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

Flow Graph is deprecated and will be removed in a future version of Lumberyard. Learn about Script Canvas (p. 682), Lumberyard's new visual scripting environment.

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

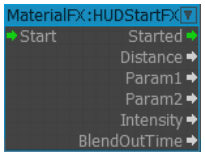
The MaterialFX end node for an HUD.

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

Port	Type	Description
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

Flow Graph is deprecated and will be removed in a future version of Lumberyard. Learn about Script Canvas (p. 682), Lumberyard's new visual scripting environment.

You can use these flow graph nodes to define math operations.

Topics

Abs node

Used to calculate the absolute value of the input.

Inputs

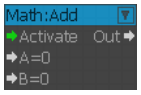
Port	Type	Description
Activate	Any	Triggers the node
A	Float	Input

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

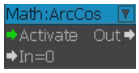
Port	Type	Description
Angles	Vec3	Input angle

Outputs

Port	Type	Description
Dir	Vec3	Direction unit vector
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

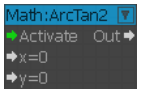
Port	Type	Description
Activate	Any	Triggers the node
In	Float	Input angle

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

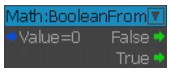
Port	Type	Description
Activate	Any	Triggers the node
X	Float	X input value
Y	Float	Y input value

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

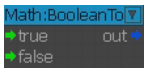
Port	Type	Description
Value	Boolean	Boolean input (0 or 1)

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

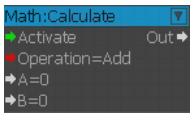
Port	Type	Description
True	Float	Will output true if event is received on this input
False	Float	Will output false if event is received on this input

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	Ingteger	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

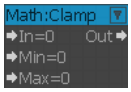
Port	Type	Description
In	Float	Input

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

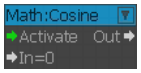
Port	Type	Description
Max	Float	Maximum clamp value

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
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

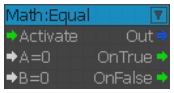
Port	Type	Description
Activate	Any	Triggers the node
A	Float	Dividend input
B	Float	Divisor input

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

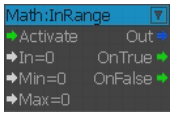
Port	Type	Description
In	Float	Input

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

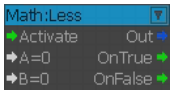
Port	Type	Description
Activate	Any	Triggers the node
In	Float	Input
Min	Float	Minimum value of the range
Max	Float	maximum value of the range

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

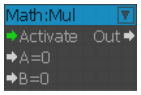
Port	Type	Description
Activate	Any	Triggers the node
A	Float	First operand
B	Float	Second operand

Outputs

Port	Type	Description
Result	Float	Modulus of the two inputs

Mul node

Used to multiply the two inputs.



Inputs

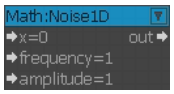
Port	Type	Description
Activate	Any	Triggers the node
A	Float	First operand
B	Float	Second operand

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

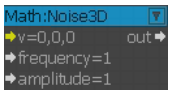
Port	Type	Description
X	Float	Scalar Input value to sample noise at
Frequency	Float	Frequency
Amplitude	Float	Amplitude

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

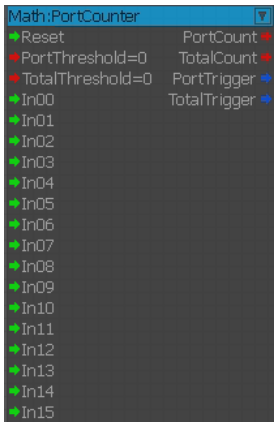
Port	Type	Description
V	Vec3	Vector input value to sample noise at
Frequency	Float	Frequency
Amplitude	Float	Amplitude

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

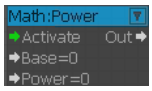
Port	Type	Description
Reset	Any	Resets PortCount and TotalCount
PortThreshold	Integer	PortCount threshold value
TotalThreshold	Integer	TotalCount threshold value
In00 - In15	Any	Inputs

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

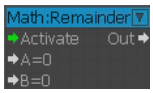
Port	Type	Description
Activate	Any	Triggers the node
A	Float	Input

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

Port	Type	Description
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
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

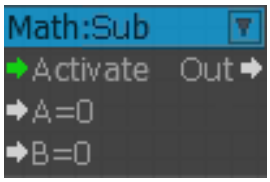
Port	Type	Description
Activate	Any	Triggers the node
A	Float	Input

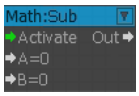
Outputs

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
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

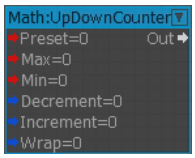
Port	Type	Description
Activate	Any	Triggers the node
In	Float	Input angle in degrees

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

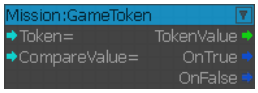
Flow Graph is deprecated and will be removed in a future version of Lumberyard. Learn about Script Canvas (p. 682), Lumberyard's new visual scripting environment.

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

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

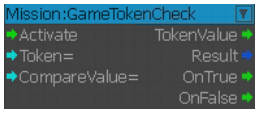
Port	Type	Description
Token	String	Game token to compare. Any change in this value will trigger the TokenValue output.
CompareValue	String	Value to compare the token value against

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
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

Port	Type	Description
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
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

Port	Type	Description
Token	String	Game token to get

Inputs

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
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

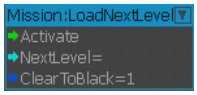
Used to store the values of all game tokens in a level.

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

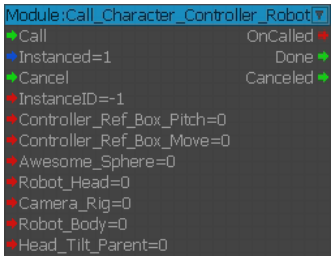
Flow Graph is deprecated and will be removed in a future version of Lumberyard. Learn about Script Canvas (p. 682), Lumberyard's new visual scripting environment.

Module nodes are project-specific, user-created nodes. The nodes listed here are used in the Samples projects, which are located at `lumberyard_version\dev\SamplesProject\Levels\Samples`.

Topics

Call_Character_Controller_Robot node

Used to call a character's controller robot.



Inputs

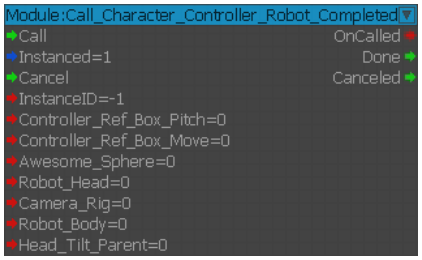
Port	Type	Description
Call	Any	Calls the module
Instanced	Boolean	Whether the module is instanced or not.
Cancel	Any	Cancels the module
InstanceID	Integer	Instance ID
Controller_Ref_Box_Pitch	Integer	Integer
Controller_Ref_Box_Move	Integer	Integer
Awesome_Sphere	Integer	Integer
Robot_Head	Integer	Integer
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

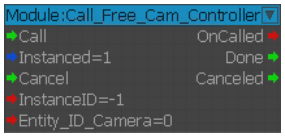
Port	Type	Description
Call	Any	Calls the module
Instanced	Boolean	Whether the module is instanced or not.
Cancel	Any	Cancels the module
InstanceID	Integer	Instance ID
Controller_Ref_Box_Pitch	Integer	Integer
Controller_Ref_Box_Move	Integer	Integer
Awesome_Sphere	Integer	Integer
Robot_Head	Integer	Integer
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

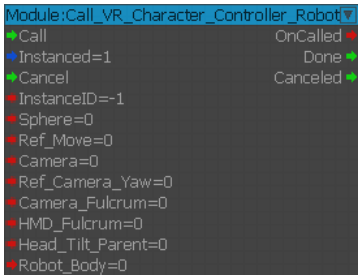
Port	Type	Description
Call	Any	Calls the module
Instanced	Boolean	Whether the module is instanced or not.
Cancel	Any	Cancels the module
InstanceID	Integer	Instance ID
Entity_ID_Camera	Integer	Integer

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	Cancels the module
InstanceID	Integer	Instance ID

Port	Type	Description
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
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

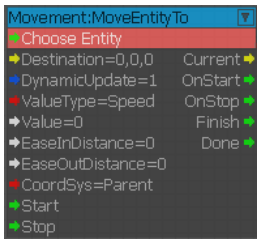
Flow Graph is deprecated and will be removed in a future version of Lumberyard. Learn about Script Canvas (p. 682), Lumberyard's new visual scripting environment.

You can use the following flow graph nodes to specify entity movements.

Topics

MoveEntityTo node

Used to move an entity to a destination position at a defined speed or in a defined interval of time.



Inputs

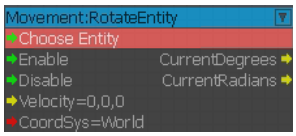
Port	Type	Description
Destination	Vec3	Position of the destination.
DynamicUpdate	Boolean	Indicates if destination position is to be followed if it changes.
ValueType	Integer	Type of input: Speed, Time,
Value	Float	Speed (m/sec) or Time (sec) value
EaseInDistance	Float	Distance from destination at which 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

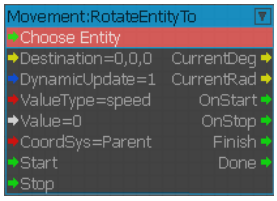
Port	Type	Description
Enable	Any	Enables updates
Disable	Any	Disables updates
Velocity	Vec3	Angular velocity (degrees/sec)
CoordSys	Integer	Coordinate system for rotation: World, Local

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
Done	Any	Activated when destination rotation is reached or Stop is triggered

Physics Nodes

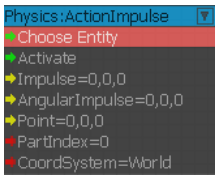
Flow Graph is deprecated and will be removed in a future version of Lumberyard. Learn about Script Canvas (p. 682), Lumberyard's new visual scripting environment.

You can use the following flow graph nodes to configure physics.

Topics

ActionImpulse node

Used to apply an impulse to an entity.

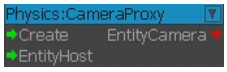


Inputs

Port	Type	Description
Activate	Any	Actives 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

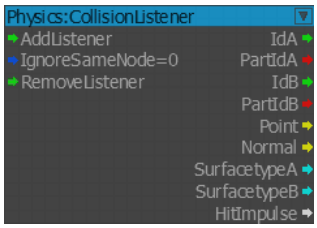
Port	Type	Description
Create	Any	Creates a physicalized camera proxy if one does not exist
EntityHost	Any	Syncs proxy rotation with the current view camera

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 colision point
Normal	Vec3	Collision normal
SurfacetypeA	String	Surface type of the first colliding entity
SurfacetypeB	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
MinTwist	Float	Lower rotation limit around TwistAxis
MaxTwist	Float	Upper rotation limit around TwistAxis
MaxBend	Float	Maximum bend of the TwistAxis

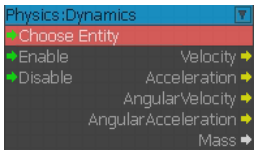
Outputs

Port	Type	Description
Id	Integer	Constraint ID

Port	Type	Description
Broken	Boolean	Triggered when the constraint breaks

Dynamics node

Used to output the dynamic state of an entity.



Inputs

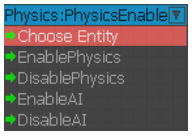
Port	Type	Description
Enable	Any	Enables updates
Disable	Any	Disables updates

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.

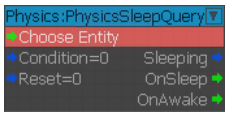


Inputs

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

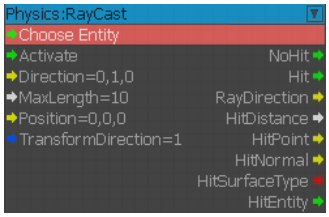
Port	Type	Description
Condition	Boolean	Sleeping state of the entity
Reset	Boolean	Resets the node

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

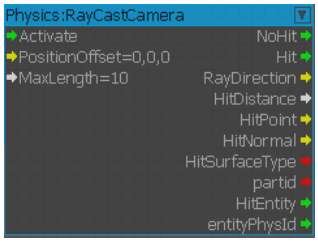
Port	Type	Description
Activate	Any	Activates the node
Direction	Vec3	Direction of the raycast
MaxLength	Float	Maximum length of the raycast
Position	Vec3	Ray start position relative to the entity
TransformDirection	Boolean	Transforms direction by entity orientation

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 suirface 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
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 suirface at the HitPoint
HitSurfaceType	Integer	Surface type index of the surface hit

Port	Type	Description
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

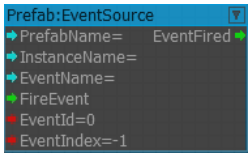
Flow Graph is deprecated and will be removed in a future version of Lumberyard. Learn about Script Canvas (p. 682), Lumberyard's new visual scripting environment.

You can use the following flow graph nodes to configure prefab settings.

Topics

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

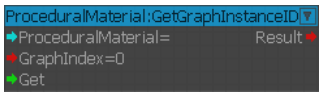
Flow Graph is deprecated and will be removed in a future version of Lumberyard. Learn about Script Canvas (p. 682), Lumberyard's new visual scripting environment.

You can use the following flow graph nodes to configure procedural material settings.

Topics

GetGraphInstanceID node

Used to get the graph instance ID.



Inputs

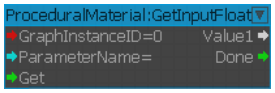
Port	Type	Description
ProceduralMaterial	String	Name of the procedural material
GraphicIndex	Integer	Graph index
Get	Any	Get the graph index

Outputs

Port	Type	Description
Result	Integer	Outputs the graph index

GetInputFloat node

Used to get the Substance input floating point value.



Inputs

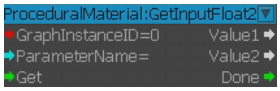
Port	Type	Description
GraphInstanceID	Integer	Graph instance ID
ParameterName	String	Parameter name
Get	Any	Get parameter value

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

Port	Type	Description
Value2	Integer	Outputs parameter value 2
Done	Any	Triggered when input completes

GetInputFloat3 node

Used to get the Substance input floating point values.

Inputs

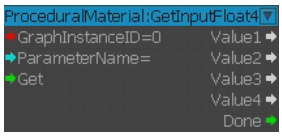
Port	Type	Description
GraphInstanceID	Integer	Graph instance ID
ParameterName	String	Parameter name
Get	Any	Get parameter value

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

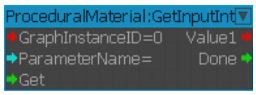
Port	Type	Description
GraphInstanceID	Integer	Graph instance ID
ParameterName	String	Parameter name
Get	Any	Get parameter value

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
Done	Any	Triggered when input completes

GetInput node

Used to get the Substance input value.



Inputs

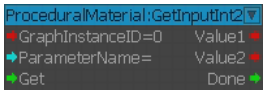
Port	Type	Description
GraphInstanceID	Integer	Graph instance ID
ParameterName	String	Parameter name
Get	Any	Get parameter value

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

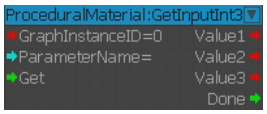
Port	Type	Description
GraphInstanceID	Integer	Graph instance ID
ParameterName	String	Parameter name
Get	Any	Get parameter value

Outputs

Port	Type	Description
Value1	Float	Outputs parameter value 1
Value2	Float	Outputs parameter value 2
Done	Any	Triggered when input completes

GetInput3 node

Used to get the Substance input value.



Inputs

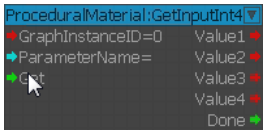
Port	Type	Description
GraphInstanceID	Integer	Graph instance ID
ParameterName	String	Parameter name
Get	Any	Get parameter value

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

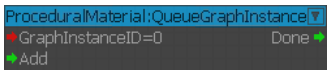
Port	Type	Description
Get	Any	Get parameter value

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 asyncronously

Outputs

Port	Type	Description
RenderBegin	Any	Triggered when rendering has started
RenderComplete	Any	Triggered when rendering has completed

RenderSync node

Used to render queued graphs synchronously.

Inputs

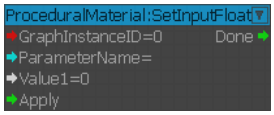
Port	Type	Description
Render	Any	Begin rendering graph instance syncronously

Outputs

Port	Type	Description
RenderComplete	Any	Triggered when rendering has completed

SetInputFloat node

Used to set the Substance input floating point value.



Inputs

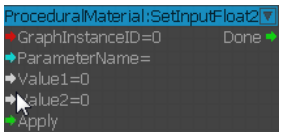
Port	Type	Description
GraphInstanceID	Integer	Graph instance ID
ParameterName	String	Parameter name
Value1	Float	Floating point parameter value to set
Apply	Any	Set parameter value

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

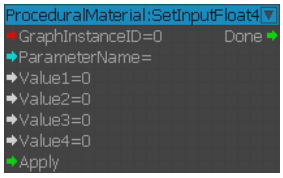
Port	Type	Description
GraphInstanceID	Integer	Graph instance ID
ParameterName	String	Parameter name
Value1	Float	Floating point parameter value 1 to set
Value2	Float	Floating point parameter value 2 to set
Value3	Float	Floating point parameter value 3 to set
Apply	Any	Set parameter value

Outputs

Port	Type	Description
Done	Any	Triggered when input completes

SetInputFloat4 node

Used to set the Substance input floating point values.



Inputs

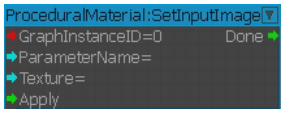
Port	Type	Description
GraphInstanceID	Integer	Graph instance ID
ParameterName	String	Parameter name
Value1	Float	Floating point parameter value 1 to set
Value2	Float	Floating point parameter value 2 to set
Value3	Float	Floating point parameter value 3 to set
Value4	Float	Floating point parameter value 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

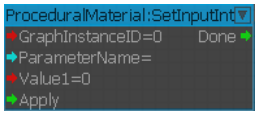
Port	Type	Description
Texture	String	Image to be set
Apply	Any	Set input image

Outputs

Port	Type	Description
Done	Any	Triggered when input completes

SetInputInt node

Used to set the Substance input value.



Inputs

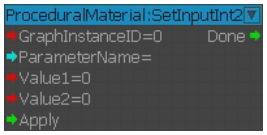
Port	Type	Description
GraphInstanceID	Integer	Graph instance ID
ParameterName	String	Parameter name
Value1	Float	Parameter value to set
Apply	Any	Set parameter value

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

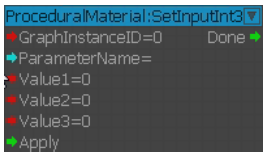
Port	Type	Description
Value1	Float	Parameter value 1 to set
Value2	Float	Parameter value 2 to set
Apply	Any	Set parameter value

Outputs

Port	Type	Description
Done	Any	Triggered when input completes

SetInputInt3 node

Used to set the Substance input values.



Inputs

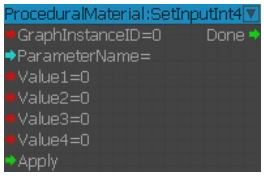
Port	Type	Description
GraphInstanceID	Integer	Graph instance ID
ParameterName	String	Parameter name
Value1	Float	Parameter value 1 to set
Value2	Float	Parameter value 2 to set
Value3	Float	Parameter value 3 to set
Apply	Any	Set parameter value

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
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

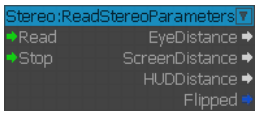
Flow Graph is deprecated and will be removed in a future version of Lumberyard. Learn about Script Canvas (p. 682), Lumberyard's new visual scripting environment.

You can use the following flow graph nodes to configure stereographic settings.

Topics

ReadStereoParameters node

Usd to read the HUD stereo display parameters.



Inputs

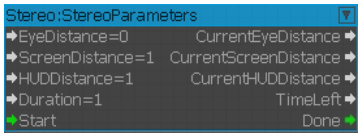
Port	Type	Description
Read	Any	Start reading stereo values
Stop	Any	Stop reading stereo values

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

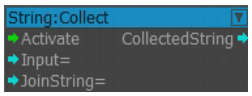
Flow Graph is deprecated and will be removed in a future version of Lumberyard. Learn about Script Canvas (p. 682), Lumberyard's new visual scripting environment.

You can use the following flow graph nodes to configure strings.

Topics

Collect node

Used to collect a string.



Inputs

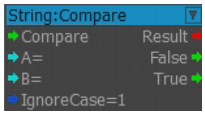
Port	Type	Description
Activate	Any	Collects the strings and triggers the output
Input	String	Each string that will be joined
JoinString	String	String to use between all collected strings

Outputs

Port	Type	Description
CollectedString	String	Outputs the collected string set

Compare node

Used to compare two strings.



Inputs

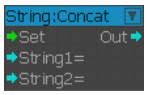
Port	Type	Description
Compare	Any	Triggers string comparison
A	String	First string to compare
B	String	Second string to compare
IgnoreCase	Boolean	Ignores casing

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

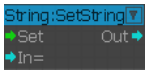
Port	Type	Description
Input	String	Triggers string replacement
Replace	String	The string to replace
ReplaceWith	String	The new string to replace with

Outputs

Port	Type	Description
OutString	String	Outputs the replaced string

SetString node

Used to set a string value.



Inputs

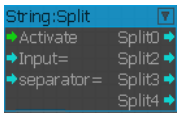
Port	Type	Description
Set	Any	Sends the string to the output
In	String	String to set on

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

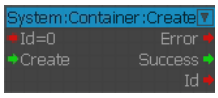
Flow Graph is deprecated and will be removed in a future version of Lumberyard. Learn about Script Canvas (p. 682), Lumberyard's new visual scripting environment.

You can use the following flow graph nodes to configure system settings.

Topics

Container:Create node

Used to create a container.



Inputs

Port	Type	Description
Id	Integer	Container ID
Create	Any	Creates a container

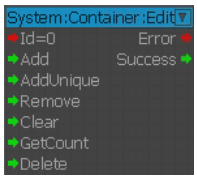
Outputs

Port	Type	Description
Error	Integer	Triggers when an error occurs

Port	Type	Description
Success	Any	Tiggers when a container is created
Id	Integer	Outputs the container ID

Container:Edit node

Used to edit a container.



Inputs

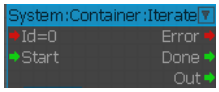
Port	Type	Description
Id	Integer	Container ID
Add	Any	Adds the passed item to the container
AddUnique	Any	Adds the passed item if it didn't exist
Remove	Any	Removes all occurrences of the current item
Clear	Any	Empties the container
GetCount	Any	Gets the number of items in the container
Delete	Any	Deletes 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

Port	Type	Description
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

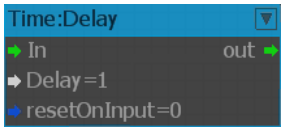
Flow Graph is deprecated and will be removed in a future version of Lumberyard. Learn about Script Canvas (p. 682), Lumberyard's new visual scripting environment.

You can use these flow graph nodes to define time settings.

Topics

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 `true`, 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

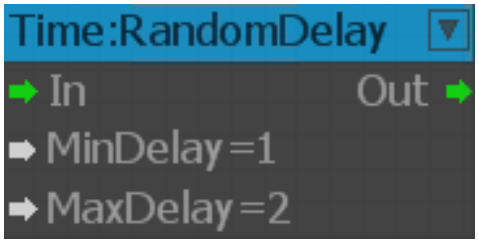
Port	Type	Description
Start	Any	Trigger to begin measuring time as it passes
Stop	Any	Trigger to stop measuring the elapsed time

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

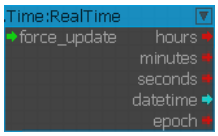
Port	Type	Description
In	Any	Value to pass after the specified delay time
MinDelay	Float	Minimum random delay time in seconds Default value: `1` Valid values: `0` – `100`
MaxDelay	Float	Maximum random delay time in seconds Default value: `2` Valid values: `0` – `100`

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

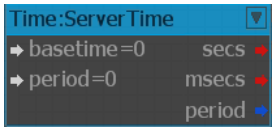
Port	Type	Description
force_update	Any	Forces an update of the system time

Outputs

Port	Type	Description
Hours	Integer	Current hour as reported by your system time
Minutes	Integer	Current minutes as reported by your system time
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`

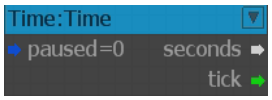
Port	Type	Description
Period	Float	Number of seconds that should pass before the timer resets to 0 Default value: `0` Valid values: `0` – `100`

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

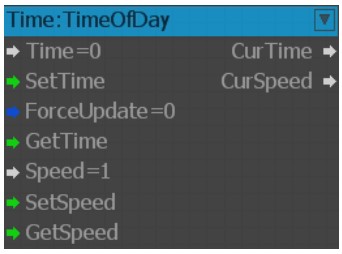
Port	Type	Description
Paused	Boolean	Pauses the time output when set to `true`. Default value: 0 Valid values: `0`=false | `1`=true

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

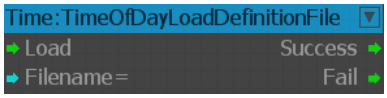
Port	Type	Description
Time	Float	Time of day in hours Default value: 0 Valid values: `0` – `24`
SetTime	Any	Trigger to change the time of day to the value specified for the Time parameter
ForceUpdate	Boolean	Immediately updates the sky when set to `true`. Default value: `0` Valid values: `0`=false | `1`=true
GetTime	Any	Retriggers 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`
SetSpeed	Any	Trigger to change the time of day speed to the value specified for the Speed parameter
GetSpeed	Any	Retriggers 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

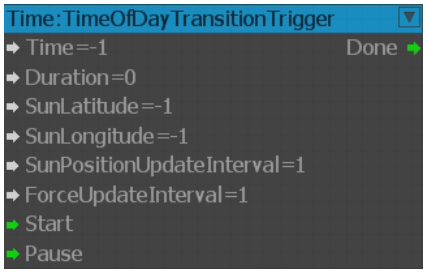
Port	Type	Description
Load	Any	Trigger to load and read the Time of Day definition file
Filename	String	Name of the XML file to load and read. The file must be in the level directory

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`

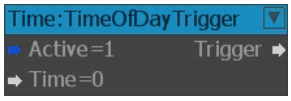
Port	Type	Description
		Valid values: `0` – `100`
SunLatitude	Float	Blends the level's current sun latitude value to the specified latitude in degrees. Set this value to `-1` to disable latitude blending. Default value: `-1` Valid values: `0` – `100`
SunLongitude	Float	Blends the level's current sun latitude value to the specified latitude in degrees. Set this value to `-1` to disable latitude blending Default value: `-1` Valid values: `0` – `100`
SunPositionUpdateIntervaFloat		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`
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

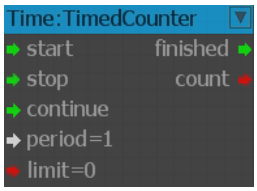
Port	Type	Description
Active	Boolean	Set this value to `true` to enable the trigger Default value: `1` Valid values: `0`=false | `1`=true
Time	Float	Triggers the action at the specified time of day Default value: `0` Valid values: `0` – `100`

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

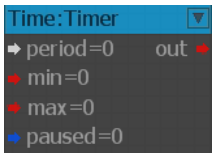
Port	Type	Description
Start	Any	Trigger to start the counter. If it is already running, this resets the counter.
Stop	Any	Stops the counter
Continue	Any	Resumes the counter
Period	Float	Tick period in seconds Default value: `1` Valid values: `0` – `100`
Limit	Integer	Default value: `0` Valid values: `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`
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 `true`. Default value: `0` Valid values: `0`=false | `1`=true

Outputs

Port	Type	Description
Out	Integer	Total count for the specified period

Twitch Nodes

Flow Graph is deprecated and will be removed in a future version of Lumberyard. Learn about Script Canvas (p. 682), Lumberyard's new visual scripting environment.

You can use these flow graph nodes to configure Twitch-related settings.

You the attached entity dynamically by using the **Choose Entity** input port that is included on a number of flow graph nodes.

Topics

Twitch ChatPlay General Nodes

You can use the following flow graph nodes to configure general Twitch ChatPlay-related settings.

Twitch:Chatplay:Available node

Inputs

Port	Type	Description
Activate	Void	Checks the availability of Twitch ChatPlay.

Outputs

Port	Type	Description
Available	Void	Indicates that Twitch ChatPlay is available
Unavailable	Void	Indicates that Twitch ChatPlay is not available

Twitch:Chatplay:Channel node

Inputs

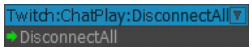
Port	Type	Description
Channel	String	Twitch channel name

Port	Type	Description
Connect	Void	Initiates connection; idempotent if called while already connected or connecting. Resets the `Error` 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

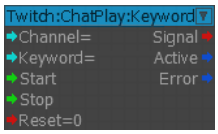
Twitch:Chatplay:DisconnectAll node



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 `Reset` 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 `Start` is triggered); otherwise, false (set as soon as `Stop` is triggered)
Error	Boolean	Indicates that an error has occurred

Twitch:Chatplay:RegisterCredentials node

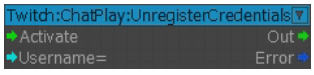
Inputs

Port	Type	Description
Activate	Void	Registers the user name and OAuth token credential pair
Username	String	Twitch user name
OAuth_Token	String	OAuth tokens are generated with the Twitch Chat OAuth Password Generator.

Outputs

Port	Type	Description
Out	Void	Signaled when done registering credentials
Error	Boolean	Indicates that an error has occurred

Twitch:Chatplay:UnregisterCredentials node



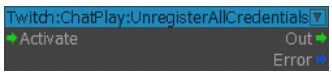
Inputs

Port	Type	Description
Activate	Void	Unregisters the user name and associated OAuth token
Username	String	Twitch user name

Outputs

Port	Type	Description
Out	Void	Indicates when the unregistering of the credential has finished
Error	Boolean	Indicates that an error has occurred

Twitch:Chatplay:UnregisterAllCredentials node



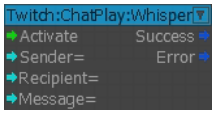
Inputs

Port	Type	Description
Activate	Void	Unregisters all credentials at once

Outputs

Port	Type	Description
Out	Void	Indicates when the unregistering of all credential has finished
Error	Boolean	Indicates 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 user name of sender; must have credentials registered to successfully send a whisper (see **Twitch:ChatPlay:RegisterCredentials** node)
Recipient	String	Twitch user name of recipient
Message	String	Message to whisper to recipient

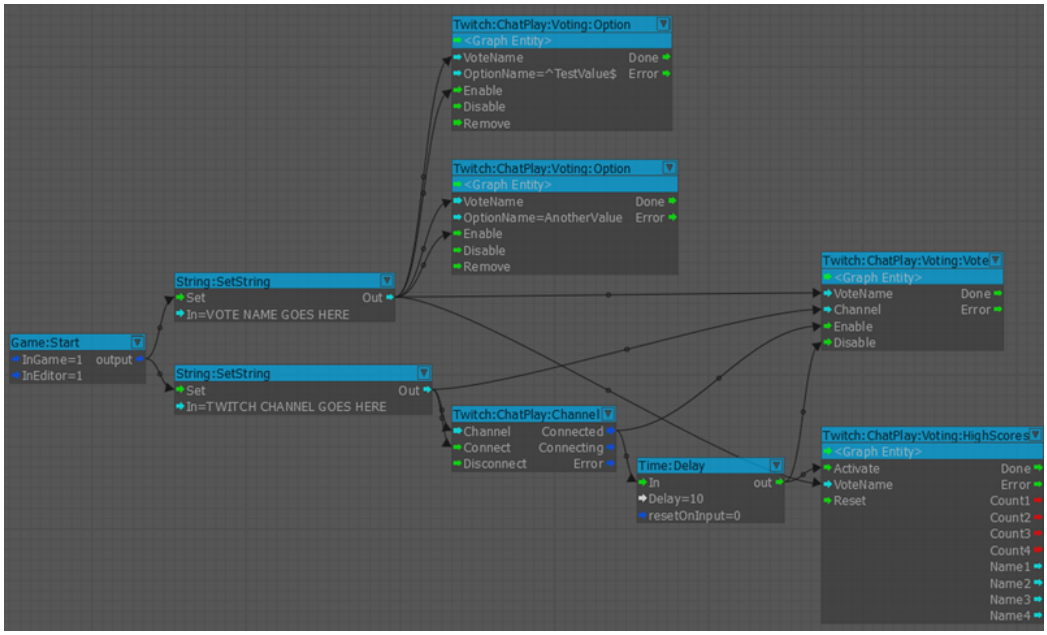
Outputs

Port	Type	Description
Success	Void	Signaled if the whisper is sent successfully.

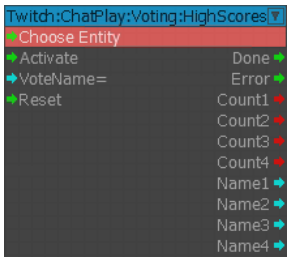
Port	Type	Description
Error	Boolean	Signaled as true if an error occurred.

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



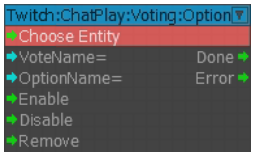
Inputs

Port	Type	Description
Activate	Void	Queries the high scores
VoteName	String	The name of the vote
Reset	Void	Resets all counts to zero

Outputs

Port	Type	Description
Done	Void	Indicates when the operation is complete
Error	Void	Indicates that an error occurred
Count1 - Count4	Integer	Indicates 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



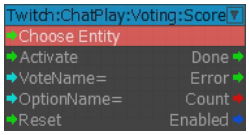
Inputs

Port	Type	Description
VoteName	String	The name of the vote
OptionName	String	The name of the voting option
Enable	Void	Enables the option and that it can be voted on
Disable	Void	Disables the ability to vote on the option
Remove	Void	Deletes the option

Outputs

Port	Type	Description
Done	Void	Indicates when the operation is complete
Error	Void	Indicates that an error occurred

Twitch:Chatplay:Voting:Score node



Inputs

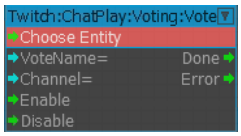
Port	Type	Description
Activate	Void	Queries the score for an option.
VoteName	String	The name of the vote
OptionName	String	The name of the voting option

Port	Type	Description
Reset	Void	Resets the count to zero

Outputs

Port	Type	Description
Done	Void	Indicates when the operation is complete
Error	Void	Indicates that an error occurred
Count	Integer	Indicates the current vote count
Enabled	Boolean	Indicates 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	Enables the vote and that it can be voted on
Disable	Void	Disables the ability to vote on the vote

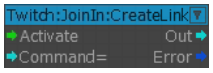
Outputs

Port	Type	Description
Done	Void	Indicates when the operation is complete
Error	Void	Indicates 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 session.

Twitch:Joinin:CreateLink node



Inputs

Port	Type	Description
Activate	Void	Generates a `game:` 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	Signaled with the generated link.
Error	Boolean	Indicates 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
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 is completed
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

Flow Graph is deprecated and will be removed in a future version of Lumberyard. Learn about Script Canvas (p. 682), Lumberyard's new visual scripting environment.

You can use the following flow graph nodes to define vector (Vec3) operations.

Topics

AddVec3 node

Used to output the sum of two vectors.

Inputs

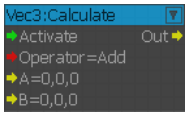
Port	Type	Description
Activate	Any	Triggers the node
A	Vec3	First operand
B	Vec3	Second operand

Outputs

Port	Type	Description
Out	Vec3	Addition of A and B

Calculate node

Used to output the specified calculation between two vectors.



Inputs

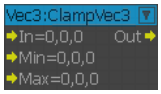
Port	Type	Description
Activate	Any	Triggers the node
Operator	Integer	Math operation to perform
A	Vec3	First operand
B	Vec3	Second operand

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

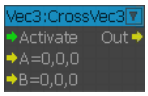
Port	Type	Description
In	Vec3	Input value
Min	Vec3	Minimum clamping value
Max	Vec3	Maximum clamping value

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

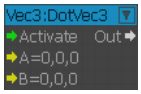
Port	Type	Description
Activate	Any	Triggers the node
A	Vec3	First operand
B	Vec3	Second operand

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

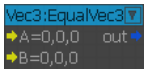
Port	Type	Description
Activate	Any	Triggers the node
A	Vec3	First operand
B	Vec3	Second operand

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

Port	Type	Description
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
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

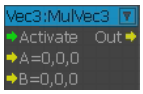
Port	Type	Description
Vector	Vec3	Input vector

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

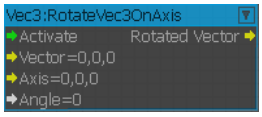
Port	Type	Description
Vector	Vec3	Input vector

Outputs

Port	Type	Description
Length	Float	Reciprocal value of input

RotateVec3onAxis node

Used to output an axis-rotated value of the vector.



Inputs

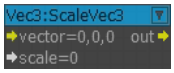
Port	Type	Description
Activate	Any	Triggers the node
Vector	Vec3	Input vector to rotate
Axis	Vec3	Axis to rotate input around
Angle	Float	Angle in degrees to rotate

Outputs

Port	Type	Description
Rotated Vector	Vec3	Result of the rotation

ScaleVec3 node

Used to output a scaled value of the vector.



Inputs

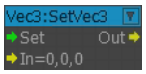
Port	Type	Description
Vector	Vec3	Input vector
Scale	Float	Scale factor to apply to the input

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

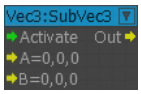
Port	Type	Description
Set	Any	Triggers the vector to the output
In	Vec3	Input value

Outputs

Port	Type	Description
Out	Vec3	Outputs the input value

SubVec3 node

Used to output the subtracted value of two vectors.



Inputs

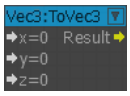
Port	Type	Description
Activate	Any	Triggers the node
A	Vec3	First operand
B	Vec3	Second operand

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

Flow Graph is deprecated and will be removed in a future version of Lumberyard. Learn about Script Canvas (p. 682), Lumberyard's new visual scripting environment.

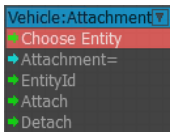
You can use the following flow graph nodes to configure vehicle behavior and related settings.

> **Note**
> These nodes will only work with the Legacy Game Sample (CryEngine GameSDK), which is available at Lumberyard Downloads.

Topics

Attachment node

Used to control vehicle entity attachments.

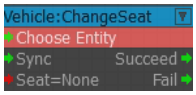


Inputs

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

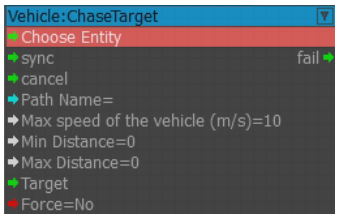
Port	Type	Description
Sync	Any	Triggers the seat change
Seat	Integer	Seat to change to

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

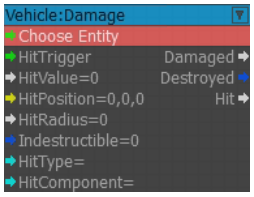
Port	Type	Description
Sync	Any	Triggers the chase
Cancel	Any	Cancels the chase
Path Name	String	Name of the path to follow
Max speed of the vehicle	Float	Maximum speed of the vehicle
Min Distance	Float	Minimum chase distance to the target
Max Distance	Float	Minimum chase distance to the target
Target	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

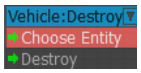
Port	Type	Description
Indestructible	Boolean	Value of true sets the vehicle to be indestructible
HitType	String	Type of damage
HitComponent	String	Vehicle component that will receive the hit

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.

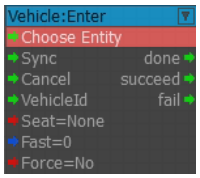


Inputs

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

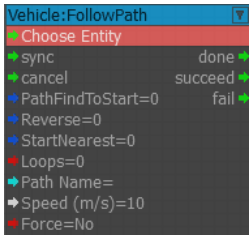
Port	Type	Description
Sync	Any	Activates the node
Cancel	Any	Cancels the operation
VehicleId	Any	ID of the vehicle
Seat	Integer	Seat to sit on
Fast	Boolean	Skip approach and enter vehicle
Force	Integer	Force execution method

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

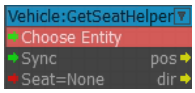
Port	Type	Description
Sync	Any	Activates the node
Cancel	Any	Cancels execution
PathFindToStart	Boolean	Whether to find the start of the path
Reverse	Boolean	Reverses the path direction
StartNearest	Boolean	Starts the path at the nearest point on path
Loops	Integer	Number of times to loop around the path
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

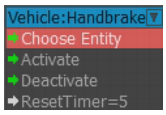
Port	Type	Description
Sync	Any	Get helper position
Seat	Integer	Seat to be entered

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.

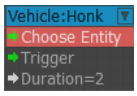


Inputs

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.

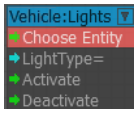


Inputs

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.

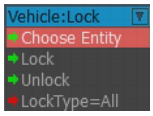


Inputs

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.

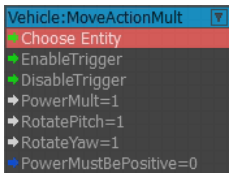


Inputs

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.



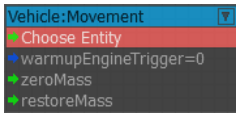
Inputs

Port	Type	Description
EnableTrigger	Any	Activates the node
DisableTrigger	Any	Deactivates the node
PowerMult	Float	Vehicle engine power multiplier

Port	Type	Description
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.

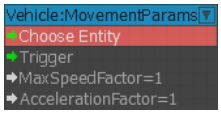


Inputs

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.

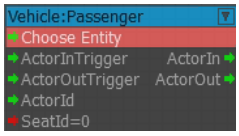


Inputs

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

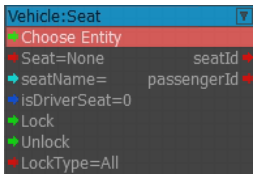
Port	Type	Description
ActorInTrigger	Any	Forces actor to get into vehicle if a seat is available
ActorOutTrigger	Any	Forces actor to get out of the vehicle
ActorId	Any	ID of the action
SeatId	Integer	ID of the seat

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

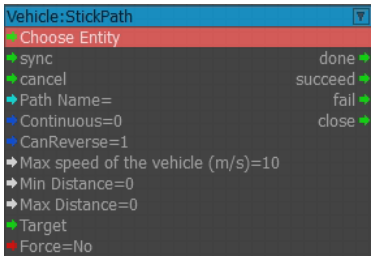
Port	Type	Description
Seat	Integer	ID of seat
SeatName	String	Name of seat
IsDriverSeat	Boolean	True is driver seat
Lock	Any	Locks the vehicle
Unlock	Any	Unlocks the vehicle
LockType	Integer	Type of vehicle lock

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

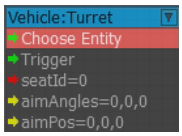
Port	Type	Description
Sync	Any	Activates the node
Cancel	Any	Cancels execution
Path Name	String	Name of path
Continuous	Boolean	Whether vehicle can continue to follow the path or stops once it reaches the targer
CanReverse	Boolean	Whether vehicle is allowed to drive in reverse to follow target or path
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.

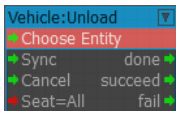


Inputs

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

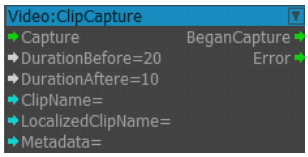
> Flow Graph is deprecated and will be removed in a future version of Lumberyard. Learn about Script Canvas (p. 682), Lumberyard's new visual scripting environment.

You can use the following flow graph nodes to configure video settings.

Topics

ClipCapture node

Used to capture video clips while a game is running and save them locally or to the cloud.



Inputs

Port	Type	Description
Capture	Any	Begins capturing a video clip
DurationBefore	Float	Records the specified number of seconds before the **Capture** input triggers
DurationAfter	Float	Records the specified number of seconds after the **Capture** input triggers
ClipName	String	Usage details are specific to the operating system
LocalizedClipName	String	Usage details are specific to the operating system
Metadata	String	(Optional) Tags video clips

Outputs

Port	Type	Description
BeganCapture	Any	Triggers when video clip capturing begins
Error	Any	Triggers when a clip capture error occurs

VideoPlayback Nodes

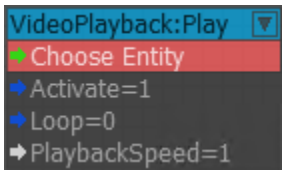
Flow Graph is deprecated and will be removed in a future version of Lumberyard. Learn about Script Canvas (p. 682), Lumberyard's new visual scripting environment.

You can use the following flow graph nodes to configure video playback settings.

Topics

Play Node

Plays the video on the defined entity with the **VideoPlayback** component.

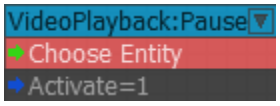


Inputs

Port	Type	Description
Activate	Boolean	Value of true begins playing video.
Loop	Boolean	Value of true loops video.
PlaybackSpeed	Float	Sets speed of playback. Value of 1 is normal speed, 0.5 is half speed, 2.0 is double speed, and so on.

Pause Nodes

Pauses the video on the defined entity with the **VideoPlayback** component.

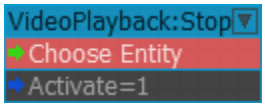


Input

Port	Type	Description
Activate	Boolean	Value of true pauses the video.

Stop Nodes

Stops the video on the defined entity with the **VideoPlayback** component.

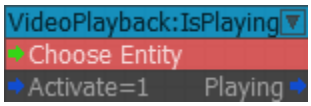


Input

Port	Type	Description
Activate	Boolean	Value of true stops the video. If video playback is resumed, it begins at the first frame of the video.

IsPlaying Nodes

Determines whether the **VideoPlayback** component on the defined entity is currently playing video.



Input

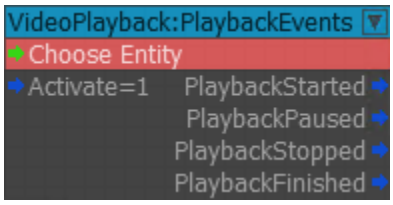
Port	Type	Description
Activate	Boolean	Value of true triggers output to the **Playing** output node. Output is triggered whether video is playing or not.

Output

Port	Type	Description
Playing	Boolean	Returns true if video is playing, false if video is paused or stopped.

PlaybackEvents Nodes

Provides a series of outputs that are triggered when the defined entity's **VideoPlayback** component reaches certain conditions.



Input

Port	Type	Description
Activate	Boolean	Value of true triggers node to listen for video playback events on the defined entity.

Outputs

Port	Type	Description
PlaybackStarted	Boolean	Triggered when the video begins playback.
PlaybackPaused	Boolean	Triggered when video playback is paused.
PlaybackStopped	Boolean	Triggered when video playback is stopped. If a video is not set to loop and finishes without user intervention, this is not triggered; this is triggered only when **Stop** is directly called.
PlaybackFinished	Boolean	Triggered when video playback finishes. If a video is set to loop, this is called every time that video finishes a loop.

Virtual Reality (VR) Nodes

Flow Graph is deprecated and will be removed in a future version of Lumberyard. Learn about Script Canvas (p. 682), Lumberyard's new visual scripting environment.

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: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 Contoller 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.

> **Note**
> This is now a legacy flow graph node, and has been replaced by VR:Playspace (p. 1037).

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:Playspace

Configures the playspace for the Oculus and OpenVR SDKs. The corners represent a rectangle, and run counter-clockwise around the z-axis.

Node Inputs

Activate

Gets information about the HMD's playspace.

Node Outputs

Corner0

Vector value. The world-space position of corner 0.

Corner1

Vector value. The world-space position of corner 1.

Corner2

Vector value. The world-space position of corner 2.

Corner3

Vector value. The world-space position of corner 3.

Center

Vector value. The world-space center of the playspace. Note that the center is on the floor.

Dimensions

The width and height 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:SetTrackingLevel

Sets the `TrackingLevel` of the attached VR device to **0** (head), **1** (floor), or **2** (fixed). When setting the tracking level to **2**, the HMD does not apply translation based on the position of the device, only rotation is applied. This is to aid VR experiences that required a fixed head position, such as viewing 360 degree video.

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 oritentation 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

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.

Weapon Nodes

Flow Graph is deprecated and will be removed in a future version of Lumberyard. Learn about Script Canvas (p. 682), Lumberyard's new visual scripting environment.

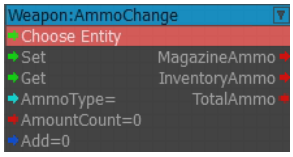
You can use the following flow graph nodes to configure weapon settings.

> **Note**
> These nodes will only work with the Legacy Game Sample (CryEngine GameSDK), which is available at Lumberyard Downloads.

Topics

AmmoChange node

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

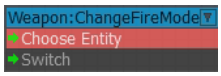
This node

Inputs

Port	Type	Description
Enemy	Vec3	Aims the weapon at the enemy's position

ChangeFireMode node

Used to change the weapon fire mode.

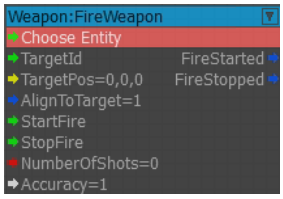


Inputs

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

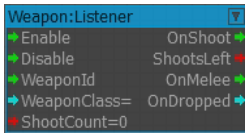
Port	Type	Description
TargetId	Any	Target ID
TargetPos	Vec3	Target position
AlignToTarget	Boolean	Aims the weapon at the target
StartFire	Any	Starts firing weapon
StopFire	Any	Stops firing weapon
NumberOfShots	Integer	Fires the specified number of shots
Accuracy	Float	Specifies firing accuracy from 0% 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

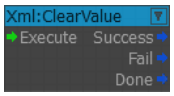
Flow Graph is deprecated and will be removed in a future version of Lumberyard. Learn about Script Canvas (p. 682), Lumberyard's new visual scripting environment.

You can use the following flow graph nodes to specify XML elements.

Topics

ClearValue node

Used to clear the value of the active element.



Inputs

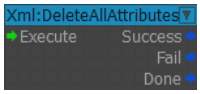
Port	Type	Description
Execute	Any	Executes the instruction

Outputs

Port	Type	Description
Success	Boolean	Called if the instruction executed successfully
Fail	Boolean	Called if the instruction failed
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

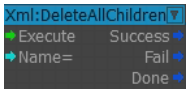
Outputs

Port	Type	Description
Success	Boolean	Called if the instruction executed successfully
Fail	Boolean	Called if the instruction failed

Port	Type	Description
Done	Boolean	Called when the instruction has completed carrying out

DeleteAllChildren node

Used to delete all children of the active element.



Inputs

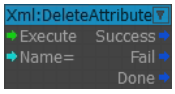
Port	Type	Description
Execute	Any	Executes the instruction
Name	String	Optional child name

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

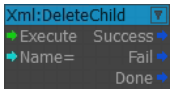
Outputs

Port	Type	Description
Success	Boolean	Called if the instruction executed successfully
Fail	Boolean	Called if the instruction failed

Port	Type	Description
Done	Boolean	Called when the instruction has completed carrying out

DeleteChild node

Used to delete the first child node with the given name.



Inputs

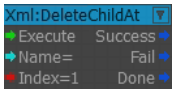
Port	Type	Description
Execute	Any	Executes the instruction
Name	String	Optional child name

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

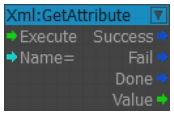
Outputs

Port	Type	Description
Success	Boolean	Called if the instruction executed successfully

Port	Type	Description
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

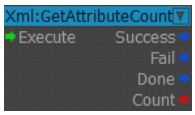
Port	Type	Description
Execute	Any	Executes the instruction
Name	String	Name of the attribute

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

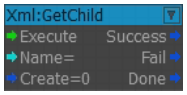
Outputs

Port	Type	Description
Success	Boolean	Called if the instruction executed successfully

Port	Type	Description
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

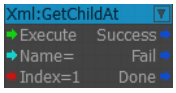
Port	Type	Description
Execute	Any	Executes the instruction
Name	String	Name of the attribute
Create	Boolean	Creates a child node if one does not exist

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

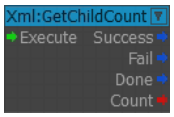
Port	Type	Description
Execute	Any	Executes the instruction
Name	String	Name of the child node
Index	Integer	Location of the child node in the list

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

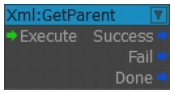
Port	Type	Description
Execute	Any	Executes the instruction

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

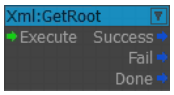
Outputs

Port	Type	Description
Success	Boolean	Called if the instruction executed successfully

Port	Type	Description
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

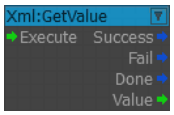
Port	Type	Description
Execute	Any	Executes the instruction

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

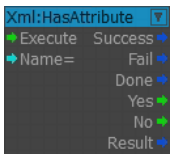
Outputs

Port	Type	Description
Success	Boolean	Called if the instruction executed successfully
Fail	Boolean	Called if the instruction failed

Port	Type	Description
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

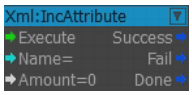
Port	Type	Description
Execute	Any	Executes the instruction
Name	String	Name of the attribute

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

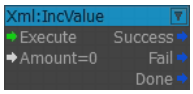
Port	Type	Description
Amount	Float	Amount to increment by

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

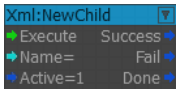
Port	Type	Description
Execute	Any	Executes the instruction
Amount	Float	Amount to increment by

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

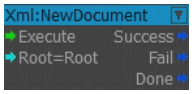
Port	Type	Description
Name	String	Name of the child node
Active	Boolean	Makes the child node the active element

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

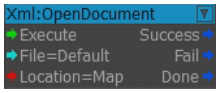
Port	Type	Description
Execute	Any	Executes the instruction
Root	String	Name of the XML root element

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

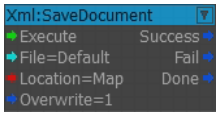
Port	Type	Description
File	String	File name of the XML document
Location	Integer	File path of the XML document

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

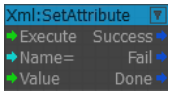
Port	Type	Description
Execute	Any	Executes the instruction
File	String	File name of the saved XML document
Location	Integer	File path of the XML document
Overwrite	Boolean	Determines where document should overwrite existing XML document

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

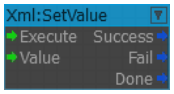
Port	Type	Description
Execute	Any	Executes the instruction
Name	String	Name of the attribute to set
Value	Any	Sets the value of the element

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

Flow Graph is deprecated and will be removed in a future version of Lumberyard. Learn about Script Canvas (p. 682), Lumberyard's new visual scripting environment.

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 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

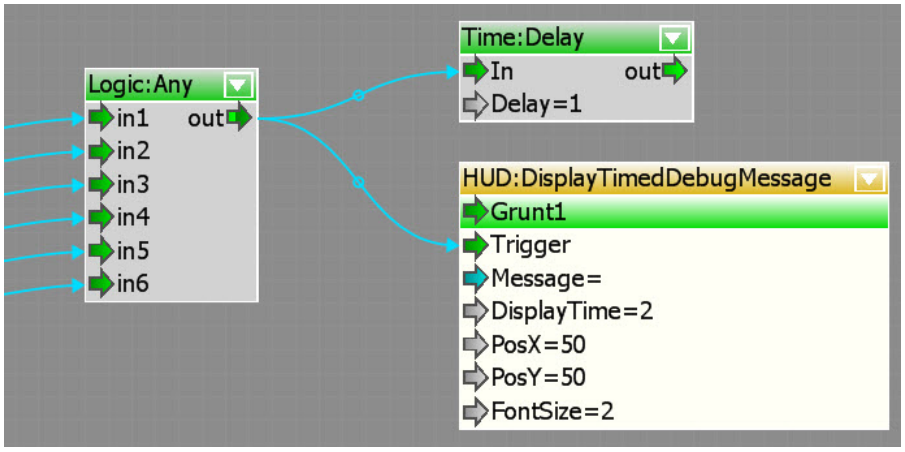
1. Click the link to select it, right-click the dot in the middle of the link, then click **Remove**.
2. Alternatively, click the input port the link is connected to and drag it away from the port. When the mouse is released, the link disappears.

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** outport 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`.

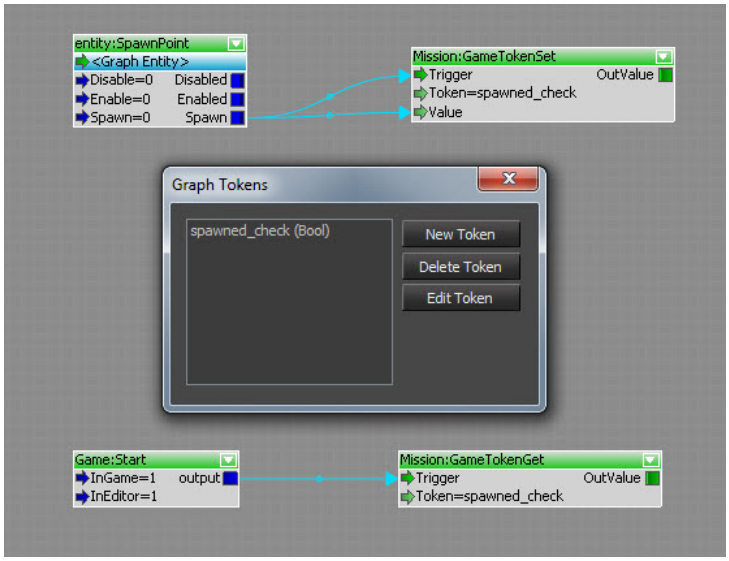
- To highlight outgoing links blue, select an output node and press `G`.

Using Flow Graph Tokens

Flow Graph is deprecated and will be removed in a future version of Lumberyard. Learn about Script Canvas (p. 682), Lumberyard's new visual scripting environment.

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

Flow Graph is deprecated and will be removed in a future version of Lumberyard. Learn about Script Canvas (p. 682), Lumberyard's new visual scripting environment.

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

Flow Graph is deprecated and will be removed in a future version of Lumberyard. Learn about Script Canvas (p. 682), Lumberyard's new visual scripting environment.

Topics

Using Flow Graph Debugger

Flow Graph is deprecated and will be removed in a future version of Lumberyard. Learn about Script Canvas (p. 682), Lumberyard's new visual scripting environment.

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

Flow Graph is deprecated and will be removed in a future version of Lumberyard. Learn about Script Canvas (p. 682), Lumberyard's new visual scripting environment.

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

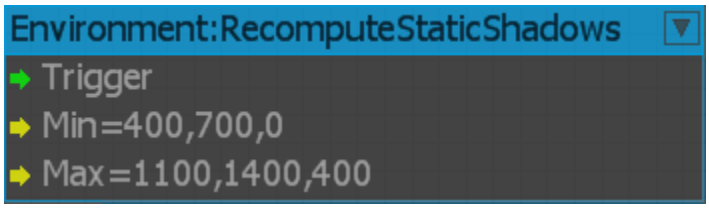
Flow Graph is deprecated and will be removed in a future version of Lumberyard. Learn about Script Canvas (p. 682), Lumberyard's new visual scripting environment.

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. With the Modular Gems System (p. 1066), you can choose the features and assets that you need for your game project without including unnecessary components. For example, you might add the Boids Gem to create entities for a flock of birds to place in your game. For more information, see Boids Gem (p. 1070).

All Lumberyard Gems are located in the following directory:

```
lumberyard_version\dev\Gems
```

The Gems that you enable are automatically detected and built through the integrated Waf Build System (p. 1797).

> **Note**
> To enable Gems, you must select the **Compile the game code** option in Lumberyard Setup Assistant. For more information, see Running Lumberyard Setup Assistant (p. 20).

You can enable gems with the Project Configurator (p. 28) or the command line (`Lmbr.exe` (p. 1208)).

Topics

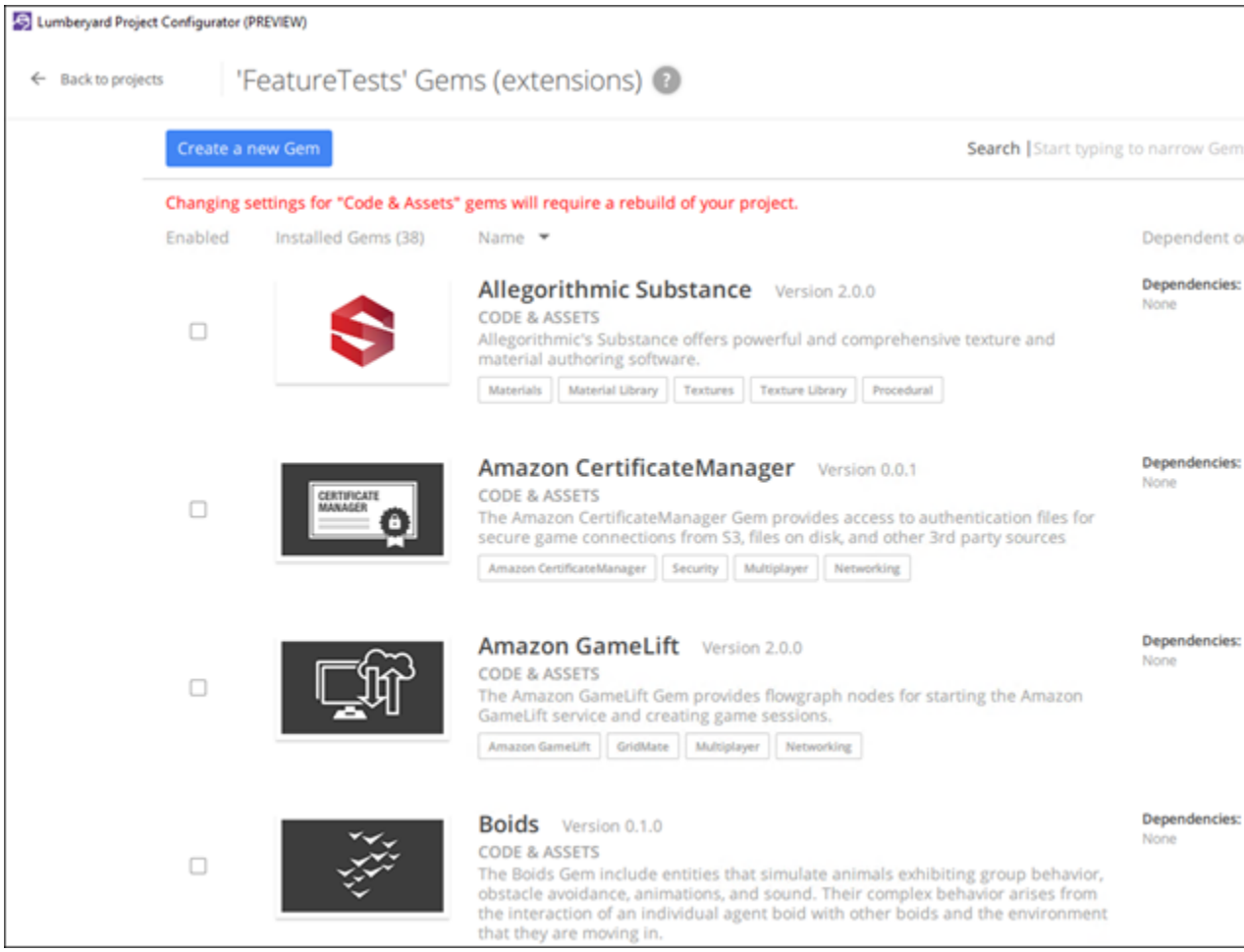
Enabling Gems with the Project Configurator

Start the Project Configurator and select the Gems that you want to enable for your game project. After you enable any Gem labeled **Code & Assets**, you will have to rebuild your project (p. 1062).

To enable Gems with the Project Configurator

1. Navigate to the `lumberyard_version\dev\Tools\LmbrSetup\Win\` directory and then start `ProjectConfigurator`.
2. In the Project Configurator, select your active game project and choose **Set as default**.
3. Click **Enable Gems**.

4. Select the Gems that you want to enable.



5. Click **Save**.
6. If you enabled any **Code & Assets** Gems, you must rebuild your project (p. 1062). **Assets Only** Gems do not require rebuilding.

Enabling Gems from the Command Line

You can also enable Gems from the command line (`Lmbr.exe` (p. 1208)). After you enable **Code & Assets** Gems, you will have to rebuild your project (p. 1062).

To enable Gems with the command line

1. Open a command line and navigate to the *`lumberyard_version`*`\dev\Tools\LmbrSetup\Win\` directory.
2. Type: **lmbr Gems enable** ***`ProjectName NameOfGem`***.

   The project name and the Gem name are case sensitive.

   Tip

   - To list all possible commands, type: **lmbr -help**.
   - Use the help command for more information about `lmbr` commands. For example, for more information about the **lmbr Gems enable** command, type: **lmbr gems enable -help**

- To list all the Gem names, type: **lmbr gems list**

3. If you enabled any **Code & Assets** Gems, you must rebuild your project (p. 1062). **Assets Only** Gems do not require rebuilding.

Rebuilding Your Game Project

After you enable one or more Gems labeled **CODE & ASSETS**, you must rebuild your game project to make the Gems available in Lumberyard Editor. If you enabled only Gems labeled **ASSETS ONLY**, you do not need to rebuild your game project before opening it in Lumberyard Editor.

To rebuild your game project after enabling Code & Assets Gems

1. Open a command line window and navigate to the `lumberyard_version\dev` directory.
2. Type: **lmbr_waf configure**
3. If the previous step was successful, type one of the following **lmbr_waf build** commands to enable the Gems for your game project:

   - For Visual Studio 2015, type: **lmbr_waf build_win_x64_vs2015_profile -p game**
   - For Visual Studio 2013, type: **lmbr_waf build_win_x64_vs2013_profile -p game**

   For more information about other build commands or variables, see Build Configuration (p. 1815).
4. After the previous command was successful, start Lumberyard Editor. You can now use your Gems for your game project.

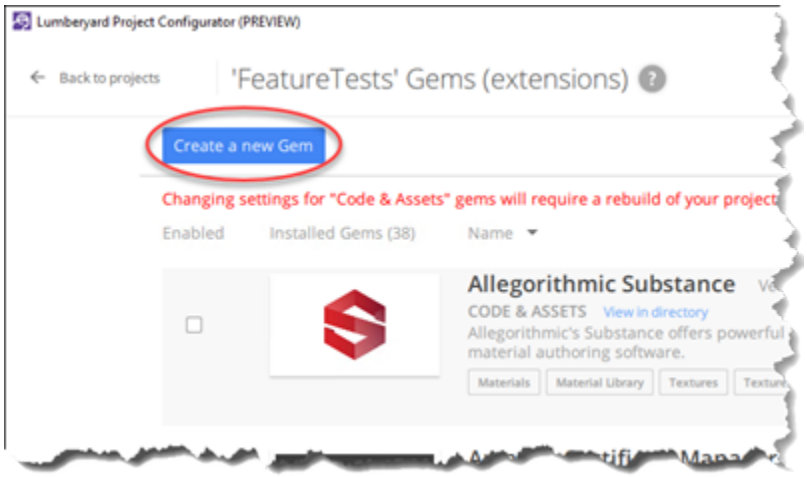
Creating a Gem with the Project Configurator

You can also create your own Gems for your game project. You can do this in either the Project Configurator or the command line (p. 1065).

If you add a **Code & Assets** Gem, you must rebuild your project (p. 1062) after adding the Gem. **Asset Only** Gems do not require a rebuild.

To create a Gem with the Project Configurator

1. Navigate to the `lumberyard_version\dev\Tools\LmbrSetup\Win\` directory and then start `ProjectConfigurator`.
2. Select the project and click **Enable Gems**.
3. Click **Create a new Gem** on the Gems page for the game project that you selected.



4. In the **Create a new Gem** dialog box:

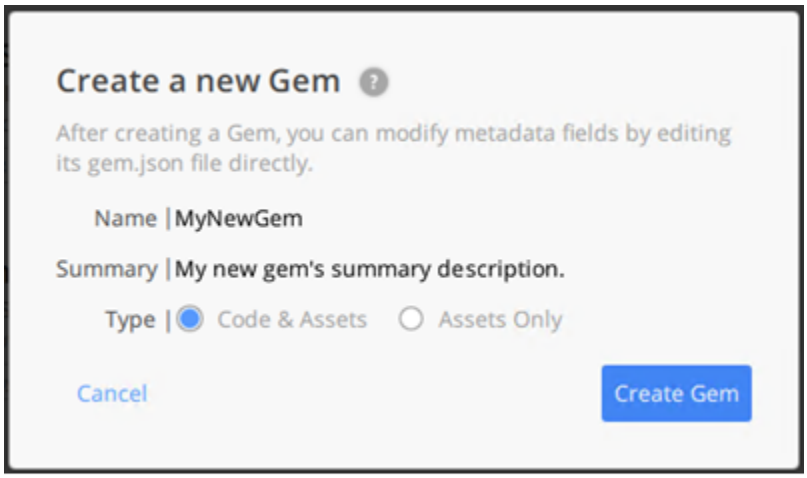
   1. For **Name**, type a name for the Gem (for example, ***MyNewGem***). The name must start with a letter, and only alphanumeric characters and underscore are allowed for the rest of the name. Other special characters or spaces are not allowed in the name.
   2. (Optional) For **Summary**, type a description for the Gem.

      This field also supports hyperlinks using a simple html anchor tag. When typing this directly into the **Summary** field in the Project Configurator, use the following syntax:

      ```
      <a href="http://www.amazon.com">Amazon</a>
      ```

      If you are editing the `Summary` parameter in the `gem.json` file, however, use the following syntax:

      ```
      "Summary": "<a href=\"http:\/\/www.amazon.com\">Amazon<\/a>",
      ```
   3. For **Type**, select **Code & Assets** or **Assets Only**.
   4. Choose **Create Gem**.



5. Close the Project Configurator.

   The following directories and files are created in `/dev/Gems/Gem_name/` for your Gem:

- `Assets` – Directory of assets for your Gem, such as materials, models, textures, and audio files. The AZ::IO system automatically includes this directory so that you can reference assets provided by your Gem. For example, if you have a material file, `/Assets/materials/MyMaterial.mtl`, you can reference it in your code with the path `materials/MyMaterial.mtl`.

  Note
  If this directory is not present, you can manually add it. If you don't have any assets in your Gem, you can remove this folder.

- `Gem.json` – Metadata for this Gem. Do not modify the `Uuid` or `Name` fields. You can specify the following Gem metadata fields:
  - `Version` – *`n.n.n`* (numerical values only).

    Gem revision is not supported, so you should not plan to increment the revision after shipping the Gem.
  - `DisplayName` – Friendly display name. Can contain spaces.
  - `Tags` – Searchable tags that are displayed in Project Configurator under the Gem's summary. Enter this as a comma-separated list of strings.
  - `IconPath` – Path to the Gem's display icon.
  - `Summary` – Detailed description of the Gem. This parameter supports hyperlinks. To specify a hyperlink, use the following syntax:

    ```
    "Summary": "<a href=\"http:\/\/www.amazon.com\">Amazon<\/a>",
    ```

  - `Dependencies` – You can specify a dependency on another Gem or on an engine version. Edit the `_comment` metadata.

    ```
    "Dependencies": [
       {
           "Uuid": "540faf970c994668b5d02c66a39c6625",
           "VersionConstraints": [
               ">1"
           ],
           "_comment": "zzzTestVer001"
       }
    ],
    ```

- `preview.png` – Preview image displayed in the **Project Configurator** Gem's list.
- `Code\Gem_name.waf_files` – Waf files json for your Gem that specifies which files are built, how files should be combined into uber files, and how files are filtered in the Visual Studio Project.
- `Code\Gem_name_test.waf_files` – Additional files to include when building with the test specification.
- `Code\wscript` – Waf wscript Python file that defines libraries and other build settings used by your Gem.
- `Code\Include\Gem_name\Gem_name_Bus.h` – Globally visible header that can be included by any project that uses your Gem, or by other Gems that depend on it. External code can call into your Gem's module and receive events from your module through public event buses. Event buses allow simple and safe function calls between different modules of code.
- `Code\Source\Gem_nameModule.cpp` – Implementation of the `Gem_nameModule`. Starting in Lumberyard 1.5, new Gems are built around AZ modules. The default module registers one system component, the `Gem_nameSystemComponent`.
- `Code\Source\Gem_nameSystemComponent.h` – Header for the `Gem_nameSystemComponent` implementation. This component is registered by the `Gem_nameModule` and is a handler for the `Gem_nameBus`.

- `Code\Source\Gem_nameSystemComponent.cpp` – `Gem_nameSystemComponent` implementation. Inside this component are the typical `Activate`, `Deactivate`, `Init`, and `Reflect` methods.
- `Code\Source\StdAfx.cpp` – `Code\Source\StdAfx.h`
- `Code\Tests\Gem_nameTest.cpp`

6. Rebuild your project:
   - If your Gem is a **Code & Assets** Gem, you must rebuild your project (p. 1062).
   - If your Gem is an **Assets Only** Gem, you do not need to rebuild your project.

Creating a Gem from the Command Line

You can also use the command line to create a Gem. You can do this in either the command line or the Project Configurator (p. 1062).

If you add a **Code & Assets** Gem, you must rebuild your project (p. 1062) after adding the Gem. **Asset Only** Gems do not require a rebuild.

To create a Gem with the command line

1. Open a command line and navigate to the *`lumberyard_version`*`\dev\Tools\LmbrSetup\Win\` directory.
2. Type one of the following to create:
   - **Code & Asset** Gem: **lmbr gems create *`MyNewGem`***
   - **Asset Only** Gem: **lmbr gems create *`MyNewGem`* -asset-only**
3. Close the command line.

   The following directories and files are created in `/dev/Gems/Gem_name/` for your Gem:

   - `Assets` – Directory of assets for your Gem, such as materials, models, textures, and audio files. The AZ::IO system automatically includes this directory so that you can reference assets provided by your Gem. For example, if you have a material file, `/Assets/materials/MyMaterial.mtl`, you can reference it in your code with the path `materials/MyMaterial.mtl`.

     Note
     If this directory is not present, you can manually add it. If you don't have any assets in your Gem, you can remove this folder.
   - `Gem.json` – Metadata for this Gem. Do not modify the `Uuid` or `Name` fields. You can specify the following Gem metadata fields:
     - `Version` – *`n.n.n`* (numerical values only).

       Gem revision is not supported, so you should not plan to increment the revision after shipping the Gem.
     - `DisplayName` – Friendly display name. Can contain spaces.
     - `Tags` – Searchable tags that are displayed in Project Configurator under the Gem's summary. Enter this as a comma-separated list of strings.
     - `IconPath` – Path to the Gem's display icon.
     - `Summary` – Detailed description of the Gem. This parameter supports hyperlinks. To specify a hyperlink, use the following syntax:

```
"Summary": "<a href=\"http:\/\/www.amazon.com\">Amazon<\/a>",
```

- `Dependencies` – You can specify a dependency on another Gem or on an engine version. Edit the `_comment` metadata.

```
"Dependencies": [
   {
       "Uuid": "540faf970c994668b5d02c66a39c6625",
       "VersionConstraints": [
           ">1"
       ],
       "_comment": "zzzTestVer001"
   }
],
```

- `preview.png` – Preview image displayed in the **Project Configurator** Gem's list.
- `Code\Gem_name.waf_files` – Waf files json for your Gem that specifies which files are built, how files should be combined into uber files, and how files are filtered in the Visual Studio Project.
- `Code\Gem_name_test.waf_files` – Additional files to include when building with the test specification.
- `Code\wscript` – Waf wscript Python file that defines libraries and other build settings used by your Gem.
- `Code\Include\Gem_name\Gem_name_Bus.h` – Globally visible header that can be included by any project that uses your Gem, or by other Gems that depend on it. External code can call into your Gem's module and receive events from your module through public event buses. Event buses allow simple and safe function calls between different modules of code.
- `Code\Source\Gem_nameModule.cpp` – Implementation of the `Gem_nameModule`. Starting in Lumberyard 1.5, new Gems are built around AZ modules. The default module registers one system component, the `Gem_nameSystemComponent`.
- `Code\Source\Gem_nameSystemComponent.h` – Header for the `Gem_nameSystemComponent` implementation. This component is registered by the `Gem_nameModule` and is a handler for the `Gem_nameBus`.
- `Code\Source\Gem_nameSystemComponent.cpp` – `Gem_nameSystemComponent` implementation. Inside this component are the typical `Activate`, `Deactivate`, `Init`, and `Reflect` methods.
- `Code\Source\StdAfx.cpp` – `Code\Source\StdAfx.h`
- `Code\Tests\Gem_nameTest.cpp`

4. If your Gem is an 'Assets Only' Gem, you are done.

   If your Gem is a 'Code & Assets' Gem, you must rebuild your project (p. 1062).

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. 28) or `Lmbr.exe` (p. 1208). A gems package contains assets, code, `gem.json` file, and an icon file. For a list of available gems, see Gems (p. 1060).

All Lumberyard gems are located in the *`lumberyard_version`*`\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. 1797).

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
    "Uuid": "4c28210b23544635aa15be668dbff15d",             // Required
    "Name": "LightningArc",                                 // Required
    "DisplayName": "Lightning Arc",                         // Optional, defaults to [Name]
    "LumberyardVersion": ["==1.9.0.0"],                     // Optional
    "Version": "1.0.0",                                     // Required
    "LinkType": "Dynamic",                                  // Required
    "Summary": "Provides an entity that can be used to
     produce electricity effects.",                         // Optional, defaults to ""
    "Tags": ["Weather", "Weather Effects","Sky"],           // Optional, defaults to []
    "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: 2.0.0 (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.
- **Uuid** – The unique ID of the gem. Used to identify the gem to the engine.
- **Name** – The name of the gem.
- **DisplayName** – The friendly name of the gem, used in the UI.
- **LumberyardVersion** – Its values can be either:
  - An array of strings (for example, ["~>1.2.345.6789", "~>1.1.1", ">= 1.0"])
  - A single string describing the constraint (for example, "==1.2.3").

  For more examples of acceptable version specifiers, see the list under **Dependencies**.
- **Version** – The API version of the gem. The version should follow the Semantic Versioning specification.
- **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. A gem with the `NoCode` link type can be added to a project without having to rebuild.
- **Summary**– A short description of the gem.
- **Tags** – A list of tags describing the gem.
- **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 Reference

Lumberyard ships with the following gems that are ready to be enabled:

Topics

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. 1071)
- Boids Entity Flow Graph Nodes (p. 1074)

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. 631) 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. • 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 by killed by anything. When invulnerability is not set, the following can kill boids: • 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 `OnBoidHit` function is used.
Mass (kg)	Mass of each individual in the group. Used when physicalizing (p. 361) the boid entity.
Model	3D model file used for the boid representation. You can use geometry files (`*.cgf`, `*.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	

Property Name	Description
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.
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.

Property Name	Description
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 `activate` 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.
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.

Property Name	Description
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. 754), 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. 755).

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** and **Camera Framework** gems are a base upon which you can build more complex camera systems. This gem contains the Camera (p. 467) and the Camera Rig (p. 468), which work together to define a basic camera and its control rig. You can customize the camera rig component through three different behaviors:

- Target acquiring behavior
- Target transform modifying behaviors
- Final camera transform modifying behaviors

ChatPlay Gem

The Twitch ChatPlay Gem 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 establishes a connection to a Twitch channel and uses incoming traffic as game input, like any other input device.

To use Twitch ChatPlay, you use flow graph nodes to connect to your Twitch channel and to listen for key words and then act on those keywords when they're sent by a Twitch user.

Additional Links

- Tutorial: Adding Twitch ChatPlay

Cloud Canvas Cloud Gems

A Cloud Canvas cloud gem includes everything required for you to include AWS cloud-connected functionality in your project. A cloud gem is a package of specific cloud-connected functionality, assets and AWS resource definitions.

Cloud Gems Included with Lumberyard

Lumberyard includes the following cloud gems. You can enable them in a project by using the Lumberyard Project Configurator (p. 28) tool. For more information, visit the following links.

- Dynamic Content – Allows Lumberyard `.pak` files that contain new and updated game assets to be uploaded to the cloud for subsequent automatic download to the game client.
- In-Game Survey – Create surveys for your game, test them, and publish them. View survey results and manage your surveys in the Cloud Gem Portal.
- Leaderboard – Stores a player's high scores and provides "leader boards" that show player rankings.
- Message of the Day – Schedules the delivery of messages (for example, new product announcements or holiday greetings) to game clients.
- Player Account – Provides a standalone player authentication and management solution that uses Amazon Cognito.
- Speech Recognition Cloud Gem Preview – Adds speech recognition and natural language processing to your Lumberyard game using Amazon Lex service.
- Text to Speech Cloud Gem – Turns text into lifelike speech using the Amazon Polly text-to-speech service, which offers dozens of lifelike voices in a variety of languages. Also creates lip synchronization from the text that you provide.

Cloud Gem Portal

A cloud gem creates a cloud gem portal for you that you can use to manage the cloud data for your game. The cloud gem portal is hosted in your AWS account and is a website that you own and can customize. For more information, see Cloud Gem Portal.

AWS services accessed through Cloud Canvas may be subject to separate charges and additional terms. For more information, see Pricing in the Amazon Lumberyard Developer Guide.

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 *Amazon Lumberyard Developer Guide*.

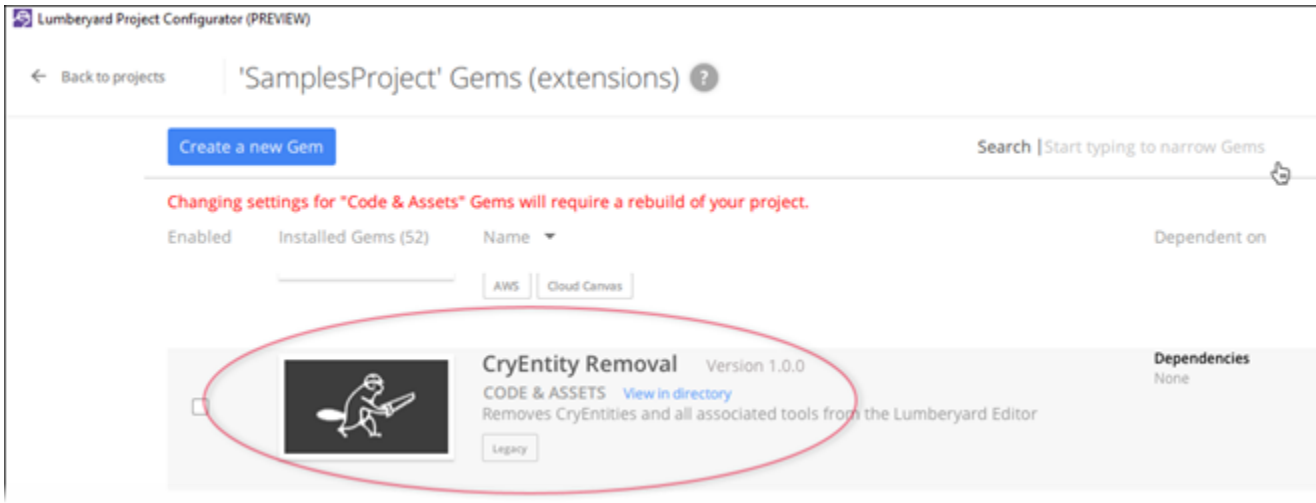
CryEntity Removal Gem

You can use the CryEntity Removal gem to disable all legacy features from Lumberyard Editor. Legacy features appear as *legacy* in Lumberyard Editor and will eventually be removed. This includes the following features:

- Database View (p. 1181) (Legacy)
- Flow Graph (p. 754) (Legacy)
- Object Selector (p. 625) (Legacy)
- Layer Editor (p. 1178) (Legacy)
- Rollup Bar (p. 88) (Legacy)
- Asset Browser (Legacy)

When the gem is enabled, Lumberyard Editor displays only the features and tools that use the new component entity system. For more information, see Component Entity System (p. 437).

By default, the gem is disabled. For more information about enabling gems, see Gems (p. 1060).



Note

- When you enable the gem and open Lumberyard Editor, you are prompted to convert your legacy entities (CryEntities) to the new component entity system. Once an entity is converted, it cannot be converted back to a legacy entity. For more information about converting your entities, see Converting Entities with the Legacy Converter (p. 604).
- If you want to enable the legacy features again, return to the Project Configurator and disable the CryEntity Removal gem. When you reload Lumberyard Editor, the legacy features reappear.

EMotion FX Animation Gem

With the EMotion FX Animation gem, you can use the **EMotion FX Animation Editor** character animation system. The node-based system provides an intuitive approach to controlling character behavior from animation graphs. You can visually build hierarchal state machines and blend trees and use parameter controls for interactive testing. Character motion is controlled through motion extraction.

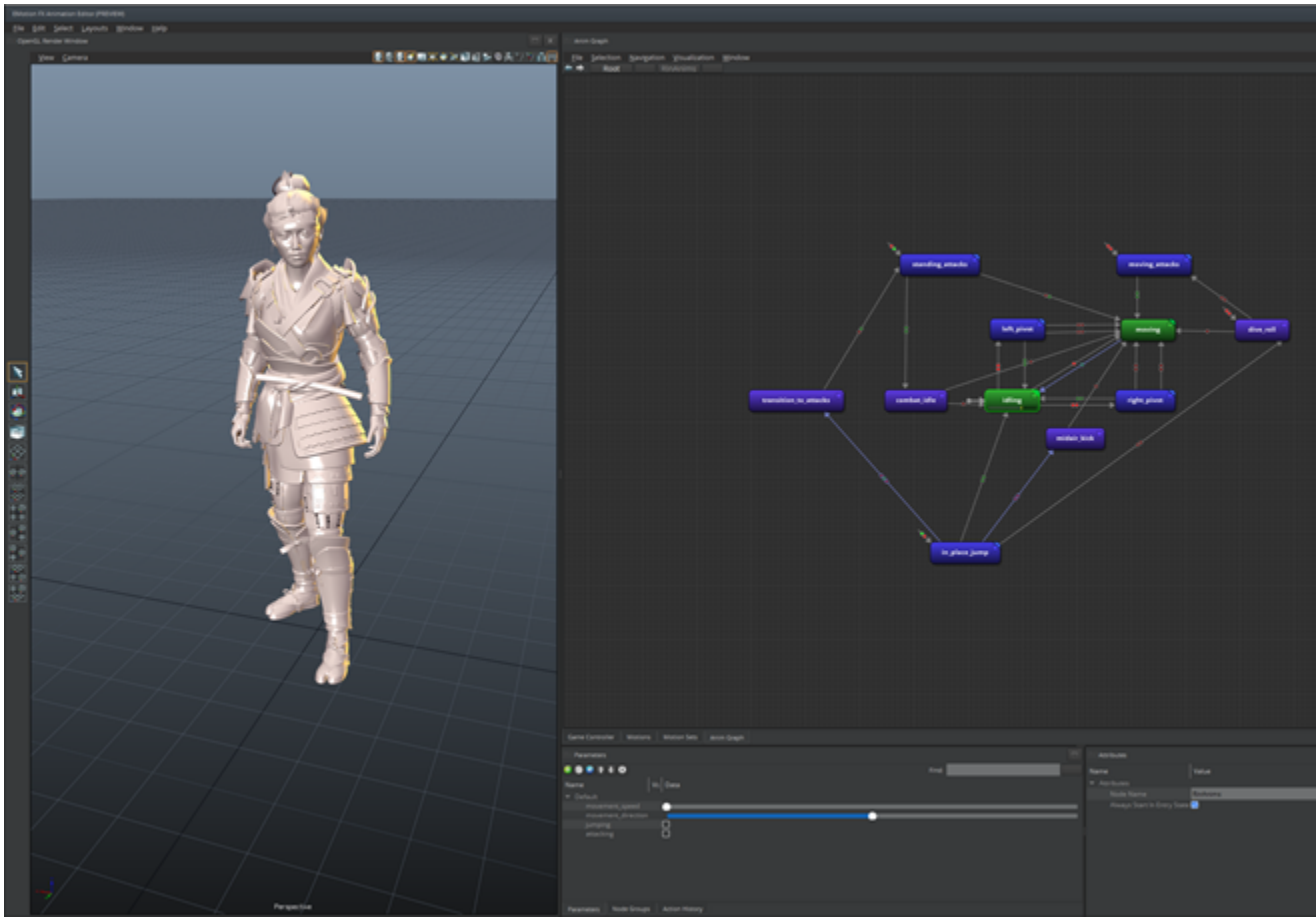
The character animation system includes tools for building the following:

- Transition conditions
- Blend spaces
- Sync tracking
- Motion events
- Mirroring animation
- Controllers

The **Animation Editor** is compatible with `.fbx` files and converts these files to the `.actor` and `.motion` formats.

For more information, see Animation Editor (EMotionFX) (p. 223).

Enable this gem for your project in the Project Configurator. For more information, see Creating a Game Project in the Project Configurator (p. 29).



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. 1100), 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

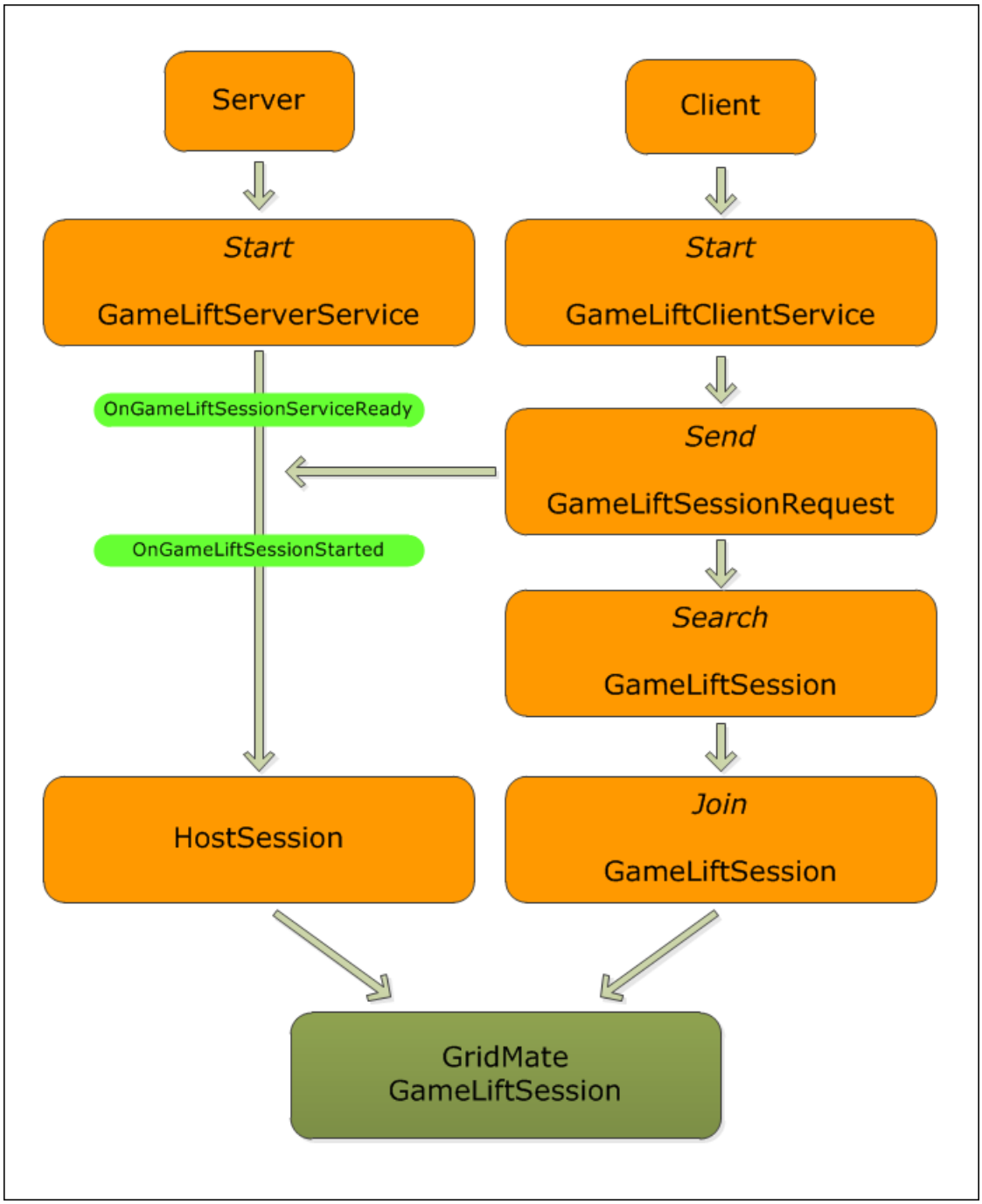
Amazon GameLift is an AWS service for deploying, operating, and scaling session-based multiplayer games. With Amazon GameLift, you can quickly scale high-performance game servers up and down to meet player demand without any additional engineering effort or upfront costs.

The GameLift Gem makes it easy for you to use Amazon GameLift in Lumberyard. To use the Amazon GameLift service, you simply enable the GameLift Gem in your Lumberyard projects.

The GameLift Gem provides two services: On the server side, it provides the `GameLiftServerService`. On the client side, it provides the `GameLiftClientService`. When you use the GameLift Gem in your game project, the following sequence of events occurs:

- The server starts the `GameLiftServerService` and listens for events delivered by the `GameLiftServerService`.
- The client starts the `GameLiftClientService`, sends `GameLiftSessionRequest`, searches `GameLiftSession`, joins `GameLiftSession`, and listens for events delivered by the `GameLiftClientService`.

These events are illustrated in the following diagram.



Sample Code

The GameLift code in this section follows the workflow illustrated in the preceding diagram and is separated into server-side code and client-side code. The code is enabled only when `BUILD_GAMELIFT_SERVER` and `BUILD_GAMELIFT_CLIENT` are defined.

Server-side Code

Use the following sample code as a guide when starting and hosting a GameLift server session.

Start GameLiftServerService

```
GridMate::GameLiftServerServiceDesc serviceDesc;
serviceDesc.m_port = settings.m_serverPort;
if (settings.m_logPath)
{
    serviceDesc.m_logPaths.push_back(settings.m_logPath);
}

m_service = GridMate::StartGridMateService<GridMate::GameLiftServerService>(m_gridMate,
 serviceDesc);
```

Host a Session

```
carrierDesc.m_port = s_gameLiftSettings.m_serverPort;
carrierDesc.m_driverIsFullPackets = false;
carrierDesc.m_driverIsCrossPlatform = true;

EBUS_EVENT_ID_RESULT(session, m_gridMate, GridMate::GameLiftServerServiceBus, HostSession,
 sp, carrierDesc);
```

Client-side Code

Use the following sample code as a guide when using the GameLift client service.

Start GameLiftClientService

```
GridMate::GameLiftClientServiceDesc serviceDesc;
serviceDesc.m_accessKey = settings.m_accessKey;
serviceDesc.m_secretKey = settings.m_secretKey;
serviceDesc.m_fleetId = settings.m_fleetId;
serviceDesc.m_endpoint = settings.m_endpoint;
serviceDesc.m_region = settings.m_region;

m_service = GridMate::StartGridMateService<GridMate::GameLiftClientService>(m_gridMate,
 serviceDesc);
```

Send GameLiftSessionRequest

```
GridMate::GameLiftSessionRequestParams reqParams;
reqParams.m_instanceName = "TestSession";
reqParams.m_numPublicSlots = 16;
reqParams.m_params[0].m_id = "param1";
reqParams.m_params[0].m_value = "value12";
reqParams.m_numParams = 1;
m_sessionRequest = m_service->RequestSession(reqParams);
```

Search Active Sessions

```
EBUS_EVENT_ID_RESULT(m_search, m_gridMate, GridMate::GameLiftClientServiceBus, StartSearch,
 GridMate::GameLiftSearchParams());
```

Join a Session

```
GridMate::CarrierDesc carrierDesc;
carrierDesc.m_port = 33435;
carrierDesc.m_enableDisconnectDetection = true;
carrierDesc.m_connectionTimeoutMS = 10000;
carrierDesc.m_threadUpdateTimeMS = 30;
```

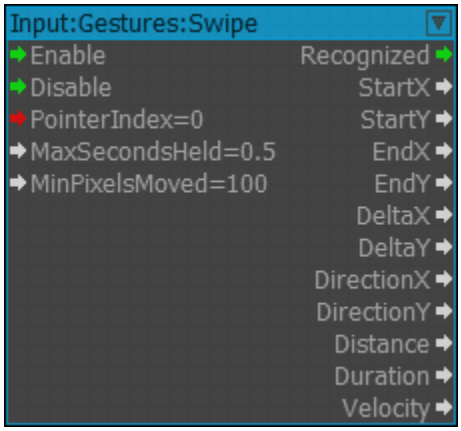
```
const GridMate::GameLiftSearchInfo& gameliftSearchInfo = static_cast<const
 GridMate::GameLiftSearchInfo&>(*gridSearch->GetResult(0));
EBUS_EVENT_ID_RESULT(m_session,m_gridMate,GridMate::GameLiftClientServiceBus,JoinSessionBySearchInfo,ga
```

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 identally 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.

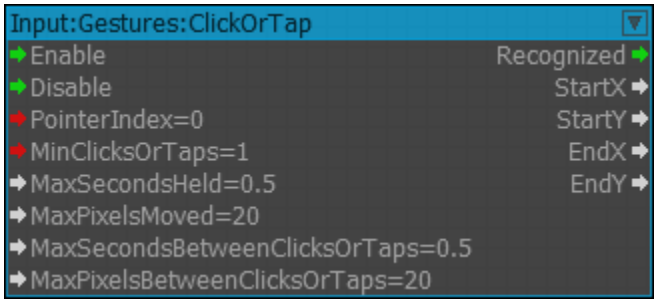
For more information on the Gestures Gem, see Gestures Gem (p. 1082).

Topics

- Drag (p. 1084)
- Hold (p. 1086)
- Pinch (p. 1087)
- Rotate (p. 1089)
- Swipe (p. 1090)

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

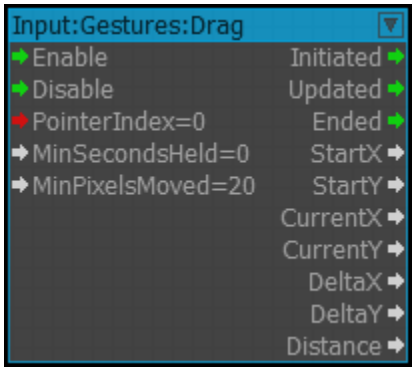
EndY

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 (`CurrentX` – `StartX`).

Type: Float

DeltaY

Y pixels dragged (`CurrentY` – `StartY`).

Type: Float

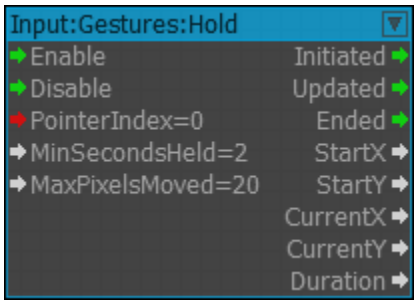
Distance

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

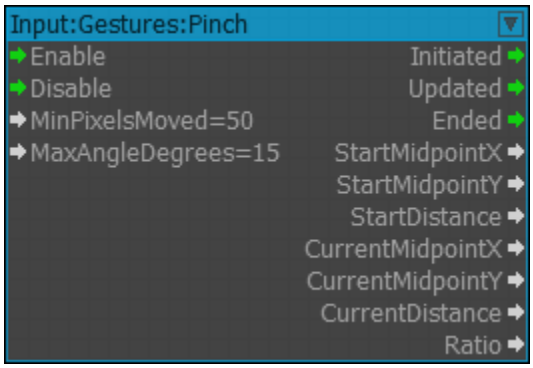
Duration

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

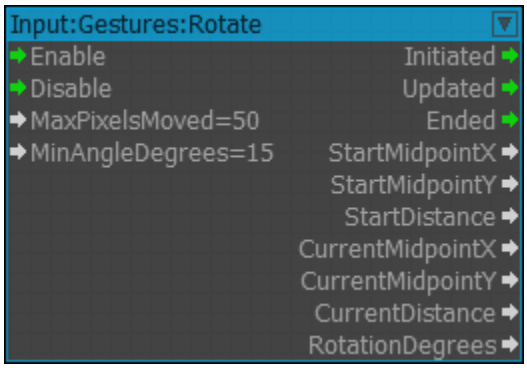
Ratio

The ratio of the pinch (`CurrentDistance` / `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

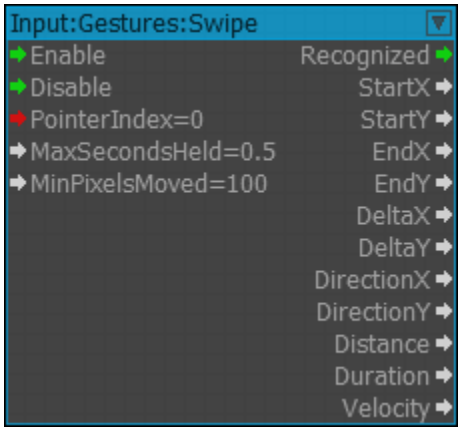
RotationDegrees

The current rotation in degress 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.

Graphics Scripting Gem

The Graphics Scripting Gem enables graphics features in your scripts. With the gem enabled, you can use Script Canvas or Lua to write scripts to control features such as full screen effects, color correction, environment settings, shadow calculations, and more.

To use the Graphics Scripting Gem, you must also enable the Script Canvas Gem. The LmbrCentral Gem, which is also required, is enabled by default and includes material scripting controls and the High Quality Shadow (p. 503) component.

To enable the Graphics Scripting Gem

1. Do one of the following to start the **Project Configurator**:
   - Start Lumberyard Setup Assistant. On the **Summary** page, click **Configure project**.
   - Navigate to the `\lumberyard_version\dev\Tools\LmbrSetup\Win` directory and then start `ProjectConfigurator`.
2. In the **Project Configurator**, choose your project and click **Set as default** to make it the default project that Lumberyard Editor loads.
3. Under your project name, choose **Enable Gems**.
4. On the **Gems (extensions)** page, choose the **Graphics Scripting Gem**.
5. Click **Save**.
6. Do the following to rebuild your project:
   a. In a command line window, change the directory to `\lumberyard_version\dev`. Type `lmbr_waf configure` to configure Lumberyard correctly.
   b. Build the game project. For information, see Game Builds (p. 1847).

HMD Framework Gem

Use the **HMD Framework** Gem to provide the developmental framework for managing VR devices in Lumberyard. You must enable this gem when you enable any other VR Gem (p.), such as NullVR (p. 1132), Oculus, OpenVR, and OSVR.

HttpRequestor Gem

You can use the HttpRequestor Gem to make asynchronous HTTP/HTTPS requests and return data through a user-provided call back function. This gem uses Event Bus (EBus) in the *Amazon Lumberyard Developer Guide* for communication and provides all requests asynchronously.

> **Note**
> This feature is only supported on Windows and replaces `CryAction::HttpRequest`.

Getting Started

In order to use the HttpRequestor Gem, you must enable it in your project.

To enable the HttpRequestor Gem

1. In Project Configurator, find your project in the list, and then click **Enable Gems**.
2. In the list of gems, select the check box next to **HttpRequestor**.
3. Click **Save**.
4. Type the following command to rebuild your project.

```
lmbr_waf configure
```

C++ API Using EBUS_EVENT

The HttpRequestor Gem has separate sets of APIs for adding requests and adding text requests.

AddRequest, AddRequestWithHeaders, AddRequestWithHeadersAndBody

You can use the `AddRequest`, `AddRequestWithHeaders`, and `AddRequestWithHeadersAndBody` APIs to send generic HTTP requests to any website and receive the return data in JSON format. The methods return the data received in the `callback` parameter.

Syntax

```
EBUS_EVENT(HttpRequestor::HttpRequestorRequestBus,AddRequest, URI, method, callback)
```

```
EBUS_EVENT(HttpRequestor::HttpRequestorRequestBus, AddRequestWithHeaders, URI, method,
 headers, callback)
```

```
EBUS_EVENT(HttpRequestor::HttpRequestorRequestBus, AddRequestWithHeadersAndBody, URI,
 method, headers, body, callback)
```

Each of the add request methods requires the URI, a method and a callback.

Parameters

Paramet	Type	Description
`URI`	`AZStd::String`	The fully qualified web address, in the form of `scheme:[//[user:password@]host[:port]][/]path[?query][#fragment]`
`method`	`Aws::Http::HttpMethod`	The method type, only supported values are `HTTP_GET`, `HTTP_POST`, `HTTP_DELETE`, `HTTP_PUT`, `HTTP_HEAD`, and `HTTP_PATCH`.
`callback`		This function is called when the HTTP request has completed. The response body and code are present in the callback.
`headers`	`HttpRequestor::HeadersList`	The list of header fields for the HTTP request.
`body`	`AZStd::String`	Optional body to send with the request.

Return: No return value.

JSON Request Callback

This callback is returned for the `AddRequest`, `AddRequestWithHeaders`, and `AddRequestWithHeadersAndBody` methods.

```
void Callback(const Aws::Utils::Json::JsonValue& json, Aws::Http::HttpResponseCode
 responseCode);
```

Parameters

Parameter	Type	Description
`json`	`Aws::Utils::Json::JsonValue`	The JSON object. The life span of this object is valid only during the scope of the callback.

Parameter	Type	Description
responseCode	Aws::Http::HttpResponseCode	The HTTP response code.

Return: No return value.

AddTextRequest, AddTextRequestWithHeaders, AddTextRequestWithHeadersAndBody

You can use the AddTextRequest, AddTextRequestWithHeaders, and AddTextRequestWithHeadersAndBody APIs to send a generic HTTP request to any website and receive the return data in a text string. The methods return the data received in the callback parameter.

Syntax

```
EBUS_EVENT(HttpRequestor::HttpRequestorRequestBus, AddTextRequest, URI, method, callback)
```

```
EBUS_EVENT(HttpRequestor::HttpRequestorRequestBus, AddTextRequestWithHeaders, URI, method,
 headers, callback)
```

```
EBUS_EVENT(HttpRequestor::HttpRequestorRequestBus, AddTextRequestWithHeadersAndBody, URI,
 method, headers, body, callback)
```

Each of the add text request methods requires the URI, a method and a callback.

Parameters

Paramet	Type	Description
URI	AZStd::String	The fully qualified web address, in the form of scheme:[//[user:password@]host[:port]][/]path[?query][#fragment]
method	Aws::Http::HttpMethod	The method type. Supported values are HTTP_GET, HTTP_POST, HTTP_DELETE, HTTP_PUT, HTTP_HEAD, and HTTP_PATCH.
callback		This function is called when the HTTP request has completed. The response body and code are present in the callback.
headers	HttpRequestor::Headers	The list of header fields for the HTTP request.
body	AZStd::String	Optional body to send with the request.

Return: No return value.

Text Request Callback

This callback is returned for AddTextRequest, AddTextRequestWithHeaders, AddTextRequestWithHeadersAndBody methods.

```
void Callback(const AZStd::string& response, Aws::Http::HttpResponseCode responseCode);
```

Parameters

Parameter	Type	Description
`response`	`AZStd::string&`	The text returned from the server. The life span of this object is valid only during the scope of the callback.
`responseCode`	`Aws::Http::HttpResponseCode`	The HTTP response code.

Return: No return value.

Example

The following example uses the Metastream C++ API to to obtain a gateway IP address.

```
EBUS_EVENT(HttpRequestor::HttpRequestorRequestBus, AddRequest, "https://httpbin.org/ip",
 Aws::Http::HttpMethod::HTTP_GET, [this](AXStd::string && data, Aws::Http::HttpResponseCode
 code)
{
     AZStd::string resultData = std::move(data);
     resultCode = code;

    /* Upon success:
    **  resultCode = Aws::Http::HttpResponceCode::Ok
    **  resultData = "{"origin": "xxx.xxx.xxx.xxx"}"  /* Your IP will be displayed */
    */
 }));
```

In-App Purchases Gem

The In-App Purchases Gem enables you to implement in-app purchases in your Android and iOS games. Platform-specific information is handled by the provided API, allowing you to use a single implementation of the In-App Purchases Gem for both Android and iOS.

> **Note**
> For Android, you must install the Google Play billing library from the Android SDK Manager. You can find this option under the extras section.

Handling Requests for Queries and Purchases

You can access the API through code by connecting to the EBus:

```
EBUS_EVENT(InAppPurchases::InAppPurchasesRequestBus, Initialize);
```

To use the API, include the `InAppPurchasesBus.h` header file.

You can also access the API through the flow graph.

Initialize

Call this method first for all platforms. This method handles all necessary setup before the API can be used.

Parameters: None

Flow Graph node: None

QueryProductInfo

Use this method when the product IDs are shipped with the game. This method looks for a product ID file (`product_ids.json` on Android or `product_ids.plist` on iOS) and retrieves product details using the IDs that are specified in the file.

Parameters: None

Flow Graph node: None

QueryProductInfoByIds

Use this method to retrieve product details if the product IDs are provided at runtime, for example if they are retrieved from a server at runtime.

Parameters: `AZStd::vector<AZStd::string>& productIds`

Flow Graph node: `InAppPurchases:QueryProductInfo`

QueryProductInfoById

Use this method to retrieve product details if the product IDs are provided at runtime, for example if they are retrieved from a server at runtime. Use this method to retrieve details for a single product only.

Parameters: `AZStd::string& productId`

Flow Graph node: `InAppPurchases:QueryProductInfo`

QueryPurchasedProducts

Use this method to query the Google Play Store for products that were already purchased by the signed-in user. On iOS, this method reads the receipt that is stored on the device and lists the items purchased by the signed-in user.

Parameters: None

Flow Graph node: `InAppPurchases:QueryPurchasedProducts`

RestorePurchasedProducts

Use this method to restore purchases that were made by the user. This applies to purchases made on a different device, where the current device does not have receipts stored locally. This method is supported on iOS only.

Parameters: None

Flow Graph node: None

ConsumePurchase

Call this method for all purchases that are consumable, such as virtual currencies, health, and ammo. This method requires the purchase token provided by the Google Play Store when the product was purchased. This method is required and supported on Android only.

Parameters: `AZStd::string& purchaseToken`

Flow Graph node: `InAppPurchases:ConsumePurchase`

FinishTransaction

Call this method at the end of a transaction. This method requires the transaction ID provided by the iOS App Store when the product was purchased. It also accepts a boolean parameter to indicate

whether or not to download content that is hosted on Apple servers. If this method is not called, the transaction will be reported each time the game is restarted. This method is required and supported on iOS only.

Parameters: `AZStd::string& transactionId|bool downloadHostedContent`

Flow Graph node: `InAppPurchases:FinishTransaction`

PurchaseProduct

Use this method to request to purchase a product from the Google Play Store or the iOS App Store. This method requires the product ID of the product being purchased.

Parameters: `AZStd::string& productId`

Flow Graph node: `InAppPurchases:PurchaseProduct`

PurchaseProductWithDeveloperPayload

Use this method to request to purchase a product from the Google Play Store or the iOS App Store. This method requires the product ID of the product being purchased. It accepts an additional parameter for the developer payload, which is used by Android to associate a purchase with a user account. When you request purchased products, you can use the developer payload to determine if the signed-in user made the purchase. On iOS, the user account is used in fraud detection.

Parameters: `AZStd::string& productId|AZStd::string& developerPayload`

Flow Graph node: `InAppPurchases:PurchaseProduct`

GetCachedProductInfo

Use this method to return product details that are stored in a cache each time a user queries for information. The cache only stores product details that were retrieved during the previous call to `QueryProductInfo/QueryProductInfoByIds/QueryProductInfoById`.

Parameters: None

Flow Graph node: None

GetCachedPurchasedProductInfo

Use this method to return details for purchased products that are stored in a cache. The cache only stores details for purchased products that were retrieved during the previous call to `QueryPurchasedProducts`.

Parameters: None

Flow Graph node: None

ClearCachedProductDetails

Use this method to clear the product details that were cached by the previous call to query product details.

Parameters: None

Flow Graph node: None

ClearCachedPurchasedProductDetails

Use this method to clear the details for purchased products that were cached by the previous call to query details for purchased products.

Parameters: None

Flow Graph node: None

Handling Responses to Queries and Purchases

When a user makes a query or a purchase, the API sends the request to Apple or Google servers. Once a response is received, the API broadcasts the response on the `InAppPurchasesResponse` bus.

To handle responses to queries or purchases, overload the functions provided in the class in the `InAppPurchasesResponseBus.h` file.

ProductInfoRetrieved

This method is called when product information is retrieved for all requested products. Product information includes product ID, name, description, price, and more. Depending on the platform, the provided pointers must be cast to the appropriate type (`ProductDetailsAndroid` or `ProductDetailsApple`).

Parameters: `const AZStd::vector<AZStd::unique_ptr<ProductDetails const> >& productDetails`

PurchasedProductsRetrieved

This method is called when details are retrieved for all purchased products. Details for purchased products include product ID, transaction ID, transaction time, and more. Depending on the platform, the provided pointers must be cast to the appropriate type (`PurchasedProductDetailsAndroid` or `PurchasedProductDetailsApple`).

Parameters: `const AZStd::vector<AZStd::unique_ptr<PurchasedProductDetails const> >& purchasedProductDetails`

NewProductPurchased

This method is called when a new product is successfully purchased.

Parameters: `const PurchasedProductDetails* purchasedProductDetails`

PurchaseCancelled

This method is called when a purchase is canceled.

Parameters: `const PurchasedProductDetails* purchasedProductDetails`

PurchaseRefunded

This method is called when a purchase is refunded.

Parameters: `const PurchasedProductDetails* purchasedProductDetails`

PurchaseFailed

This method is called when a purchase fails.

Parameters: `const PurchasedProductDetails* purchasedProductDetails`

HostedContentDownloadComplete

This method is called when content that is hosted on Apple servers is downloaded successfully. The transaction ID and download path are provided.

Parameters: `const AZStd::string& transactionId | AZStd::string& downloadedFileLocation`

HostedContentDownloadFailed

This method is called when content that is hosted on Apple servers fails to download. The transaction ID and content ID of the failed content download are provided.

Parameters: `const AZStd::string& transactionId` | `const AZStd::string& contentId`

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.

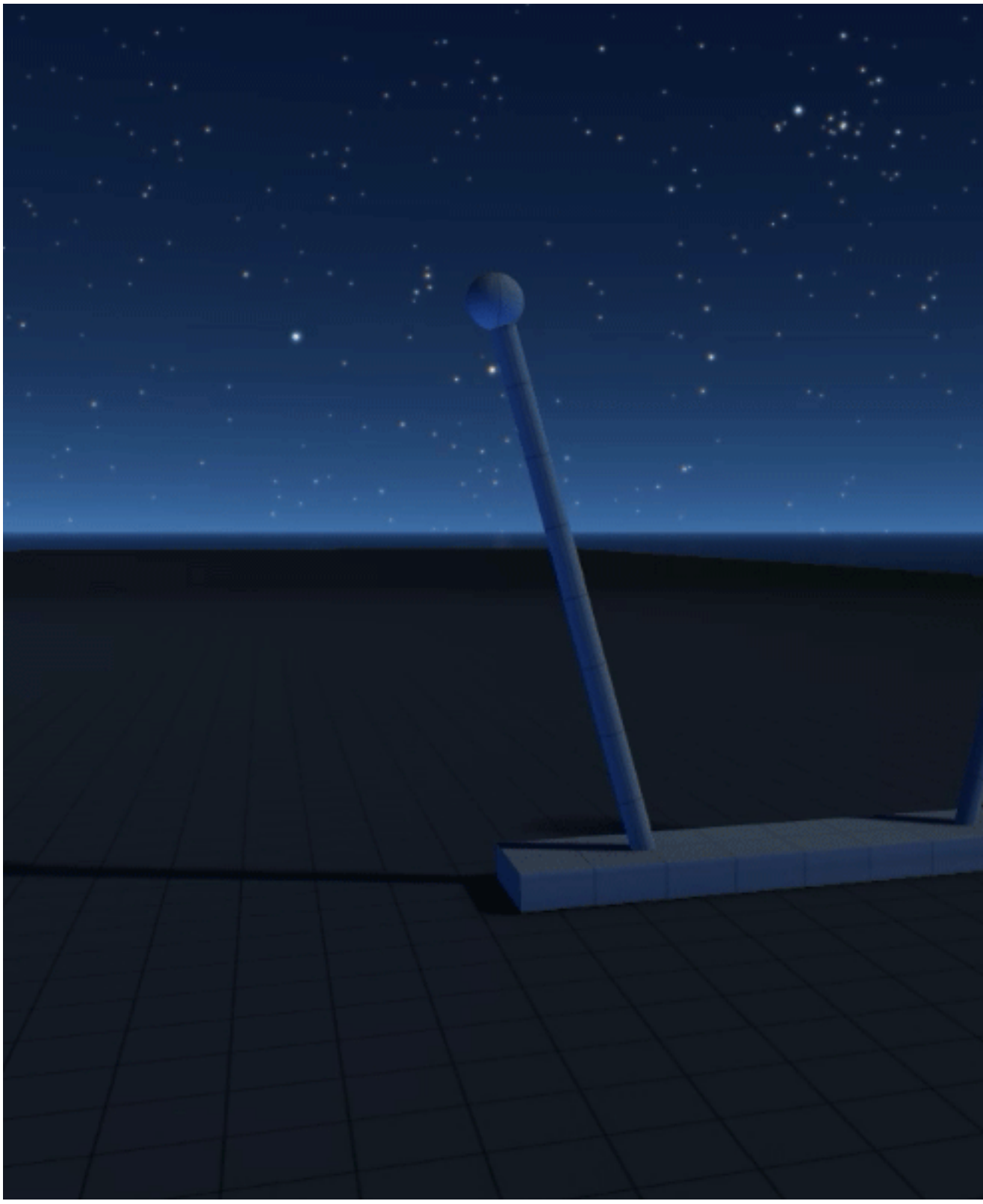
Lightning Arc Gem

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 LightingArc Sample

The LightningArc Sample uses the LightningArc gem to demonstrate the various prescripted arc types.



Topics

Enabling the Lightning Arc Gem

You enable the Lightning Arc Gem from the **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 the **Project Configurator**. For more information, see Using the Project Configurator (p. 28).

To enable the lightning arc Gem

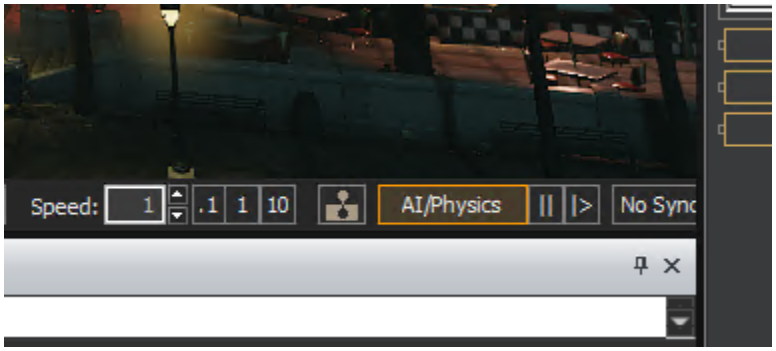
1. Start the **Project Configurator** and click **Enable Packages**.
2. Select **Lightning Arc** and **Game Effect System**.
3. Click **Save**.
4. Use the procedure in Gems (p. 1060) to rebuild your project.

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**, on the **Object** tab, click **Entity**.
2. Under **Browser**, expand **Environment**, and then drag **LightningArc** into your scene.
3. Under **Entity Properties**, ensure that **Active** is selected.
4. Click **AI/Physics** in the bottom toolbar. This makes the lightning arc visible in Lumberyard Editor after you place and link the targets.



5. To place one or more targets, in the **Rollup Bar**'s **Objects** tab, click **AI**. Under **Object Type**, click **Tagpoint**.
6. Move your mouse into the scene, and click to place the tag point where your lightning will arc.
7. To link your tag point, select your lightning arc entity in the scene.
8. 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`.
9. Assign a lightning material: Beneath **Entity**, click in the **Mtl** text box. The **Material Editor** appears.

10. 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. 631) 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 `lightningarceffects.xml` file as explained in Customizing a Lightning Arc Preset (p. 1103).
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
lightningDeviation	The smoothness of the effect in meters.
lightningFuzzyness	The noisiness of the effect in meters.
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: • 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 `strikeNumSegments * strikeNumPoints`. 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 `strikeNumSegments*strikeNumPoint*2.` Since `maxNumStrikes` is the hard limit of potential number of sparks active at any time, the potential number polygons of a given lightning effect is `strikeNumSegments*strikeNumPoint*2*maxNumStrike.` 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.

Parameter	Description
beamTexShift	Rate at which the U coordinate moves in a given direction. While `beamTexTiling` 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.

LyShine Gem

The **LyShine** Gem is Lumberyard's in-game UI system. With this gem, you can create dynamic user interfaces for your games.

The LyShine Gem is required and enabled by default in new Lumberyard projects.

For more information about how to use the LyShine Gem to create game user interfaces, see UI System (p. 1539).

Metastream Gem

Twitch Metastream is a feature that allows broadcasters to customize their streams with overlays of statistics and events from their game session. Using any web authoring tool, such as Dreamweaver or CoffeeCup, broadcasters can create custom HTML5 pages to control the information, graphics, layout, and behavior of each unique overlay. With Metastream, broadcasters can create more polished, interactive viewing experiences on any of their favorite streaming services, similar to what you see in professional Esports and TV broadcasts.

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. Enable the Metastream gem in your project.
2. 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

Enable the Metastream Gem in your project to turn on the local HTTP Metastream server that is included with Lumberyard.

To enable the Metastream Gem

1. Open the **Project Configurator** (located in the `\dev\Tools\LmbrSetup\Win` 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. 1060).

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. You can set the following options for the HTTP server.

`metastream_enabled`

Read-only console variable (CVAR) that describes the current state of the embedded HTTP server. `0` = disabled. `1` = enabled.

`metastream_serverOptions`

Sets the options for the embedded HTTP server. Options are a set of semicolon separated `key=value` pairs.

If a key value requires a semi-colon character ';' or the equal character '=' you may use the `$semi` macro or `$equ`.

The embedded HTTP server is based on CivetWeb. You can find the full list of options that can be set at:

https://github.com/civetweb/civetweb/blob/master/docs/UserManual.md

For security reasons, the following CivetWeb options are ignored:

- `enable_directory_listing`
- `cgi_interpreter`
- `run_as_user`
- `put_delete_auth_file`

Files that you place inside the document root will be served by the HTTP server. See Accessing Data through the HTTP API (p. 1113).

You can also use the following console commands:

`metastream_start`

Starts the embedded HTTP server

`metastream_stop`

Stops the embedded HTTP server

Exposing Data through Metastream

Metastream exposes data through the C++ API or the `Metastream:CacheData` flow graph node.

C++ API

Controlling the HTTP Server

The Metastream Gem uses the following API to start the HTTP server:

```
bool result; // True when the server is successfully started, false when an error occurs.
Metastream::MetastreamRequestBus::BroadcastResult(result,&Metastream::MetastreamRequests::StartHTTPServ
```

Attempting to start the HTTP server when it is already running has no effect, and the call returns true.

The Metastream Gem uses the following API to stop the HTTP server:

```
Metastream::MetastreamRequestBus::Broadcast(&Metastream::MetastreamRequests::StopHTTPServer);
```

If the server is not running, attempting to stop the server has no effect.

Exposing Data

The Metastream Gem uses a simple API to expose in-game data using the EBus system. See Accessing Data through the HTTP API (p. 1113) for these values. Currently, the Metastream API supports UTF8 strings, bools, Vec3, doubles, signed and unsigned 64-bit values. The EBus system requires these calls to be discrete. This API allows you to add values to objects and/or arrays and the cache. This allows for a fully flexible system in exposing data to a Web client.

Add to Cache

To add an object to the root cache, use the following syntax:

```
void MetastreamRequests::AddStringToCache(const char* table, const char* key, const
 char* value)
void MetastreamRequests::AddBoolToCache(const char* table, const char* key, bool value)
void MetastreamRequests::AddVec3ToCache(const char* table, const char* key, const Vec3
 & value)
void MetastreamRequests::AddDoubleToCache(const char* table, const char* key, double value)
void MetastreamRequests::AddUnsigned64ToCache(const char* table, const char* key,
 AZ::u64 value)
void MetastreamRequests::AddSigned64ToCache(const char* table, const char* key,
 AZ::s64 value)
void MetastreamRequests::AddArrayToCache(const char* table, const char* key, const
 char* arrayName)
void MetastreamRequests::AddObjectToCache(const char* table, const char* key, const
 char* objectName)
```

Definitions for the parameters listed:

`table`

Name of the table.

`key`

Name of the key.

`value`

The value to add. If the value exists, it is updated. If the value type is bool, it is represented in the `JSON` doc as `true` or `false`. Signed, unsigned, and double types are represented as JSON numbers; strings are UTF8 and will be escaped, if necessary.

`arrayName`

The name of the array to add. If no array exists, then empty array is added. The array is deleted after it is added to the cache.

`objectName`

The name of the object to add. If no object exists, then a NULL object is added. The object is deleted after it is added to the cache.

None of the above returns any values.

Add to Array

To add an object to an array, use the following syntax:

```
void MetastreamRequests::AddStringToArray(const char* table, const char* arrayName, const
 char* value)
void MetastreamRequests::AddBoolToArray(const char* table, const char* arrayName,
 bool value)
void MetastreamRequests::AddVec3ToArray(const char* table, const char* arrayName, const
 Vec3 & value)
void MetastreamRequests::AddDoubleToArray(const char* table, const char* arrayName,
 double value)
void MetastreamRequests::AddUnsigned64ToArray(const char* table, const char* arrayName,
 AZ::u64 value)
void MetastreamRequests::AddSigned64ToArray(const char* table, const char* arrayName,
 AZ::s64 value)
void MetastreamRequests::AddObjectToArray(const char* table, const char* destArrayName,
 const char* sourceObjectName)
```

Definitions for the parameters listed:

`table`

Name of the table.

`arrayName`

The name of the array to add. If no array exists, then an empty array is added. The array is deleted after it is added to the cache.

`value`

The value to add to the array. If value type is bool, it is represented in the JSON doc as `true` or `false`. Signed, unsigned, and double types are represented as JSON numbers; strings are UTF8 and will be escaped, if necessary.

`arrayName`

The name of the array to add to. If no array exists, one is created.

`destArrayName`

The name of destination array to add to. If no array exists, one is created.

`sourceObjectName`

The name of the object to add to the array. If no object exists, then a NULL object is added. The object is deleted after it is added to the cache.

None of the above returns any values.

Add to Object

To add an object to an object, use the following syntax:

```
void MetastreamRequests::AddArrayToObject(const char* table, const char* destObjectName,
 const char* key, const char* srcArrayName)
void MetastreamRequests::AddObjectToObject(const char* table, const char* destObjectName,
 const char* key, const char* sourceObjectName)
void MetastreamRequests::AddStringToObject(const char* table, const char* objectName, const
 char* key, const char* value)
void MetastreamRequests::AddBoolToObject(const char* table, const char* objectName, const
 char* key, bool value)
void MetastreamRequests::AddVec3ToObject(const char* table, const char* objectName, const
 char* key, const Vec3 & value)
void MetastreamRequests::AddDoubleToObject(const char* table, const char* objectName, const
 char* key, double value)
void MetastreamRequests::AddUnsigned64ToObject(const char* table, const char* objectName,
 const char* key, AZ::u64 value)
void MetastreamRequests::AddSigned64ToObject(const char* table, const char* objectName,
 const char* key, AZ::s64 value)
```

Definitions for the parameters listed:

`table`

Name of the table.

`key`

Name of the key.

`objectName`

The name of the object to add. If no object exists, then one is created.

`value`

The value to add. If the value exists, it is updated. If value type is bool, it is represented in the JSON doc as `true` or `false`. Signed, unsigned, and double types are represented as JSON numbers; strings are UTF8 and will be escaped, if necessary.

`srcArrayName`

The name of the array to add to. If no array exists, one is created. The array is deleted after it is added to the object.

`sourceObjectName`

The name of the object to add. If no object exists, then a NULL object is added. The object is deleted after it is added to the cache.

None of the above returns any values.

Examples

The following example shows how to use the Metastream C++ API in a project:

```
Metastream::MetastreamRequestBus::Broadcast(&Metastream::MetastreamRequestBus::Events::AddToCache,
 table, key, value);
```

Note
Any value that is added to the cache should be JSON compliant. For information, see the JSON RFC.

The following example shows how to reflect the system information. Basically all of the info is added to a object name, `sysInfo`, and this object is then added to the cache as `systeminfo`.

Compound object:

```
Metastream::MetastreamRequestBus::Broadcast(
    &Metastream::MetastreamRequestBus::Events::AddUnsigned64ToObject,
    kDataBaseName.c_str(), "sysInfo",  "drivespace", GetFreeDriveSpace());

Metastream::MetastreamRequestBus::Broadcast(
    &Metastream::MetastreamRequestBus::Events::AddUnsigned64ToObject,
    kDataBaseName.c_str(),  "sysInfo",  "memoryload", GetMemoryLoad());

Metastream::MetastreamRequestBus::Broadcast(
    &Metastream::MetastreamRequestBus::Events::AddDoubleToObject,
    kDataBaseName.c_str(),  "sysInfo",  "cpuloadsystem", GetCPULoadSystem());

Metastream::MetastreamRequestBus::Broadcast(
    &Metastream::MetastreamRequestBus::Events::AddDoubleToObject,
    kDataBaseName.c_str(),  "sysInfo",  "cpuloadprocess", GetCPULoadProcess());

Metastream::MetastreamRequestBus::Broadcast(
    &Metastream::MetastreamRequestBus::Events::AddUnsigned64ToObject,
    kDataBaseName.c_str(),  "sysInfo",  "tickcount", GetTickCount64());

Metastream::MetastreamRequestBus::Broadcast(
    &Metastream::MetastreamRequestBus::Events::AddObjectToCache,
    kDataBaseName.c_str(),  "systeminfo",  "sysInfo");
```

Metastream Lua Bindings

As of Lumberyard 1.10, Twitch Metastream reflects using the behavior context, which enables you to use Metastream through Script Canvas and Lua.

The following methods can be invoked from Lua:

```
-- Controlling the HTTP server:
MetastreamRequestBus.Broadcast.StartHTTPServer();
MetastreamRequestBus.Broadcast.StopHTTPServer();

-- Adding to the root cache directly:
MetastreamRequestBus.Broadcast.AddStringToCache(table, key, value);      -- where value is
 a string
MetastreamRequestBus.Broadcast.AddBoolToCache(table, key, value);        -- where value is
 a bool
MetastreamRequestBus.Broadcast.AddDoubleToCache(table, key, value);      -- where value is
 a double
MetastreamRequestBus.Broadcast.AddUnsigned64ToCache(table, key, value);  -- where value is
 an unsigned 64-bit number
MetastreamRequestBus.Broadcast.AddSigned64ToCache(table, key, value);    -- where value is
 a signed 64-bit number

MetastreamRequestBus.Broadcast.AddArrayToCache(table, key, arrayName);   -- where arrayName
 is the name of a temporary array (see below)
```

```
MetastreamRequestBus.Broadcast.AddObjectToCache(table, key, objectName); -- where
 objectName is the name of a temporary object (see below)

-- Adding to a temporary Array:
MetastreamRequestBus.Broadcast.AddStringToArray(table, arrayName, value);
MetastreamRequestBus.Broadcast.AddBoolToArray(table, arrayName, value);
MetastreamRequestBus.Broadcast.AddDoubleToArray(table, arrayName, value);
MetastreamRequestBus.Broadcast.AddUnsigned64ToArray(table, arrayName, value);
MetastreamRequestBus.Broadcast.AddSigned64ToArray(table, arrayName, value);
MetastreamRequestBus.Broadcast.AddObjectToArray(table, arrayName, objectName);

-- Adding to a temporary Object:
MetastreamRequestBus.Broadcast.AddStringToObject(table, objectName, key, value);
MetastreamRequestBus.Broadcast.AddBoolToObject(table, objectName, key, value);
MetastreamRequestBus.Broadcast.AddDoubleToObject(table, objectName, key, value);
MetastreamRequestBus.Broadcast.AddUnsigned64ToObject(table, objectName, key, value);
MetastreamRequestBus.Broadcast.AddSigned64ToObject(table, objectName, key, value);
MetastreamRequestBus.Broadcast.AddArrayToObject(table, objectName, key, arrayName);
MetastreamRequestBus.Broadcast.AddObjectToObject(table, objectName, key, objectName);
```

Exposing Data through Flow Graph

Metastream:HTTPServer Node

The `Metastream:HTTPServer` node controls the embedded HTTP server:.

Node Inputs

Start

Starts the embedded HTTP server.

Stop

Stops the embedded HTTP server.

Node Outputs

Out

Triggers an output when the server start or stop is finished.

Error

Triggers an output of `true` if an error occurrs.

Type: Bool

Metastream:CacheData Node

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

Triggers an output when the data is 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`

`http://localhost:`*`port`*`/data?table=`*`table_name`*`&key=`*`key_name`*

Returns the value for the specified key in the specified table.

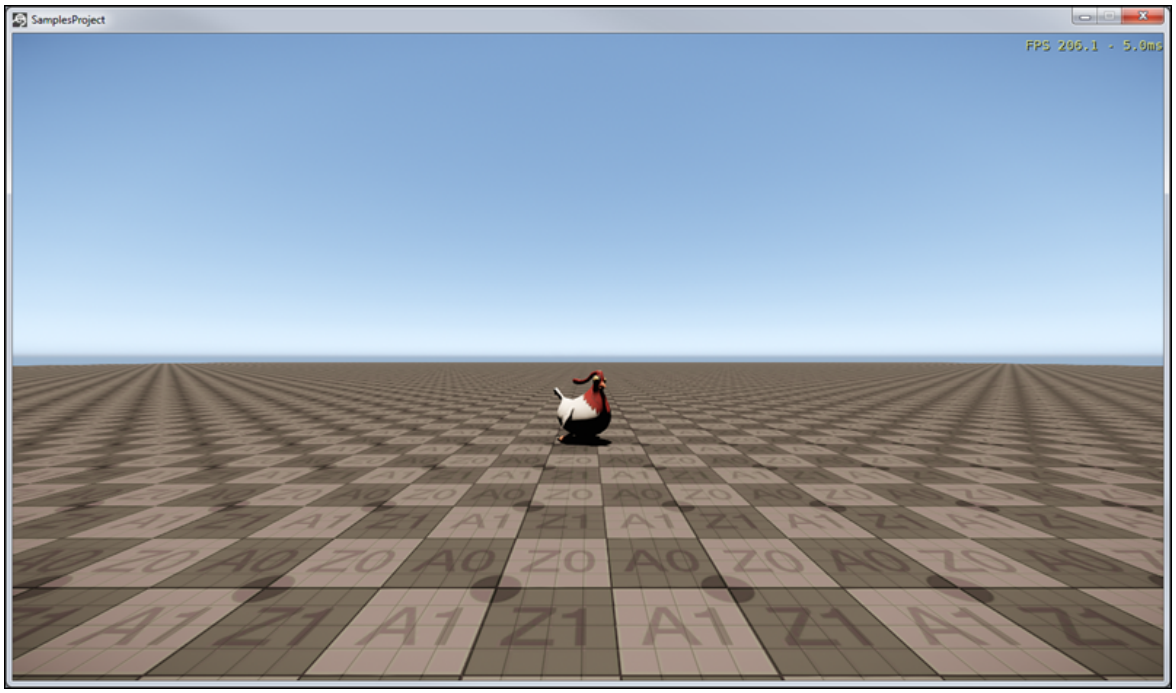
Note
Multiple `key=value` pairs can be retrieved in a single request by listing the desired keys separated by commas. For example, http://localhost:*8082*/data?table=*sample*&key=*key1,key2,key3*.
To list all keys and their values for a table: http://localhost:*8082*/data?table=*sample*&key=*

Data requests are returned in the following format:

Request	Return
`/data`	`{ "tables": [ "table1", "table2", ... ] }`
`/data?table=`*`table_name`*	`{ "keys": [ "key1", "key2", ... ] }`
`/data?table=`*`table_name`*`&key=`*`key_name`*	`{ "key_name": value }`
`/data?table=`*`table_name`*`&key=`*`keys_list`*	`{ "key1": value1, "key2": value2, ... }`

Using the Metastream Sample

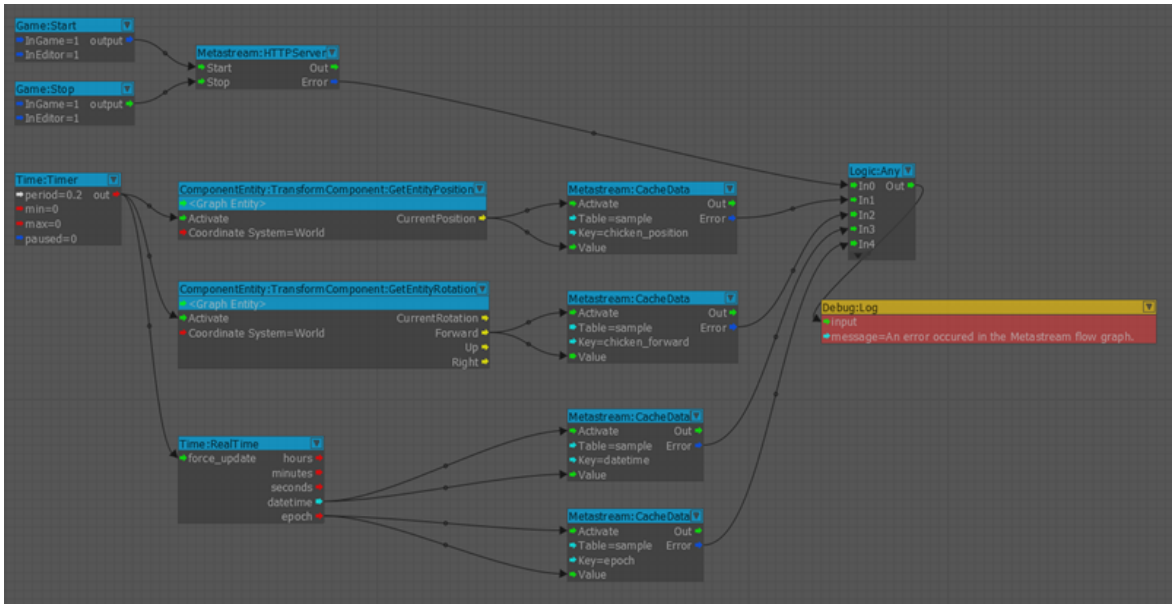
Located in the `\dev\SamplesProject\Levels\Samples\Metastream_Sample` directory, the Metastream sample level demonstrates how to expose data through the flow graph.



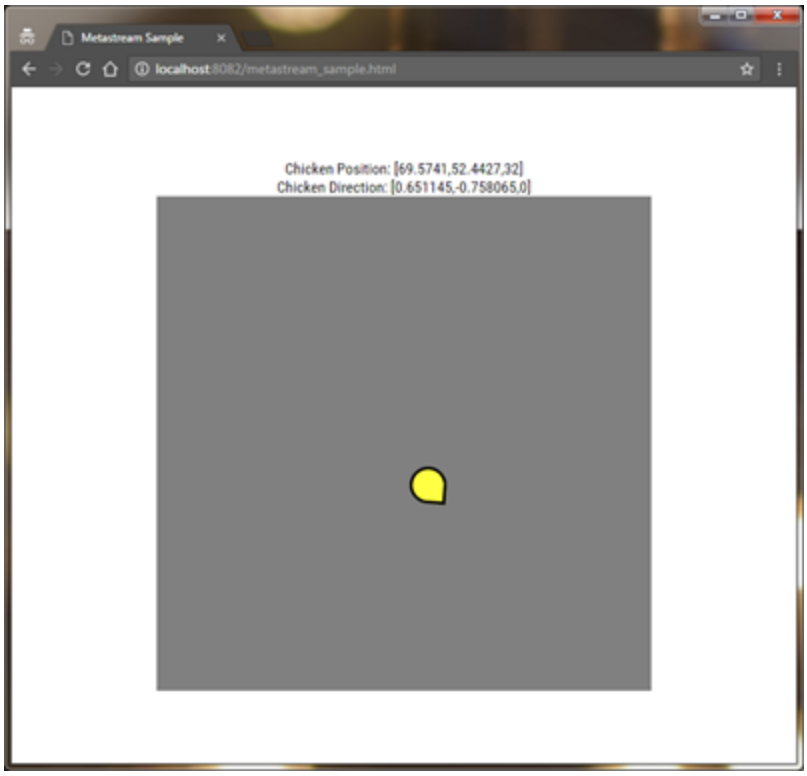
In conjunction with the sample level, the Metastream sample HTML file (located in the `\dev\Gems\Metastream\Files` directory) shows how to use the data to create a simple, dynamic overlay.

These overlays can provide a more engaging experience for viewers without creating visual clutter on a broadcaster's game screen.

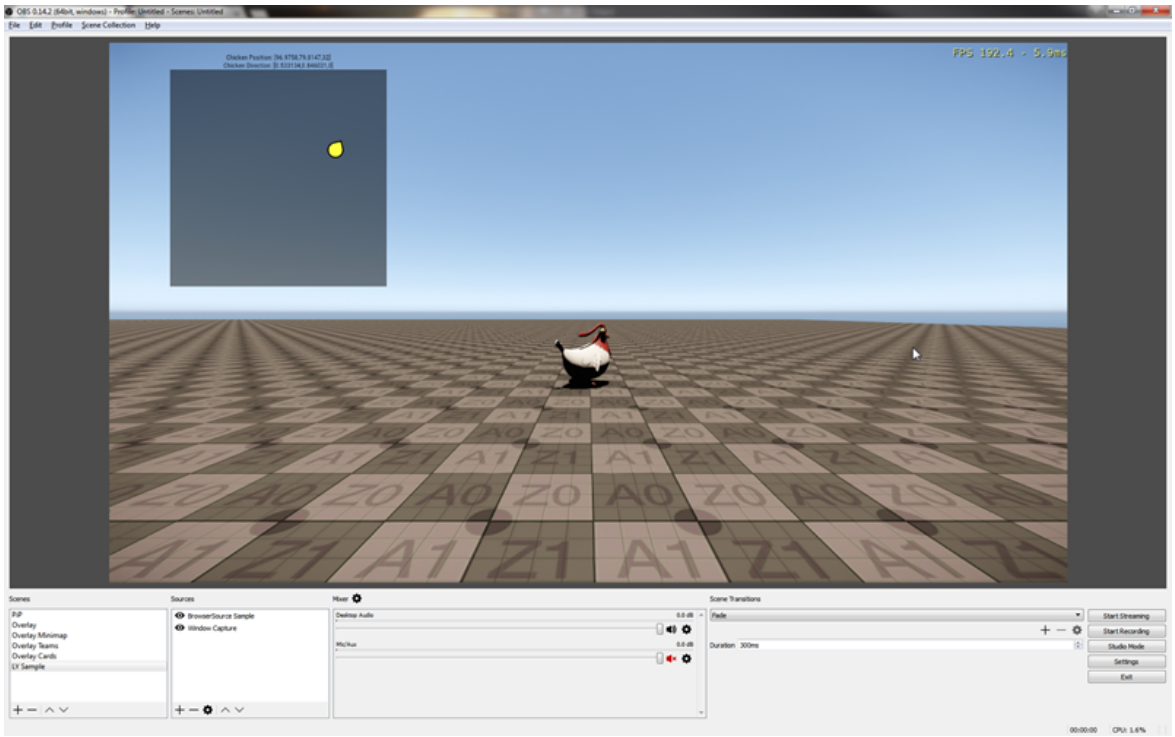
The following is the flow graph from the Metastream sample level.



The sample overlay file `metastream_sample.html`" (located in `dev/Gems/Metastream/Files`) is shown here rendered in Chrome.



The position and direction of the yellow pin update dynamically to represent the position and orientation of the chicken in the sample level. The following example shows the overlay rendered on top of the game stream.



The following example from Amazon Game Studios' game Breakaway shows an overlay with stats from the two leaders in a match.

Microphone Gem

The **Microphone** Gem is a dependency for the Cloud Canvas Speech Recognition Gem (p. 1076).This means that when you enable the Cloud Canvas Speech Recognition Gem in the Project Configurator, the Microphone Gem is also automatically added.

The Microphone Gem's API is located at `\dev\Code\CryEngine\CryCommon\MicrophoneBus.h`.

This gem connects to a hardware recording device and enables you to capture an audio signal. Before running the game or editor, set the default recording device in your Operating System. Check your levels and adjust the gain accordingly. Once set up, the Microphone Gem connects to that device at application startup. Capturing starts when you start a capturing session and stops when you end the capturing session.

Clients using the mic data are expected to consume the data at a pace that meets or exceeds the data rate of the microphone. If consumption rate is too slow, the captured data can lag progressively behind real-time.

When consuming samples, you can specify a desired output format configuration, and the output is automatically converted.

Note the following:

- Only Windows is currently supported.
- Mono and stereo configurations only are supported for microphone input and client output.
- Only linear PCM samples are supported. This can be in either IEEE float (32-bit) or 16-bit signed integer formats. Typically mics will capture float samples.
- Hardware configuration changes, such as changing the default recording device, aren't handled at run time.
- Choosing a device to connect to is not yet supported. The Microphone gem connects only to the default device.
- Only a single consumer should be used. This is because consuming samples from the Microphone gem advances the read position of internal buffers containing the mic data. If two clients attempt to consume the data, each client obtains only portions of the original data.

Multiplayer Gem

The Multiplayer Gem makes it easy for you to add some basic network functionality to your games without digging too deeply into specific details or customization. To make this possible, Lumberyard has created a set of flow graph nodes, a Multiplayer Lobby component, and CVars for general networking use.

You can also use the Multiplayer sample project (p. 1487) for an example of how to build and structure multiplayer games that use the various features of the GridMate networking library.

For more information about networking in Lumberyard, see Networking System in the *Amazon Lumberyard Developer Guide*.

Topics

Flow Graph Nodes

The flow graph nodes in the Multiplayer Gem provide networking features to list and connect to servers that are using a particular service or to host a server on that service. You can use these flow graph nodes to create a simple lobby through which players can search, join, and host an instance of a game.

Multiplayer:Functions:LAN:Host

Attempts to host a server on the `LANSessionService`.

Inputs

Port	Description
Activate	Attempts to host the server.

Port	Description
ServerName	The name of the server. Returned as part of the result from the `LAN:ListServers` call.
MaxPlayers	The maximum number of players allowed to join the server.
Map	The map to load after the session is created on the host. If empty, the host remains on the current map.
Port	The port to use for hosting the game server. If `-1` is specified, the value of the `sv_port` cvar is used.
DisconnectDetection	Enables disconnect detection on the server (that is, whether clients that do not respond within a particular time are dropped from the session). For disconnect detection to be enabled, both the `gm_disconnectDetection` CVar (true by default) and the `DisconnectDetection` value must be true.
ConnectionTimeout	How long a connection must fail to respond before being disconnected from the session. To use this setting, you must enable disconnect detection.

Outputs

Port	Description
Success	A `GridSession` was successfully created.
Failed	An error occurred during the attempt to create the `GridSession`.

Multiplayer:Functions:LAN:ListServers

Uses the `LANSessionService` to perform a `GridSearch`. The results can be displayed or used to connect to a particular server.

Inputs

Port	Description
Activate	Causes the flow graph node to perform a `GridSearch` for active sessions.
MaxResults	Specifies the maximum number of results that should be returned from the search.

Outputs

Port	Description
Success	The search ran successfully.
Failed	An error occurred during the search.
NumResults	The number of search results that were found.
Results	Returns the information about a search result. This output is returned once for each search result.

> **Note**
> This node does not provide any options for configuring encryption. Encryption must be configured before you use this flow graph node.

Multiplayer:Functions:LAN:Connect

Uses the `LANSessionService` to attempt to join the specified server.

Inputs

Port	Description
Activate	Causes the client to attempt to join the selected session.
Server Address	The network address of the server.
Port	The network port to use. If `-1` is specified, the value of the `cl_serverport` CVar is used.
Result	The result generated from `LAN:ListServers`.

Outputs

Port	Description
Success	The client successfully joined the chosen session.
Failed	An error occurred when the client tried to join the chosen session.

> **Note**
> This node does not provide any options for configuring encryption. Encryption must be configured before you use this flow graph node.

Multiplayer:Functions:XBone:Host

Attempts to host a server on the `XBoneSessionService`.

Inputs

Port	Description
Activate	Attempts to host the server.

Port	Description
ServerName	The name of the server. Returned as part of the result from the `XBone:ListServers` call.
MaxPlayers	The maximum number of players that can join the specified server.
Map	The map to load after the session is created on the host. If empty, the host remains on the current map.
Port	The port to use for hosting the game server. If `-1` is specified, the value of the `sv_port` CVar is used.
DisconnectDetection	Enables disconnect detection on the server (that is, whether clients that do not respond within a particular time are dropped from the session). For disconnect detect-ion to be enabled, both the `gm_disconnectDetection` CVar (true by default) and the `DisconnectDetection` value must be true.
ConnectionTimeout	How long a connection must fail to respond before being disconnected from the session. To use this setting, you must enable disconnect detection.

Outputs

Port	Description
Success	Activated when a `GridSession` is successfully created.
Failed	Activated when an error is encountered when an attempt is made to create the `GridSession`.

Multiplayer:Functions:XBone:ListServers

Uses the `XBoneSessionService` to perform a `GridSearch`. The results can be displayed or used to connect to a particular server.

Inputs

Port	Description
Activate	Causes the flow graph node to perform a `GridSearch` for active sessions.
MaxResults	Specifies the maximum number of results that should be returned from the search.

Outputs

Port	Description
Success	The search ran successfully.
Failed	The search failed to run.
NumResults	The number of search results that were found.
Results	Returns the information about a search result. This output is returned once for each search result.

> **Note**
> This node does not provide any options for configuring encryption. Encryption must be configured before you use this flow graph node.

Multiplayer:Functions:XBone:Connect

Attempts to join the given specified server using the `XBoneSessionService`.

Inputs

Port	Description
Activate	Causes the client to attempt to join the selected session.
Server Address	The network address of the server.
Port	The network port to use. If `-1` is specified, the value of the `cl_serverport` CVar is used.
Result	The result generated from `XBone:ListServers`.

Outputs

Port	Description
Success	Activated when the client successfully joins the chosen session.
Failed	Activated when the client encounters an error when attempting to join the chosen session.

> **Note**
> This node does not provide any options for configuring encryption. Encryption must be configured before you use this flow graph node.

Multiplayer:Functions:PSN:Host

Attempts to host a server on the `PSNSessionService`

Inputs

Port	Description
Activate	Attempts to host the sever.
ServerName	The name of the server. Returned as part of the result from the `PSN:ListServers` call.
MaxPlayers	The maximum number of players that can join the server.
Map	The map to load after the session is created on the host. If empty, the host remains on the current map.
Port	The port to use for hosting the game server. If `-1` is specified, the value of the `sv_port` cvar is used.
DisconnectDetection	Enables disconnect detection on the server (that is, whether clients that do not respond within a particular time are dropped from the session). For disconnect detection to be enabled, both the `gm_disconnectDetection` CVar (true by default) and the `DisconnectDetection` value must be true.
ConnectionTimeout	How long a connection must fail to respond before being disconnected from the session. To use this setting, you must enable disconnect detection.

Outputs

Port	Description
Success	Activated when a `GridSession` is successfully created.
Failed	Activated when an error is encountered when an attempt is made to create the `GridSession`.

Multiplayer:Functions:PSN:ListServers

Uses the `PSNSessionService` to perform a `GridSearch`. The results can be displayed or used to connect to a particular server.

Inputs

Port	Description
Activate	Causes the flow graph node to perform a `GridSearch` for active sessions.
MaxResults	Specifies the maximum number of results that should be returned from the search.

Outputs

Port	Description
Success	The search ran successfully.
Failed	The search failed to run.
NumResults	The number of search results that were found.
Results	Returns the information about a search result. This output is returned once for each search result.

> **Note**
> This node does not provide any options for configuring encryption. Encryption must be configured before you use this flow graph node.

Multiplayer:Functions:PSN:Connect

Uses the `PSNSessionService` to attempts to join the specified server.

Inputs

Port	Description
Activate	Causes the client to attempt to join the selected session.
Server Address	The network address of the server.
Port	The network port to use. If `-1` is specified, the value of the `cl_serverport` CVar is used.
Result	The result generated from `PSN:ListServers`

Outputs

Port	Description
Success	Activated when the client successfully joins the chosen session.
Failed	Activated when the client encounters an error when attempting to join the chosen session.

> **Note**
> This node does not provide any options for configuring encryption. Encryption must be configured before you use this flow graph node.

Multiplayer:Gamelift:Start

Starts the `GameLiftSessionService` using the supplied parameters. The `GameLiftSessionService` must be started before the GameLift service can be queried or return results.

Inputs

Port	Description
AWSAccessKey	The access key to use with AWS.
AWSSecretKey	The secret key to use with AWS.
AWSRegion	The region to use with AWS.
GameLiftEndpoint	The GameLift endpoint. This parameter is optional.
FleetID	The fleet ID on AWS.
AliasID	The alias ID to use on AWS. Required if no `FleetID` is supplied.
PlayerID	The player ID to use on AWS. This parameter is optional.

Outputs

Port	Description
Success	The `GameliftSessionService` started successfully.
Failure	An error occurred when attempting to start the `GameLiftSessionService`.

Multiplayer:Gamelift:CreateGameSession

Creates a GameLift session using the `GameLiftSessionService`.

Inputs

Port	Description
Activate	Creates the session.
ServerName	The server name of the GameLift server that is instantiated. The server name is returned in the `GameLift:ListServers` call.
Map	The map that the server should be loaded into.
MaxPlayers	The maximum number of players allowed to join the session.

Outputs

Port	Description
Success	A session was successfully created.
Failed	An error occurred during the attempt to create a session.

Port	Description
Results	The session that was created, if session creation was successful.

Multiplayer:Gamelift:ListServers

Uses the `GameLiftSessionService` to perform a `GridSearch` and return a list of the `GridSessions` it was able to find.

Inputs

Port	Description
Activate	Starts the search.
MaxResults	Specifies the maximum number of results that should be returned from the search.

Outputs

Port	Description
Success	The search ran successfully.
Failure	An error occurred during the search.
NumResults	The number of search results that were found.
Results	The results that were found. This port activates once per result.

Multiplayer:Functions:GameLift:Connect

Attempts to connect to the specified GameLift session.

Inputs

Port	Description
Server Address	The network address of the server.
Port	The network port to use. If `-1` is specified, the value of the `cl_serverport` CVar is used.
Result	The result generated from `GameLift:ListServers`.

Outputs

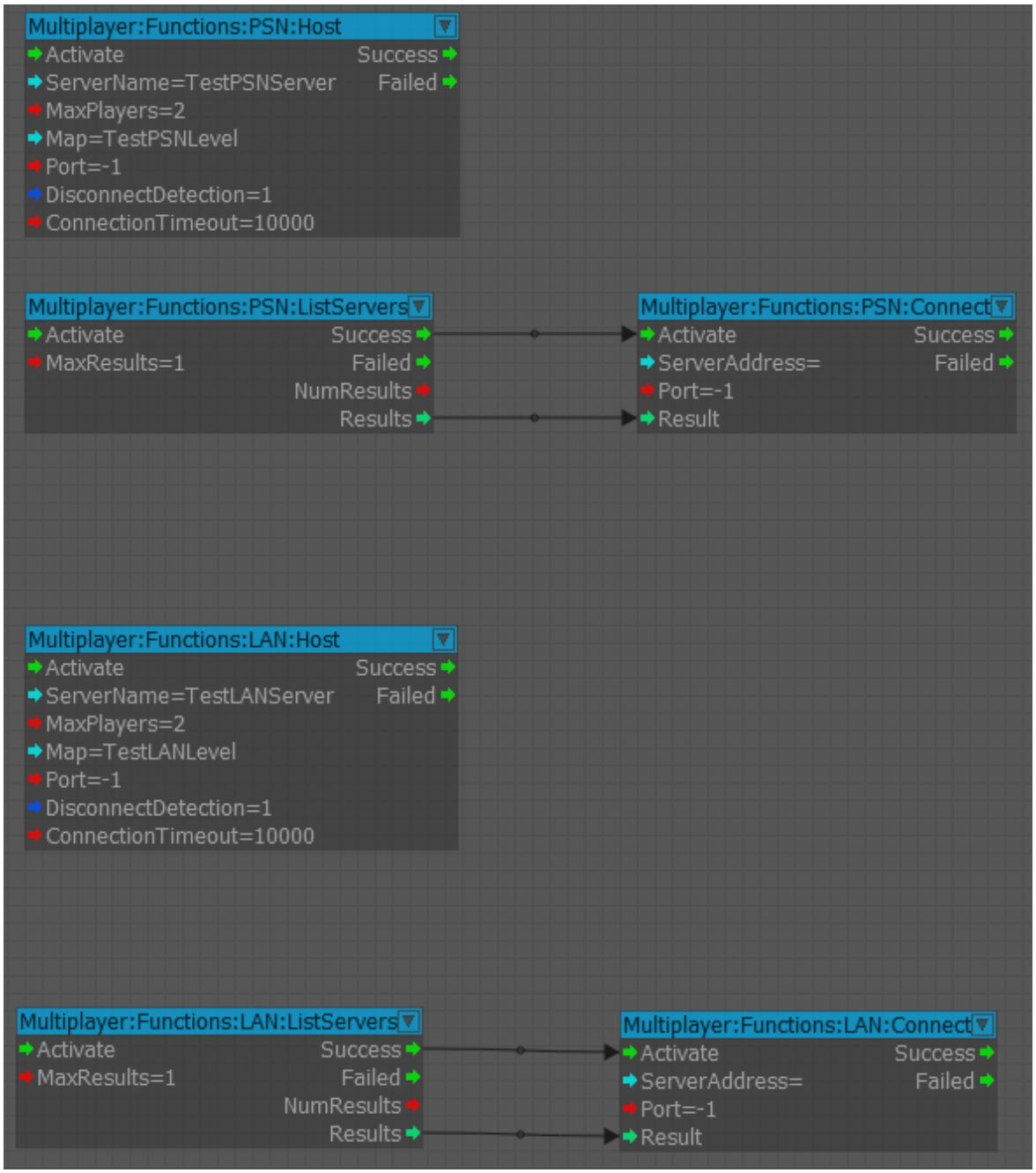
Port	Description
Success	The client successfully joined the chosen session.
Failed	An error occurred when the client tried to join the chosen session.

Note
If the GameLift service has not started, the flow graph node attempts to start it by using the configuration values specified in the following GameLift CVars.

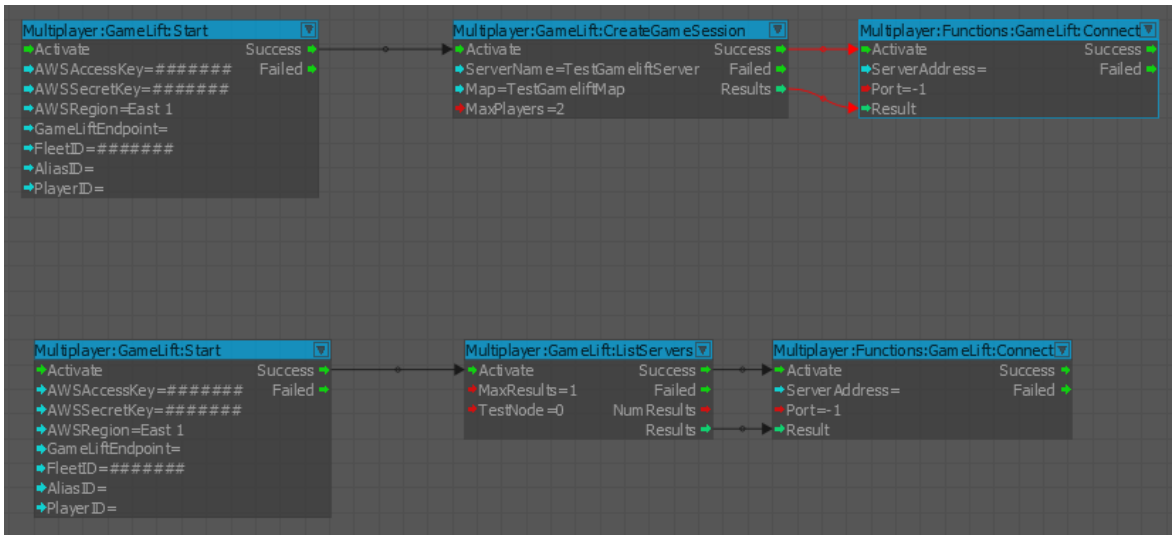
- `gamelift_aws_region`
- `gamelift_aws_access_key`
- `gamelift_aws_secret_key`
- `gamelift_fleet_id`

Examples

The following example shows how to use flow graph nodes to create and join PSN and LAN sessions.



The following example shows how to use flow graph nodes to create and join a session using GameLift.



The Multiplayer Lobby Component

In addition using the flow graph nodes to create a simple lobby, you can use the self-contained lobby from Lumberyard's component entity system. You can add the `MultiplayerLobbyComponent` to a component entity in a scene. The `MultiplayerLobbyComponent` provides a basic lobby that can perform the following tasks:

- Search for an active game session
- Create a visual list of game sessions
- Join a particular game session
- Create a game session
- Name a game session
- Determine the map to load into
- Report errors

Supported Session Services

The `MultiplayerLobbyComponent` currently supports the following session services:

- `LANSessionService`
- `PSNSessionService`
- `XBoneSessionService`
- `GameLiftSessionService`

Configuration Settings

The `MultiplayerSampleComponent` has configuration settings that you can use to customize the hosted sessions that the component creates.

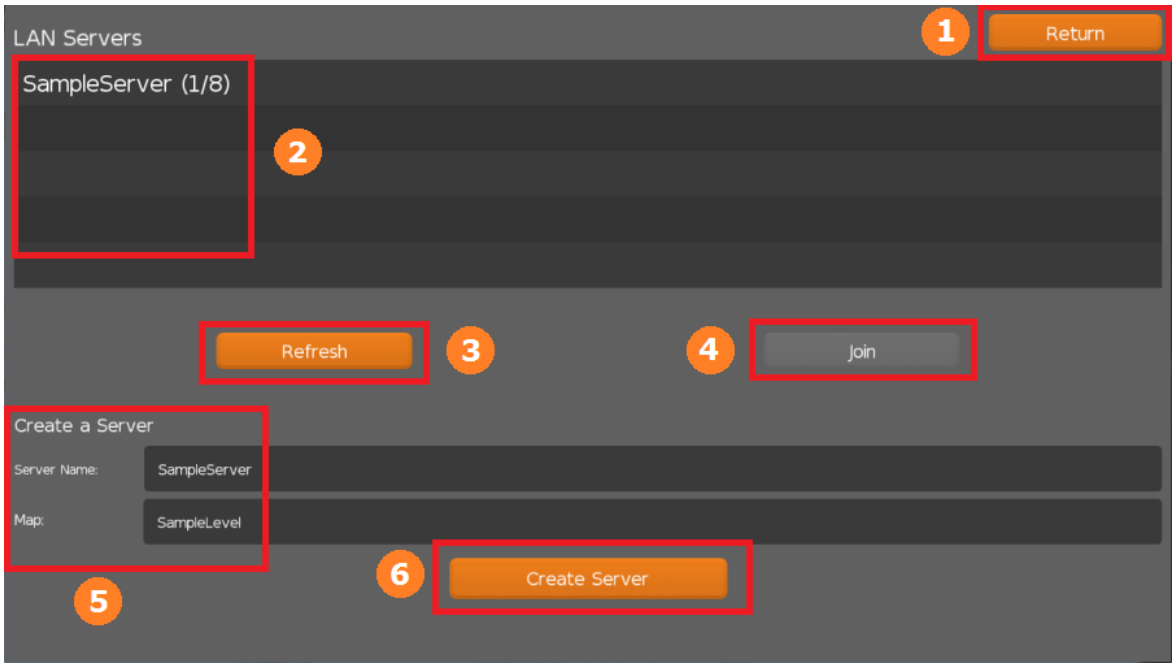
Setting	Description
Max Players	The maximum number of players allowed to join the session.
Port	The port on which the game session operates.

Setting	Description
Enable Disconnect Detection	Enables disconnect detection. If a player's connection does not respond to inquiries from the session host within the specified timeout window, the player is disconnected from the session.
Timeout	The timeout window, in milliseconds, that a client has to respond to inquiries from the session host before being disconnected, if disconnect detection is enabled.
Default Map	The default value for the **Map** field of the display.
Default Server Name	The default value for the **Server Name** field of the display.

> **Note**
> This component does not handle the initialization of encryption, but utilizes encryption if it is already enabled. If you want to use encryption with the component, you must configure encryption before you use the component.

Lobby Description

The following image shows a sample lobby and its details.



1. **Return** – Exits the current lobby and returns to the `SessionService` selection screen.
2. **Session List** – Populates with server results from a successful `GridSearch`. This lists all of the sessions available, the name of each session, and the number of players in each session.
3. **Refresh** –Performs a `GridSearch` on the selected `SessionService` and displays the results in the session list.
4. **Join** – Attempts to join the currently selected session in the session list. If an error occurs, an error message is displayed. If no `GridSession` has been selected, this option is disabled.
5. **Create a Server** – Use these text boxes to configure sessions.
   - **Server Name** – Specifies the name that will be displayed in the session list when `GridSession` instances are searched for and the created `GridSession` is returned.

- **Map** – Use to specify the map to be loaded when the `GridSession` is successfully created.

6. **Create Server** – Attempts to create a `GridSession` in the selected `SessionService`.

Multiplayer Gem CVars

For convenience, the Multiplayer Gem provides the following CVar commands that you can use to test features or configurations.

mphost <port number>

Creates a `GridSession` using the `LANSessionService` on the local machine. The optional port number determines the port on which the grid session runs.

mpjoin <server address> <port number>

Performs a `GridSearch` using the `LANSessionService` and joins the first session returned. The server address and port number are optional parameters that can be used to refine the `GridSearch`.

mpsearch

Performs a `GridSearch` using the `LANSessionService`.

mpdisconnect

Disconnects the local client from its active `GridSession`, if any.

gm_version

Sets the GridMate version number.

gm_netsec_enable

Specifies whether encryption is used for connections. The default is `false`.

> **Note**
> If encryption parameters are not present or invalid, the connection for the session is not encrypted.

gm_netsec_private_key

Specifies the private key file (PEM format) for a secure network connection when `FileDataSource` is used as a certificate manager. This setting overrides any previously configured values in `FileDataSource`. If network encryption is enabled, this value must be set on the host of a `GridSession` when the session is created. This setting is disabled in release builds.

gm_netsec_certificate

Specifies the private key file (PEM format) for a secure network connection when `FileDataSource` is used as a certificate manager. This setting overrides any previously configured values in `FileDataSource`. If network encryption is enabled, this value must be set on the host of a `GridSession` when the session is created. This setting is disabled in release builds.

gm_netsec_ca

Specifies the CA Certificate file (PEM format) for a secure network connection when the `FileDataSoruce` is used as a certificate manager. This setting overrides any previously configured values in `FileDataSource`. This value must be set on the client when trying to join a `GridSession` using encryption. This setting is disabled in release builds.

gm_netsec_verify_client

Enables client verification. If not enabled, only the server is authenticated. Must be set on the host of a `GridSession`. The default is `false`.

gm_disconnectDetection

Sets whether disconnect detection is enabled for a particular `GridSession`. Must be set on the host of `GridSession`. If disconnect detection is enabled, clients that do not respond to inquiries within the timeout window are disconnected. The default is `true`.

gm_disconnectDetectionRttThreshold

A RTT threshold, in milliseconds, that specifies the time after which a connection is dropped when there is no response. The default is `500`.

gm_disconnectDetectionPacketLossThreshold

A value between 0 and 1 that represents the percentage of packets that are permitted to be lost in communication with a particular client. If a client experiences a packet loss greater than the specified value, the connection is dropped. The default is `0.3`.

gm_recvPacketsLimit

The maximum number of packets per second allowed to be received from an existing connection. The default is `0` (no limit).

gm_ipversion

Specifies which Internet protocol version to use. Possible values are `IPv4` or `IPv6`. The default is `IPv4`.

gm_securityData

Specifies the security data for session.

gm_replicasSendTime

Specifies, in milliseconds, the time interval between replica updates. A value of `0` binds the interval to the GridMate tick rate. The default is `0`.

gm_replicasSendLimit

Limits the amount of replica data sent per second. `0` specifies no limit. The default is `0`. This value is disabled in release builds.

gm_burstTimeLimit

Specifies, in seconds, an interval window during which bandwidth capping is not applied. If bandwidth usage continues to exceed the replica send limit outside of this interval, the sent amount is capped at the send limit. The default is `10`. This variable is disabled in release builds.

gamelift_fleet_id

Specifies the GameLift `FleetId` to use for this client when starting `GameLiftSessionService`.

gamelift_aws_access_key

Specifies the AWS access key associated with the GameLift fleet.

gamelift_aws_secret_key

Specifies the AWS secret key associated with the GameLift fleet.

gamelift_aws_region

Specifies the AWS region associated with the GameLift fleet. The default is `us-west-2`.

gamelift_endpoint

Specifies the GameLift service endpoint. The default is `gamelift.us-west-2.amazonaws.com`.

gamelift_alias_id

Specifies the GameLift alias ID to use with the client.

gamelift_player_id

Specifies the GameLift player ID associated with the client.

gamelift_stop_client

Stops the `GameLiftSessionService` and terminates the current `GridSession`, if one is connected.

sv_port *`<local_port>`*

Sets the local UDP port that initializes the socket. The default port is 30090. To use the ephemeral port, set the port to 0. This is useful if you want to connect to a server on the same computer as the client.

NullVR Gem

Use the **NullVR Gem** to run your level through the head-mounted displays (HMD) framework without a connected VR device. This gem is useful for VR graphics debugging.

The **NullVR Gem** has a lower priority than other VR gems and their devices. That means that if any other VR gem is enabled and its appropriate VR device is connected, then the HMD framework uses that VR gem. The **NullVR Gem** is enabled only if no other VR Gem is enabled or no VR device is connected.

When Lumberyard is using the **NullVR Gem**, the following appears in the console:

```
[HMD][Null] - Null Device
```

Console Variables

The NullVR Gem, when enabled, makes certain console variables available. These console variables take effect only if the NullVR Gem is in use; that is, no other VR gems are enabled or no other VR devices are connected.

> **Note**
> The default values in the following list came from Vive at the time of this writing.

`hmd_null_aspectRatio`

Aspect ratio of the null VR implementation.

Default: 0.9

`hmd_null_eyeOffsetX`

Eye X offset (horizontal distance).

Default: -0.0346999988

`hmd_null_fov`

The field of view in radians. Can be updated at runtime.

Default: 1.91986

`hmd_null_fovH`

The field of view height in radians. Used on start.

Default: 1.84633982

`hmd_null_fovV`

The field of view width in radians. Used on start.

Default: 1.94590104

`hmd_null_frustumPlane_horizontalDistance`

Frustum plane horizontal distance.

Default: -0.0701720715

`hmd_null_frustumPlane_verticalDistance`

Frustum plane vertical distance.

Default: -0.00206005573

`hmd_null_renderHeight`

The height of the render texture in pixels. Set once on start.

Default: 1680

`hmd_null_renderWidth`

The width of the render texture in pixels. Set once on start.

Default: 1512

Physics Entities Gem

The Physics Entities 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.

Primitive Assets Gem

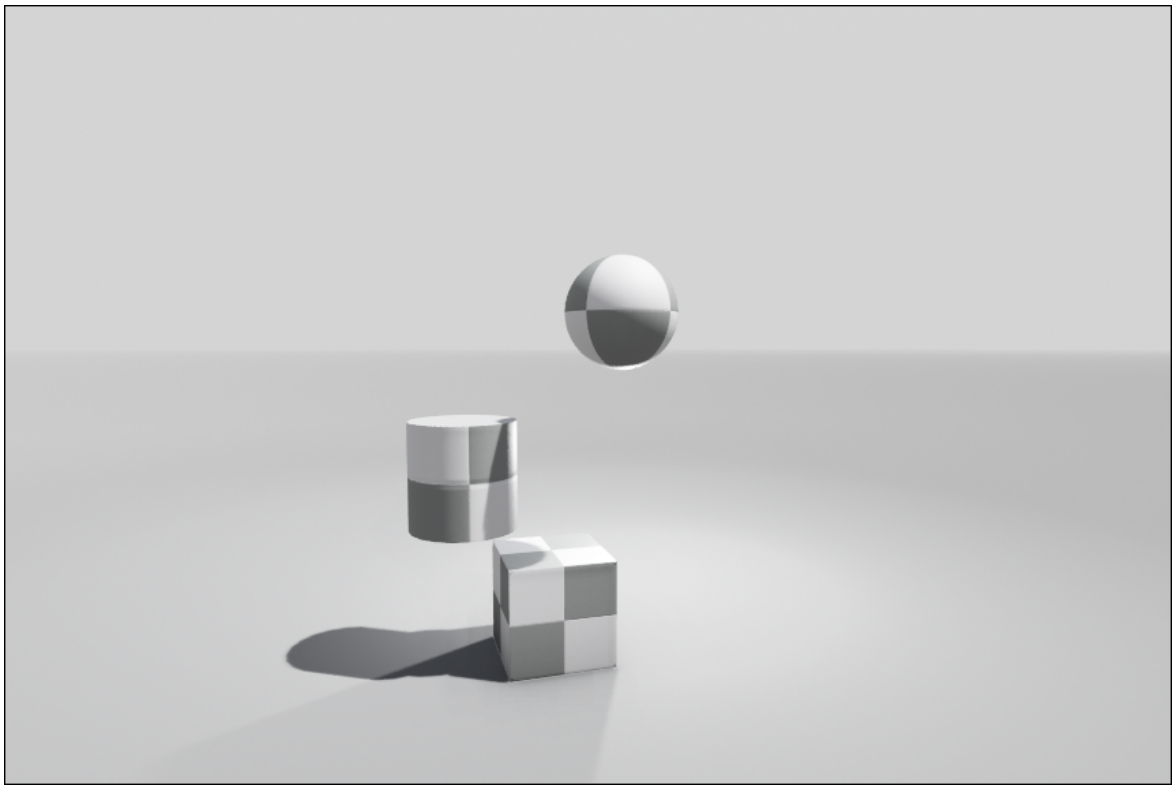
The Primitive Assets Gem provides primitive objects that are physics enabled and unit size (1x1x1), with a gray checkerboard texture. The checkerboard texture is RGB middle gray (127,127,127) and sRGB middle gray (187,187,187). Diffuse, normal, and specular maps are included. The primitive object models have import settings, a material, and a slice. Each slice contains the object mesh and the appropriate **Shape** and **Rigid Body Physics** components. You can use these to add the cube, sphere, and cylinder objects to your level and manipulate the objects, create a placeholder, or test Lumberyard Editor features.

The asset files for the primitive objects are in the `\lumberyard_version\dev\Gems\PrimitiveAssets\Assets` directory.

You can add the primitive objects to your game project by doing either of the following:

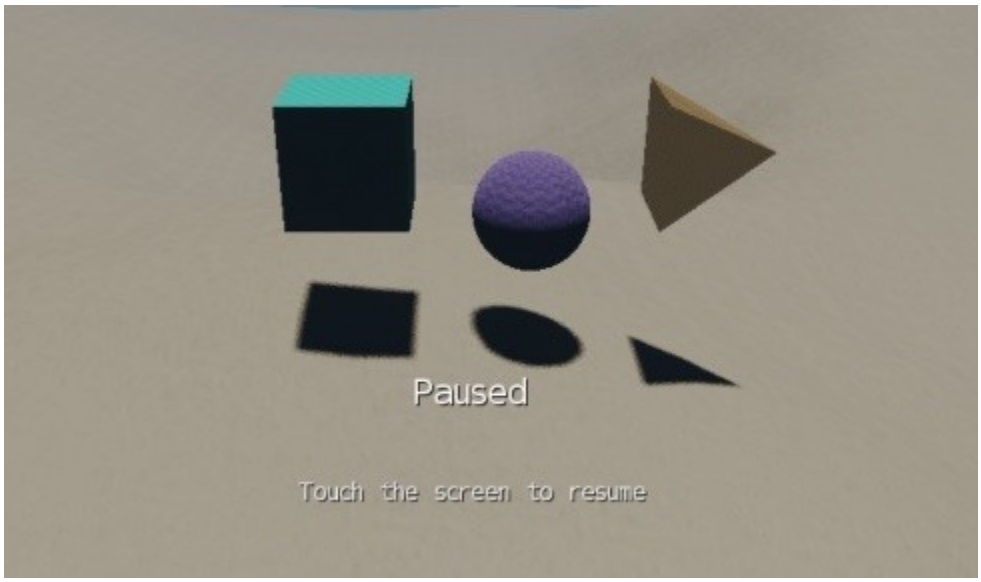
- Create a new project from the **Default** template in the Project Configurator. By default the Primitive Assets Gem is enabled for projects that are created from this template.
- Enable the Primitive Assets Gem for your existing project in the Project Configurator.

For more information, see Creating a Game Project in the Project Configurator (p. 29).



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. 1135)

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()
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();
```

RAD Telemetry Gem

RAD Telemetry 3 is an instrumentation-based profiling and performance visualization middleware product created by RAD Game Tools. The RAD Telemetry Gem provides a RAD Telemetry 3 integration for Lumberyard for those who have licensed RAD Telemetry.

The RAD Telemetry Gem provides one example of how to integrate instrumentation-based profiling middleware. You could write a gem for your own instrumentation-based profiler and leverage Lumberyard's existing performance markers. See the `Profiler.h` file in the `/dev/Code/Framework/`

`AzCore/AzCore/Debug` directory to see how the RAD Telemetry Gem ties into Lumberyard's existing performance markers.

> **Note**
> If the RAD Telemetry Gem is disabled or you do not have the RAD redistributable files for a particular platform, the performance instrumentation compiles to Lumberyard's performance markers.

Another option is to use Lumberyard's own built-in profiler, Statoscope (p. 1516).

To enable the RAD Telemetry Gem

1. Extract the Telemetry redistributables from RAD Game Tools to `/Gems/RADTelemetry/External`. You should see the following subdirectories, which match the subdirectories provided by RAD:

   - `Dll`
   - `Lib`
   - `Include`
   - `Docs`

   > **Note**
   >
   > - If you are compiling for case-sensitive platforms, note that the first letter of each subdirectory listed above is capitalized.
   > - When compiling a project with the RAD Telemetry Gem enabled, the static `lib` file for the target platform must exist in the `Gems\RADTelemetry\External\Lib` directory. The Waf build system then compiles with the `AZ_PROFILE_TELEMETRY` defined globally and links RAD Telemetry libraries for the specified platform. You can add additional platforms by editing the `rad_telemetry.json` file (located in the `Gems\RADTelemetry\3rdParty` directory).

2. Extract RAD Telemetry Tools to the `/Gems/RADTelemetry/Tools` directory. This should include `telemetry.exe` and your license file from RAD.
3. Use the **Project Configurator** to enable (p. 1060) the RAD Telemetry Gem for your project. You must enable this gem (and any other gem) for each project. Gems are not globally enabled.
4. Rebuild your project (p. 1060). You must build using `profile` (preferred) or `debug` (p. 1847) in order to enable RAD Telemetry.

Instrumenting Your Code

With the RAD Telemetry Gem, Lumberyard introduces a set of scoped performance markers:

- `AZ_PROFILE_FUNCTION` – Instruments entire functions. Automatically names the performance event with the function's name.
- `AZ_PROFILE_SCOPE` – Instruments a local scope within a function of interest. You must provide the name.
- `AZ_PROFILE_SCOPE_DYNAMIC` – Instruments a `printf` style format string to dynamically generate a performance event name. Use the dynamic name sparingly because there may be performance overhead of a string copy and transmission over the network.

A scoped performance marker constructs an object that calls a `start` event and calls a `stop` event when it is destroyed. This means that you do not have to worry about early returns. We recommend that you use the `AZ_PROFILE` events when marking up your code, as it allows you to switch to Driller for a different view of profiling data.

Lumberyard also uses the following legacy performance event markers:

- `PROFILE_FUNCTION`
- `PROFILE_FRAME`

If you do not regularly use **Statoscope** (p. 1516) or profile 1, you can compile with legacy performance events disabled (and lower profiling overhead) by commenting out `USE_FRAME_PROFILER` in the `ProjectDefines.h` file.

Lumberyard also includes a performance marker, `FUNCTION_PROFILER_LEGACYONLY`, which does not include an `AZ_PROFILE` performance event. This is useful for removing events that are overly chatty in RAD Telemetry for your title but for which you still want to retain the Statoscope event.

Lumberyard has disabled but not removed performance events around interaction with the graphics device. These performance events are not useful in tracking higher level CPU performance issues and send thousands of events per frame.

Capturing with RAD Telemetry

See the RAD Telemetry documentation for more information. The basic procedure is described below.

To capture with RAD Telemetry

1. Set the console variables for the RAD Telemetry capture using either a configuration file or by typing them into the console at runtime.

   All RAD Telemetry console variables start with `radtm_`. For example, `radtm_Address` and `radtm_Port` if your Telemetry server is on another machine. The console variables are defined in the `RADTelemetryModule.cpp` file. You can augment these console variables to better suit profiling for your title.
2. Use either `telemetry.exe` or the standalone telemetry server to start your RAD Telemetry server.
3. Run `radtm_ToggleEnabled` to begin a capture. You can also set this from the command line when launching your application with `+radtm_ToggleEnabled`.
4. Run `radtm_ToggleEnabled` again from the console or the remote console, or shut down your game process to end the session. As currently instrumented, captures can get large quickly, so you may want to perform focused captures around performance scenarios of interest (under 30 seconds in length).
5. Use the Telemetry visualizer (`telemetry.exe`) to analyze your capture.

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. 1060).

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 frequence 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. 631) 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

Properties	Description
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
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_RainMaxViewDist_Deferred	Sets maximum distance (in meters) at which the deferred rain reflection layer is visible
r_RainOccluderSizeThreshold	Blocks rain for objects bigger than this value

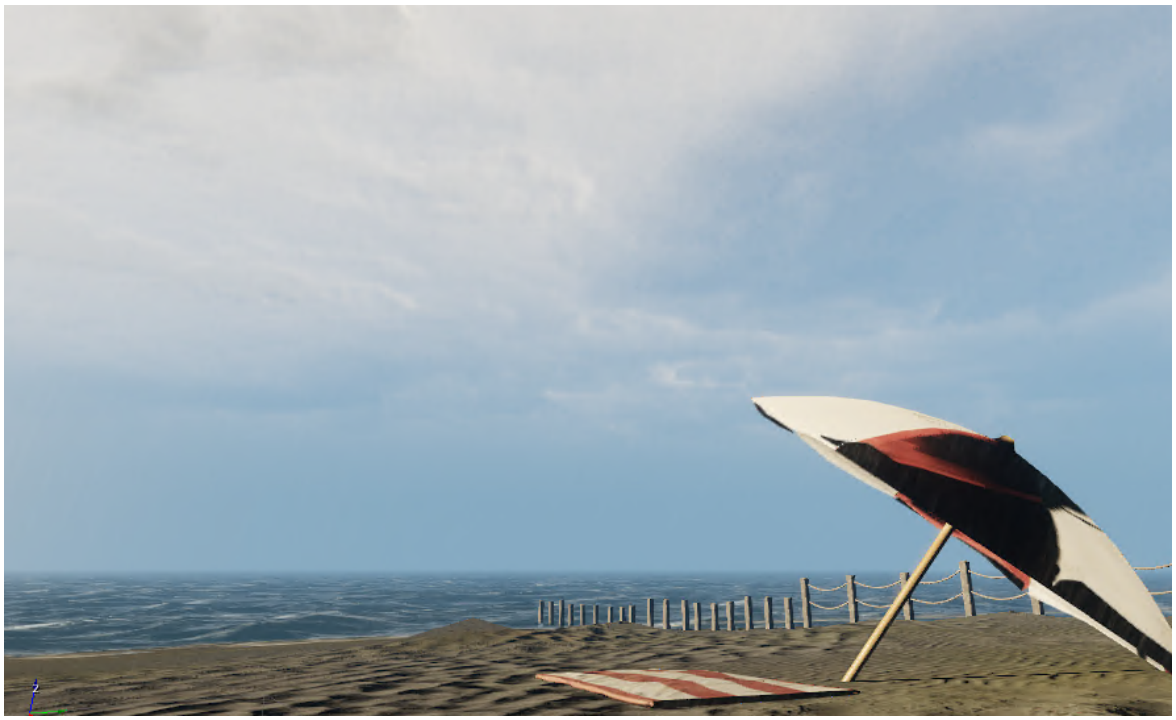
Using the Rain Sample

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.

Sky Clouds Gem

The Sky 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 Sky Clouds Gem in your project, see Gems (p. 1060).

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. 1197).



Topics

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 `.xml` file containing the description of the cloud
Scale	The scale of the cloud. **Note** This feature will be unavailable in a future release.

Properties	Description
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

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. 1060).

> **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. 631) 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	
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

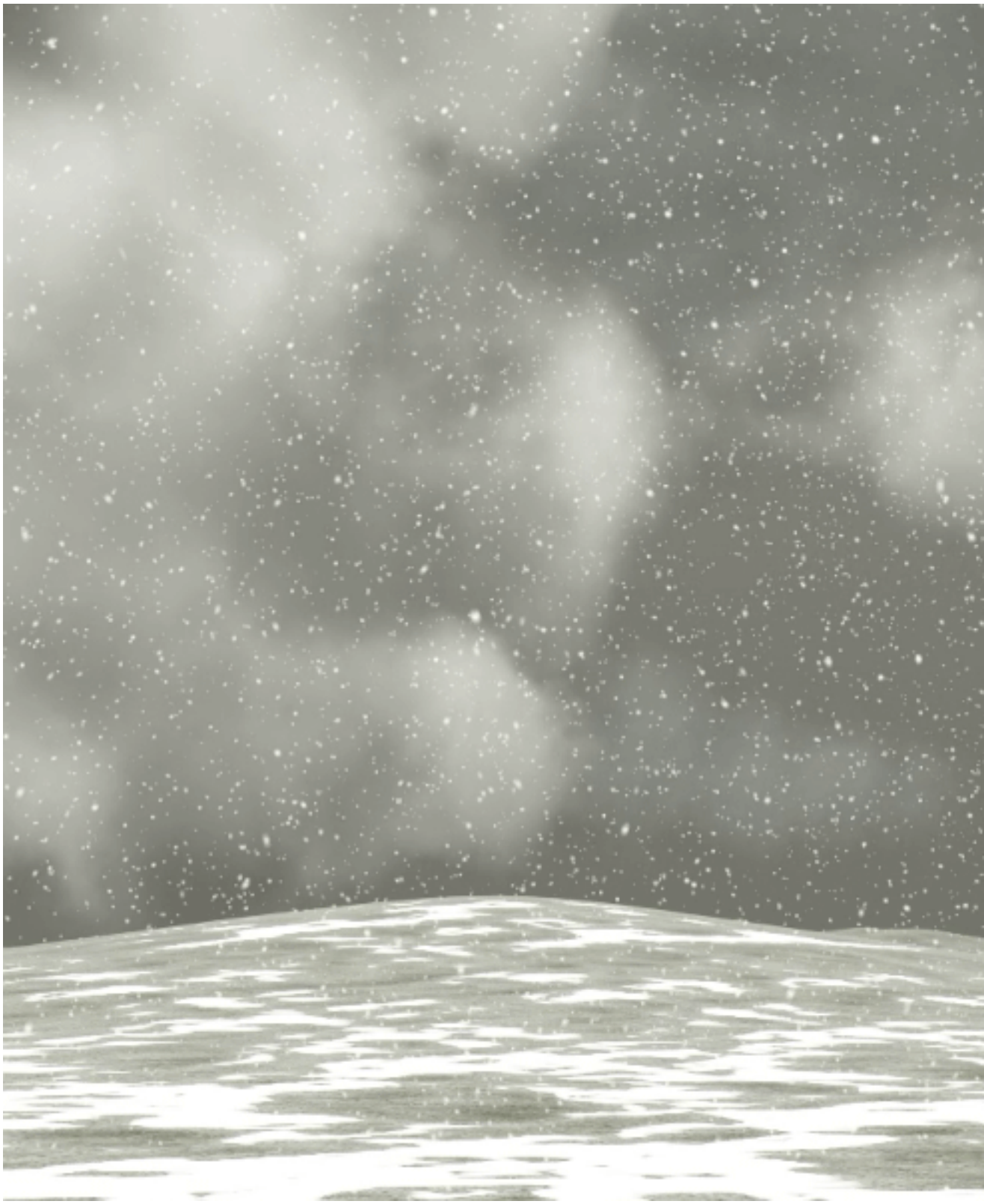
You can use the following console variables console variables (p. 93) for the snow entity.

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.



Tornadoes Gem

This gem is deprecated and will be removed in a future release.

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. 1060).

To add a tornado to your level

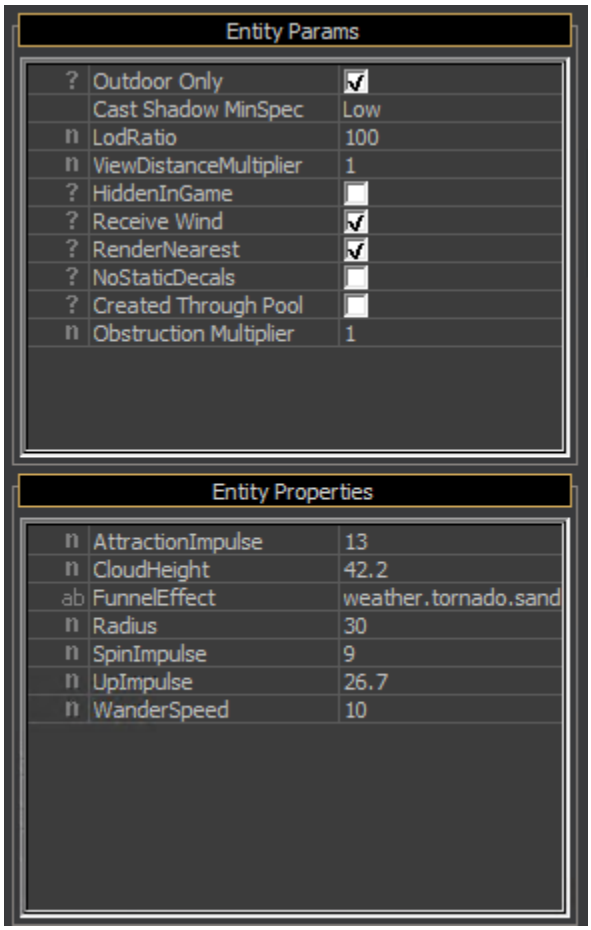
1. In the **Rollup Bar**, click **Entity**.
2. Under **Browser**, expand **Environment**.
3. Drag the **Tornado** entity into your scene.

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. 631) 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. 1149).
Radius	Sets the area that the tornando 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 (see example of a preset below).
   - 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=0.925,z=1))
 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>
```

Twitch Gem

Install the Twitch Gem into your Lumberyard project to access Twitch features. This gem makes it simple to engage broadcasters, support the Twitch Game Commerce platform, and leverage existing Twitch functionality for your game.

The Twitch Gem supports the following features:

- Twitch Friends – With the Twitch Gem installed, you can take advantage of Twitch Friends from within your product. Gamers can use this feature to stay connected and communicate with one another, whether they are in-game or on the web watching their favorite broadcaster.
- Activity Sharing (rich presence) – This technology creates presence awareness, where Twitch users can see what channels their friends are watching, whether their friends are broadcasting, and what games their friends are playing. This can help you design and build a more connected gaming experience while expanding your product's visibility across the Twitch platform.
- Channels – Currently available in the released Twitch API.
- Commerce – With the Twitch Gem installed, your users can purchase items from your games catalog through the Twitch desktop app.

Prerequisites

To use the Twitch Gem, you must:

- Be authorized as a Twitch development partner. To register as a developer on Twitch's Developer Portal, visit https://devportal.twitch.tv/.
- Obtain and extract the Fuel SDK from Twitch. See Installing the Fuel SDK and Twitch Gem (p. 1152) below.

- Extract and enable the Twitch Gem from Lumberyard. See Installing the Fuel SDK and Twitch Gem (p. 1152) below.

Installing the Fuel SDK and Twitch Gem

You must receive approval from Twitch for the launcher application to use your application.

To install the Fuel SDK and Twitch Gem

1. After you are authorized as a Twitch development partner, register your application and generate a client ID:
   - Visit https://www.twitch.tv/settings/connections.
   - Under **Developer Applications**, click **Register your application**.
   - On the **Register your Application** page, provide your application name, redirect URI, and application category.
   - Click **Register**.
2. To receive access to the Twitch API operations and Fuel SDK, contact Twitch Developer Success at integrationsuccess@twitch.tv and provide your application client's ID. Twitch will send you the Fuel SDK.
3. Extract the Fuel SDK `.zip` file to the following directory: `\`*`lumberyard_version`*`\3rdParty\Twitch\FUEL\`*`fuel_sdk_version`*.
4. Run the Lumberyard Setup Assistant to verify that the Fuel SDK is correctly detected. For more information, see Running Lumberyard Setup Assistant (p. 20).
5. After you have the Fuel SDK, Lumberyard sends you the Twitch Gem. Extract the Twitch Gem `.zip` file to the following directory: `\`*`lumberyard_version`*`\dev\Gems`.
6. Use the Project Configurator to enable the Twitch Gem for your project. You must also enable the HttpRequestor Gem. For instructions, see Gems (p. 1060).

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. 1539).

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.

Video Playback Gem

With the **Video Playback** gem, you can play a video file on an object in your Lumberyard level. When you enable the video playback gem, the video playback component is enabled. For instructions on how to prepare Lumberyard for video playback, and how to set up video playback in your level, see Video Playback Component (p. 584).

Virtual Reality Gems

The Oculus, OpenVR, and OSVR Virtual Reality Gems support a variety of Virtual Reality head-mounted displays (HMDs).

For more information about how to use and configure the Virtual Reality Gems, see Configuring your Project for Virtual Reality (p. 1782).

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. 1361) and DistanceClouds Shader (p. 1362).
   - For skies use the Sky Shader (p. 1383) and SkyHDR Shader (p. 1384) shaders
   - For terrain use the Terrain.Layer Shader (p. 1386).
   - For water use the Water Shader (p. 1390), Waterfall Shader (p. 1392), and WaterVolume Shader (p. 1394).

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.

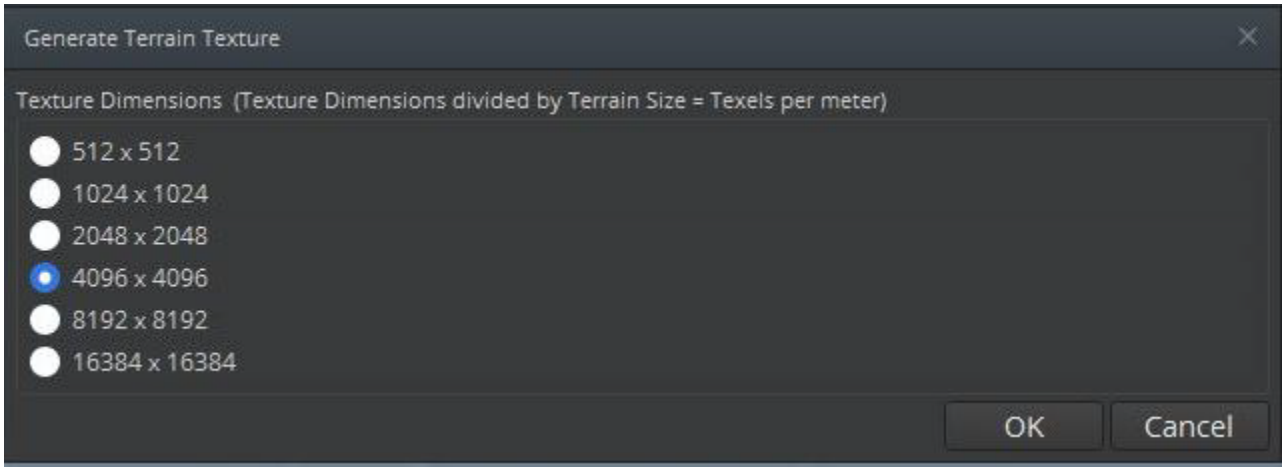
Topics

Creating a New Level

The first step in creating a game world is to create a level.

To create a new level

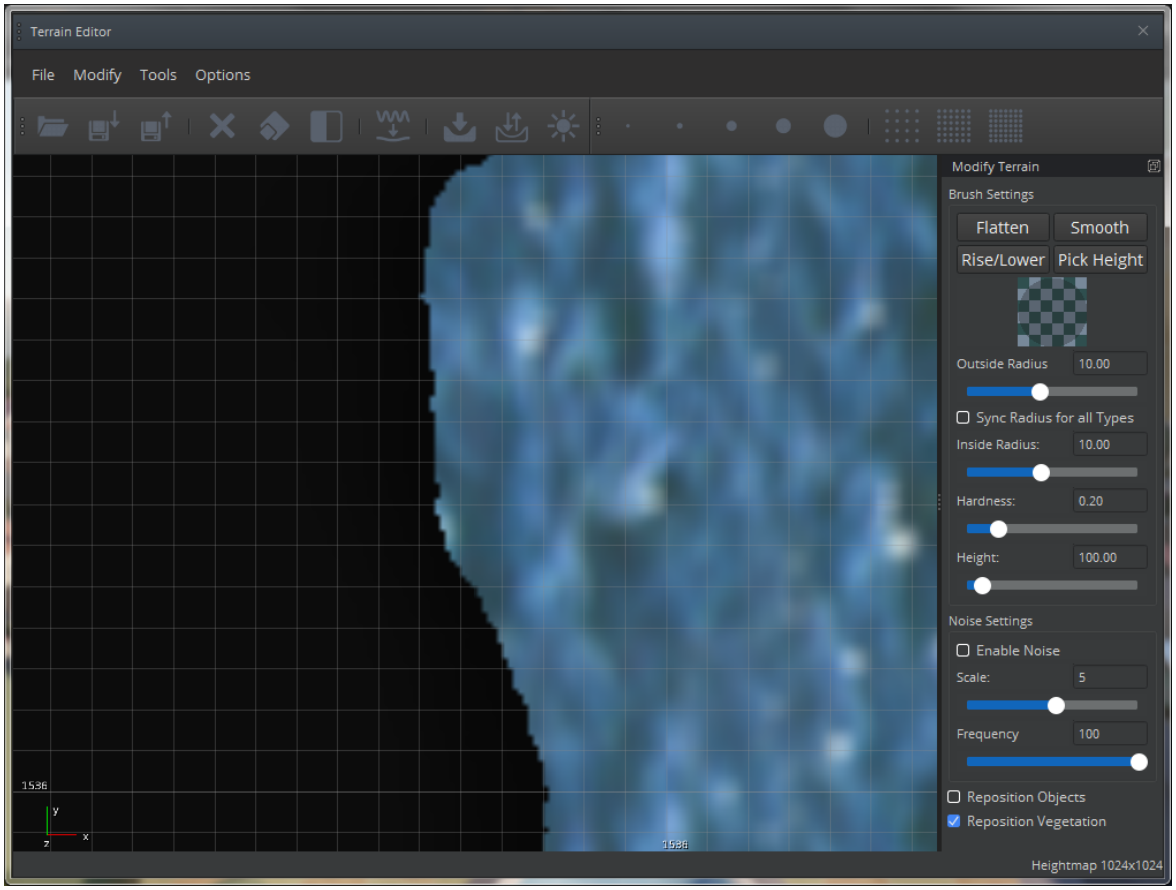
1. In Lumberyard Editor, click **File**, **New**.
2. In the **New Level** window, type a file name and select a directory location for the file.
3. Select the desired **Heightmap Resolution** and **Meters per Texel** values. Click **OK**.
4. In the **Generate Terrain Texture** window, for **Texture Dimensions**, select texture dimensions to determine texture tile resolution.



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

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

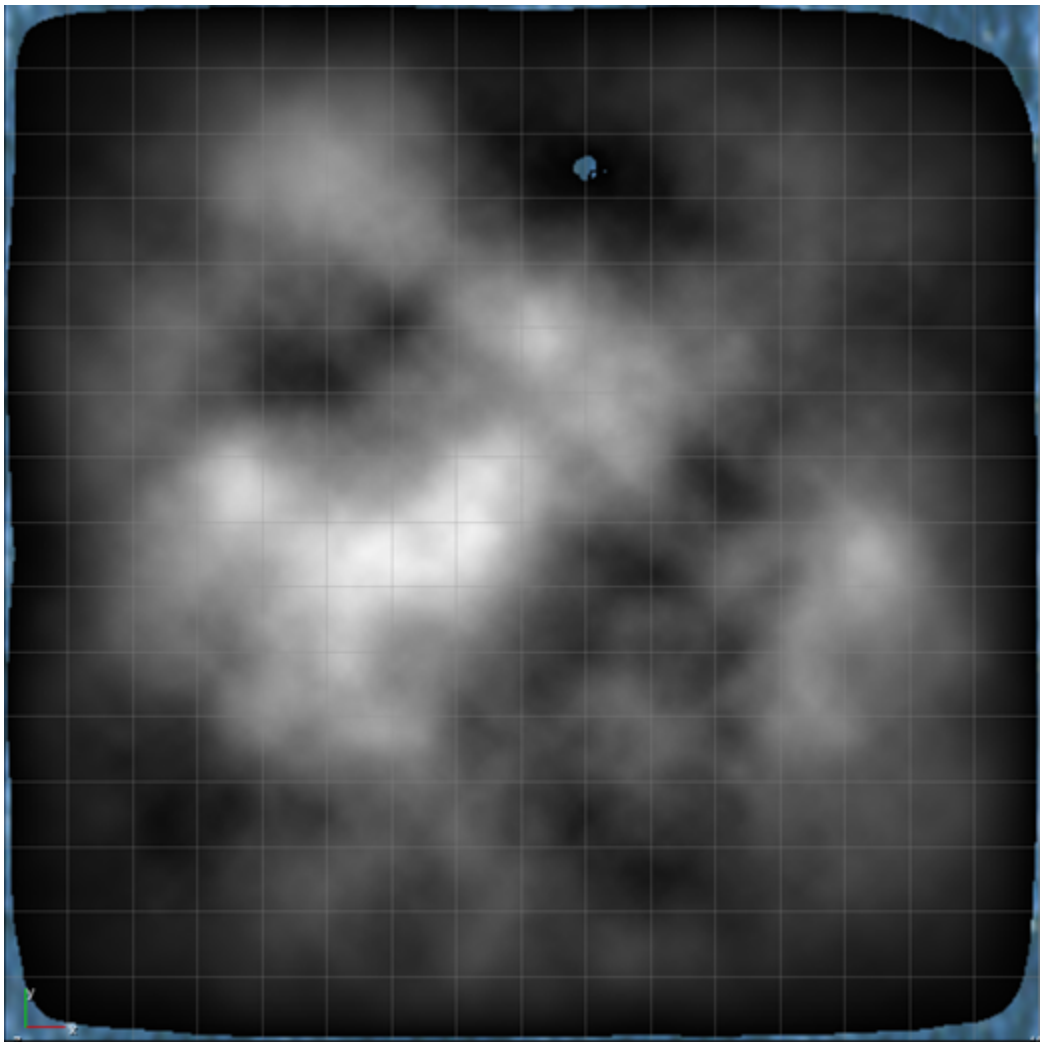
Topics

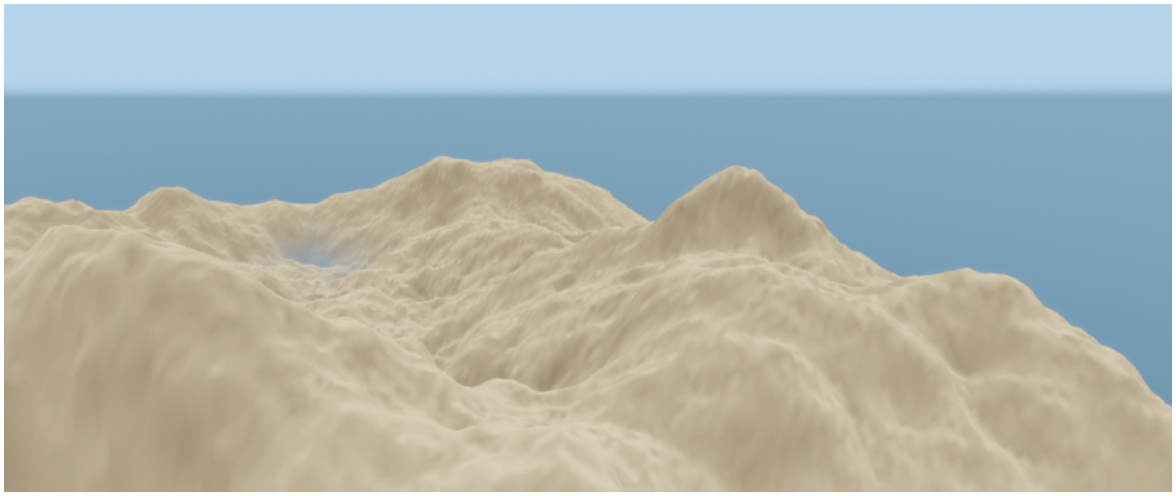
Creating a Terrain Heightmap

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, and then click **OK**.
4. In the editor menu, click **Game**, **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.

   Cover (Exp. Substract)

   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 **Game**, **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

In order to support full, 32-bit heightmap resolution, use one of the following file formats when you import a heightmap file.

- `.asc`
- `.bt`
- `.tif` (32-bit)

The following file formats are supported but not recommended. These file formats are imported with less than 32-bit resolution, which can cause blocky-looking terrain.

- `.pgm` (16-bit)
- `.raw` (16-bit)
- `.r16` (16-bit)
- `.png` (8-bit)
- `.bmp` (8-bit)
- `.tif` (8-bit or 16-bit)

If you attempt to import a file that was created with a different resolution than your existing heightmap, Lumberyard prompts you to do one of the following:

- Clip your imported file to remove any values that are outside the boundary of your existing heightmap.
- Resize your imported file to shrink or stretch to fit the heightmap. Resizing may introduce artifacts that can cause blocky-looking terrain.

To import a heightmap

1. In Lumberyard Editor, choose **Game**, **Terrain**, **Edit Terrain**.
2. In **Terrain Editor**, choose **File**, **Import Heightmap**.

Exporting a Terrain Heightmap

You can export a heightmap file that was created in the Terrain Editor to the following file formats:

- `.asc` (32-bit)
- `.bt` (32-bit)
- `.tif` (32-bit)
- `.pgm` (16-bit)
- `.raw` (16-bit)
- `.r16` (16-bit)
- `.bmp` (8-bit)
- `.png` (8-bit)

To export a heightmap

1. In Lumberyard Editor, choose **Game**, **Terrain**, **Edit Terrain**.
2. In **Terrain Editor**, choose **File**, **Export Heightmap** and type 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 **Game**, **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

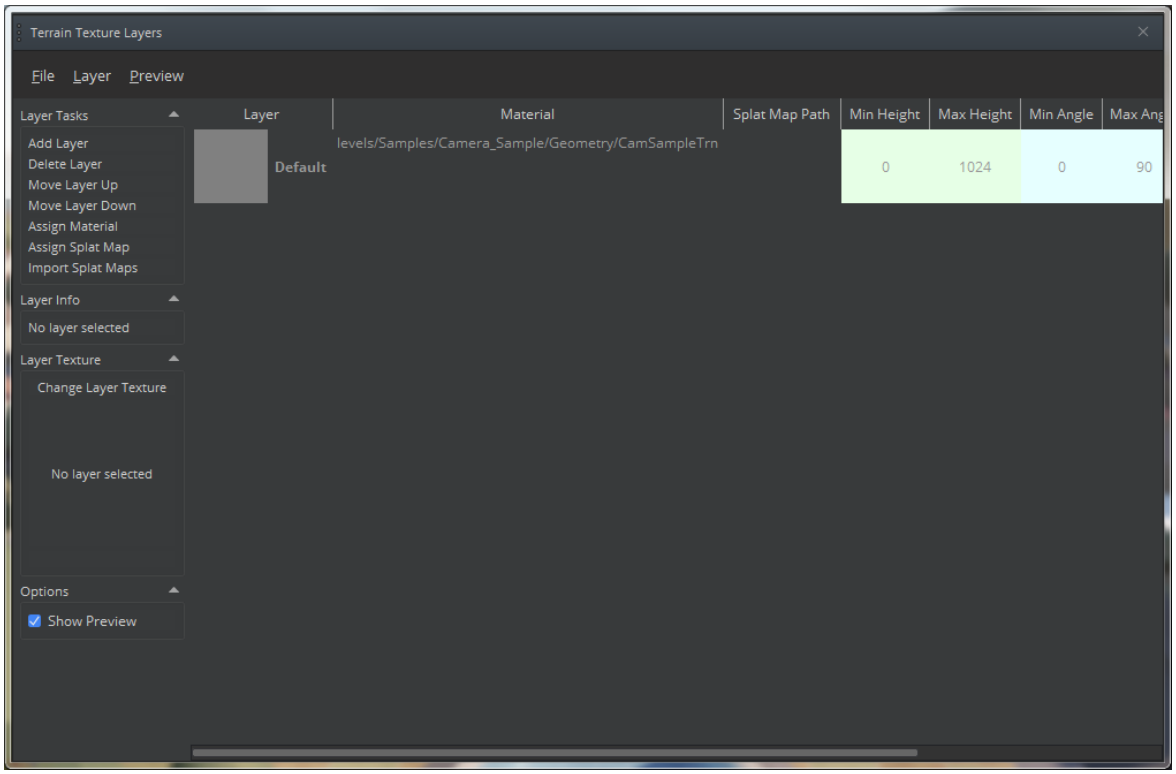
1. In the **Rollup Bar**, on the **Terrain** tab, click **Move Area**.
2. Click **Select Source**.

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

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 **Tools**, **Other**, **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. 1386). 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 Lumberyard Editor, click **Tools**, **Other**, **Terrain Texture Layers**.
2. In **Terrain Texture Layers**, double-click the layer you want to apply or edit a material for.
3. In **Material Editor**, expand the tree and select your asset.
4. Change settings and shader parameters as needed.
5. In **Terrain Texture Layers**, click **Assign Material**.
6. 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 **Tools**, **Other**, **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 reused it in multiple levels.

To export a terrain texture layer

1. In Lumberyard Editor, click **Tools**, **Other**, **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, in the **Rollup Bar**, on the **Terrain** tab, click **Environment** and adjust the value as needed.

To paint a terrain texture layer

1. In Lumberyard Editor, choose **Game**, **Terrain**, **Paint Layers**.
2. In the **Rollup Bar**, on the **Terrain** tab, click **Layer Painter**.
3. Adjust the terrain brush settings as needed.

   Radius

   Specifies the size of the brush. Use the slider to adjust the size, or use the following keyboard shortcuts: **[** to increase the brush radius size or **]** to decrease the brush radius size.

   Color Opacity

   Specifies the strength of the brush when applying the layer color. The brush is a spray brush that uses the color opacity value to determine the opacity at the center of the brush. The opacity decreases to transparent at the edge of the brush. Lower values create a more translucent brush, and higher values create a more opaque brush. Set the value to **0** to disable the color opacity. You can use the slider to adjust the opacity level, or you can use the following keyboard shortcuts:

   - **Shift+[** decreases the opacity.
   - **Shift+]** increases the opacity.

   You can also use the following keyboard shortcuts to adjust both parameters simultaneously with the same value:

   - **Shift+Ctrl+[** decreases both color opacity and detail intensity.
   - **Shift+Ctrl+]** increases both color opacity and detail intensity.

   Detail Intensity

   Specifies the strength of the brush when applying the detail texture to the layer. Like color opacity, the detail intensity value determines the intensity (opacity) at the center of the brush. The opacity decreases to transparent at the edge of the brush. Lower values create a more translucent (less intense) texture, and higher values create a more opaque texture. Set the value to **0** to disable detail intensity. You can use the slider to adjust the intensity level, or you can use the following keyboard shortcuts:

   - **Ctrl+[** decreases the intensity.
   - **Ctrl+]** increases the intensity.

   You can also use the following keyboard shortcuts to adjust both parameters simultaneously with the same value:

   - **Shift+Ctrl+[** decreases both color opacity and detail intensity.
   - **Shift+Ctrl+]** increases both color opacity and detail intensity.

   Mask by Layer Altitude and Slope

   Sets the material to paint only between the layer **Altitude** and **Slope** parameters defined below.

   Mask by

   Selects a layer to prevent it from being painted over.

4. Adjust the 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:

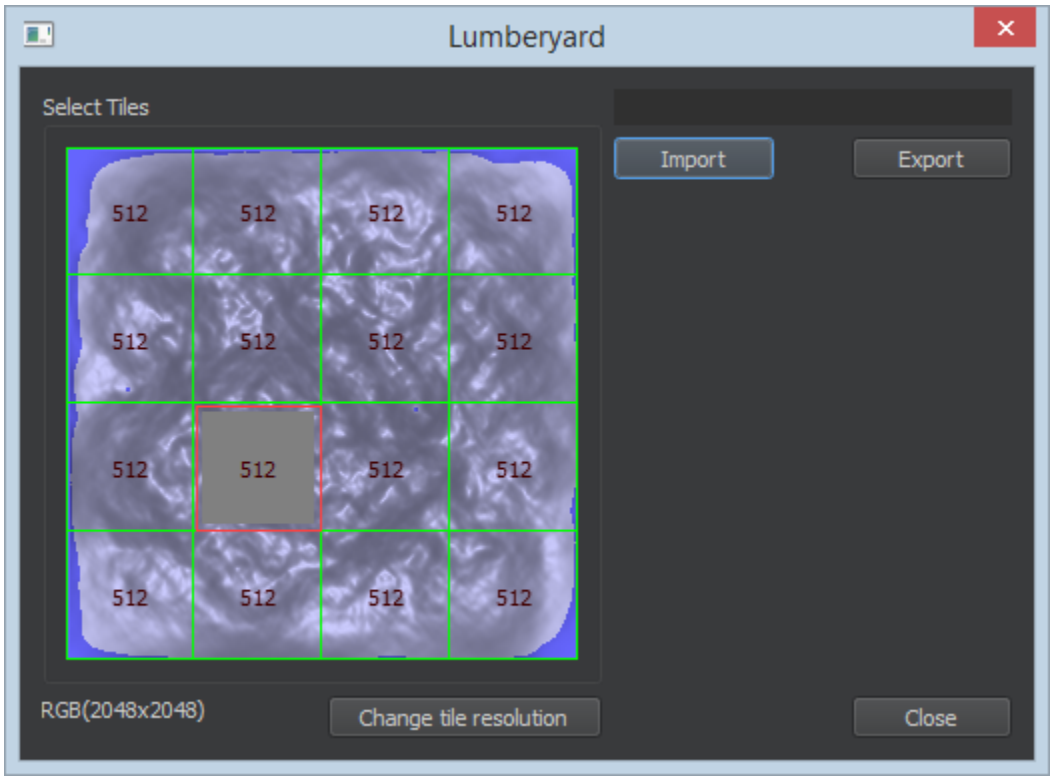
You first subdivide the texture layer, then change the individual tile resolution, as follows:

To subdivide the terrain texture layer

1. In Lumberyard Editor, click **Tools**, **Other**, **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 **Game**, **Terrain**, **Export/Import Megaterrain Texture**.
2. In **Terrain Texture**, click a tile whose resolution you want to change. Then click **Change tile resolution**.



3. Choose a new resolution and then click **OK**. Click **Close**.

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. 1158).

Topics

- Creating Roads (p. 1168)

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. 1166) and Using the Smooth Brush (p. 1166) to learn how to control the shape and overall visual look.

To use the Rise/Lower brush

1. In the **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. 1167) 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 the **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. 1167) 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 the **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. 1167) 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 the **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. 675).

Topics

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. 675).

To create and place the Road entity

1. In the **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 the **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 the **Rollup Bar**, on the **Objects** tab, click **Misc**, **Road**.
2. Click **<No Custom Material>** to open the Material Editor.
3. In the 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 the 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 the **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 the **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. 1390)** – For oceans only
- **WaterVolume Shader (p. 1394)** – For lakes, rivers, ponds and all other water volumes
- **Waterfall Shader (p. 1392)** – 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

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. 1171).

To prepare the terrain for bodies of water

1. In the **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. 1167) 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. 1390). You can change the ocean's various properties and effects.

To set ocean parameters

1. In the **Rollup Bar**, on the **Terrain** tab, click **Environment**.
2. Under **Ocean**, adjust the following parameter values:
   - **Material** – Click the (**...**) icon 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. 674).

Topics

- Splitting and Merging Rivers (p. 1173)

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.

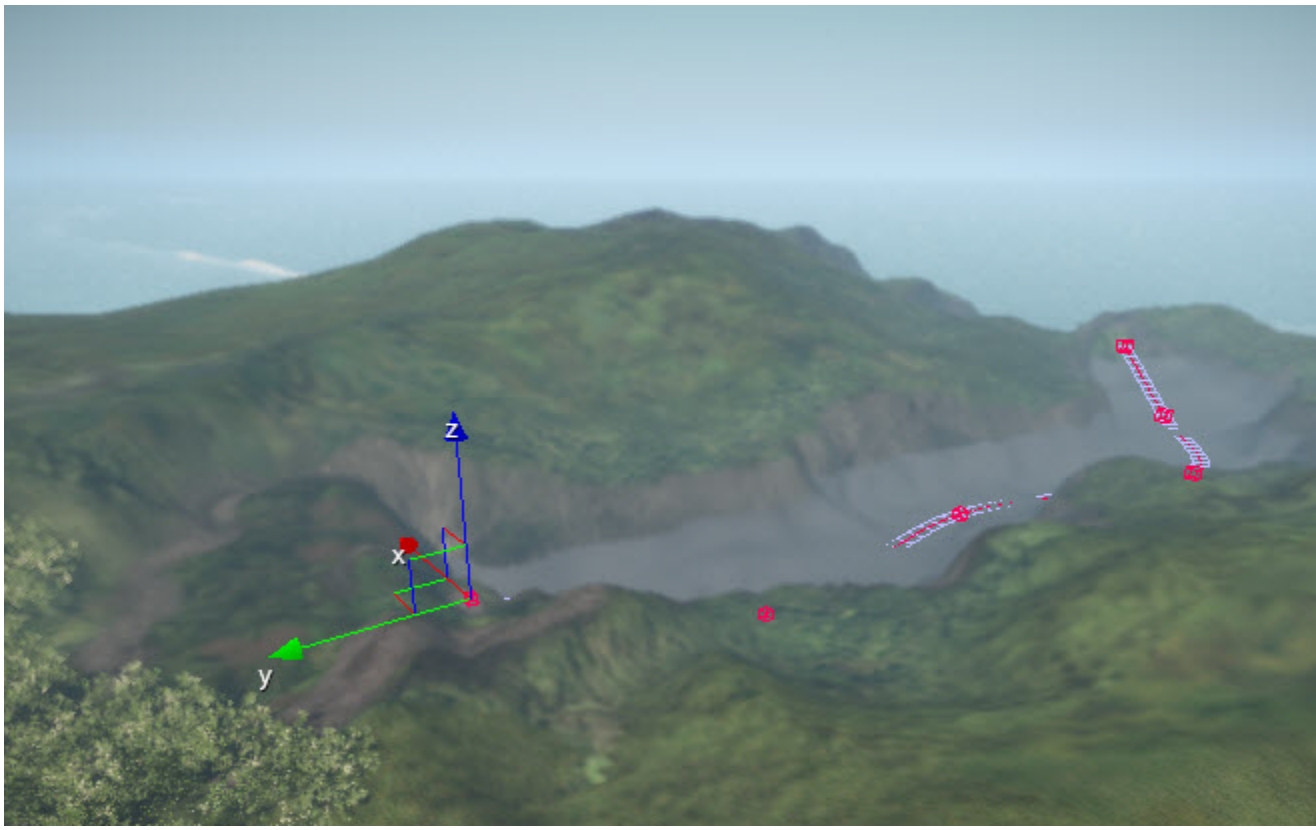
To create the riverbed and walls

1. In the **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. 1167) 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. 674).



To create and place the River entity

1. In the **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 the **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. 1394).

To apply a material to a river

1. In the **Rollup Bar**, on the **Objects** tab, click **Misc**, **River**.
2. Click **<No Custom Material>** to open Material Editor.
3. In the Material Editor, expand the tree and select your asset.
4. Modify material settings and WaterVolume Shader (p. 1394) 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 the **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 the **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 **Tools**, **Material Editor**.
2. In the 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 **Diffuse\Oscillator** 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. 1411).

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 **Tools**, **Material Editor**.
2. In the 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 the **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. 1439).

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 and Console Variables (p. 1185) **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

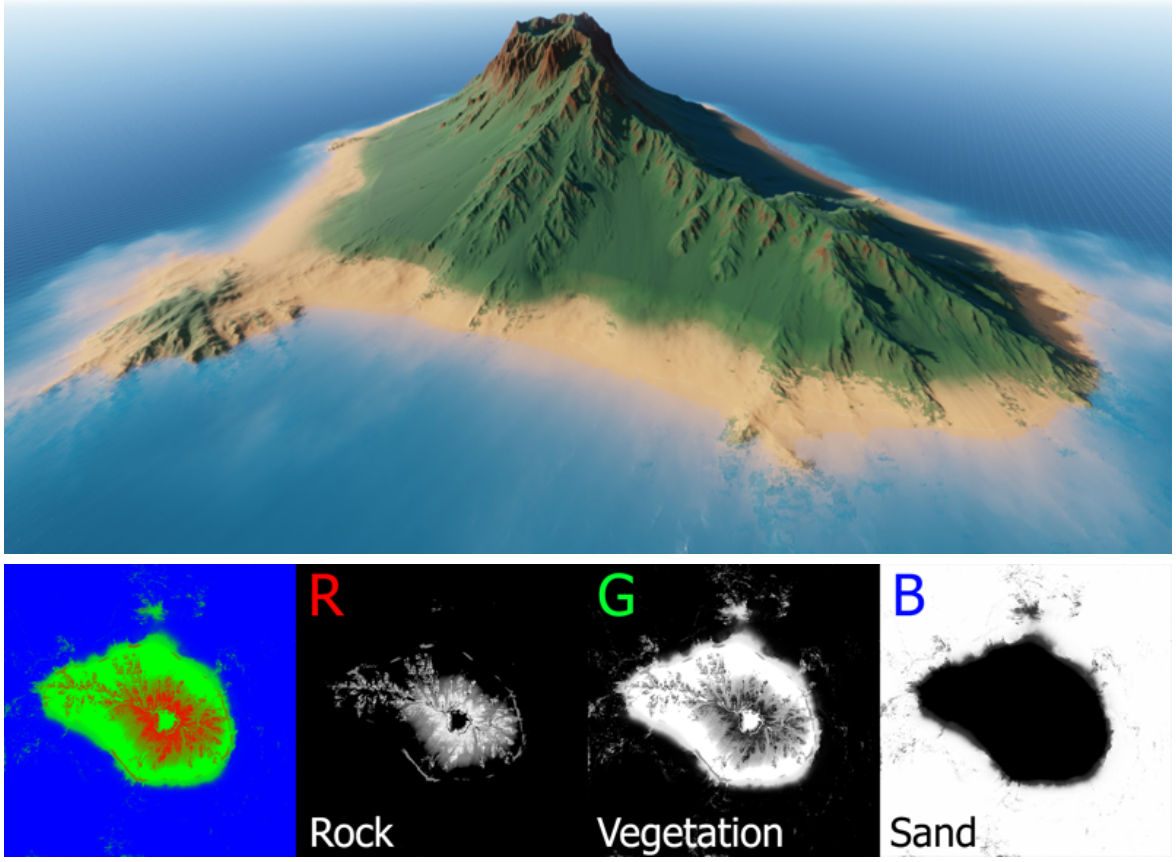
1. In Lumberyard Editor, click **Game**, **Terrain**, **Import Terrain Block** and select a `.trb` file to import.
2. Click **Terrain**, **Terrain Texture Layers**.
3. Click **File**, **Import Layers** and select terrain texture files to import.

To export a terrain block and texture layers

1. In Lumberyard Editor, click **Game**, **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 **Tools**, **Other**, **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. 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 Maps**. This clears the current weight map for the terrain and then rebuilds it using the selected splat maps.
5. In **Lumberyard Editor**, choose **Game**, **Terrain**, **Generate Terrain Texture**.

   Note
   You cannot apply masking to an imported splat map.

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.

Note
You can place or assign objects (p. 613) to layers, as described in Assigning Objects to Layers (p. 1179). Entities (p. 593), however, cannot be placed in layers. Even if it appears to work initially, all entities will be reverted to the base layer upon exiting the level file.

Topics

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 the **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 the **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 the **Rollup Bar**, click the **Layers** tab.
2. Right-click the layer and choose **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. 1180).

   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 the **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 the **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 all of the layers are visible (not hidden) in the Lumberyard Editor. If you hide layers in the editor, they will not be visible in the game, and flow graph cannot unhide them. You can, however, use flow graph to hide layers that are visible in the editor.
- Set the `Load by default` (p. 1179) option on the layer settings, or else the layer won't unhide correctly.
- 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.
- Use the `e_ObjectLayersActivation 1` console variable (default is `0`) to activate layer switching.

Layer Streaming

Layer streaming is disabled by default. To use it, you must first enabled it.

To enable layer streaming

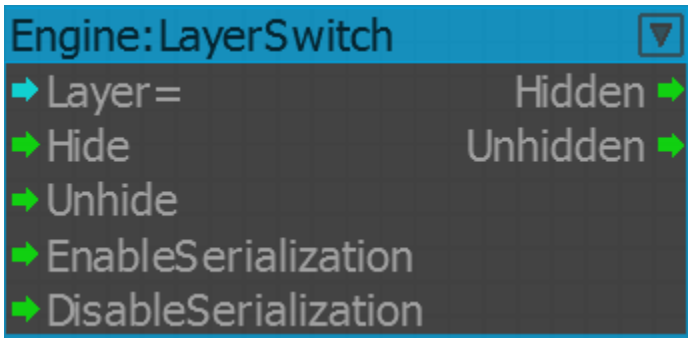
1. In the **Rollup Bar**, on the **Terrain** tab, click **Environment**.
2. Under **EnvState**, select **Use layers activation**.

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 the **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. 754)

Working with Database Libraries

With Lumberyard's **Database View** editor, you can organize and work within the libraries of your game items. You can add new or load existing libraries to your level, or remove libraries from your level. Within a library, you can organize items by customized groups, clone and edit items, assign items to selected objects, and retrieve properties from the current selection.

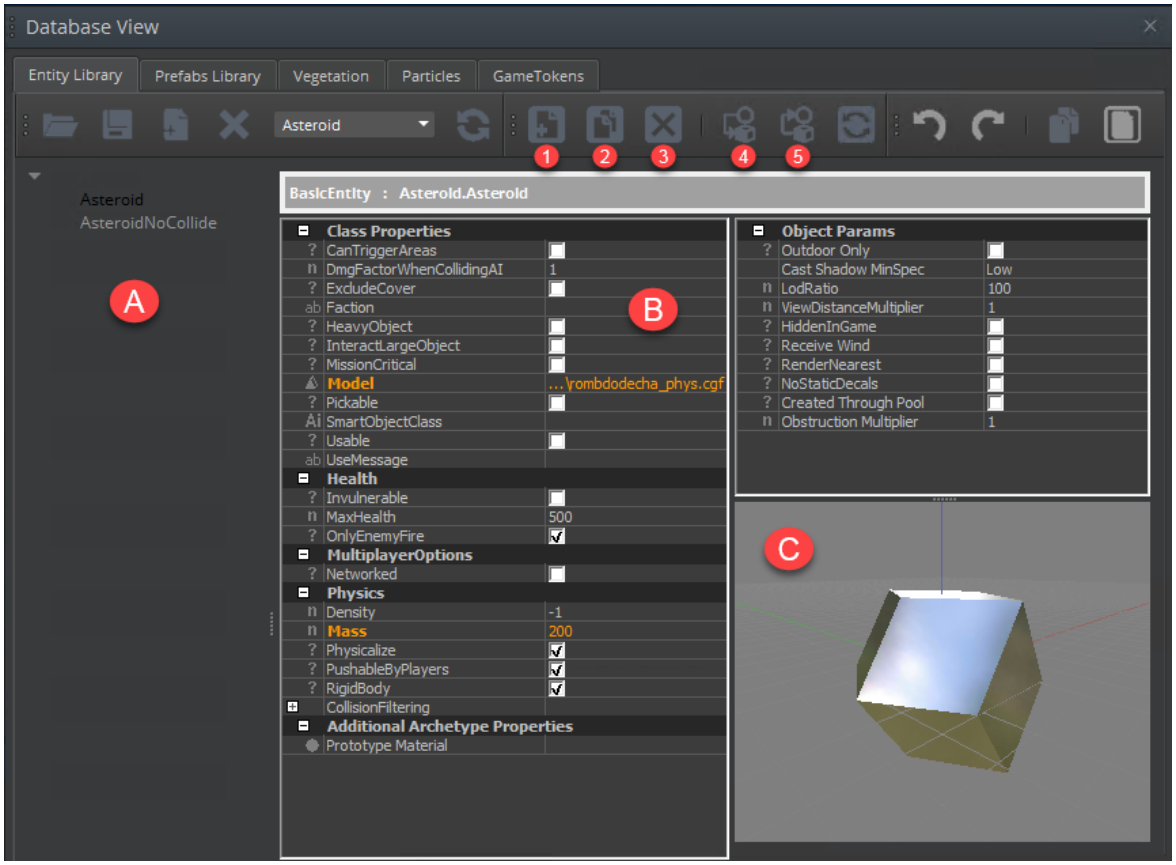
The **Database View** editor displays the following categories of libraries on its tabs:

- **Entity (p. 634) Library**
- **Prefabs (p. 629) Library**
- **Vegetation (p. 1200)**
- **Particles (p. 1274)**
- **GameTokens (p. 1055)**

Using the **Database View** editor's toolbar, you can load, save, or add new libraries, or remove a library. The drop-down menu features a list of libraries that you've already opened; select an open library from

this list. Use the **Add new item** tool (1) to add items to your library, **Clone library item** (2) to create a duplicate of an existing item, and **Remove item** (3) to delete the item from the library. Use **Assign item to selected objects** (4) to assign the selected item to objects currently selected in your viewport. Use **Get properties from selection** (5) to use the properties of the object selected in the viewport and apply those properties to the item selected in the item browser.

The item browser pane (A) shows the groups and items that are available in the current library. The properties pane (B) displays the properties of the item selected in the item browser. Some tabs also feature a preview pane (C).

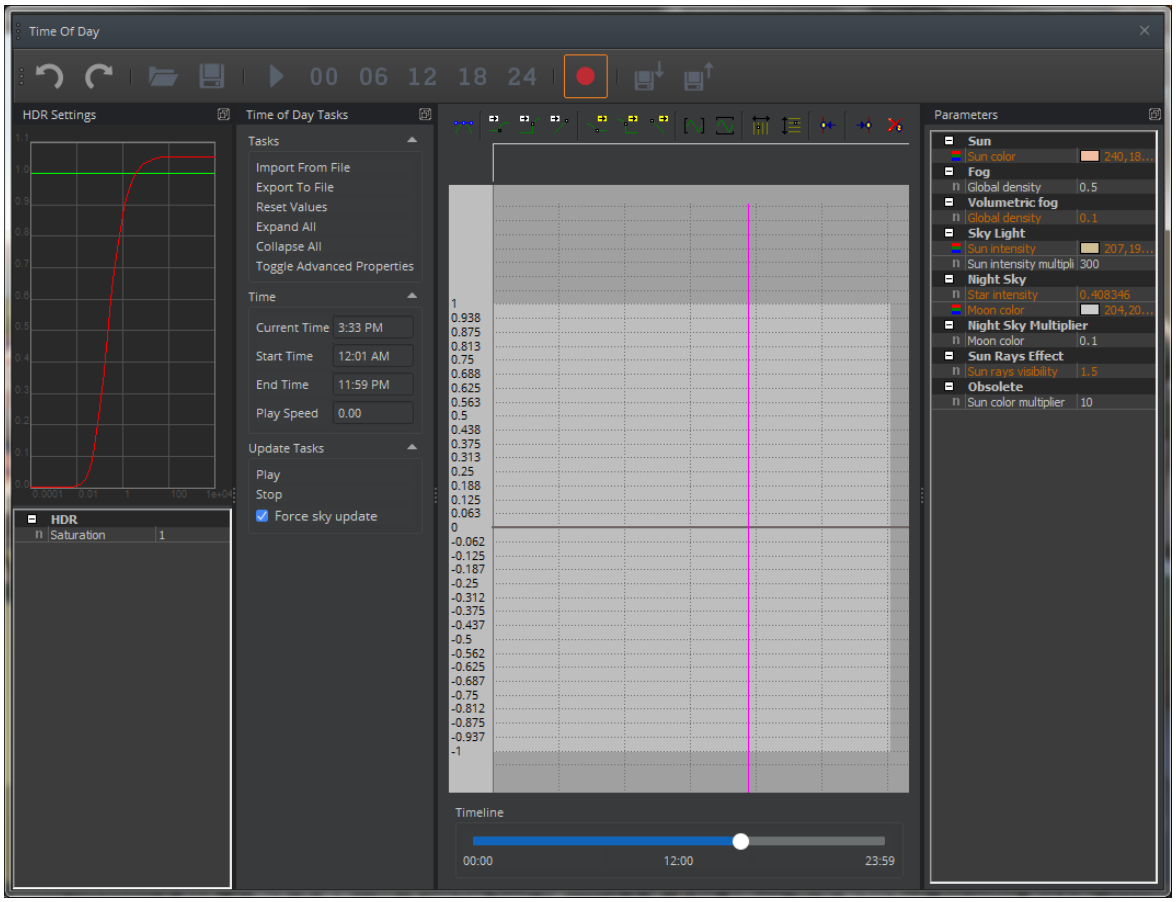


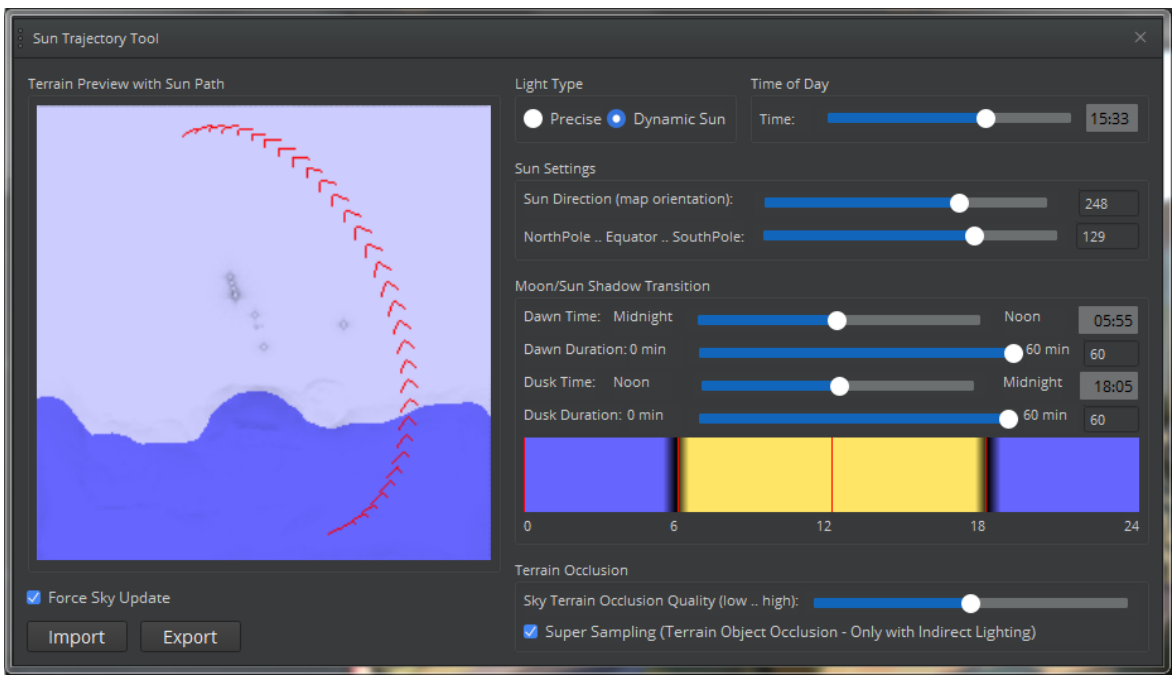
Adding Sky Effects

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. 1384) 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:





Topics

- Creating a Dynamic Daytime Sky (p. 1184)

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. 1384).

All properties and parameters in the following topics are ignored when you use a static sky (SkyBox), which uses the Sky Shader (p. 1383).

Topics

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 **Tools**, **Other**, **Time Of Day**.
2. In the **Time of Day Tasks** pane, under **Tasks**, choose **Toggle Advanced Properties** to view all parameters.
3. In the **Parameters** pane, under **Sky Light**, adjust the following parameters as needed:

   Sun intensity

   Sets the illuminance of the sun and uses an RGB sun color value to compute the atmosphere color. You can use this parameter in conjunction with **Sun color** to provide desired scene luminance. For bright sunlight, you can use an average illuminance value of 120000 lux. The intensity of the sun decreases as it nears the horizon, so you can add keys to the timeline to lower the intensity value appropriately. To accurately render shadows at night time, you can add keys to the timeline and set the value to 5.

   Sun intensity multiplier

   Sets the brightness of the sun. The brightness is multiplied by the sun intensity to yield the overall color.You can use this parameter in conjunction with **Sun color multiplier** to provide desired scene luminance. Higher values result in brighter skies. Lower values simulate an eclipse.

   Mie scattering

   Controls mie scattering, which is caused by pollen, dust, smoke, water droplets, and other particles in the lower portion of the atmosphere. Mie scattering occurs when the particles that cause the scattering are larger than the wavelengths of radiation that are in contact with them. Mie scattering is responsible for the white appearance of clouds.

   Higher values create a hazy sky. Lower values create a clear sky. For a balanced sky, you can set this value to 4.8.

Rayleigh scattering

Controls rayleigh scattering, which is sunlight scattering from atmospheric gases. Rayleigh scattering occurs when the particles that cause the scattering are smaller than the wavelengths of radiation that are in contact with them. As the wavelength decreases, the amount of scattering increases. Rayleigh scattering is responsible for the blue appearance of the sky.

Higher values create a red-yellow sky. Lower values create a blue sky. For a blue daytime sky and red-yellow sunset sky, you can use the default value of 2.0.

Sun anisotropy factor

Controls the sun's apparent size. As this value approaches -1.0, the sun's disk becomes sharper and smaller. Higher values create a larger, fuzzier disk. For a balanced size, you can set this value to -0.995.

Wavelength R, G, B

Sets the hue (RGB values) of the atmosphere. You can create different atmospheres by adjusting the color values and gradients. This can be particularly useful with rayleigh scattering, when you choose a sun intensity of pure, bright white.

Setting Sun Parameters and Console Variables

You can define how the sun appears in the daytime sky.

To set sun parameters

1. In Lumberyard Editor, choose **Tools**, **Other**, **Time Of Day**.
2. In the **Time of Day Tasks** pane, under **Tasks**, choose **Toggle Advanced Properties** to view all parameters.
3. In the **Parameters** pane, under **Sun**, adjust the following parameters as needed:
   - **Sun color** – Sets the color (RGB values) of the sun. You can use this parameter along with **Sun intensity** to provide the preferred scene luminance. This parameter is also used to control the color and intensity of the light from the moon.
   - **Sun intensity (lux)** – Sets the brightness of the sun. This value is multiplied by the sun color to yield the overall color.
   - **Sun specular multiplier** – Controls the brightness and intensity of the sun on specular materials in your scene.

You can define how the sun renders in the daytime sky.

To set console variables for sun rendering behavior

1. In Lumberyard Editor, in the **Console** pane, click the icon in the bottom left corner.
2. In the **Console Variables** window, search for and set the following console variables to control the sun's rendering behavior:
   - `e_Sun` – Activates or deactivates the sun's light source. 0 = deactivate | 1 = activate.
   - `e_SunAngleSnapDot` – Updates the cached sun direction if the dot product (cached sun direction and real-time sun direction) is less than the specified value for this console variable.
   - `e_SunAngleSnapSec` – Determines how often, in seconds, to update the cached sun direction.
   - `e_SkyUpdateRate` – Specifies the percentage of a full dynamic sky update, calculated per frame. This console variable affects how smoothly the sun moves in the sky, if the speed value in the

Time of Day Editor is greater than 0. Higher values may impact rendering performance. Valid values are 0 to 100.

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 **Tools**, **Other**, **Time Of Day**.
2. In the **Time of Day Tasks** pane, under **Tasks**, choose **Toggle Advanced Properties** to view all parameters.
3. In the **Parameters** pane, under **Sun Rays Effect**, adjust the following parameters as needed:

   - **Sun shafts visibility** – Sets the visibility of the sun shafts. Higher values accentuate the shadow streaks that are caused by the sun light penetrating objects.
   - **Sun rays visibility** – Sets the visibility of the sun rays. Higher values create brighter rays around the sun.
   - **Sun rays attenuation** – Sets the length of the sun rays. Higher values create shorter rays around the sun.
   - **Sun rays sun color influence** – Sets the degree to which the color of the sun contributes to the color of the sun rays. Set this value to 1 to use the color of the sun for the sun rays. Set this value to 0 to use a custom color. Set this value to any number between 0 and 1 to use a color between the sun color and custom color.
   - **Sun rays custom color** – Specifies a custom color for the sun rays. To use this parameter, you must set the **Sun rays sun color influence** parameter to a value greater than 0.

Setting Sun Shadow Settings

You can define how sun shadows appear in your level.

To set sun shadow settings

1. In Lumberyard Editor, click **Tools**, **Rollup Bar**.
2. In the **Rollup Bar**, on the **Terrain** tab, choose **Environment**.
3. Under **EnvState**, adjust the following parameters as needed:

   - **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 **Tools**, **Other**, **Time Of Day**.
2. In the **Time of Day Tasks** pane, under **Tasks**, click **Toggle Advanced Properties** to view all parameters.

3. In the **Parameters** pane, under **Shadows**, adjust the following parameters as needed for each shadow cascade:

   - **Cascade *number* Bias** – Specifies the distance of the shadow connection from the shadow-casting object. For a more realistic effect, set the value between 0.01 and 0.05.
   - **Cascade *number* Slope Bias** – Specifies the slope gradient for the shadows. Higher values reduce shadows that are cast from an object with a high light angle. For a more realistic effect, set the value between 32 and 64. Slope bias has little to no impact on performance.
   - **Shadow Jittering** – Sets the softness of all cascaded sun shadows. Higher values may 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. 1384).

All properties and settings in the following topics are ignored when using a static sky (SkyBox).

Topics

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 **Tools**, **Other**, **Time Of Day**.
2. In the **Time of Day Tasks** pane, under **Tasks**, choose **Toggle Advanced Properties** to view all parameters.
3. In the **Parameters** pane, under **Night Sky** and **Night Sky Multiplier**, adjust the following parameters as needed:

   Horizon color (and multiplier)

   Specifies the horizon color of the night sky gradient. The RGB value is scaled by the multiplier. You can make a light pollution effect more pronounced by setting this value to a dark blue color (20, 36, 51).

   Zenith color (and multiplier)

   Specifies the zenith color of the night sky gradient. The RGB value is scaled by the multiplier. You can make the night sky gradually appear darker further from the horizon by setting this value to black.

   Zenith shift

   Sets the transition for the two colors in a night sky gradient. Smaller values shift the transition towards the bottom. Larger values shift the transition towards the top. You can create a smooth transition by setting this value to 0.8.

   Star intensity

   Sets the overall brightness of the stars. Star flickering is by design and cannot be controlled. You can make the stars bright for night time by setting this value to 0.01 between 00:00 – 06:00 and 18:00 – 23:59.

Moon color (and multiplier)

Specifies the moon's emissive color. The RGB value is scaled by the multiplier. You can make the moon a less saturated blue by setting this value to 51, 58, 65. You can create a light blue transition color by creating two keys at 07:00 and 17:00 and setting the value to 200, 228, 255. Setting this value to 0 removes the moon texture.

Moon inner corona color (and multiplier)

Specifies the color of the moon's inner corona (glow around the moon). The RGB value is scaled by the multiplier.

Moon inner corona scale

Specifies the size and blurriness of the moon's inner corona. Smaller values create a bigger, blurry corona. Larger values create a smaller, focused corona.

Moon outer corona color (and multiplier)

Specifies the color of the moon's outer corona. The RGB value is scaled by the multiplier.

Moon outer corona scale

Specifies the size and blurriness of the moon's outer corona. Smaller values create a bigger, blurry corona. Larger values create a smaller, focused corona.

Setting Moon Parameters

You can define how the moon appears in the nighttime sky.

To set moon parameters

1. In Lumberyard Editor, click **Tools**, **Rollup Bar**.
2. In the **Rollup Bar**, on the **Terrain** tab, choose **Environment**.
3. Under **Moon**, adjust the following parameters as needed:
   - **Latitude** – Sets the latitude of the moon.
   - **Longitude** – Sets the longitude of the moon.
   - **Size** – Adjusts the size of the moon.
   - **Texture** – Specifies the asset for creating the moon texture.

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

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 Lumberyard Editor, click **Tools**, **Layer Editor**.
2. In the **Layer Editor**, select **Terrain** and **Lighting**.
3. Close the **Layer Editor**.
4. In Lumberyard Editor, click **Tools**, **Other**, **Sun Trajectory Tool**.
5. In the **Sun Trajectory Tool**, set the following properties and parameter values as needed:

   Time of Day

   Sets the current time.

   Sun Direction

   Specifies the direction where the sun rises.

   Dawn Time

   Specifies the time the sun will rise.

   Dawn Duration

   Sets the transition duration of moon-to-sun lighting.

   Dusk Time

   Specifies the time the sun will set.

   Dusk Duration

   Sets the transition duration of sun-to-moon lighting.

   Force sky update

   If selected, updates the sky light calculations for each frame. If cleared, distributes calculations over several frames.

   Import

   Imports settings from a saved lighting file (`.lgt`).

   Export

   Exports current settings to a lighting file (`.lgt`).

   Terrain Occlusion

   Creates the effect of indirect lighting.

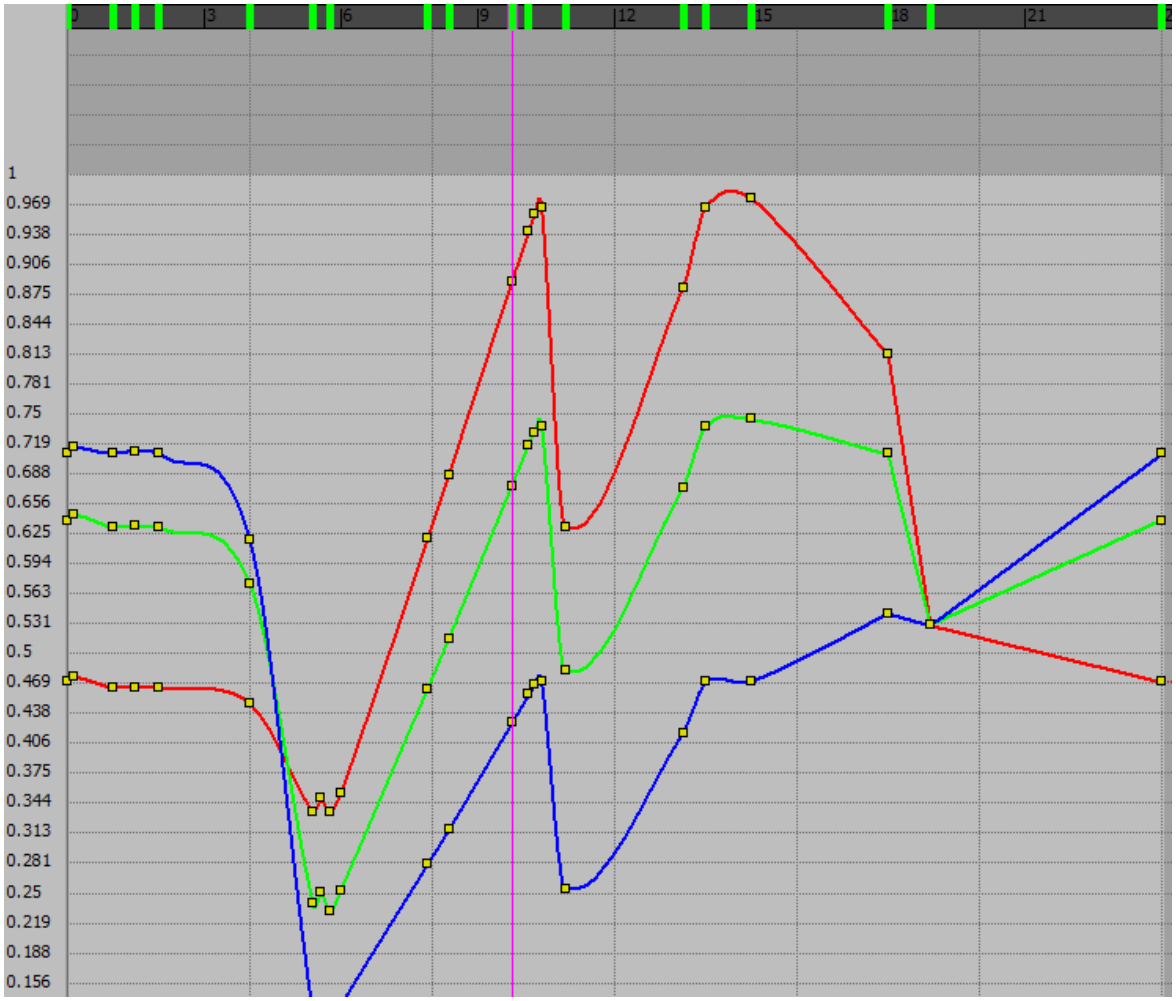
   Super Sampling

   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. Select the red button to record.

Environment parameter values that you change in the **Parameters** panel of the Time of Day Editor are set for the currently selected time. The Time of Day graph shows changes to the selected parameter over time. When 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 **Tools**, **Other**, **Time Of Day**.
2. In the **Time of Day Tasks** pane, under **Tasks**, click **Toggle Advanced Properties** to view all parameters.
3. In the **Parameters** pane, adjust the parameter value for each cycle that you want to create. Then do the following:

   a. Click the red button to start recording.

   b. In the **Time of Day Tasks** pane, under **Time**, set the **Current 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. In the **Time of Day Tasks** pane, 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

   Sets the current time in the **Time of Day** editor.

   Start Time

   Sets the time to use when the game starts. This is not the same value as the current time.

   End Time

   Sets the time to use when the game ends. If you set the end time to 23.59, the time loops and starts the next cycle when the day is over.

   Play Speed

   Sets the speed at which time advances in the cycle.

   Play

   Starts or resumes the playback of the cycle in the **Time of Day** 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 Static Skyboxes

Static skies use the Sky Shader (p. 1383) and a skybox, which is a cube that has textures on five sides (excluding the bottom). This allows you to simulate the sky in your level. Static skies cannot use dynamic or animated **Time of Day** effects, HDR settings, and sun or moon parameters.

If you want to create a static sky, create a skybox material and then apply the material to the skybox.

Topics

Creating Skybox Materials

Skyboxes in Lumberyard are 5-sided. Sides 1 through 4 are the sides of the box, and side 5 is the top of the box. Skyboxes in Lumberyard do not have a bottom.

The sides of the box are represented by three textures that follow a specific naming convention:

1. `skybox_12.tif`
2. `skybox_34.tif`
3. `skybox_5.tf`

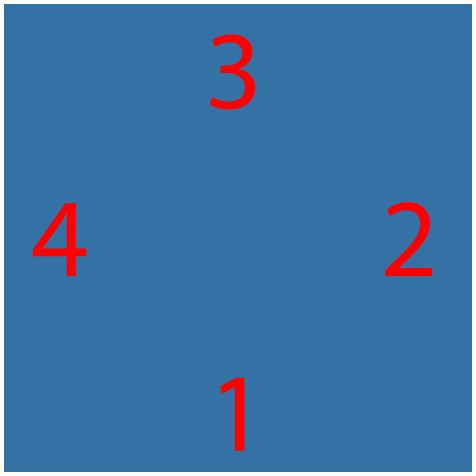
The suffix in the name indicates which sides of the box that the texture is mapped to.

You can use any image editing software to create a skybox texture. We used Adobe Photoshop to create this example and followed these specifications:

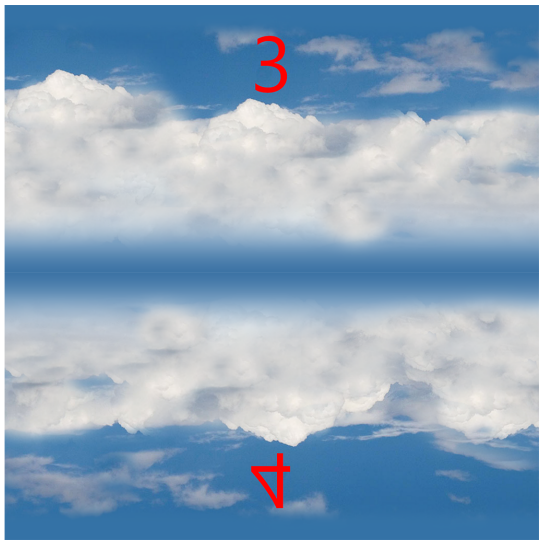
- The source sky image is authored at 8192 x 1024.
- The image uses 16 bits per channel to help preserve smooth gradients in the sky.
- The image is divided into four sections that are 2048 x 1024 each (for illustration purposes).

The numbers represent each side of the skybox that the texture is mapped to. Sides 1 and 2 will become `skybox_12.tif` and sides 3 and 4 will become `skybox_34.tif`.

The top of the example sky is a separate texture. The numbers represent which edge of the texture to map to which side. This is important for you to manage texture seams. You must name this 2048 x 2048 texture `skybox_5.tif`.



When you combine sides 1 and 2 into a single texture (`skybox_12.tif`), you must flip side 2 horizontally. When you combine sides 3 and 4 into a single texture (`skybox_34.tif`), you must flip side 4 horizontally. This results in two textures that are 2048 x 2048 each.



To prepare the skybox textures for Lumberyard

1. Navigate to the directory with your newly created textures.
2. Right-click each texture and choose **RC Open Image**.

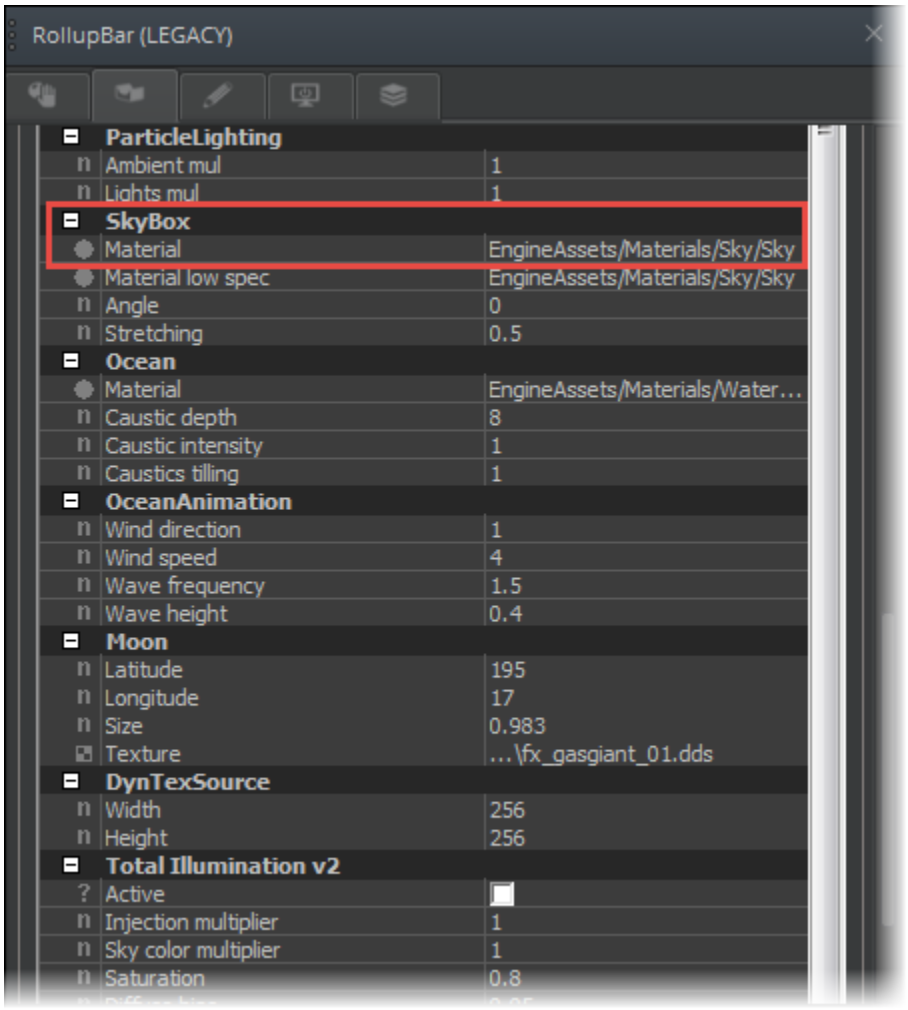
   Note
   If you do not see **RC Open Image**, you must install RC Shell Commands using the Lumberyard Setup Assistant. For more information, see Using Lumberyard Setup Assistant to Set Up Your Development Environment (p. 20).
3. In the texture dialog box, under **Preset**, choose **SkyboxHDR** from the drop-down list. Click **OK**. This will create a text file (`imagename.exportsettings`) that Lumberyard uses to compile the texture correctly.
4. If you use source control, you must check in all images and `.exportsettings` files. If you do not check in these files, your scene will not render similarly for each person on your project.

Applying Skybox Materials to a Skybox

Do the following to apply a skybox material to a skybox.

To apply a skybox material to a skybox

1. In Lumberyard Editor, choose **Tools**, **Material Editor**.
2. In the **Material Editor**, select an existing skybox material from the tree pane or create a new material.
3. Under **Material Settings**, choose **Sky** from the **Shader** drop-down list.
4. Under **Texture Maps**, for the **Diffuse** texture, choose one of the three skybox textures that you created.
5. In Lumberyard Editor, choose **Tools**, **Rollup Bar**.
6. In the **Rollup Bar**, on the **Terrain** tab, click **Environment**.
7. Under **SkyBox**, click the browse (**..**) button to choose a new material for the **Material** parameter.



8. In the **Material Editor**, choose a material that uses the **Sky** shader.
9. Return to the **SkyBox** settings in the **Rollup Bar**, and click the assign (**<**) button to apply the selected material to the skybox.
10. (Optional) Configure the following parameters:
    - **Material low spec** – Select the skybox material to display at low resolution.
    - **Angle** – Specify the angle to rotate the skybox.
    - **Stretching** – Specify the amount to stretch the skybox texture to reduce the horizon line.
11. Close the **Material Editor**.

Changing Skybox Parameters

Do the following to modify the parameters of the dynamic sky in your level.

To change dynamic sky parameters

1. In Lumberyard Editor, choose **Tools**, **Other**, **Time of Day**.
2. In the **Time of Day** editor, in the **Time of Day Tasks** pane, under **Tasks**, click **Toggle Advanced Properties** to view all parameters.
3. In the **Parameters** pane, explore how the settings change the look of the scene.

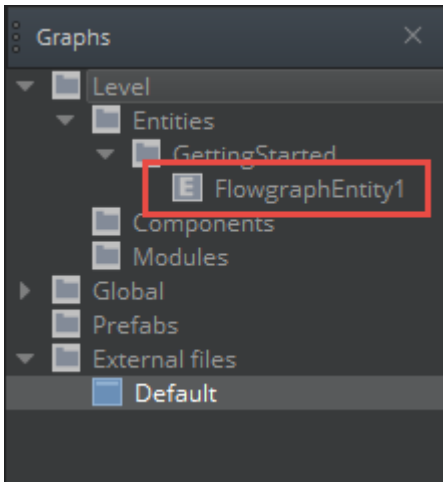
4. Close the **Time of Day** editor.

Switching Skyboxes Asynchronously

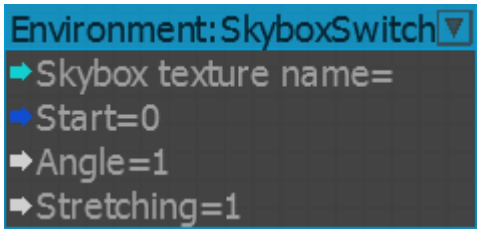
You can use the Flow Graph editor to perform asynchronous skybox switching.

To switch skyboxes asynchronously

1. In Lumberyard Editor, click **Tools**, **Rollup Bar**.
2. In the **Rollup Bar**, select your asset.
3. In Lumberyard Editor, click **Tools**, **Flow Graph**.
4. In the **Flow Graph** window, click **File**, **New**.
5. In the **Graphs** pane, select your entity.



6. Right-click the flow graph viewport 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

You can create realistic wind effects in your level environment.

Topics

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

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 **Tools**, **Other**, **Time Of Day**.
2. In the **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

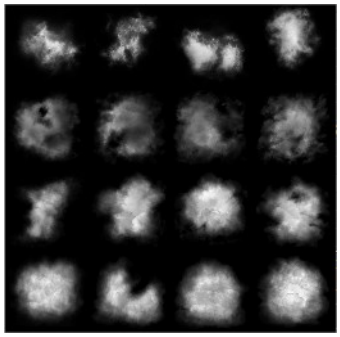
1. In **Rollup Bar**, under **Terrain**, click **Environment**.
2. Under **CloudShadows**, click **Cloud shadow texture** and the folder icon.
3. In **Preview**, select a suitable asset.
4. Drag the shadow to the desired location in your level.
5. Under **CloudShadows**, adjust the following parameters:
   - **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 **(...)** icon.
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. 1197).

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. 1361), 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 the **Rollup Bar**, under **Objects**, click **Area**, **AreaBox**.
2. Under **AreaBox**, click <**No Custom Material**>.
3. In the **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 **Game**, **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

   Specifies that each box has sprites.

   Density

   Defines the density of the cloud.

   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.

Adding Vegetation

You can add realistic trees, bushes, grasses, and other vegetation to your Lumberyard terrain.

Topics

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. 1406).
- 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.

Vegetatior	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

Vegetatior	Polygon Range	Texture Size	Proxies	Material IDs
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

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 **Tools**, **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 the **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

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.

Color	RGB Values	Bending Influence
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. 1196).

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. 1203) and Adding Detail (Wind) Bending Effects (p. 1203).

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. 1204).

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. 1196).

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. 1204).

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.

Using SpeedTree 8 for Lumberyard

SpeedTree 8 for Lumberyard is a procedural vegetation modeling program that takes advantage of the native Lumberyard vegetation tools. You can use SpeedTree 8 for Lumberyard to create and quickly iterate on trees, bushes, exotic alien foliage species, and other plant life.

The SpeedTree integration provides a simple workflow to add level of detail (LOD) and preview wind accurately with Lumberyard's wind physics integrated directly into the app.

The SpeedTree modeler exports Lumberyard meshes and materials into your game project. You can then use these meshes and materials the same way you would use any other mesh or vegetation object.

Download SpeedTree 8 for Lumberyard.

> **Note**
> You will need to create a SpeedTree account and activate your free license in order to download SpeedTree 8 for Lumberyard.

View the SpeedTree 8 documentation.

Lmbr.exe

> The Project Configurator is in preview release and is subject to change.

`Lmbr.exe` is a command-line tool for managing capabilities, game projects, and gems.

You can run `Lmbr.exe` from the `\dev\Tools\LmbrSetup\Win` directory at the root of your Lumberyard installation.

To see all possible commands that you can run, type `lmbr -help`.

```
Available commands:
lmbr
    engines
        list: List all engines currently installed.
    capabilities
        list: List out lumberyard capabilities.
        create: Create a new lumberyard capability.
        disable: Disables a Lumberyard capability.
        enable: Enables a Lumberyard capability.
        istagset: Query if tag is set in enabled capabilities.
    projects
        list: List all Projects.
        create: Create a new project.
        get-active: Get the name of the active project.
        set-active: Set a project as active.
        populate-appdescriptors: Populate appdescriptors files from Gems list.
    gems
        list: List all Gems installed (or enabled in a project).
        disable: Disable a Gem for a project.
        enable: Enable a Gem for a project.
        create: Create a new Gem.
        add-path: Add a path to use when resolving Gems
```

You can also use `-help` on other commands. For example, to see more information about Lumberyard capabilities, you can type the following: `lmbr capabilities -help` or `lmbr capabilities list -help`.

Capabilities Commands

Use the following commands to create and modify Lumberyard capabilities.

list

Lists all of the Lumberyard capabilities.

```
lmbr capabilities list
```

create

Creates a new Lumberyard capability. Include the ID, description, and tooltip.

```
lmbr capabilities create new_capability_name "This is the description." "This is the
 tooltip."
```

You can also specify the following arguments:

- `-help (bool)`: Displays help descriptions of available commands and options.
- `-tag (list)`: String tags associated with the capability.
- `-default (bool)`: True if capability is default; otherwise false.
- `-enable (bool)`: True if you want to enable the capability right away.

disable

Disables a Lumberyard capability.

```
lmbr capabilities disable lumberyard_capability
```

enable

Enables a Lumberyard capability.

```
lmbr capabilities enable lumberyard_capability
```

istagset

Queries if the tag is set in enabled capabilities.

```
lmbr capabilities istagset tag_of_capability_to_check
```

Project Commands

Use the following commands to create and modify game projects.

list

Lists all projects in the current engine directory.

```
lmbr projects list
```

create

Creates a new project using `EmptyTemplate`, which is located at `dev\ProjectTemplates\EmptyTemplate`, as a template.

```
lmbr projects create project_name
```

get-active

Displays the name of the active project.

```
lmbr projects get-active
```

set-active

Sets the active project for building and executing Lumberyard. This command modifies `_WAF_\user_settings.options` and `bootstrap.cfg` to reference the specified project.

```
lmbr projects set-active project_name
```

populate-appdescriptors

Populates the appdescriptors files from the gems list. If you use the Project Configurator to change gems, Lumberyard automatically updates the application descriptor files. If you manually edit a `gems.json` file, run this command to update these two application descriptor files in a project's asset directory:

- `dev/`*`project_asset_directory`*`/Config/Game.xml`
- `dev/`*`project_asset_directory`*`/Config/Editor.xml`

```
lmbr projects populate-appdescriptors
```

Gem Commands

Use the following commands to create gems and modify a project's use of gems.

list

Lists all gems that are installed or enabled in the specified project.

```
lmbr gems list (-project project_name)
```

disable

Disables the specified gem in the specified project. If `-disable-deps` is specified, all dependencies of the gem are also disabled.

```
lmbr gems disable project_name gem_name (-disable-deps)
```

enable

Enables the specified gem in the specified project. If a version is specified, it's used, otherwise the latest version installed is used.

```
lmbr gems enable project_name gem_name  (-version version)
```

create

Creates a gem with the given name. If you specify `id` or `version`, those values are used. If you do not specify `-out-folder`, `name` is used.

```
lmbr gems create gem_name (-version version) (-out-folder gems\relative_folder)
```

Mobile Support

Mobile support is in preview release and is subject to change.

You can use Lumberyard to build your games for Android devices and iOS devices that use the A8 GPUs. 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.

Lumberyard supports the following minimum specifications for Android and iOS:

	Android	**iOS**
CPU	ARM quad-core or newer (for example, ARM Krait 400, Cortex-A53)	ARM v8 or newer
GPU	Adreno 330, Mali-T760 or newer Supports OpenGL 3.0	PowerVR G6430 (quad-core) or newer Supports Metal
OS	5.1 or newer	iOS 7.0 or newer
Example devices	Samsung Galaxy Note 4 LG Nexus 5 Kindle Fire HDX	iPhone 5s and newer

Topics

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 Samsung Galaxy Note 4, LG Nexus 5, and Kindle Fire HDX. For more information, see Mobile Support (p. 1211). 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

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. 20).
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 OS-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:

```
lmbr_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 `/Tools/LmbrSetup/Mac` 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. 20).
4. In a command line window, change to the `/lumberyard/dev` directory.
5. To initialize the build system, run the following command:

   ```
   sh lmbr_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 lmbr_waf.sh configure
    ```

Topics

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. 1211).
- 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 `lmbr_waf configure` command:

   - On a PC, run the following command: `lmbr_waf.bat configure`
   - On a Mac, run the following command: `./lmbr_waf.sh configure`

   If an error occurs while configuring your game project for Android, a warning message similar to the following is displayed:

   ```
   [WARN] Android settings not found in SamplesProject/project.json, skipping.
   ```

7. You can view the contents of the generated Android project in the Android Studio project directory. For example, `\dev\Solutions\LumberyardAndroidSDK\SamplesProjectLauncher`.
8. Build and test your game project on Android. For information, see Building Android Games (p. 1219).

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

Description: Used for generating the project specific Java activity class and in the AndroidManifest.xml

Tag name: `"package_name"`

Type: String in dot-separated format

Example: `"com.mycompany.mygame"`

Manifest code version number

Description: Internal application version number. Used to set the `android:versionCode` tag in `AndroidManifest.xml`.

Tag name: `"version_number"`

Type: Whole number value

Default: `1`

Manifest version name

Human readable version number. Used to set the `android: versionName` tag in `AndroidManifest.xml`.

Tag name: `"version_name"`

Type: String

Example: "1.0.0"

Orientation

Description: Orientation of the Android application. Used to set the `android:screenOrientation` tag in `AndroidManifest.xml`.

Tag name: `"orientation"`

Type: String

Valid values: See the Android Developers page for valid values.

Default: "landscape"

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

Description: Forces the assets to be packed in the APK in non-release builds

Tag name: `"place_assets_in_apk"`

Type: Whole number value

Valid values: `0` (No) or `1` (Yes)

Default: `0`

Google Play application license key

Description: Application license key provided by Google Play. Required for using APK Expansion files or other Google Play Services.

Tag name: `"app_public_key"`

Type: String (Base64-encoded RSA public key)

Default value: `"NoKey"`

Application specific salt value for (un)obfuscation when using APK Expansion files

Tag name: `"app_obfuscator_salt"`

Type: String containing a series of random bytes

Default value: `""`

Specify whether to use Main APK Expansion

Description: Toggles APK Expansion file mode in release builds.

Tag name: `"use_main_obb"`

Type: String value containing either `"true"` or `"false"`

Default Value: `"false"`

Specify whether to use the "Patch" APK Expansion file

Description: Toggles APK Expansion file mode in release builds.

Tag name: `"use_patch_obb"`

Type: String value containing either `"true"` or `"false"`

Default Value: `"false"`

Force APK Expansion file mode in non-release builds

Tag name: `"enable_obb_in_dev"`

Type: String value containing either `"true"` or `"false"`

Default Value: `"false"`

Enable or disable the screen wake lock

Description: Toggles whether device will go to sleep while the application is running.

Tag name: `"enable_keep_screen_on"`

Type: String value containing either `"true"` or `"false"`

Default Value: `"false"`

RC job override for generating the normal PAK files used in release builds

Tag name: `"rc_pak_job"`

Type: String containing the XML file name relative to `\dev\Bin64\rc`

o Default Value: `"RcJob_Generic_MakePaks.xml"`

RC job override for generating the APK Expansion file(s)

Tag name: `"rc_obb_job"`

Type: String containing the XML file name relative to `\dev\Bin64\rc`

Default Value: `"RCJob_Generic_Android_MakeObb.xml"`

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,

    "app_public_key" :

        "MIIBIjANBgkqhkiG9w0BALuMbErYaRdAMIIBCgKCAQEAjvkl+K7rVA
        SNkLExAmPlEoWwsxCX1vx7uV3IIH5CQIZBRGT8KeYr6ThWlIPhSMKMIm
        j7KxjdYcil8J0rwrVL3cmAYdMM+02ntnBEemGvRVOKxkDaFc5Fw6tJVv
        3SJ6UyjVtehB7tJupaYdfFe9SVhW0xJZu2YsZLuMbErYaRdrrgXU2Upr
        547mxuyEHJ7jG7YFVrQgxou1W/71QnExAmPlExi6mlsUJBFN4xADikNW
        ZDlI70iHF6ZYyOspZCbVZ9DScN+D5oS3KeY/KKd5WOU6BB8NmTY5VZVd
        UOd4VPRXrYMnRY7FjZJMPujLNvlrAJs5H/G+0wUTR4SI61AiGJiQIDAQ
        AB",

        "app_obfuscator_salt" : "8d87473f5b24852836d0652abbd9e9b9869c208",
        "use_main_obb" : "true",
        "use_patch_obb" : "false",
        "enable_obb_in_dev" : "false",
        "enable_keep_screen_on" : "true",
        "rc_pak_job" : "RcJob_MakeCustomPaks.xml",
        "rc_obb_job" : "RcJob_MakeCustomAndroidObb.xml"
},
```

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 `lmbr_waf configure` command:

- On a PC, run the following command: `lmbr_waf.bat configure`
- On a Mac, run the following command: `./lmbr_waf.sh configure`

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. 1224).
- You are using Android NDK r11 or above. For information, see Android Support (p. 1211).

Building Android Games Using the Lumberyard Editor Deployment Tool Plugin

Lumberyard Editor includes a plugin that allows you to build and deploy Android applications. You can access the plugin by clicking **Tools**, **Other**, **Deployment Tool**.

> **Note**
> If you change your default project, you must ensure the new default project is set for **enabled_game_options** in the `user_settings.options` file (located in the `\_WAF_` directory at the root of your Lumberyard installation). Then you can run the `lmbr_waf.bat configure` command in a command line window prior to starting Lumberyard Editor and using the **Deployment Tool**. This allows the appropriate files to be processed and created for your project.

To build and deploy your game for Android using the Deployment Tool

1. In Lumberyard Editor, click **File**, **Project Settings**, **Deployment Tool**.
2. If this is your first time building and deploying to an Android device, do the following to ensure that the Deployment Tool will build the game before deployment and will install the game and APK to the Android device before attempting to launch it:
   a. For **Deploy Options**, select the **Build Game** check box.
   b. Click **Advanced Options** and select the **Install the Executable** check box.
3. (Optional) Use the **File Transfer Option** drop-down list to specify how the Deployment Tool should deploy the assets for the game:
   - **Push Files** – Assets are copied and pushed to the Android device before the game runs. The game will read all assets from the Android device, allowing the game to run faster. This option disables the ability to hot reload assets.
   - **Use Virtual File System** – Assets remain on the PC and the game retrieves the assets at runtime as needed. This option allows assets that support hot reloading to automatically update in-game if changes are detected on the PC.
4. Click **Deploy** to build and deploy the game to the Android device, start the remote shader compiler, and launch the game on the Android device. You will see the process output in the **Deployment Console**.
5. (Optional) After your initial deployment, if you plan to change only art and level assets, do the following to decrease the deployment time:

a. For **Deploy Options**, clear the **Build Game** check box.
b. Click **Advanced Options** and clear the **Install the Executable** check box.

Clearing these options will prevent the Deployment Tool from building and installing the executable, which is only necessary if you change code.

> **Note**
> If you add new gems to your game project, you must select the **Build Game** and **Install the Executable** check boxes in order to include the functionality of those gems.

Setting Deployment Tool Options

The Deployment Tool provides the following basic options that you can set to build and deploy your game:

- **Hardware Platform** – Android is the only device supported.
- **Select Build Configuration** – Choose **Debug**, **Profile**, or **Release**. In most cases **Profile** is the best option to use for performance and ease-of-use. To use **Release**, you must have already built the `.pak` files for a release build.
- **Select Compiler** – Choose **Clang** or **GCC**. We recommend using Clang. GCC may no longer be supported in a future release.
- **File Transfer Option** – Choose how the Deployment Tool handles game assets:
  - **Push Files** – Assets are copied and pushed to the Android device before the game runs. The game will read all assets from the Android device, allowing the game to run faster. This option disables the ability to hot reload assets.
  - **Use Virtual File System** – Assets remain on the PC and the game retrieves the assets at runtime as needed. This option allows assets that support hot reloading to automatically update in-game if changes are detected on the PC.
- **Build Game** – Select to build the game before deployment and launch.

The Deployment Tool provides the following advanced options that you can set for finer control:

- **Clear Device Before Deploying** – Select to uninstall the previous version of the game and delete any assets that were copied to the device prior to deployment.
- **Install the Executable** – Select to install the game on the Android device. If cleared, the Deployment Tool will only process the assets.
- **VFS Remote IP** and **Port** – Configure the IP address and port for the virtual file system (VFS) to use when communicating with the Android device. We recommend using the default values.

  > **Note**
  > If you change the port number in the Deployment Tool and then click **Deploy**, the tool will update the `bootstrap.cfg` file (located in the `\dev` directory at the root of your Lumberyard installation) with the new port value. To ensure that VFS works properly and the asset processor can listen to the new port, you must restart the asset processor after changing the port number. Alternatively, you can change the port number in the `bootstrap.cfg` file before restarting the asset processor, and then open the Deployment Tool.
- **Shader Compiler Settings**:
  - **Connect through Asset Processor** – Select to enable the asset processor to process shader compiler requests. The game will not connect directly to the remote shader compiler.
  - **Shader Compiler IP** and **Port** – Provide the IP address of the remote shader compiler to which the game should connect. The game will not use the asset processor to process shader compiler requests.

Building Android Games Using Clang

You can build your game for Android using Clang. Building for Android (ARM) 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. 1211).
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, run the following command: `lmbr_waf.bat build_android_armv7_clang_debug -p all`
     - On a Mac, run the following command: `sh lmbr_waf.sh build_android_armv7_clang_debug -p all`
   - To build profile
     - On a PC, run the following command: `lmbr_waf.bat build_android_armv7_clang_profile -p all`
     - On a Mac, run the following command: `sh lmbr_waf.sh build_android_armv7_clang_profile -p all`
   - To build release
     - On a PC, run the following command: `lmbr_waf.bat build_android_armv7_clang_release -p all`
     - On a Mac, run the following command: `sh lmbr_waf.sh build_android_armv7_clang_release -p all`
4. Debug your application. For information, see Android Debugging (p. 1226).

Building Android Games Using GCC

You can build your game for Android using the GNU Compiler Collection (GCC).

> **Note**
> Google will deprecate the GCC in the near future.

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, run the following command: `lmbr_waf.bat build_android_armv7_gcc_debug -p all`
     - On a Mac, run the following command: `sh lmbr_waf.sh build_android_armv7_gcc_debug -p all`
   - To build profile
     - On a PC, run the following command: `lmbr_waf.bat build_android_armv7_gcc_profile -p all`
     - On a Mac, run the following command: `sh lmbr_waf.sh build_android_armv7_gcc_profile -p all`
   - To build release

- On a PC, run the following command: `lmbr_waf.bat build_android_armv7_gcc_release -p all`
- On a Mac, run the following command: `sh lmbr_waf.sh build_android_armv7_gcc_release -p all`

3. Debug your application. For information, see Android Debugging (p. 1226).

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 macOS.

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 macOS 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, run the following command: `sudo ln -s /Volumes/Cache Cache`

   If prompted, type the password for your macOS 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.

You can use a whitelist to specify the IP addresses that are allowed to connect to your remote shader compiler. For information, see Creating a Whitelist for the Remote Shader Compiler (p. 1358).

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 type one of the following:

- If you are using Visual Studio 2015, run this command:

```
lmbr_waf build_win_x64_vs2015_profile -p all --targets=CrySCompileServer
```

- If you are using Visual Studio 2013, run this command:

```
lmbr_waf build_win_x64_vs2013_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. 1219)
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, type a command that provides the name of the game project for which to build the shaders as an argument: `BuildShaderPak_ES3.bat <Game Project Name>`.

   For example, to build the shaders for the Samples Project, type the following in a command line window:

   ```
   BuildShaderPak_ES3.bat SamplesProject
   ```

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\es3\Game Project Name\` 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. 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`).
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.

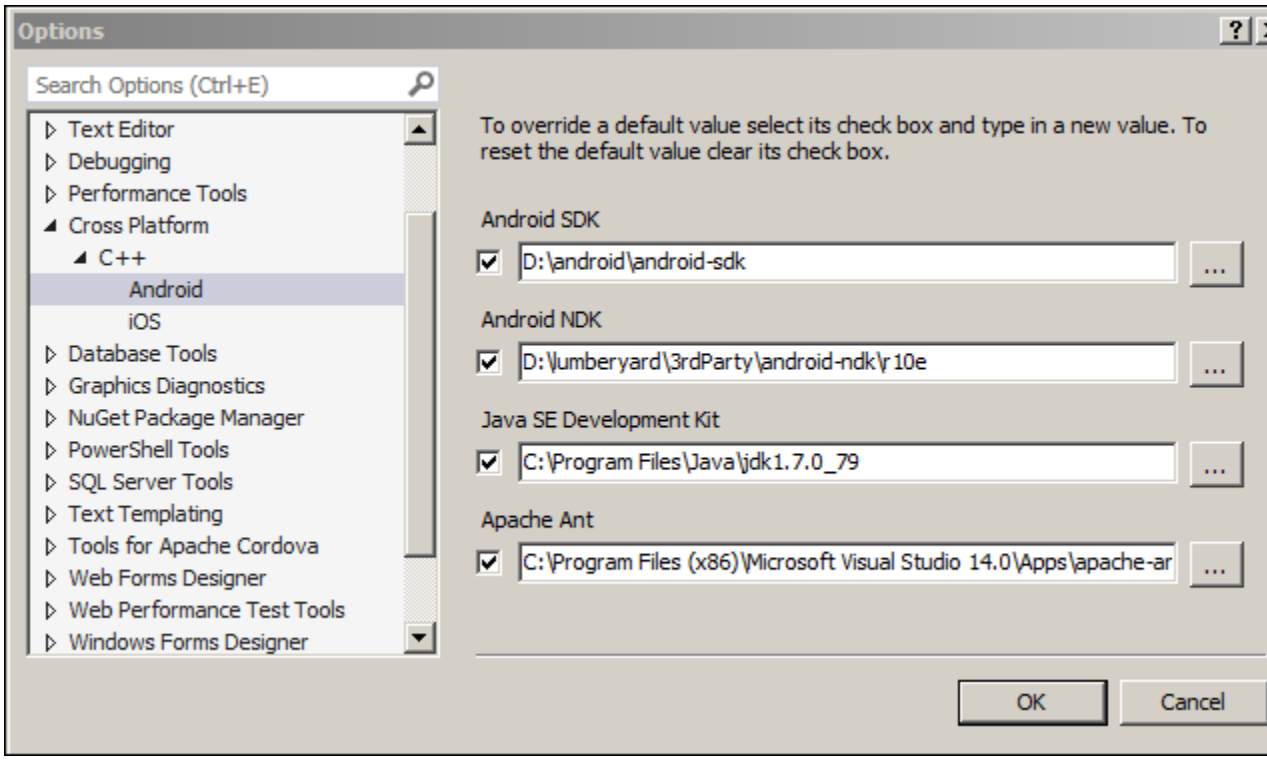
Android Debugging

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

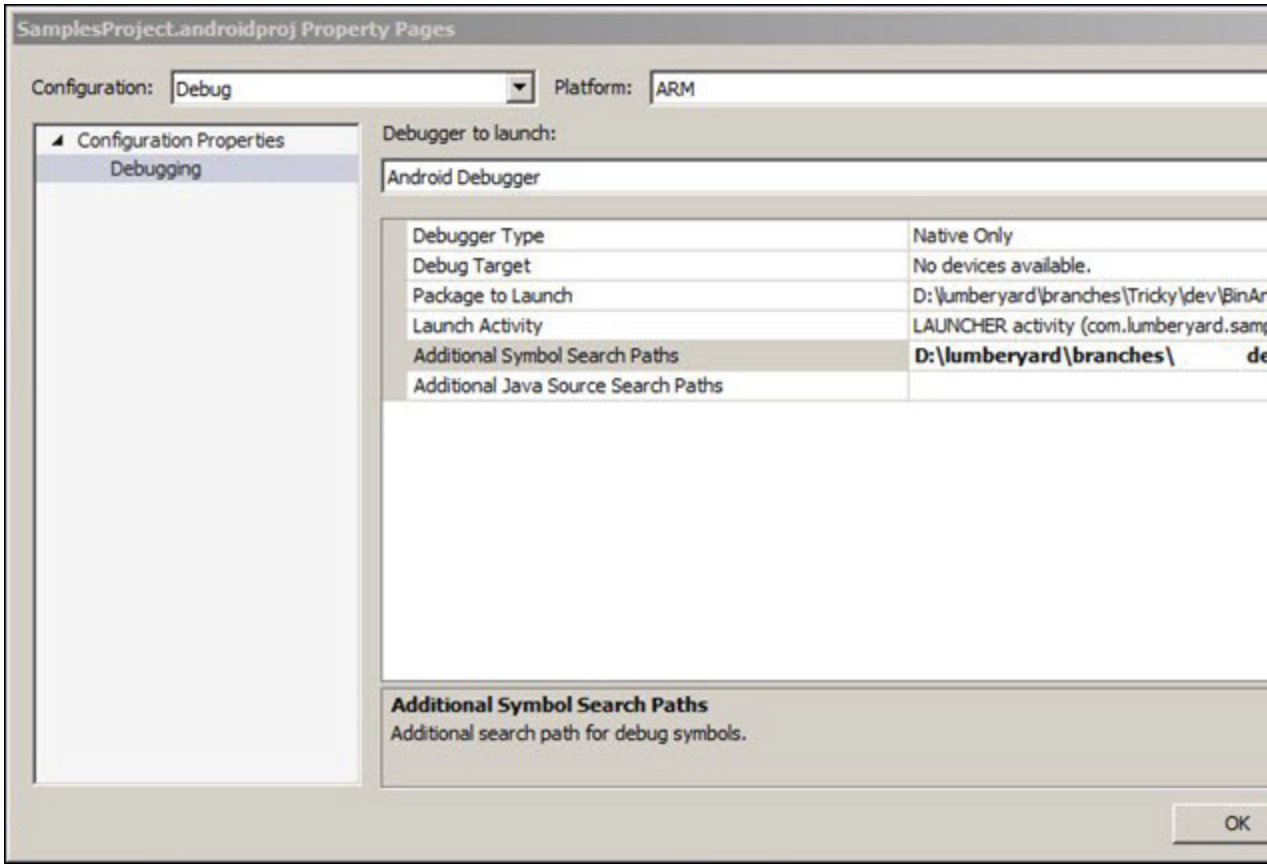
1. In the Visual Studio 2015 installer, select **Cross platform tools for C++ development**. Follow the on-screen instructions to complete the installation.
2. In Visual Studio 2015, click **Tools**, **Options**.
3. In the left pane of the **Options** dialog box, expand **Cross Platform**, **C++**, and 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. 1222).

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. 1224).

You can deploy your game to Android devices using the Remote Console.

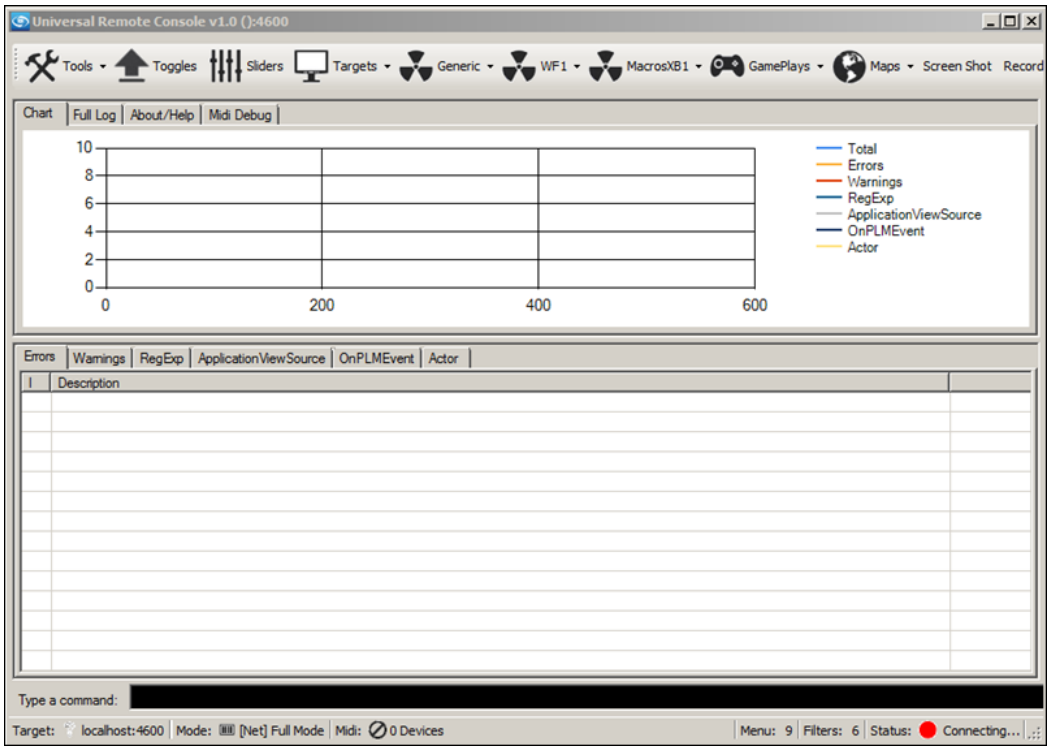
Once you have deployed your game, see Running Android Games (p. 1229).

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).



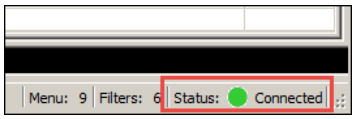
2. On the **Full Log** tab, view the output from the runtime engine's logging system.
3. Start running a Lumberyard application on your Android device.
4. In the Remote Console, click **Targets** and then type the IP address of the Android device for **Custom IP**.
5. (Optional) If your network allows 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. 1227).

To run the game on release, certain read-only console variables (cvars) must load during runtime in order for the mobile environment to configure properly. By default, read-only cvars cannot be modified when running on release. You must modify this behavior by editing the `IConsole.h` file (located in the `\dev\Code\CryEngine\CryCommon` directory) to set the define `ALLOW_CONST_CVAR_MODIFICATIONS` (line 39) to `1`.

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 **es3**. 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 Advanced_RinLocomotion level.
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 Rin 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
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.
   - If you have a carrier-locked device, run the following command:

```
adb push bootstrap.cfg /storage/sdcard0/SamplesProject/bootstrap.cfg
```

   - If you have an unlocked device, run the following command:

```
adb push bootstrap.cfg /storage/emulated/0/SamplesProject/bootstrap.cfg
```

   Note
   You are responsible for complying with the terms applicable to your Android device, including restrictions on unlocking the Android bootloader.
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. 1222).
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 **es3**. 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. 1211).
- 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

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: `lmbr_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 `lmbr_waf android_studio` or `lmbr_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)
```

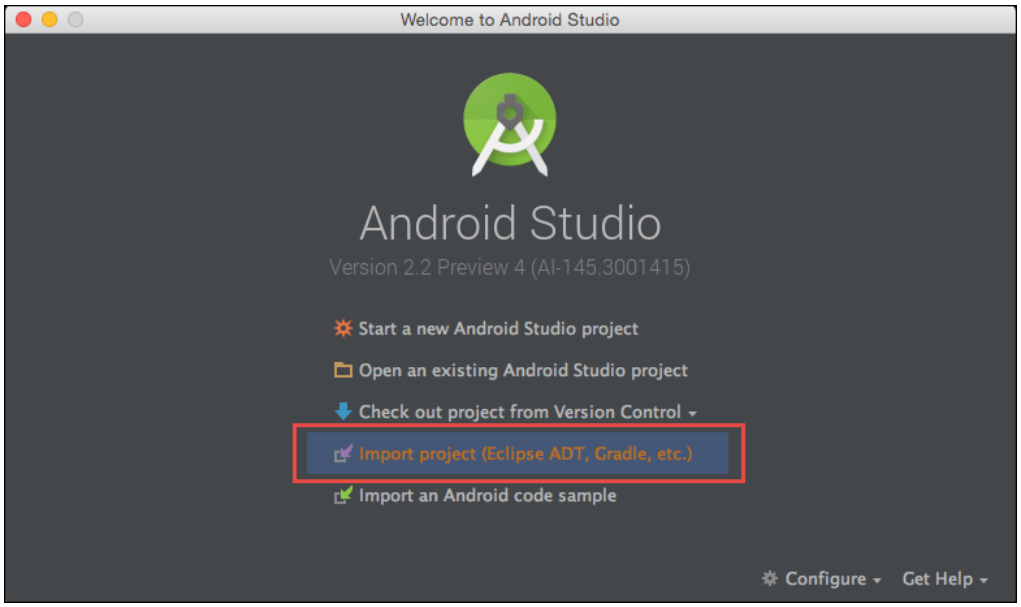
Importing Your Lumberyard Project into Android Studio

Mobile support is in preview release and is subject to change.

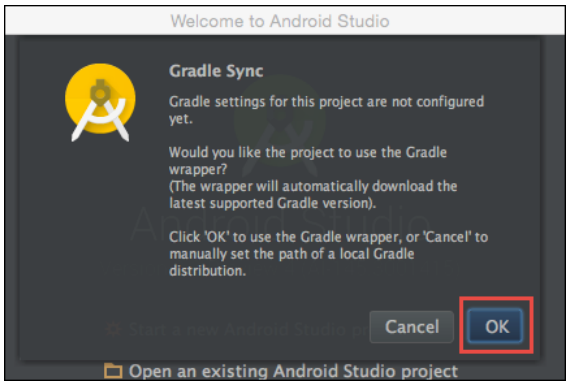
After you create a Lumberyard project, you can import it into Android Studio.

To import your Lumberyard project into Android Studio

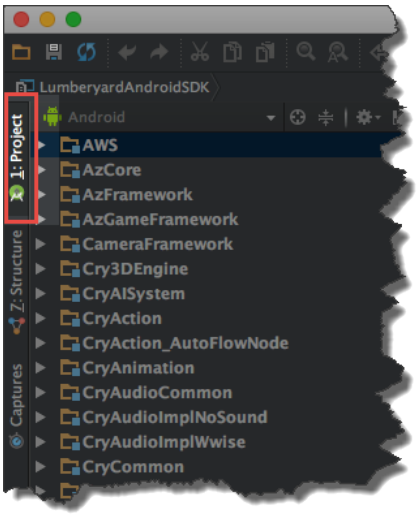
1. Open Android Studio.
2. On the **Welcome to Android Studio** screen, click **Import project (Eclipse ADT, Gradle, etc.)**.



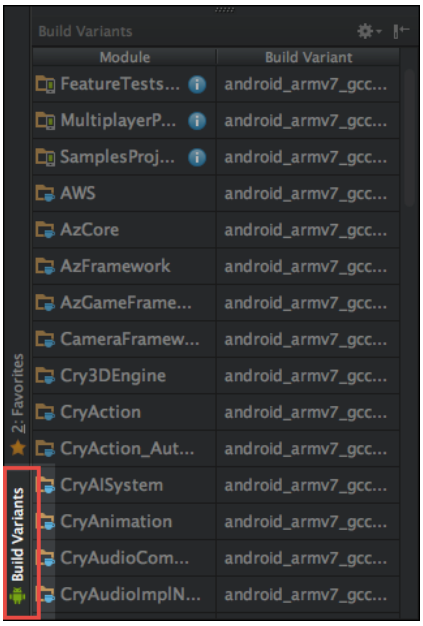
3. Locate the Android Studio project that was created when you ran the `lmbr_waf configure` command. The default location is `\lumberyard\dev\Solutions\LumberyardAndroidSDK`.
4. In the **Gradle Sync** dialog box indicating that Gradle settings are not configured for this project, click **OK**.



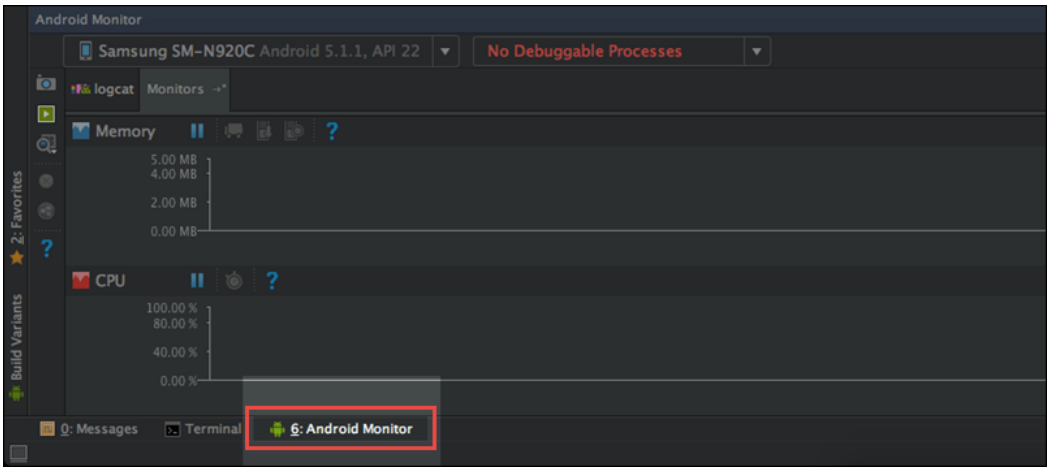
5. When the project has been imported and opens, do the following:
   - Click **Project** to view the project source view pane.



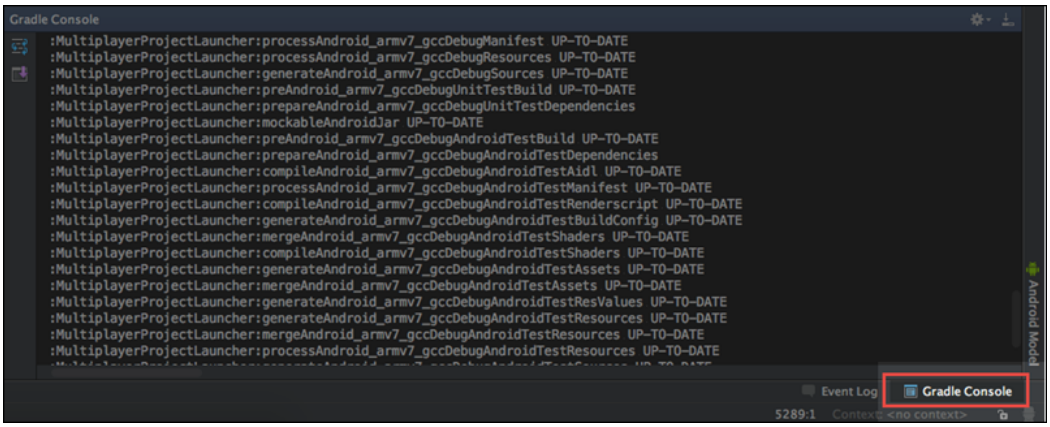
- Click **Build Variants** to view the build targets and change the build configuration for your target project. This impacts the FeatureTestsLauncher, MultiplayerProjectLauncher, and SamplesProjectLauncher only. The build configuration for all other targets during the build process is ignored.



- 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
   - MultiplayerProjectLauncher
   - MultiplayerProjectLauncher-native
   - SamplesProjectLauncher-native

The Canary version might display the following:

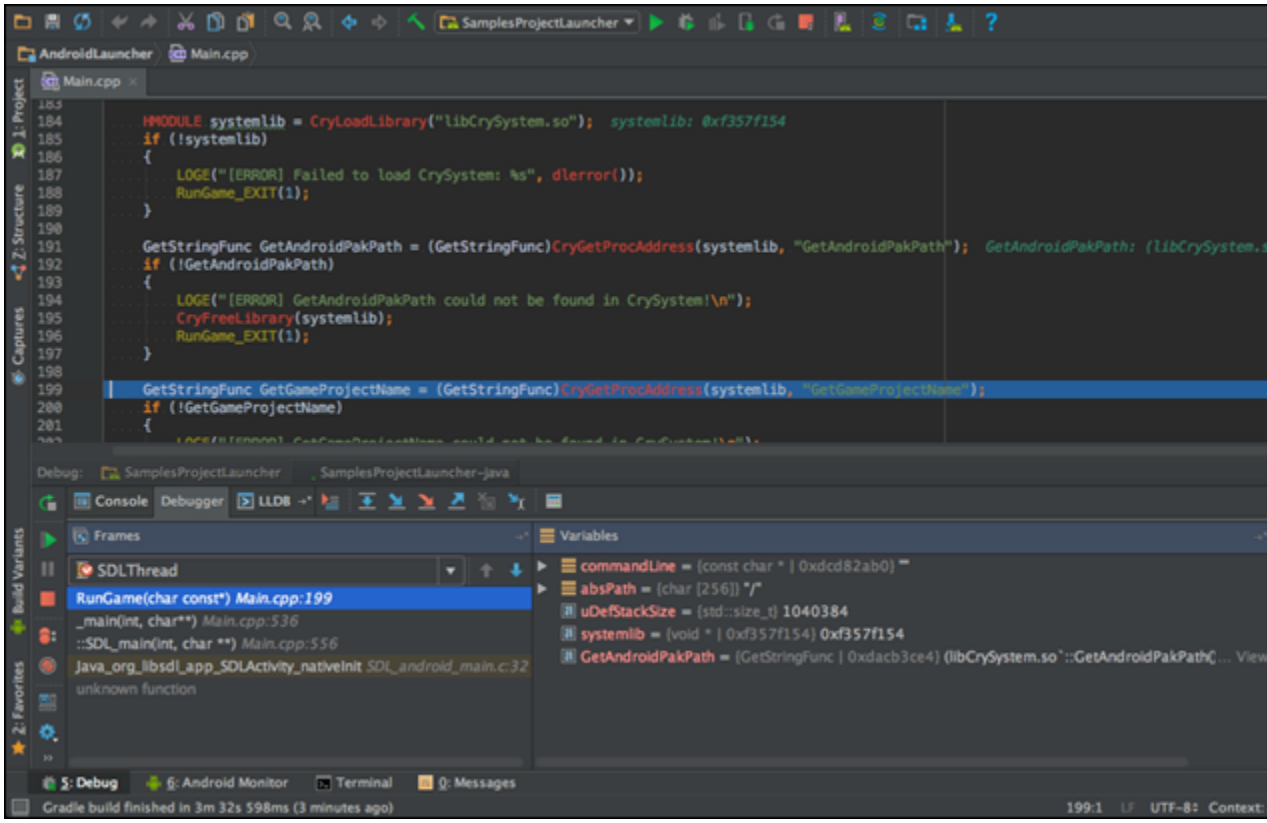
- SamplesProjectLauncher
- MultiplayerProjectLauncher

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.

Using the AWS Device Farm for Android Builds

Mobile support is in preview release and is subject to change.

You can use the AWS Device Farm to test your Lumberyard game across a range of devices. This topic demonstrates how to run an Android build created with Lumberyard on the AWS Device Farm.

Topics

Prerequisites

To run a Lumberyard game build on the AWS Device Farm, you must have the following:

- AWS account
- Familiarity with the AWS Management Console
- Understanding of Amazon EC2 instances, VNC, security groups, and the Lumberyard tool chain

If you prefer not to use your PC on a public IP address, you can use Amazon Elastic Compute Cloud (Amazon EC2) to run the Lumberyard shader compiler for mobile development.

Creating the Build

Follow the steps below to create your build to deploy to the AWS Device Farm.

1. Navigate to the `\dev` directory for your game project. For example, the SamplesProject directory is `\dev\SamplesProject`.
2. Edit the game's `project.json` file to set `place_assets_in_apk` (under `android_settings`) to **`1`**.
3. Navigate to the `\lumberyard\dev` directory.
4. Edit the `system_android_es3.cfg` file to set the IP address for the remote shader compiler to point to the Amazon EC2 instance or your computer's public IP address.
5. Build the game.

Deploying to the AWS Device Farm

Follow the steps below to deploy your build to the AWS Device Farm.

1. Open the AWS Management Console.
2. On the **Device Farm** tab, do the following:
   a. Click **Create a new project** and type a name for your project.
   b. Click **Create a new run** and then click the button with the Android and iOS logo.
   c. Click **Upload** and select the `.apk` file that you created previously.
   d. When the upload completes, type a name for the run.
   e. On the **Configure a test** page, click **Built-in Fuzz**.

   f. Allow the test to run for 10-15 minutes by setting the **Event count** to **600** and **Event throttle** to **1000**.
   g. For the device pool, click **Create a new device pool**.
   h. Search for and select **Samsung Galaxy S7**. Type the name and description for the pool and save it.
   i. Click through the remaining confirmation screens.
3. When the tests are complete, click to the **Screenshots** tab to see your deployed content.

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 5s and newer. For more information, see Mobile Support (p. 1211). 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 a Terminal window on your Mac.

Setting Up Your Mac

Download and extract Lumberyard on your Mac using the Lumberyard Mac 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 on your Mac

1. On your Mac, in a Terminal window, navigate to the `/dev/Tools/LmbrSetup/Mac/` directory at the root of your Lumberyard installation.
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** and **Compile the Game Code**. 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. 20).
5. To get started with building and deploying your Lumberyard game for iOS, do the following:
   a. Build your iOS game executable. For information, see Configuring and Building iOS Games (p. 1240).
   b. Build the assets to include with your iOS game. For information, see Building Game Assets for iOS Games (p. 1241).
   c. Launch the remote shader compiler. For information, see Running the Remote Shader Compiler (p. 1243).
   d. Deploy your iOS game. For information, see Deploying iOS Games (p. 1243).
   e. Prepare your iOS game for distribution. For information, see Creating iOS Release Builds for Distribution (p. 1244).

Topics

Configuring and Building iOS Games

Mobile support is in preview release and is subject to change.

Configure and build various targets of your game before deploying your games to iOS devices.

When you build your game project using Waf, you must specify the enabled game project by doing one of the following:

- Provide the game project name in the build command: `--enabled-game-projects=[project name]`
- Edit the `user_settings.options` file (located in the `\lumberyard\dev\_WAF_` directory) to set `enabled_game_projects` to the name of your project.

```
[Game Projects]
enabled_game_projects = MyProject
```

You can also modify the `configure` command to include the following option, which automatically updates the `user_settings.options` file: `--update-settings=True`

To configure your game to build for iOS

1. On your Mac, in a Terminal window, navigate to the `/dev` folder at the root of your Lumberyard installation.

2. To generate an Xcode project and prepare the Lumberyard build system to build your iOS app, run the following command: `sh lmbr_waf.sh configure --enabled-game-projects=[project name]`

   For example, if you are using the Samples Project, run the following command: `sh lmbr_waf.sh configure --enabled-game-projects=SamplesProject`

To build your game for iOS

1. On your Mac, in a Terminal window, navigate to the `/dev` folder at the root of your Lumberyard installation.
2. Build various targets of your game:
   - To build debug, run the following command: `sh lmbr_waf.sh build_ios_debug -p all`
   - To build profile, run the following command: `sh lmbr_waf.sh build_ios_profile -p all`
   - To build release, run the following command: `sh lmbr_waf.sh build_ios_release -p all`

Building Game Assets for iOS Games

Mobile support is in preview release and is subject to change.

When you build an iOS game in 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 *engine_root*`\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.

> **Note**
> If you make changes to your game in Lumberyard Editor on a Windows system, you must copy the updated assets to a macOS system using your preferred method. For example, you can use source control. For information, see Using the Perforce Plugin with Lumberyard (p. 26).

To build iOS game assets on a macOS system

1. On your macOS system, close all instances of the Asset Processor.
2. Edit the `bootstrap.cfg` file (located in the `/`*engine_root*`/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 `/`*engine_root*`/dev` directory) to uncomment `ios=enabled` (remove the preceding semicolon) and to comment out `pc=enabled` (add a preceding semicolon). Save the file.
4. In a command line window, navigate to the `/`*lumberyard_version*`/dev/BinMac64` directory and run the Asset Processor (GUI or batch version) to process and build your game assets.

   > **Note**
   > Symbolic links are not supported when you use the Asset Processor on macOS. To ensure that the Asset Processor works properly on macOS, follow these guidelines:
   >
   > - Do not use a symbolic link for your cache directory when you store compiled assets in a central location.
   > - Do not store your source project assets in a symbolic link directory.
   > - Use a unique cache directory. Do not share the cache directory with a Windows system that is also running the Asset Processor.

If you prefer to run the Asset Processor on your Windows system, you must share the cache folder with your assets between your Windows and macOS systems. For instructions, see Sharing Game Assets Between Windows and macOS Systems (p. 1242).

Sharing Game Assets Between Windows and macOS Systems

Mobile support is in preview release and is subject to change.

After you build your iOS game assets, you can share the `cache` directory between Windows and macOS systems. This ensures that changes you make in Lumberyard Editor on Windows are automatically retrieved by your macOS system. If you choose to run the Asset Processor on macOS, we do not recommend sharing assets between platforms due to potential compatibility issues with sharing asset database files.

To set up asset sharing in Windows

1. In Windows, navigate to the *`lumberyard_version`*`\dev` directory.
2. Right-click the `cache` directory and choose **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 macOS system.

To view shared assets in macOS

1. On your macOS system, in the Finder, click **Go**, **Connect to Server**.
2. For the **Server Address**, type **`smb://`*`IP address or DNS name of your Windows system`*`/Cache`**
3. Click **Connect**.
4. (Optional) Configure your system preferences to automatically connect to the shared directory when macOS starts:
   a. Open **System Preferences**, **Users & Groups**, **Login Items**.
   b. In the **Login Items** dialog box, click **+** to add a new login item.
   c. In the **Shared** pane, locate and select your Windows system. In the right pane, select your shared `cache` directory and click **Add**.
5. In a terminal window, navigate to the *`lumberyard_version`*`/dev` directory.
6. To create a symbolic link to the shared `cache` directory, run the following command: `sudo ln -s /Volumes/Cache Cache`

   If prompted, type the password for your macOS login.

Running the Remote Shader Compiler

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.

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.

You can use a whitelist to specify the IP addresses that are allowed to connect to your remote shader compiler. For information, see Creating a Whitelist for the Remote Shader Compiler (p. 1358).

To allow your iOS game to connect to the shader compiler on your PC

1. On your PC, launch the shader compiler (located in the `\dev\Tools\CrySCompileServer\x64\profile` directory).
   - If you are using Visual Studio 2013, launch `CrySCompileServer_vc120x64`.
   - If you are using Visual Studio 2015, launch `CrySCompileServer_vc140x64`.
2. On your PC, edit the `system_ios_ios.cfg` file (located in the `\dev` directory) to set the **r_ShaderCompilerServer** variable to the IP address of the PC on which you are running the shader compiler. The default value is **localhost**.
3. If it is not already running, launch the Asset Processor on your PC.

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` directory) is running on your PC. For more information, see Running the Remote Shader Compiler (p. 1243).

To deploy your game to an iOS device

1. On your Mac, open the Xcode solution that you generated (located in the `Solutions` folder in the directory where you installed Lumberyard). For example, the file could be called `LumberyardiOSSDK.xcodeproj`.

   > **Note**
   > In the **Autocreate Schemes** dialog box, select **Automatically Create Schemes**. This will happen the first time that you open your project.

2. 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.

3. (Optional) Load different levels by editing the `SamplesProject/autoexec.cfg` file and running the game from Xcode again. iOS supports the Advanced_RinLocomotion level.
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 Rin 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.

Creating iOS Release Builds 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 packed assets and cached binary shaders (for information, see Running the Remote Shader Compiler (p. 1243)), 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.

Building Asset .Pak Files

Use a command line prompt and batch file to build a `.pak` file that includes your assets.

To create required assets to run a release build for iOS

1. On your PC, in a command line window, navigate to the `dev` directory of your Lumberyard installation.
2. Run the `BuildSamplesProject_Paks_iOS.bat` file. The `.bat` file generates a `\Cache\game project name\ios_paks` directory that includes the files required to run your game, excluding shaders and executables.

   > **Note**
   > The `BuildSamplesProject_Paks_iOS.bat` file will only work with the levels in the `SamplesProject` directory. If you are building a different project, you can copy the `.bat` file and modify it to reference your project. Each project needs its own `.bat` file.

Building Shader .Pak Files

Explore every area in your game level to capture all shader permutations, and then use a command line prompt and batch file to build a `.pak` file that includes your shaders.

To generate and retrieve shaders for your iOS game

1. Build, deploy, and run your game on an iOS device. For information, see Configuring and Building iOS Games (p. 1240) and Deploying iOS Games (p. 1243).
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/Library/Application Support/` 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.

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, type a command that provides the name of the game project for which to build the shaders as an argument: `BuildShaderPak_Metal.bat` *`<Game Project Name>`*

   For example, to build the shaders for the Samples Project, type the following command in a command line window

   ```
   BuildShaderPak_Metal.bat SamplesProject
   ```

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\ios\`*`game project name`*`\` 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 release game to run the shaders from the .pak files

1. 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`).
2. When the shader `.pak` files are in the correct cache location, edit the scheme in XCode to use **release** as the build configuration.
3. Run the build process.

Deploying and Archiving Release Builds from XCode

After generating all required asset and shader `.pak` files, you can use Xcode to build, deploy, run, and archive the release version of your iOS game. Use the same method that you would use for any Xcode project. For more information, see Prepare for app distribution in the *XCode Help*.

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 your PC and view changes immediately on the iOS device without needing to redeploy the game.
- 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 Windows and macOS Systems (p. 1242).

To build the Asset Processor on your Mac

- In a command line window, navigate to the `/lumberyard/dev` directory and run the following command: `lmbr_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\CrySShaderCompiler\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 Configuring and Building iOS Games (p. 1240) and Deploying iOS Games (p. 1243). 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 run the following command: `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.

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 Running the Remote Shader Compiler (p. 1243) 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.

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, run the following command: `lmbr_waf.sh clean_ios_debug`
- To build profile, run the following command: `lmbr_waf.sh clean_ios_profile`
- To build release, run the following command: `lmbr_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.

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.

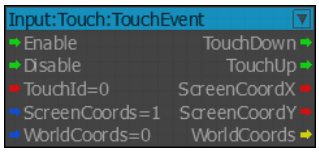
Input

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. 754).

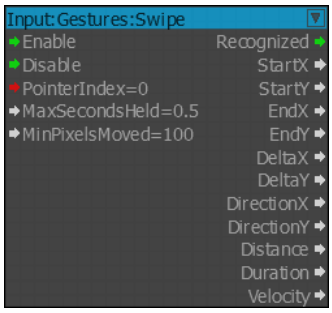
If you have created your game logic to use mouse-based input, Lumberyard provides a way to emulate mouse events using the primary touch on mobile devices. To enable the ability to emulate

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)

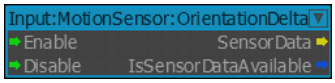


Gestures that require only a single touch to be recognized (Tap, Drag, Hold, and Swipe) function the same when using mouse input on PC. Multi-touch gestures (Pinch and Rotate) can only be recognized through multiple, simultaneous touches.

Motion Sensors

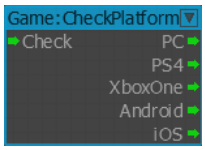
You can use a range of MotionSensor nodes in the Flow Graph Editor to return motion sensor data generated by mobile devices from the accelerometer, gyroscope, and magnetometer. Each flow node returns a vector (or quaternion for orientation) for the device's:

- 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	applicationWillResignActive	onPause()
OnApplicationUnconstrained	applicationDidBecomeActive	onResume()
OnApplicationSuspended	applicationDidEnterBackground	onPause()
OnApplicationResumed	applicationWillEnterForeground	onResume()
OnMobileApplicationWillTerminate	applicationWillTerminate	onDestroy()
OnMobileApplicationLowMemoryWarning	applicationDidReceiveMemoryWarning	onLowMemory()

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.

Running the Shader Compiler on Amazon EC2

Mobile support is in preview release and is subject to change.

Amazon Elastic Compute Cloud (Amazon EC2) provides a GPU instance that you can use to run the Lumberyard shader compiler for mobile (Android and iOS) and macOS development. For example, you can use the AWS device farm to test a build rather than hosting a PC on a public IP address. Amazon EC2 also provides a G2 instance type that supports advanced rendering features such as texturing, shadows, and anti-aliasing. For more information, see Amazon EC2 Instances.

To run the Lumberyard shader compiler, you must do the following:

1. Set up the Amazon EC2 instance.
2. Install virtual network computing (VNC) software.
3. Connect to the shader compiler.

Topics

Prerequisites

To run the Lumberyard shader compiler on Amazon EC2, you must have the following:

- AWS account
- Familiarity with the AWS Management Console
- Understanding of Amazon EC2 instances, VNC, security groups, and the Lumberyard tool chain

Setting Up the Amazon EC2 Instance

Before you can set up the Amazon EC2 instance, you must request a GPU instance using the Request to Increase Amazon EC2 Instance Limit link. When you are done, follow the steps below. For information about Windows GPU instances, see Windows Accelerated Computing Instances.

To set up the Amazon EC2 instance

1. Open the Amazon EC2 console at https://console.aws.amazon.com/ec2/.
2. From the console dashboard, choose **Launch Instance**.
3. On the **Amazon Machine Image (AMI)** page, type `GPU`.
4. In the search results, select **Windows Server 2012 R2 with NVIDIA GRID GPU Driver**.
5. On the **Choose an Instance Type** page, select the **g2.2xlarge** type.
6. Choose **Review and Launch** to let the wizard complete the other configuration settings for you.
7. On the **Add Storage** page, add a drive with sufficient space (512 GB minimum).
8. On the **Add Tags** page, add a memorable tag for the computer. For example, `ShaderCompilerMachine`.
9. On the **Review Instance Launch** page, create a new security group or use an existing one.
10. Open port number **61453** for the shader compiler.
11. On the **Review Instance Launch** page, click **Launch**.
12. When prompted for a key pair, select **Choose an existing key pair**, and then select the key pair that you created when getting set up.
13. When you are ready, select the acknowledgement check box, and then click **Launch Instances**.
14. A confirmation page lets you know that your instance is launching. Click **View Instances** to close the confirmation page and return to the console.

Installing VNC Software

Once the Amazon EC2 instance is set up, you can install virtual network computing (VNC) software to run the shader compiler on the instance.

To install VNC software

1. Using a remote desktop connection, log in to the Amazon EC2 instance.

Note
If you are using a Mac, you must download Remote Desktop App from the App Store.

2. Download and install your preferred VNC software.
3. On your Amazon EC2 instance, run the VNC server.
4. Terminate the remote desktop session.
5. On your local PC, install a VNC client.
6. Using the VNC connection, run the shader compiler on the Amazon EC2 instance.
7. Leave the VNC window open and then follow the steps in the section called "Connecting to the Shader Compiler" (p. 1254).

Connecting to the Shader Compiler

Follow these steps to connect to the shader compiler.

To connect to the shader compiler

1. On your computer, open the platform configuration file:
   - For Android, open the `system_android_es3.cfg` file (located in the `\lumberyard\dev` directory).
   - For iOS, open the `system_ios_ios.cfg` file (located in the `\lumberyard\dev` directory).
2. Edit the configuration file to set `r_ShaderCompilerServer` to the public IP address of your Amazon EC2 instance. You can retrieve the IP address from the Amazon EC2 console.
3. Run your game on your device to connect to the EC2 instance and compile the shaders.

Using the AppDetective to Test Your Build on the AWS Device Farm

Mobile support is in preview release and is subject to change.

Lumberyard provides a script called **AppDetective** that you can use to deploy your game builds to the AWS Device Farm. This allows you to test your game on numerous devices without having to set up and manage your own hardware. For example, the Samsung Galaxy S7 uses different graphics processors depending on the sales region. Rather than acquiring all of the variations for each model that you want to test, you can use the **AppDetective** to deploy to these devices on the AWS Device Farm. You can access the **AppDetective** script in the `\dev\Tools\AppDetective` directory.

Testing your game build using the AppDetective and AWS Device Farm requires the following steps:

1. Determine which devices you want to test on.
2. Configure your game build to test on these devices in the AWS Device Farm.
3. Use the AppDetective to check the status of the run.
4. Download your game build artifacts.

Topics

Prerequisites

To use the **AppDetective** to test your build, you must have the following installed:

- AWS command line tools

  For information and installation steps, see AWS Command Line Interface.
- Python modules

  In a command line window, type the following to install the required Python modules:

  ```
  python -m pip install boto3
  python -m pip install requests
  python -m pip install grequests
  ```

  For information, see pip 9.0.1.

Setting Your Test Devices

The `test_specs.json` file (located in the `\dev\Tools\AppDetective` directory) contains a list of the devices on which you can run and test your game. You can edit this file to modify the list of devices.

To set the devices on which to test your game

1. Navigate to the `\dev\Tools\AppDetective` directory and open the `test_specs.json` file.
2. Verify or edit the devices that are listed under `device_filter`. This list determines the devices that are selected for the test.

   For example, to use Sony Xperia Z4 devices with Android OS version 5 or 6, edit the `device_filter` section as follows:

   ```
   {
    "android_fuzz_test": {
     "platform": "android",
     "appType": "ANDROID_APP",
     "device_filter": {
      "os_list": ["6", "5"],
      "name_list": ["xperia z4"]
     }
   ```

3. Test your filter by typing the following in a command line window:

   ```
   C:\> python appDetective.py -tf -tn android_fuzz_test
   ```

4. Verify that the output displays the device name that you specified under `device_filter`.

   For example, the output for the `device_filter` with `"os_list": ["6", "5"]` and `"name_list": ["xperia z4"]` should appear as follows:

   ```
   Name: [Sony Xperia Z4 Tablet] OS Version: [5.0.2]
   ```

Alternatively, if you had specified **S7** for the `name_list`, the output should appear as follows:

```
Name: [Samsung Galaxy S7 Edge (AT&T)] OS Version: [6.0.1]
Name: [Samsung Galaxy S7 SM-G930F] OS Version: [6.0.1]
Name: [Samsung Galaxy S7 (AT&T)] OS Version: [6.0.1]
Name: [Samsung Galaxy S7 (T-Mobile)] OS Version: [6.0.1]
Name: [Samsung Galaxy S7 Edge (T-Mobile)] OS Version: [6.0.1]
Name: [Samsung Galaxy S7 Edge SM-G935F] OS Version: [6.0.1]
```

Configuring and Testing Your Game

Before you can test your game on the AWS Device Farm, you must configure your game build to include required assets in the executable file.

To configure and test your game

1. Include your game assets in the executable file:
   a. For iOS games, no action is required.
   b. For Android games, do the following:
      i. Edit the `project.json` file for your game to set `place_assets_in_apk` to **1**.
      ii. Build and pack the shaders into your game, or run the shader compiler on an Amazon EC2 instance for the devices in the AWS Device Farm to compile the shaders. For information, see Running the Shader Compiler on Amazon EC2 (p. 1252).
2. In a command line window, run the following command:

```
python appDetective.py -tn [name of the test spec from the test_specs.json file] -
a [game name] -n [project name in the AWS Device Farm] -s
```

   For example, if your test spec name is `android_fuzz_test`, your game name is `SamplesProject.apk`, and your project name is `DFTestTutorial`, run the following command:

```
python appDetective.py -tn android_fuzz_test -a SamplesProject.apk -n DFTestTutorial -s
```

3. Verify that the output displays the following, with your specified values:

```
Created project DFTestTutorial with ARN:arn:aws:devicefarm:us-west-
2:330708257384:project:625319c8-b78c-400e-86a6-7ee1f5fab7ee

Creating device pool with the following devices:
Samsung Galaxy S7 Edge (AT&T)
Samsung Galaxy S7 SM-G930F
Samsung Galaxy S7 (AT&T)
Samsung Galaxy S7 (T-Mobile)
Samsung Galaxy S7 Edge (T-Mobile)
Samsung Galaxy S7 Edge SM-G935F

Created Device Pool with ARN:arn:aws:devicefarm:us-west-
2:330708257384:devicepool:625319c8-b78c-400e-86a6-7ee1f5fab7ee/7a24d8c3-5cbd-4c69-
afaf-de6e3be6ec02
Sending data...
Sending data...
Sending data...
Sending data...
Sending data...
```

```
Upload complete
Started run on Device Farm...
```

4. Allow the build to run and complete. By default, the run will complete in 10 minutes (600 seconds). You can override the run time by setting the `-d` command and specifiying a time, in seconds, in a command line window.

Checking the Run Status

You can use the AppDetective to check the status at any time during a run. Do the following to check the run status.

In a command line window, run the following command:

```
python appDetective.py -tn [name of the test spec from the test_specs.json file] -
n [project name in the AWS Device Farm] -g
```

For example, if your test spec name is `android_fuzz_test` and your project name is `DFTestTutorial`, run the following command:

```
python appDetective.py -tn android_fuzz_test -n DFTestTutorial -g
```

Downloading Your Game Build Artifacts

When your game has finished running on the devices in the AWS Device Farm, you can download the artifacts that were generated during the run. These artifacts include screenshots, videos, and logs. Do the following to download your game build artifacts.

In a command line window, run the following command:

```
python appDetective.py -tn [name of the test spec from the test_specs.json file] -
n [project name in the AWS Device Farm] -f
```

For example, if your test spec name is `android_fuzz_test` and your project name is `DFTestTutorial`, run the following command:

```
python appDetective.py -tn android_fuzz_test -n DFTestTutorial -f
```

The downloaded files are saved to a directory with the name of your project in the AWS Device Farm. In this example, the directory is called `DFTestTutorial`.

AppDetective Command Line Parameters

The AppDetective uses the following command line parameters.

`--help`

Displays all of the available parameters.

Shortcut: `-h`

`--verbose`

Displays a verbose output from the AppDetective and its modules. This may help to debug any issues with running the AppDetective.

Shortcut: `-v`

`--startrun`

Immediately runs your game on the AWS Device Farm.

Shortcut: `-s`

`--repeatrun`

Restarts a run with the currently uploaded game. The `--projectname` parameter is required with this command.

Shortcut: `-r`

`--runduration`

Time, in seconds, that the game should run on each device. The `--projectname` parameter is required with this command.

Shortcut: `-d`

`--getrunstatus`

Returns the current state of the run. The `--projectname` parameter is required with this command.

Shortcut: `-g`

`--fetchartifacts`

Retrieves all artifacts—such as log files, screenshots, and videos—for every device in the current run. The `--projectname` parameter is required with this command.

Shortcut: `-f`

`--fetchartifactsbyjob`

Retrieves all artifacts—such as log files, screenshots, and videos—for every device in the specified job. A job is a specific task issued to the AWS Device Farm and can be found with the `--listjobs` parameter. The `--projectname` parameter is required with this command.

Shortcut: `-fj`

`--updateapp`

Uploads a new version of your game to the current project. The `--projectname` parameter is required with this command.

Shortcut: `-u`

`--listjobs`

Lists the jobs that are currently running on the AWS Device Farm. You can use this information and the `--fetchartifactsbyjob` parameter to retrieve artifacts for a specific device. The `--projectname` parameter is required with this command.

Shortcut: `-l`

`--testfilter`

Tests the `test_specs.json` file filter and outputs the devices that will be used for the specified settings.

Shortcut: `-tf`

`--deleteallprojects`

Deletes all projects and related files and data from the AWS Device Farm. This action cannot be undone.

Shortcut: `-x`

`--projectname`

Name of the project on the AWS Device Farm.

Shortcut: `-n`

`--app`

The APK or IPA for your project that you want to upload and run on the AWS Device Farm.

Shortcut: `-a`

`--test_name`

The specific test that you want to use from the `test_specs.json` file.

Shortcut: `-tn`

macOS Support

macOS support is in preview release and is subject to change.

You can use Lumberyard to build macOS applications. Lumberyard includes four macOS-supported sample projects that you can use to learn how to build assets for macOS games using the Asset Processor, build shaders using the remote shader compiler, and build and deploy macOS applications using the Lumberyard build tools.

Topics

Prerequisites

To build games for macOS or 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. 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 `/Tools/LmbrSetup/Mac` 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**:
   - **Compile the game code**
   - **Compile the engine and asset pipeline**
   - **Compile the Lumberyard Editor and tools**

     Note
     Lumberyard Editor is not supported on macOS. Selecting this option enables the ability to build the asset processor and resource compiler only.
   - **Compile for iOS devices**
   - (Optional) **Compile for Android devices**

     Note
     Select this option if you are developing for Android devices. You must have the Android SDK installed on your Mac.
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. 20).
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.

Building macOS Games

macOS support is in preview release and is subject to change.

Before you can run your game on a macOS computer, you must ensure the shader compiler (located in the `\lumberyard\dev\Tools\CrySCompileServer\x64\profile` directory) is running on your PC. For more information, see Building Shaders for macOS Games (p. 1262).

To build your game for macOS

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, run the following command: `sh lmbr_waf.sh configure xcode_mac`

   Note
   Metal is the default renderer.
3. Do one of the following to build your game:
   - Use a command line to build various targets of your game:
     - To build debug, run the following command: `sh lmbr_waf.sh build_darwin_x64_debug -p all`
     - To build profile, run the following command: `sh lmbr_waf.sh build_darwin_x64_profile -p all`

- To build release, run the following command: `sh lmbr_waf.sh build_darwin_x64_release -p all`
- Use Xcode and the generated solution located in the `Solutions` folder in the directory where you installed Lumberyard to build your game.

Building Game Assets for macOS Games

macOS support is in preview release and is subject to change.

When you build a macOS 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.

Note
If you make changes to your game in Lumberyard Editor on your PC, you must copy the updated assets to your Mac using your preferred method. For example, you can use source control. For information, see Using the Perforce Plugin with Lumberyard (p. 26).

To build macOS game assets on your Mac

1. On your Mac, close all instances of 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` (remove the preceding semicolon) and to comment out `pc=enabled` (add a preceding semicolon). Save the file.
4. In a command line window, navigate to the `/Lumberyard/dev/BinMac64` directory and run the Asset Processor (GUI or batch version) to process and build your game assets.

Building Shaders for macOS Games

macOS 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 macOS during development. This compiles the subset of shaders required by your game, on demand.

Note
You must connect your PC and macOS 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 macOS 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.

You can use a whitelist to specify the IP addresses that are allowed to connect to your remote shader compiler. For information, see Creating a Whitelist for the Remote Shader Compiler (p. 1358).

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 one of the following:

- If you are using Visual Studio 2015: `lmbr_waf build_win_x64_vs2015_profile -p all --targets=CrySCompileServer`
- If you are using Visual Studio 2013: `lmbr_waf build_win_x64_vs2013_profile -p all --targets=CrySCompileServer`

To run the shader compiler on your PC

1. Edit the `system_osx_osx_gx.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_osx_gx.cfg` directory).

Running macOS Games

macOS support is in preview release and is subject to change.

Before you can run your game on a macOS computer, you must ensure the shader compiler (located in the `\lumberyard\dev\Tools\CrySCompileServer\x64\profile` directory) is running on your PC. For more information, see Building Shaders for macOS Games (p. 1262).

To run your game on a macOS 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 Mac App.
3. (Optional) Load different levels by doing one of the following:
   - Open the console window using the tilde (**~**) key, and then type `map <name of map or level to load>`. The console supports tab completion, so you can press **Tab** after the map command to see the list of supported levels and maps to load.
   - Edit the `autoexec.cfg` file for your game to change the `map` command to load a different map or level. Run the game from Xcode again.

     For example, if you are using the Samples Project, edit the `autoexec.cfg` file located in the `/dev/SamplesProject` directory. macOS supports the Advanced_RinLocomotion level.
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 Rin in the Character Controller view by using the mouse or keyboard (**WASD**).
   - Jump in the Character Controller view by pressing the **Space** key.

macOS Debugging and Troubleshooting

macOS support is in preview release and is subject to change.

Lumberyard provides full access to the source code, which allows you to debug your macOS application using Xcode without additional Lumberyard-specific steps to follow. For information about debugging and profiling your macOS 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 macOS Games (p. 1262) page.

Cleaning the project does not create a full rebuild of the macOS 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 macOS 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, run the following command: `lmbr_waf.sh clean_darwin_x64_debug`
- To build profile, run the following command: `lmbr_waf.sh clean_darwin_x64_profile`
- To build release, run the following command: `lmbr_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 a Project for Your macOS Games

macOS support is in preview release and is subject to change.

The topics in macOS Support (p. 1260) demonstrate how to use the Samples Project that is included with Lumberyard to build game assets, shaders, and macOS applications. You can follow the same instructions to create a project for your own macOS game.

> **Note**
> Ensure you have the prerequisites (see macOS Support (p. 1260)) and your Mac is properly set up to compile for macOS computers.

To create a project for your macOS game

1. On your PC, use the Project Configurator to create a new project. For information, see Using the Project Configurator (p. 28).
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 macOS Games (p. 1261) and Running macOS Games (p. 1263) pages.

   > **Note**
   > If you enabled multiple projects, you can switch between multiple targets in your Xcode project.

Linux Support

Linux support is in preview release and is subject to change.

Lumberyard supports compiling and deploying the Windows client for a multiplayer project on a Linux dedicated server. You must do the following for the Windows client to work properly on a Linux dedicated server:

- Compile the assets on a Windows computer (p. 1266)
- Compile the server executable for use on a Linux computer (p. 1267)
- Compile a Windows client to use to connect to the Linux server (p. 1847)

Once compiled, you can deploy assets to either a Linux server or Windows client.

Topics

Prerequisites

You must have the following:

- Lumberyard source code
- Windows computer with a Lumberyard development environment and the ability to generate a Windows build
- Linux computer with Ubuntu on Amazon Elastic Compute Cloud (Amazon EC2)

  For information, see Setting up Ubuntu on Amazon EC2 (p. 1270).
- Familiarity with the following:
  - Amazon Elastic Compute Cloud (Amazon EC2)
  - Connect to your Linux instance

Compiling Assets on a Windows Computer

Linux support is in preview release and is subject to change.

You must compile assets for the multiplayer project on a Windows computer. Once compiled, you can deploy assets to either a Linux server or Windows client.

To compile assets on a Windows computer

1. Run the Project Configurator (located in the `\lumberyard_version\dev\Bin64` directory).
2. In the Project Configurator, select **MultiplayerProject** and click **Set as default**.
3. Close the Project Configurator.
4. Edit the `bootstrap.cfg` file (located in the `\dev` directory) to set `connect_to_remote` to **0**.
5. Run the following command from the `\dev` directory: `lmbr_waf.bat configure`
6. Follow these instructions for Creating Release Builds for PC (p. 1848). Alternatively, you can run the `BuildMultiplayerSample_Paks_PC_dedicated.bat` file (located in the `\dev` directory).
7. To create a deployment package for Linux, run as Administrator the `MultiplayerSample_LinuxPacker.bat` file (located in the `\dev` directory).

   Note
   You do not need the following steps to deploy files to a Linux computer if you use the same computer to compile and to deploy files.

To deploy files to a Linux computer

1. (Optional) Compress the `MultiplayerSample_LinuxPackage.tar` file to any location on your Windows computer.
2. Use your preferred file transfer tool to deploy the `MultiplayerSample_LinuxPackage.tar` file on your Windows computer to your Linux computer. For information, see Connect to Your Linux Instance in the *Amazon EC2 User Guide for Linux Instances*.

Building the Lumberyard Executable for Linux

Linux support is in preview release and is subject to change.

After compiling the assets on a Windows computer, you must build the Lumberyard executable for use on a Linux computer.

To build the Lumberyard executable for Linux

1. On your Linux computer, in a command line window, run the following command: `tar -xvf MultiplayerSample_LinuxPackage.tar`
2. Change directory to the `MultiplayerSample_LinuxPackage/dev` directory.
3. Verify that you are using the following supported versions:
   - Ubuntu 16.04
   - Clang Compiler 3.8
   - C++ 3.7.0
   - C++ ABI 3.7.0
4. Run the `SetupAssistantBatch.sh` file and type the following to create symlinks:

```
Tools/LmbrSetup/Linux/SetupAssistantBatch --none --3rdpartypath path to "3rdParty
 folder" --enablecapability compilegame --enablecapability compileengine
```

Note
Using an invalid third-party path may create invalid junctions. Type the following to remove these junctions and then run the `SetupAssistantBatch.sh` file again: `rm -rf Code/Tools/SDKs Code/Sandbox/SDKs and Code/SDKs`

5. Edit the `system_linux_pc.cfg` file (located in the `/dev/MultiplayerSample_pc_Paks_Dedicated` directory) to set `log_RemoteConsoleAllowedAddresses` to your IP address.
6. Build the Lumberyard executable for Linux:
   - To build profile, run the following command:
     - (Optional for nonrelease builds) `./lmbr_waf.sh build_linux_x64_profile -p host_tools`
     - `./lmbr_waf.sh build_linux_x64_profile_dedicated -p game_and_engine`
   - To build debug, run the following command:
     - (Optional for nonrelease builds) `./lmbr_waf.sh build_linux_x64_debug -p host_tools`
     - `./lmbr_waf.sh build_linux_x64_debug_dedicated -p game_and_engine`

Testing the Windows Client to Linux Server Connection

Linux support is in preview release and is subject to change.

Lumberyard supports compiling and deploying the Windows client for a multiplayer project on a Linux dedicated server. You must have completed the following:

- Compiled the assets on a Windows computer (p. 1266).
- Built the Lumberyard executable for use on a Linux computer (p. 1267).

Once the Lumberyard executable is ready, you can test the connection between the Windows client and the Linux server.

To test the client and server connection

1. Start the Linux server:
   a. In a terminal window, change directory to the `BinLinux64.Dedicated` directory.

   ```
   $ cd BinLinux64.Dedicated
   ```

   b. The launcher requires that the current working directory contain the data and `multiplayergame` folder, which includes the `.pak` files. Copy the assets from the `MultiplayerSample_pc_Paks_Dedicated` directory to the `BinLinux64.Dedicated` directory.

   ```
   $ cp -r ../MultiplayerSample_pc_Paks_Dedicated/* ./
   ```

   c. Run the `MultiplayerSampleLauncher_Server`:

```
$ ./MultiplayerSampleLauncher_Server
```

2. Start the remote console on your Windows computer:

   a. Run the `RemoteConsole.exe` file (located in the `\dev\Tools\RemoteConsole` directory).
   b. In the **Targets** window, for the **Custom Ip**, type the server's IP address.
   c. For the **Custom Port**, enter the server's port.

      The Linux server echoes the chosen port, which appears in the last line, for example, *Remote console listening on: 4600*. The server also echoes when the remote console is attached. If you restart the server, the remote console is automatically reattached.
   d. In the command line window, run the following command:

```
mphost
map multiplayersample
```

3. Start the Windows client:

   a. Edit the `bootstrap.cfg` file (located in the `\dev` directory) to set `connect_to_remote` to **1**.
   b. Run the `MultiplayerSampleLauncher.exe` file:

      i. For Microsoft Visual Studio 2013, this file is located in the `\dev\Bin64vc120\` directory.
      ii. For Microsoft Visual Studio 2015, this file is located in the `\dev\Bin64vc140\` directory.
   c. After the launcher loads, open the console by pressing the accent grave key and typing: `` `mpjoin *server IP address* ``

Preparing the GameLift Package

Linux support is in preview release and is subject to change.

Lumberyard supports compiling and deploying the Windows client for a multiplayer project on a Linux dedicated server. You must have completed the following:

- Compiled the assets on a Windows computer (p. 1266).
- Built the Lumberyard executable for use on a Linux computer (p. 1267).
- Tested the connection between the Windows client and the Linux server (p. 1268).

Once you have tested the client and server connection, you can prepare your GameLift package, which allows you to use Amazon GameLift to deploy your game servers.

To prepare the GameLift package

1. On your Windows computer, in a command line window, run the following command: `./MultiplayerSample_CreateGameLiftPackage.sh`
2. In the AWS CLI, type the following to upload your package to Amazon GameLift:

```
aws gamelift upload-build --operating-system AMAZON_LINUX --build-root "./
GameLiftPackageLinux" --name "your package name" --build-version "your build version"
 --region us-west-2
```

Note
If you do not have the AWS CLI installed, type the following in the command line window:

```
sudo pip install awscli
sudo pip install --upgrade awscli
```

Setting up Ubuntu on Amazon EC2

Linux support is in preview release and is subject to change.

Before you can install the required software to compile Lumberyard on Linux, you must set up an Ubuntu Linux virtual machine on Amazon Elastic Compute Cloud (Amazon EC2). For instructions, see How to Launch a Linux Virtual Machine and select Ubuntu 16.04 LTS. The Linux virtual machine (instance) is launched on Amazon EC2 within the AWS Free Tier.

To install required software for Lumberyard on the Ubuntu Linux virtual machine

1. In your Linux instance, in a command line window, do the following to install updates and required packages:

   a. To install updates, run the following command: `sudo apt-get update`
   b. To install Python 2.7.12, run the following command: `sudo apt-get install python-tk`
   c. To add a Mozilla Security repo and install binutils-2.26, run the following commands:

   ```
   sudo add-apt-repository ppa:ubuntu-mozilla-security/ppa
   sudo apt-get update
   sudo apt-get install binutils-2.26
   ```

   d. To install LibCurl3, run the following command: `sudo apt-get install libcurl4-nss-dev`
   e. To install the libc++ standard library for Clang, run the following command: `sudo apt-get install libc++-dev`
   f. To install the libc++ ABI standard library, run the following command: `sudo apt-get install libc++abi-dev`
   g. To install Clang version 3.8, run the following command: `sudo apt-get install clang-3.8`
   h. To install universally unique ID headers and static libraries, run the following command: `sudo apt-get install uuid-dev`
   i. To install the Python package manager, run the following command: `sudo apt-get install python-pip`
   j. To install the AWS command line interface, run the following command: `sudo pip install awscli`
2. An issue with Clang 3.8 may require you to configure certain packages. Verify that the following directories exist in your environment:

   - `/usr/bin/clang`
   - `/usr/bin/clang++`
   - `/usr/include/c++/v1`

   If they do not exist, type the following to configure the required packages:

a. `sudo ln -s /usr/bin/clang-3.8 /usr/bin/clang`
b. `sudo ln -s /usr/bin/clang++-3.8 /usr/bin/clang++`
c. `sudo cp -v /usr/include/libcxxabi/* /usr/include/c++/v1/`
d. `hash -r`

Particle Effects System

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. Use the **Particle Editor** to create and manage particle effects in your game. You can place particle emitters in your level, link them to an object, set up a material to define a custom effect, and control these effects from the **Flow Graph** and the **Track View** editors.

Lumberyard uses two shaders for rendering particles:

- Particles Shader (p. 1380) – Render particle effects that are affected by light. These effects can cast shadows and cause reflections.
- ParticleImposter Shader (p. 1380) – Render mesh particle effects that are not affected by light. These effects do not cast shadows or cause reflections.

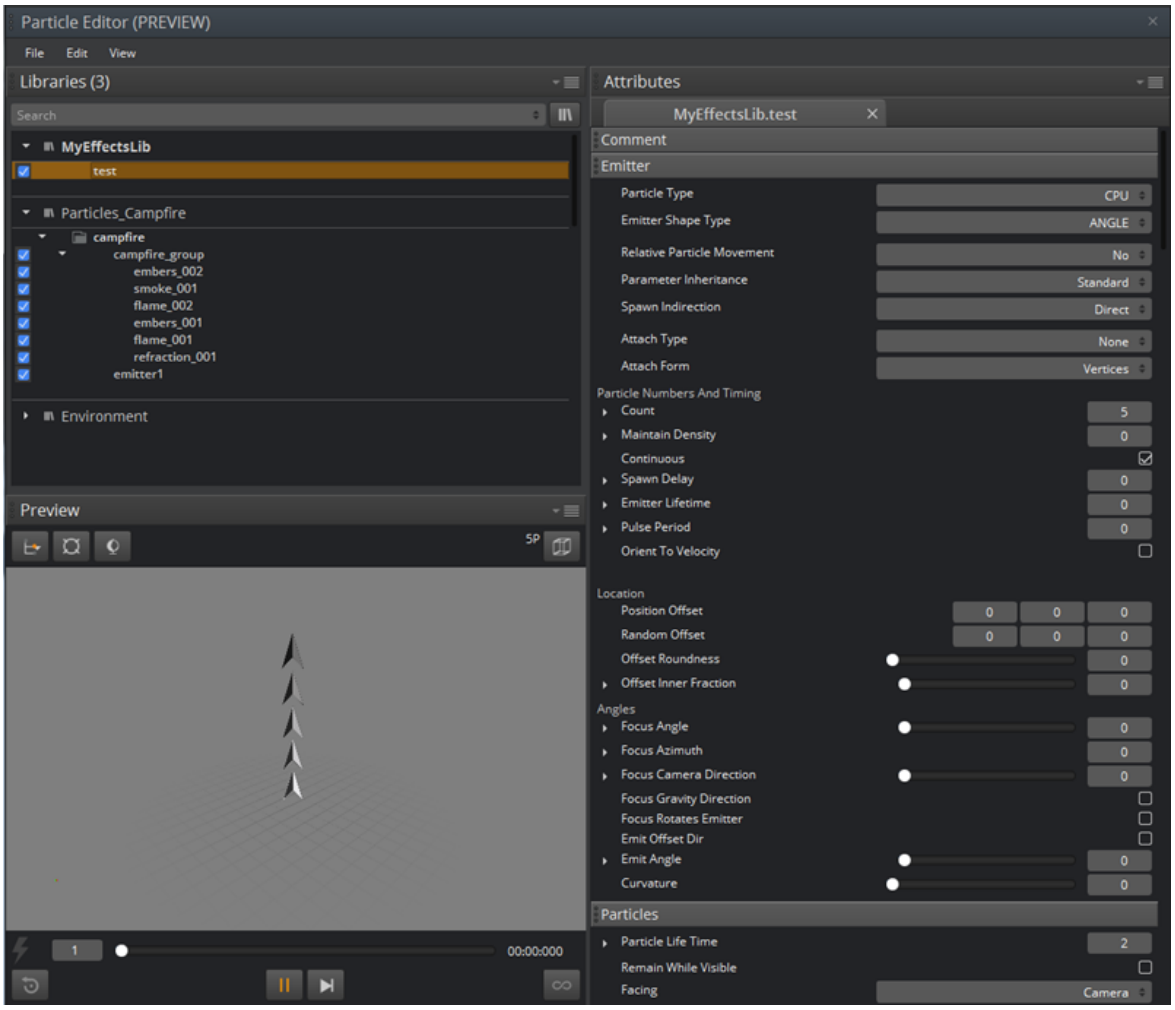
Topics

Using the Particle Editor

Use the **Particle Editor** to create, edit, preview, and save particle libraries and emitters that are used with entities in your levels.

To open the Particle Editor

- In Lumberyard Editor, choose **Tools**, **Particle Editor**. You can also click the particle editor icon in the Lumberyard Editor toolbar.



The **Particle Editor** includes the following UI elements:

- **Libraries** panel – Lists the particle art assets. You can view and interact with multiple libraries simultaneously.
- **Preview** panel – Displays the selected, active particle effect. The preview camera automatically positions to capture the entire particle. Use the **W**, **A**, **S**, and **D** keys and mouse controls to pan, zoom, and rotate the camera.
- **Attributes** panel – Lists the properties for the selected particle.
- **Level of Detail** panel – Displays level of detail (LOD) information for added particles. For more information, see Managing Particle Level of Detail (LOD) (p. 1337).

Adding a Particle Component

To add a particle component to your level

- Do one of the following:
  - Drag a particle emitter from the **Particle Editor** library and drop it into the **Lumberyard Editor** viewport.

- Drag a particle effect asset from the **Asset Browser** into your scene. Choose the emitter that you want to use from the **Entity Inspector** properties. For more information, see Entity Inspector (p. 441).
- Create a **Particle** component entity and assign the particle effect with the **Particle effect library** parameter. For more information, see the **Particle (p. 527)** component and Adding Components to an Entity (p. 593).

Customizing the Particle Editor UI

The **Particle Editor** has a customizable interface so you can set up your workspace in different layouts. All panels are moveable. You can undock panels to become floating windows, or you can dock panels in new locations to rearrange the look and feel. Save your custom layouts to access later or to share with others.

You can customize the **Particle Editor** in the following ways:

Floating panels

Drag the panel title bar to separate the panel from the editor and make it a floating window.

Docking panels

Dock panels on the edges of the window or anywhere an orange highlight appears. To dock a floating panel, drag the panel title bar into the **Particle Editor** window.

Tabbing panels

You can dock a panel inside another panel to create a tabbed view. Toggle the tabs to display one panel and hide the other. This option minimizes the number of panels that are visible at a time.

Showing or hiding panels

Customize which panels are visible by clicking **View** and choosing a panel to show or hide.

Resetting the layout

To reset the **Particle Editor** layout, click **View**, **Reset to Default Layout**. This positions the **Libraries** panel on the top left, the **Preview** panel on the middle left, the **Level of detail** panel on the bottom left, and the **Attributes** panel on the right.

Exporting a layout

You can export and share custom layouts. After you create a layout, click **View**, **Export layout**. Navigate to the location where you want to export the layout file.

Importing a layout

To import custom layouts, click **View**, **Import layout**. Locate and select the layout file.

Managing Particle Libraries

Particle libraries are `.xml` assets that contain the data for all effects in the library. The `.xml` files are located in the `\`*`game_project_name`*`\libs\particles` directory.

Use the **Particle Editor** to add and manage particle libraries. You can rename library assets by duplicating and renaming a library in the **Particle Editor**. Do not rename the base `.xml` file; doing so can negatively affect the entities in your levels.

Multiple selection

Press **Ctrl** or **Shift** to select multiple emitters. The emitters can be from different libraries. You can do any of the following:

Copy – Copies the selected item. If you copy multiple items, they become children of the item that you paste them on.

Delete – Deletes all selected emitters.

Group – Groups the selected items that share the same parent. Select the emitters that you want to group and press **Ctrl+G**. Alternatively, you can right-click the selected emitters and choose **Group**.

If you use hot keys while multiple items are selected, the hot keys apply to all selected items.

Drag and drop

Add emitters from any of the libraries into a specific library. To do this, drag and drop the emitters onto the library name. You can also drag emitters from the library to a new parent in the same library.

Search

Type queries into the **Libraries** panel search box to instantly view results. Click the arrow icon on the left to display previous search results.

Level library (Deprecated)

The level library is no longer updated. You can import particle data stored in level files that were created in Lumberyard 1.8 and earlier. To do so, open a level and, in the **Particle Editor**, choose **File**, **Import Level Library**. This imports the data to a project particle library in the `\game_project_name\libs\particles` directory.

We recommended that you update all entities that reference the level library emitters to use this newly created library.

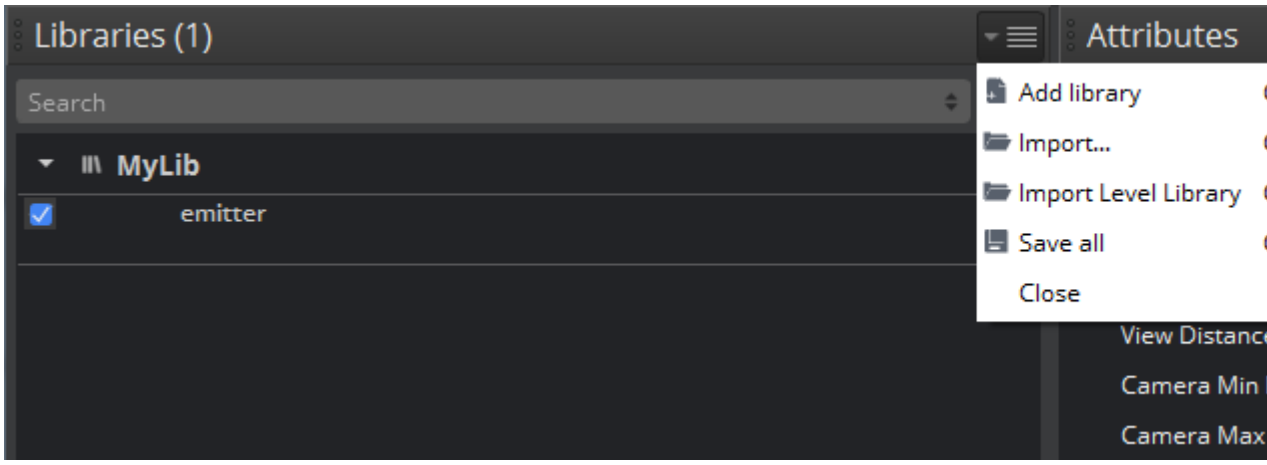
Topics

Adding Particle Libraries

Particle effect data is stored in an XML-based library file.

To add a particle library

1. In the **Particle Editor**, do one of the following:
   - Choose **Create new library**.
   - Choose **File**, **Add library**.
   - Choose the drop-down arrow in the **Libraries** panel, and choose **Add library**.
   - Choose the  icon to the right of the search bar.



2. In **Libraries**, where the default name is highlighted, type a name for the library.
3. Choose **Add Particle** or **Add Folder** and then type a name for the new particle or folder.

Importing Particle Libraries

Particle effect data is stored in an XML-based library file. You can import a particle library for your game.

To import a particle library

1. In the **Particle Editor**, choose **File**, **Import**.
2. In the **Pick Particles** dialog box, select a library to load. Choose **OK**. You can also choose the drop-down arrow to access the menu.

Saving Particle Libraries

You can save your particle library in the **Particle Editor**.

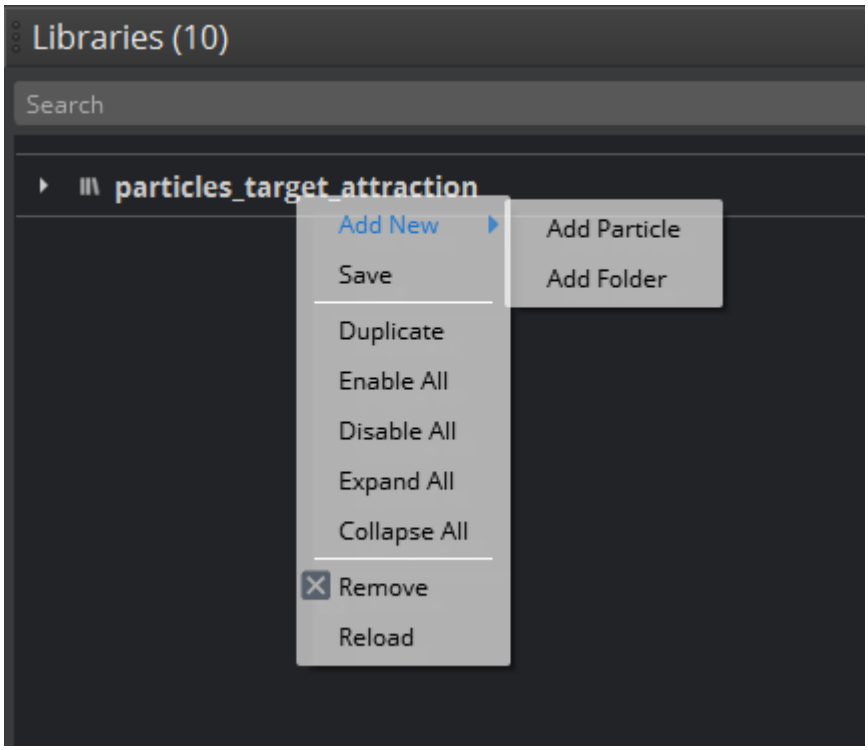
To save a particle library

- In the **Particle Editor**, right-click the library name and choose **Save**. You can also press **Ctrl+S** to save all libraries that are loaded in the editor.

  Your library is now saved to the `\game_project_name\libs\particles` directory. You can nest libraries in any hierarchy in the `\libs\particles` directory.

Using Particle Libraries

The **Particle Editor** provides a context menu with the following options for you to manage your particle libraries. Right-click a library name to access the context menu.



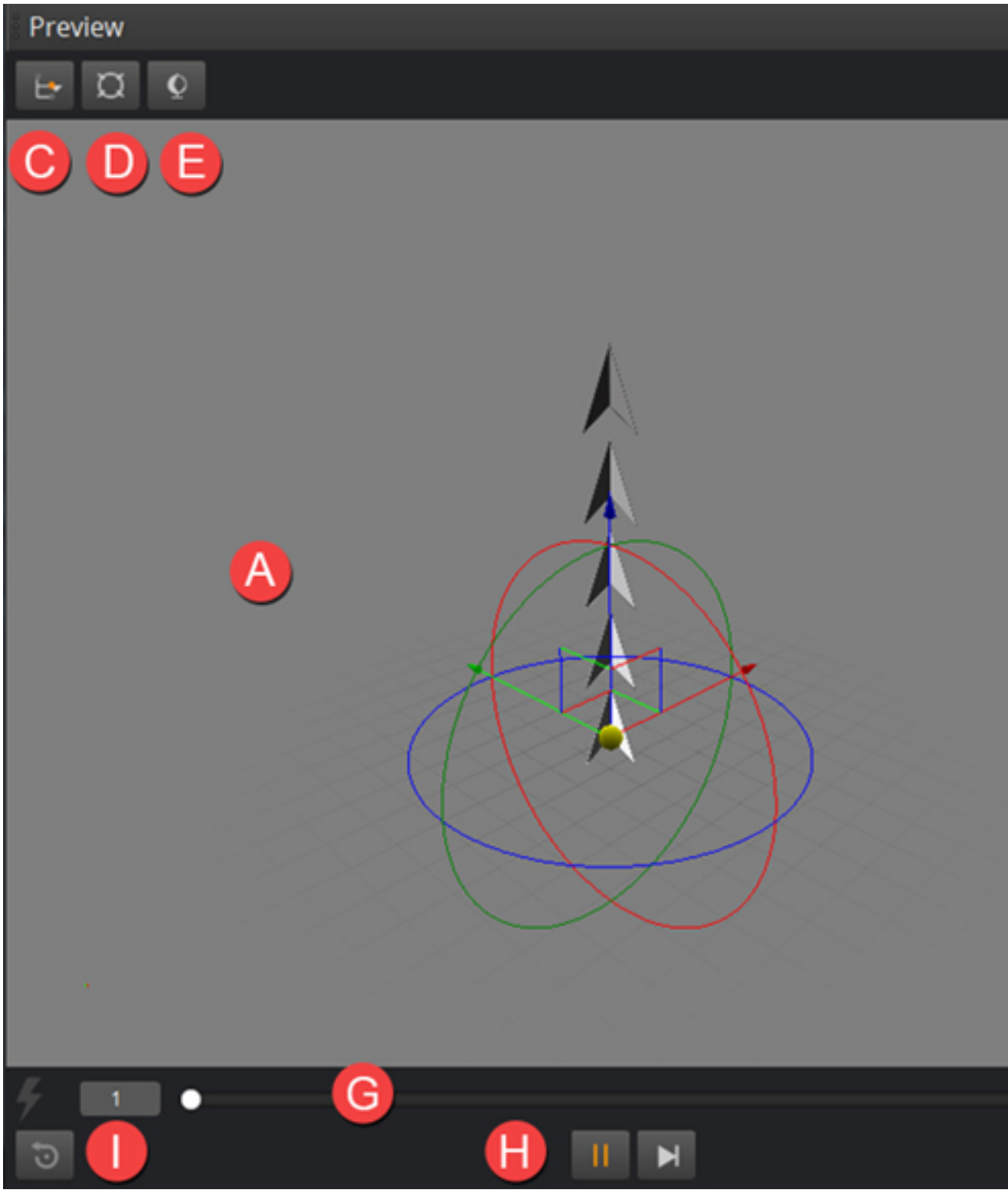
Option	Description
Add New	**Add Particle** – Adds a new emitter to the library. The default keyboard shortcut is **Ctrl+N**. **Add Folder** – Adds a new folder to the library. The default keyboard shortcut is **Ctrl+Alt+N**.
Save	Saves changes to the selected library. Libraries are saved in the `\game_project_name\libs\particles` directory.
Duplicate	Duplicates and renames the library.
Disable/Enable All	Disables or enables all items in the library.
Expand/Collapse All	Expands or collapses all branches in the library. You can click a library name to collapse or expand the entire library. The contents do not lose their collapsed or expanded state when you do so.
Remove	Removes the library from the library list. The library list is still available on disk.
Reload	Reloads the library.

You can access the following toolbar menu items and buttons in the **Particle Editor** and **Libraries** panel.

Option	Description
Add Particle	Adds a particle effect. By default, the particle effect is a child of the selected particle, folder, or library. Set the particle name in the **New Particle Name** window.
Add Folder	Adds a directory in the library so you can organize your particle effects.
Add library	Adds a particle library.
Import	Opens the file browser to import the selected particle libraries.
Import Level Library	Imports particle data stored in level files that were created in Lumberyard 1.8 or earlier.
Save All	Saves all modified particle libraries to disk.
Close	Closes the **Particle Editor**.
Undo	Clears the last change.
Redo	Removes the last undo.
Copy	Copies all of the settings for the selected item to the clipboard.
Paste	Writes data from the clipboard to the selected item.
Duplicate	Duplicates the selected particle effect.
Add LOD	Adds a level of detail (LOD) to the selected emitter.
Rename	Renames the selected item.
Delete	Deletes the selected item.
Reset to default	Resets all properties and parameters for the selected item to the default values and states.
Edit Hotkeys	Opens the **HotKey Configuration** window for you to edit keyboard shortcuts (hotkeys).

Using the Preview Panel

You can use the **Preview** panel in the **Particle Editor** to view the particle effects.



This panel has the following features:

A. Display content in the viewport

B. Display the main menu
C. Choose which emitter hierarchies to display in the viewport
D. Reset the viewport to default settings
E. Toggle time of day visibility to simulate the approximate time of day
F. Toggle the display of the particles wireframe
G. Play back the timeline
H. Play, pause, and step forward
I. Reset emitter playback
J. Loop playback

You can access the following **Preview** panel options in the main menu and context menu.

Option	Description
Import Mesh	Imports a game mesh to view.
Play	Toggles playback.
Pause	Pauses playback.
Step forward	Steps forward one frame.
Play loop	Toggles the playback from once to looping.
Reset	Resets the playback to the beginning.
Reset camera position	Resets the preview camera to the default position.
Show/Hide overdraw	Toggles display of overdraw in the editor.
Wireframe view	Toggles display of the particles wireframe.
Show/Hide particle count	Toggles display of the particle count.
Show/Hide bounding box	Toggles display of the emitter bounding box.
Show/Hide gizmo	Toggles display of the preview window transform gizmo.
Show/Hide playback controls	Toggles display of the preview playback controls.
Show/Hide grid	Toggles display of the preview grid.
Show/Hide emitter shape	Toggles display for the visualization of shape emitters and their relative coordinates.
Background color	Opens the color picker for you to set the color of the preview background.
Grid color	Opens the color picker for you to set the color of the preview grid.
Reset colors	Resets the background and grid to the default colors.
Move on spline	Displays a spline in the **Preview** panel for the emitter to move along. This helps to preview the emitter motion. **Spline mode** • Single path – Traverses the path once

Option	Description
	• Loop – Traverses the path indefinitely • Ping-pong – Traverses back and forth on the path **Spline shape** • Line • Sine wave • Coil **Spline speed** • Speed x1 • Speed x2 • Speed x3 • Speed x5
Reset spline settings	Disables all spline settings.
Reset to default	Resets the preview window to the default configuration.
Close	Closes the **Preview** panel.

Particles Attributes Reference

Use the particle parameters and attributes in the **Particle Editor** to modify how emitters and particles look and behave. Except for a parameter's base value, most numeric parameters allow random variation over a particle or emitter lifetime. The following reference lists the particle attributes and associated parameters that you can adjust for the preferred effect. All particle parameters and attributes are available in the **Attributes** panel.

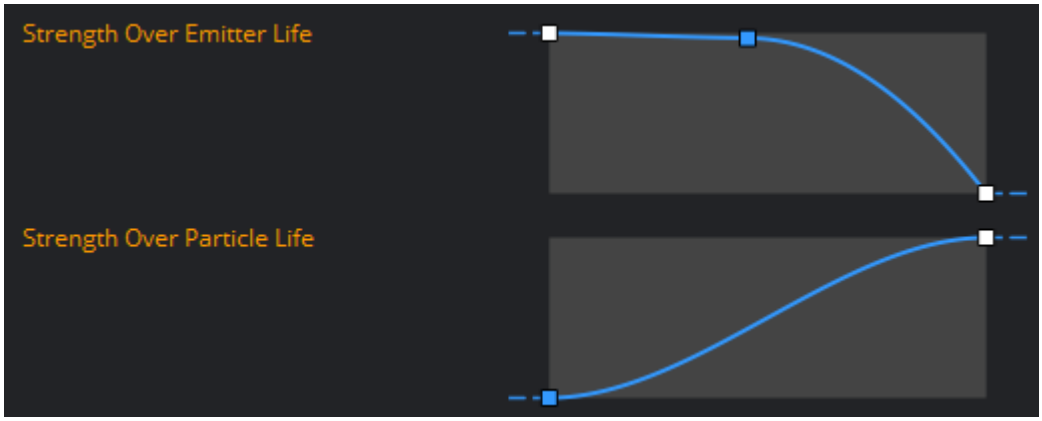
Some parameters have the following subparameters:

- **Random** – Specifies how much a particle's parameter value deviates from the default value of 0 (no variation).
- **Strength Over Emitter Life** – Controls the alpha strength over the lifetime of the particle. This subparameter works with finite particles only. This subparameter has no effect if set to continuous.
- **Strength Over Particle Life** – Controls the alpha strength over the lifetime of an individual particle. For example, you can use this subparameter to make a smoke particle fade to nothing once its lifetime has finished. The particle fades out earlier or later depending on where you reduce the value to zero.

Topics

Using the Curve Editor

Use the curve editor to edit the attribute values over the emitter and particle lifetimes. **Strength Over Emitter Life** is active only if you have an emitter lifetime that is greater than zero.



To edit the curves

1. In the curve editor, double-click along the curve timeline to add a new key.
2. Use your mouse to drag the keyframe to the preferred value and shape.

Advanced Attribute

Parameters in this attribute include advanced appearance and optimization settings.

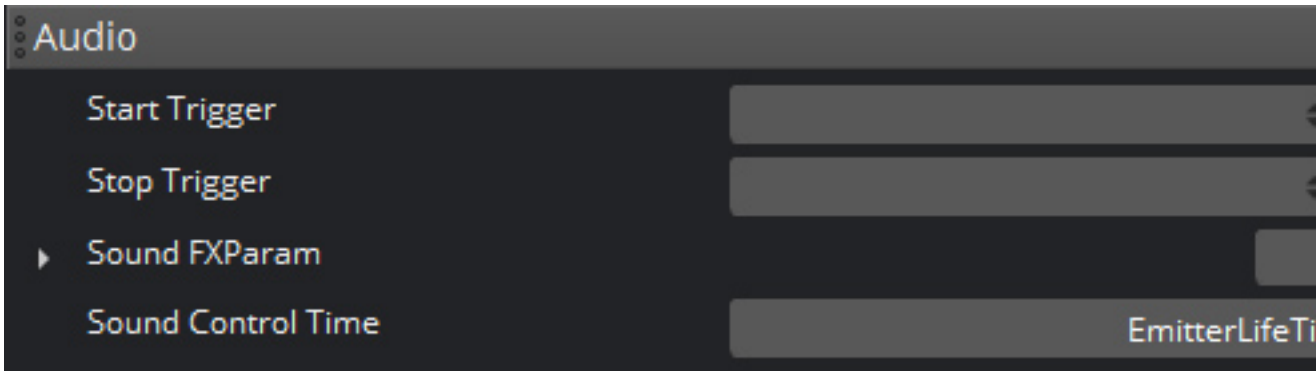
Advanced Attribute Parameters

Parameter Function	Description
Force Generation	(CPU only) Adds a force that is generated by the emitter. • **None** – Does not add a 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 a negative value creates the wind force in the opposite direction. You can use this to create a vacuum force. • **Gravity** – Creates a physical gravity force similar to wind, but creates a gravitational acceleration force instead of wind velocity.

Parameter Function	Description
	Default value: **None**
Fill Rate Cost	(CPU only) Multiplies the emitter's contribution to the total fill rate. This affects automatic culling of large particles when the global limit is reached. Set this to a value greater than 1 if the effect is relatively expensive or unimportant. Set this to a value less than or equal to 0 if the effect is important. You should not experience automatic culling. Valid values: 0+ Default value: 1
Heat Scale	(CPU only) Multiplies thermal vision. Valid values: 0 – 4 Default value: 0
Sort Quality	(CPU only) Specifies more accurate sorting of new particles in the emitter's list. To avoid the popping that results from changing the render particle order, particles are never re-sorted after emission. Instead they are sorted only when emitted, based on the main camera's position. • **0** (default, fastest) – Places the particle at the front or back of the list, depending on its position relative to the center of the emitter bounding box. Additional force is not added. • **1** (medium slow) – Sorts existing particles into a temporary list. New particles do a quick binary search to find an approximate position. • **2** (slow) – Sorts existing particles into a temporary list. New particles do a full linear search to find the position of least sort error. Valid values: 0 – 2 Default value: 0
Half Res	Renders particles in a separate, half resolution pass, reducing rendering cost. To enable this feature, set the console variable `r_ParticlesHalfRes` to 1. Default value: false
Particle Size Discard	(GPU only) Discards particles below the defined screen space pixel size. Valid values: 0 – 255 Default value: 0
Streamable	(CPU only) Allows texture or geometry assets to stream from storage as normal. Default value: true

Audio Attribute

Parameters in this attribute determine which sounds are emitted by the particle system and when.

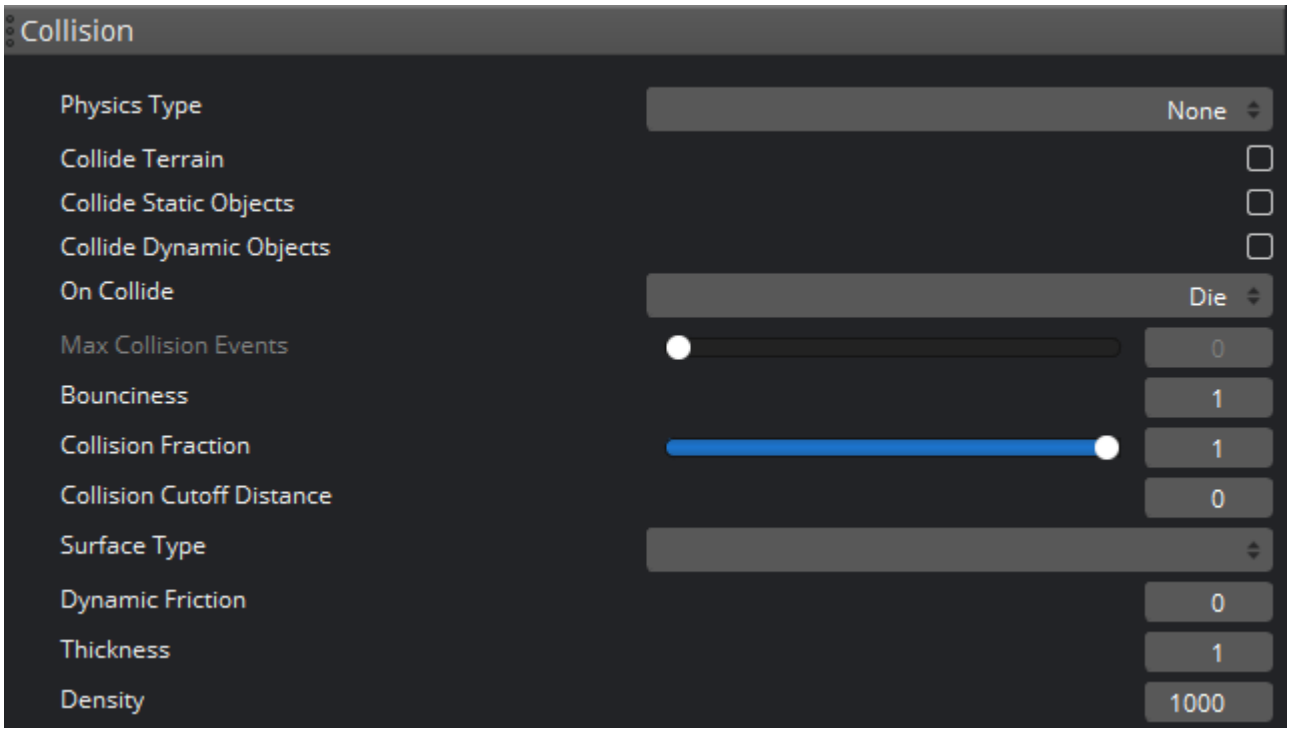


Audio Attribute Parameters for CPU Emitters

Parameter Function	Description
Start Trigger	Selects the start trigger sound asset to play with the emitter.
Stop Trigger	Selects the stop trigger sound asset to play with the emitter.
Sound FXParam	Applies the modulate value to the sound. The effect depends on how the **particlefx** parameter for the individual sound is defined. Depending on the sound, this value might affect volume, pitch, or other attributes. You can set a **Random** value and **Strength Over Emitter Lifetime** curve. Valid values: 0+ Default value: 1
Sound Control Time	• **EmitterLifeTime** – Plays for the length of the emitter's lifetime. • **EmitterExtendedLifeTime** –Plays for the length of the emitter's lifetime plus all particles' lifetimes (until all particles die). • **EmitterPulsePeriod** – Plays for the length of the pulse period.

Collision Attribute

Parameters in this attribute control the particle's physical setup.



CPU Collision Attribute Parameters

Parameter Function	Description
Physics Type	Specifies how the particle interacts physically. • **None** – No collisions or other physics. This setting is used by default. • **SimpleCollision** – Collides the particle with the static environment using simple physics. This is the simplest mode. • **SimplePhysics (Geometry particles)** – Creates a particle in the physics system and collides using a spherical particle model. Use **Surface Type** to set the bounciness. • **RigidBody (Geometry particles)** – Creates a particle in the physics system and collides using the full collision mesh of the assigned geometry. This is the most expensive mode. Use **Surface Type** to set the bounciness.
Collide Terrain	Includes terrain in particle collisions. Default value: false
Collide Static Objects	Includes nonterrain, static objects in particle collisions. This is expensive. Default value: false
Collide Dynamic Objects	Includes nonterrain, dynamic objects in particle collisions. This is expensive. Default value: false

Parameter Function	Description
On Collide	Upon impact with the static environment, the particle dies. • **Die** – The particle dies upon colliding with an object. • **Bounce** – If you enable **Collide Terrain**, **Collide Static Objects**, or **Collide Dynamic Objects**, the particle bounces off the respective type. If none of these are enabled, the selected particle ignores the object that it collides with and passes through. This is useful with **Spawn Indirection** – **Parent Collide**. • **Stop** – The particle stops when it collides with an object. Default value: **Die**
Max Collision Events	Limits the number of collisions that the particle can have in its physics simulation. This affects only particles that have **Physics Type** set to **Rigid Body**. Valid values: 0 – 255 Default value: 0
Bounciness	Controls the elasticity for collision response. This affects only particles that have **Physics Type** set to **Simple Collision**. You can override this feature by setting **Surface Type**. You can also use a special value of –1 to have the particle die on first collision. Valid values: any Default value: 0
Collision Fraction	Determines the fraction of emitted particles that perform collisions. Valid values: 0 – 1 Default value: 1
Collision Cutoff Distance	Specifies the maximum distance from the camera at which collisions are performed. A value of 0 means infinite. Valid values: 0+ Default value: 0
Surface Type	Selects from a variety of surface material types for the collision behavior. If set, **Surface Type** overrides **Bounciness** and **Dynamic Friction**. Default value: none
Dynamic Friction	The coefficient of dynamic friction. If set, **Surface Type** overrides **Dynamic Friction**. This affects only particles that have **Physics Type** set to **Simple Collision**. Valid values: 0+ Default value: 1

Parameter Function	Description
Thickness	Controls the fraction of the particle's visible radius to use for the physical radius. This affects particles that have **Physics Type** set to **Simple Physics** and geometry particles that have **Physics Type** set to **Simple Collision**. Valid values: 0+ Default value: 1
Density	Controls the density of particle in kg/m^3. An example of a physically correct value is Water = 1000. This affects only particles that have **Physics Type** set to **Simple Physics** or **Rigid Body**. Valid values: 0+ Default value: 1000

GPU Collision Attribute Parameters

Parameter Function	Description
Depth Collision	Specifies how the particle interacts physically. • **None** – No collisions or other physics. This setting is used by default. • **Frame Buffer** – Collides with the screen space depth buffer. • **Cubemap** – Collides with a depth cubemap buffer that is generated around the emitter.
Cubemap Far Plane	Sets the far plane distance for cubemap depth buffer generation. Default value: 20
Die on Collide	Upon impact with the static environment, the particle dies. • **Die** – The particle dies upon colliding with an object. • **Bounce** – The particle bounces off the object that it collides with. • **Stop** – The particle stops when it collides with an object.

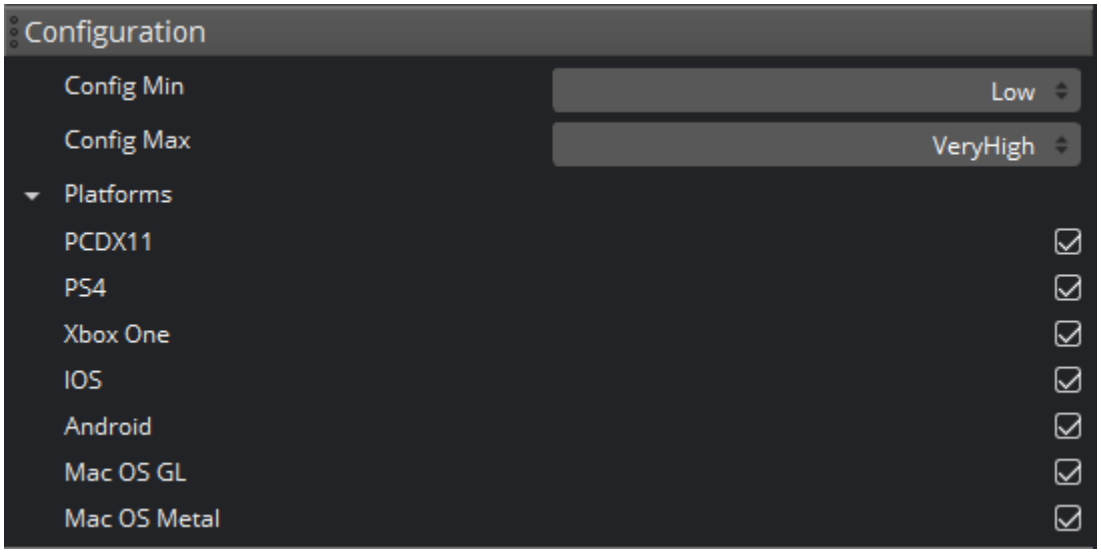
Parameter Function	Description
	Default value: Die
Bounciness	Controls the elasticity for collision response. This affects only particles that have **Physics Type** set to **Simple Collision**. You can override this feature by setting **Surface Type**. You can also use a special value of –1 to have the particle die on first collision. Valid values: any Default value: 0
Thickness	Controls the fraction of the particle's visible radius to use for the physical radius. This affects particles that have **Physics Type** set to **Simple Physics**. Valid values: 0+ Default value: 1

Comment Attribute

Save or edit comments about an emitter in the **Comment** text box.

Configuration Attribute

Parameters in this attribute allow you to specify the minimum and maximum specs for your particle emitters.

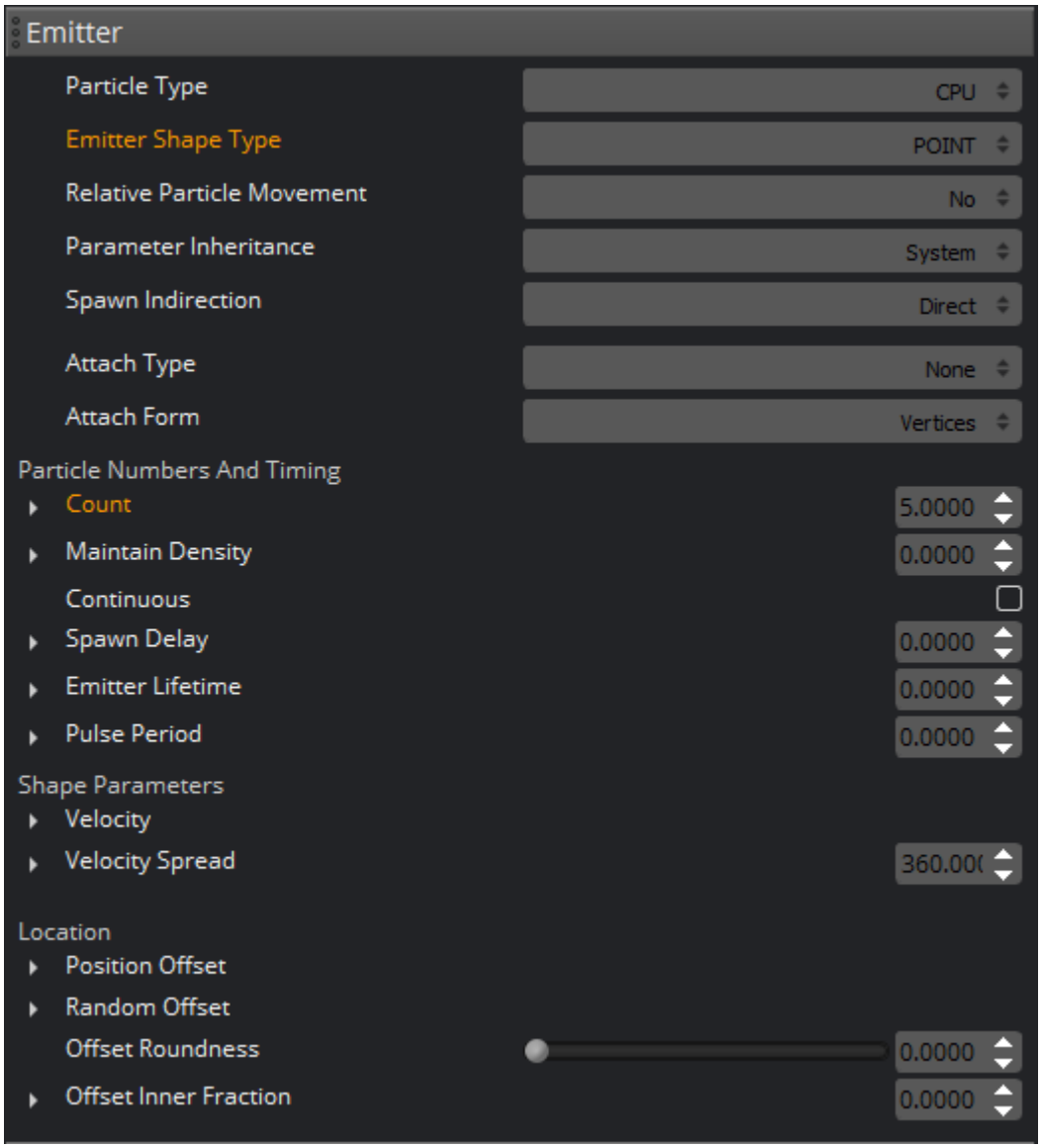


Configuration Attribute Parameters for CPU Emitters

Parameter Function	Description
Config Min	Specifies the minimum system configuration level for the effect. If the configuration is lower than the set value, the item will not appear. Choose from **Low**, **Medium**, **High**, or **VeryHigh**. Default value: **Low**
Config Max	Specifies the maximum system configuration level for the effect. If the configuration is higher than the set value, the item will not appear. Choose from **Low**, **Medium**, **High**, or **VeryHigh**. Default value: **VeryHigh**
Platforms	Defines which platform to use with the effect. • **PCDX11** • **iOS** • **Android** • **macOS GL** • **macOS Metal** Default value: all checked (true)

Emitter Attribute

Parameters in this attribute control the location and spawning attributes of the particle and emitter shape. The parameters are updated based on the type that you select.



Emitter Attribute Parameters

Changing **Emitter Shape Type** resets the attributes to the default settings for the selected emitter. You will receive a confirmation message that you can prevent by selecting **Do not prompt again**. This prompt is reset when you select **Reset to default layout**.

Emitter Attribute Parameters

Parameter Function	Description
Particle Type	Specifies whether the emitter is a CPU or GPU type.
Emitter Shape Type	Specifies the emitter shape type. • **Angle** – Standard emitter type based on angle parameters. This is the default value. • **Point** – Spherical distribution emitter.

Parameter Function	Description
	• **Sphere** – Sphere shape emitter with particles controlled along the sphere's circumference coordinates. • **Circle** – Circle shape emitter with particles controlled along the circumference and vertical coordinates. • **Box** – Box shape emitter with particles controlled along the planes of the box. • **Trail** – Emitter to create trail type effects. • **Beam** – Emitter to create beam type effects.
Relative Particle Movement	Determines the particle motion in the emitter's space. The following is an example of particles that are emitted upward from an emitter: • **No** – The emitted particles stay in world space and fall behind as the emitter moves away, for example, like smoke from the chimney of a train. • **Yes** – The emitted particles stay relative to the emitter's local space and won't fall behind as the emitter is moved. An example is a smoke column going straight up from the chimney and staying vertical with the emitter. This excludes tail particles. • **Yes with Tail** – The emitted particles stay relative to the emitter's local space, including any associated tail particles. Default value: **No**

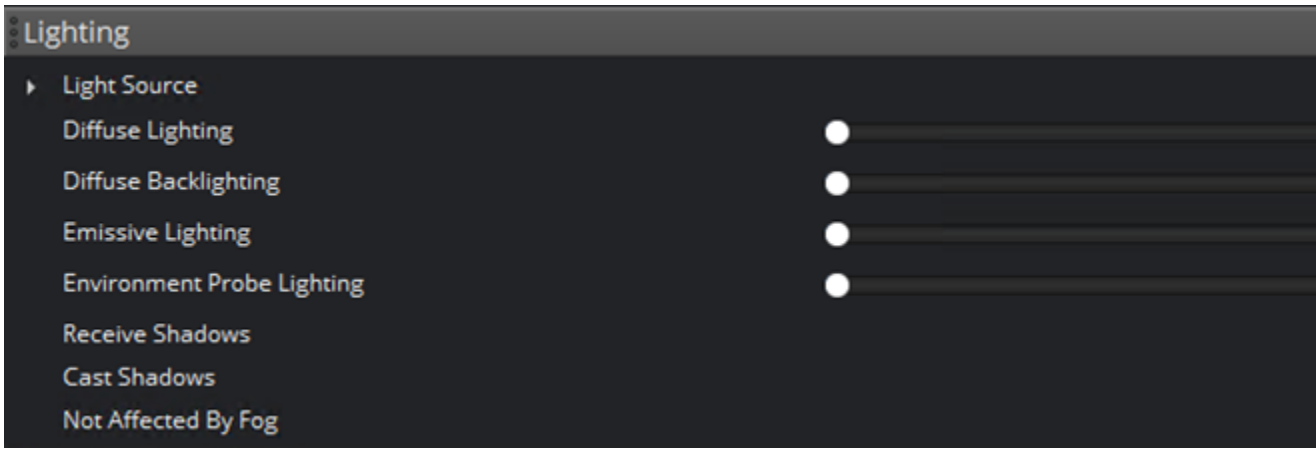
Parameter Function	Description
Parameter Inheritance	Specifies the source for default (starting) effect parameters. • **Standard** (default) – Uses the values set in `DefaultParticleEmitters.xml`. • **System** (deprecated) – Reads the `System.Default.xml` file to use as defaults for the angle emitter. If this effect does not exist, you can use standard defaults. You can generate the `.xml` file by adding a library named **`System`** and an emitter named **`default`**. Save the library to your `\<project>\libs\particles` directory. • **Parent** – Uses the parent particle effect for defaults. This is useful when creating a parent effect with one set of parameters and a variety of subeffects that alter some of the parameters for variation. Subeffects can spawn on their own. Edit the parent effect to update the default values of all subeffects. The selected source has the following consequences: • When you create a new effect, default parameters are set from the following file:`\dev\Editor\Plugins\ParticleEditorPlugin\defaults\DefaultParticleEmitters.xml`. • Changing the inheritance source does not change any other parameters. However, different parameters may be highlighted if their defaults have changed. To reset parameters to the default values that are located in the `DefaultParticleEmitters.xml` file, right-click the parameter and choose **Reset to default**. You can modify the default values for the project using the same file. When you save effects to `.xml` libraries, only nondefault values are saved. When you load effects from the `.xml` libraries, the current default values for the effect's inheritance are used as a base. When you edit a parent effect's parameters, the nonedited parameters of all children (and descendants) that have **Parent** selected are instantly updated.
Spawn Indirection	**Note** Spawn indirection is supported only within the same CPU or GPU particle type. This parameter has the following values: • **Direct** – Spawns without relying on the parent's input for timing. This is the default value. • **ParentStart** – Spawns once the parent has spawned. • **ParentCollide** – Spawns a particle with this setting when the parent particle has collided with an object. • **ParentDeath** – Spawns a particle with this setting when the parent particle has lived out its lifetime.

Parameter Function	Description
Attach Type	Specifies the emission location when the parent emitter has geometry. • **None** – Particles ignore geometry and emit from the emitter center as normal. • **Bounding Box** – Particles emit from the object's bounding box. • **Physics** – Particles emit from the geometry of the attached physics object (mesh or simple primitive). • **Render** – Particles emit from the full mesh of the render object (static or animated mesh). This is generally more CPU-intensive than emitting from physics. Default value: **None**
Attach Form	Specifies the elements of the geometry from which particles emit. • **None** – Not active. • **Vertices** – Emits randomly from the vertices of the geometry. This is the most efficient form of mesh emission. • **Edges** – Emits randomly from the edges of the geometry. This is useful for effects on breaking element pieces. • **Surface** – Emits randomly from the surfaces (faces) of the geometry. • **Volume** – Emits randomly inside the volume of the geometry. Default value: **Vertices**
Count	Specifies the total number of particles that are active at a given time and determines the emission rate (**Count/Particle Lifetime**). You can set a **Random** value and the **Strength Over Emitter Life** curve. Valid values: 0+
Maintain Density	Increases the emission rate (and particle count) when an emitter moves in order to maintain the same spatial density as when motionless. You can scale the increase from 0 to 1. • **Reduce Alpha** – When **Maintain Density** is active, this reduces particle alpha in order to maintain the same overall emitter alpha. Valid values: 0+
Continuous	If false, emits all particles at once and then dies. If true, emits particles gradually over the emitter lifetime. If true and **Emitter Lifetime** = 0, emits particles gradually at a rate of **Count/Particle Lifetime** per second, indefinitely. Default value: false

Parameter Function	Description
Spawn Delay	Delays the start of the emitter for the specified time. This is useful to delay subeffects relative to the overall emitter creation time. Can set a **Random** value. Valid values: 0+ Default value: 0
Emitter Lifetime	If **Continuous** = true, specifies the lifetime of the emitter. Emitter lifetime does not apply to noncontinuous effects, which always disappear as soon as they have emitted all of their particles. You can set a **Random** value. Valid values: 0+ Default value: 0 (infinite lifetime)
Pulse Period	If greater than 0 and **Continuous** = false, restarts the emitter repeatedly at this interval. You can set a **Random** value. Valid values: any Default value: 0
Orient to Velocity	Forces the particle x-axis to align to the velocity direction. You can use **Rotation** parameters to rotate the particle further. Default value: false
Position Offset	XYZ values define the spawning position away from the emitter itself in emitter space. Valid values: any Default values: 0, 0, 0
Random Offset	XYZ values define the range of a random spawning box in both directions, away from the position offset. Valid values: any Default values: 0, 0, 0
Offset Roundness	Specifies the fraction of spawning volume corners to round. Valid values: 0 (box shape) – 1 (ellipsoid shape) Default value: 0
Offset Inner Fraction	Specifies the ratio of inner to outer spawning volume. Valid values: 0 (spawn within entire volume) – 1 (spawn only at surface) Default value: 0

Lighting Attribute

Specify the parameters for the Lighting attribute to control the particle lighting.



Lighting Attribute Parameters

Parameter Function	Description
Light Source	(CPU only) Causes each particle to create a deferred light, where color is equal to the **Color** value. • **Affects This Area Only** – Use with clip volumes. When enabled, the particle lights do not exceed the volume boundary. • **Radius** – Radius of the light. You can set a **Random** value and **Strength Over Emitter Lifetime** and **Strength Over Particle Lifetime** curves. • **Intensity** – Intensity of the light. You can set a **Random** value and **Strength Over Emitter Lifetime** and **Strength Over Particle Lifetime** curves. Valid values: 0+ Default value: false, 0, 0
Diffuse Lighting	Multiplies the particle color for dynamic (diffuse) lighting. Valid values: 0+ Default value: 1
Diffuse Backlighting	Specifies the fraction of diffuse lighting that is applied to unlit particle directions. Valid values: 0 (standard diffuse and normals facing the light are lit the most) – 1 (omnidirectional diffuse and light affects all normals equally). Default value: 0
Emissive Lighting	Multiplies the particle color for constantly emitting light. Adding a value can make a particle appear as if it's glowing. Valid values: 0+

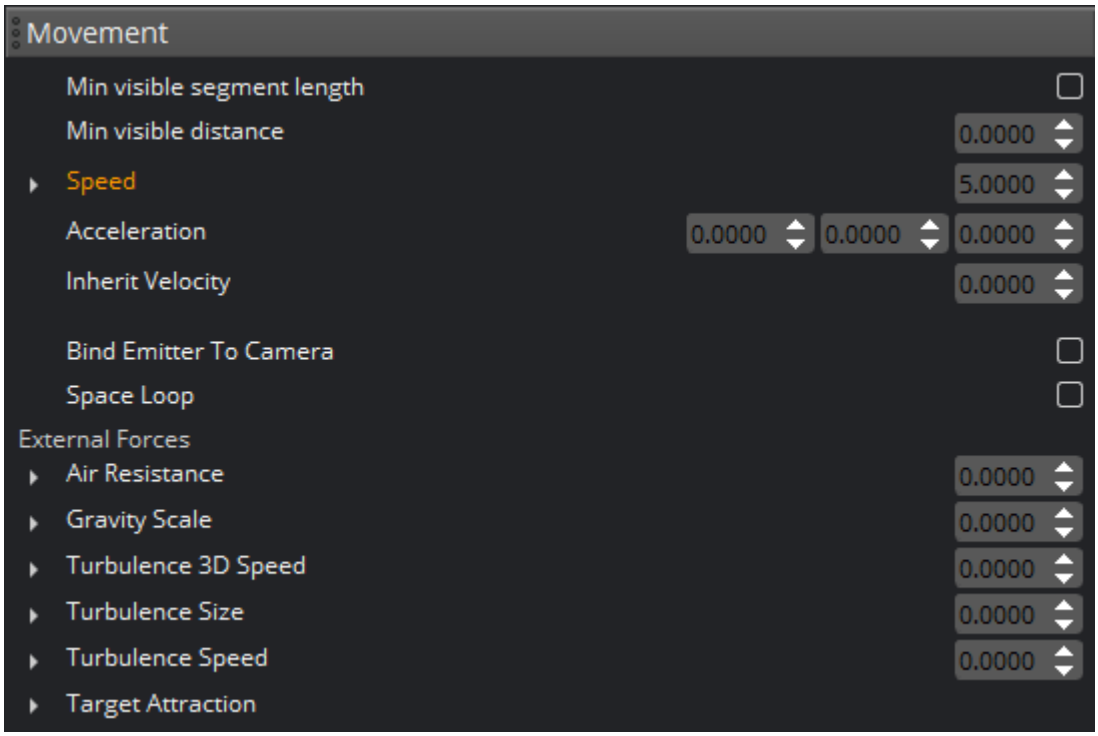
Parameter Function	Description
	Default value: 0
Environment Probe Lighting	(CPU only) Controls the amount of diffuse lighting that is contributed from environment probes. Valid values: 0 – 1 Default value: 0
Receive Shadows	(CPU only) Allows shadows to cast on the particles. Default value: false
Cast Shadows	(GPU and geometry particles only) Allows particles to cast shadows. Default value: false
Not Affected by Fog	(CPU only) Causes particles to ignore scene fog. Default value: false

Note
When there is a single light source, some or all particles can appear unlit (black). To address this:

1. Add an Environment Probe to the scene to create indirect lighting. For more information, see Environment Probe (p. 499).
2. Enable Environment Probe Lighting on the particle system by specifying a value for the Environment Probe Lighting parameter. For example, a value of `0.5` applies light from the Environment Probe at half intensity.

Movement Attribute

Parameters in this attribute control the sprite's movement.



For the **Air Resistance**, **Gravity Scale**, **Turbulence 3D Speed**, **Turbulence Size**, and **Turbulence Speed** parameters, you can set a **Random** value and **Strength Over Emitter Lifetime** and **Strength Over Particle Lifetime** curves.

Movement Attribute Parameters

Parameter Function	Description
Min visible segment length	Trail Emitter (CPU) (p. 1326) only) Tail particles are visible only when they have moved the specified distance. Default value: false
Min visible distance	Trail Emitter (CPU) (p. 1326) only) Specifies the minimum distance between the start and end of a trail segment. Segments that are smaller than this value become transparent. Valid values: 0+ Default value: 0
Speed	Specifies the initial speed of particles. You can set a **Random** value and **Strength Over Emitter Lifetime** curve. Valid values: any Default value: 5
Acceleration	XYZ values define the constant acceleration that is applied to particles in world space. Valid values: any

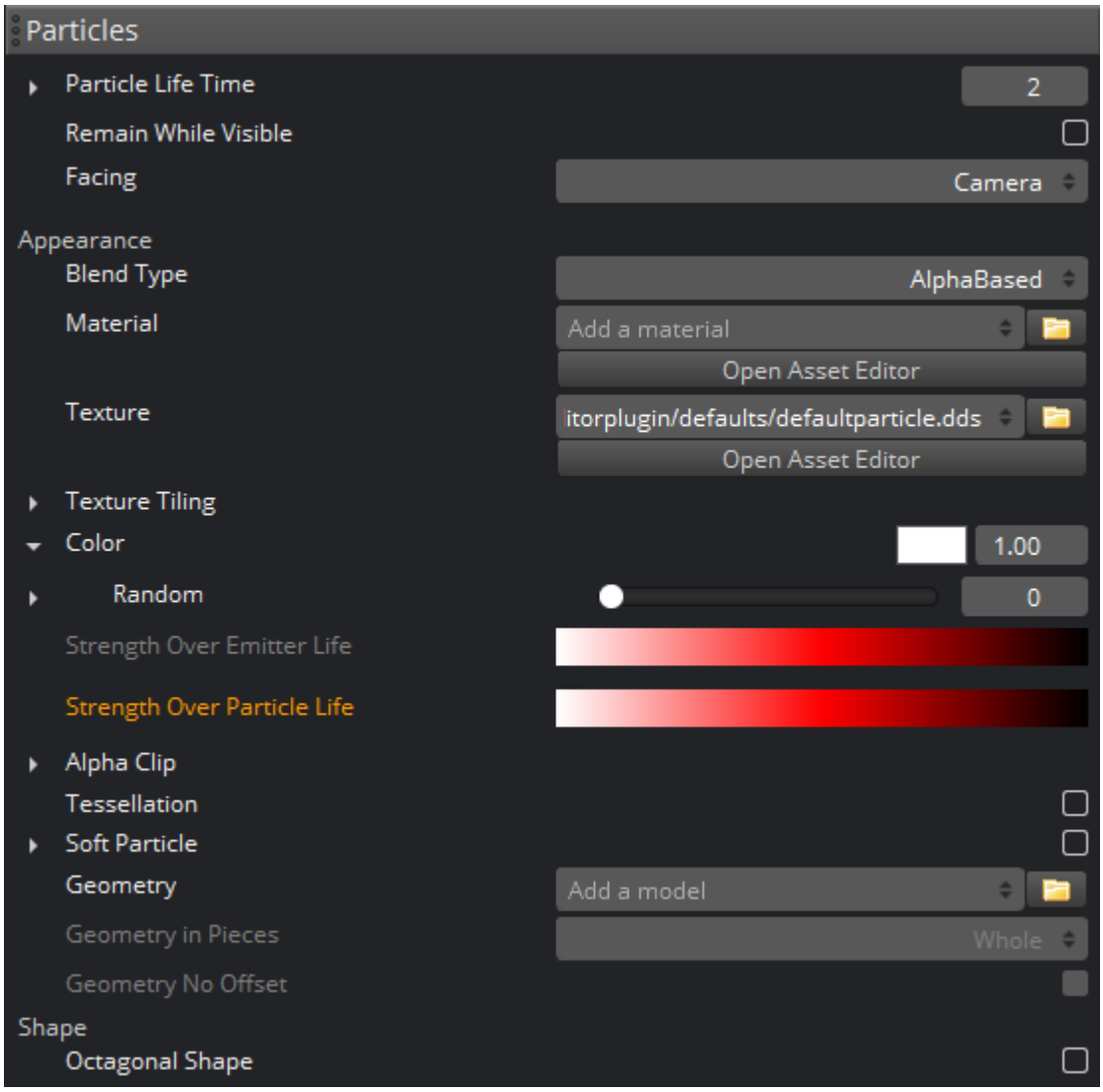
Parameter Function	Description
	Default value: 0, 0, 0
Inherit Velocity	Specifies the fraction of initial velocity that is inherited from the particle's parent. For indirect particles, the parent particle's velocity is inherited. For direct particles, the emitter's velocity is inherited. Valid values: any Default value: 0
Bind Emitter to Camera	(CPU only) Forces the emitter to relocate to the main camera's position. This is useful (with **Space Loop**) for making a rain or snow effect, which the player cannot pass. Default value: false
Space Loop	(CPU only) Loops particles within a region around the camera, as defined by **Camera Min/Max Distance** (under the **Visibility** tab). This is useful for making a rain or snow effect, which has an effective infinite spawning area. Default value: false
Air Resistance	Particles behave as if encountering resistance and slow down over time. Valid values: 0+ Default value: 0
Gravity Scale	Applies multiple of world gravity to particles. Set most physicalized particles to 1 (use air resistance to provide drag). Set the parameter to a negative value for buoyant particles such as smoke. Valid values: any Default value: 0, 0, 0
Turbulence 3D Speed	Adds a 3D, random, turbulent movement to the particle, with the specified average speed. Valid values: 0+ Default value: 0
Turbulence Size	Adds a spiral movement to the particle, with the specified radius. Set the axis of the spiral from the particle's velocity. Valid values: 0+
Turbulence Speed	When **Turbulence Size** is greater than 0, specifies the angular speed, in degrees/second, of the spiral motion. Valid values: any Default value: 0

Parameter Function	Description

Target Attraction	Specifies how particles behave if the emitter is attached to a target. By default, all particles are attracted to any target to which the emitter is linked. These parameters customize that behavior.
	• **Target**
	  • **External** – Particles are attracted to a target entity, if the emitter is linked to one (default).
	  • **OwnEmitter** – Particles are attracted to their 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 the target when reached. Otherwise, they disappear into the target.
	• **Radius** – Distance from the target that particles orbit around or disappear. You can set a **Random** value and **Strength Over Emitter Lifetime** and **Strength Over Particle Lifetime** curves.
	Valid values: any
	Default value: 0

Particles Attribute

Parameters in this attribute control the particle's basic appearance. You should set up this attribute first because it includes the **Texture** slot, which is used for most particles.



Particles Attribute Parameters

Parameter Function	Description
Particle Life Time	Specifies the lifetime of individual particles. After an emitter's lifetime has expired, spawned particles live out their own lifetime. Valid values: 0+
Remain While Visible	Indicates that particles do not die until the entire emitter is out of view. This is typically used for emitters that move through space quickly and die to avoid particle pooling and other unintended consequences. Default value: false
Facing	Determines how the sprite or geometry is oriented in space. You can further modify texture orientation using the following rotational parameters.

Parameter Function	Description
	• **Camera** (default) – Faces the viewer, with 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. In all other modes, particles are lit as flat polygons. • **CameraX** – Rotates about the local y-axis only and faces the camera as much as possible. • **Free** – Rotates freely in 3D. Be sure to give it some rotation. The default orientation is equal to the emitter's orientation. • **Horizontal**– Faces horizontal to the emitter's z-axis. • **Velocity** – Faces the direction of movement. • **Water** – Faces upward and is moved and aligned to the nearest water plane. • **Terrain** – Faces upward and is moved and aligned to the nearest terrain location. • **Decal** – Renders the particle as a deferred decal, projected onto the nearest surface along the entity's y-axis. The **Thickness** parameter under **Collision** controls the projection depth. This parameter works only with materials and does not work with textures. • **Shape** – Faces the nearest edge of the emitter shape boundary. You can use this parameter with the following shape types: **Circle**, **Sphere**, and **Box Emitter**. Default value: **Camera**
Material	Opens the **Asset Browser** for you to assign a material for the 2D sprite particles. Different shaders are uniquely affected by the lighting and environment. Default value: empty
Blend Type	Applies to 2D particles only and determines how the sprite blends with the background. • **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 • **Opaque** – Final color = Particle color Default value: **Alpha Based**
Sorting Method	(GPU only) Enables intra-GPU emitter sorting. • **None** – No sorting • **Bitonic** – Sorts using a bitonic sorting algorithm • **OddEven** – Sorts using an odd even merge sorting Default value: None

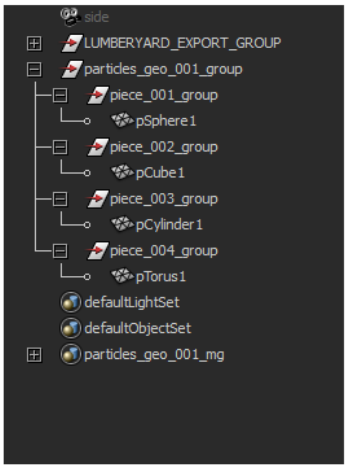
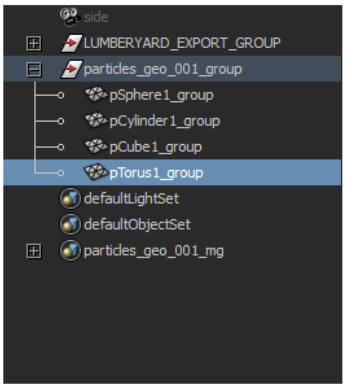
Parameter Function	Description
Sorting Convergence	(GPU only) Sorts convergence per frame for an odd–even merge sort. Default value: 1.0
Texture	Opens the **Asset Browser** for you to assign a texture for the 2D sprite particles. When you hover over the input box, a preview of the texture displays. Default value: `/Editor/Plugins/ParticleEdiotrPlugin/defaultparticle.dds`
Normal Map	(GPU only) Opens the **Asset Browser** for you to assign a normal map for the 2D GPU particles.
Glow Map	(GPU only) Opens the **Asset Browser** for you to assign a glow map for the 2D GPU particles.

Parameter Function	Description

Texture Tiling	Splits the texture into tiles for variation and animation.
	• **Tiles X, Y** – The number of tiles that the texture is split into.
	Valid values: 1 – 256
	Default value: 1
	• **First Tile** – The first of the range of tiles used by the particle.
	Valid values: 0 – 255
	Default value: 0
	• **Variant Count** – The number of consecutive tiles in the texture that the particle randomly selects from.
	Valid values: 1 – 256
	Default value: 1
	• **Anims Frame Count** – The number of tiles that compose an animation sequence. You can use **Variant Count** and **Anim Frames Count** together. For example, if **Variant Count** = 2 and **Anim Frames Count** = 8, the particle randomly chooses between tiles 0 – 7 or 8 – 15, as an animated sequence.
	Valid values: 1 – 256
	Default value: 1
	• **Anim Framerate** – The number of frames per second for the animation. If 0, the animation runs through one sequence in the particle lifetime.
	Valid values: 0+
	Default value: 1
	• **Anim Cycle** – This parameter has the following values.
	  • **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 animation frames. This has a performance impact.
	Default value: false
	• **Horizontal Flip Chance** – Specifies the fraction of particles that are rendered and mirrored in texture X.
	• **Vertical Flip Chance** – Specifies the fraction of particles that are rendered and mirrored in texture Y.
	Valid values: 0 – 1
	Default value: 0
	• **Anim Curve** – Modifies the speed and direction of the animation.

Parameter Function	Description
Color	Selects the color and alpha to apply to a particle. • **Random Between Two Colors** – Right-click the color input and choose the option to randomize between two colors and alpha values. Right-click to choose and return to **Single Color**. • **Random** – Specifies how much a particle's initial color varies from the default. Valid values: 0 = no variation | 1 = random black to default • **Random Hue** – Causes the **Random** color variation to occur separately in the three color channels. If false, the variation is in luminance only. Default value: false • **Strength Over Emitter Lifetime** – Defines the color of the particle over the emitter's lifetime. Double-click to open the **Gradient Editor**. • **Strength Over Particle Lifetime** – Defines the color of the particle over the particle's lifetime. Double-click to open the **Gradient Editor**.

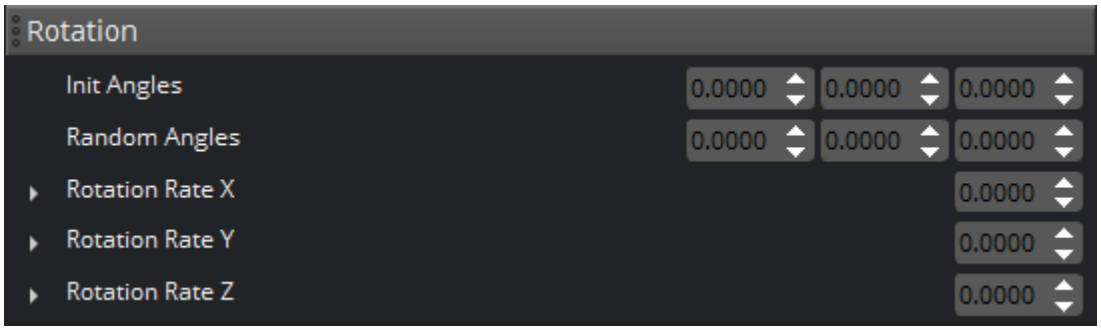
Parameter Function	Description
Alpha clip	(CPU only) Customizes how the particle **Alpha** value controls opacity or alpha test values. Each parameter has two values that correspond when particle **Alpha** = 0 and 1. They are interpolated for each particle with its **Alpha** value and used in the shader with the following equation: `FinalOpacity = saturate( (TextureAlpha - SourceMin) / min(SourceWidth, 1 - SourceMin) ) * Scale` • **Scale** – Multiplies the final alpha value. When set to the default value (0, 1) , the particle **Alpha** directly scales the final opacity. Valid values: 0+ • **Source Min** – Specifies the minimum source (texture) alpha to render (alpha test); values below the minimum become transparent. When set to the default value (0, 0), there is no alpha test. Valid values: 0+ • **Source Width** – Specifies the feathering range of alpha clipping. When set to the default value (1, 1), texture alpha is fully used. A value of 0 specifies hard clipping and a value of 1 specifies soft clipping. Valid values: 0+ Default value: 0 • Default – Alpha controls opacity with no alpha clipping: Scale = (0, 1), Source min = (0, 0), Source width = (1, 1). • Hard clipping at texture alpha = C, with 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 with the 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)
Tessellation	Enables tessellation, rendering more vertices in the sprite. You must have a minimum of DirectX 11. This parameter is useful when **Receive Shadows** is set, increasing the resolution of shadows. It is also useful when **Tail Length** or **Connection** are set, creating smoother curves in connected particles. This produces more accurate lighting when receiving light from point lights. Default value: false

Parameter Function	Description
Soft Particles	Applies rendering that softens the intersection between sprites and nearby objects to prevent unnatural seams. Use sparingly on particles that need it, such as smoke, because this is slightly more expensive. Use the **Softness** subparameter to define the amount of rendering to apply. Default value: false
Motion Blur	(GPU only) Simulates motion blur on GPU particles. Use **Blur Strength** to set the strength of the blur effect.
Geometry	(CPU only) Opens the **Asset Browser** window to select a 3D object to use for the particles. Default value: empty
Geometry in Pieces	(CPU only) Emits the geometry in pieces, originating at each piece's location in the asset. This applies only if the **Geometry** asset contains multiple subobjects. • **Whole** – Emits the whole geometry asset as one object. • **Random Pieces** – Emits geometry fragments randomly over the emitter lifetime. • **All Pieces** – Emits geometry pieces in order over the emitter lifetime. Default value: Whole In order for this feature to work, you must provide specific names for the nodes in the Maya outliner. The following images show examples of acceptable node names:   In these examples, the groups and individual nodes have a `_group` suffix. As a result, the Maya exporter assumes there is no geometry. If you receive a "group has no geometry" error, you can safely ignore it.

Parameter Function	Description
Geometry No Offset	(CPU only) Uses the geometry pivot for centering geometry particles. Default value: false
Octagonal Shape	(CPU only) 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

Rotation Attribute

Parameters in this attribute control the particle's rotation.

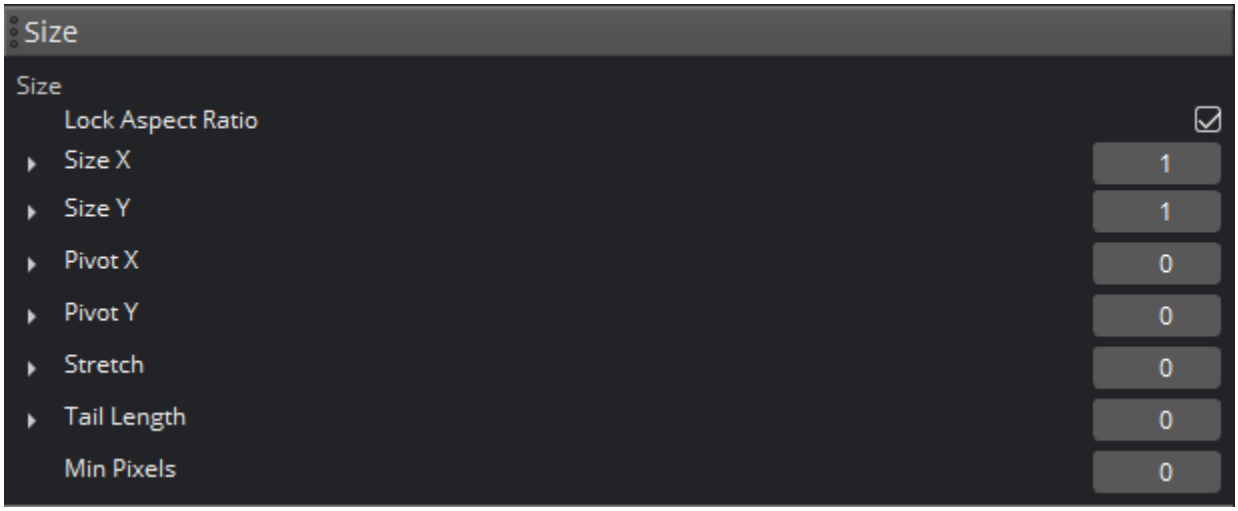


Rotation Attribute Parameters

Parameter Function	Description
Init Angles	XYZ values define the initial angle, in degrees, that is applied to the particles upon spawning. 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. Valid values: any Default value: 0
Random Angles	XYZ values define the random variation (bidirectional), in degrees, to **Init Angles**. Valid values: 0+ Default value: 0
Rotation Rate X, Y, Z	Specifies the constant particle rotation in degrees/second. The axes are the same as **Init Angles**. You can set a **Random** value and **Strength Over Emitter Lifetime** and **Strength Over Particle Lifetime** curves. Valid values: any Default value: 0

Size Attribute

Parameters in this attribute control the sprite's size and shape.



For the **Size**, **Pivot**, and **Stretch** parameters, you can set a **Random** value and **Strength Over Emitter Lifetime** and **Strength Over Particle Lifetime** curves.

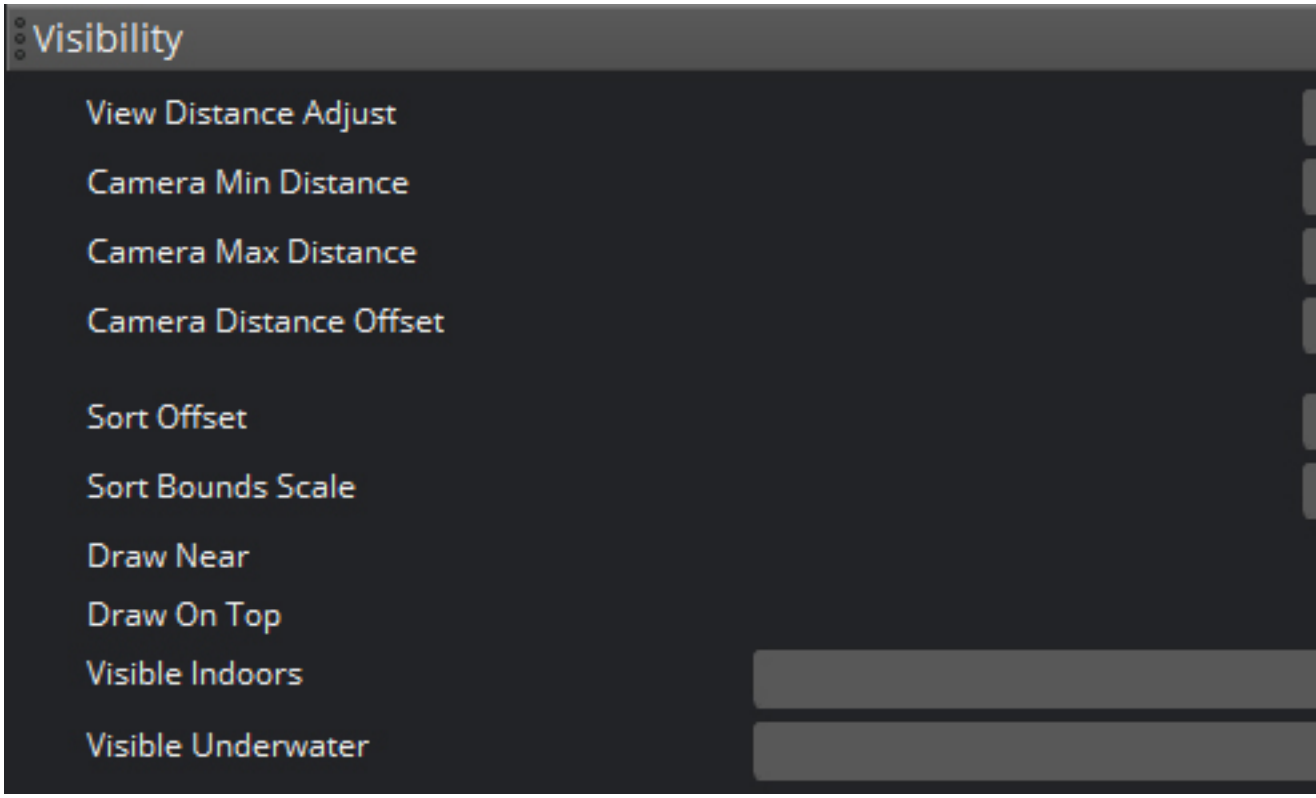
Size Attribute Parameters

Parameter Function	Description
Lock Aspect Ratio	Maintains the particle aspect ratio. Default value: false
Size X, Y	For 2D particles, specifies the world sprite radius. Valid values: 0+ Default value: 1
Pivot X, Y	Moves the sprite's pivot point. Valid values: -1 – +1 Default value: 0 (texture center)
Stretch	Specifies the amount of stretch, in seconds, that is applied to the particle in the direction of travel. This is based on the current velocity and 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. Valid values: 0+ Default value: 0
Tail Length	(CPU only) The length of the particle's tail in seconds. The particle texture is stretched through the tail. **Tail Steps** – The number of segments for the tail. A higher number produces smoother tail curves for nonlinear, moving particles.

Parameter Function	Description
	Valid values: 0+ Default value: 0
Min Pixels	(CPU only) Adds the specified number of pixels to the particle's true size when rendering. This is useful for important effects that should be visible even at a distance. Valid values: 0+ Default value: 0

Visibility Attribute

Parameters in this attribute control the particle's visibility.



Visibility Attribute Parameters

Parameter Function	Description
View Distance Adjust	Multiplies the automatically computed fade-out camera distance. Valid values: 0+ Default value: 1
Camera Min/Max Distance	Determines the camera range that particles render in. A default value of 0 indicates an unlimited range.

Parameter Function	Description
	Valid values: 0+ Default value: 0
Camera Distance Offset	Offsets the emitter away from the camera. Valid values: any Default value: 0
Fade Strength Min/Max Distance	(GPU only) Specifies the distance from the camera at which particles fade. • **Fade Strength Min Distance** – Minimum distance from the camera at which particles fade. • **Fade Strength Max Distance** – Maximum distance from the camera at which particles fade. Default value: 0, 0
Sort Offset	Bias the distance used for sorting. You can customize the sort order in an emitter tree. By default, subemitters render in the order they are listed in the effect. A bias of 0.01 or greater overrides that order. You can use larger biases to adjust the sorting order, with respect to other transparent objects in the level. Valid values: any Default value: 0
Sort Bounds Scale	Specifies the emitter point for sorting. 1 = bounds nearest | 0 = origin | -1 = bounds farthest Valid values: any Default value: 0
Draw Near	Renders particles in a near first-person space (with weapons). Default value: false
Draw on Top	Renders particles on top of everything (no depth test). Default value: false
Visible Indoors	For use in VisAreas. • **If_False** – Hides particles when indoors. • **If_True** – Hides particles when outdoors. • **Both** – Shows particles when indoors or outdoors. This is the default value.

Parameter Function	Description
Visible Underwater	For use with ocean and water volumes. • **If_False** – Hides particles when under water. • **If_True** – Hides particles when above water. • **Both** – Shows particles when under or above water. This is the default value.

Creating and Managing Particle Emitters

You can create, edit, and manage emitters for particle effects in the **Particle Editor**. For more information about emitter attributes and parameters displayed in the **Attributes** panel, see Particles Attributes Reference (p. 1281).

Before you can create or edit emitters, you must first set up a particle library. For more information, see Adding Particle Libraries (p. 1275).

Creating Emitters

Do the following to create an emitter. You must have already created a library.

To create new emitters

1. In Lumberyard Editor, choose **Tools**, **Particle Editor**.
2. In the **Libraries** panel, under your selected library, click **Add Particle**. Alternatively, right-click the library name and then choose **Add New**, **Add Particle**.
3. Type a name for the emitter and press **Enter**. Do not use special characters in the name.
4. In the **Attributes** panel, edit the attributes and parameters as needed.

Editing Emitters

Do the following to edit emitter attributes and parameters.

To edit emitter attributes and parameters

1. In Lumberyard Editor, click **Tools**, **Particle Editor**.
2. In the **Libraries** panel, choose an emitter.
3. In the **Attributes** panel, adjust the attribute and parameter settings and values to achieve the intended effect.

Duplicating Emitters

Do the following to duplicate an emitter. This will also duplicate any associated child emitters.

To duplicate emitters

1. In Lumberyard Editor, click **Tools**, **Particle Editor**.
2. In the **Libraries** panel, select the emitter that you want to duplicate. Right-click the emitter and choose **Duplicate**. Alternatively, press **Ctrl+D**.
3. In the dialog box, type a name for the emitter and click **OK**.

Topics

Creating Child Emitters

To create a child emitter, you first set the parent effect and then attach the child emitter to the parent particles. You can attach multiple child emitters to the parent particle. A particle effect can have any number of child effects (also called subeffects), which you can nest in a library by dragging and dropping where needed.

To create child emitters

1. In Lumberyard Editor, choose **Tools**, **Particle Editor**.
2. In the **Libraries** panel, right-click the emitter for which you want to create a child emitter. Choose **Add New**, **Add Particle**.
3. Type a name for the child emitter and then press **Enter**.

To assign an existing emitter to be a child

- Drag the emitter on top of another emitter. The selected emitter is nested underneath as a child.

To remove a child emitter from a parent

- Drag the child emitter to the preferred location. Alternatively, you can drag the child emitter to the library name to detach it from its parent and make it appear at the bottom of the emitter stack.

Lumberyard has two types of child effects:

- **Regular child effects** – These effects behave like separate effects, though 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 effects are attached to 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 more complex effects. You can nest second-generation effects multiple times in order to create third-generation (and greater) effects.

An example of a child effect is attaching an emitter to a parent particle and leaving trailing particles behind. You can achieve this effect using the **Spawn Indirection** attribute.

Organizing Emitters in a Library

All particle emitters are listed in the **Libraries** panel. You can organize and create relationships between your emitters. For example, you can have single emitters or emitters with various child hierarchies. You can also create directories and groups in each library to organize your particle effects. This relationship is displayed in a tree hierarchy in the **Libraries** panel.

A visual indicator shows you the placement of an emitter based on the position of your cursor. For example, if you place an emitter on another emitter or a folder for grouping, the folder row appears highlighted with a blue stroke.

Reverting Changes to Emitter Attributes

Emitters have a list of attributes or property types that are categorized for easier identification. You can reorder and rearrange the categories, including combining categories into a tabbed view.

- To reorder categories, drag and drop the category to the preferred position. An orange highlight appears to indicate a valid docking location.
- To combine categories into tabs, drag a category on another category title bar. If you have not expanded the category, the tabs appear collapsed until you click a category title bar.
- To revert changes made to the **Attributes** panel layout, click the menu in the title bar and choose **Reset to default**.

By default, emitter parameters are set to the attribute as a common starting point. These default attributes have a white text label. When you change the attribute parameter from the default state, the text label changes from white to orange.

- To revert the last change to the emitter attribute, click **Edit**, **Undo**. Alternatively, you can press **Ctrl+Z**.
- To revert all changes to the attributes parameter, right-click the attribute name and choose **Reset to default**.

Working with Emitter Shape Types

The Particle Editor provides emitter shape types that you can use to achieve a variety of effects, such as using the **Beam** emitter to create fire effects.

To choose the emitter shape type

1. In Lumberyard Editor, choose **Tools**, **Particle Editor**.
2. In the **Libraries** panel, choose an emitter.
3. In the **Attributes** panel, choose **Emitter**.
4. For **Emitter Shape Type**, choose the shape type. The following emitter shape types are supported:
   - **Angle**
   - **Beam**
   - **Box**
   - **Circle**
   - **Point**
   - **Sphere**
   - **Trail**

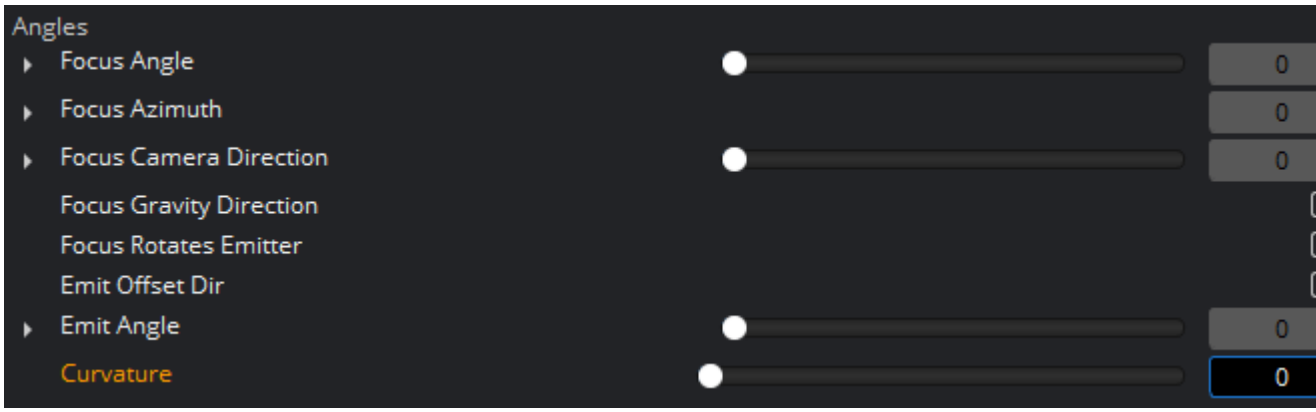
Most emitters have CPU and GPU particle types. Each emitter's velocity functions on either world or shape axial coordinates, based on type. To view the shape XYZ coordinates, right-click the **Preview** viewport and choose **Show Emitter Shape**.

Topics

Angle Emitter (CPU and GPU)

The **Angle** emitter spawns particles in a directional pattern based on the parameters. This is done to control the angle of emission from its source.

The following are parameters for the **Angle** emitter (CPU).



Angle Emitter Parameters for CPU Attributes

Parameter	Description
Focus Angle	Specifies the number of degrees to rotate from the y-axis. Valid values: 0 (up) – 180 (down) Default value: 0
Focus Azimuth	Specifies the number of degrees to rotate the new axis about the y-axis. Valid values: any (0, 360 = North, 90 = West, 180 = South, 270 = East) Default value: 0
Focus Camera Direction	Sets the focus direction to face the camera. You can set a **Random** value and the **Strength over Emitter Lifetime** curve. Valid values: 0 – 1

Parameter	Description
	Default value: 0
Focus Gravity Direction	Ignores the emitter rotation and focuses gravity on world Z. Default value: false
Focus Rotates Emitter	Default value: false
Emit Offset Direction	If true, changes each particle's emission direction to align with its offset from the origin. Default value: false
Emit Angle	Sets the angle deviation of an emitted particle from the default focus (+Y) axis. 0 = up, 90 = horizontal, 180 = down. This is the maximum angle from the focus. You can set a **Random** value (determines minimum angle) and the **Strength over Emitter Lifetime** curve. • 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. Valid values: 0 – 180 Default value: 0
Curvature	Sets how far to bend the vertex normals for **Facing=Camera** particles into a spherical shape. This affects lighting. Valid values: 0 (flat) – 1 (hemispherical shape) Default value: 1

The following are parameters for the **Angle** emitter (GPU).

Angle Emitter Parameters for GPU Attributes

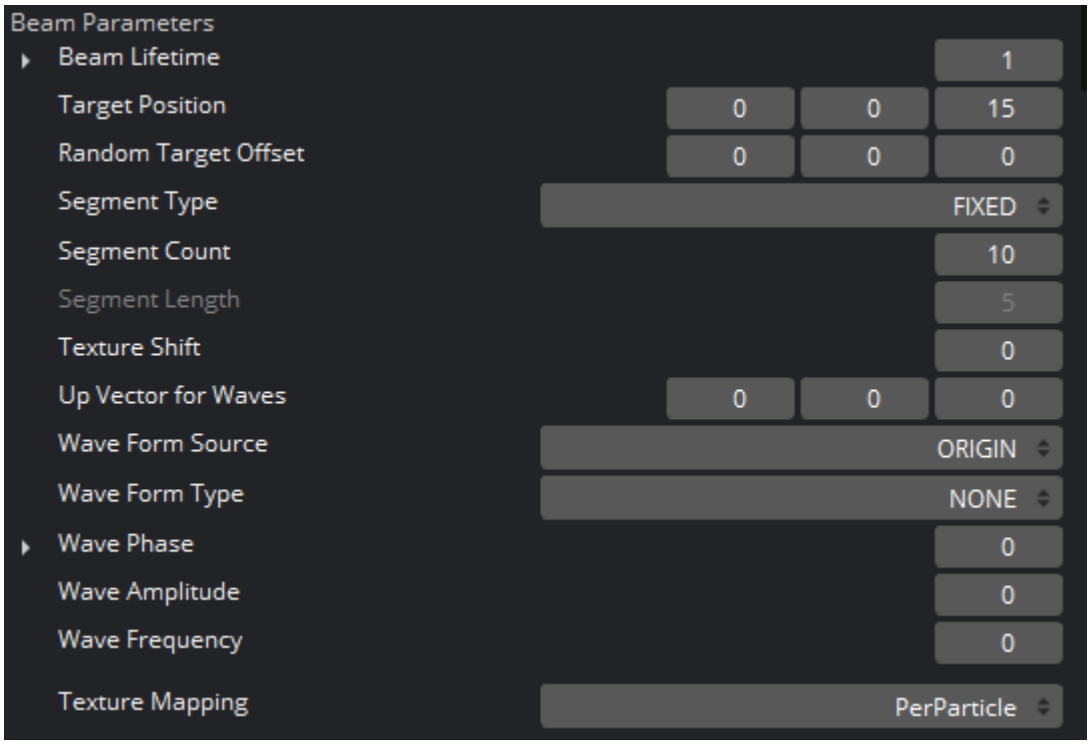
Parameter	Description
Focus Angle	Specifies the number of degrees to rotate from the y-axis Valid values: 0 (up) – 180 (down) Default value: 0
Focus Azimuth	Specifies the number of degrees to rotate the new axis about the y-axis.

Parameter	Description
	Valid values: any (0, 360 = North, 90 = West, 180 = South, 270 = East) Default value: 0
Emit Angle	Sets the angle deviation of an emitted particle from the default focus (+Y) axis. 0 = up, 90 = horizontal, 180 = down. This is the maximum angle from the focus. You can set a **Random** value (determines minimum angle) and the **Strength over Emitter Lifetime** curve • 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. Valid values: 0 – 180 Default value: 0

Beam Emitter (CPU)

The **Beam** emitter spawns a length of connected particles at once, based on the origin and specified target position. You can also set wave form attributes to shape or animate the beam.

The following are parameters for the **Beam** emitter (CPU).



Beam Emitter Parameters for CPU Attributes

Parameter	Description
Beam Lifetime	Sets the beam lifetime, in seconds. You must use this in conjunction with particle lifetime. Valid values: 0+ Default value: 1
Target Position	Specifies the XYZ position to set the beam target, offset from the origin. Default value: 0, 0, 15
Random Target Offset	Randomizes the XYZ beam target, offset from the target position. Valid values: any Default value: 0, 0, 0
Segment Type	Specifies how each segment generates over the length of the beam. Use **Fixed** to set the number of segments over the length. Use **Length** to set the length of each segment over the beam length. Valid values: **Fixed**, **Length** Default value: **Fixed**
Segment Count	Defines the number of segments with the **Fixed** type set. Valid values: 0+ Default value: 10
Segment Length	Defines the length of each segment with the **Length** type set. Valid values: 0+ Default value: 5
Texture Shift	Shifts the texture coordinate in the V direction at the specified rate. Valid values: 0+ Default value: 0
Up Vector for Waves	Specifies the XYZ vector on which the waveform occurs. Valid values: any Default value: 0, 0, 0
Wave Form Source	Specifies the side of the beam from which to generate the waveform. • **Origin** – Waveform is calculated from the emitter origin. • **Target** – Waveform is calculated from the target specified by the target attributes.

Parameter	Description
	Default value: **Origin**
Wave Form Type	Sets the behavior for the waveform. • **None** – No waveform is used. • **Sine** – A periodic waveform generates a smooth, repetitive curve. • **Square** – A nonsinusoidal, periodic waveform in which the amplitude alternates. • **Noise** – A repetitive waveform with a randomized amplitude. Default value: **None**
Wave Phase	Sets the position of the waveform at the specified source. A value of 360 completes one full cycle of the waveform. You can animate the beam waveform by using **Emitter Lifetime**, **Pulse Period**, and **Strength over Emitter Lifetime** in conjunction. **Note** The beam animation is updated when a new one is spawned based on the **Beam Lifetime** parameter. A lower lifetime provides a smoother animation. The following example demonstrates the wave phase parameter: Beam lifetime = .01, Emitter lifetime = 2, Pulse period = 2, Up vector = 1, 0, 0, Wave form type = Sine, wave phase = 180 (with a curve set on **Strength over Emitter Lifetime**), Wave amplitude = 2, Wave frequency = 0.5 Valid values: 0+ Default value: 0
Wave Amplitude	Sets the strength of the waveform deformation. Valid values: 0+ Default value: 0
Wave Frequency	Sets the number of wave cycles over the waveform length. Valid values: 0+ Default value: 0
Texture Mapping	Maps the assigned texture to each particle quad or across the trail stream. • **Per Particle** – Texture is mapped on each quad. • **Per Stream** – Texture is mapped over the length of the beam. Default value: **Per Particle**

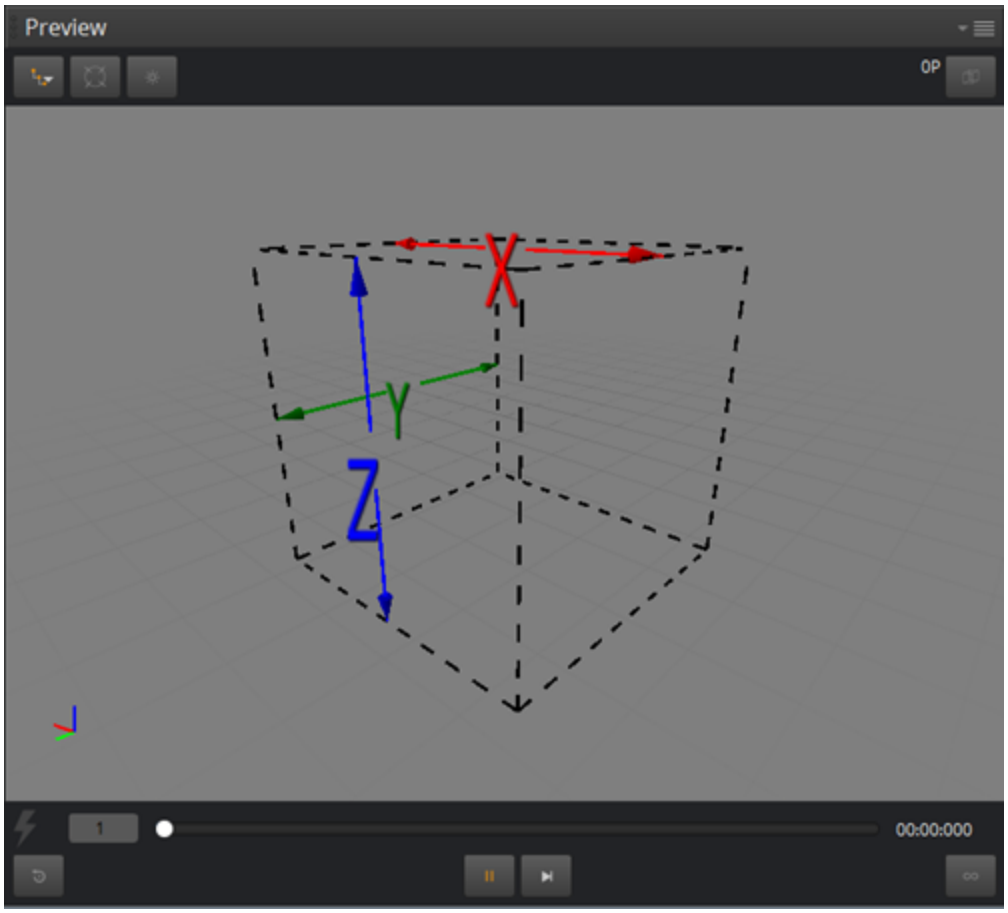
Note
When there is a single light source, some or all particles for **Trail** emitters can appear unlit (black). This is most apparent when the sun is the only light source and the time and day setting is noon. To address this:

1. Add an Environment Probe to the scene to create indirect lighting. For more information, see Environment Probe (p. 499).
2. Enable Environment Probe Lighting on the particle system by specifying a value for the Environment Probe Lighting parameter. For example, a value of `0.5` applies light from the Environment Probe at half intensity. For more information, see Lighting Attribute (p. 1295).

Box Emitter (CPU and GPU)

The **Box** emitter enables spawning of particles along the shapes axial coordinate system. This allows you to create complex planar and rectangular effects.

The following is an example of the **Box** emitter in the **Preview** viewport.



The following are parameters for the **Box** emitter.

Box Emitter Parameters

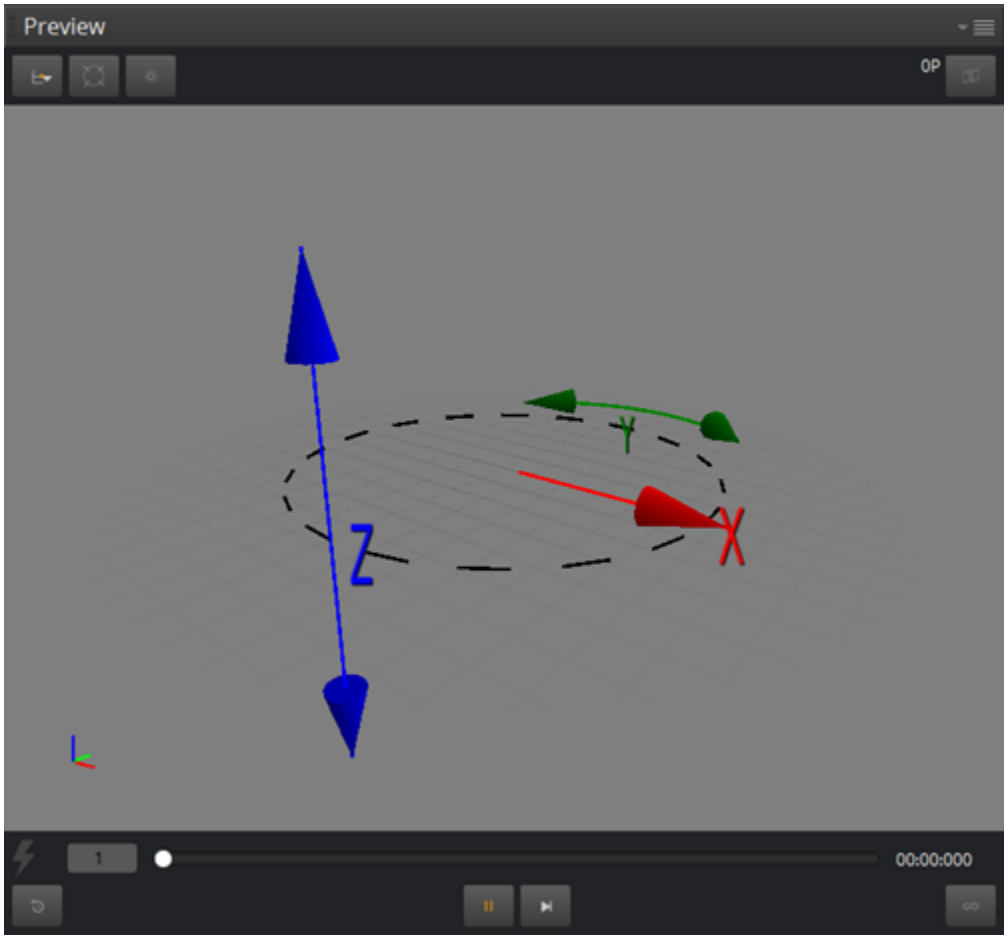
Parameter	Description
Emitter Size XYZ	Sets the size of the box emitter in meters. Valid values: 0+ (radius) Default value: 5
Confine XYZ	Confines the particles to render within the box size on each axis. Default value: false
Spawn Pos XYZ	XYZ values define the spawning position away from the emitter in emitter space. You can set a **Random** value and **Strength over Emitter Lifetime** curve. Valid values: any Default value: 0, 0, 0
Spawn Pos XYZ Random	For additional random layering, XYZ values define the range of random spawning in both directions, away from the spawn position. You can set a **Random** value and **Strength over Emitter Lifetime** curve. Valid values: any Default value: 0, 0, 0
Velocity XYZ	XYZ values define the velocity applied to particles in world space. The velocity direction also sets the axis in which velocity spread occurs around. You can set a **Random** value and **Strength over Emitter Lifetime** curve. Valid values: any Default values: 0, 0, 0
Velocity XYZ Random	For additional random layering, XYZ values define the random velocity applied to particles in world space. You can set a **Random** value and **Strength over Emitter Lifetime** curve. Valid values: any

Parameter	Description
	Default value: 0, 0, 0

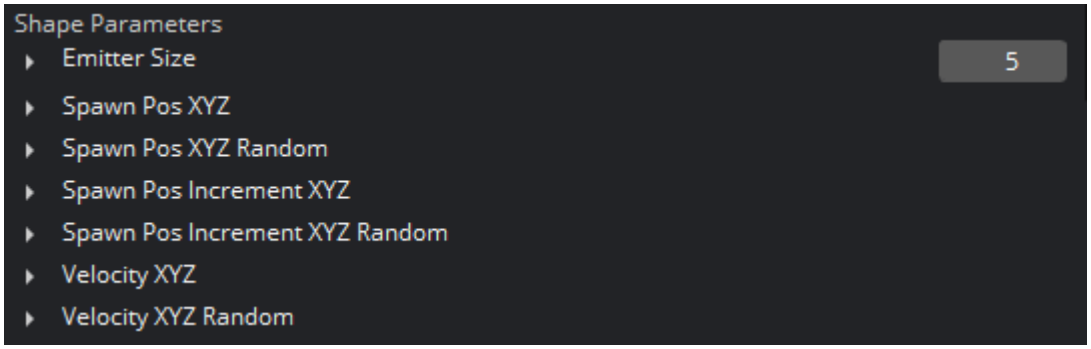
Circle Emitter (CPU and GPU)

The **Circle** emitter enables spawning of particles along the shapes axial coordinate system (see diagram below). This allows you to create complex circular and cylindrical effects.

The following is an example of the **Circle** emitter in the **Preview** viewport.



The following are parameters for the **Circle** emitter.



Circle Emitter Parameters

Parameter	Description
Emitter Size	Sets the size of the circle emitter in meters. Valid values: 0+ (radius) Default value: 5
Spawn Pos XYZ	XYZ values define the spawning position away from the emitter itself in emitter space. You can set a **Random** value and **Strength over Emitter Lifetime** curve. Valid values: any Default value: 0, 0, 0
Spawn Pos XYZ Random	For additional random layering, XYZ values define the range of random spawning in both directions, away from the spawn position. You can set a **Random** value and **Strength over Emitter Lifetime** curve. Valid values: any Default value: 0, 0, 0
Spawn Pos Increment XYZ	Spawns each subsequent particle at incremental positions along the axis over the emitter size, based on a percentage value. For example, if you set a value of 20, a particle spawn every 20% along the axis, making 5 particles on the axis to equal 100%. You can set a **Random** value and **Strength over Emitter Lifetime** curve. Valid values: any (percentage) Default value: 0
Spawn Pos Increment XYZ Random	Provides another layer of randomization for each particle to spawn at percentages over the axis, independent of the **Spawn Pos Increment** parameter. You can set a **Random** value and **Strength Over Emitter Lifetime** curve. Valid values: any (percentage) Default value: 0
Velocity XYZ	XYZ values define the velocity applied to particles in world space. The velocity direction also sets the axis around which velocity spread occurs. You can set a **Random** value and **Strength over Emitter Lifetime** curve. Valid values: any Default values: 0, 0, 0
Velocity XYZ Random	For additional random layering, XYZ values define the random velocity applied to particles in world space. You can set a **Random** value and **Strength over Emitter Lifetime** curve. Valid values: any

Parameter	Description
	Default value: 0, 0, 0

Point Emitter (CPU and GPU)

The **Point** emitter spawns particles in a spherical distribution from the origin. This allows you to control the spread angle to create complex conical and spherical effects.

The following are parameters for the **Point** emitter.

Point Emitter Parameters

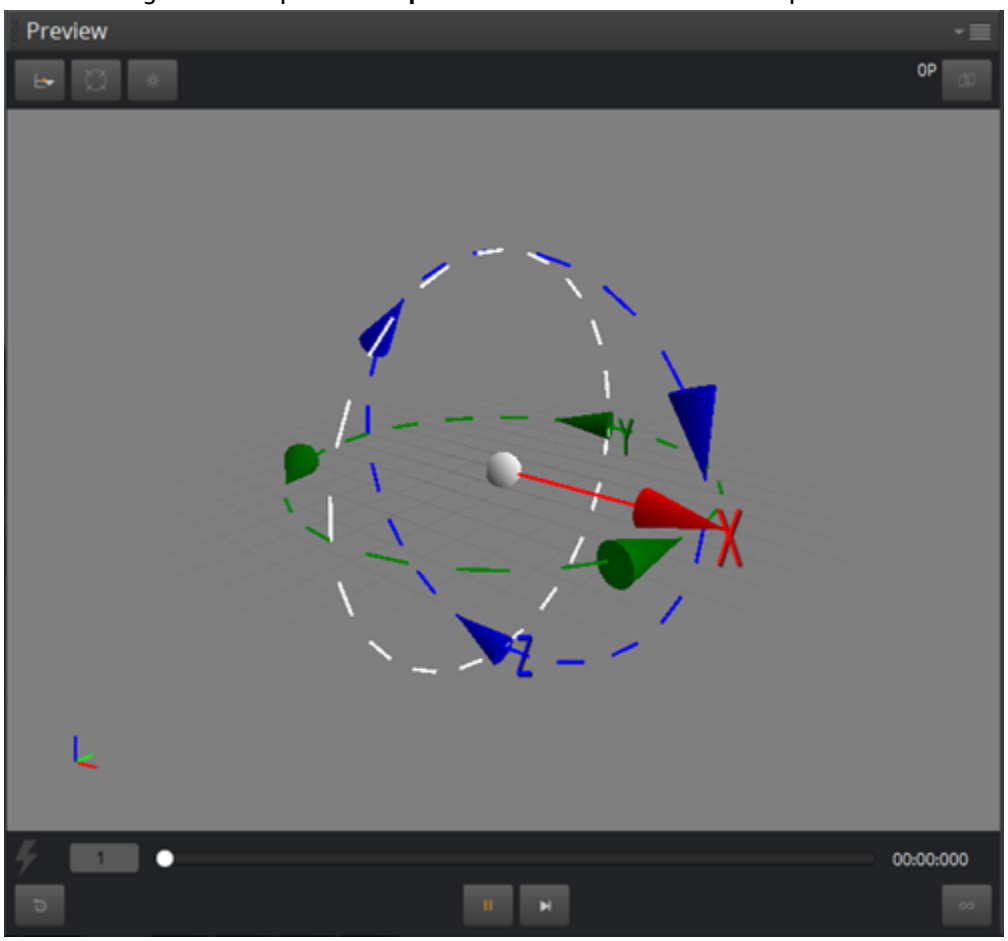
Parameter	Description
Spawn Offset	Offsets the spawning of particles spherically at distance from the emitter origin. Valid values: 0+ (radius) Default value: 0
Spawn Pos XYZ	XYZ values define the spawning position away from the emitter itself in emitter space. You can set a **Random** value and **Strength over Emitter Lifetime** curve. Valid values: any Default value: 0, 0, 0
Spawn Pos XYZ Random	For additional random layering, XYZ values define the range of random spawning in both directions, away from the spawn position. You can set a **Random** value and **Strength over Emitter Lifetime** curve. Valid values: any Default value: 0, 0, 0
Velocity XYZ	XYZ values define the velocity applied to particles in world space. The velocity direction also sets the axis around which velocity spread occurs. You can set a **Random** value and **Strength over Emitter Lifetime** curve. Valid values: any Default values: 0, 0, 0

Parameter	Description
Velocity XYZ Random	For additional random layering, XYZ values define the random velocity applied to particles in world space. You can set a **Random** value and **Strength over Emitter Lifetime** curve. Valid values: any Default value: 0, 0, 0
Velocity Spread	Restricts the angle of spherical distribution in the direction of the velocity XYZ vector. For example, a value of 360 = Sphere, 180 = Hemisphere, and so on. You can set a **Random** value and **Strength over Emitter Lifetime** curve. Valid values: 0 – 360 Default value: 360

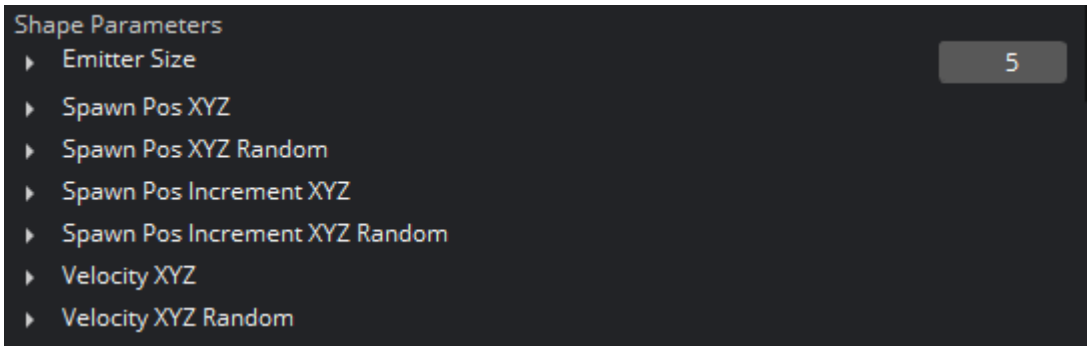
Sphere Emitter (CPU and GPU)

The **Sphere** emitter enables spawning of particles along the shapes axial coordinate system (see diagram below). This allows you to create complex spherical effects.

The following is an example of the **Sphere** emitter in the **Preview** viewport.



The following are parameters for the **Sphere** emitter.



Sphere Emitter Parameters

Parameter	Description
Emitter Size	Sets the size of the sphere emitter in meters. Valid values: 0+ (radius) Default value: 5
Spawn Pos XYZ	XYZ values define the spawning position away from the emitter itself in emitter space. You can set a **Random** value and **Strength over Emitter Lifetime** curve. Valid values: any Default value: 0, 0, 0
Spawn Pos XYZ Random	For additional random layering, XYZ values define the range of random spawning in both directions, away from the spawn position. You can set a **Random** value and **Strength over Emitter Lifetime** curve. Valid values: any Default value: 0, 0, 0
Spawn Pos Increment XYZ	Spawns each subsequent particle at incremental positions along the axis over the emitter size, based on a percentage value. For example, if you set a value of 20, a particle spawns every 20% along the axis, making 5 particles on the axis to equal 100%. You can set a **Random** value and **Strength over Emitter Lifetime** curve. Valid values: any (percentage) Default value: 0
Spawn Pos Increment XYZ Random	Provides another layer of randomization for each particle to spawn at percentages over the axis, independent of the **Spawn Pos Increment** parameter. You can set a **Random** value and **Strength Over Emitter Lifetime** curve. Valid values: any (percentage) Default value: 0

Parameter	Description
Velocity XYZ	XYZ values define the velocity applied to particles in world space. The velocity direction also sets the axis around which velocity spread occurs. You can set a **Random** value and **Strength over Emitter Lifetime** curve. Valid values: any Default values: 0, 0, 0
Velocity XYZ Random	For additional random layering, XYZ values define the random velocity applied to particles in world space. You can set a **Random** value and **Strength over Emitter Lifetime** curve. Valid values: any Default value: 0, 0, 0

Trail Emitter (CPU)

The **Trail** emitter connects particles together to create a trailing effect as the system moves through space.

The following are parameters for the **Trail** emitter.

Trail Emitter Parameters

Parameter Function	Description
Connect to Origin	Connects a newly spawned particle to the origin. Default value: false
Texture Mirror	Mirrors alternating texture tiles. Default value: true
Texture Frequency	Specifies the number of texture wraps per trail emitter sequence, based on the **Texture Mapping** type. Valid values: 0+ Default value: 1

Parameter Function	Description
Lock Anchor Points	Locks the UV anchor points of the texture to stay in place, rather than follow the emitter location. Default value: false
Texture Mapping	Maps the assigned texture to each particle quad or across the trail stream. • **Per Particle** – Texture is mapped on each quad. • **Per Stream** – Texture is mapped over the length of the beam. Default value: **Per Particle**

Note
When there is a single light source, some or all particles for **Trail** emitters can appear unlit (black). This is most apparent when the sun is the only light source and the time and day setting is noon. To address this:

1. Add an Environment Probe to the scene to create indirect lighting. For more information, see Environment Probe (p. 499).
2. Enable Environment Probe Lighting on the particle system by specifying a value for the Environment Probe Lighting parameter. For example, a value of `0.5` applies light from the Environment Probe at half intensity. For more information, see Lighting Attribute (p. 1295).

Particle Trail Visibility

Trail segments are drawn when the distance between the start and end of a segment exceed the value for **Min visible distance**. You can choose to automatically disable the drawing of trails that are not moving or are moving too slowly. This is useful if you have particle trail effects that need to be drawn only when the emitter is moving.

The following are movement parameters for the **Trail** emitter.

Related Movement Attributes

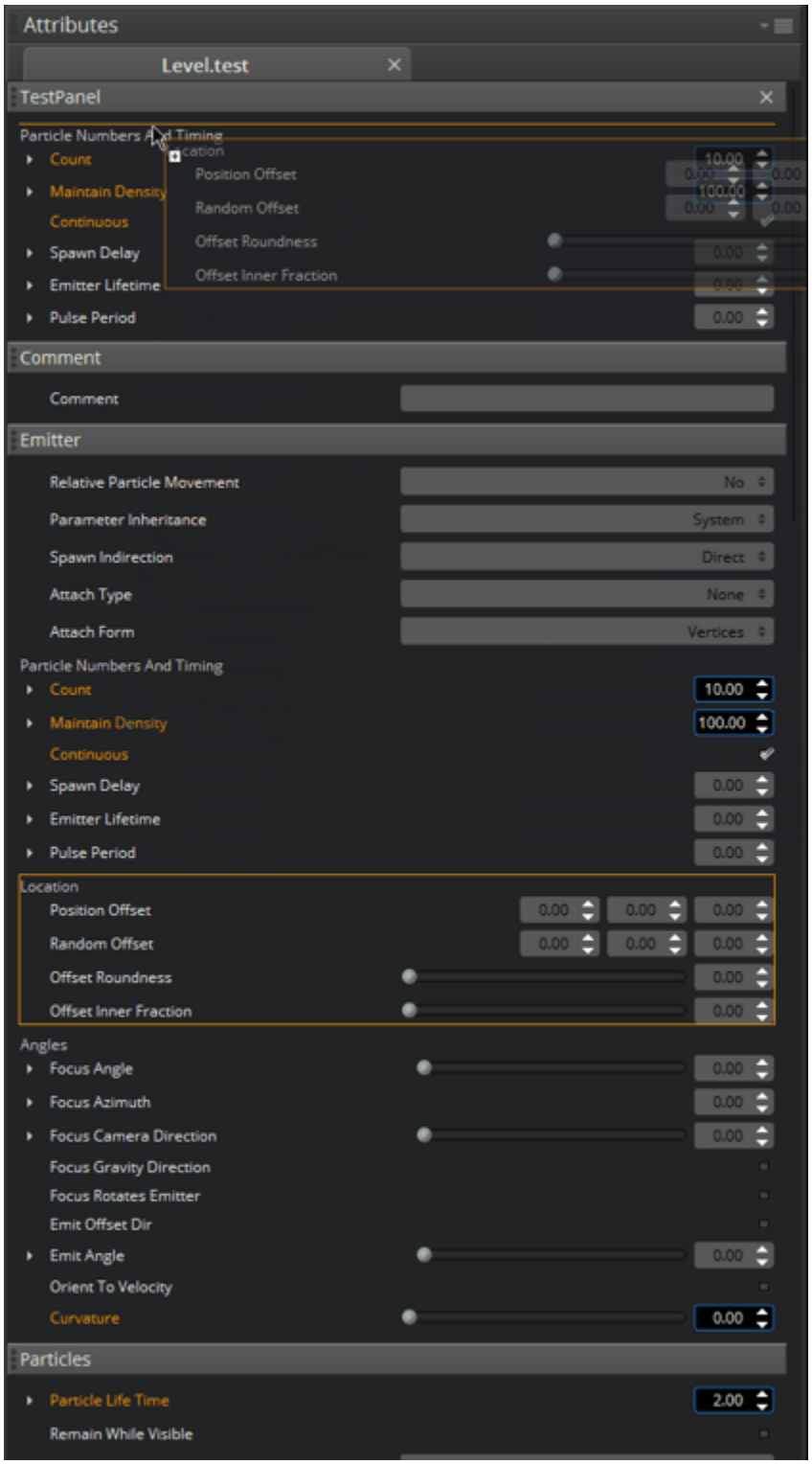
Parameter	Description
Min visible segment length	(Trail Emitter (CPU) (p. 1326) only) Tail particles are visible only when they have moved the specified distance. Default value: false
Min visible distance	(Trail Emitter (CPU) (p. 1326) only) Specifies the minimum distance between the start and end of a trail segment. Segments that are smaller than this value become transparent. Valid values: 0+

Parameter	Description
	Default value: 0

Creating Custom Attribute Panels

Create a custom particle attribute panel so you can quickly access the parameters that you frequently use. You can drag parameters from an existing standard panel to copy the parameters into the custom panel. Or you can drag parameters from a custom panel to move the parameters to the new custom panel. If you want to simultaneously add multiple parameters, press **Ctrl** while selecting the parameters.

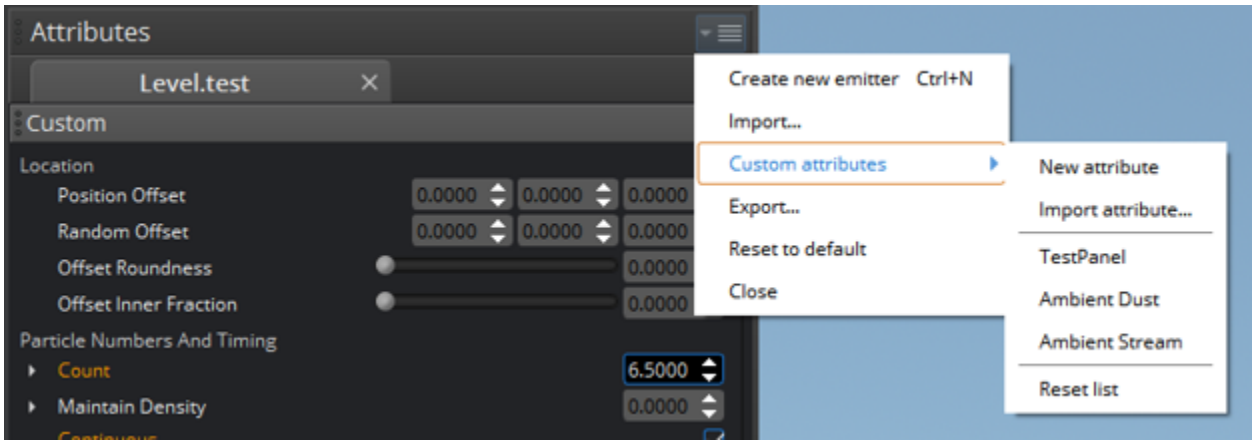
When you move a panel to a new location, an indicator appears so you can preview a valid drop location. If you do not see an indicator, the parameter is inserted at the end of the panel. The following example shows a custom panel with drop indicators.



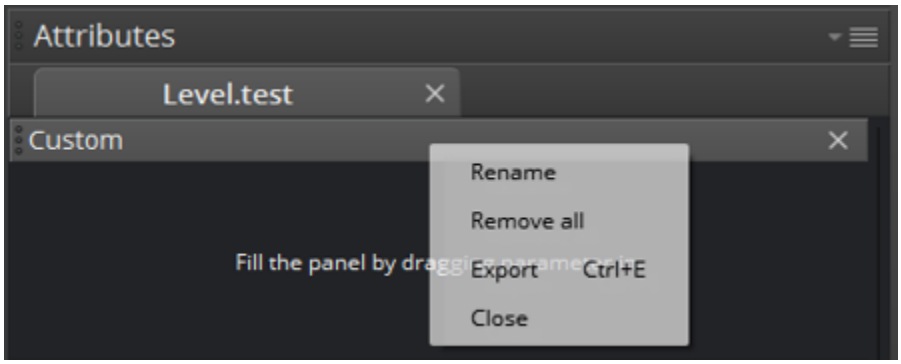
To create a custom attribute panel

1. In Lumberyard Editor, choose **Tools**, **Particle Editor**.
2. In the **Particle Editor**, click the menu in the **Attributes** title bar.

3. Do one of the following:
   - Choose **Custom attributes**, **New attribute**.
   - Choose **Custom attributes**, **Import attribute**.



4. Name your custom attribute or browse to the attribute that you want to import.
5. Once created, you can right-click the attribute title bar to rename, empty, or export custom attribute panels.



Custom Attributes Panel Menu Options

Use the following menu options to modify your custom attribute panel.

Option	Description
New attribute	Adds an empty, custom panel to the **Attributes** panel.
Import attribute	Loads an existing custom attribute panel. The panel is also added to the panel preset list.
Panel preset list	Lists the custom panel presets.
Reset list	Resets the preset list.
Rename	Renames the custom panel.
Remove all	Removes all attributes in the panel.

Option	Description
Export	Exports the selected custom panel as a `.custom_attribute` file. The panel is also added to the panel preset list.
Close	Closes the custom panel.

Using the Color Picker

Use the color picker to apply a color tint and alpha (opacity) to your particles. You can save your palette libraries to share with others.

To use the Color Picker

1. In Lumberyard Editor, choose **Tools**, **Particle Editor**.
2. In the **Libraries** panel, select an emitter.
3. In the **Attributes** panel, under **Particles**, click the color swatch for the **Color** parameter.

The **Color Picker** includes the following:

A. **Eye dropper** – Selects a color on your screen with magnification.

B. **Current** and **New** color – Displays your current color selection and the new color selection.

C. **HTML and RGB values** – Specifies the color value inputs.

D. Color window – View the range of colors to select.

E. **Channels** – Focuses the hue range based on RGB and CMY color sets.

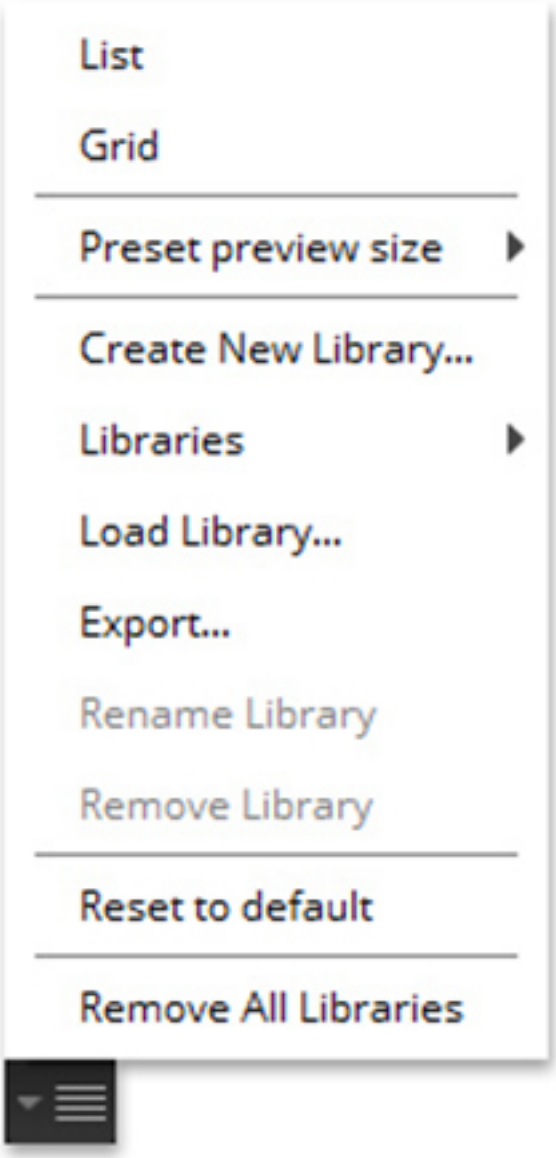
F. Hue, Saturation, Value, and Alpha – Slides to select a new color.

G. **Default Library** palette – A list or grid of the currently selected library palette.

H. Help icon – Displays the online help topic for the **Color Picker**.

Using the Palette Libraries

Create a palette library to save your collection of custom colors. You can create multiple palette libraries.



Use the following actions in the palette library menu.

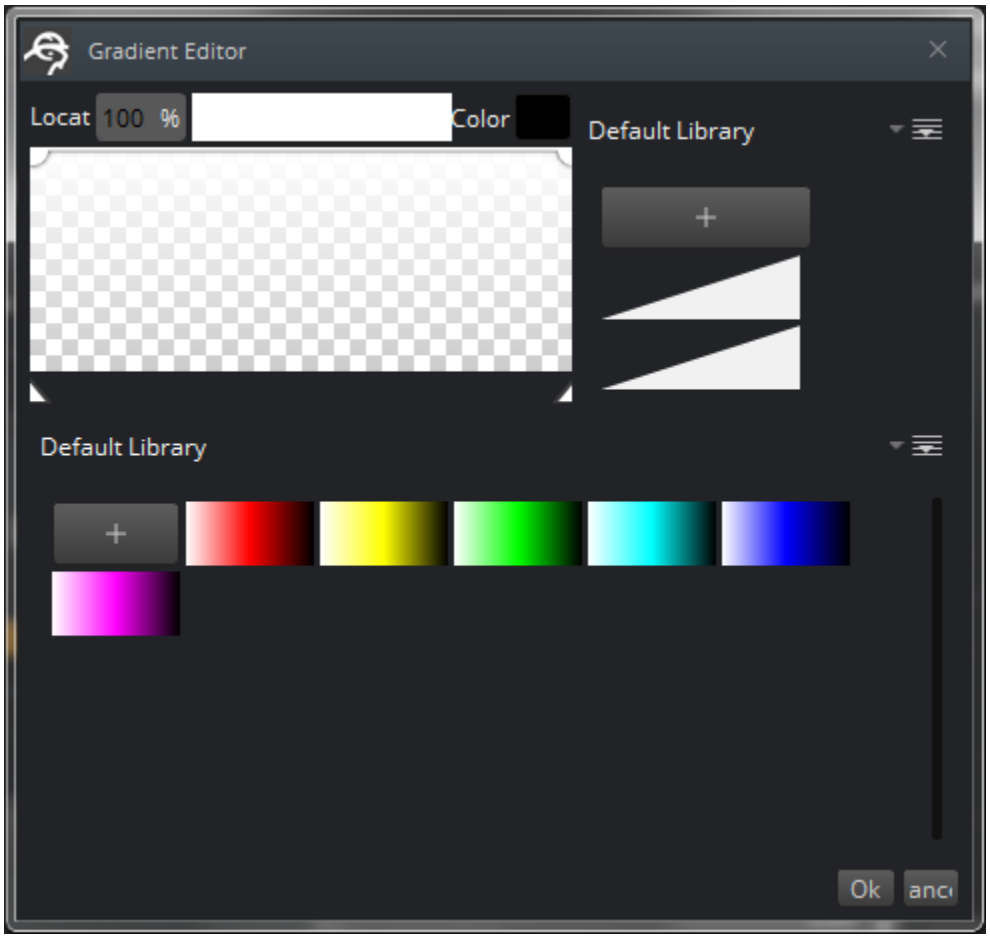
Option	Description
List	Displays a list view with name or RGBA values.
Grid	Displays the palette as a grid of color swatches.
Preset preview size	Changes the palette swatch size.
Create New Library	Creates a blank palette library.
Libraries	Displays a list of imported palette libraries.
Load Library	Imports a saved palette library.
Export	Exports the current palette library to a file.
Rename Library	Renames the current palette library.
Remove Library	Removes the current palette library.
Reset to default	Resets the current library to the default palette.
Remove All Libraries	Removes all palette libraries from the libraries list. This does not delete exported library palettes.

Using the Gradient Editor

Use the **Gradient Editor** to apply color ranges to an emitter and configure additional subparameters.

To use the Gradient Editor

1. In Lumberyard Editor, choose **Tools**, **Particle Editor**.
2. In the **Libraries** panel, select an emitter.
3. In the **Attributes** panel, under **Particles**, expand **Color**.
4. Click the color swatch for the **Strength Over Emitter Lifetime** or **Strength Over Particle Lifetime** parameter.



Gradient Editor UI Options

The **Gradient Editor** includes the following:

- **Location** – Sets the location value range from 0% – 100%.
- **Color** – Opens the **Color Picker**.
- **Gradient box** – Applies the combined gradient and alpha.
- Gradient viewport
  - The x-axis represents the gradient generator of the color change over the full gradient.
  - The y-axis represents 0 – 100% alpha of the gradient color.
- **Default Library**, alpha curve – Provides alpha curves to use as a starting point.
- **Default Library**, gradient – Provides gradients to use as a starting point.

Working with Color Gradients

When you select a gradient from the **Default Library**, it displays with the alpha curve in the **Gradient Editor** viewport. You can do the following when selecting a gradient:

To change the gradient color

- Click the triangle keyframes and select a new color from the **Color Picker**.

To add a color to the gradient

- Double-click the X-axis to generate a color keyframe. The color in the color thumbnail is added to the gradient viewport. Any adjustments you make to the gradient is reflected in the gradient viewport.

To display the RGBA values

- Pause on the color keyframe.

To delete a color keyframe

- Select the keyframe and press **Delete**. The selected keyframe has an orange outline.

To adjust the alpha curve

- Click and drag the circle (alpha keyframe) in the gradient viewport. Move the circle up (towards 100%) or down (towards 0%) to adjust the alpha percentage. Moving the circle left or right adjusts the curve based on the curve endpoints.

To display the alpha curve context menu

- Right-click the alpha curve keyframe. The following options are available: **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**, and **Reset the curve to defaults**.

To add an alpha key

- Double-click the curve in the gradient viewport.

To delete an alpha key

- Select the circle key and press **Delete**.

To add a generated alpha curve to the preset list

- Click the **+** button.

To add a generated gradient to the gradient preset list

- Click the **+** button.

To delete a curve or gradient preset

- Right-click the gradient or curve and click **Remove**.

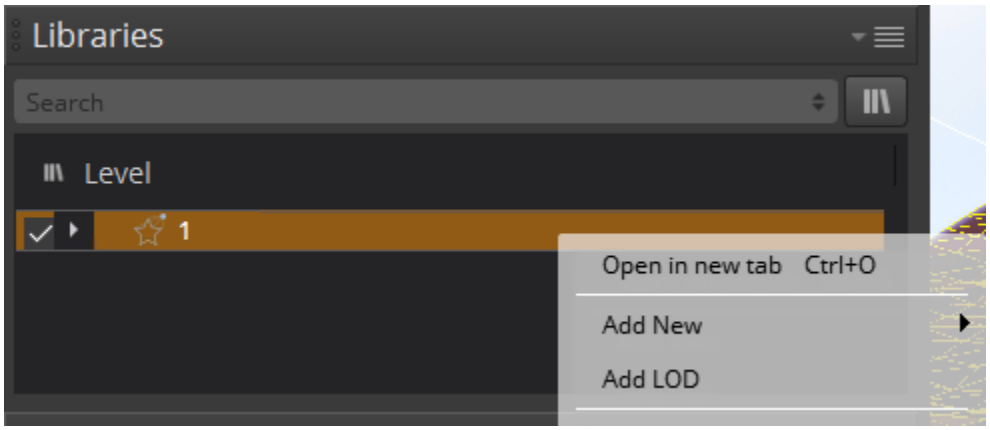
Managing Particle Level of Detail (LOD)

The Level of Detail (LOD) system blends multiple particle emitters based on their distance from the camera. This allows you to use emitters that require less computation and rendering time, rather than computationally heavy particle emitters.

To add an LOD for a particle emitter

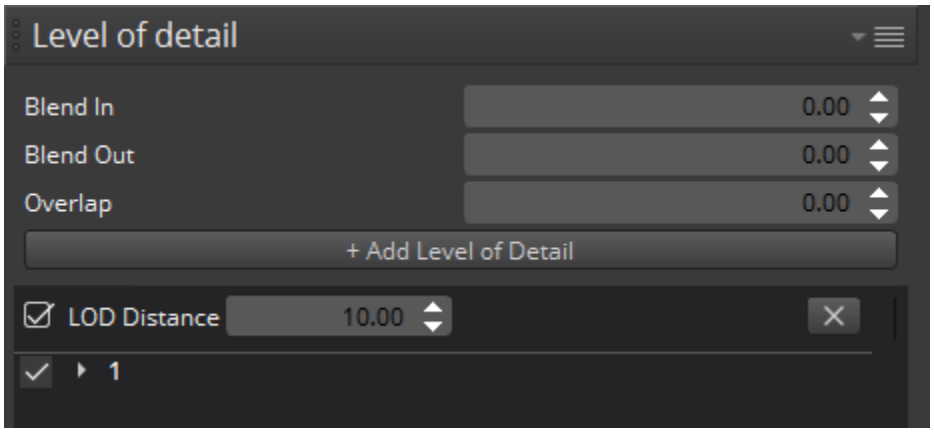
1. In Lumberyard Editor, choose **Tools**, **Particle Editor**.
2. In the **Particle Editor**, in the **Libraries** panel, right-click an emitter.
3. Select **Add LOD**.

   The LOD is a copy of the base particle emitter and has the same settings. The LOD also applies to all parent or child particle emitters in the hierarchy that belong to the selected emitter.



Level of Detail Panel

The **Level of Detail** panel appears when you add an LOD. This panel shows the level of detail that you selected from the **View** menu in the **Particle Editor**.

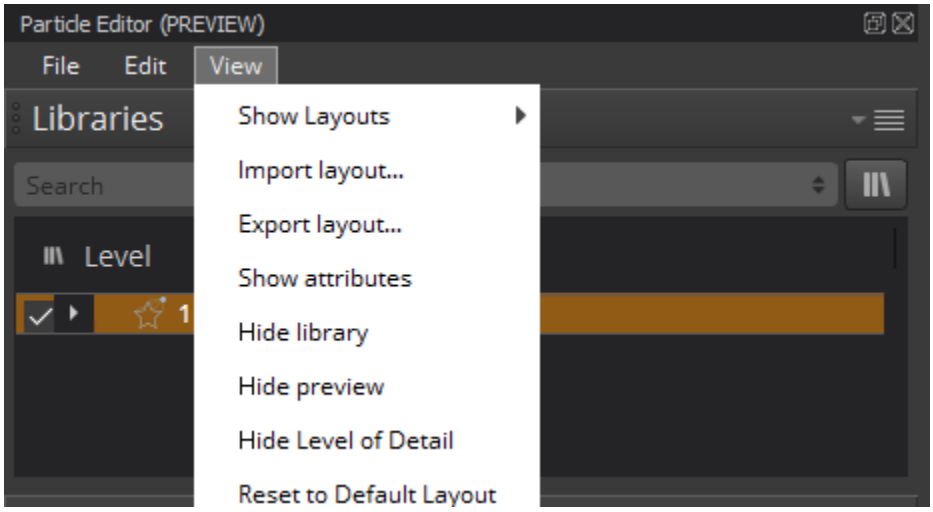


Option	Description
Blend In	The amount of time, in seconds, for an LOD to blend in.
Blend Out	The amount of time, in seconds, for an LOD to blend out.

Option	Description
Overlap	The amount of time, in seconds, that both LODs are shown before the old LOD blends out and the new LOD blends in.
+ Add Level of Detail	Adds an LOD. The new LOD distance is set to 10 additional units from the farthest LOD. The list of added LOD levels appears under **+ Add Level of Detail**.

Hide or Display Options

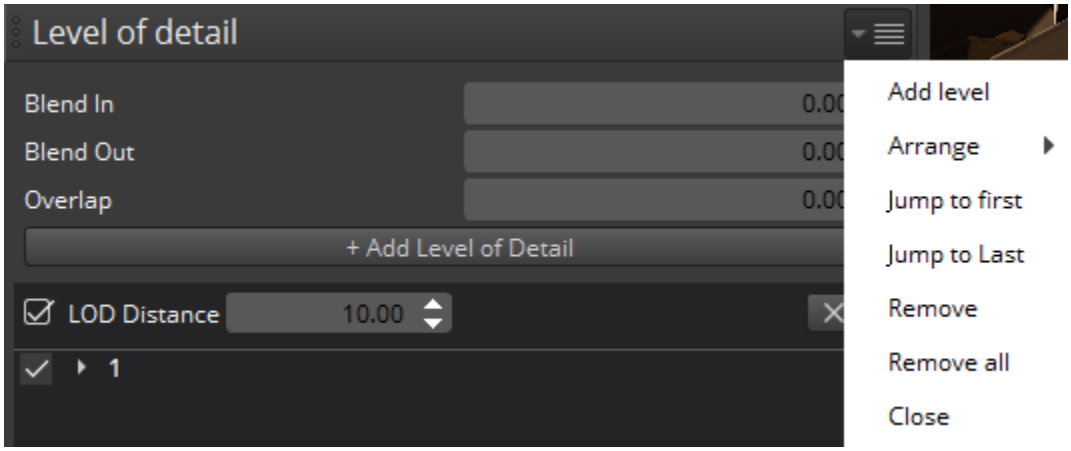
Hide or display the **Level of Detail** panel by clicking **View**, **Hide Level of Detail** in the **Particle Editor**.



Option	Description
Hide Level of Detail	Hides the LOD panel if the panel is visible.
Show Level of Detail	Displays the LOD panel if the panel is hidden.

Manage LOD Options

Manage your LOD levels in the list by clicking the drop-down menu in the **Level of Detail** title bar.

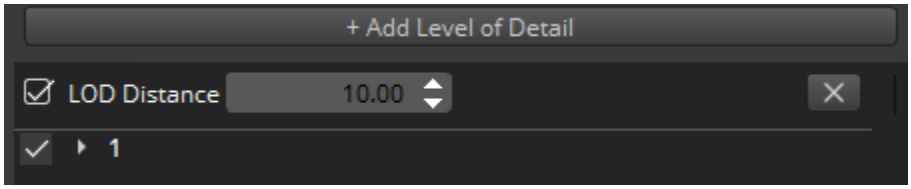


Option	Description
Add level	Adds an LOD level to the end of the list.
Arrange	Shows the **Arrange** submenu: • **Move up** – Moves the selected LOD level one position up in the list. This also changes the **Level LOD Distance** to 1.0 lower than the previous level. • **Move down** – Moves the selected LOD level one position down in the list. This also changes the **Level LOD Distance** to 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 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 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 the list and loads it in the **Attributes** panel.
Jump to last	Selects the last LOD level in the list. This also selects the top particle emitter in the list and loads it in the **Attributes** panel.
Remove	Removes the selected LOD level.
Remove All	Removes all LOD levels for the 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**
> The base particle emitter is shown if the camera distance is lower than the top LOD level. This makes the base particle the starting LOD level.



Option	Description
Top left check box	Turns on/off the entire level.
LOD distance value	Specifies the camera distance at which the LOD level becomes active. At this level, the particle emitter blends toward the LOD level and blends out the previous LOD level.
Top right button	Deletes the corresponding LOD level.

Option	Description
Particle check box	Turns off the particle emitter at this level. When the emitter is off, nothing is drawn. You can use this to turn off particle emitters based on the LOD level.
Particle name	Click a particle name to load the LOD level particle emitter for the selected particle emitter in the **Attributes** panel. You can use this to change the LOD level particle emitter. Right-click a particle name and click **Remove** to remove the particle from the LOD level. Any child particle emitters are also removed from the LOD level.

Using Particle Editor Keyboard Shortcuts

Most of the commands in the **Particle Editor** menus have corresponding keyboard shortcuts. You can edit the keyboard shortcuts by clicking **Edit**, **Edit Hotkeys**. Modify as needed in the **HotKey Configuration** window.

Hotkey Configuration

Export

Exports the keyboard shortcuts (hotkey) list to a file.

Import

Imports a keyboard shortcuts (hotkey) list from a file.

Reset to Default

Resets all keyboard shortcuts (hotkeys) to the editor default settings.

Click to assign

Records a new keyboard shortcut when you click the shortcut. Clears the keyboard shortcut when you right-click the shortcut. Click **OK** to save your changes.

The **Particle Editor** uses the following keyboard shortcuts.

HotKey Configuration

Edit Menu	
Assign to selected	Ctrl+Space
Copy	Ctrl+C
Delete	Del
Duplicate	Ctrl+D
Edit Hotkeys	Click to assign
Enable/Disable Emitter	Ctrl+E
Group	Ctrl+G
Insert Comment	Ctrl+Alt+M
Open in new tab	Ctrl+O
Paste	Ctrl+V
Redo	Ctrl+Shift+Z
Rename	Ctrl+R
Reset	Click to assign
Undo	Ctrl+Z
Ungroup	Ctrl+Shift+G
File Menu	
Close	Ctrl+Q
Create new emitter	Ctrl+N
Create new folder	Click to assign
Create new library	Ctrl+Shift+N
Disable All	Click to assign
Enable All	Click to assign
Import	Ctrl+I
Save	Ctrl+S
Menus	
Edit Menu	Alt+E
File Menu	Alt+F
View Menu	Alt+V
Previewer	
Focus	Ctrl+F
Loop Toggle	Z
Pan Left	A
Pan Right	D
Play/Pause Toggle	Space
Reset Playback	X
Step forward through time	C
Zoom In	W
Zoom Out	S
View Menu	
Reset Layout	Click to assign

Actions Ok Cancel

Particle Effects Best Practices

The total number of particles in a scene is 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 subemitters that are near the ground and have only small particles. Create similar subemitters higher up that emit particles that never intersect with the ground and don't require soft particles.
- If sharp details are not required, use low-resolution textures and texture compression.
- Use an alpha texture with high or average opacity rather than additive blending.
- Each second-generation effect causes an emitter to be created for each particle in the parent effect. Use this sparingly because it can be expensive.
- 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. The rest go through the ground or fade out quickly.
- Instead of multiple overlaid sprites for chaotic glow effects, use only two particles at a time. Carefully tune the lifetime and rotation rate, and set curves for **Alpha**, **Color**, and **Size** to combine in chaotic ways. Or, 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 that they aren't rendered as small, single pixel particles. Use the lowest **Config spec** setting to turn off small particles that are used in collisions.

Advanced Particle Techniques

Use the following advanced techniques in the Particle Editor to emit particles from geometry, create a particle effect that explodes, preview your effects on an animated character, or generate particles from surface properties.

Emitting Particles from Geometry

Do the following to emit particles from the parent geometry.

To emit particles from geometry

1. Create a parent emitter with a `.cgf` mesh asset that is assigned in the **Geometry** input under **Particles**. For more information, see Creating and Managing Particle Emitters (p. 1311).
2. Create a new emitter as a child of the first emitter.
3. On the child emitter, in the emitter attributes, set **Spawn Indirection** to **Parent Start**.
4. Set the **AttachType** to **Render**, and the **AttachForm** to **Vertices**.

   The child's particles spawn from the mesh of its parent.

Note
Use any **Attach Type** and **AttachForm** for the intended effect.

Creating Exploding Geometry Particle Effect

Do the following to create a particle effect that instantly spawns exploding chunks.

To create an exploding geometry particle effect

1. Create an emitter. In the **Attributes** panel, under **Particles**, assign a multipart `.cgf` file to the **Geometry** input.
2. Set **Geometry** in **Pieces** to the option for your use case: **RandomPieces** or **AllPieces**.
3. Set appropriate values for **Gravity**, **Speed**, **Rotation Rate**, and more to create an exploding effect.
4. Optionally set **Collision** parameters for physicalized pieces.

Attaching Particles to Character Animations for Previewing

Do the following to preview the effects on an animated character in your level.

To attach a particle to an animation

1. In Lumberyard Editor, choose **Tools**, **Asset Browser**.
2. In the **Asset Browser**, select and drag a `.cdf` character asset that has animations into the Lumberyard Editor viewport. Close the **Asset Browser**.
3. In the **Entity Inspector**, click **Add Component**. Under **Animation**, choose **Simple Animation**.
4. Under **Simple Animation**, click the **+** button to add an element. Select an animation to play.
5. Open the **Particle Editor** and drag the effect that you want to preview into your level.
6. In the **Entity Inspector**, click **Add Component**. Under **Animation**, choose **Attachment**.
7. Under **Attachment**, click the target entity picker and select your character in the viewport.
8. In the joint list, select a joint name to which to attach the effect. Select an offset as needed.
9. In the bottom toolbar, click **AI/Physics**. Alternatively, press **Ctrl+G** to start the animation and preview the effect.

Generating Particles from Surface Properties

Use the properties for an object's material surface to define the event-driven effects that can occur when an object experiences an event. You can specify these events on a render material or on individual pieces or surfaces of a `.cgf` asset.

Many of the properties define the particle effects that are spawned by events such as walking or a bullet hit. To specify the effect that is spawned when a geometry piece breaks off of an object, set the following parameters in a Lua script:

Name

Specifies the name of the particle effect.

Scale

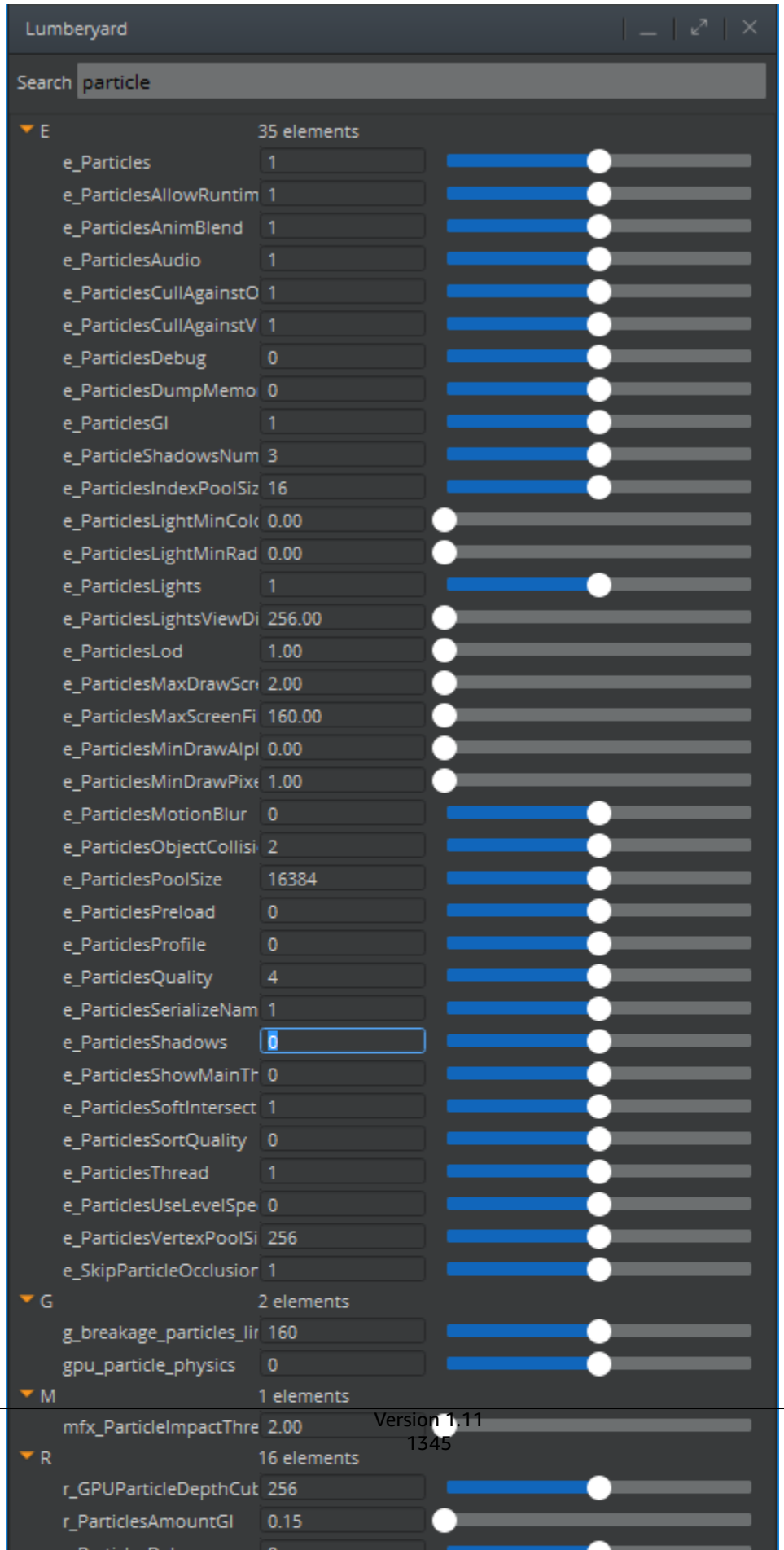
Multiplies the size of the particle.

Count_scale

Multiplies the particle counts.

Particle Debugging

Use the following console variables to monitor and debug particle system issues. To access the console variables, click the **x** button in the **Console** panel in Lumberyard Editor. For more information, see Using the Console Window (p. 93).


Lumberyard
Search particle
E 35 elements
e_Particles 1
e_ParticlesAllowRuntim 1
e_ParticlesAnimBlend 1
e_ParticlesAudio 1
e_ParticlesCullAgainstO 1
e_ParticlesCullAgainstV 1
e_ParticlesDebug 0
e_ParticlesDumpMemo 0
e_ParticlesGI 1
e_ParticleShadowsNum 3
e_ParticlesIndexPoolSiz 16
e_ParticlesLightMinColc 0.00
e_ParticlesLightMinRad 0.00
e_ParticlesLights 1
e_ParticlesLightsViewDi 256.00
e_ParticlesLod 1.00
e_ParticlesMaxDrawScr 2.00
e_ParticlesMaxScreenFi 160.00
e_ParticlesMinDrawAlpl 0.00
e_ParticlesMinDrawPixe 1.00
e_ParticlesMotionBlur 0
e_ParticlesObjectCollisi 2
e_ParticlesPoolSize 16384
e_ParticlesPreload 0
e_ParticlesProfile 0
e_ParticlesQuality 4
e_ParticlesSerializeNam 1
e_ParticlesShadows 0
e_ParticlesShowMainTh 0
e_ParticlesSoftIntersect 1
e_ParticlesSortQuality 0
e_ParticlesThread 1
e_ParticlesUseLevelSpe 0
e_ParticlesVertexPoolSi 256
e_SkipParticleOcclusior 1
G 2 elements
g_breakage_particles_lir 160
gpu_particle_physics 0
M 1 elements
mfx_ParticleImpactThre 2.00
R 16 elements
r_GPUParticleDepthCut 256
r_ParticlesAmountGI 0.15

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. 664).

For information on character physics, see Physicalizing Characters (Ragdoll) (p. 361).

For information on character attachment physics (simulations), see Secondary Animations (Simulations) (p. 298).

For information on physics flow graph nodes, see Physics Nodes (p. 969).

Topics

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.

You can use the following console variables to debug 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.

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.

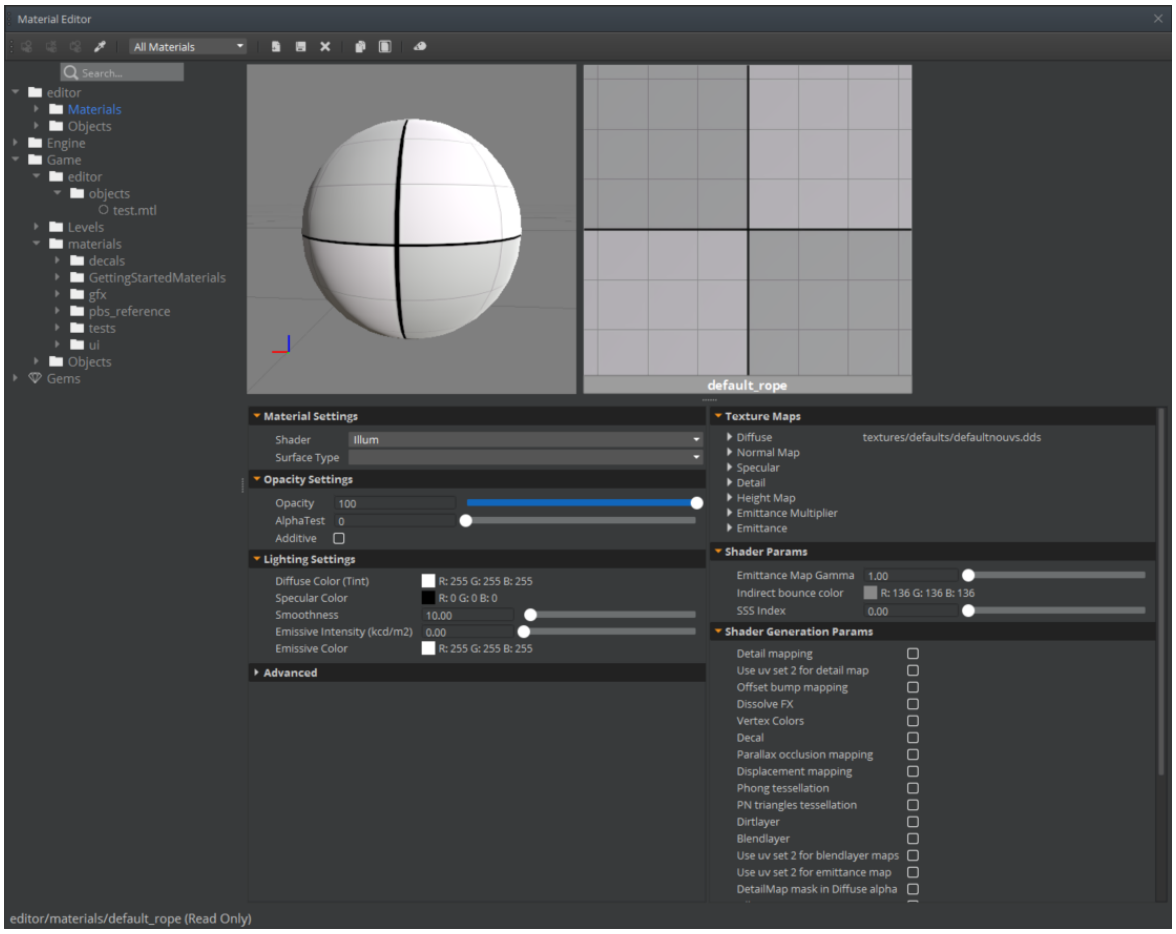
Topics

Materials and Shaders

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. 1351).

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 be 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

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. 1359).

There are two categories of materials that are relevant for shader rendering: metals such as like 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 **Tools**, **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

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. 1429)

Cube maps use image-based lighting. For more information see Image-Based Lighting (p. 1352).

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

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.

> **Important**
> Ensure the server or computer that is running the remote shader compiler is in a controlled environment that restricts incoming network requests to only authorized and trusted users or devices. Do not run the remote shader compiler with escalated root, admin, or super-user privileges.

Topics

- Running the Remote Shader Compiler (p. 1356)

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 containi 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 ith 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.

Creating a Whitelist for the Remote Shader Compiler

You can use a whitelist to specify the IP addresses that are allowed to connect to your remote shader compiler. If a computer has an IP address that is not in the whitelist, the remote shader compiler will provide a message that an invalid computer tried to connect and then close the connection. This prevents data from being read or sent on the invalid connection.

To create a whitelist for the remote shader compiler, create a text file called `config.ini` and save it in the same directory as the remote shader compiler executable. Add the following parameter to the file:

- `whitelist` – Provide a comma separated list of IP addresses in CIDR format. The remote shader compiler uses this list to validate incoming connection requests. The remote shader compiler automatically adds the loopback IP address (127.0.0.1) and its own IP address.

The following example allows computers or devices with an IP address of 192.168.0.1 to connect to the remote shader compiler.

```
whitelist=192.168.0.1
```

The following example allows computers or devices with an IP address of 192.168.0.* to connect to the remote shader compiler. The /24 specifies a net mask of 24-bits. If you specify /8, any address that starts with 192 is allowed given only an 8-bit net mask.

```
whitelist=192.168.0.1/24
```

Generating Shader Combinations

Make sure that the Remote Shader Compiler (p. 1355) 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 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 **Tools**, **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. 1361)	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. 1362)	Use to render cheap 2D clouds that are distantly placed in a sky scene.
Eye Shader (p. 1364)	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. 1370).
GeometryBeam Shader (p. 1365)	Use to create volumetric light beams that feature dust and turbulence effects.
Glass Shader (p. 1367)	Use to render glass surfaces with various refractive, reflective, ripple, tint, and cracking effects.
Hair Shader (p. 1370)	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. 1364) for realistic eyes.
HumanSkin Shader (p. 1372)	Use to render skin and it's various physical properties including color, oiliness, pores, stubble, and wrinkles.
Illum Shader (p. 1374)	The most common shader - use to create an extremely wide variety of render effects.
LightBeam Shader (p. 1378)	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. 1380)	Use to create particle effects that are not affected by light and hence do not cast shadows or cause reflections.
Particles Shader (p. 1380)	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. 1383)	Use to render cheap static sky (SkyBox) effects.
SkyHDR Shader (p. 1384)	Use to render realistic dynamic sky effects that change based on time of day in the level.
TemplBeamProc Shader (p. 1384)	Use to create cheap fog-like effects for light beams.

Shader Name	Description
Terrain.Layer Shader (p. 1386)	Use for painting and blending terrain texture layers in a level.
Vegetation Shader (p. 1387)	Use to render trees, bushes, grass, and other vegetation, as well as imparting various bending motion effects.
VolumeObject Shader (p. 1389)	Use to render various volumetric objects such as clouds, fog, and smoke, and to impart realistic shading and self-shadowing effects.
Water Shader (p. 1390)	Use to render the ocean exclusively, and to impart various reflection, ripple, and foam effects.
Waterfall Shader (p. 1392)	Use to render waterfalls exclusively, and provides layering and tiling, as well as motion effects.
WaterVolume Shader (p. 1394)	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. 1374) 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 it's 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. You can use this shader to create a variety of effects.

Shader Parameters

Blend Factor

Controls the visibility of the blended layer.

To use this parameter, you must enable the **Blendlayer** shader generation parameter.

Default value: 8

Blend Falloff

Controls blending falloff.

To use this parameter, you must enable the **Blendlayer** shader generation parameter.

Default value: 32

Blend Layer 2 Tiling

Controls tiling of the second blend layer.

To use this parameter, you must enable the **Blendlayer** shader generation parameter.

Default value: 1

Blend Layer 2-Diffuse(Tint)

Controls the diffuse (tint) of the second blend layer.

To use this parameter, you must enable the **Blendlayer** shader generation parameter.

Blend Layer 2-Smoothness

Controls the smoothness of the second blend layer.

To use this parameter, you must enable the **Blendlayer** shader generation parameter.

Default value: 10

Blend Layer 2-Specular

Controls specular intensity of the second blend layer.

To use this parameter, you must enable the **Blendlayer** shader generation parameter.

Default value: 132, 132, 132

Blend Mask Tiling

Controls tiling of the blend mask.

To use this parameter, you must enable the **Blendlayer** shader generation parameter.

Default value: 1

Detail bump scale

Sets detail bump scale.

To use this parameter, you must enable the **Detail mapping** shader generation parameter.

Default value: 0.5

Detail diffuse scale

Sets diffuse detail blend scale.

To use this parameter, you must enable the **Detail mapping** shader generation parameter.

Default value: 0.5

Detail gloss scale

Sets gloss detail blend scale.

To use this parameter, you must enable the **Detail mapping** shader generation parameter.

Default value: 0.5

Dissolve Color

Determines color of the edge if edge thickness is greater than 0

To use this parameter, you must enable the **Dissolve FX** shader generation parameter.

Default value: 255,255,255

Dissolve Edge Thickness

Determines the thickness of a border that is the **Dissolve Color** around the edge of the effect.

To use this parameter, you must enable the **Dissolve FX** shader generation parameter.

Default value: 0

Valid values: 0.000 to 1

Dissolve Noise Map

A grayscale map that defines the dissolve pattern. Black areas dissolve first; white areas dissolve last.

Appears under **Texture Maps** after you enable **Dissolve FX** shader generation parameter.

Dissolve Percentage

Controls the amount that the texture is dissolved.

To use this parameter, you must enable the **Dissolve FX** shader generation parameter.

Default value: 0

Valid values: 0.000 to 1

Height bias

Controls the height bias.

To use this parameter, you must enable the **Parallax occlusion mapping** shader generation parameter.

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 offset bump mapping (OBM).

To use this parameter, you must enable the **Offset bump mapping** shader generation parameter.

Default value: 0.01

POM Displacement

Controls the amount of displacement for parallax occlusion mapping (POM).

To use this parameter, you must enable the **Parallax occlusion mapping** shader generation parameter.

Default value: 0.01

Self shadow strength

Allows movable objects, such as interactive objects or game characters, to cast shadows on themselves and each other. Without self-shadowing, for example, if a character holds their right arm over the left, the right arm does not cast a shadow on the left arm.

To use this parameter, you must enable the **Parallax occlusion mapping** shader generation parameter.

Default value: 3

Roughness maximum footprint

Specifies the maximum allowed area of the projected pixel footprint at any point in the scene.

To use this parameter, you must enable the **Specular Antialiasing** shader generation parameter.

Valid value ranges: 0–10

SAA Roughness Boost

Specifies the degree to which the effect should be applied.

To use this parameter, you must enable the **Specular Antialiasing** shader generation parameter.

Valid value ranges: 0–10

SSS Index

Controls subsurface scattering profile (SSS) and the 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. This option requires a **Detail** map, which you can select under the **Texture Maps** heading.

Use UV set 2 for detail map

Enables a second UV channel, if available, for detail map.

Offset bump mapping

Enables offset bump mapping. This option requires a heightmap, which you can select for the **Normal Map** option under the **Texture Maps** heading.

Dissolve FX

Enables dissolve effect. Selecting this parameter enables the **Dissolve Noise Map** under **Texture Maps**, and three **Shader Parameters**: **Dissolve Color**, **Dissolve Edge Thickness**, and **Dissolve Percentage**.

These three parameters work in tandem to determine the dissolve effect.

For example, assume your **Dissolve Color** is `white`, **Dissolve Edge Thickness** is `0.1`, and **Dissolve Percentage** is `0.5`. In that case, any area on the surface of the **Dissolve Noise Map** that is less than `0.5` (the value set for **Dissolve Percentage**) is completely transparent. Areas from `0.5` to `0.6`, which is a difference of `0.1` (the value set for **Dissolve Edge Thickness**) are white (the value set for **Dissolve Color**). Areas from `0.6` to `1.0` are unchanged and appear with the original material.

Vertex Colors

Allows the use of fake ambient occlusion. Also adds more depth and contrast to the model.

You must add vertex colors to the geometry in the DCC tool.

Decal

Applies a decal appearance when enabled for a material. Decal planes are normally placed close to other geometry.

Use this parameter to prevent flickering and z-fighting (p. 1858) when faces are close to each other.

Parallax occlusion mapping (p. 1424)

Enables parallax occlusion mapping. This option requires a heightmap, which you can select for the **Normal Map** option under the **Texture Maps** heading.

Displacement mapping

Enables displacement mapping. This option requires a heightmap, which you can select for the **Normal Map** option under the **Texture Maps** heading.

Phong tessellation

Enables the rough approximation of smooth surface subdivision.

PN triangles tessellation

Enables the rough approximation of smooth surface subdivision.

Blendlayer

Enables the blending of the normal-mapped diffuse layer on top of the base material.

Use UV set 2 for blendlayer (p. 1420) maps

Enables a second UV channel, if available, for blend layer texture maps and blend mask.

Use UV set 2 for emittance map

Enables a second UV channel, if available, for emittance map.

DetailMap mask in Diffuse alpha

Enables diffuse map alpha for masking detail maps. With this option you can use the alpha channel in the RGBA texture map to mask the decal.

Silhouette POM (p. 1424)

Enables parallax occlusion mapping with silhouettes. This option requires a heightmap, which you can select for the **Normal Map** option under the **Texture Maps** heading.

Depth Fixup

Enables write depth for depth of field and postprocessing.

Specular Antialiasing

Enables specular antialiasing. Enable this feature for highly glossy highlights on curved surfaces, which can be challenging to render without specular shimmering and crawling. This antialiasing feature estimates the subpixel region of the scene over which the specular normal distribution function should be filtered. The computation of the region is based on the projected size of each pixel. The normal distribution function (NDF) used in the specular bidirectional reflection distribution function (BRDF) is then filtered over this region, minimizing shimmering, crawling, and other specular aliasing artifacts. This method has almost no impact to GPU performance, is temporally stable, and compatible with Lumberyard's deferred shading system. Normal maps and normal map filtering techniques are compatible with this feature.

Occlusion Map

Enables support for a single channel occlusion map. If provided, this map decreases the amount of influence that indirect light has on a surface. When calculating the occlusion for a given pixel, if the occlusion map occludes more than the dynamically calculated occlusion, then that occlusion map is used instead. Otherwise it is ignored.

LightBeam Shader

The LightBeam shader creates various fog-like volumetric and atmospheric effects for light beams. The shader can only be applied to light entities. To use the LightBeam shader, create a Projector Light component and assign the shader to the material slot.

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.

TemplBeamProc Shader

The TemplBeamProc 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. 1389) 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. 1390) 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 tilling.

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 tilling 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 **Tools**, **Material Editor**.
2. In the left pane, click to select the desired asset.
3. Under **Material Settings**, make a selection for **Surface Type**.

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 **Tools**, **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

You can use the Material Editor to specify color, specular reflection, and lighting effects such as specularity, glossiness, and glow.

Specify the diffuse, specular, or emissive color values in RGB format, or select the values using the **Colors** dialog box. Click the color square next to the parameter to open the **Colors** dialog box. You can then use the color picker to select standard or custom colors for the hue, saturation, and luminance values.

To set lighting and color settings for a material

1. In Lumberyard Editor, click **Tools**, **Material Editor**.
2. In the left pane, select the asset that you want to modify.
3. Under **Lighting Settings**, set 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. Gray-scale values affect reflective brightness levels. To tint the specular color, use the color picker to select the desired tint.
Smoothness	The acuity or sharpness of a specular reflection. For values of 10 or less, there is a scattered reflection. For values greater than 10, there is a sharp reflection. You cannot have glossiness without specular color (reflection), as glossiness determines the sharpness of the reflection.
Emissive Color	Enable objects to emit light and be visible in the dark. Use this parameter to add brightness to objects. Unlike glow, this parameter does not emit light onto other objects. It does not work with deferred shading.
Emissive Intensity	Enable objects to glow, and simulate light that emits from extremely bright surfaces. Use this parameter in dark scenes for computer monitors, lamps, fire, neon lights, and similar objects. Unlike emissive color, this parameter emits light onto other objects. You can use a diffuse texture RGB channel to specify glow color, and a diffuse texture alpha channel to specify glow mapping. This allows you to mask the pixels where you want less (or no) glow. You can use glow with the Cloth, HumanSkin, and Illum shaders. To enable or disable glow, use the `r_Glow` console variable.

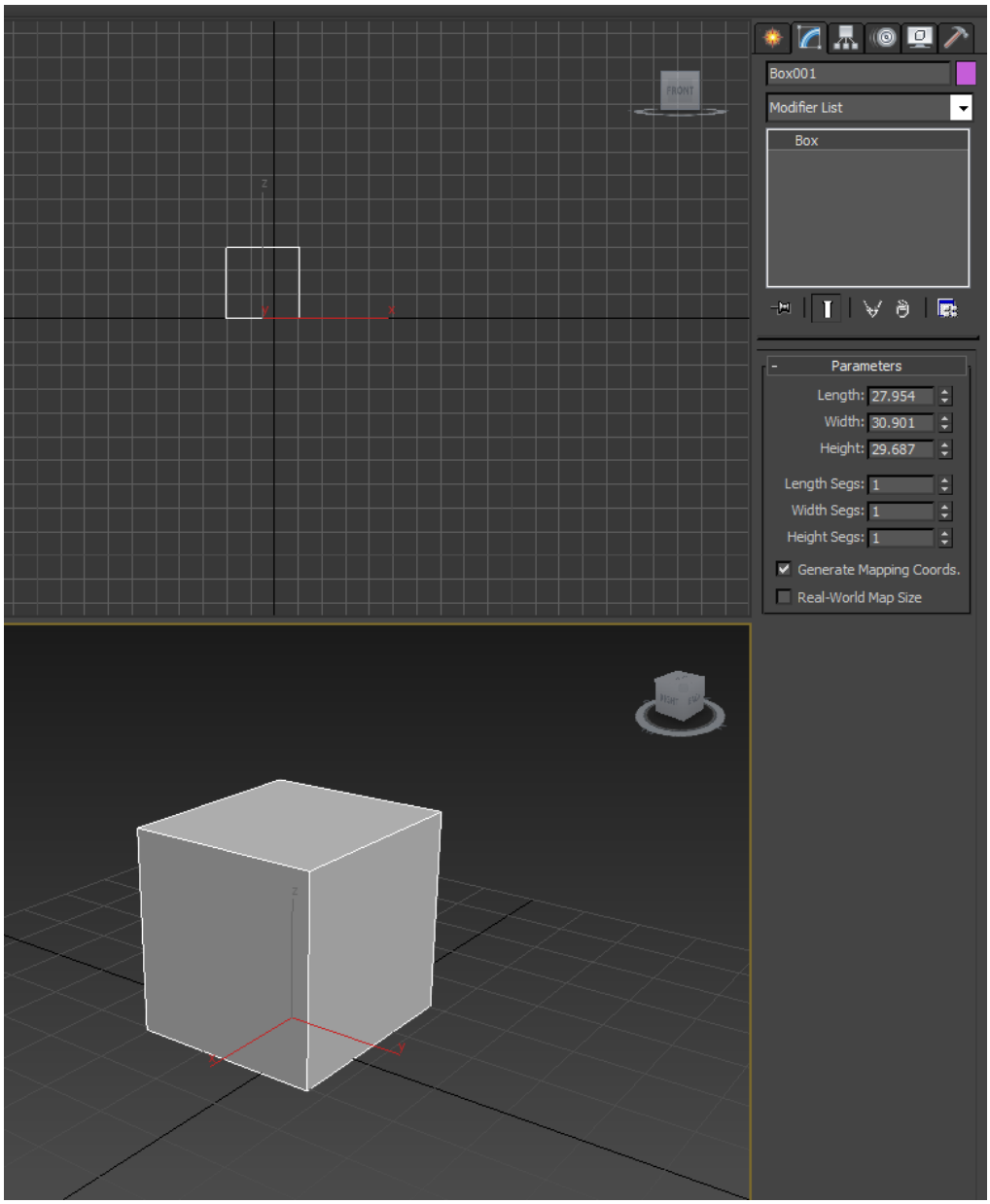
Material ID Mapping in Autodesk 3ds Max

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.

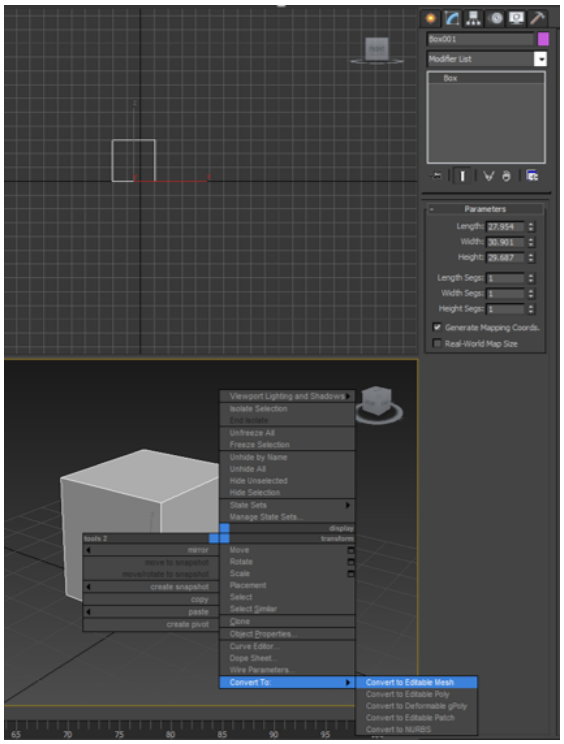
The following procedure presents an example that uses a multimaterial cube.

To map multi-material IDs in 3ds Max

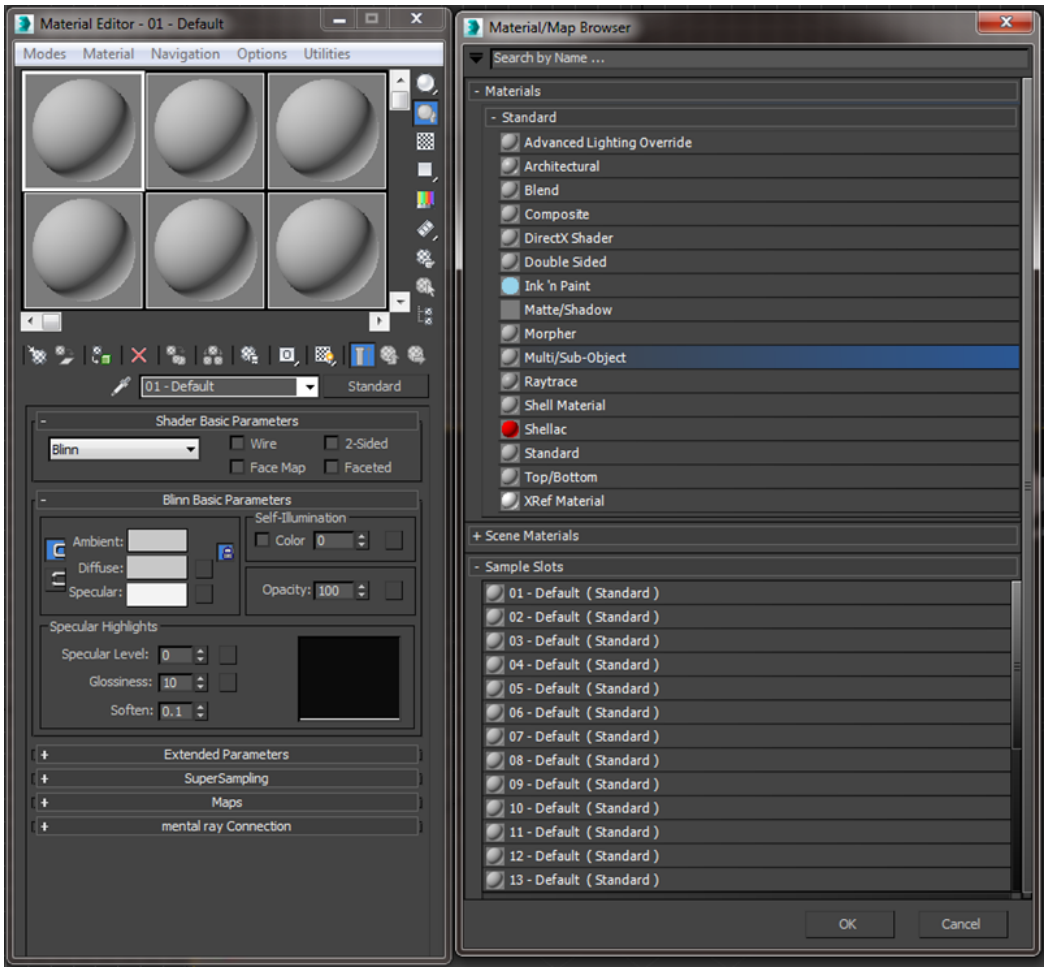
1. Open 3ds Max. Then create and place a cube in the viewport.



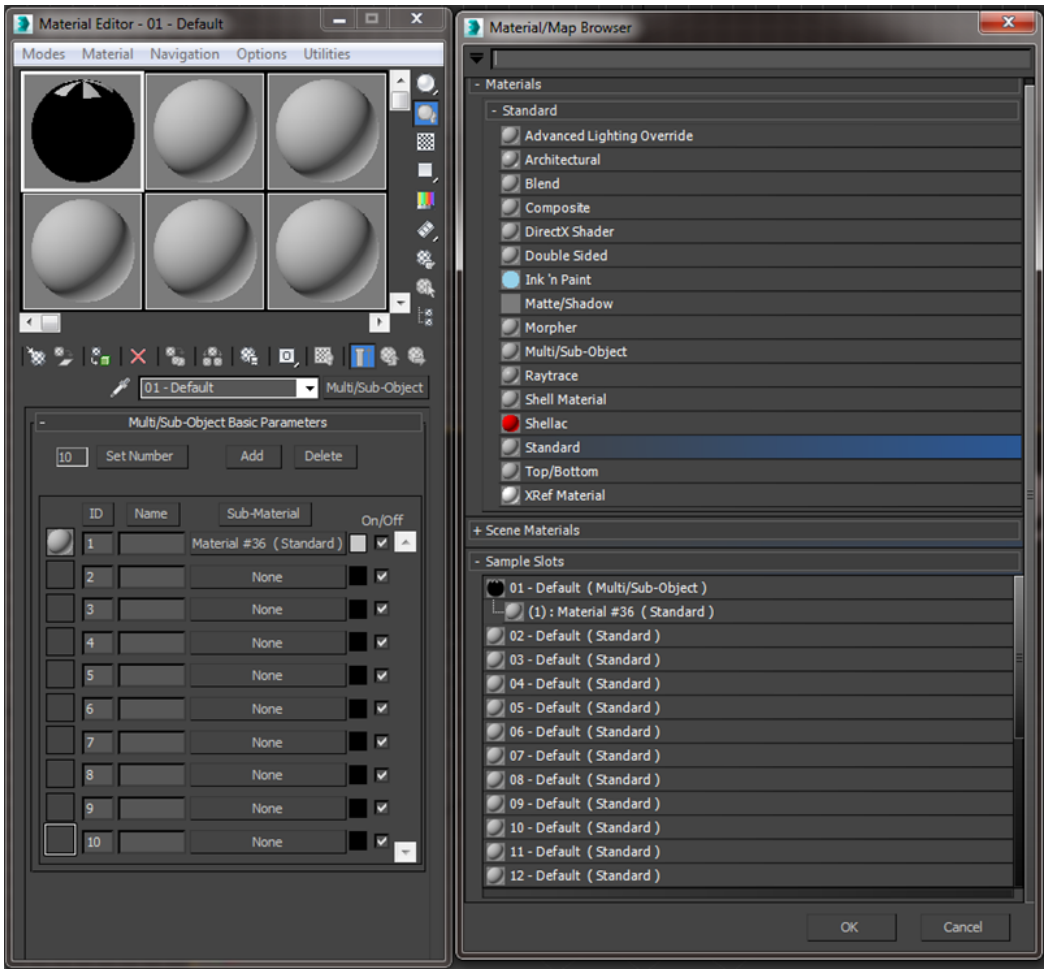
2. Right-click the cube and click **Convert To**, **Convert to Editable Mesh**. You can now assign different material IDs to the faces.



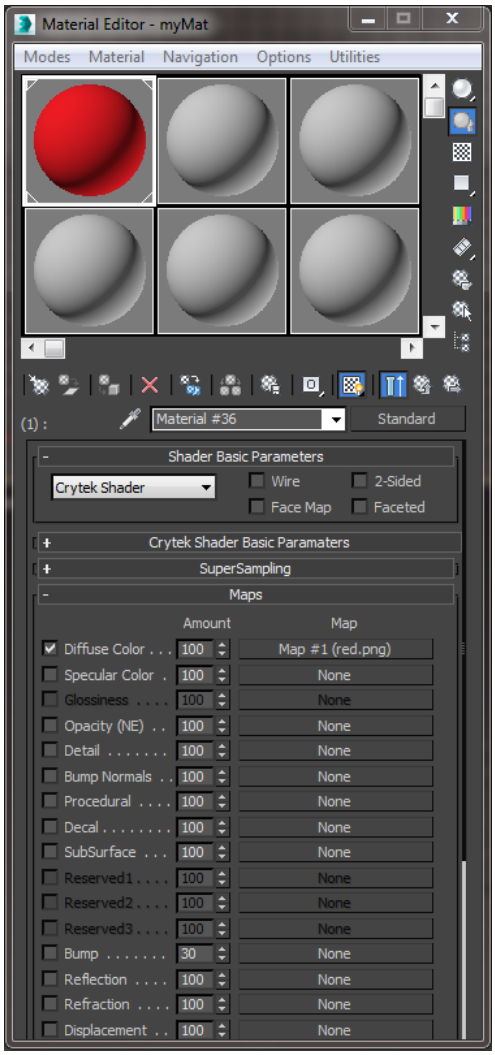
3. From the 3ds Max top menu, choose **Rendering**, **Material Editor**, **Compact Material Editor**.
4. From the 3ds Max top menu, choose **Rendering**, **Material/Map Browser.**



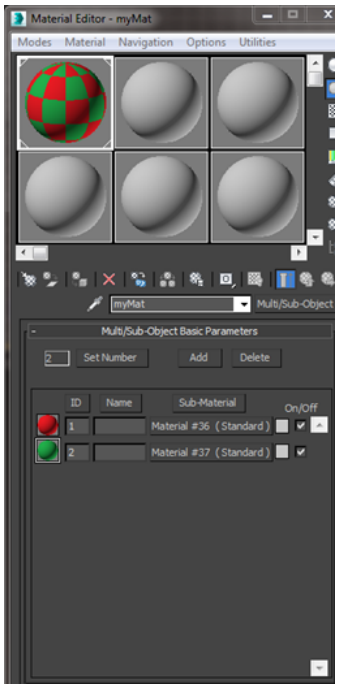
5. In **Material/Map Browser**, under **Materials**, expand **Standard**. Then double-click **Multi/Sub-Object**. In the 3ds Max **Material Editor**, under **Multi/Sub-Object Basic Parameters**, look for a material ID list to fill in. Select the first entry by clicking **None** in the **Sub-Material** column. Select **Standard** under the **Standard** material rollout.



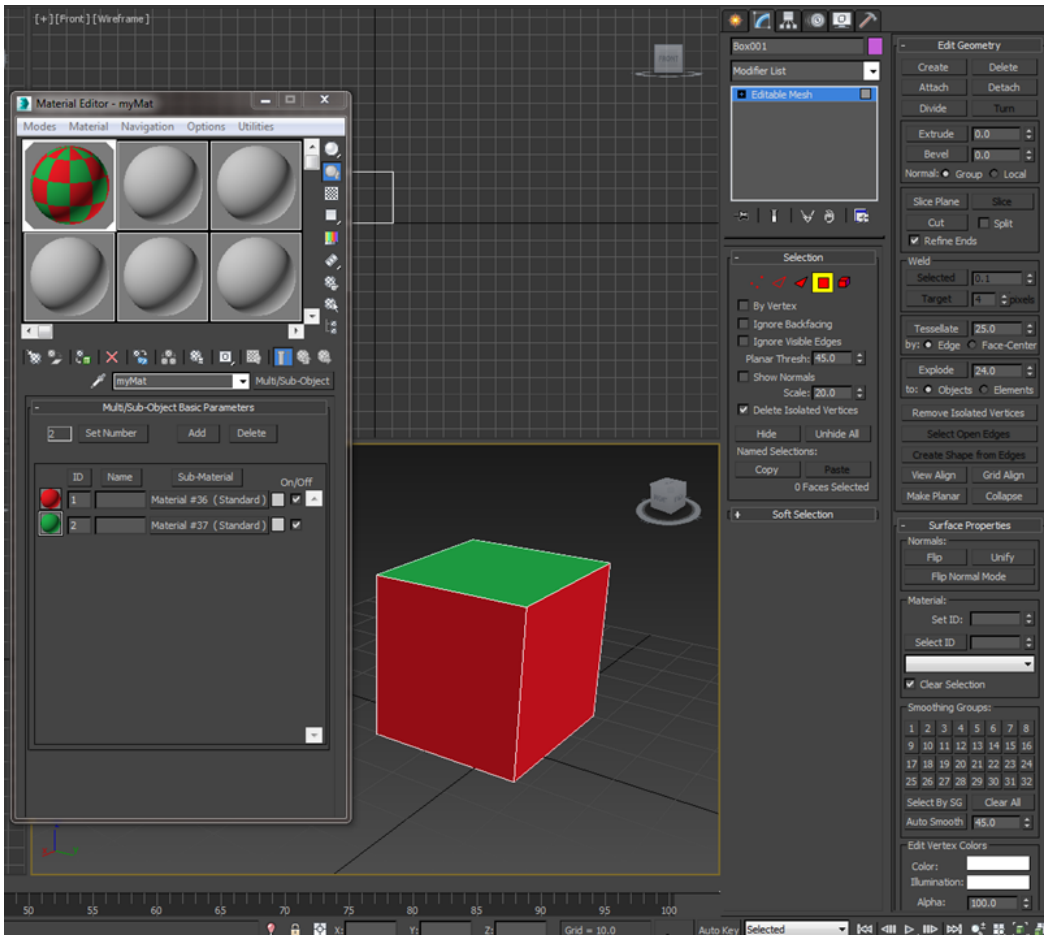
6. In the 3Ds Max **Material Editor**, under **Shader Basic Parameters**, select **Crytek Shader**.
7. Under **Maps**, next to **Diffuse Color**, select **None**.
8. In **Material/Map Browser**, under **Maps**, double-click **Bitmap**. Then double-click to select the desired image file. Afterward the image file appears in the 3DS Max **Material Editor** for the **Diffuse Color** parameter.



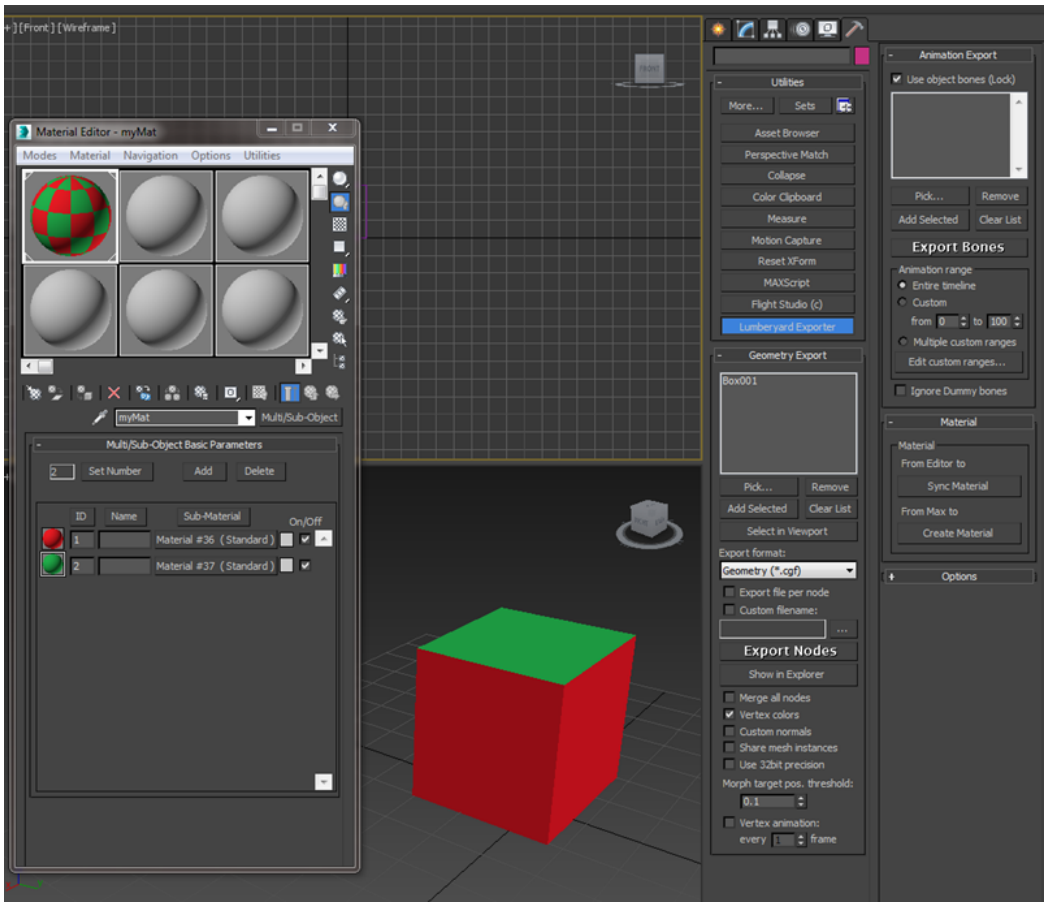
9. While still in **Material Editor**, choose **Navigation**, **Go to Parent**. Then repeat to get back to the material ID list.
10. Create a second subshader by repeating steps 5 through 9 for the second entry in the list. Click **Set Number**, then type `2` in the **Number of Materials** pop-up window. The list shows only two submaterials.
11. In **Material Editor**, under **Name**, type in a name.



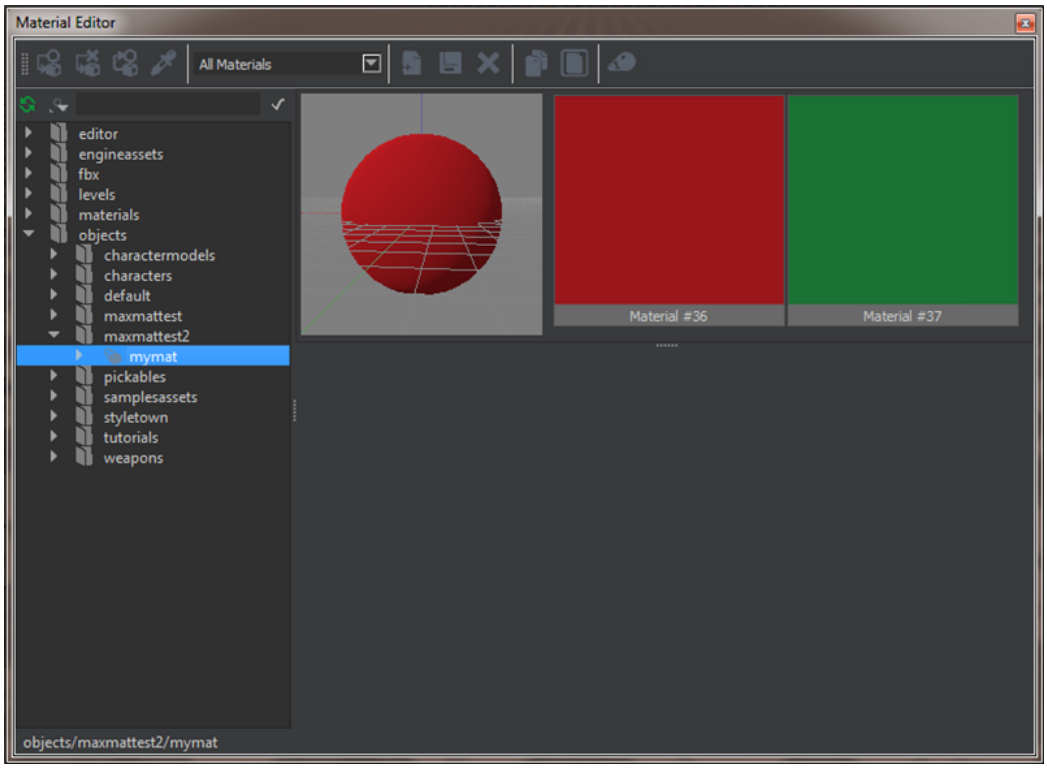
12. With the object selected in the viewport, go to **Material Editor** and choose **Material**, **Assign to Selection**.
13. Click the hammer icon. Under **Utilities**, select **Lumberyard Export**, select the object, and then choose **Add Selected** to place the object in the **Geometry Export** list.
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

- Working with Detail Maps (p. 1410)
- Working with Decals (p. 1411)
- Displacement Maps and Tessellation (p. 1415)

Texture Map Types

Source texture files are converted and compiled in `.DDS` format by the Resource Compiler (RC). If you do not specify presets for the source file, 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	_diff	Defines the main color for an object.
Normal map	_ddn	Defines the direction of normals on an object surface. The RGB normal map stores the normal direction for each pixel in the texture.
Normal with Gloss map	_ddna	Achieves physically-correct results. DDNA textures are standard DDN textures with the gloss map stored inside the Alpha channel.
Environment map	N/A	Makes an object reflective. The environment map stores the image that is reflected off the object.
Displacement map	_displ	Gives more depth and definition to an object when used in tessellations, parallax occlusion mapping (POM), and offset bump mapping (OBM).
Detail map	_detail	Adds more detail to a surface. This texture map works like a second material layer and is not affected by the mapping of the object.
Emittance Multiplier	N/A	Multiplies the emissive color.
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, and 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 saves on materials and drawcalls.
- Do not make textures bigger than they appear onscreen. A roof texture on a tall building that neither the player nor 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 to add depth to objects and make them more interesting.
- Use grayscale textures that can be color-tinted to save on texture memory. Objects that 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. 1410) .

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*x + y*y). This makes it possible to use two-channel textures (2 bytes per texel) to store normal maps.

Topics

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 it's 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 **Tools**, **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. 1374).

> **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

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.

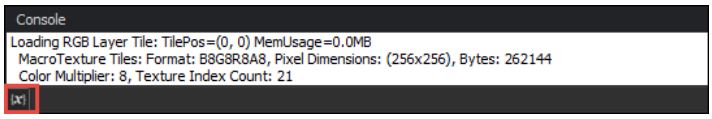
> **Note**
> If you apply decals to an object that can be moved by a player, the decal will not move with the object.

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. 1411).
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.



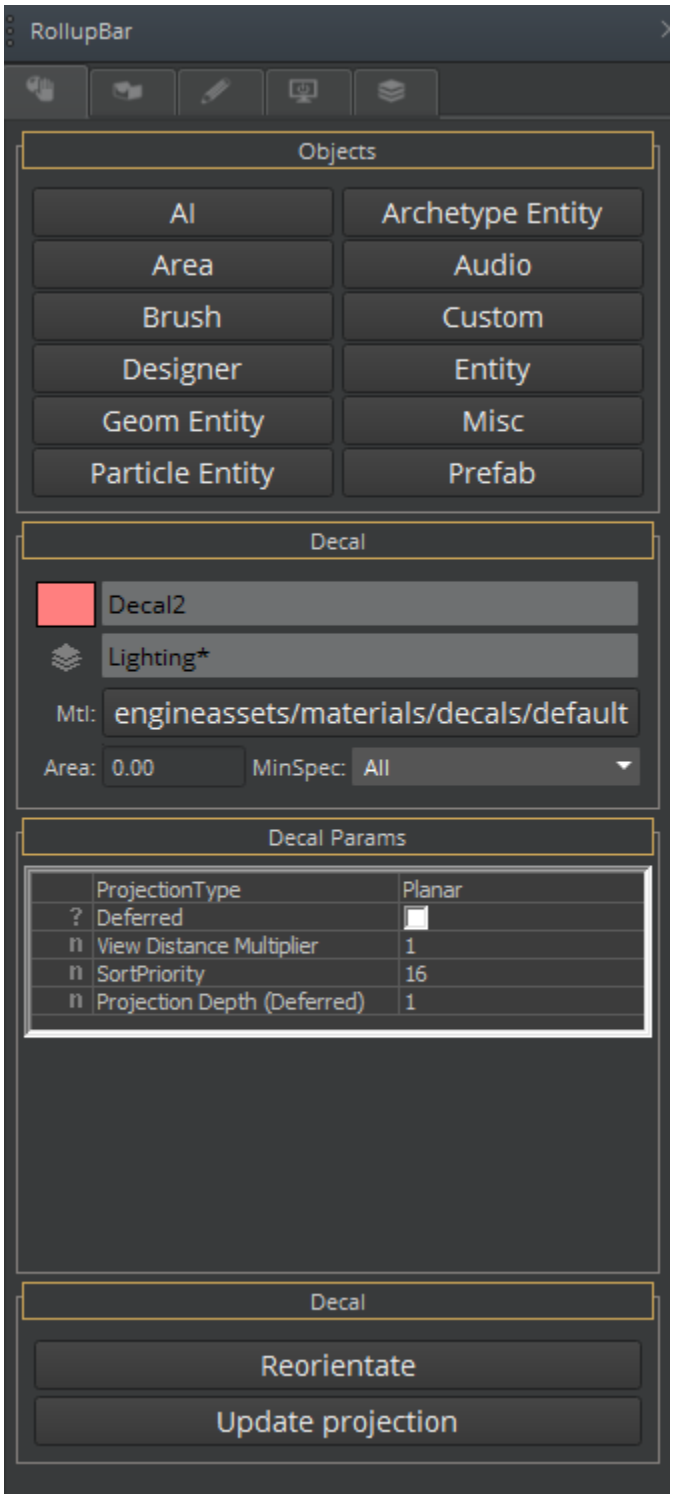
   b. In the **Console Variables** window, search for `r_deferredDecalsOnDynamicObjects`
   c. Set the value to any positive number, for example 1.
   d. Close the **Console Variables** window to save the new value.
3. Follow the instructions above for placing a decal.

Setting Decal Parameters

Complete the following procedures for setting decal mapping parameters.

To set decal parameters in the Rollup Bar

Most decal parameters are in the Rollup Bar on the **Objects** tab, under **Misc**, **Decal**.

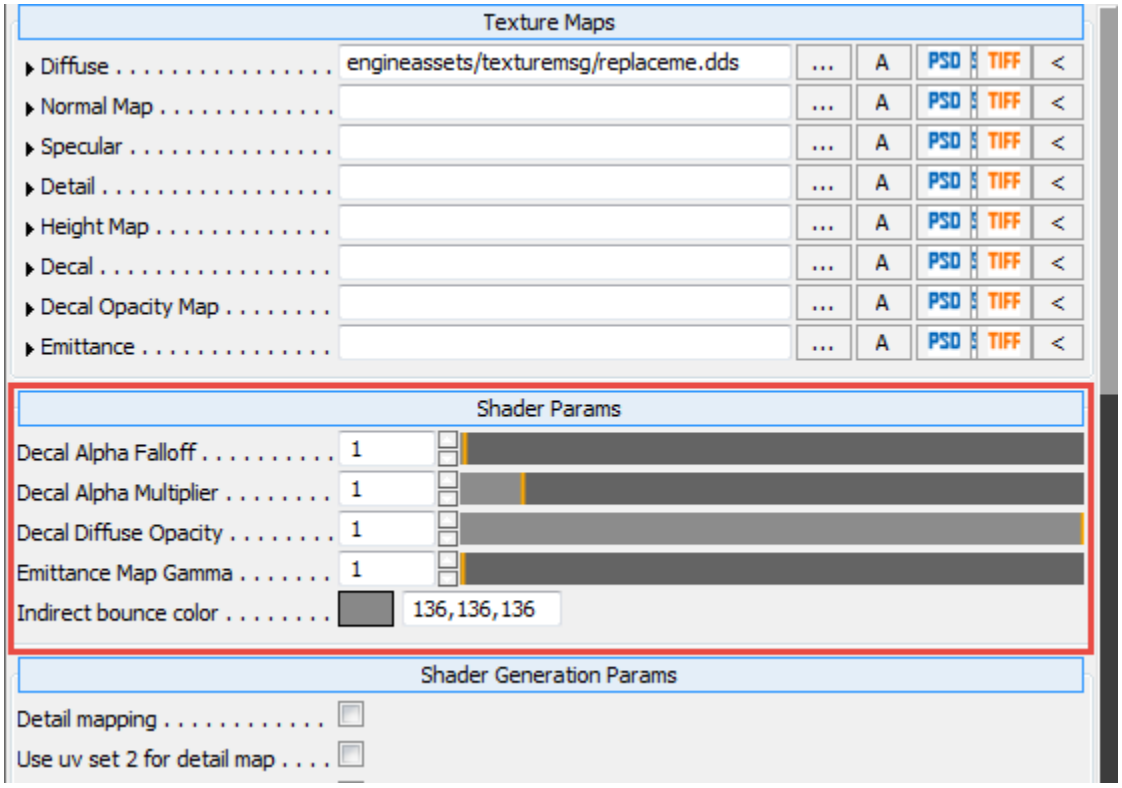


To set decal parameters in the Rollup Bar

1. In the Rollup Bar, under **Objects**, click **Misc**, **Decal**.
2. Under **Decal Params**, adjust the following parameters:
   - **ProjectionType** – Choose the projection type from the drop-down list: **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** – Specify if the decal will appear on top of another decal.
   - **Projection Depth** – Set the projection depth (distance) of the decal from the object. This can also affect blending of decals.

To set shader decal parameters

Some decal parameters are set in the Material Editor under **Shader Params**.



To set decal mapping parameters

1. In Lumberyard Editor, click **Tools**, **Material Editor**.
2. Click the **Add New Item** button.
3. Select a decals folder, select a subfolder, and then click **Save**. The new material will be selected automatically with the default settings.
4. Under **Shader Generation Params**, select **Decal**.
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** – Set the power applied to the decal alpha.
b. **Decal Alpha Multiplier** – Set the multiplier applied to the decal alpha.
c. **Decal Diffuse Opacity** – Set the opacity multiplier for the 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

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 **Tools**, **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. 1374) and HumanSkin Shader (p. 1372).

Topics

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 **Tools**, **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	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. 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. 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 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.

Exporting Substances for Lumberyard

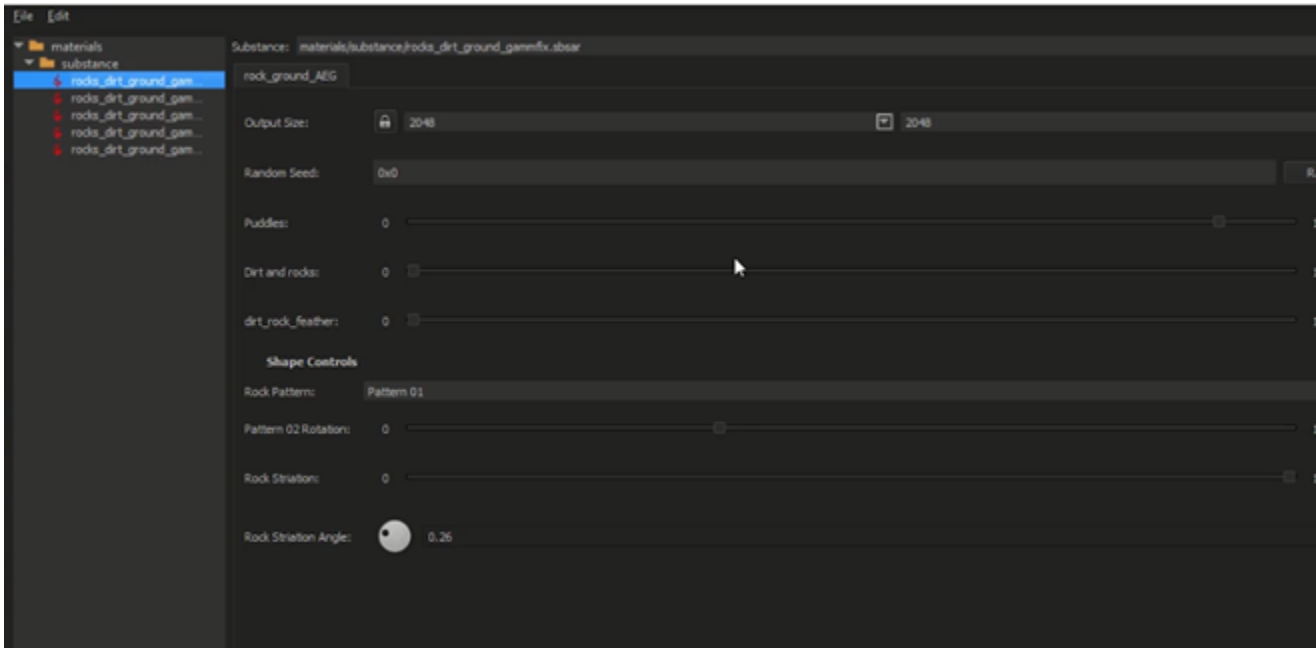
When using Allegorithmic's Substance Designer to export textures to Lumberyard, take extra care to select the correct file type. The following file types export correctly: `.tif`, `.tga`, `.bmp`. Avoid exporting with the following common file types: `.png`, `.jpg`, `.psd`.

To export textures in Substance Designer for Lumberyard

1. In the **File** menu, click **Export Textures**.
2. Click the **Configuration** tab, and then under **Presets**, select **Amazon Lumberyard**.
3. Click the **Export** tab. In the file type drop-down menu (upper right corner), select one of the following file types:
   - `.tif`
   - `.tga`
   - `.bmp`
4. Click **Export**.

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 Using the Project Configurator (p. 28). For more information on Gems, see Gems (p. 1060).
- When importing substance files, you must restart Lumberyard Editor before substance textures are rendered correctly.
- A `.smtl` (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 **Tools**, **Plug-Ins**, **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.

Console Variables for Substances

To control how Substances are handled by CPU and memory, you can set the following CVars in your `system_windows_pc.cfg` file or `editor.cfg` file.

`substance_engineLibrary`

Enables substances to run on the GPU instead of the CPU, permitting the output of higher resolution textures. Specifies the engine to load for substance plugin. Possible values are PC: [`sse2`|`d3d10`|`d3d11`]. The default is `sse2`, which specifies CPU.

`substance_commitRenderOptions`

Applies CPU and memory changes immediately, rather than waiting for the next render call.

`substance_coreCount`

Sets how many CPU cores are used for Substance rendering. A value of `32` specifies all cores. This setting is relevant only when using CPU based engines.

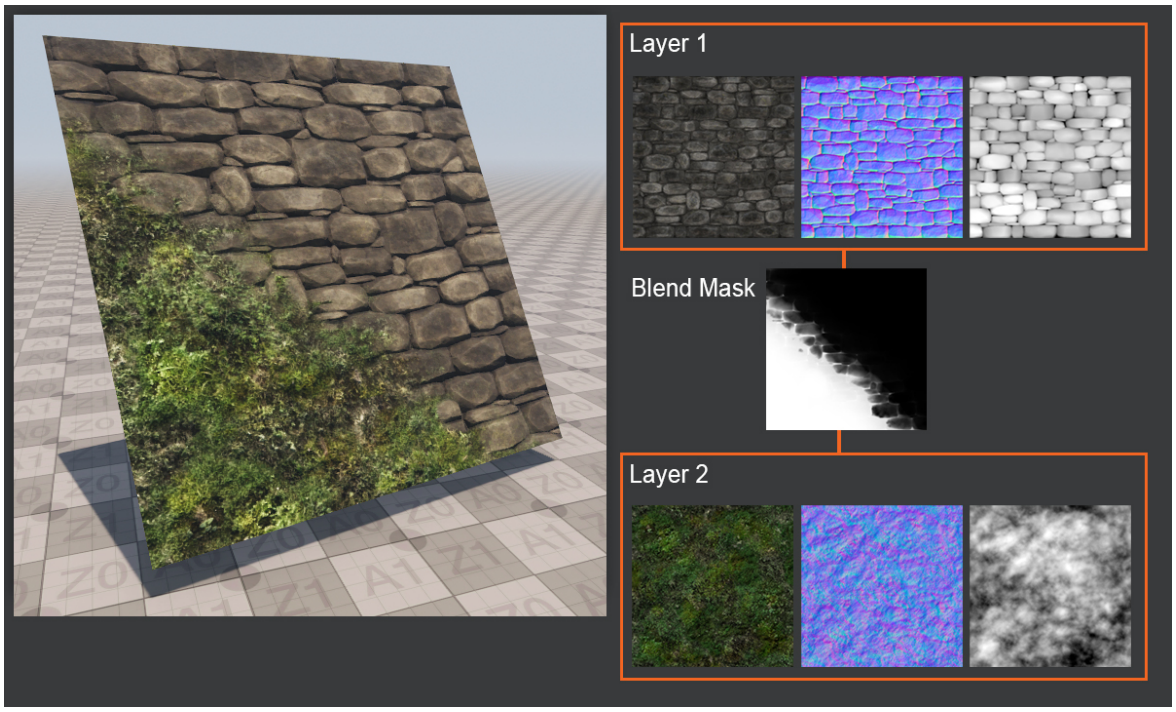
`substance_memoryBudget`

Sets, in megabytes, the amount of memory used for Substance rendering. The default is 512.

Working with Blend Layers

Blend layers can be thought of as a layered material. You can create one material with a second set of per-pixel functions, including texture maps, UV tiling and offset, rotation and oscillation animation, second UV set, diffuse and specular color tinting, smoothness, and blend masking.

Integrate blend layers in the Illum and Vegetation shaders and activate them by setting the blend layer shader generation parameters.



Topics

Shader Generation Parameters

You can set the following parameters for generating shaders.

Blend layer

Activate the texture inputs and parameters for the blend layer.

Use UV set 2 for blend layers maps

Use a second UV channel, if available, for the second layer.

Blend Layer Textures

You can set the following parameters for your blend layer textures.

Second Height Map

Displacement map for the blend layer (grayscale)

Second Diffuse Map

Diffuse map for the blend layer (RGB)

Second Normal Map

Normal map for the blend layer; the second layer gloss map is contained in the alpha (RGBA)

Second Specular Map

Specular map for the blend layer (RGB)

Blend Map

Blending map to blend between the first and second layers (grayscale)

Blend Layer Parameters

You can set the following parameters for your blend layer.

Blend Factor

Control visibility of the blend layer, with a bias toward black or white

Valid values: 0 – 16

Blend Falloff

Control the fall off of the blend range

Valid values: 0.1 – 128

Blend Layer 2 Tiling

Control tiling of the second blend layer

Valid values: 0 – 20

Default: 1

Blend Layer 2 Diffuse (Tint)

Base color tint for the second layer (RGB)

Valid values: 0 – 255

Blend Layer 2 Smoothness

Sharpness of specular reflection for the second layer

Valid values: 0 – 255

Default: 10

Blend Layer 2 Specular

Reflective brightness and color for the second layer (RGB)

Valid values: 0 – 255

Blend Mask Tiling

Control tiling of the blend map

Valid values: 0.05 – 20

Default: 1

Parallax Mapping

Parallax occlusion mapping (POM) is an enhancement of the traditional parallax mapping technique that creates detail in a texture to add the illusion of depth. This depth perception changes based on perspective.

Parallax occlusion mapping (POM) and offset bump mapping (OBM) are similar to displacement mapping and tessellation, but their computational requirements are not as demanding. However, POM is not suitable in every situation. Use POM only for high-performance computers and OBM for devices such as consoles. When you use POM, you must enable both shader generation parameters. Lumberyard automatically defaults to OBM for setups that cannot run POM.

Topics

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 **Tools**, **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 **Tools**, **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. 1424) 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. 1424) 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, colorizing 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. 1203).

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

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 **Tools**, **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.

Using the Screen Fader Effect

You can use the screen fader effect to control all screen fade properties, including duration of the fade effect and the color or texture to use for the fade.

You can stack multiple screen fader effects; multiple screen faders are rendered concurrently and in order of post effect group priority. Screen fader effects do not blend the parameters between layers.

The following example demonstrates how to use the screen fader effect in an `.xml` file:

```
<PostEffectGroup priority="2" hold="1">
    <Effect name="ScreenFader">
        <Param name="Enable" floatValue="1.0"/>
        <Param name="FadeInTime" floatValue="2.5"/>
        <Param name="FadeOutTime" floatValue="1.0"/>
        <Param name="ScreenCoordinates" vec4Value="0.0,0.0,1.0,1.0"/>
        <Param name="FadeColor" vec4Value="0.2,0.7,0.7,0.5"/>
        <Param name="TextureName" stringValue="textures/StyleTown/_dev_Blue_Light.tif"/>
    </Effect>
</PostEffectGroup>
```

Parameters

You can set the following parameters:

Enable

Determines whether or not the ScreenFader is active for the post-processing effect group.

Valid values: `0` = Disable | `1` = Enable

FadeInTime

Time, in seconds, for the screen fader to fade in, once enabled.

FadeOutTime

Time, in seconds, for the screen fader to fade out, once disabled.

ScreenCoordinates

Determines the rectangle where the screen fader is rendered. Specify the coordinates in the format (Left, Top, Right, Bottom). For example, a fullscreen quad is specified as `(0.0, 0.0, 1.0, 1.0)` and a quad that fills half of the screen is specified as `(0.0, 0.0, 0.5, 1.0)`.

Valid values: `0.0 - 1.0`

FadeColor

Sets the quad color by multiplying the color by the specified texture. If no texture is specified, the quad will be a solid color.

TextureName

Path of the texture to use for the screen fader.

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. 655).

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. 1188).

Topics

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. 655).

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. 1188).

Topics

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. 1431).

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. 1184) for more information.

- Sun color
- Sun color multiplier
- Sun intensity
- Sun intensity multiplier

HDR Settings

As discussed in Illuminance and Auto Exposure Key (p. 1430), 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 **Tools**, **Other**, **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. 1433).

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. 1432).

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. 1432).
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

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.

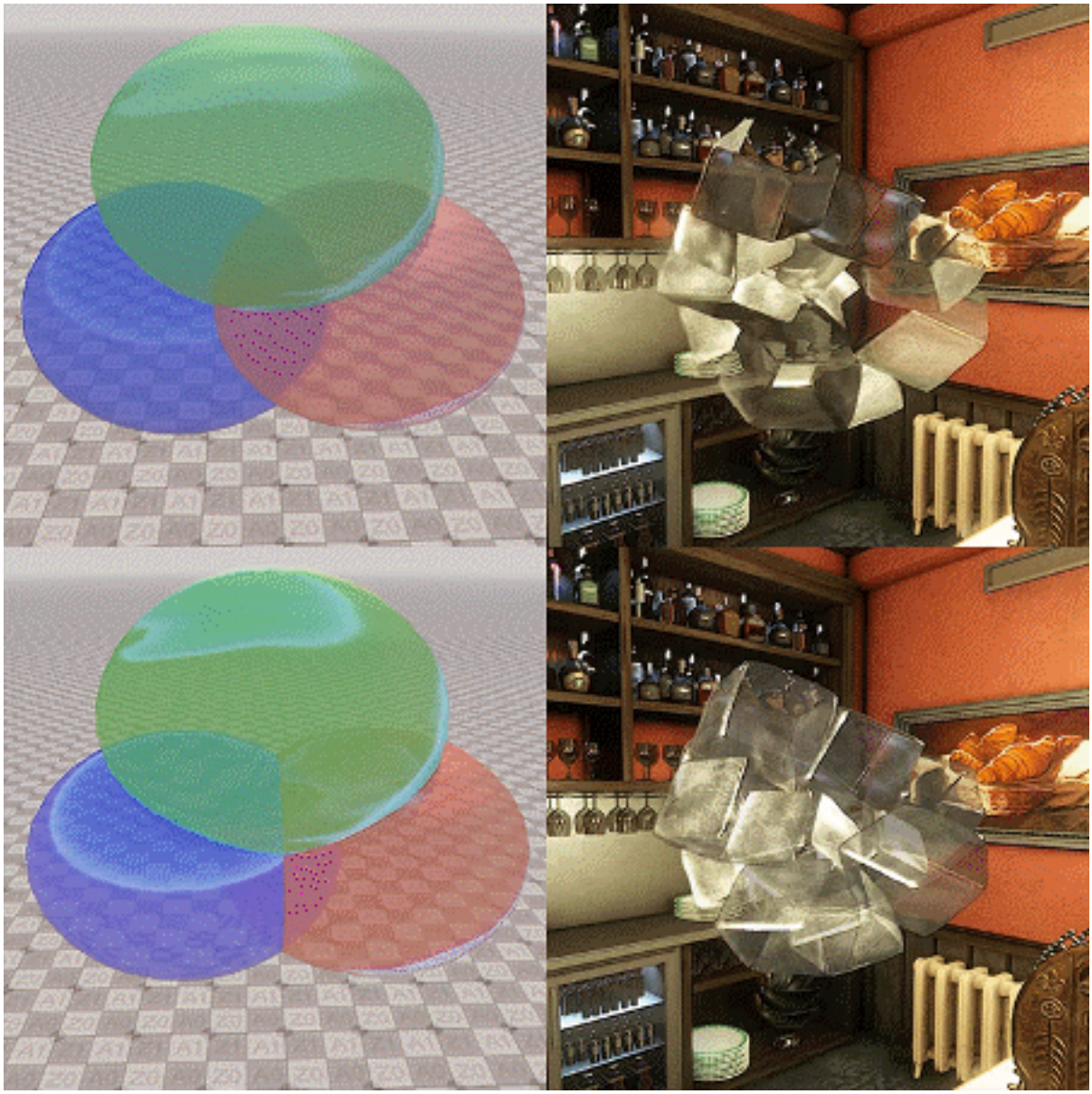
No material setup is required in your DCC tool. Instead you use the Material Editor to set up shadow proxies in the material using Material Editor. Place the shadow proxy on its own submaterial, setting **Opacity** to 0 and ensuring that **No Shadow** is not selected (the default).

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.

Order-Independent Transparency

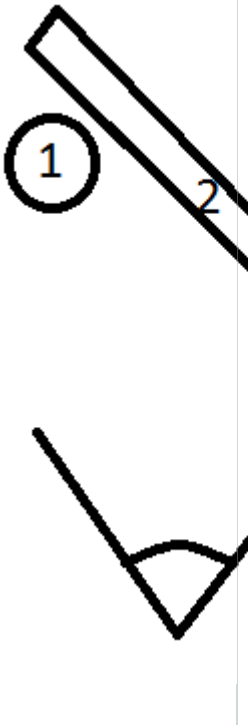
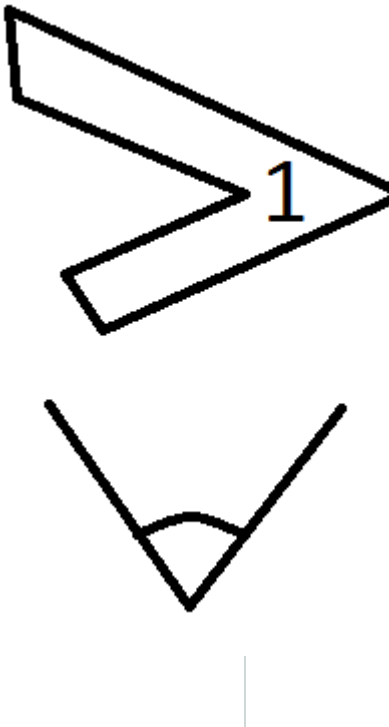
Order-independent transparency (OIT) corrects the display of transparent objects that are drawn out of order.



OIT is useful when creating the following:

- Concave geometry – When you create concave geometry, such as a glass, wine glass, or glass sculpture, some triangles may cover the same pixels and are drawn on top of each other. OIT solves the out-of-order issues that appear from certain angles.
- Intersecting geometry – When you create intersecting geometry, such as hair planes, some triangles may intersect in separate draws. OIT properly orders the triangles for each pixel.
- Transparent objects inside transparent objects – This includes liquids inside of glasses, holograms, or x-ray style effects.

To further demonstrate, see the following images.

OIT is not needed in the following example. Drawing object 1, then object 2, and finally object 3 works.	OIT is needed in the following example. Drawing object 1 and then object 2 will not work because certain pixels from object 1 must be in front of pixels from object 2 (and vice versa).	OIT is needed in the following example. Object 2 is long and its center point is in front of object 1, but it must draw behind object 1.	OIT is needed in the following example. Certain parts of object 1 draw behind other parts of the object, and you cannot control which draws first.
			

Configuring Order-Independent Transparency for Lumberyard

You can configure OIT with the console or create a level configuration file with this setting.

Specify the value for `r_AlphaBlendLayerCount` to the number of transparency layers that OIT can solve. A value of 1 is sufficient to solve most out-of-order transparency issues.

Valid values: 0 – 4

0 = Disables OIT

- Specify the value for the `r_AlphaBlendLayerCount` console variable
- Specify the value for the `level.cfg` file in the *`lumberyard_version`*`\dev\`*`<project_name>`*`\Levels\`*`<level_name>`* directory.

For more information, see Configuring Console Variables (p. 94).

> **Note**
> OIT requires the following:
>
> - Hardware requirements: DirectX 12_1 feature level compatible graphics card (Nvidia Maxwell & Pascal, 4th generation Intel core processors)
> - Software requirements: DirectX 11.3 and 12 runtime on Windows 10 compiled with Windows 10 SDK

Render Cameras and Effects

Topics

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. 1174) and Adding Volumetric Fog Shadows (p. 1450).

Topics

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

You can use global fog to simulate particles that are distributed uniformly along the ground and fall off exponentially with height above sea level. Global fog accurately accounts for time of day lighting and for scattered sunlight rays to produce haze at the horizon and halos around the sun. You can also use this effect to cast shadows for objects and clouds through the fog. For more information, see Adding Volumetric Fog Shadows (p. 1450).

To set global fog parameters

1. In Lumberyard Editor, click **Tools**, **Other**, **Time Of Day**.
2. In the **Time of Day Tasks** pane, under **Tasks**, click **Toggle Advanced Properties** to view all parameters.
3. In the **Parameters** pane, under **Fog**, adjust the following parameters as needed:

Parameter	Description
Color (bottom)	Specifies the bottom color for the fog.
Color (bottom) multiplier	Specifies the intensity of the bottom fog color. This value is multiplied by the top fog color to set the brightness of the bottom fog color. You can set this value to 3 for midday and add three keys to the timeline with the values set at 0.02. This creates a localized orange bloom around the sun as it descends.
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, this value is the height at which the vertical density falloff reaches the specified density.
Density (bottom)	Specifies the fog density at the bottom. Values greater than 0 or less than 1 cause the fog to gradually fall off.
Color (top)	Specifies the color of the fog component that produces halos around the sun and sunlight scatter.
Color (top) multiplier	Specifies the intensity of the top fog color. This value is multiplied by the bottom fog color to set the brightness of the top fog color.
Height (top)	Sets the reference height for the vertical fog gradient. For fog color, this value is the height at which the specified color reaches at the top.

Parameter	Description
	For fog density, this value is the height at which the vertical density falloff reaches the specified density.
Density (top)	Specifies the fog density at the top. You can set the top density to a higher value than the bottom density. Doing so would reverse the vertical falloff and produce thick fog in the sky and clear views at the bottom. You can also set the top and bottom density values to be equal. Volumetric fog computations treat a level as a continuous unbound volume. If you specify a density greater than 0 at the specified top height, the fog won't suddenly stop at that height. Instead the fog will continue to fall off gradually. The same action is true for bottom boundary or density values that are less than 1.
Color height offset	Offsets the height of the vertical fog gradient between the top and bottom colors.
Color (radial) multiplier	Specifies the multiplier of the fog color component that produces halos around the sun and sunlight scatter. Radial fog is more noticeable as the light intensity decreases, so you can decrease the multiplier towards the end of the timeline. Radial fog is also applied to the moon at night, so you can create two keys on the timeline and set the value to 0.
Radial size	Specifies the size of the radial fog glow (bloom around the sun), perpendicular to the camera. This value can also enhance effects such as the colors around the sun at sunrise or sunset.
Radial lobe	Specifies the size of the radial fog glow towards the camera or how much the radial fog is affected by distance. Higher values create a foggier scene. Lower values affect only the horizon.
Final density clamp	Sets the maximum density that the fog can reach. This enables the sky, horizon, and other bright, distant objects to be visible through dense fog. Setting this value too low can compromise depth perception and result in implausible visuals and apparent artifacts, especially when moving the camera.
Global density	Sets the density of the global fog. Higher values create denser fog.
Ramp start	Specifies the distance from the camera at which the fog starts to render at 0 density. This allows you to remove the fog around the camera and fade in at a specified distance.

Parameter	Description
Ramp end	Specifies the distance from the camera at which the fog ceases to render at 0 density. This allows you to remove the fog around the camera and fade out at a specified distance.
Ramp influence	Specifies how much the ramp values affect the rendering of the fog.
Shadow darkening	Controls the appearance of the fog in shadow areas. Specifies how much the fog color is darkened per pixel based on the volumetric shadow value per pixel. This value is applied after calculating the darkened fog color using the sun and ambient darkening factor. See the next two parameters.
Shadow darkening sun	Specifies how much the sun influences the radial fog color.
Shadow darkening ambient	Specifies how much the environment influences the ambient fog color height gradient.
Shadow range	Sets the distance that the volumetric shadows are rendered until 10% (0.1) of the level's clipping plane distance is reached. Lower values produce more accurate results; however, the shadows are not rendered as far as with higher values.

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 the Rollup Bar.

To add a fog volume to your level

1. In the **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.
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).

Parameter	Description
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 **Tools**, **Other**, **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 the Rollup Bar, on the **Terrain** tab, click **Environment**.
2. Under **Fog**, adjust the following values as needed:
   a. **View distance** – Specifies the distance at which the fog fades away.
   b. **View distance low spec** – Specifies the 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.
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. 655) in the Object and Entity System (p. 613).

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 Particles Attributes Reference (p. 1281).

To add localized nonvolumetric regions of fog, see Using Fog Volumes (p. 1442).

Topics

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

You can use global volumetric fog to simulate particles that are distributed uniformly along the ground and fall off exponentially with height above sea level. Global volumetric fog accurately accounts for time of day lighting and for scattered sunlight rays to produce halos around the sun.

Use the **Anisotropy** parameters to control the amount of sunlight that is scattered through fog and the direction. Set the **Anisotropy (atmosphere)** parameter close to 0 to achieve a uniform look across the entire sky. Set the **Anisotropy (sun radial)** parameter close to 1 to create a bloom effect around the sun.

The **Radial blend** parameters blend the **Anisotropy** parameters to create various effects. For example, you can create sun radial scattering by setting **Radial blend mode** to 1 and **Radial blend factor** to 1.

To set global volumetric fog parameters

1. In Lumberyard Editor, click **Tools**, **Other**, **Time Of Day.**
2. In the **Time of Day Tasks** pane, under **Tasks**, click **Toggle Advanced Properties** to view all parameters.
3. In the **Parameters** pane, under **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, this value is the height at which the vertical density falloff reaches the specified density.
Density (bottom)	Specifies the fog density at the bottom. Values greater than 0 or less than 1 cause the fog to gradually fall off.
Height (top)	Sets the reference height for the vertical fog gradient. For fog color, this value is the height at which the specified color reaches the top. For fog density, this value is the height at which the vertical density falloff reaches the specified density.
Density (top)	Specifies the fog density at the top. You can set the top density to a higher value than the bottom density. Doing so would reverse the vertical falloff and produce thick fog in the sky and clear views at the bottom. You can also set the top and bottom density values to be equal. Volumetric fog computations treat a level as a continuous unbound volume. If you specify a density greater than 0 at the specified top height, the fog won't suddenly stop at that height. Instead the fog will continue to fall off gradually. The same action is true for bottom boundary or density values that are less than 1.
Global density	Sets the density of the global volumetric fog. Higher values create denser fog.
Ramp start	Specifies the distance from the camera at which the fog starts to render at 0 density.
Ramp end	Specifies the distance from the camera at which the fog ceases to render at 0 density.
Color (atmosphere)	Specifies the fog color for the atmosphere. For example, you can set the fog color to light blue (140, 230, 255) for midday, darker blue (90, 148, 164) for sunset or sunrise, and white for night time.
Anisotropy (atmosphere)	Adjusts the anisotropy for sun atmosphere scattering. Valid values: -1 to 1. Values less than 0 will shift the fog in the opposite direction of the sun. Values greater than 0 create a consistent appearance of atmospheric fog.
Color (sun radial)	Specifies the color of the glow around the sun.
Anisotropy (sun radial)	Adjusts the anisotropy for sun radial scattering. Values less than 0 will shift the glow effect to the opposite side of the sun. Values greater than 0

Parameter	Description
	make the radial color visible even when looking away from the sun.
Radial blend factor	Blends the sun radial color with the atmosphere color. Valid values: 0 to 1. Set the value to 0 to turn off radial fog.
Radial blend mode	Adjusts the blend mode for blending between atmosphere fog and radial fog. Set the value to 0 to use additive blending. Set the value to 1 to use linear interpolation.
Color (entities)	Specifies the global fog color for volumetric lights, excluding the sun.
Anisotropy (entities)	Adjusts the appearance of volumetric fog entities based on the viewing angle in relation to the sun. 0 = isotropic | 1 = perfect forward | -1 = perfect backward in-scattering.
Range	Adjusts the maximum radius of volumetric fog. The default setting is 64.
In-scattering	Specifies how much light (including sun) is scattered by fog. Higher values create a foggier scene (fog density remains unchanged) with bigger and brighter glow effects.
Extinction	Specifies how much light the fog absorbs. Higher values create a thick, atmosphere effect that is tough for sun light to penetrate.
Analytical fog visibility	Adjusts the global visibility of analytical fog. Set the value to 0 to hide analytical volumetric fog. Set the value to 1 to display analytical volumetric fog.
Final density clamp	Sets the maximum density that the fog can reach. This enables the sky, horizon, and other bright, distant objects to be visible through dense fog. Setting this value too low can compromise depth perception and result 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.

For more information about how to add and use rendering cameras, see Camera Entity (p. 653).

Depth of Field

Lumberyard uses an efficient gather-based depth of field (DOF) implementation. Depth of field is used to 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.

Temporal Antialiasing and Supersampling

Supersampling is a method of antialiasing that attempts to reduce jagged, pixelated edges (aliasing) in images. It works by sampling a higher resolution version of the image to get the average color of a pixel before reducing it to the intended size. The averaged color values create smoother transitions along an edge of colors, reducing the jagged look.

Because supersampling is memory intensive, Lumberyard uses temporal antialiasing (TAA) to approximate supersampling. While supersampling analyzes pixels spatially, TAA analyzes frames over time, even when the camera is static. The current frame is projected onto the previous frame and samples are blended into an accumulation buffer. This technique reduces ghosting artifacts in motion and gives you control over the amount of antialiasing needed for your graphics. You can create sharp images or softer, blurred images. You can also use supersampling for very high quality rendering.

TAA is useful for reducing aliasing from the following:

- Specular highlights and bright pixels, especially when combined with convolution filters such as **Depth of Field** or **Bloom**.
- Geometric and alpha-tested edges.

You can configure TAA by setting the `r_AntialiasingMode` console variable, editing the platform configuration file, or creating a level configuration file with this setting.

Controlling Antialiasing

The following table lists the antialiasing modes that are available in Lumberyard when you use the `r_AntialiasingMode` console variable.

Mode	Console Variable Value	Description
No antialiasing	0	Disables postprocessing-based antialiasing. This is useful for debugging or when the technique is unnecessary. You may also use a higher resolution if you prefer not to spend system resources on antialiasing.
FXAA	1	Enables fast approximate antialiasing (FXAA), which filters edges using a postprocessing edge detection technique.
SMAA (1TX)	2	Enables subpixel morphological antialiasing (SMAA), which uses an advanced postprocessing technique to detect edges in order to filter edges. This mode includes a basic temporal antialiasing component but does not address subpixel jitter.
TAA	3	Enables temporal antialiasing (TAA). This is the default mode.

The following table lists the additional console variables that you can configure for temporal antialiasing.

Console Variable	Description
`r_AntialiasingModeEditor`	Specifies whether to use subpixel jitter in the editor. This can eliminate jittering artifacts on helper objects at the expense of losing antialiasing on static scenes. Valid values: 0 = disables this mode | 1 = enables this mode
`r_AntialiasingTAAClampingFactor`	Controls the clamping factor in standard deviations. Set the value at which to limit the TAA response. Higher values create a more stable scene with less flickering but more ghosting. Lower values create a scene with less ghosting but more aliasing and flickering. Valid values: 0.75 – 2.0
`r_AntialiasingTAAJitterPattern`	Specifies the sampling pattern for temporal antialiasing. Valid values: 0 = no subsamples | 1 = 2x | 2 = 3x | 3 = 4x | 4 = 8x | 5 = sparse grid, 8x8 | 6 = random | 7 = Halton 8x (default) | 8 = Halton random
`r_AntialiasingTAALuminanceMax`	Clamps the input luminance before temporal filtering to help with image stability. Extra bright pixels can ghost and cause bloom artifacts. Default value: 100.0
`r_AntialiasingTAAMotionDifferenceMax`	Specifies the maximum difference of speed between the current pixel and its history pixel at which the current pixel is considered fully disoccluded. Lower values create a scene with less ghosting but more aliasing on disoccluded pixels.
`r_AntialiasingTAAMotionDifferenceMaxWeight`	Specifies the blend weight for the current frame at the maximum speed difference (defined by the `r_AntialiasingTAAMotionDifferenceMax` console variable). Valid values: 0 = 100% of the history pixel | 1 = 100% of the current pixel Default value: 0.5
`r_AntialiasingTAANewFrameFalloff`	Represents the amount of time, in seconds, for the history signal to reach 63% of the source signal. Lower values create faster convergence, which can reduce ghosting but introduce some aliasing. Higher values create slower convergence. Because higher values may not improve stability, you should adjust the color clamping factor before adjusting this value. Default value: 0.15

Console Variable	Description
`r_AntialiasingTAASharpening`	Controls the sharpening filter to help retain sharpness. This is useful when temporal antialiasing introduces blur to a scene during motion. Default value: 0.2
`r_AntialiasingTAAUseAntiFlickerFilter`	Reduces jitter-based flickering in certain scenarios. Valid values: 0 = disables this mode | 1 = enables this mode
`r_AntialiasingTAAUseJitterMipBias`	Enables mipmap biasing on textures when jitter is enabled. This creates a scene with decreased blur but more flickering. Valid values: 0 = disables this mode | 1 = enables this mode Default value: 1
`r_AntialiasingTAAUseVarianceClamping`	Enables variance color clamping to help reduce ghosting. This may increase flickering in the scene. The `r_AntialiasingTAAClampingFactor` console variable affects only this mode. Valid values: 0 = disables this mode | 1 = enables this mode Default value: 0

The following images illustrate the range of graphics quality that you can achieve depending on which antialiasing setting you use.

Temporal Antialiasing Limitations

Temporal antialiasing (TAA) is an inexpensive solution that reduces aliasing from all sources and works well with most content. Because it relies on screen-space information from previous frames, TAA has limitations in the following scenarios:

- Objects occlude other objects while in motion. In this scenario, the newly visible areas in the current frame don't have a history value to blend with and can manifest as ghosting.
- Motion vectors are not present for objects in motion, which can result in subtle smearing artifacts. Motion vectors are not generated for merged vegetation, non-CGF-based particles, or transparent materials. In certain high-frequency signal scenarios, the most apparent artifact is flickering caused by subpixel jittering that alternates between bright and dark pixels.
- The content has a lot of subpixel triangles and normals. In this scenario, the subpixel detail can cause flickering artifacts with temporal subpixel jittering. When a bright edge that is surrounded by dark pixels becomes smaller, the neighborhood clamping heuristic causes the pixels to flicker between light and dark.
- Transparency does not write depth. In this scenario, certain transparent content can exhibit subtle smearing artifacts.

In these cases, you may prefer to use subpixel morphological antialiasing (SMAA) or fast approximate antialiasing (FXAA).

Temporal Antialiasing Best Practices

We recommend following these best practices when using temporal antialiasing (TAA).

- Build content to alias as little as possible.
- Use level of detail (LOD) to reduce subpixel detail for objects that are far away.
- Use the antiflicker filter to help with flickering. The `r_AntialiasingTAAUseAntiFlickerFilter` console variable is enabled by default.

Controlling Supersampling

In addition to antialiasing, Lumberyard supports supersampling for very high quality rendering. Supersampling renders the scene at a higher resolution and downscales the image to obtain smooth and stable edges. Due to the high internal rendering resolution, supersampling is performance heavy and only suitable for games intended to be played on high-end computers.

Toon Shading (Experimental Feature)

Toon shading is a feature with which you can simulate cartoon effects for your game. Instead of using a shade gradient, toon shading uses less shading color to make 3D graphics appear flat. You can use toon shading to create a comic book or cartoon style for your game. Lumberyard's toon shading feature projects the scene luminance into a lookup table, which controls the smoothness of shading on the surface to achieve the intended look.

As a best practice for toon shading, use clear textures with fewer patterns when designing game art. Also, create models with fewer features at a lower level of detail (LOD). Otherwise, your models can have darker meshes in the distance due to their feature lines. For example, if you have a high density of polygons for vegetation, the vegetation can appear too dark or have too many black lines.

To enable toon shading, use the console to specify the console variable (CVAR): `r_AlphaBlendLayerCount`.

Valid values: 0 – 1

For more information, see Configuring Console Variables (p. 94).

The following example level does not have toon shading enabled (`r_AlphaBlendLayerCount=0`).

The following example level has toon shading enabled (`r_AlphaBlendLayerCount=1`).

See additional examples of toon shading enabled:

TrueType Font Rendering

CryFont is used to generate font textures that are required to render text on the screen. The various features of font rendering can be seen by using the `r_DebugFontRendering` console variable. For more information, see Configuring Console Variables (p. 94).

The output is not only to test the functionality but also to document how the features can be used.

Supported Features

CryFont supports the following features:

- Font shaders – Used to configure the appearance of fonts. Multiple passes with configurable offset and color are supported to enable generation of shadows or outlines. A sample font shader is shown in the following XML example.

```
<fontshader>
  <font path="VeraMono.ttf" w="288" h="416"/>
  <effect name="default">
    <pass>
      <color r="0" g="0" b="0" a="1"/>
      <pos x="1" y="1"/>
    </pass>
  </effect>
  <effect name="console">
    <pass>
      <color r="0" g="0" b="0" a="0.5"/>
```

```
      <pos x="2" y="2"/>
    </pass>
  </effect>
</fontshader>
```

The attributes *w* and *h* of the XML font element specify the width and height of the font texture. The order of the passes in XML defines the order in which the passes are rendered. A `<pass>` element without child elements means that the pass is rendered with the default settings. The `<pos>` tag is used to offset the font, while the `<color>` tag is used to set font color and define the transparency (with the alpha channel *a*).

- Unicode – The default font used does not support all Unicode characters (to save memory), but other fonts can be used.
- TrueType fonts as source – Cached in a small texture. Common characters are pre-cached, but runtime updates are possible and supported.
- Colored text rendering
- Adjustable transparency
- Color variations within a string – Use a value of **`$0..9`** to set one of the 10 available colors. Use **`$$`** to print the $ symbol, and **`$o`** to switch off the feature.
- Returns and tabs within a string
- Text alignment
- Computation of a string's width and height – Used internally to handle center and right alignment.
- Font size variations – Bilinear filtering allows some blurring, but no mipmaps are used so this feature has limitations in minification.
- Proportional and monospace fonts
- Pixel-perfect rendering with exact texel-to-pixel mapping for best quality.

Useful Console Commands

The following console commands provide information about font rendering. For more information, see Configuring Console Variables (p. 94).

r_DebugFontRendering

Provides information on various font rendering features, useful for verifying function and documenting usage.

- 0=off
- 1=display

r_DumpFontNames

Logs a list of fonts currently loaded.

r_DumpFontTexture

Dumps the texture of a specified font to a bitmap file. You can use `r_DumpFontTexture` to get the loaded font names.

Voxel-based Global Illumination (SVOGI)

Sparse voxel octree global illumination (SVOGI), 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

You can apply voxel GI through several modes.

Mode 0

With mode 0, only opacity is voxelized. The bounced light is sampled directly from shadow maps—extended to reflective shadow maps—and compute shaders are not used.

Mode 0 has these 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 has these 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 (Experimental)

With 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:

- Modes 1 and 2 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.

Modes 1 and 2 have these disadvantages:

- Modes 1 and 2 use more GPU memory (64 MB+).
- Large semistatic multibounce lights cannot be moved freely, but moving sun may work fine.
- Dynamic objects cannot affect GI, but can receive it.

> **Note**
> If you receive a message that the display driver has stopped responding and has recovered, try this workaround from Microsoft.

Voxel GI Parameters

The following parameters are global for an entire level. You can use normal ambient lights to modulate or tint indirect light intensity locally.

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	Activate voxel GI for the level.
Injection multiplier	Modulate light injection by controlling the intensity of bounce light.

Parameter	Description
Sky color multiplier	Control the amount of the sky light. You can multiply this value with the **Time of Day** fog color.
Saturation	Control the color saturation of propagated light.
Diffuse bias	Add constant ambient value to GI to prevent completely black areas. Set this parameter to negative to modulate the ambient value with near-range ambient occlusion and prevent constant ambient light in completely occluded indoor areas.
Cone max length	Set the maximum length, in meters, of the tracing rays. Shorter rays work faster.
Update geometry	Enable this parameter to force single complete revoxelization of the scene. This is required if terrain, brushes, or vegetation are modified.
Low spec mode	Set this parameter to a value greater than 0 to simplify shaders and scale down internal render targets. Set this parameter to –2 to initialize the mode by the value specified in the `sys_spec_Shading.cfg` file when the level is loaded.
Use light probes	For integration mode 0, enable this parameter to multiply environment probes lighting with GI. Disable this parameter to replace diffuse contribution of environment probes with GI. For integration modes 1 and 2, enable this parameter to use global environment probe for sky light instead of **Time Of Day** fog color.

3. (Optional) Use the **Voxel Coverage** advanced parameter on each material:
   a. Click **Tools**, **Material Editor**.
   b. Select your material.
   c. In the lower pane, click **Advanced**.
   d. Under **Advanced**, modify the **Voxel Coverage** parameter to control the transparency of voxels for each material and manually fix overoccluded areas. The default value is 1.

      Note
      This parameter takes effect only when voxel global illumination is enabled.

Debugging

You can use the following console variables to help debug voxel GI issues:

- `e_svoDebug=6` – Visualizes the voxels. Voxelize all important objects in the scene; otherwise, they will not cast occlusion and secondary shadows. Exclude all unwanted and unnecessary occluders from voxelization.
- `r_ShowRenderTarget svo_fin` – Displays the output of the voxel GI system.
- `r_profiler 1 | 2` – Retrieves GPU profiling information.

Do not use the `e_svoTI_*` console variables (for example, `e_svoTI_IntegrationMode`) to configure the voxel GI system. Any changes to these console variables in the configuration file will be overwritten by the individual level environment settings file.

Configure your settings in the **Total Illumination** panel in Lumberyard Editor. To enable the extended total illumination settings in the editor, click **Edit**, **Editor Settings**, **Global Preferences**. In the **Preferences** window, choose **Experimental Features**, **Lighting** and select **Total Illumination**. Click **OK** to save your selection.

Current Limits

The following limitations exist for the voxel GI system:

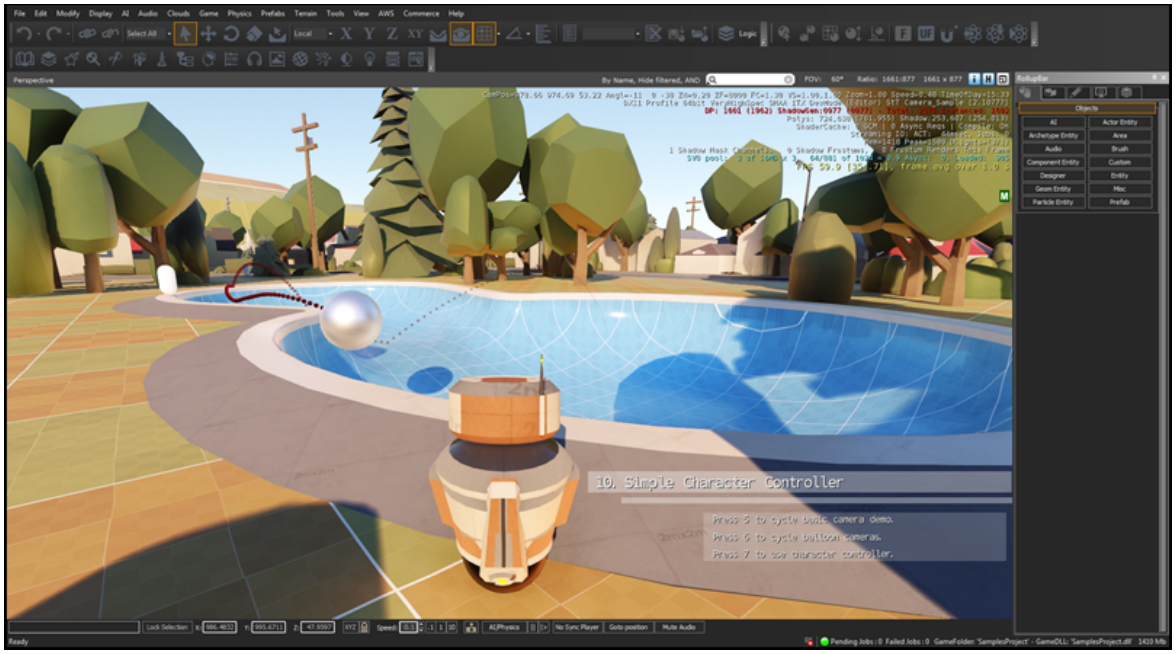
- Large-scale ambient occlusion and indirect shadows are properly cast only by static geometry.
- Voxel GI does not function on some forward-rendering components like particles or water.
- Some artifacts like ghosting, aliasing, light leaking, and noise may be noticeable.
- Procedural vegetation and 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 works properly.
- Only objects and materials with shadow map casting enabled will generate correct bounced light.
- For dynamic objects, indirect light bounce will function 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 adversely affect the appearance of the voxel GI. You can set the **LowSpecMode** value two times lower to restore the appearance of the voxel GI. **Temporal AA** using `r_AntialiasingMode 2/3` works correctly as well.

Sample Projects and Levels

Lumberyard offers a variety of sample projects, levels, and assets for you, which are located in the `\lumberyard_version\dev` directory. Use the Project Configurator to set or change your default project. For more information, see Using the Project Configurator (p. 28).

Samples Project

The Samples Project includes several gameplay levels and content that you need to follow the Lumberyard tutorials.

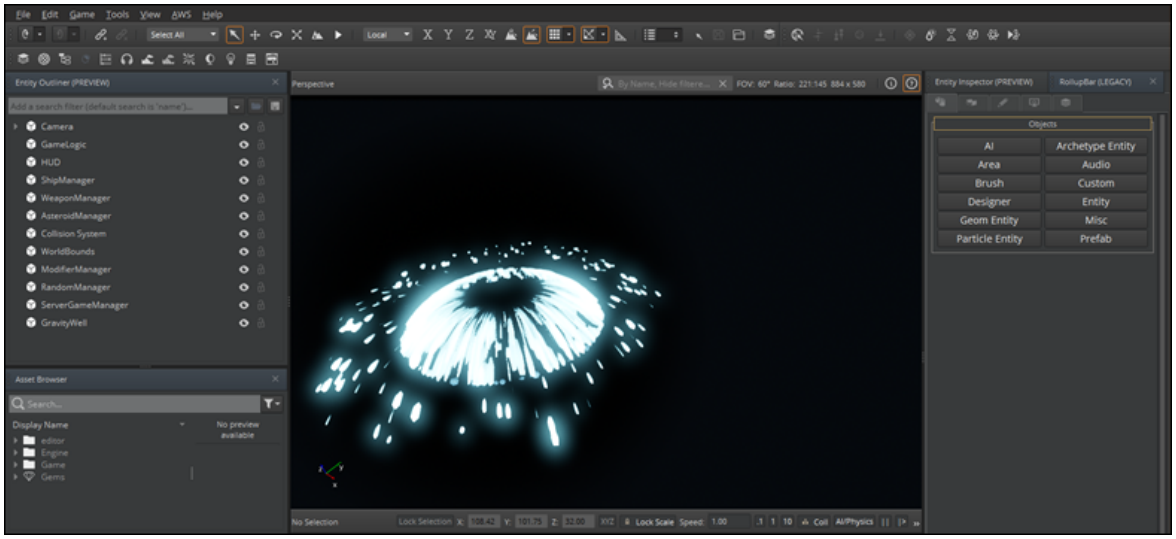


Starter Game

The Starter Game sample demonstrates how Lumberyard systems work together to make a game. The systems and features include the component entity system, bipedal locomotion, voxel-based global illumination, and the time of day system.

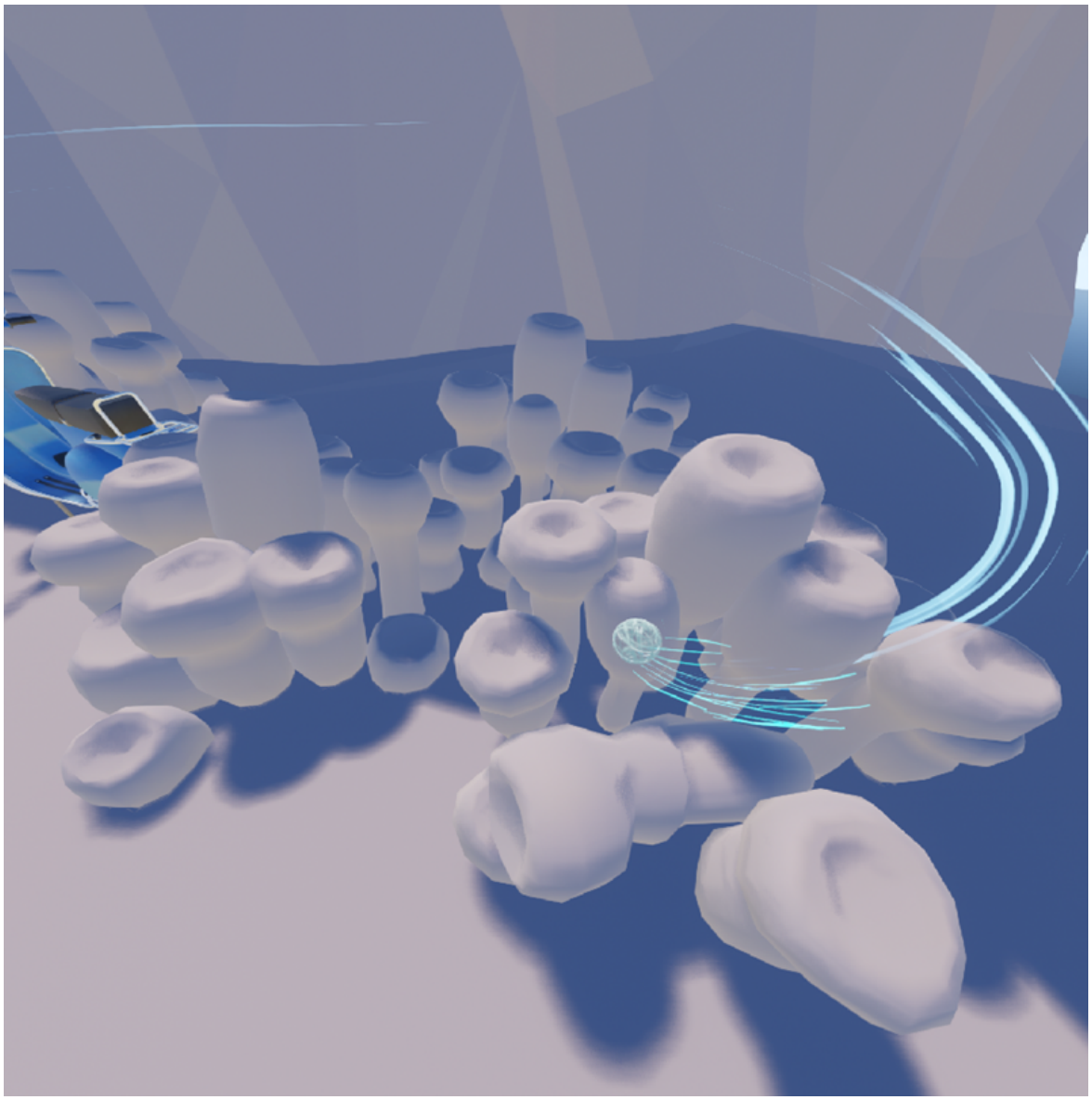
Multiplayer Sample

The Multiplayer Sample game enables you to test the Lumberyard GridMate networking features.



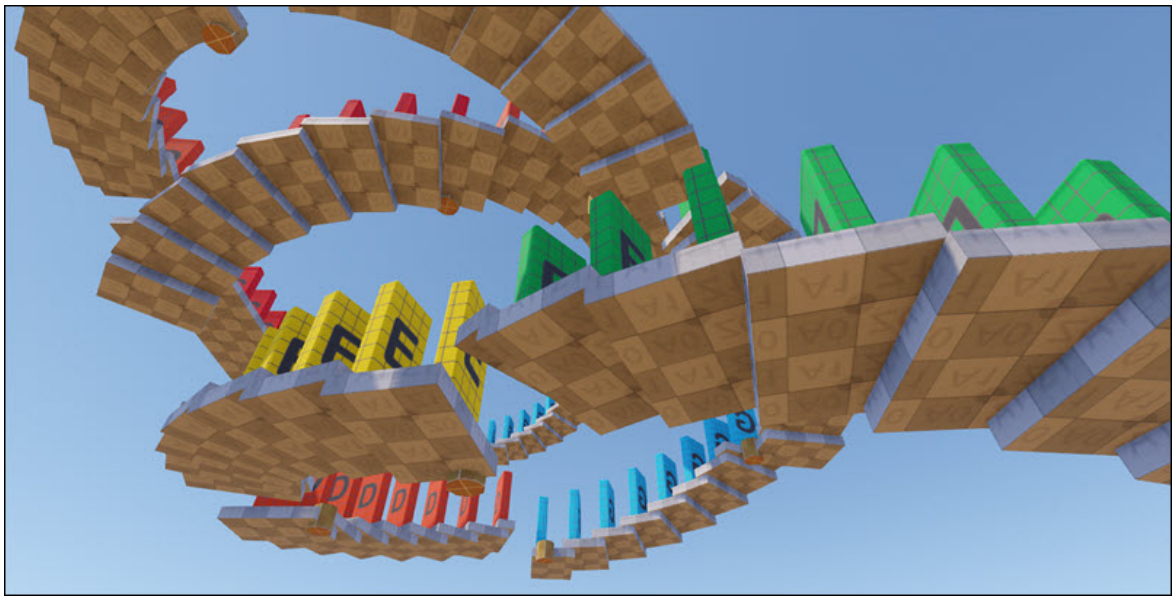
VR Islands Level

The VR Islands level showcases the potential of a simple VR level started from instantVR. This level features floating islands between which a player can teleport.



Virtual Reality Project Sample

The Virtual Reality Project sample is a template that you can use to build VR applications for any supported device.

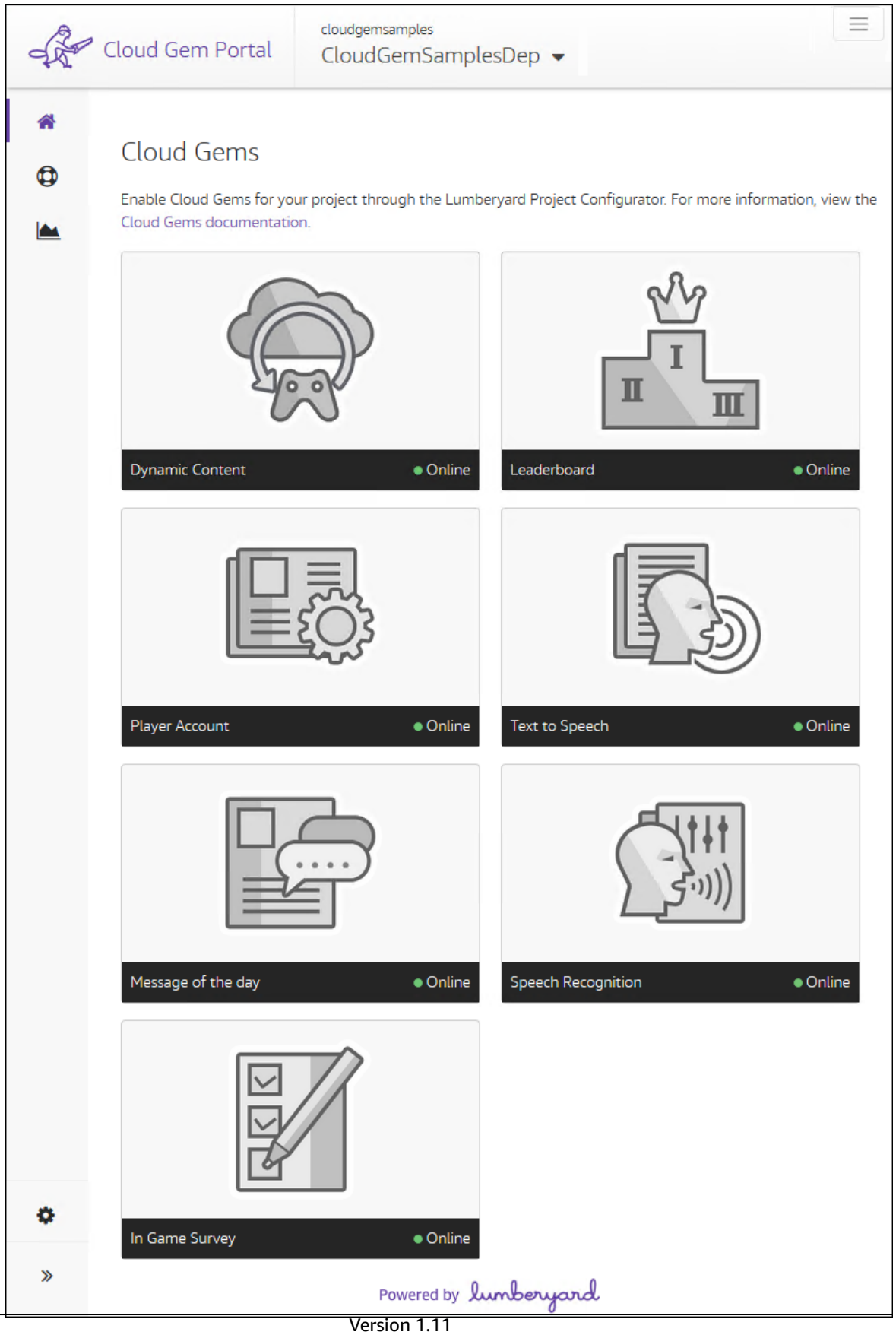


In-App Purchases

The In-App Purchases sample demonstrates how to use the In-App Purchases (IAP) Gem for a Lumberyard application that runs on a mobile device.

Cloud Gem Samples Project

The Cloud Gem Samples Project has cloud gems that you can use to add cloud-connected functionality to your game.


Cloud Gem Portal
cloudgemsamples
CloudGemSamplesDep
Cloud Gems
Enable Cloud Gems for your project through the Lumberyard Project Configurator. For more information, view the Cloud Gems documentation.
Dynamic Content
Online
Leaderboard
Online
Player Account
Online
Text to Speech
Online
Message of the day
Online
Speech Recognition
Online
In Game Survey
Online
Powered by lumberyard

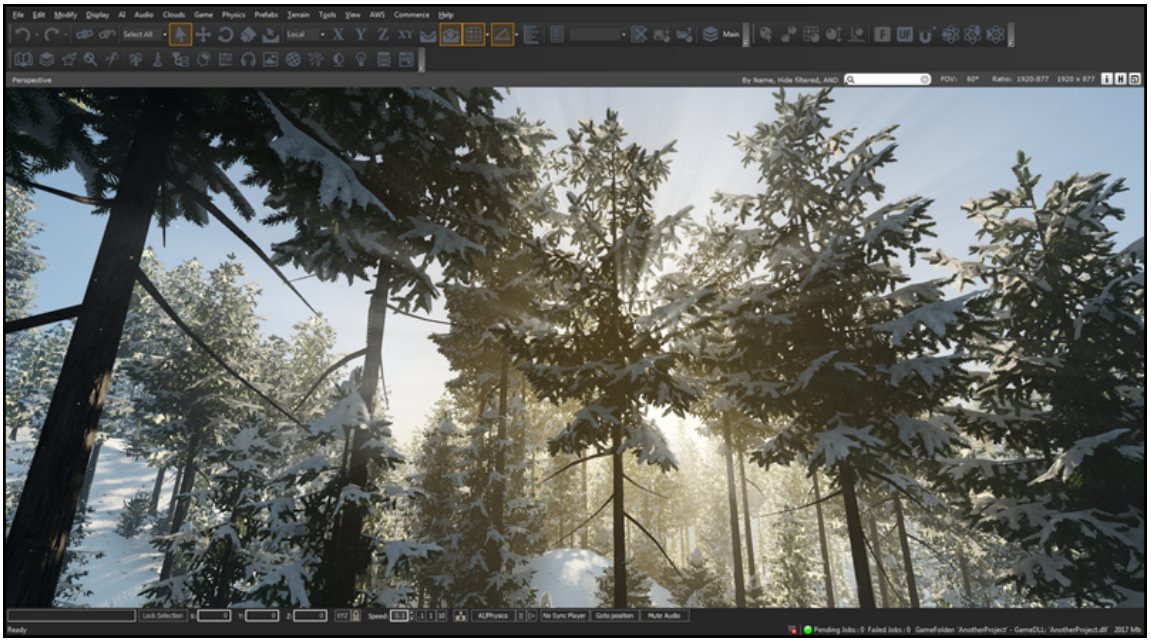
- Dynamic Content – Upload new or updated game assets to the cloud that can be automatically downloaded to game clients.
- In-Game Survey – Create surveys for your game, test them, and publish them. You can view the results of your surveys and manage them in the Cloud Gem Portal.
- Leaderboard – Store player high scores and provide leader boards that show player rankings.
- Message of the Day – Schedule the delivery of messages like new product announcements or holiday greetings to game clients.
- Player Account – Use a standalone player authentication and management solution for your game.
- Speech Recognition – Add speech recognition and natural language processing to your Lumberyard game. The Speech Recognition Cloud Gem uses Amazon Lex, which recognizes and understands spoken user input.
- Text to Speech Cloud Gem – Enhance your gameplay and workflows with synthesized speech. The cloud gem uses Amazon Polly, a text-to-speech service that turns text into lifelike speech.

For more information, see Cloud Gems in the *Amazon Lumberyard Developer Guide*.

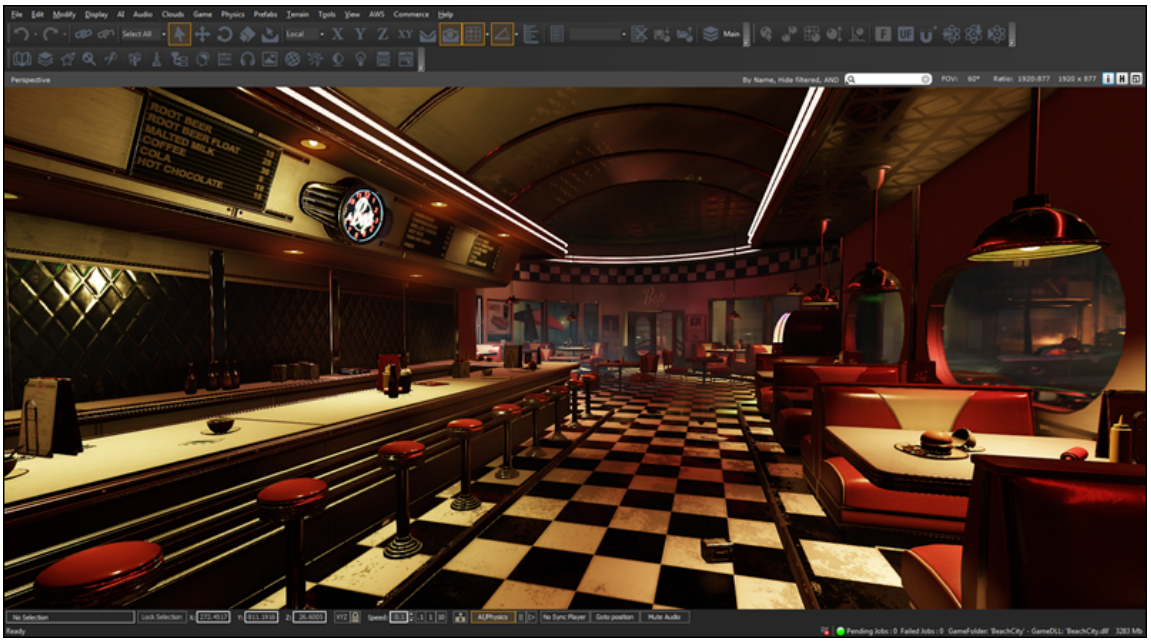
Legacy Sample Projects

The following sample projects and assets are available as a separate download from the Lumberyard Legacy Downloads page.

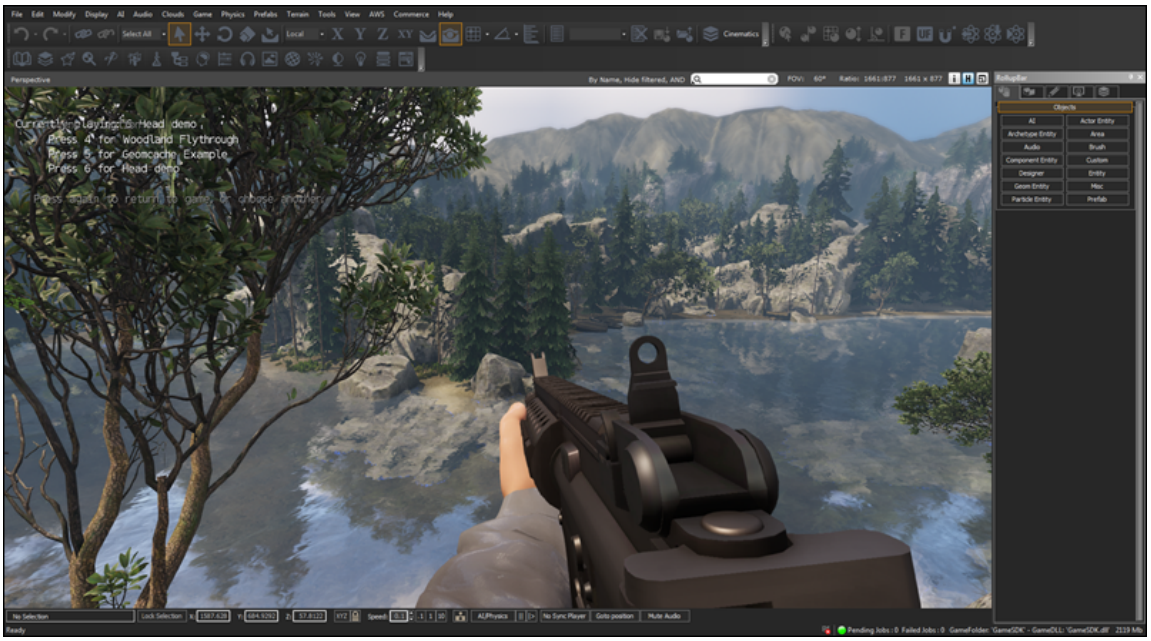
- **Woodland Asset Collection** – Free assets for you to use to create your levels.



- **Beach City Night Asset Collection** – Collection of free assets that you can use to try Lumberyard or make your own games.



- **Legacy Project (GameSDK)** – Enables you to use GameSDK functionality.



Topics

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 `\lumberyard_version\dev\SamplesProject\Levels` directory.

The **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.

> **Note**
> By default, Cloud Canvas functionality is not enabled for the Samples Project. In Project Configurator (p. 28), select the **CloudGemSamples** project instead, or enable the **Cloud Gem Framework** gem for the Samples Project and then rebuild (p. 1060) the Samples Project.

Getting Started Project

The `\GettingStartedFiles` directory contains a complete level with items used in each step of the *Lumberyard Basics* tutorial. Among these are terrain, lighting, cameras, objects, materials, flow graph scripting, and code. You can use these levels to skip over parts of the tutorial. The levels also show 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.

Audio Sample

This sample demonstrates how to use the **Audio Trigger** (p. 462) and **Audio Rtpc** (p. 460) components with Lua scripts to associate sounds of a door opening and closing.

This example is fully annotated within the Lua script of the level file. The following sounds are associated:

- Sounds
  - Door open
  - Door creak
  - Door creak stop
  - Door slam
- Rtpc
  - Creak volume
  - Creak pitch

To play the example, do the following:

- Press **W/S** to swing the door.
- Press **Q** to open the door.
- Press **E** to open the door.
- Press **spacebar** to open or close the door.

To see the Lua script, select the `Door` entity (a child of `DoorTest`) and then click the { } button next to the script property to open the Lua IDE.

For more information about audio, setting up sounds and using Wwise LTX, see Audio System (p. 181).

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 tag point, 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.

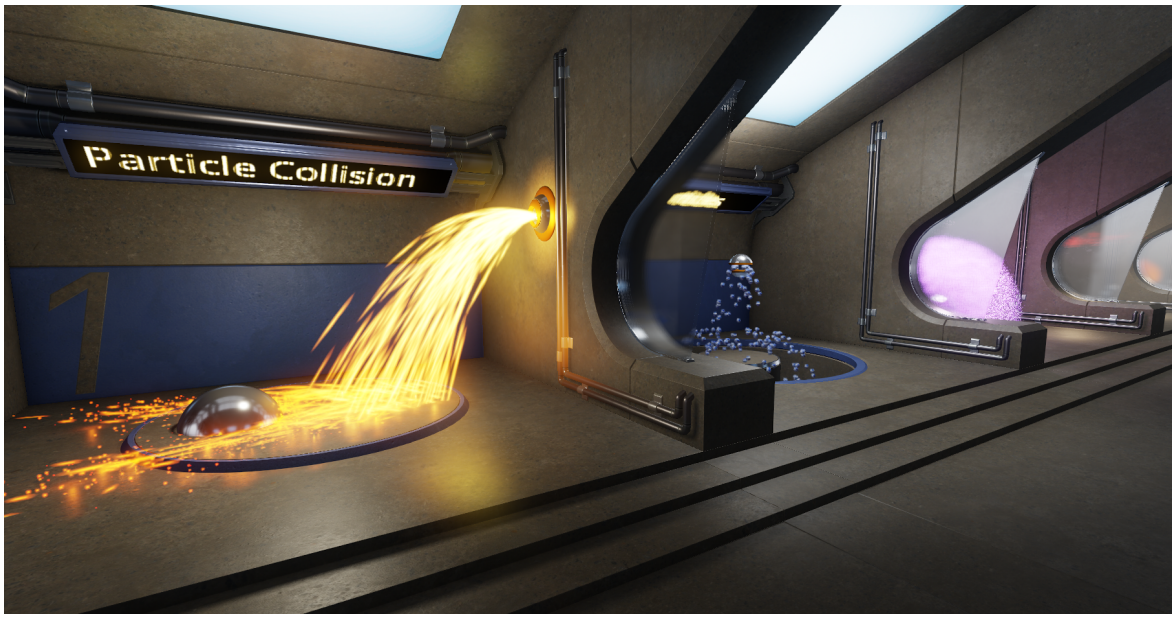
- **Example 11** – Uses keyboard keys **I**, **K**, **J**, and **L** to move a box around in the viewport. Shows a second entity moving toward the first one.

Use the following robot keyboard keys and mouse controls:

- To move forward, left, backward, and right, press **W**, **A**, **S**, and **D** keys, respectively.
- To look around, move the mouse.
- To jump, press the **spacebar**.

To activate the trigger and see an annotated explanation, drive the game robot to each numbered display in this sample.

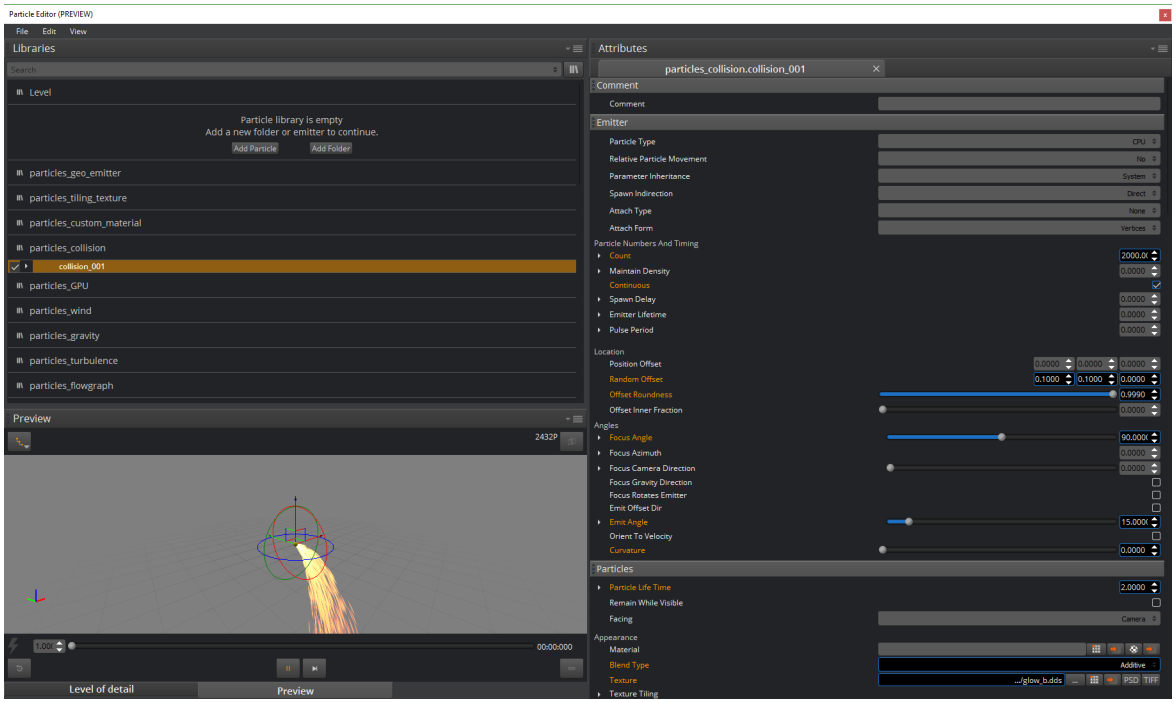
Particles Technical Sample



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 **Tools**, **Particle Editor**. The particle effects are located in the **Libraries** pane.



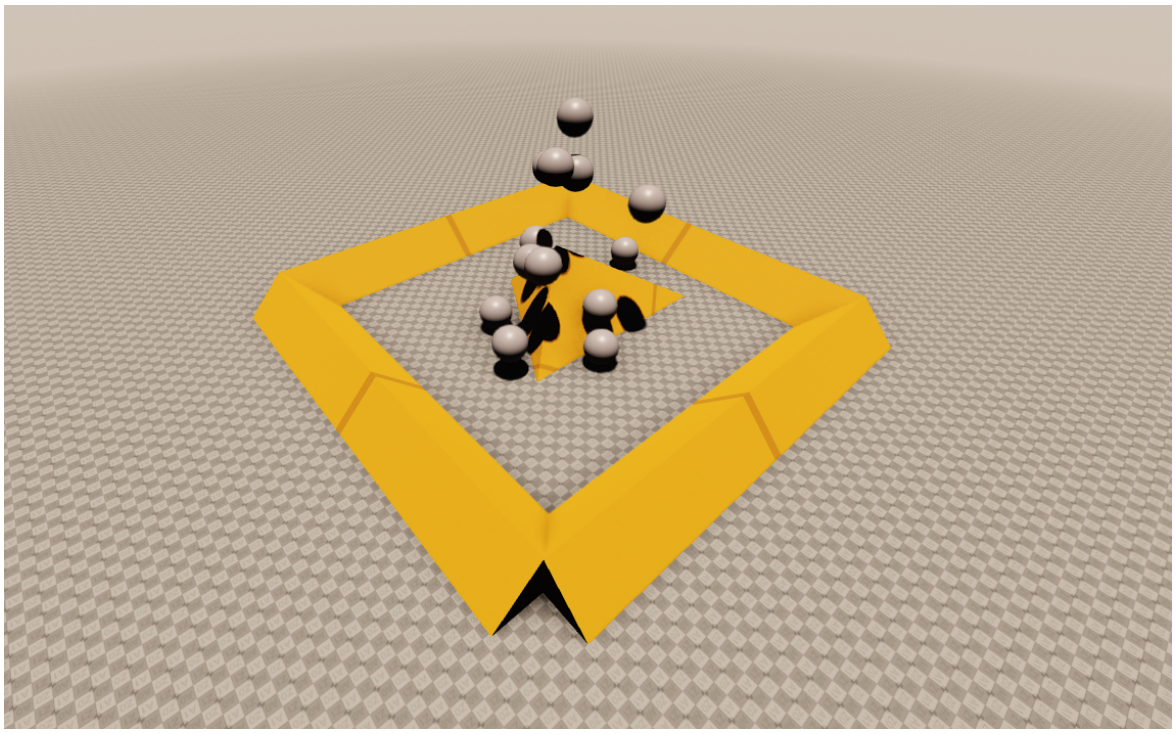
Once the particles technical sample level is open, in the **Perspective** viewport in Lumberyard Editor, use the following keyboard keys and mouse controls:

- To start the level, press **Ctrl+G**.
- To move forward, left, backward, and right, press the **W**, **A**, **S**, and **D** keys, respectively.
- To look around, move the mouse.
- To exit game mode, press **Esc**.

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** – Particles with **Gravity Entities** demonstrate particles that are manipulated by the physics entity called **GravitySphere**.
- **Station 5** – Particles with **Turbulence** demonstrates how the turbulence attributes are used to create a vortex effect.
- **Station 6** – Particles with **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** – Particles with **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.

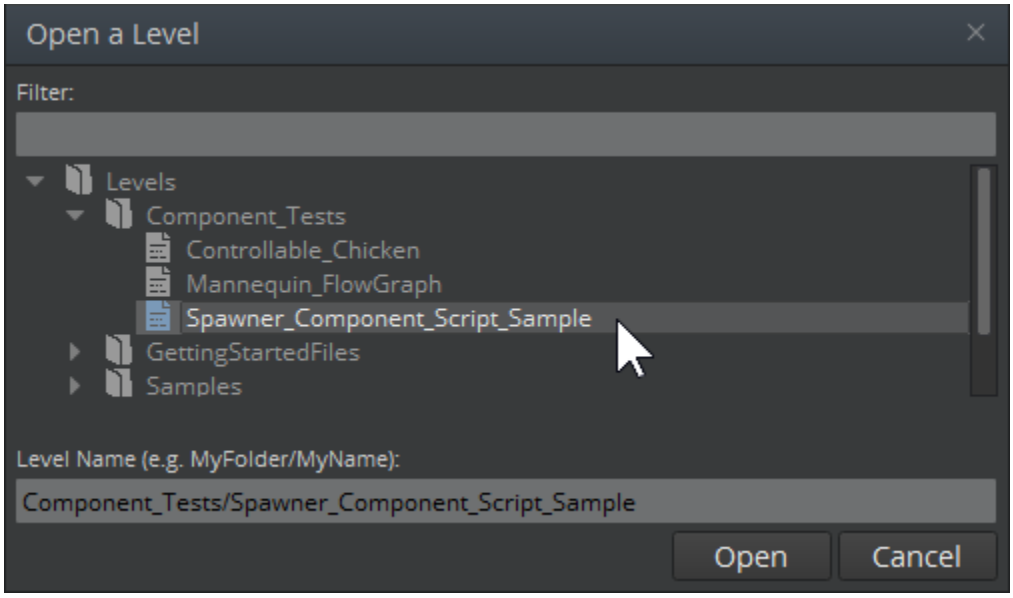
Spawner Component Script Sample



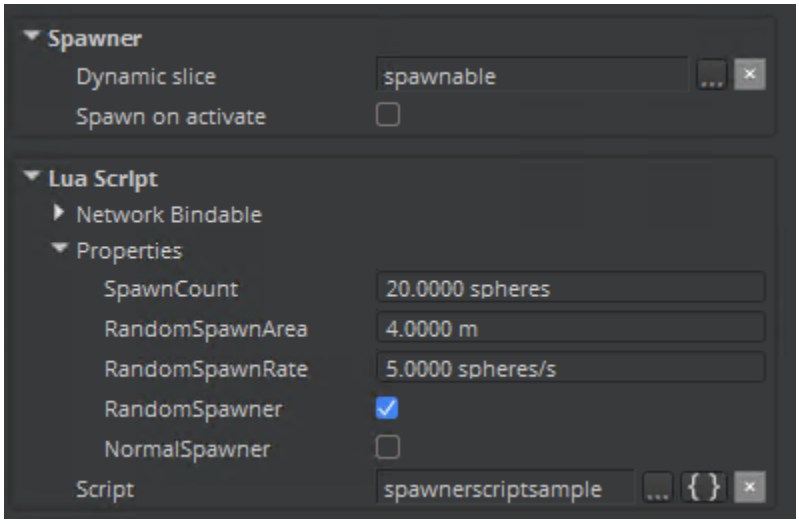
The spawner component script sample demonstrates how to set up a spawner component and drive it using a simple Lua script. It facilitates the spawning of a design-time selected or run-time provided dynamic slice (`.dynamicslice` file) at an entity's location with an optional offset. The sample, located in the `\dev\SamplesProject\Levels\Component_Tests\Spawner_Component_Script_Sample` directory, includes examples of all of the spawner component's events and notifications.

To run the spawner component sample

1. Open the sample level at `Levels\Component_Tests\Spawner_Component_Script_Sample`.



2. Press **Ctrl+G** to start the simulation.
3. The test runs automatically. No user input is expected, but when the spawner component is selected in the viewport, you can modify the simulation parameters in the **Spawner** and **Lua Script** sections of **Entity Inspector**.



Editor Properties

Name	Description	Default
Dynamic slice	The slice to spawn.	Null
Spawn on activate	Specifies whether the component spawns the selected slice upon activation.	False

4. Press **Esc** to stop the simulation.

Buses

The following table shows the EBus request events that are exposed to Lua for the spawner component.

SpawnerComponentRequestBus

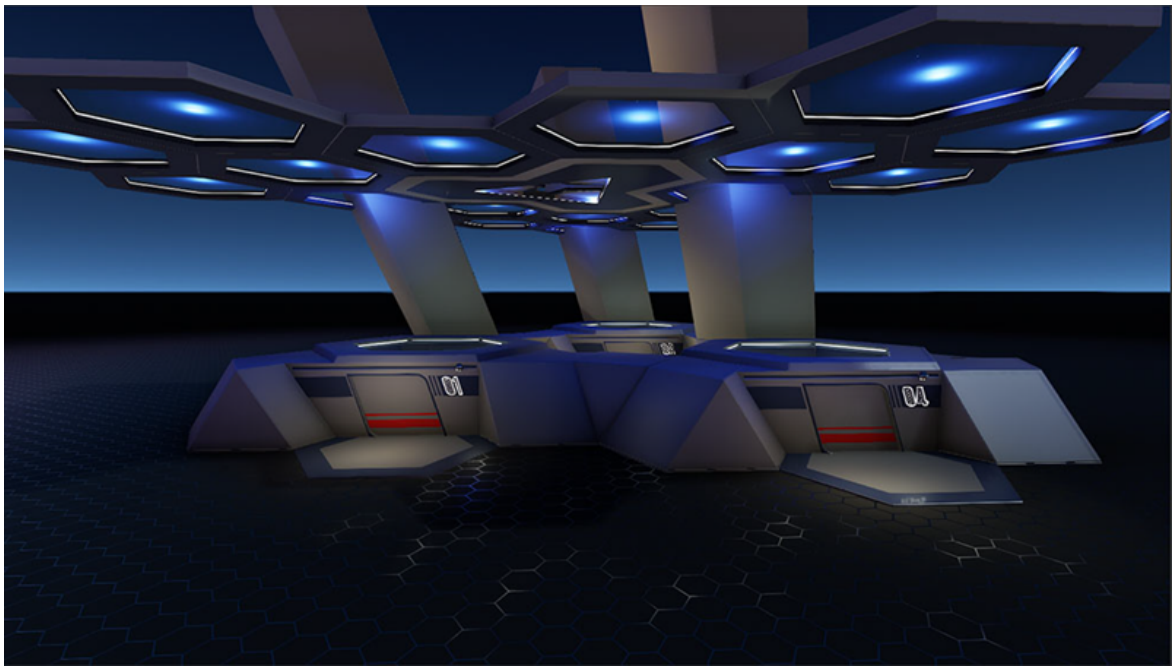
Function	Parameters	Return	Available to Script	Description
`Spawn`	(none)	`SliceInstantiationTicket`	Yes	Spawns the selected slice at the entity's location.
`SpawnRelative`	`Transform`	`SliceInstantiationTicket`	Yes	Spawns the selected slice at the entity's location with the provided relative offset.
`SpawnSlice`	`slice`	`SliceInstantiationTicket`	No	Spawns the provided slice at the entity's location.
`SpawnSliceRelative`	`slice`, `Transform`	`SliceInstantiationTicket`	No	Spawns the provided slice at the entity's location with the provided relative offset.

The following table shows the events for `SpawnerComponentNotificationBus`. The events have no return values.

SpawnerComponentNotificationBus

Function	Parameters	Availabl to Script	Description
`OnSpawnBegin`	`SliceInstantiationTicket`	Yes	Notification that the spawn process for a slice has begun.
`OnSpawnEnd`	`SliceInstantiationTicket`	Yes	Notification that the spawn process for a slice has finished.
`OnEntitySpawned`	`SliceInstantiationTicket`, *`<EntityID>`*	Yes	Notification that the specified entity has been spawned.
`OnEntitiesSpawned`	`SliceInstantiationTicket`, `vector`*`<EntityID>`*	No	Notification that the specified entities have been spawned.

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:

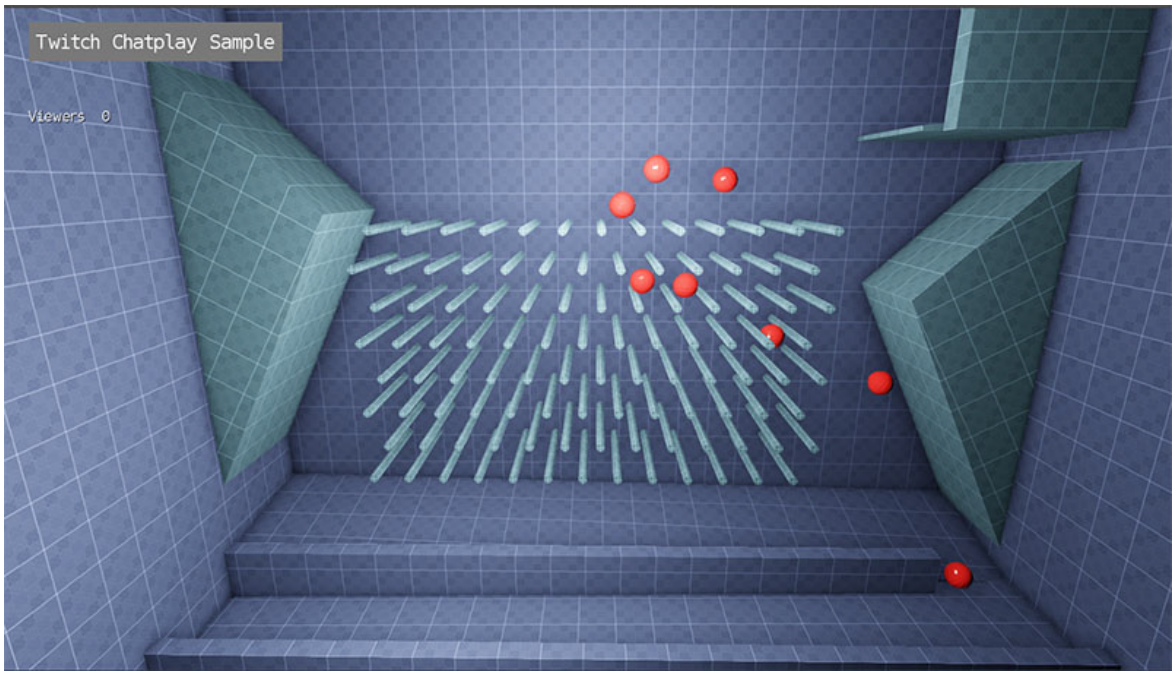
- To move forward, left, backward, and right, press the **W**, **A**, **S**, and **D** keys, respectively.
- To look around, move the mouse.
- To jump, press **Spacebar**.

To activate the trigger and see an annotated explanation, drive the game robot to each numbered door in this sample .

UIEditor Sample

This sample demonstrates how to create a basic main menu using the UI Editor. For more information, see UI System (p. 1539).

Twitch Chat Basics



Located in the `\TwitchChatBasics` directory, this sample demonstrates Twitch ChatPlayfeatures. For example, it shows 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. When they do, 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. 1532).

Also included is a debug script with which you can manually push the number of users up to a set number. This can help you verify that the count of users works when no users are available for testing.

Starter Game Sample

You can use the Starter Game sample to see how Lumberyard systems work together to make a game. Starter Game is a small, third-person game that is built with the Lumberyard component entity system. In addition to component entities, Starter Game demonstrates bipedal locomotion, voxel-based global illumination, the time of day system, and more. In this sample, you play as Jack, a robot that has crashed on a distant planet. Jack can explore the world and must defend himself against enemy robots that occupy the planet. You can use Jack or any other assets in the Starter Game sample to prototype your own projects.

You can obtain the Starter Game sample from the following sources:

- Lumberyard package (Lumberyard 1.11 and newer)

  The Starter Game sample is included in the engine package.
- Lumberyard Downloads page

  The Starter Game sample download includes the latest features and improvements that are not yet integrated into the sample that's included in the engine package. An example is the Starter Game Action Update.
- Legacy Downloads page

  The download includes the Starter Game sample that is supported in the previous version of Lumberyard. For example, if the newest version is Lumberyard 1.11, this download is supported in Lumberyard 1.10.

  Note
  Currently, only Windows supports the Starter Game sample.

If you have been making your own changes to Starter Game and don't want to lose your work when upgrading from Lumberyard 1.9 to a newer release, see Upgrading Your Starter Game Project (p. 41).

To install the Starter Game sample (if not yet installed)

1. Download the `StarterGame.zip` package from the Downloads page and extract it to your Lumberyard directory. For example, `\Lumberyard\1.x.0.0`.
2. Use the desktop icon to launch the **Project Configurator**.
3. In the **Project Configurator**, select **Starter Game**.
4. Click **Set as default**. Close the **Project Configurator**.
5. Start **Lumberyard Editor**.
6. In the **Welcome to Lumberyard Editor** dialog box, click **Open level**.
7. In the **Open a Level** dialog box, select **StarterGame** and click **Open**.
8. To start the game, press **Ctrl+G**. Use the following keyboard keys and mouse controls:

   - To move forward, strafe left, move backward, and strafe right, press the **W**, **A**, **S**, and **D** keys, respectively.
   - To look around, move the mouse.
   - To shoot, click the primary mouse button.
   - To toggle weapon selection, click the secondary mouse button.
   - To toggle time of day, press the **F** key.
   - To exit the game, press **Esc**.

   If you are running Starter Game in the editor and Jack the robot dies, press **Esc** to exit the game.

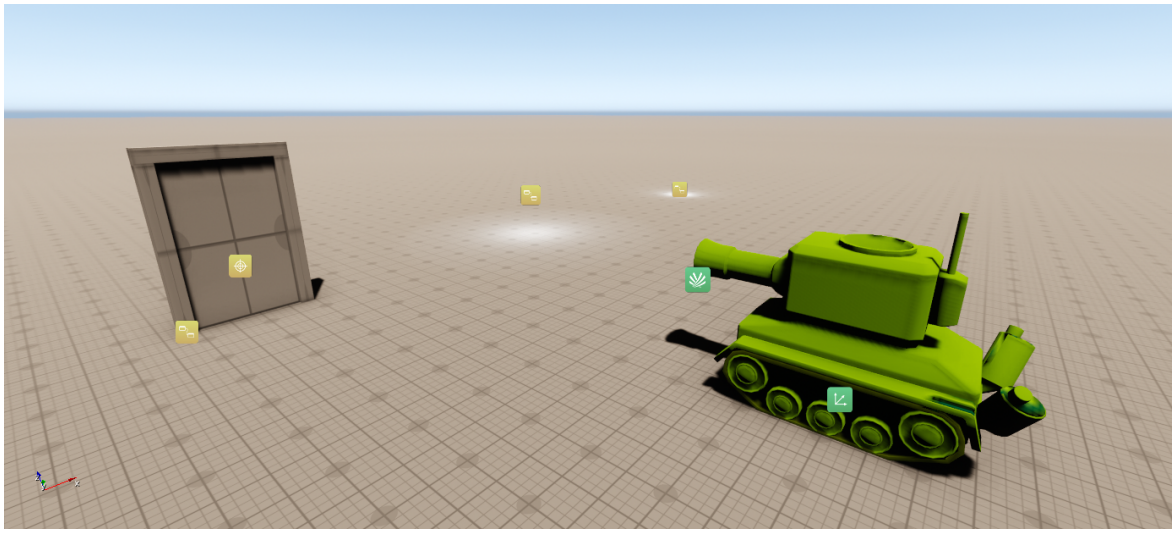
   Note
   The viewport window displays debugging information by default when you are in gameplay mode (**Ctrl+G**). You can toggle this information on or off by pressing the tilde (**~**) key. For more information, see Using Console Debug Views (p. 1527).

Script Canvas Basic Sample

Script Canvas is in preview release and is subject to change.

The Script Canvas Basic Sample contains a collection of five Script Canvas scripts that demonstrate the following gameplay functions:

- Basic player controller
- Ballistic projectile
- Door that opens based on a trigger volume
- Light that toggles based on a trigger volume
- Flickering light



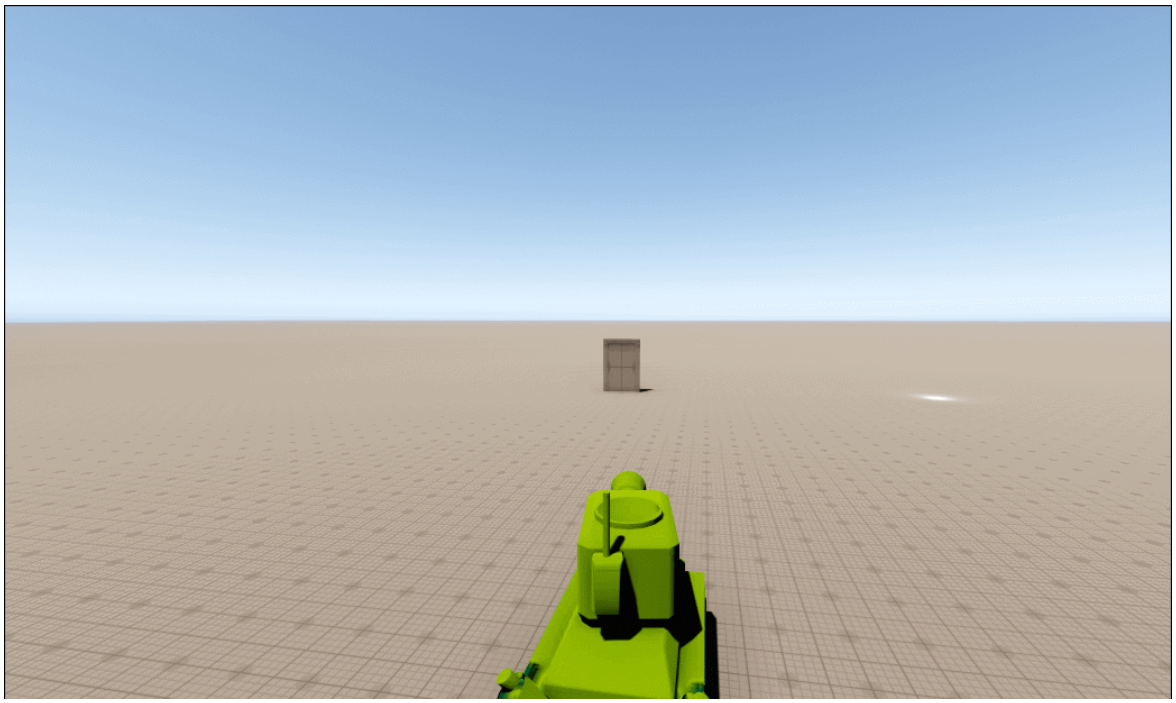
To see the Script Canvas graphs

1. In Lumberyard Editor, choose **Tools**, **Script Canvas**.
2. In **Script Canvas**, choose **File**, **Open**.
3. Navigate to the `lumberyard_version\dev\SamplesProject\Levels\Samples\ScriptCanvas_Sample\ScriptCanvas_Basic_Sample\ScriptCanvas` directory.
4. Open one of the Script Canvas scripts:
   - `doortriggerarea.scriptcanvas`
   - `flickerlight.scriptcanvas`
   - `lumbertankcontrols.scriptcanvas`
   - `projectile.scriptcanvas`
   - `togglelight.scriptcanvas`

To play the game

Do the following:

- To start the level, press **Ctrl+G**.
- To control the tank, do the following:
  - To move forward and backward, press the **W** and **S** keys, respectively.
  - To turn left and right, press the **A** and **D** keys, respectively.
  - To control the camera, move the mouse pointer.
  - To fire the tank's gun, use the left mouse button.
- To open the door, drive up to it or shoot it.
- To turn off the light, drive up to the light.
- To turn the light on, drive away from the light.
- To exit the level, press **Esc**.



Advanced_RinLocomotion Sample

Animation Editor is in preview release and is subject to change.

This sample level showcases the Rin character in a simplified environment and demonstrates how the **Animation Editor** uses actors, motion sets, and animation graphs to control character behavior. Playable animations include idle, walk, run, turn, and multiple attacks.

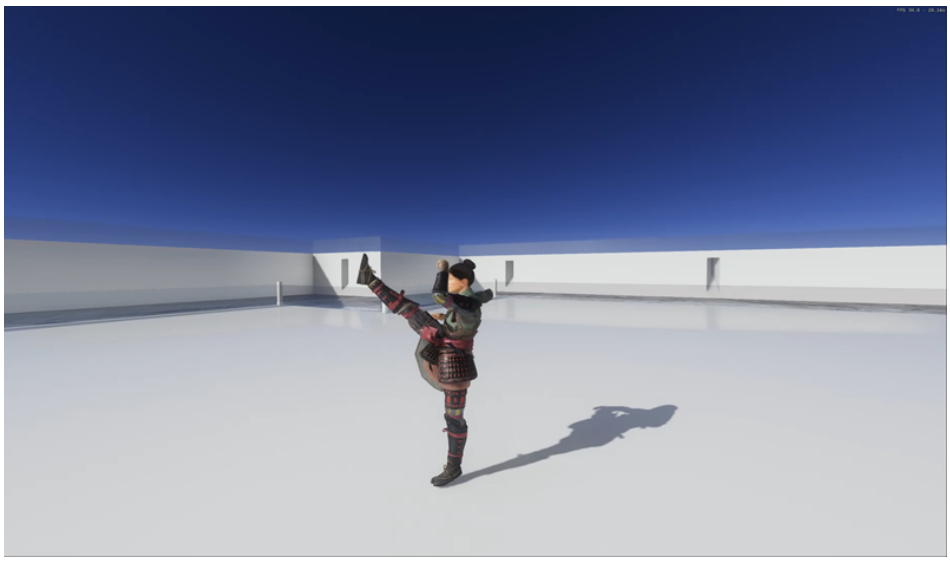
The Advanced_RinLocomotion sample level is located in the `\dev\SamplesProject\AnimationSamples` directory. The assets are located in the `\dev\SamplesProject\AnimationSamples\Advanced_RinLocomotion` directory.

Once the sample level is open, in the **Perspective** viewport in Lumberyard Editor, use the following keyboard keys and mouse controls:

- To start the level, press **Ctrl+G**.
- To move forward, left, backward, and right, press the **W**, **A**, **S**, and **D** keys, respectively.
- To look around, move the pointer.
- To attack, press the left mouse button.
- To jump, press the **Space** key.
- To exit game mode, press **Esc**.

You can also use an Xbox game controller connected to your computer:

- To control the movement direction, use the left joystick.
- To control the camera navigation around the character, use the right joystick.
- To attack, press **X**.
- To jump or dive roll, press **A**.
- To zoom in the camera, press the right trigger.
- To zoom out the camera, press the left trigger.
- To snap the camera to the back of the character, press the right joystick.



Simple_JackLocomotion Sample

Animation Editor is in preview release and is subject to change.

This sample level showcases Jack the robot in a basic environment and demonstrates how the **Animation Editor** uses actors, motion sets, and animation graphs to control character behavior. Playable animations include idle, walk, run, and navigate.

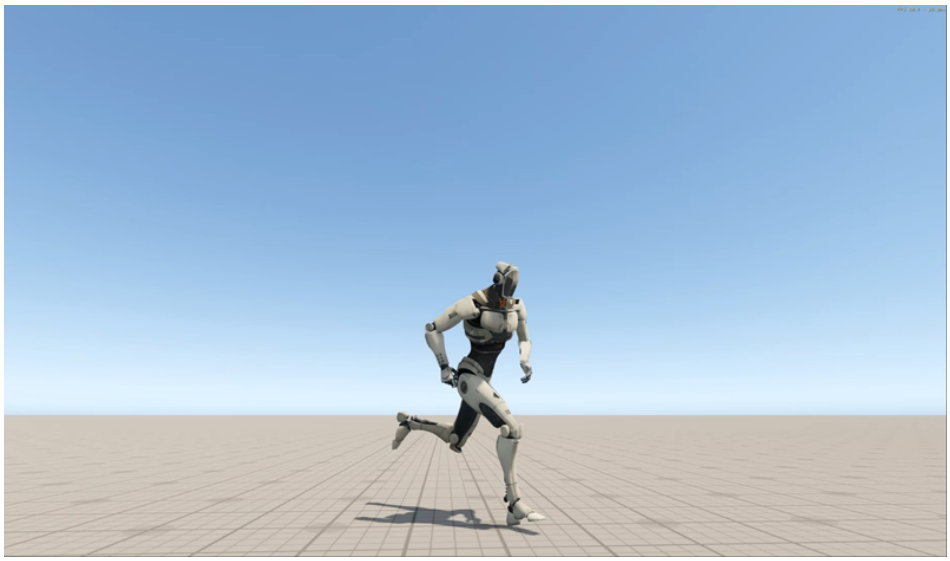
The Simple_JackLocomotion sample level is located in the `\dev\SamplesProject\AnimationSamples` directory. The assets are located in the `\dev\SamplesProject\AnimationSamples\Simple_JackLocomotion` directory.

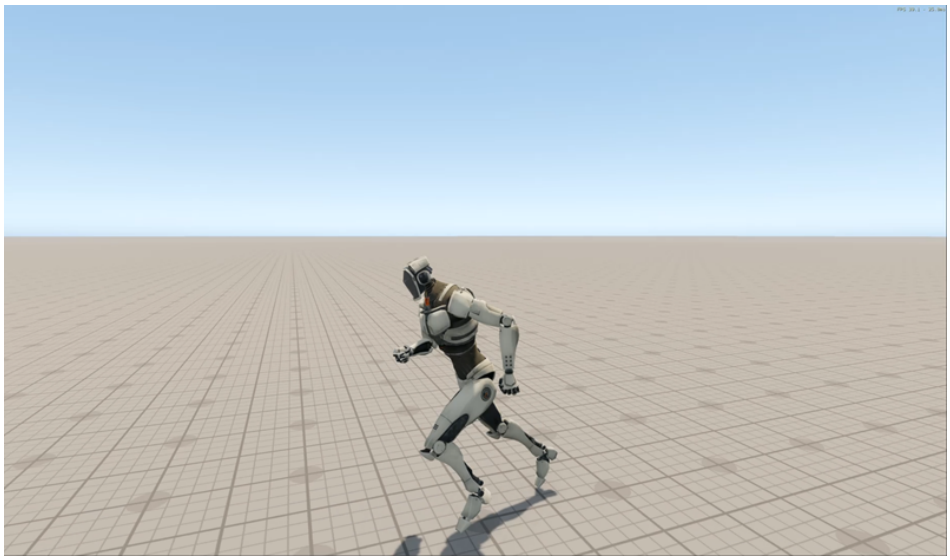
Once the sample level is open, in the **Perspective** viewport in Lumberyard Editor, use the following keyboard keys and mouse controls:

- To start the level, press **Ctrl+G**.
- To move forward, left, backward, and right, press the **W**, **A**, **S**, and **D** keys, respectively.
- To look around, move the pointer.
- To toggle the **LookAt** node on and off during the idle animation, press **Tab**.
- To aim, hold the left mouse button.
- To exit game mode, press **Esc**.

You can also use an Xbox game controller connected to your computer:

- To control the movement direction, use the left joystick.
- To control the camera navigation around the character, use the right joystick.
- To toggle the character aim on and off, press the right trigger.
- To toggle the **LookAt** node on and off during the idle animation, press the left trigger.





Multiplayer Sample

The Multiplayer sample shows you how to build and structure multiplayer games that use the various features of the GridMate networking library. The new preview attempts to incorporate as many GridMate features as possible but is not comprehensive. Instead it is intended as a living snapshot of the current state of the networking library. Consequently, as the sample evolves, features might be added to enhance functionality or removed to keep the sample concise.

Building and Running the Sample

Follow these steps to build and run the sample.

To build and run the Multiplayer sample

1. In Lumberyard Project Configurator set **MultiplayerSample** as the active project.
2. Run the following command:

```
lmbr_waf configure
```

3. Run one of the following commands, which builds everything in the profile.
   - If you are using Visual Studio 2015, run this command:

```
lmbr_waf.bat build_win_x64_vs2015_profile -p all
```

   - If you are using Visual Studio 2013, run this command:

```
lmbr_waf.bat build_win_x64_vs2013_profile -p all
```

4. Run the `MultiplayerSampleLauncher.exe` file from your default bin directory.
   - For Visual Studio 13, use `Bin64vc120.Dedicated`.
   - For Visual Studio 15, use `Bin64vc140.Dedicated`.

MultiplayerLobby

This level demonstrates a multiplayer lobby that uses GridMate networking. Currently, the level performs the following tasks:

- Creates a server
- Displays a list of servers on the local LAN
- Connects to a server

To create a server

1. Type the server name in the **Create a Server** form.
2. Click **Create Server**.

   The game starts hosting and loads the selected map.

To connect to a server

1. Find your server in the server browser list. If your server doesn't appear, click **Refresh**.
2. To select your server, click the row that contains your server name.
3. Click **Connect**.

Game Overview

The Multiplayer sample is an arcade space shooter. Each player controls a ship in a giant field of asteroids. Each player must fly through the asteroids and destroy as many of them as possible in an effort to achieve the highest score. Additional points are awarded when other players are killed. Points are deducted every time the player dies. In addition to the threats posed by asteroids and other players, gravity wells scattered around the map can draw players in and trap them. Players who become trapped in a gravity well die instantly.

When a ship is spawned, three kinds of ships are assigned randomly:

- **Fighter** – An average space ship.
- **Speeder** – A faster and more maneuverable space ship but less sturdy as a result.
- **Rammer** – A sturdier ship that is weaponless but designed to smash into asteroids and destroy them. The ship is fast but has poor maneuverability.

When destroyed, some asteroids spawn collectibles that award a short-time power-up bonus. These power-ups include extra damage against certain targets, a longer power-up time, and extra protection against certain sources of damage.

Game Controls

The following tables list command controls for keyboards and Xbox controllers.

Keyboard

Key	Action
WASD keys	Move the ship relative to the screen (**W** is always up, **S** is always down, and so forth).

Key	Action
Arrow left, arrow right	Rotate the ship clockwise or counterclockwise.
Spacebar	Fire the ship's weapon (if applicable).
Tab (hold)	Open the **Player Stats** window.

Xbox Controller

Control	Action
Left joystick	Move the ship in absolute terms relative to the screen.
Right joystick	Rotate the ship clockwise or counterclockwise.
Right shoulder button	Fire the ship's weapon (if applicable).
Back button (hold)	Open the **Player Stats** window.

PS4 Controller

Control	Action
Left joystick	Move the ship in absolute terms relative to the screen.
Right joystick	Rotate the ship clockwise or counterclockwise.
R2	Fire the ship's weapon (if applicable).
Options button (hold)	Open the **Player Stats** window.

Touch Controls

Control	Action
Virtual joystick (left side of screen, no visual indication)	Move the ship in absolute terms relative to the screen.
Virtual joystick (right side of screen, no visual indication)	Rotate the ship clockwise or counterclockwise.

Notes

- As soon as you begin moving with either virtual joystick, the ship begins firing (if applicable).
- These controls can also be accessed by using the mouse (click to touch).
- If you fire manually, the automatic firing stops until the next input from one of the virtual joysticks.

Game CVars

The following console variables can be used with the game.

`mps_AISteeringDebug`

Specifies whether the debug drawing information for the bot is drawn.

`mps_ControllerType <number>`

Controls the type of `PlayerController` that the client uses to control the assigned ship. The following table shows the possible values for *`<number>`*:

Value	Description
`0`	Specifies manual control. A user controls a single ship.
`1`	A bot that plays the game.

`mps_DrawBucketSystem`

Specifies whether the debug information for the bucket system is drawn.

`mps_SpawnTag`

Controls what ship is spawned for a player when the player spawns a ship. Possible values are shown in the following table.

Value	Description
`Random`	Chooses an available ship at random. This is the default value.
`Fighter`	Spawns a fighter ship.
`Speeder`	Spawns a speeder ship.
`Rammer`	Spawns a rammer ship.

`mps_UserName`

Specifies the name for the client in game elements such as the scoreboard or action notification.

Creating a Dedicated Server

To create a dedicated server for the game, perform the following steps.

To create a dedicated server

1. In Lumberyard Project Configurator set **MultiplayerSample** as the active project.
2. Run the following command:

```
lmbr_waf configure
```

3. Run one of the following commands, which builds everything in the profile.

   If you are using Visual Studio 2015, run this command:

```
lmbr_waf build_win_x64_vs2015_profile -p all
```

If you are using Visual Studio 2013, run this command:

```
lmbr_waf build_win_x64_vs2013_profile -p all
```

4. Run one of the following commands, which builds `game_and_engine` in `profile_dedicated`.

   For Visual Studio 2015, run the following command:

   ```
   lmbr_waf build_win_x64_vs2015_profile_dedicated -p game_and_engine
   ```

   For Visual Studio 2013, run the following command:

   ```
   lmbr_waf build_win_x64_vs2013_profile_dedicated -p game_and_engine
   ```

5. Run the `MultiplayerSampleLauncher_Server.exe` file from your default bin directory.
   - For Visual Studio 13, use `Bin64vc120.Dedicated`.
   - For Visual Studio 15, use `Bin64vc140.Dedicated`.

Supported Platforms

The Multiplayer sample is currently supported on a variety of client and server platforms.

Clients

The following playable client platforms are supported:

- Windows (Visual Studio 2013, 2015)
- Android
- iOS

Dedicated Server Platforms

The following dedicated server platforms are supported:

- Windows (Visual Studio 2013, 2015)
- Linux (Ubuntu)

About Self-Signed Certificates in the Multiplayer Sample

In Windows, the Multiplayer sample enables client-side self authentication that uses a self-signed certificate, which the build script generates for each user. Regarding this certificate, keep in mind the following points:

- The self-signed certificate is not an official certificate. If you want to ship a publically available game, you must replace it with a public key certificate.
- On the macOS and Linux platforms, you must generate self-signed certificates manually.
- On the macOS and Linux platforms, the Multiplayer sample has the following limitations:
  - macOS: Only iOS client builds can be made.

- Linux: Only dedicated server builds can be made.

Troubleshooting

The following are some connection troubleshooting tips.

- If clients are unable to discover a server, ensure that the server and client are on the same subnet in order to allow for UDP discovery.
- If clients are unable to connect to a server, ensure that the server and client are using the same key and certificate files.
- When trying to host or join a session, you might receive one of the following error messages:

```
Invalid Secure Socket configuration given for hosting a session.
Ensure that a Public and Private key are being supplied.
```

```
Invalid Secure Socket configuration given for joining an encrypted session.
Ensure that a Certificate Authority is being supplied.
```

  To resolve these issues, perform one of the following tasks:

  - Run the command `lmbr_waf configure` to generate a set of certificates.
  - In the `MultiplayerSample/certificates` directory, provide your own appropriately named certificates.

  For more information about self-signed certificates, see **Encryption** in the Multiplayer Sample Network Features (p. 1495) section and About Self-Signed Certificates in the Multiplayer Sample (p. 1491).

Topics

Multiplayer Sample Gameplay Architecture

The Multiplayer sample consists of five main entities and an equal number of supporting gameplay systems. The following overview provides a conceptual introduction to the architecture of the Multiplayer sample.

Principal Entities in the Multiplayer Sample

The principal entities in the Multiplayer sample are ships, asteroids, gravity wells, bullets, and collectibles. These entities also have some common components.

Common Components

The principal entities in the game share the following components:

- `SimpleSpacePhysicsComponent` – This component enables objects to move around in space and be affected by gravity.
- `CollisionSystemComponent` – These components enable objects to interact with each other in the world.
- `NetBindingComponent` – Activates the net binding code of other components on an entity and signals the game that the entity should be replicated across the network.

Ships

The Multiplayer sample uses a variety of ships that demonstrate how a selectable game state can be synchronized from a client to a session when the player first joins the game. Different types of ships also provide a variety of game experiences. Some ships have long range weapons, and others are short range and have no weapons. The following components are related to ships:

- `ShipComponent` – Identifies an object as a ship to Lumberyard. It also takes commands from the player and converts them on the host into the motion required to move the ship.
- `ShipGunComponent` – Handles the firing of bullets.
- `ActiveModifiersComponent` – Manages the modifiers that affect the ship.
- `CollectibleInterpreterComponent` – Enables a layer of interpretation between a collectible and the actual modifier that is granted. This enables implementations of a collectible according to the ship type. For example, ships with guns that receive a collectible get a special laser, while weaponless ships get a nonweapons-related buff.
- `HealthComponent` – Determines the relative health of the ship as long as it remains operational.

Asteroids

Asteroids are the most common slice in the Multiplayer sample. They exist in three sizes (small, medium, and large). These asteroid types are functionally the same while they exist. When destroyed, they react differently by spawning varying amounts of asteroids and collectibles. Blanketing the playfield with these objects allows for optimization techniques to be implemented that create a reasonable visual experience without overusing bandwidth. The following components are related to asteroids:

- `AsteroidComponent` – - Identifies an object as an asteroid to the game engine.
- `DeathActionComponent` – Allows for actions to be taken upon the destruction of the object. In the sample, the `DeathActionComponent` spawns additional asteroids and collectibles.
- `DamageComponent` – Determines how much damage the asteroid does when it collides with a ship.
- `HealthComponent` – Determines how much damage an asteroid can take before it is destroyed.

Gravity Wells

Gravity wells create action inside the game by causing objects to be destroyed and spawned even in the absence of user interaction. This prevents the game state from falling into stasis. The following components are related to gravity wells:

- `GravityWellComponent` – Identifies an object as a gravity well.
- `GravitySourceComponent` – Applies gravity to physical objects and draws them towards the source object.

Bullets

Ships use bullets to damage asteroids and each other. The use of bullets drives gameplay and generates points. The following components are related to bullets.

- `BulletComponent` – Identifies an object as a bullet.
- `DamageComponent` – Determines how much damage a bullet does when it hits a ship or asteroid.

Collectibles

Collectibles add fun and excitement to the game and provide a way to enable and test dynamic in-game changes to otherwise static systems. The sample uses collectibles to test the handling of constantly

shifting objects to find the right balance between natural appearance and controllability. The following component is related to collectibles.

- `CollectibleComponent` – Identifies an object as a collectible.

Multiplayer Sample Gameplay Systems

The gameplay systems in the Multiplayer sample handle the game logic essential to game operations.

Game Manager

The `GameManagerComponent` handles all of the general game logic, such as controlling game start and end, managing the results screens, and other tasks not performed by individual game modes. The `GameManagerComponent` functions as a server-only component. A component like this would normally be reserved for use in a dedicated server and not be shipped in a player version of the game. The component synchronizes no data and is not replicated across the network.

Score Attack Game Mode

The `ScoreAttackGameModeComponent` works with the `GameManagerComponent` to provide the rules for play and synchronize data to players for score attack mode.

Asteroid Manager

The `AsteroidManagerComponent` creates new asteroids and manages asteroid lifespans. It decides where, how, and how many asteroids are spawned into the game. Although it is exported with the level, this component synchronizes no data and should not be included in a player version of the game.

Collectible Manager

The `CollectibleManagerComponent` spawns collectibles in response to the destruction of asteroids. Although it is exported with the level, this component synchronizes no data and should not be included in a player version of the game.

Spawn Manager

The `SpawnManagerComponent` is a wrapper for the Spawner component that allows the sample to spawn objects in complex ways without requiring the use of multiple spawner components. The spawn manager facilitates the automation of variation among spawned entities. For example, when an asteroid is spawned, a general type can be specified and the specific type randomly selected in the spawn manager from the supported types. This removes the need to manually specify a type with each spawn request.

This approach also promotes ease of maintenance because it enables a single object to handle all objects of a single type. For example, all asteroids are defined in a single place and are identified by a specific tag (`SmallAsteroid`, `MediumAsteroid`, or `LargeAsteroid`). This limits the number of interactions required among the objects and the amount of information that must be exchanged.

In the sample, only one object manager (for example, asteroid or collectible) is aware of the location of the spawner for an object. This location is usually on the object itself.

Multiplayer Sample Network Architecture

The sample uses a server-authoritative architecture. In the sample, client-side input is processed on the player and converted into RPCs that are processed by the server.

Player Object

When each client joins the session, it creates an object on the server that represents the player. The object provides information relevant to player configuration. The client maintains full control over player configuration, which the server reads from the object. The player object consists of a `GamePlayerComponent` and a related `GamePlayerChunk`.

- `GamePlayerComponent` – The `GamePlayerComponent` represents the player on the server. The component is owned by the client and replicated to the server. This component handles local player configuration such as the player's name and the ship to spawn. The user can change this information during gameplay. These changes must be reflected to the server.
- `GamePlayerChunk` – The replica chunk that represents player information on the server. The `GamePlayerChunk` specifies the display name of the player and the ship that the player wants to spawn.

Ship Object

The ship object consists of a `ShipComponent` and a related `ShipComponentReplicaChunk`.

- `ShipComponent` – Manages the overall ship logic, controls, and ship configurations.
- `ShipComponentReplicaChunk` – Contains the RPCs that are invoked on the client and passed to the server. The server then validates, sanitizes, and applies the results to the game state.

See Also

For more information about networking in the Multiplayer sample, see Multiplayer Sample Network Features (p. 1495).

Multiplayer Sample Network Features

The Multiplayer sample offers the following features.

- **Dedicated Server Split (Partial)** – Because the clients can also host game sessions for LAN play, a true split is never done even when the client has no server code. However, the code itself is compartmentalized in such a way that the split can be done easily. A dedicated server split is used in the following component:
  - `GameManagerComponent` – Designed to be available only on the server. It relies on the `GameModeBus` to synchronize the state to display (for example, whether the game is ready to start a round, or whether the current round is complete). For details, see the source code files in the `\dev\Code\MultiplayerSample\Modules\MultiplayerSampleModule\Source\Components\GameLogic` directory.
- **Client Authoritative Control** – Enables the client to maintain exclusive control of an object whose actions are processed on the server. Client authoritative control is used in the following components:
  - `ShipComponent` – Client commands that control a ship trigger a series of RPCs that the `ShipComponent` exposes. The RPCs are then processed on the server. The commands are restricted to the owner of the object. For details, see the source code files in the `\dev\Code\MultiplayerSample\Modules\MultiplayerSampleModule\Source\Components\Ship` directory.
  - `ManualPlayerControllerComponent` – Represents the human-controllable player. For details, see the source code files in the `\dev\Code\MultiplayerSample\Modules\MultiplayerSampleModule\Source\Components\ShipController` directory.
  - `BaseAutomatedPlayerControllerComponent` – Represents an AI-controllable player. It maps computer-generated input to the RPC calls. For details, see the source code files in the `\dev`

`\Code\MultiplayerSample\Modules\MultiplayerSampleModule\Source\Components\ShipController` directory.

- **Encryption** – Encryption is performed using `FileDataSource` with self-signed certificates and certificate pinning. The files expected are a set of X.509 PEM files, which are split into a nonpassword-protected key (`multiplayersample.key.pem`) and a cert file (`multiplayer.cert.pem`). These are located in the `MultiplayerSample\certificates` directory. If either of these files is missing, a set of certificates is generated using RSA2048 and some default answers that can be customized from the `wscript` file. For more information, see How To Generate a Private Key and Public Certificate. For important information about self-signed certificates in the Multiplayer sample, see About Self-Signed Certificates in the Multiplayer Sample (p. 1491). In the Multiplayer sample, encryption is used in the following features:
  - `GameManager` – Exposes a function called `SetupEncryption`, which handles the configuration of certificates for the game. For details, see the source code files in the `\dev\Code\MultiplayerSample\Game\Game` directory.
  - `MultiplayerUtils` (in Multiplayer Gem) – Has utility functions that handle configuration of the carrier description for hosting and joining. The code shows how to configure the SecureSocketDriver and use an EBus event to signal its use for the connection. For details, see the source code files in the `\dev\Gems\Multiplayer\Code\Include\Multiplayer` directory.
- **RPC Traits** – In Lumberyard, RPCs allow games to send events or requests to remote nodes through GridMate replicas that synchronize the state of the session. In the Multiplayer sample, RPC traits are used to control ships, manage the HUD state, and manage audio controllers. To see how they are attached to RPCs, see the source code files in the following locations.

Component	Location
`ShipComponent`	`\dev\Code\MultiplayerSample\Modules\MultiplayerSampleModule\Source\Components\Ship`
`CoreGameHUDController`	`\dev\Code\MultiplayerSample\Modules\MultiplayerSampleModule\Source\Components\GameLogic`
`ShipAudioController`	`\dev\Code\MultiplayerSample\Modules\MultiplayerSampleModule\Source\Components\Audio`

- **Custom Component Net Binding** – For a Lumberyard component to share data on the network, it must include the NetBindingComponent. A number of Multiplayer sample components demonstrate custom component net binding.
  - `CoreGameHUDController` – An illustrative example that has datasets, RPCs, RPC traits, and callbacks. For details, see the source code files in the `\dev\Code\MultiplayerSample\Modules\MultiplayerSampleModule\Source\Components\GameLogic` directory.
- **Custom Type Marshalling** – In GridMate, all data marshalling, whether for a dataset or an RPC, is written using a specialized `Marshaler` type. If the type is a complex type like a class or container, then that marshaler marshals each of its fields with nested marshalers. Custom marshalers can be implemented to support custom types. Custom type marshalling is used in the following component:
  - `ScoreAttackGameModeComponent` – The `PlayerRoundInfo` struct demonstrates the simplest way of defining a custom marshaler for a custom type without needing to specifically override it in the dataset definition. For details, see the source code files in the `\dev\Code\MultiplayerSample\Modules\MultiplayerSampleModule\Source\Components\GameLogic` directory.

VR Islands Level

The VR Islands level showcases the potential of a simple VR level started from instantVR. This level features floating islands between which a player can teleport.

Lumberyard has customized instantVR with new controllers that have physics collision on them.

A player can teleport around the islands using the trigger button on the Oculus or Vive motion controllers. Assuming the teleport location is valid, holding the trigger picks the location, and releasing it initiates the teleport.

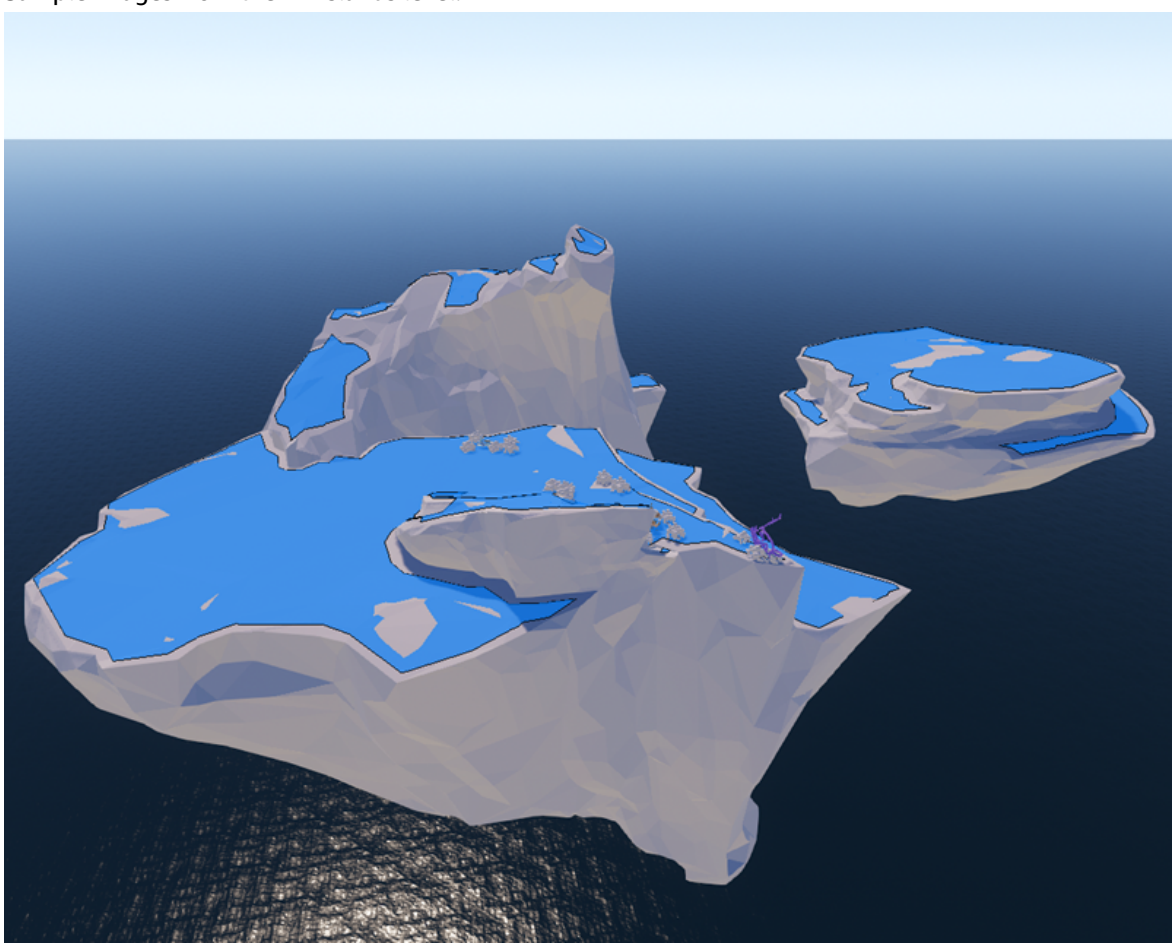
The motion controllers, visualized as spheres, can push the blobby vegetation around.

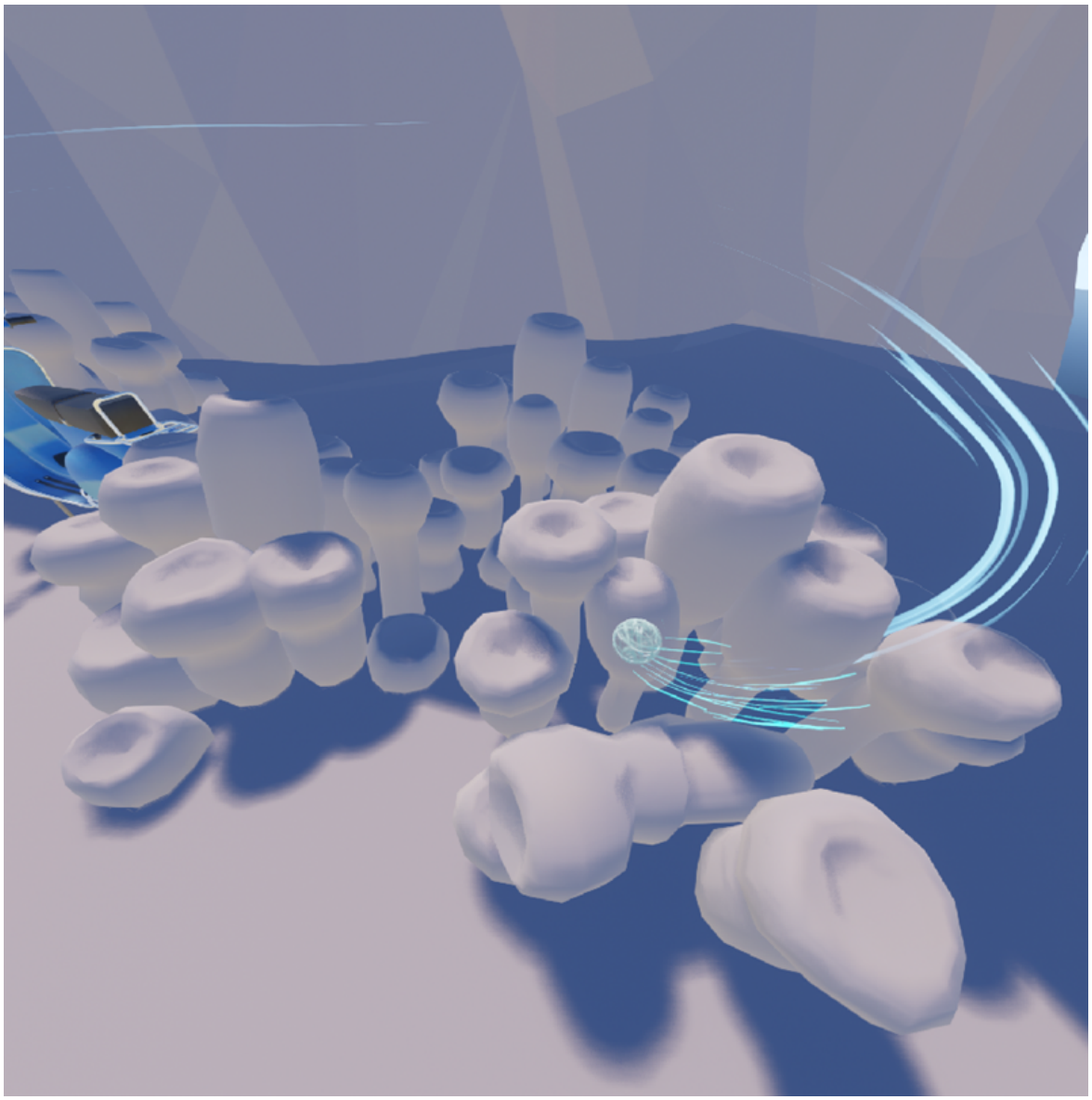
To install the VR Islands level

1. Download the `vrproject.zip` package at Lumberyard Downloads and extract it in your Lumberyard directory. For example, `\Lumberyard\1.x.0.0`
2. Open the **Lumberyard Setup Assistant** (p. 20). On the **Summary** page, click **Project Configurator**.

   Note
   To ensure that the VR Islands level launches, you must use **Lumberyard Setup Assistant** to open **Project Configurator**. **Lumberyard Setup Assistant** copies required SDKs from the `3rdParty` directory into the `\dev\VirtualRealityProject` directory.
3. In the Project Configurator, select **VirtualRealityProject**.
4. Click **Set as default**. Close the **Project Configurator**.
5. Open **Lumberyard Setup Assistant**. On the **Summary** page, click **Launch editor**.
6. Open the level named **VR_Islands_Sample**.

Sample images from the VR Islands level:

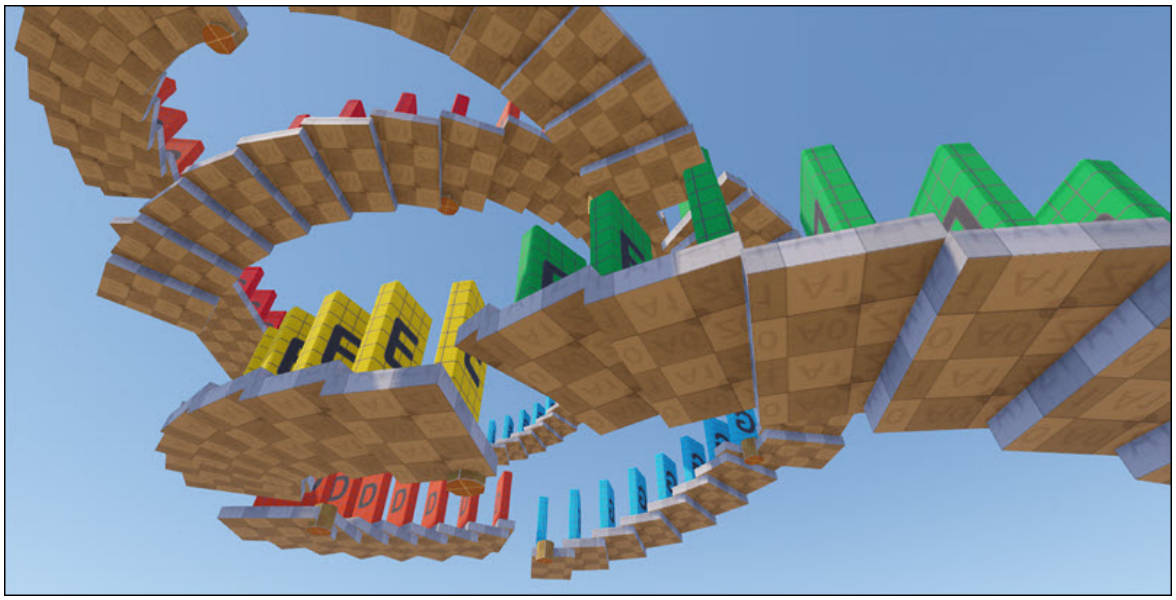




Virtual Reality Project Sample

The Virtual Reality Project sample is a template that you can use to build virtual reality (VR) applications for any supported device. The sample is configured with the base set of Gems (HMD Framework, Oculus, OpenVR, and OSVR) that you need for VR and demonstrates the following essential VR features:

- Room-scale VR setup
- Tracked controller input system
- Spatialized audio playback
- Stereo 360 video playback



To install the Virtual Reality Project sample

1. Download the `vrproject.zip` package at Lumberyard Downloads and extract it in your Lumberyard directory. For example, `\Lumberyard\1.x.0.0`.
2. Open the **Lumberyard Setup Assistant** (p. 20). On the **Summary** page, click **Project Configurator**.

   Note
   To ensure that the VR project launches, you must use **Setup Assistant** to open **Project Configurator**. **Lumberyard Setup Assistant** copies required SDKs from the `3rdParty` directory into the `dev\VirtualRealityProject` directory.
3. In the **Project Configurator**, select **VirtualRealityProject**.
4. Click **Set as default**. Close the **Project Configurator**.
5. Open **Lumberyard Setup Assistant**. On the **Summary** page, click **Launch editor**.

Note
The video playback on the TV in the **VR_TVRoom_Sample** level must be enabled before you can view video playback. For instructions on settting up video playback, see Setting up Video Playback (p. 584). After completing video playback setup, you must also rebuild (p. 1060) the virtual reality project.

In-App Purchases Sample

The In-App Purchases sample demonstrates how to use the In-App Purchases (IAP) Gem for a Lumberyard application that runs on a mobile device. For more information about this gem, see In-App Purchases Gem (p. 1096).

Topics

Prerequisites

To use the In-App Purchases sample, you must have the following:

- An understanding of how to build a Lumberyard application and deploy to a mobile device. For more information, see Mobile Support (p. 1211).
- Access to iTunes Connect to properly configure the sample for an iOS device.
- Access to the Google Play Developer Console to properly configure the sample for an Android device.

To load the In-App Purchases sample, you must edit the `autoexec.cfg` file (located in the `\dev\SamplesProject` directory) to include the following: `map Gems_InAppPurchases_Sample`

You must also update the product IDs in the `product_ids.json` file to match the product IDs in iTunes Connect or the Google Play Developer Console. This file is located in the `\dev\SamplesProject\IAP_ProductIds` directory.

Testing the In-App Purchases Sample on Android

You can test the In-App Purchase samples on an Android device.

To test the In-App Purchases sample on an Android device

1. In a command line window, build the SamplesProject application.
2. Go to the Google Play Developer Console.
3. In the left pane, click **All Applications**.
4. In the right pane, click **Create Application**.
5. Follow the steps to create a new application. The package name for your application must match the package name under `android_settings` in the `project.json` file. This file is located in the `\dev\SamplesProject` directory.
6. Once the application is created, it appears on the **All Applications** tab in the Google Play Developer Console. Click the application. In the left pane, navigate to **In-app Products**.
7. On the **In-app Products** page, click **Add new product** to add products that can be purchased through the application.
8. In the left pane of the application page, click **APK** to upload the APK that you built in step 1.
9. Create a closed beta test and add test accounts for testing.
10. Deploy the application to your device. The in-app products that you created in the Google Play Developer Console appear when you select **Available Products** in the application.

You can now buy products, view purchased products, and consume purchased products in the application.

Testing the In-App Purchases Sample on iOS

You can test the In-App Purchases sample on an iOS device.

To test the In-App Purchases sample on an iOS device

1. Go to iTunes Connect.

   You must have an admin account to create applications.
2. Click **My Apps**.

3. In the top left of the page, click **+** to create a new application.

   The bundle identifier must match the bundle identifier for your product in Xcode. To find the bundle identifier, select your Lumberyard project (for example, SamplesProject) in Xcode. You can also find the bundle identifier in the `Info.plist` file (located in the `\Code\SamplesProject\Resources\IOSLauncher` directory).
4. On the **My Apps** page, click your newly created application.
5. In the top pane of the application page, click **Features**.
6. In the left pane of the **Features** page, click **In-App Purchases**.
7. On the **In-App Purchases** page, click **+** to add products that can be purchased through the application.
8. When you finish adding products for in-app purchases, go to the iTunes Connect home page.
9. In iTunes Connect, click **Users and Roles**.
10. On the **Users and Roles** page, click **Sandbox Testers**.
11. On the **Sandbox Testers** page, click **+** to add sandbox tester accounts.

    Important
    Do not log in using this account on your device. This will deactivate the sandbox testing account and you'll need to create a new account.
12. On your device, choose **Settings**, **iTunes**. Log out of any accounts that are logged in.
13. Deploy and run the application from Xcode. When prompted, sign in with your sandbox tester account on the device.

You can now view available products, buy products, view purchased products, and more. When you buy a product, a note appears that indicates you are in the sandbox environment and will not be charged for the purchase.

Woodland Asset Package – Legacy

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.

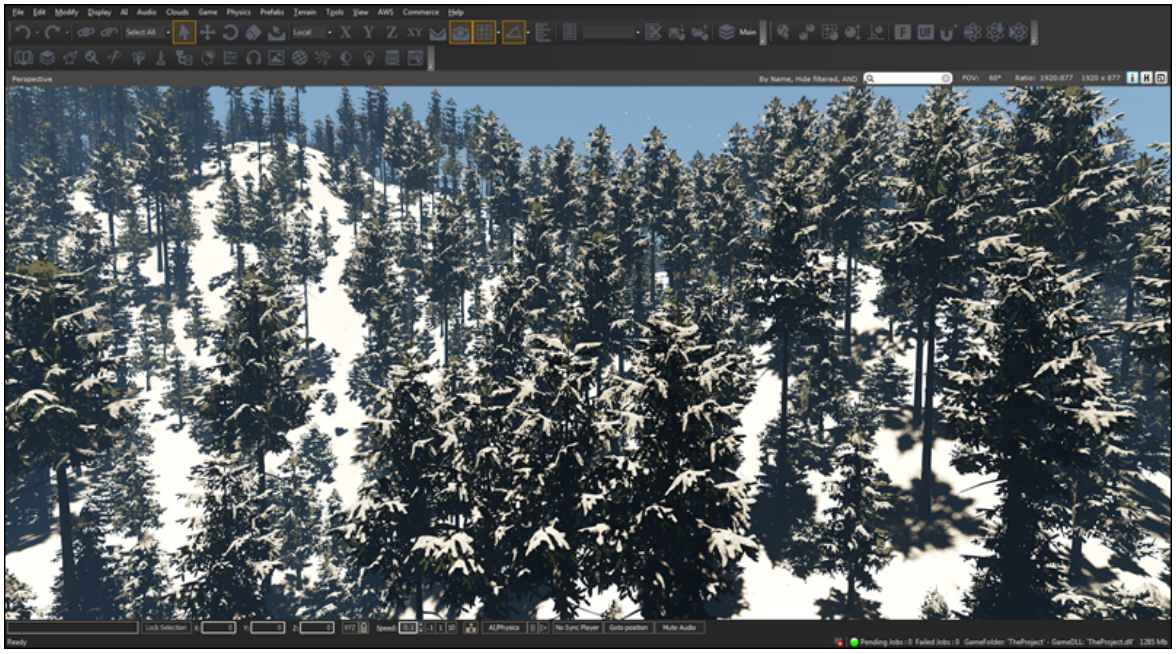
Note
This package is compatible with Lumberyard 1.9 and earlier.

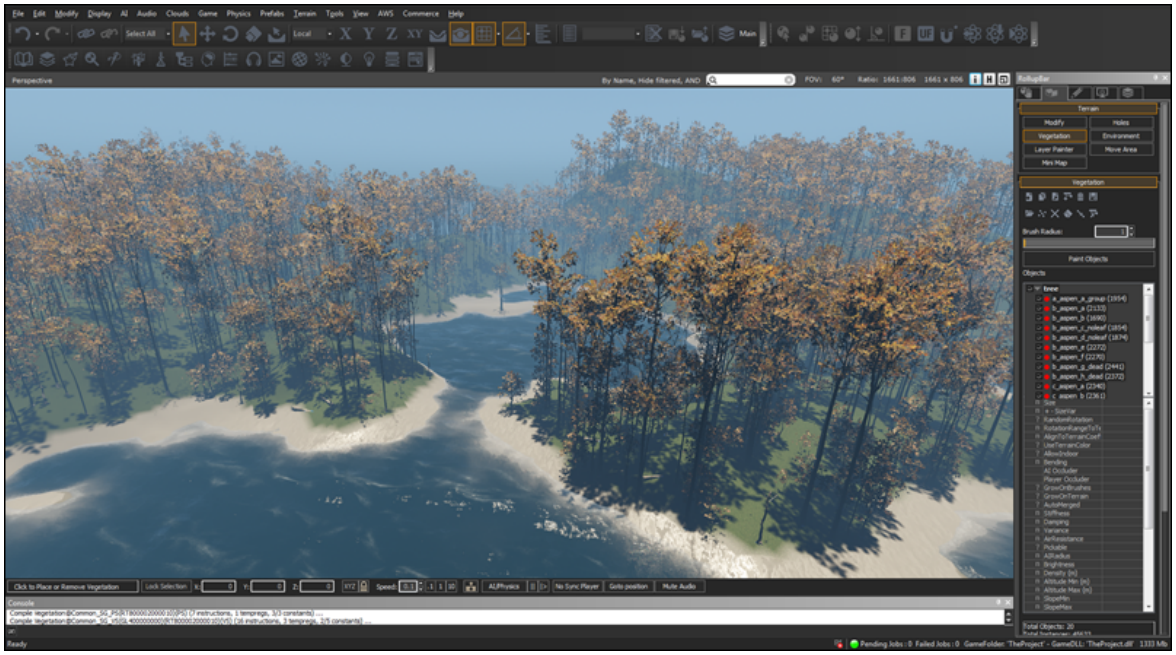
To install the Woodland asset package

1. Download the Woodland Asset Package at Lumberyard Previous Versions 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.
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 **Tools**, **Material Editor** and select your materials and textures.

- Click **Tools**, **Geppetto** and select your animations.

Sample images from the Woodland asset collection:





Beach City Sample Project – Legacy

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.

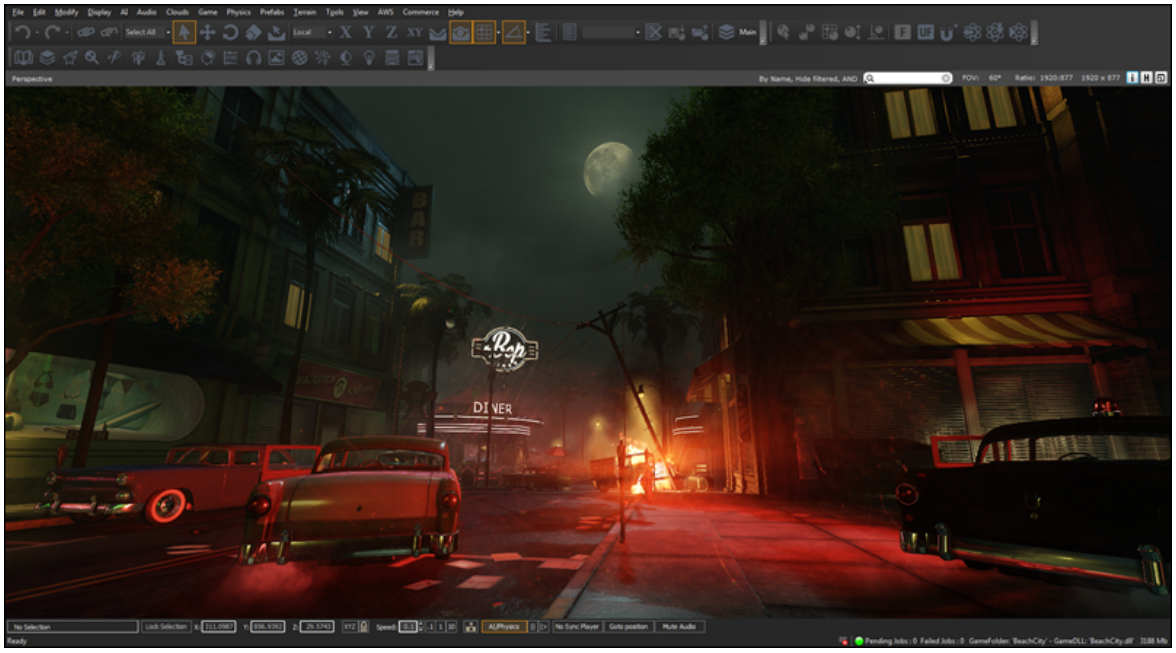
Note
This package is compatible with Lumberyard 1.9 and earlier.

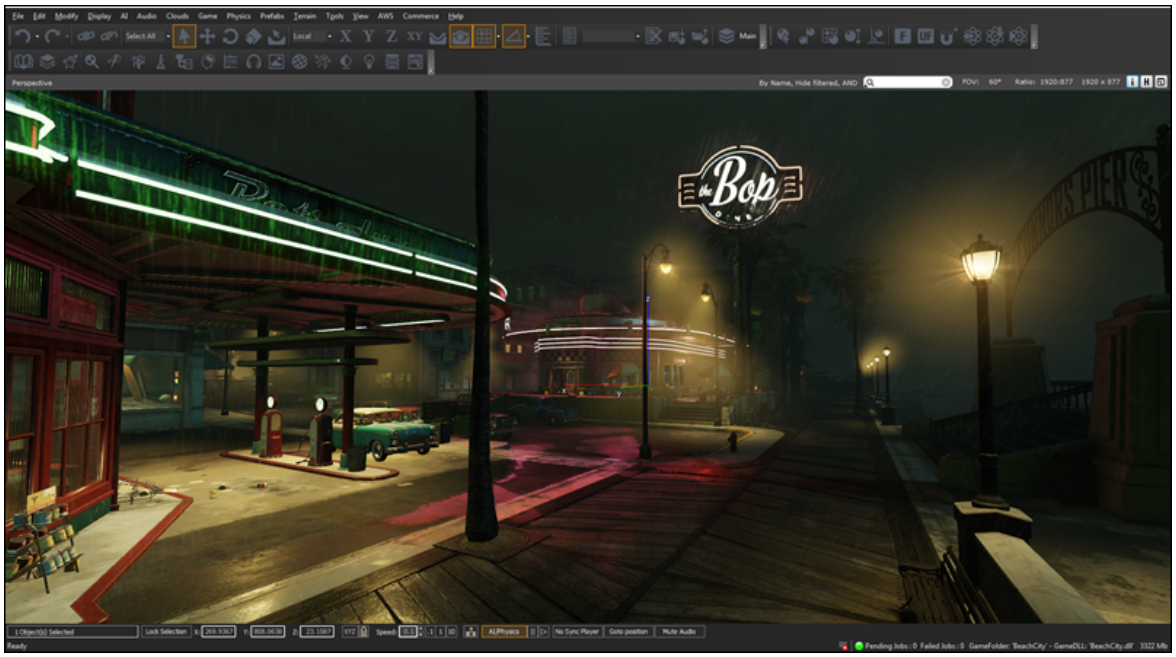
To install the Beach City sample project

1. Download the `BeachCity.zip` package at Lumberyard Previous Versions and extract it in your Lumberyard directory. For example, `\Lumberyard\1.x.0.0`.
2. Open the **Lumberyard Setup Assistant** (p. 20). On the **Summary** page, click **Project Configurator**.

   Note
   To ensure that the Beach City project launches, you must use **Lumberyard Setup Assistant** to open **Project Configurator**. **Lumberyard Setup Assistant** copies required SDKs from the `3rdParty` directory into the `dev\BeachCity` directory.
3. In the Project Configurator, select **BeachCity**.
4. Click **Set as default**. Close the **Project Configurator**.
5. Open **Lumberyard Setup Assistant**. On the **Summary** page, click **Launch editor**.

Sample images from the Beach City sample project:





Legacy Sample Project (GameSDK)

Lumberyard provides a package called Legacy Game Sample that demonstrates how to use Lumberyard features such as CryAction and CryAISystem. The fully functional game implements the necessary pieces that allow a game to communicate with some of Lumberyard's low-level systems. If you are familiar with CryEngine, the legacy game sample is a repurposed version of the GameSDK project.

You can use the legacy game sample to evaluate changes you make to the engine core and ensure your changes do not have unintended side effects. The legacy game sample is intended to be used as a reference for how to communicate with various systems of the Lumberyard engine. We do not recommend using the legacy game sample as the starting point of a new game.

> **Note**
> The legacy game sample is Windows only and is not supported on console or mobile platforms. This sample is compatible with Lumberyard 1.9 and earlier.

To download and access GameSDK

1. Download the `GameSDK.zip` package at Lumberyard Previous Versions and extract it in your Lumberyard directory. For example, `\Lumberyard\1.x.0.0`.
2. Open the **Lumberyard Setup Assistant** (p. 20). On the **Summary** page, click **Project Configurator**.

   > **Note**
   > To ensure that the GameSDK project launches, you must use **Lumberyard Setup Assistant** to open **Project Configurator**. **Lumberyard Setup Assistant** copies required SDKs from the `3rdParty` directory into the `dev\GameSDK` directory.

3. In the Project Configurator, select **GameSDK**.
4. Click **Set as default**. Close the **Project Configurator**.
5. Rebuild (p. 1060) the GameSDK project.
6. Open **Lumberyard Setup Assistant**. On the **Summary** page, click **Launch editor**.

Note
Audiokinetic Wave Works Interactive Sound Engine (Wwise) version 2014.1.14 or later is required to access audio for this project.

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

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. It also provides flexibility for developers to focus on testing the parts of Lumberyard that they care about.

The AZ test scanner has two components:

- An AZ test runner executable that loads libraries to test and capture 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. 1797) 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 \Bin64vc140.Debug.Test
 folder.
lmbr_waf.bat build_win_x64_vs2015_debug_test -p all

// Build with tests using profile configuration. Outputs to the \Bin64vc140.Test folder.
lmbr_waf.bat build_win_x64_vs2015_profile_test -p all
```

> **Note**
> Only Windows debug and profile builds are supported for testing. Other platforms are not supported. Release builds are not supported either.

For more information on writing tests, see Writing Tests for AzTest (p. 1511).

Running Unit and Integration Test Builds

A completed test build includes the file `AzTestRunner.exe` in the `\Bin64vc140.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 `lmbr_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 `lmbr_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 scan your entire test build:

```
// Scan entire test build and run all found tests
lmbr_test.cmd scan --dir Bin64vc140.Debug.Test
```

> **Note**
> The default scan tests libraries only. 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
lmbr_test.cmd scan --dir Bin64vc140.Debug.Test --only CrySystem.dll --integ
```

> **Note**
> For best results, run integration tests on a single library or use a whitelist. Scanning the full build might take hours to complete.

Option	Required?	Description
`--dir, -d`	Yes	The directory to scan for tests.
`--runner-path`	No	Path to the AZ test runner executable (the default is to look in the directory specified by `--dir`).
`--add-path`	No	Adds path to system path before running tests; used for resolving library or executable dependencies.
`--output-path`	No	Sets the path for output folder prefix (the default is `\dev\TestResults`).

Option	Required?	Description
`--integ, -i`	No	If set, runs integration tests instead of unit tests.
`--no-timestamp`	No	If set, removes the time stamp from output files.
`--wait-for-debugger`	No	If set, tells the AZ test runner executable to wait for a debugger to be attached before running tests.
`--bootstrap-config`	No	Path to a JSON configuration file for bootstrapping applications required by libraries.
`--limit, -n`	No	Sets a limit for the maximum number of modules to scan.
`--only, -o`	No	Sets a filter to run tests on only the specified library or executable name.
`--whitelist-file`	No	Path to a new line-delimited file used for whitelisting. The new line-delimited file allows for regular expressions when matching.
`--blacklist-file`	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.
`--exe`	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
lmbr_test.cmd scan --dir Bin64vc140.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` calls 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_vs2015_debug_test -p all run_tests
```

The `run_tests` command automatically points to the `\Bin64vc140.Debug.Test` folder to scan. It also uses the `all` option for whitelisting. The build step does not require the use of `run_tests`; it always matches the last build. 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="--no-timestamp"
```

You can also use the `--target` flag to build and test just one module:

```
lmbr_waf.bat build_win_x64_vs2015_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 test. Both whitelisting and blacklisting are off by default unless the scanner finds a default whitelist or blacklist file, respectively. In all cases, modules that are blacklisted are never tested even if they are included in the whitelist.

Both whitelisting and blacklisting use a new line-delimited text file for defining what modules to scan. 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 Bin64vc140.Test --whitelist-file my_whitelist.txt --blacklist-file
 my_blacklist.txt
```

You can specify as many whitelists or blacklists as you need. The patterns in each file are combined into one set. For example:

```
lmbr_test.cmd scan --dir Bin64vc140.Test --whitelist-file whitelist1.txt --whitelist-file
 whitelist2.txt
```

To set a default whitelist that is always included, create a text file called `lmbr_test_whitelist.txt`. Place the text file in the root directory of the build (where `lmbr_test.cmd` is). The default blacklist is similarly called `lmbr_test_blacklist.txt`. A default blacklist is provided with the build to capture known libraries that do not need to be scanned.

Writing Tests for AzTest

Automated testing is important for any game project. To perform automated testing in Lumberyard, you can use AzTest and the AZ test scanner. This document shows you how to write the tests to build and run. For information on building and running unit and integration tests, see Using AZ Test Scanner (p. 1508).

Lumberyard and AzTest use the GoogleTest and GoogleMock frameworks for unit and integration tests. When you write your tests, we recommend that you consult the corresponding documentation.

Testing code with AzTest has three steps: Configure your module for testing, write the tests, and define global environments.

Configuring Your Module for Testing

If you want to add tests to a module or a gem that is included with Lumberyard, the configuration has already been done for you and you can skip this step. If you want to add tests to non-Lumberyard modules, projects, or tools, then you must configure them to use AzTest.

All modules and libraries in Lumberyard must be configured for use with Waf. If you are unfamiliar with configuring build modules in Waf, see Adding a Build Module (p. 1829) before continuing.

To configure your module for Waf

1. Create a `.waf_files` content file that lists the files for the test. A separate `.waf_files` content file for test files keeps the test files from being compiled in normal builds. Your file name should make it clear that it contains only test files. Existing Lumberyard modules and gems use the naming convention *<module_name>*`_test.waf_files`.

   The following example shows a `.waf_files` file for a HelloWorld module.

   ```
   {
   "none":
       {
           "Tests":
           [
               "HelloWorldTestMain.cpp"
           ]
       }
   }
   ```

   After you create the file, you must reference it in your module's `wscript`. To specify that the file is intended only for test builds, you must add `test_all_file_list` to the `wscript` configuration, as in the following example.

   ```
   # HelloWorld wscript
   def build(bld):

       bld.CryEngineModule(
           target    = 'HelloWorld',
           vs_filter = 'Engine',
           file_list = 'helloworld.waf_files',
           use       = ['AzCore'],

           # Testing
           test_all_file_list = 'helloworld_test.waf_files'
   ```

2. Create a test hook for AZ Test Scanner. How you create a test hook depends on whether you want to build a dynamic library or an executable file. Static libraries are not currently supported for testing.

   - **To create a test hook for a dynamic library**

     Because the tests built into a dynamic library are not exposed, you must expose a separate test function for the AZ test scanner. To expose the test function, use a convenience macro that AzTest provides in a test-only `*_test.waf_files` file, as in the following example.

     ```
     // HelloWorldTestMain.cpp

     #include <AzTest/AzTest.h>

     AZ_UNIT_TEST_HOOK();  // Runs unit tests
     AZ_INTEG_TEST_HOOK(); // Runs integration tests
     ```

     For differences between unit tests and integration tests, see Creating Unit Tests and Integration Tests (p. 1514).

   - **To create a test hook for an executable file**

To build an executable file, you must expose a test function and modify the main function of the executable to run tests instead of normal program functions. The test function informs the AZ test scanner that tests have been included in the executable and that it is safe to continue.

The following example shows how to modify your main function:

```
// Main.cpp

#if defined(AZ_TESTS_ENABLED)
#include <AzTest/AzTest.h>
DECLARE_AZ_UNIT_TEST_MAIN()
#endif

int main(int argc, char* argv[])
{
#if defined(AZ_TESTS_ENABLED)
    INVOKE_AZ_UNIT_TEST_MAIN();
#endif

// Rest of your program

}
```

Note
Because the `AZ_TESTS_ENABLED` definition is defined only in test builds, it is a convenient definition to use in test-only code.

3. If necessary, use the `use` parameter in the `wscript` file to link AzTest to the module.

   Note
   In most cases, you do not have to perform this step. Waf automatically links AzTest in test builds for almost all build modules, including gems. The `LumberyardApp` build module is not linked automatically. If you build a Lumberyard app, you must link it manually.

   As with the `*_test.waf_files` files, you must use AzTest only in a test build. To specify this when linking, add `test_all_use` to the `wscript` configuration, as in the following example.

```
# HelloWorld wscript
def build(bld):

    bld.CryEngineModule(
        target    = 'HelloWorld',
        vs_filter = 'Engine',
        file_list = 'helloworld.waf_files',
        use       = ['AzCore'],

        # Testing
        test_all_file_list = 'helloworld_test.waf_files',
        test_all_use       = ['AzTest'],
```

After you have performed these three steps, you can create a test build and test it with the AZ test scanner. Your `HelloWorld` library or executable should appear in the report, although at this point it does not have tests.

Creating Tests

After your module is configured for testing and visible to the AZ test scanner, you can start writing tests for it. Detailed information about writing tests is available in the GoogleTest and GoogleMock documentation. The following information describes characteristics specific to AzTest.

Creating a Simple Example Test

When using AzTest to write tests, include only the `AzTest.h` file in your test files instead of GoogleTest and GoogleMock, as in the following example.

```
// HelloWorldTests.cpp

#include <AzTest/AzTest.h>
#include "HelloWorld.h"

TEST(HelloWorldTests, HelloWorld_ReturnsHelloWorld)
{
    // Call HelloWorld() and compare it to the expected C-string
    ASSERT_STREQ("Hello World!", HelloWorld());
}
```

Creating Unit Tests and Integration Tests

Unit tests are designed to be run in isolation. Unit tests are expected to pass without requiring Lumberyard's engine systems or other assets. Integration tests are designed to be run with the game engine and have access to the engine's systems or assets.

To prevent unexpected failures, the different kinds of tests must be kept separate. Lumberyard does this by providing different hooks for each kind of test. Declare the tests as unit or integration, and call the AZ test scanner accordingly.

For tests that do not require fixtures, simply replace the `TEST` macro with `INTEG_TEST`, as in the following example.

```
// HelloWorldTests.cpp

#include <AzTest/AzTest.h>
#include "HelloWorld.h"

INTEG_TEST(HelloWorldTests, MessageOfTheDay_ReturnsCurrentMessageOfTheDay)
{
    // Call MessageOfTheDay() and compare it to the expected C-string
    // This is an integration test because MessageOfTheDay is stored in a database
    MessageDatabase messageDatabase = HelloWorld::GetTestMessageDatabase();
    ASSERT_STREQ("This is your message for the day!", messageDatabase.MessageOfTheDay());
}
```

For tests that do use fixtures, add the `Integ_` prefix to the fixture name to indicate that it is to be used only for integration tests, as in the following example.

```
// HelloWorldTests.cpp

class Integ_MessageDatabaseTests : public ::testing::Test
{
protected:
    void SetUp() override
    {
        m_messageDatabase = HelloWorld::GetTestMessageDatabase();
    }

    MessageDatabase m_messageDatabase;
}

TEST_F(Integ_MessageDatabaseTests, MessageOfTheDay_ReturnsCurrentMessageOfTheDay)
```

```
{
    ASSERT_STR("This is your message for the day!", messageDatabase.MessageOfTheDay());
}
```

Note
Fixtures cannot be shared between unit and integration tests. If you want to use the same fixture for both kinds of tests, create a base fixture and subclass for each type of test.

Using Global Environments

It is often useful to create some global variables or mocks for testing the module. In Lumberyard, this might mean creating memory allocators or a test system environment. Because AzTest extends the support that GoogleTest provides, you must use the AzTest environments.

The following example shows how to define an AzTest global environment:

```
// HelloWorldTestMain.cpp

#include <AzTest/AzTest.h>

class HelloWorldEnvironment : public AZ::Test::ITestEnvironment
{
protected:
    void SetupEnvironment() override
    {
        // Environment setup here
    }
    void TeardownEnvironment() override
    {
        // Environment teardown here
    }
}
```

To initialize the environments, use the test hooks that were mentioned previously. Because the macros are variadic, you can initialize as many environments as you require. The following example initializes the environment in the previous example.

```
// HelloWorldTestMain.cpp

#include <AzTest/AzTest.h>

class HelloWorldEnvironment : public AZ::Test::ITestEnvironment
{
protected:
    void SetupEnvironment() override
    {
        // Environment setup here
    }
    void TeardownEnvironment() override
    {
        // Environment teardown here
    }
}

AZ_UNIT_TEST_HOOK(new HelloWorldEnvironment);
```

Note
You must dynamically allocate environments before you use the macro. This gives you full control over how environments are created at runtime.

The order of environments is also significant. Environments earlier in the list are initialized earlier and removed later. This is important if a global environment depends on another environment that already exists.

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.

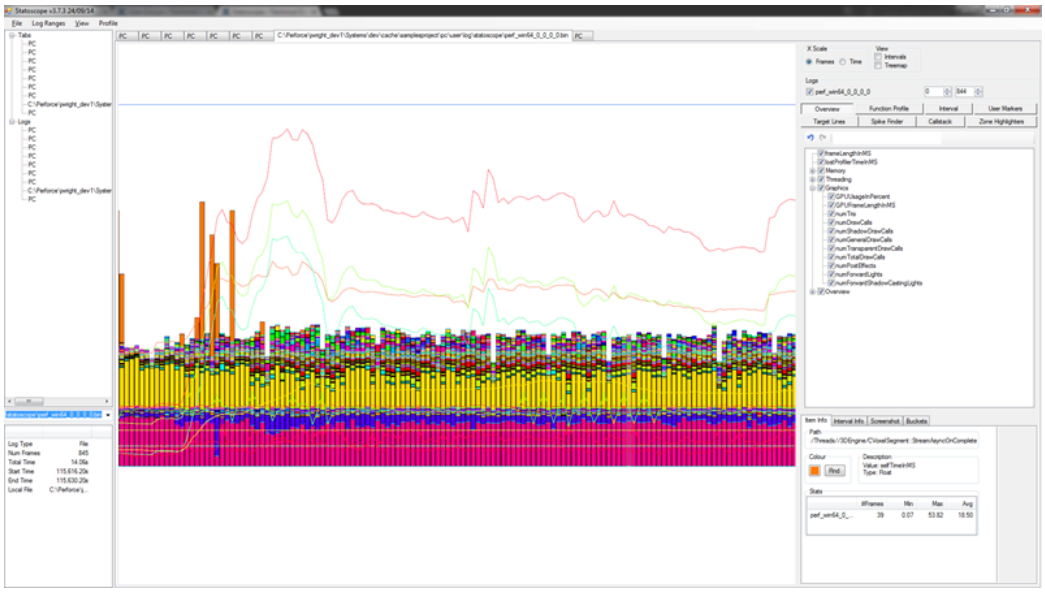
Statoscope will connect to any platform or console that is connected directly to the PC or via an IP address specified in the connection settings dialog.

Topics

User Interface

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 millseconds.

- 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.

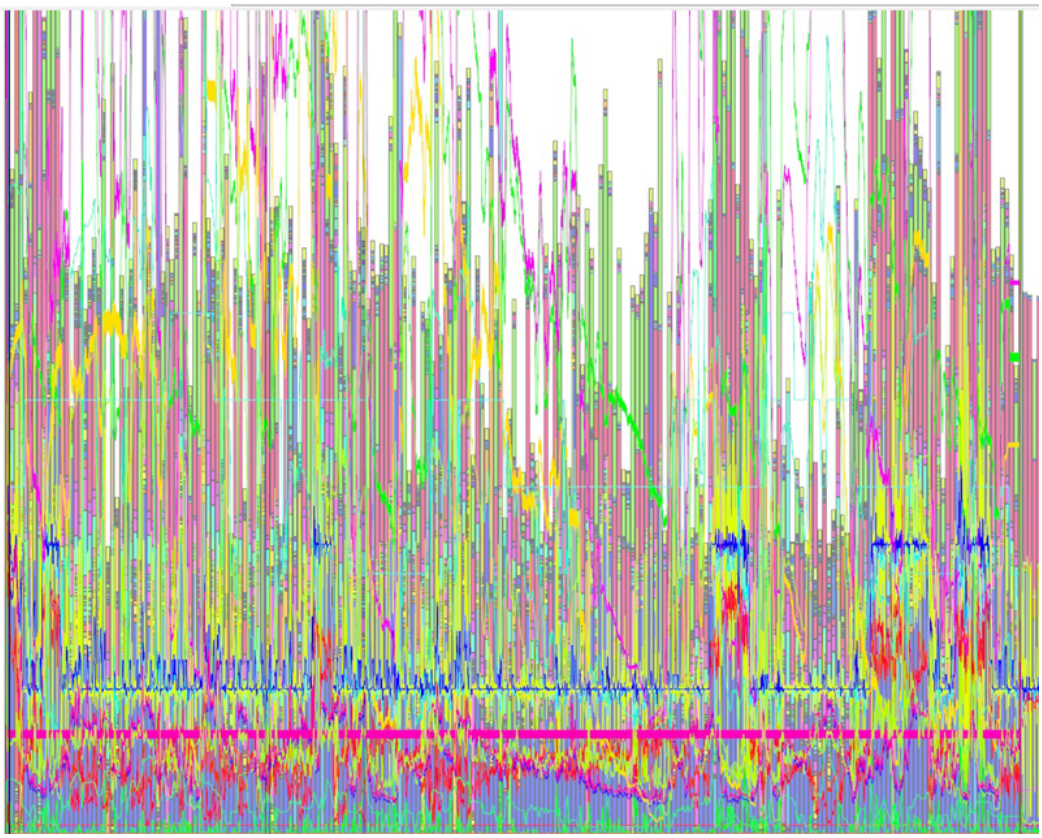
The y-axis can be scaled in order to compare data that varies greatly in value, such as number of drawcalls.

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 `fgmtuO`.
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. 1526).

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 `fgmtuO`.
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.

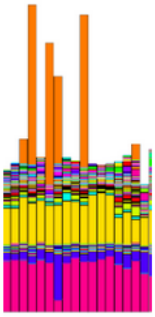
Data Groups

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 `frame profilers` 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

- RTSceneDrawningLengthInMS

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 implemetation 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 provides the following built-in debugging and profiling tools that you can use to locate and fix performance issues.

- Character skeleton debugging (p. 204) – Use the `p_draw_helpers` console variable to debug character skeleton issues.
- Cinematics debugging (p. 435) – Debug cinematics issues.
- Flow Graph debugging (p. 1057) – Use the Flow Graph debugger and console variables to debug flow graph issues.
- Mannequin debugging (p. 360) – Debug Mannequin system issues.
- Particle debugging (p. 1344) – Debug particles.
- Vegetation debugging (p. 1206) – Debug vegetation objects.

Using Console Debug Views

The viewport window displays debugging information by default when you are in gameplay mode (**Ctrl +G**). You can toggle this information on or off by pressing the tilde (**~**) key. Use the following console variables and values to generate 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 for debugging object culling and LOD.
- `e_defaultmaterial 1` – Applies a uniform, flat, gray material to every surface in the level.
- `e_terrainbboxes` – Displays terrain bounding boxes.
- `p_debug_joints 1` – Shows the mass of objects in kilograms and the joint that is linked to the object. To display joints, you must first enable `p_draw_helpers 1`.
- `p_draw_helpers 1` – Shows physics proxy meshes in addition to the render geometry.
- `r_DisplayInfo 0` – Disables the debugging text.
- `r_DisplayInfo 1 | 2 | 3 | 4` – Enables various levels of debugging text. Displays memory consumption, frame rate, triangle count, visible light sources, and drawcall count. Use a value of `2` to display more detailed information. Use a value of `3` to display only frames per second (FPS) and frame time in milliseconds.
- `r_wireframe 1 | 2` – Use a value of `1` to draw the level in wireframe mode. Use a value of `2` to draw the level in vertex mode. Both include the objects that are hidden from view.
- `r_showlines 2` – Overlays the wireframe only on the front-facing geometry. Anything behind this geometry isn't rendered.
- `r_texbindmode 6` – Applies a uniform, flat, gray material with normal map information to every surface in the level.

Using DebugDraw Console Variables

Use the following console variables and values to display 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).
- `e_DebugDraw 15` – Displays helpers.
- `e_DebugDraw 16` – Displays debug gun.
- `e_DebugDraw 17` – Displays streaming information (buffer sizes).
- `e_DebugDraw 18` – Displays streaming information (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

Use the following console variables and values to display 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 the roughness or glossiness of the surfaces.
- `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. Gray = standard | Yellow = transmittance | Blue = POM self-shadowing.
- `r_DebugGBuffer 6` – Shows the translucency values that are set on assets in the level. 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** – The specular color is too low.
  - **Orange** – The specular color is too high for dielectric materials.
  - **Pink** – The specular color is valid only for rusted or oxidized metals.

Twitch

Lumberyard is integrated with Twitch so that you can build games that engage with broadcasters and 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.

Twitch JoinIn

The Twitch JoinIn feature within Lumberyard helps you build multiplayer games that allow Twitch broadcasters to 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.

Twitch API

TwitchAPI is a Twitch-specific implementation of the BroadcastAPI interface that allows you to make calls to Twitch's REST API from within Lumberyard.

Topics

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.

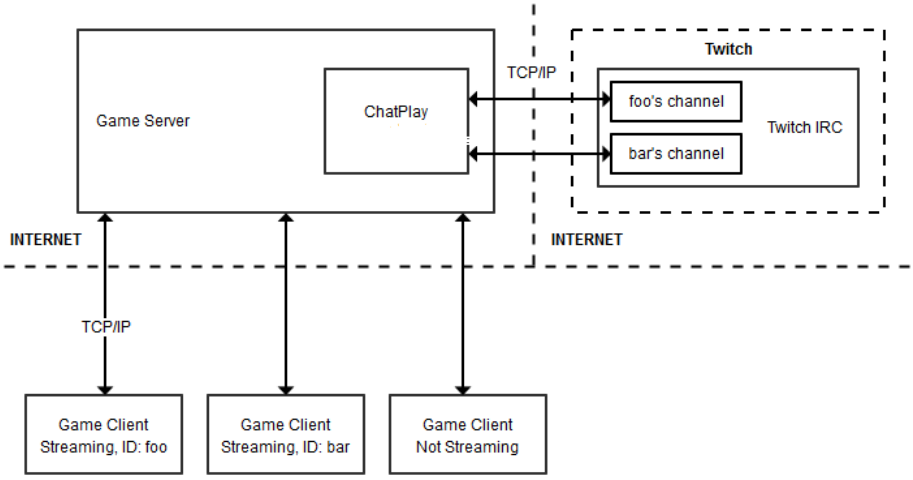
For a tutorial on Twitch ChatPlay, see Amazon Lumberyard Tutorials.

Twitch ChatPlay includes the following components and services:

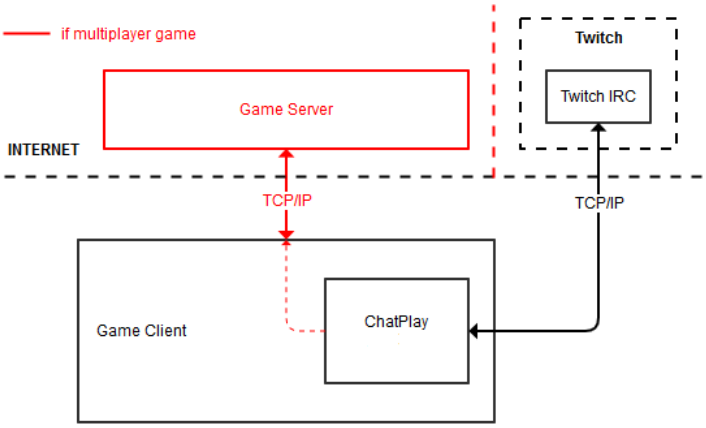
- Twitch IRC servers
- Twitch ID authentication
- Twitch account
- Twitch Nodes (p. 1005)

In addition, Twitch JoinIn (p. 1537) 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

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.

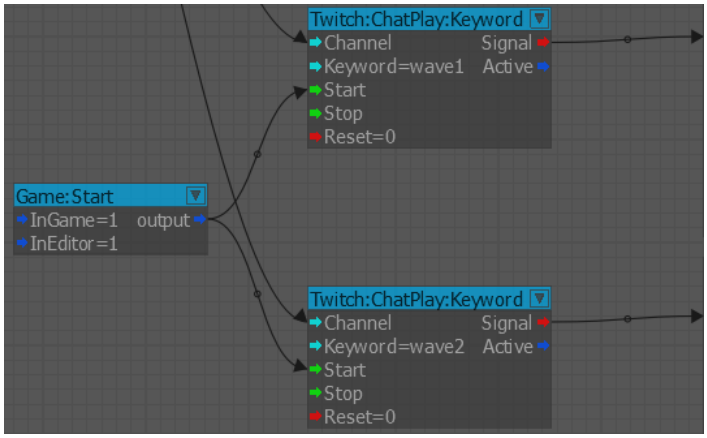
> **Note**
> Channels are automatically disconnected when flow nodes are uninitialized. This means that disconnection is automatic in most situations without need for further action.

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.

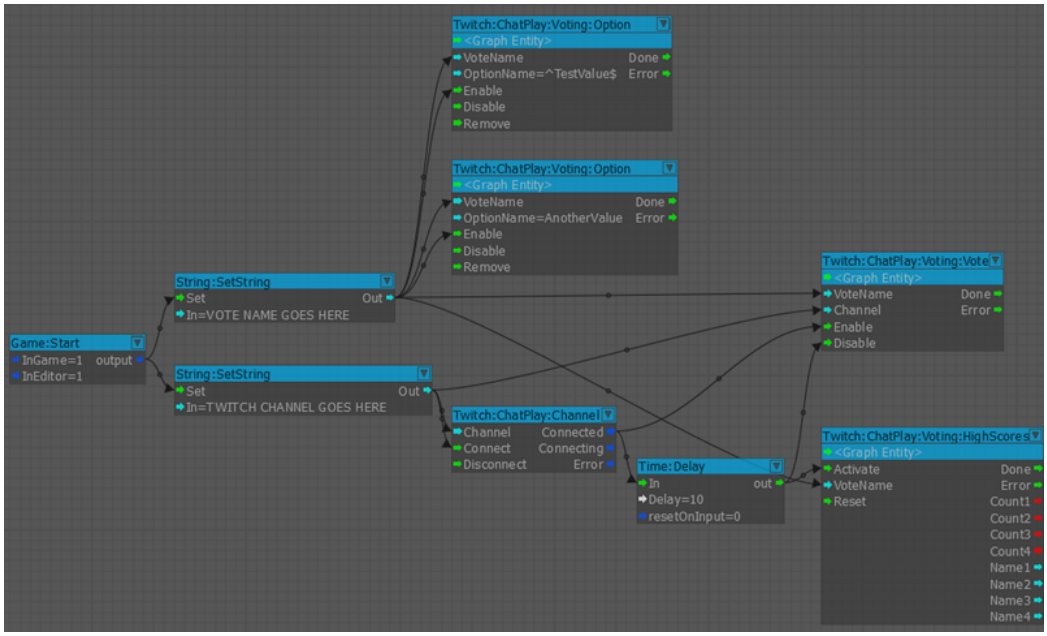


Using Flow Graph with Twitch ChatPlay

There are a number of flow graph nodes you can use to configure Twitch ChatPlay-related settings. For more information, see Twitch Nodes (p. 1005).

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. 1005).

Twitch ChatPlay Console Variables, Classes, and Connection Methods

This topic includes console variables, classes, and connection methods for Twitch ChatPlay.

Topics

Console Variables

Twitch ChatPlay uses the following console variables.

`chatplay_Enabled`

Enables or disables the Twitch ChatPlay feature.

Default: `1`

Valid values: `0` = Disable | `1` = Enable

`chatplay_UserName`

Default username to use when logging in to the Twitch IRC. The default value represents an anonymous user that can read chat but cannot post.

Default value: `justinfan12345`

`chatplay_Password`

Default password to use when logging in to the Twitch IRC.

Default value: `blah`

`chatplay_ClientID`

Client ID that is generated from Twitch.tv for making API requests. See Generating and Setting a Twitch Client ID (p. 1534).

`chatplay_ServerListEndpoint`

Base address to use for making API requests. Set the client ID in order for these API calls to succeed.

Default: `tmi.twitch.tv`

Classes

The Twitch ChatPlay module is implemented as a part of CryAction and is accessible from the `GetChatPlay` method. The main interface is the `ChatPlay.h` file.

Twitch ChatPlay uses the following classes.

`IChatChannel`

Interface that represents a Twitch ChatPlay channel and 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.

Connection Methods

You can use various methods and ports to connect Twitch ChatPlay to the Twitch IRC.

Twitch ChatPlay uses the following console variables to configure the connection methods. Set these console variables with comma delimited `priority:port` pairs. Ports with a lower number have higher priority. Ports with a priority of `-1` will be ignored. Twitch ChatPlay will attempt to connect to the Twitch IRC using each method and port combination in priority order, until a connection is successful or all available combinations fail.

`chatplay_IRCPorts`

List of ports and their priorities that are used for connecting to the Twitch IRC.

Default: `1:6667;3:80`

`chatplay_IRCSSLPorts`

List of ports and their priorities that are used for connecting to the Twitch IRC over SSL.

Default: `-1:6697;-1:443`

`chatplay_WebsocketPorts`

List of ports and their priorities that are used for connecting to the Twitch IRC over websockets.

Default: `2:80`

`chatplay_WebsocketSSLPorts`

List of ports and their priorities that are used for connecting to the Twitch IRC over secure websockets.

Default: `-1:443`

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.

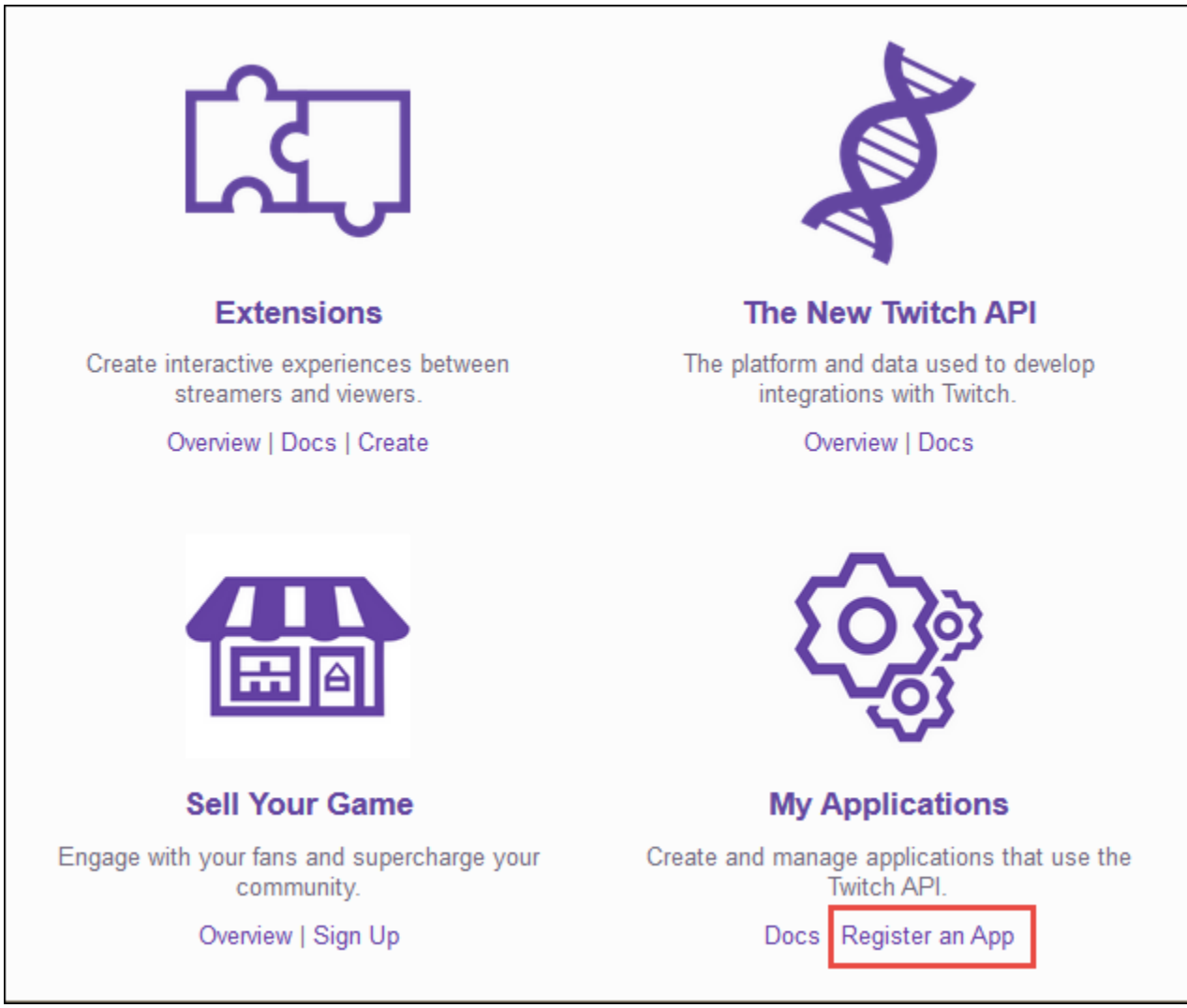
If you have already registered your application with Twitch, you can locate your client ID on the Twitch Dev site. Click **My Applications** and then select **Manage** on the name of your application.

Generate a Client ID

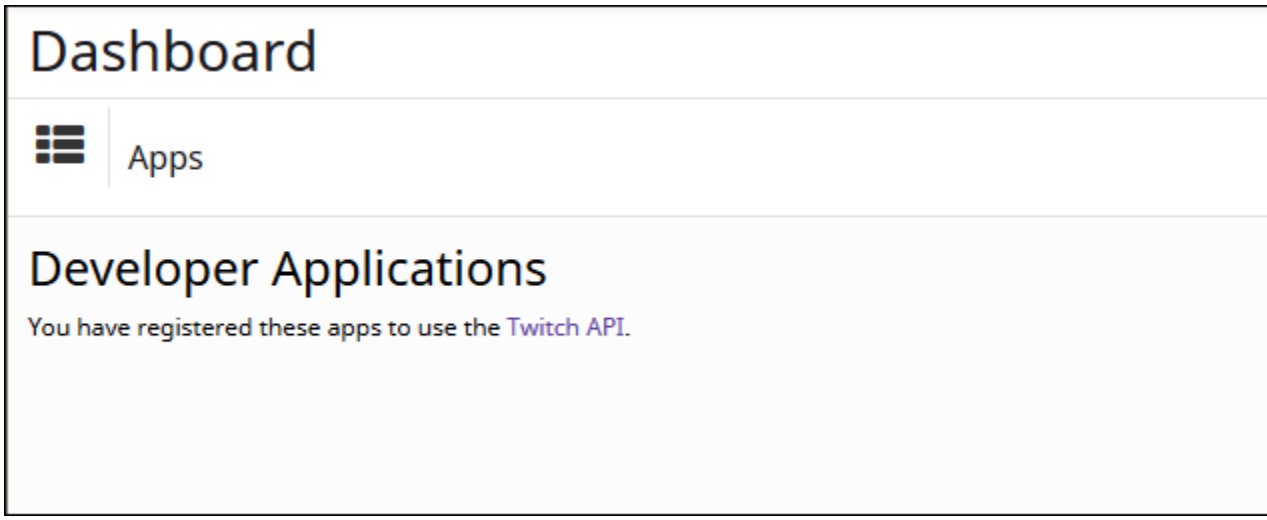
Generate a client ID by following the instructions below.

To generate the client ID

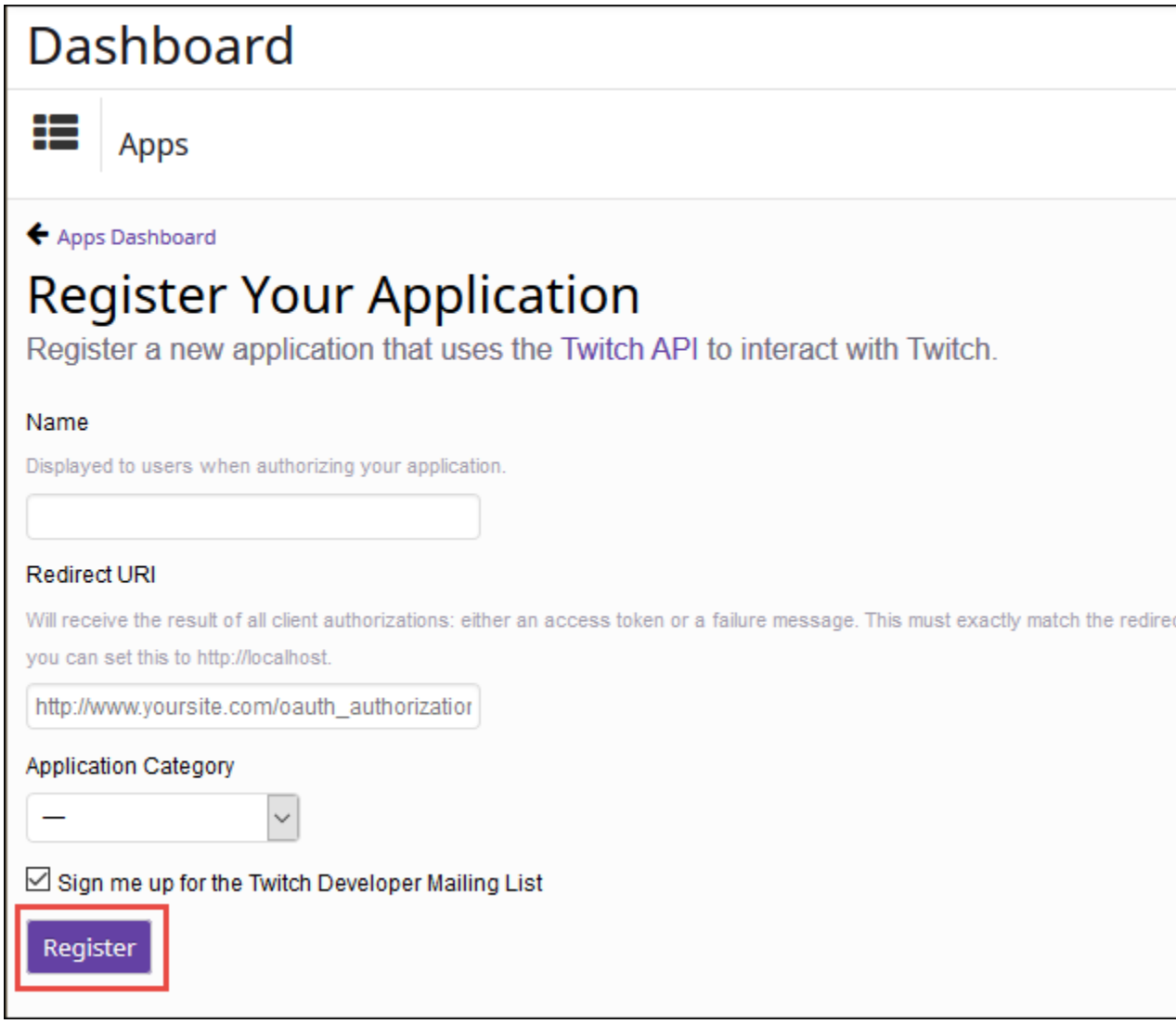
1. Go to the Twitch Dev site and log in to your account.
2. Under **My Applications**, choose **Register an App**.



3. On the **Dashboard**, under **Developer Applications**, choose **Register Your Application**.



4. On the **Register Your Application** page, complete the form and choose **Register**.



5. 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`. You can use that node create a link that includes all the multiplayer session information necessary for other players to connect to the session. 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 `mphost` 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 the following:
  - 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. 1011).

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. 1005).

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. 1534).

UI System

You can use the **UI Editor** to create and customize various parts of the game user interface, such as images, text, buttons, menus, scroll boxes, and heads-up displays (HUDs). For a tutorial about UI creation for games, see Lumberyard Tutorials.

Topics

Using the UI Editor

You can use the **UI Editor** to create, customize, and animate various game user interface elements and components such as menus, buttons, and heads-up displays (HUDs).

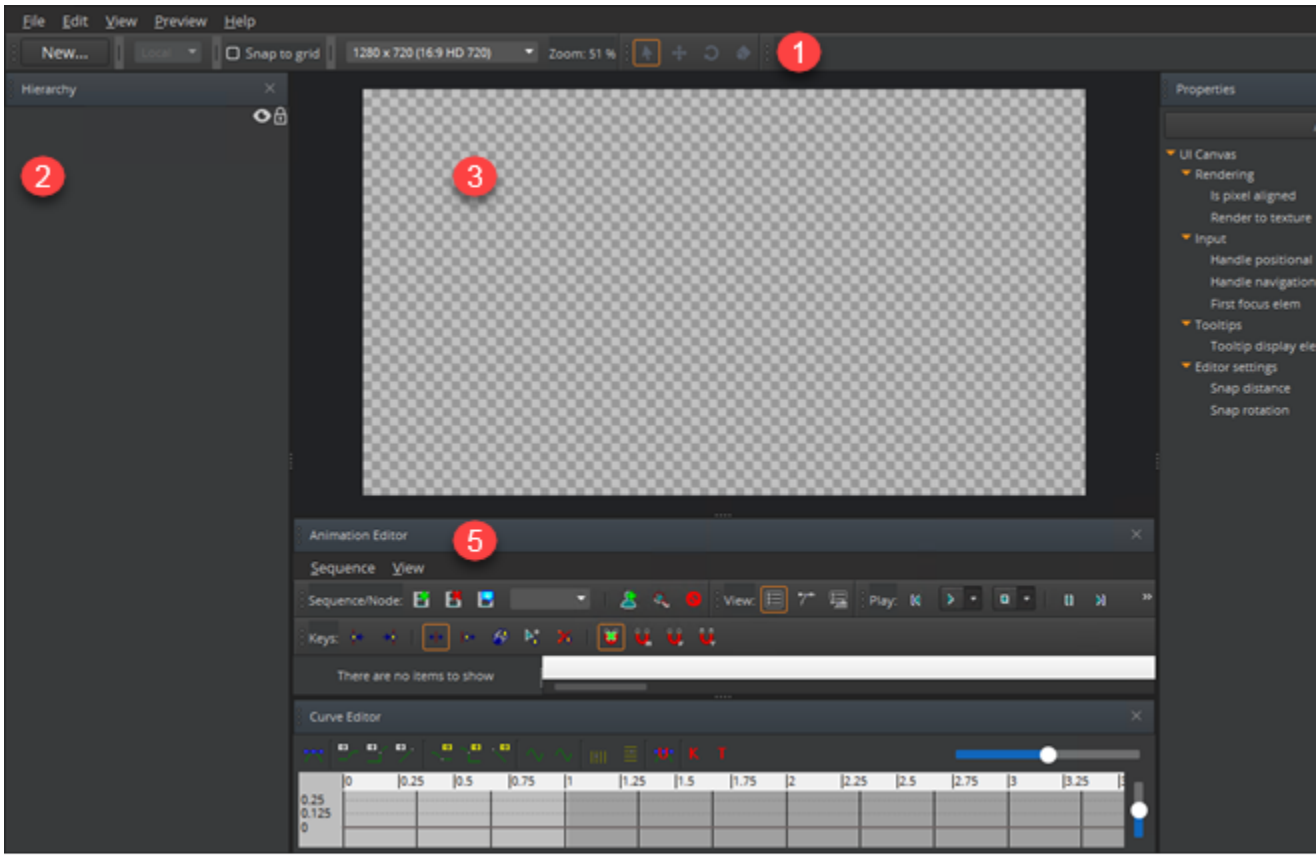
The **UI Editor** consists of the following:

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 **Tools**, **UI Editor**.



Working with UI Canvases

The **UI Editor** uses the concept of a canvas as an invisible backdrop for your game user interface elements. Once you create a canvas, you can add elements such as images, text, and buttons.

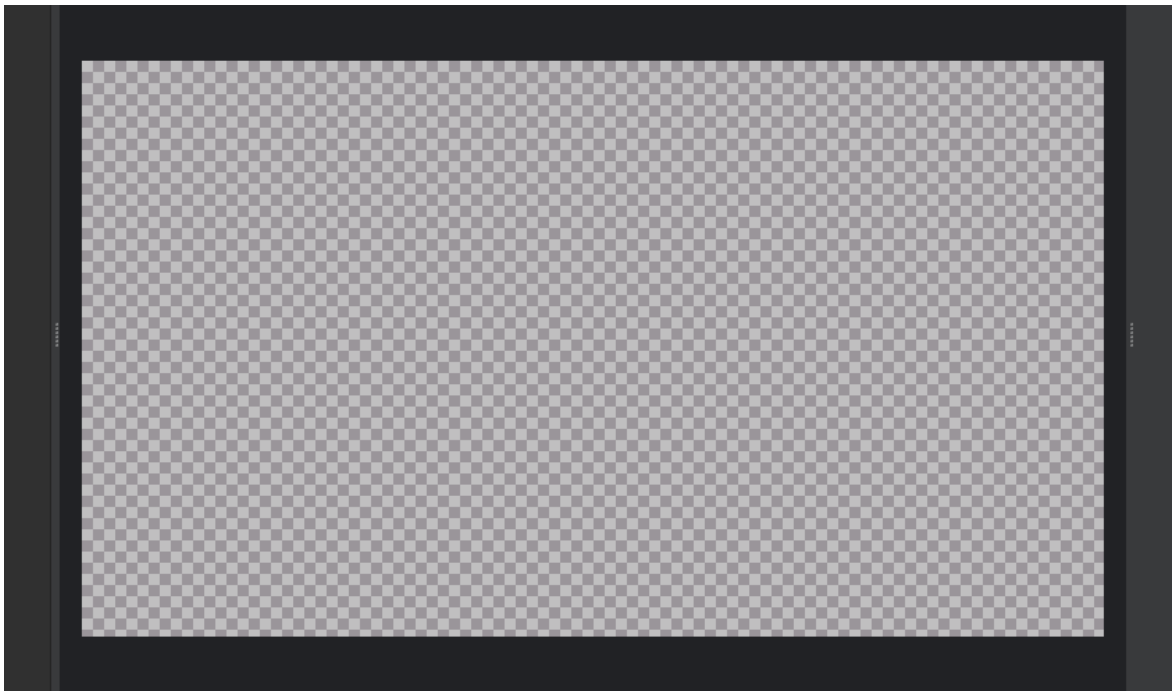
To create a UI canvas

1. In Lumberyard Editor, click **Tools**, **UI Editor**.
2. In the UI Editor, add elements (p. 1557), components (p. 1559), slices (p. 1554), and prefabs (p. 1558).
3. Click **File**, **Save As**. Name the canvas with a `.uicanvas` file extension, and then click **Save**.

Topics

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**.

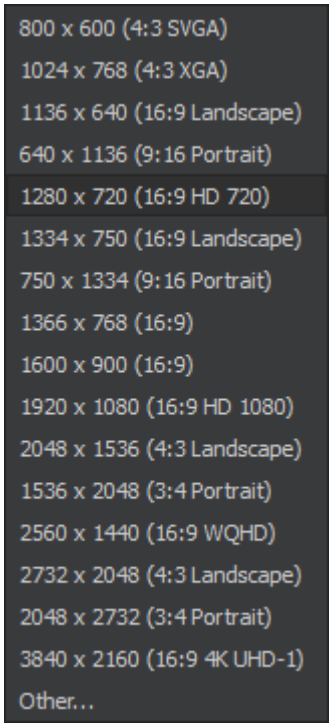
Changing the Canvas Size

Change your game UI 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 reference 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`

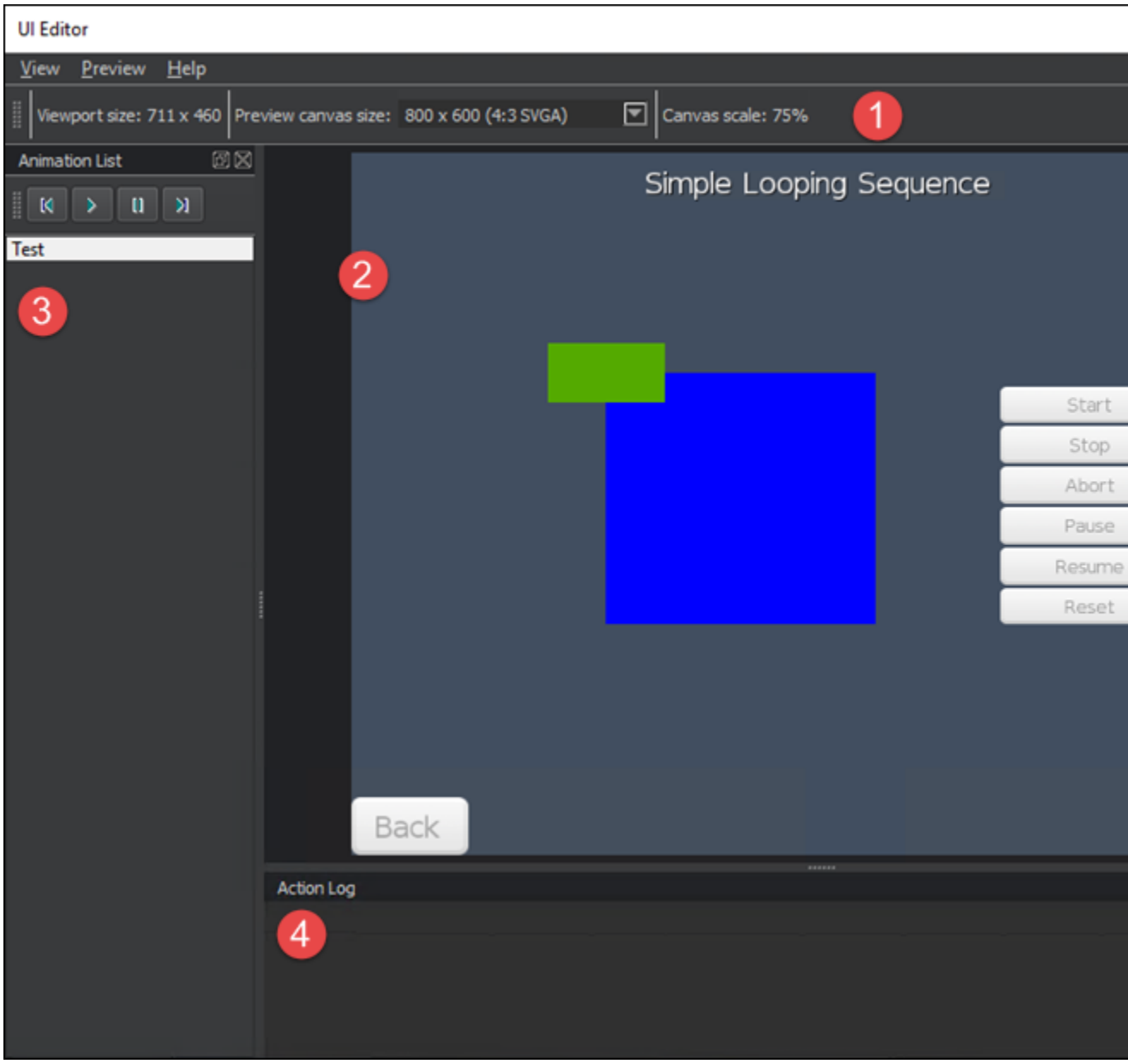
Previewing Canvases

You can preview your game UI canvas to visualize how it might look at different screen resolutions and to see how the interactive elements change states.

Topics

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. 1544) – 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. 1544) – 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 game UI 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. 1542).

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. 1560) 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 Behavior

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.

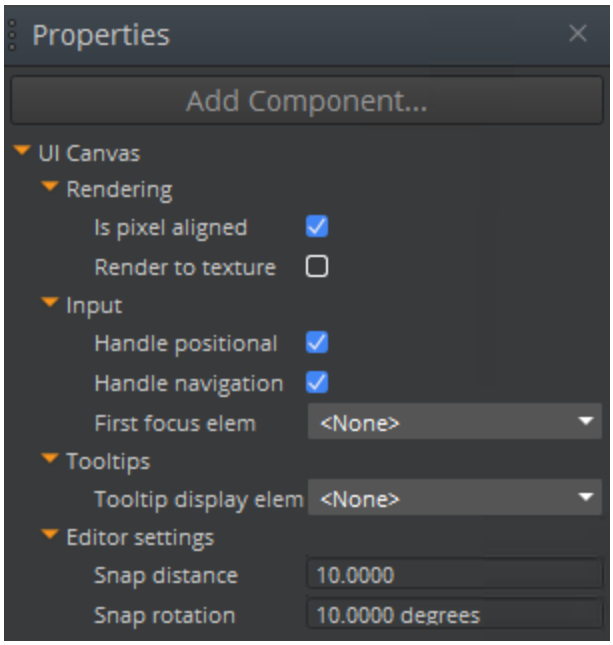
   The text strings are fully customizable; you can type any string that helps you ensure that the correct actions are being triggered.

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. 1570).

Tooltips Properties

The **Tooltips** property controls which element is displayed when hovering over an interactive element. Select an element from the drop-down list. This list is composed of the elements on your current canvas that contain the **TooltipDisplay** component. For more information about the **Tooltips** components, see Tooltip Components (p. 1603).

Editor Settings 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.

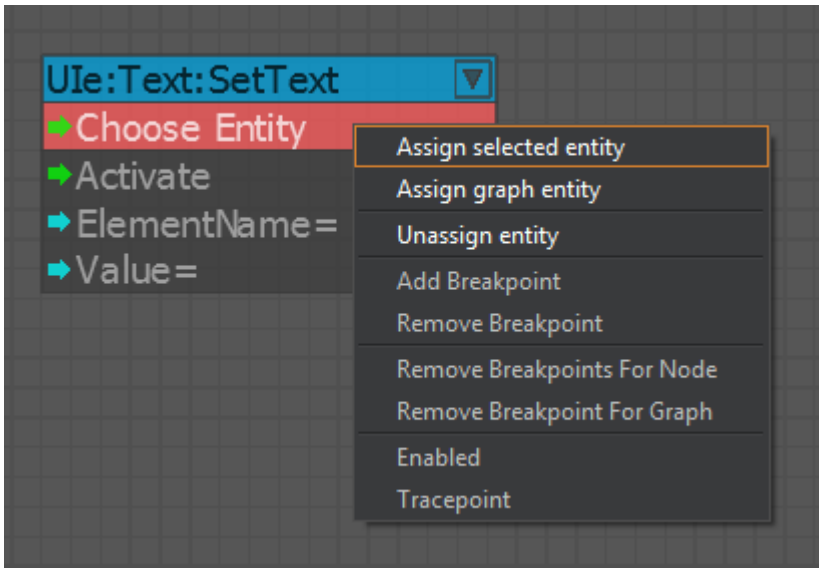
Using the Component Entity system to associate a UI canvas with a UIe flow graph node

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 **Tools**, **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.



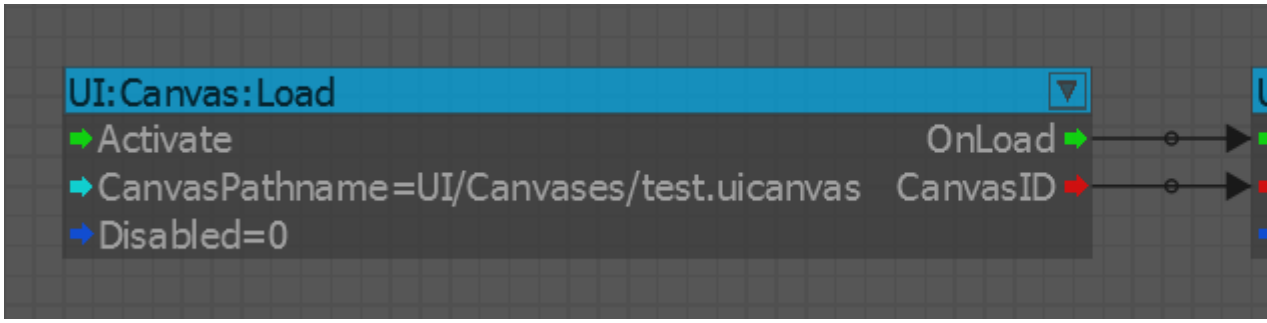
Using the Legacy Entity system to associate a UI canvas with a UIe flow graph node

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 **Tools**, **Flow Graph**.
4. In the **Flow Graph** editor's **Graphs** pane, select a flow graph.
5. Add any **UIe** 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. 1548) 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.

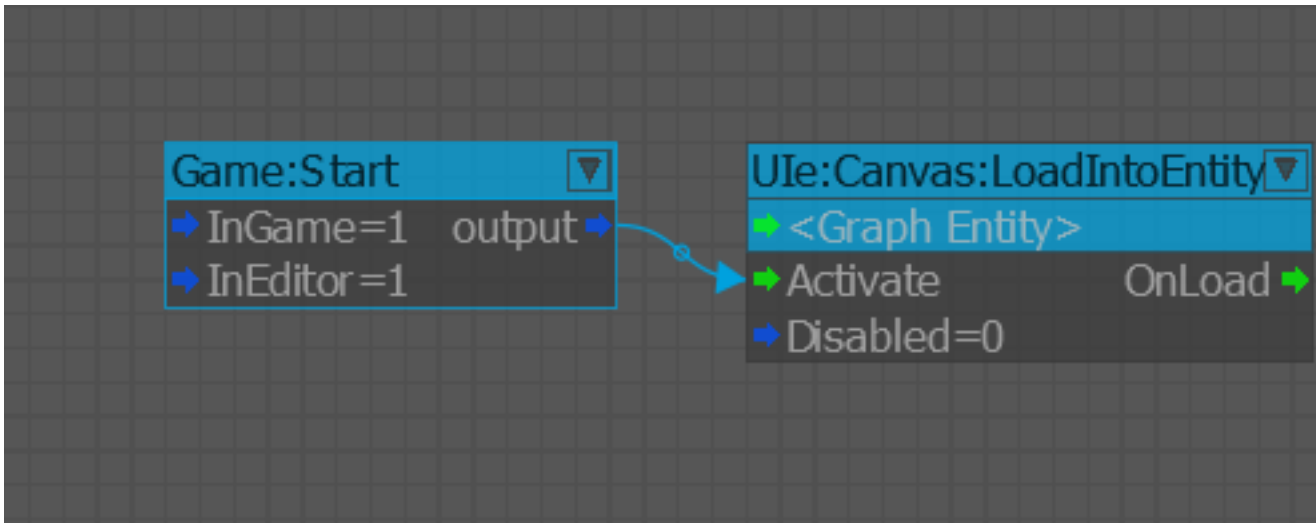


Loading Canvases in the Flow Graph Editor

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. 754).

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. 1626).

You can load canvases in the **Flow Graph Editor** using either the **UIe** 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.

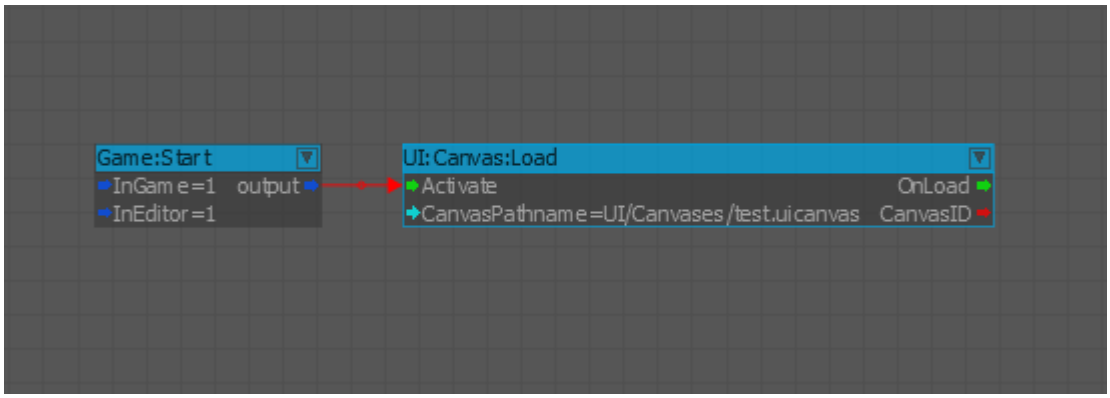


Use the following procedure to load canvases using the **UIe** node set. This is the recommended method of loading canvases in flow graph.

To load a canvas in the Flow Graph Editor using the UIe node set

1. In Lumberyard Editor, click **Tools**, **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. 1546).
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

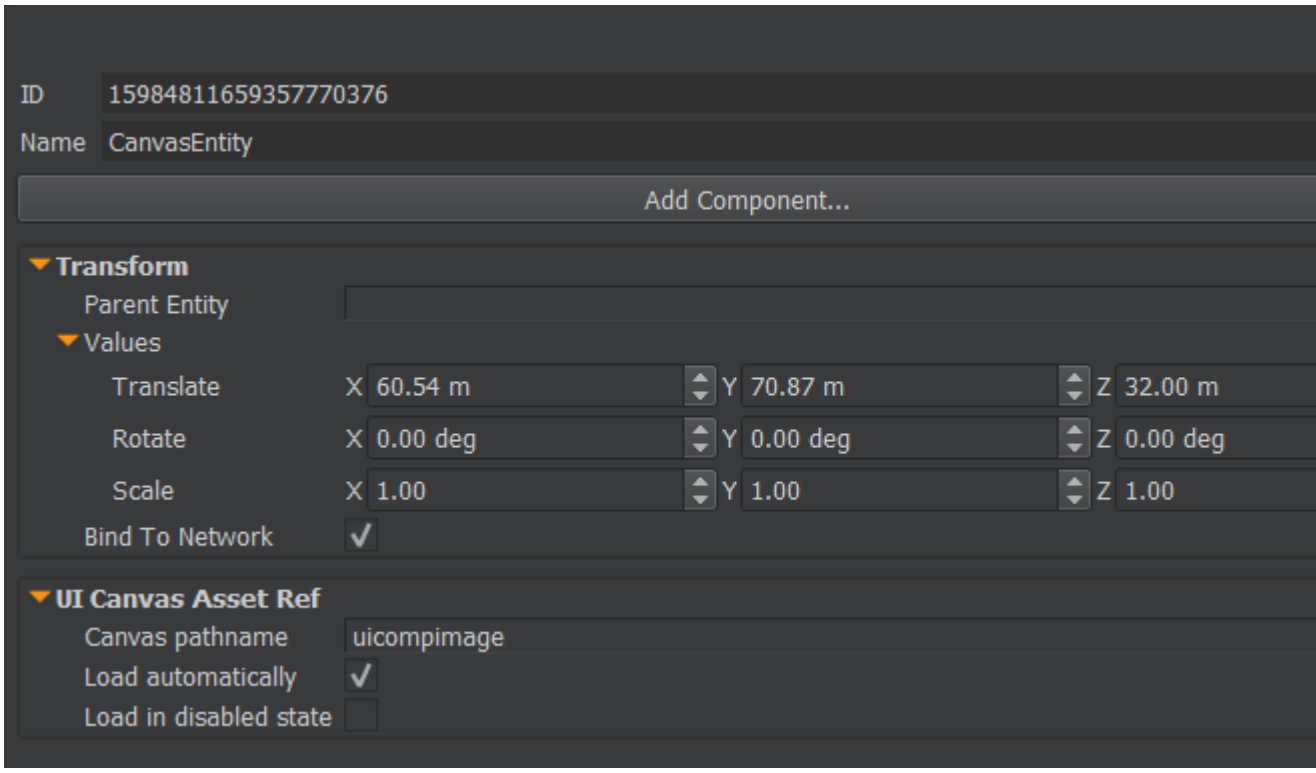
1. In Lumberyard Editor, click **Tools**, **Flow Graph**.
2. In the **Flow Graph** editor, select a flow graph from the **Graphs** pane.
3. Right-click anywhere in the graphs pane and select **Add Node**, **Game**, **Start**.
4. Right-click anywhere in the graphs pane and select **Add Node**, **UI**, **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. 593)
2. In the **Entity Inspector**, add to this component entity (p. 594) 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. For more information, see Loading Canvases in Lua in the Amazon Lumberyard Developer Guide.

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.

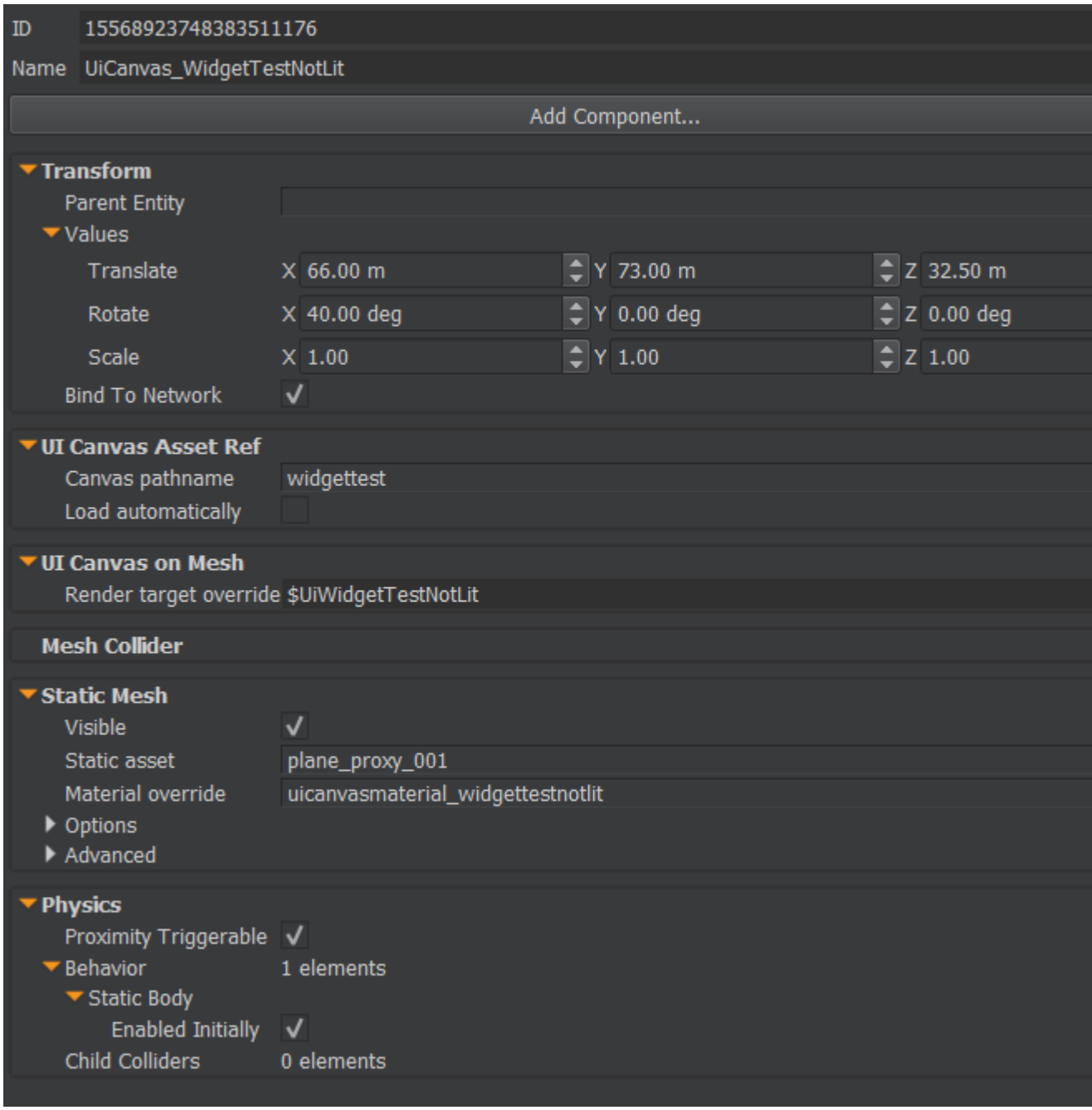
To see an example of a UI canvas on an object in a 3D world, open the **UiIn3DWorld** level in the Samples Project.

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. 1540). In the canvas properties (p. 1545), select **Render to texture** and type a name in the **Render target** text box. The name must start with the `$` character. This indicates to the graphics system that the texture is a dynamic texture.
2. In the level, create a component entity (p. 593).

3. In the **Entity Inspector**, add to this component entity (p. 594) 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. 1548) for other ways to load the canvas.
4. In the **Material Editor** (p. 1396), create a material that uses the render target texture that is rendered by your canvas.
5. Add a Static Mesh (p. 521) 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 Mesh Collider component (p. 522). 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.



Defining Game and Level Load Screens

You can create a game or level loading screen with the **UI Editor**. The game loading screen is displayed while the game loads. The level loading screen is displayed while a level loads—you can create and define a loading screen for each level.

To define the game and level loading screens, you set the file paths as parameters in `game.cfg` and `level.cfg`. Alternatively, you can use **Flow Graph** to specify a level loading screen.

Defining a Game Loading Screen

To define a game loading screen, you first create the loading screen canvas in the **UI Editor** and save it within your project directory. For example, for the Samples Project, the project directory is `\dev\SamplesProject\`. You then add or modify parameters in `game.cfg`, which is at the root of your project directory.

To add game loading screen parameters to `game.cfg`

1. Use a text editor to open `game.cfg` at the root of your project directory.
2. Add or modify the following parameters in `game.cfg`:

   - `game_load_screen_uicanvas_path` – File path to the `.uicanvas` game load screen file relative to your project path.

     For example, if your game load canvas is at `\dev\SamplesProject\UI\Canvases\UiAnimMultiSequence.uicanvas`, you would type `\UI\Canvases\UiAnimMultiSequence.uicanvas` for this parameter.
   - `game_load_screen_sequence_to_auto_play` – The name of the game load screen animation sequence to play on load.
   - `game_load_screen_sequence_fix_fps` – A fixed frame rate for the game load screen animation to play on load. Default is `60`. To ignore this setting and use the real time-delta, enter `-1`.

The following is an example of these parameters in a `game.cfg` file:

```
game_load_screen_uicanvas_path="UI/Canvases/UiAnimMultiSequence.uicanvas"
game_load_screen_sequence_to_auto_play="TopRowMove"
game_load_screen_sequence_fix_fps=4.0
```

Defining a Level Loading Screen

To define a level loading screen, you first create the loading screen canvas in the **UI Editor** and save it within that level's directory. You then add or modify parameters in `level.cfg`, which is at the root of your level directory.

To add game loading screen parameters to `level.cfg`

1. Use a text editor to open `level.cfg` at the root of your level directory.
2. Add or modify the following parameters in `level.cfg`:

   - `level_load_screen_uicanvas_path` – File path to the `.uicanvas` level load screen file relative to your level path.

     For example, if your level load canvas is at `\dev\StarterGame\Levels\StarterGame\UiAnimMultiSequence.uicanvas` (root of the level directory— same as `level.cfg`), you would type `UiAnimMultiSequence.uicanvas` for this parameter.
   - `level_load_screen_sequence_to_auto_play` – The name of the level load screen animation sequence to play on load.
   - `level_load_screen_sequence_fix_fps` – A fixed frame rate for the level load screen animation to play on load. Default is `60`. To ignore this setting and use the real time-delta, enter `-1`.

You can also specify a level loading screen using **Flow Graph** by using the **Mission:LoadNextLevel** node.

To specify a level loading screen with Flow Graph

- In Flow Graph, open a level flow graph. Add the **Mission:LoadNextLevel** node.

  Define the following parameters:

  - `LoadScreenUiCanvasPath` – File path to the `.uicanvas` level load screen file relative to your level path.

    For example, if your level load canvas is at `\dev\StarterGame\Levels\StarterGame\UiAnimMultiSequence.uicanvas` (root of the level directory— same as `level.cfg`), you would type `UiAnimMultiSequence.uicanvas` for this parameter.
  - `LoadScreenSequenceToAutoPlay` – The name of the level load screen animation sequence to play on load.
  - `LoadScreenSequenceFixFps` – A fixed frame rate for the level load screen animation to play on load. Default is `60`. To ignore this setting and use the real time-delta, enter `-1`.

Working with UI Slices

The UI system uses the slices technology, which is a cascaded data management system for entities. This slice technology is similar to what the main Lumberyard Editor uses, but the user interface and some of the requirements are slightly different for the UI system. For more information about component entity system slices, see Working with Slices (p. 597).

UI slices replace UI prefabs. The UI prefab functionality is still available and functional, but its menu items are marked as deprecated and will be removed in a future release.

Topics

Creating a UI Slice

A slice can contain any number of UI elements. However, all of the UI elements in a slice must be contained within one parent element.

You don't need to select all of the elements to go into the new slice. If you just select one element, all of its child elements are included in the slice.

To create a UI slice

1. In the **UI Editor**, in the **Hierarchy** pane or in the viewport, select the top-level parent entity to include in the slice.
2. Right-click on the parent entity and click **Make New Slice from Selection**.

3. Save the slice with a descriptive name.

   Because you may create many slices for different purposes, we recommend that you name your slices meaningfully and organize them purposefully into directories and subdirectories.

If you want the slice to appear in the slice library **New**, **Element** menu, you must save it within the `UI/Slices/Library` directory. This directory is either in the project root or within any enabled gems asset root. Adding a slice to the slice library menu makes it easy and convenient to use as a common building block for your UI canvases.

Slices that you create in the **UI Editor** use the same `.slice` extension as the slices that you create in the main Lumberyard Editor. However, you cannot instantiate any slices within the **UI Editor** that you created using the main Lumberyard Editor. If you attempt to do so, Lumberyard displays a warning. For workflow ease, save your UI slices in a different directory than your main Lumberyard Editor slices.

Instantiating a UI Slice

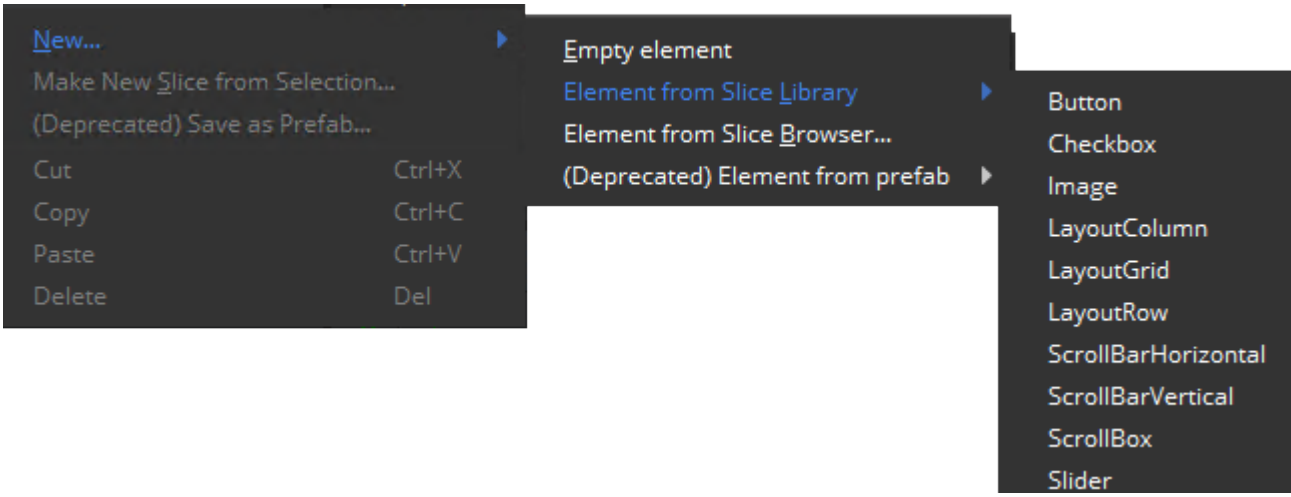
You can create an instance of a slice in your UI canvas.

To instantiate a slice

Do one of the following in the **UI Editor**:

- Click **New**, **Element from Slice Library**.

  This action displays a hierarchical menu of all the slices under `\UI\Slices\Library`.



- Click **New**, **Element from Slice Browser**.

  This action displays a file browser to select your slice file.

Creating a Cascaded UI Slice

A cascaded slice is a slice that contains instances of other slices. Cascaded slices can store their own overrides for component and entity properties and can also contain their own entities.

When you create a cascaded slice, the UI system maintains references to the slices that are child entities. This means, for example, that you could instantiate an image slice as a parent and a text slice as a child. Then you could select them both and make a cascaded slice called `Button`. If you then create instances

of the `Button` slice, they all still contain a reference to the image slice and the text slice. If you then push a font change to the text slice, it affects all instances of the `Button` slice as well as any other instances of the text slice.

To create a cascaded slice

1. Select the root of a set of elements. The child entities within that root can be individual elements, slices, or a combination of both.

   Note
   If your root is not already in a slice, and you want to maintain child slice references, you must also select the child slices in this step. If you select only the root, only one option appears: **Make New Slice from Selection**. This creates a detached UI slice (flattens child references).
2. Right-click the selection and then click **Make Cascaded Slice from Selected Slices & Entities**.
3. Save the cascaded slice with a descriptive name.

Creating a Detached UI Slice

You can create a detached UI slice from an existing UI slice instance. When you create a detached UI slice, the UI system removes, or flattens, all references to child slices. Using the button example from the previous section, let's say you saved the image slice instance and its child text slice instance as a detached slice and called it `Button2`. The detached `Button2` would not reference any other slices; it is a slice that contains two entities. If you pushed a change to the text slice, it would not affect the text within any instances of the `Button2` slice.

To create a detached UI slice

1. Select the root of a set of elements, or a root and one or more child entities. The child entities within that root can be individual elements, slices, or a combination of both.
2. Right-click the selection. Click **Make Detached Slice from Selected Entities**.

   Note
   If the any of the elements that you selected are slice instances, two options appear: **Make Detached Slice from Selected Entities** and **Make Cascaded Slice from Selected Slices & Entities**.
3. Save the slice with a descriptive name.

Modifying a UI Slice and Pushing Changes

In the **UI Editor** you modify UI slices and push changes the same way that you do in the main Lumberyard Editor. For more information on pushing changes in Lumberyard's component entity system, see Working with Slices (p. 597).

If the currently selected entity is part of a slice instance, the **Properties** pane highlights in orange any properties that are different between the selected entity and the slice to which it belongs.

In addition, the UI system enforces a restriction on slice changes: A UI slice cannot contain references to any entities not within the slice.

To push local changes to the slice

1. Right-click in the **Properties** pane or on selected UI elements in the **Hierarchy** pane.
2. In the context menu, click **Push to Slice**.

Flagging Dynamic Slices

A slice flagged as a dynamic slice can be used like any other slice, but it can also be instantiated at runtime.

You can flag a slice as dynamic in the **Asset Browser**. To do this, right-click on the `.slice` file in the **Asset Browser** and click **Set Dynamic Slice**.

To instantiate a UI slice at runtime, use the **UiSpawner** component on a UI element. This causes the system to automatically spawn a dynamic slice on activation. It also exposes a bus to Lua and C++ that allows the slice to be instantiated whenever needed.

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. 1558), such as a scroll box, which is an element with components already attached. You can also create your own prefab elements (p. 1558).

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. 1563) (image or text), one interactive component (p. 1567) (button, check box, scroll box, slider, or text input), and one layout component (p. 1590) (layout column, layout row, or layout grid). The remaining components (p. 1610) 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. 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.

Task	Steps
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. 1560).

Using and Creating UI Prefabs

This system has been deprecated and will be removed in a future release.

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 **(Deprecated) 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**, **(Deprecated) 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, scroll box, slider, or text input), and one layout component (layout column, layout row, or layout grid). You can attach any or all of the remaining components: dynamic components, tooltips, tooltips display, mask, and fader.

Topics

Adding or Deleting Components

You can easily add or delete components in the **UI Editor** (p. 1539).

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, check box, slider, text input, scroll box, 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

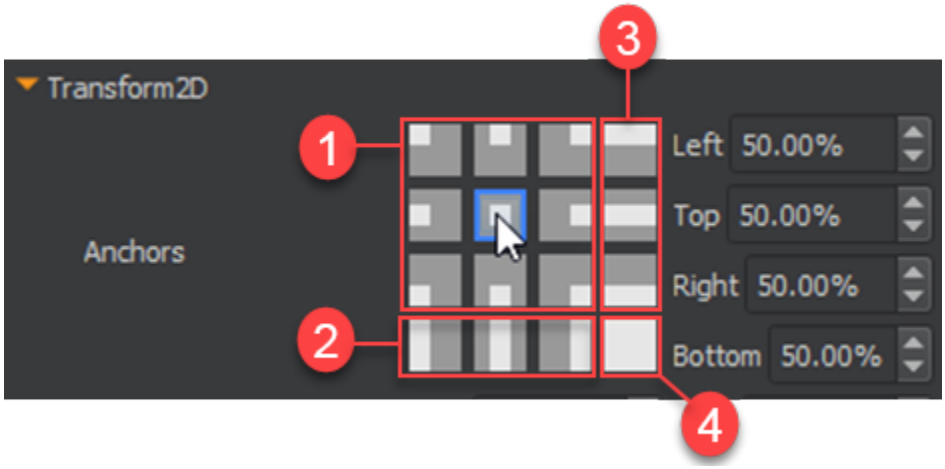
1. In the **Hierarchy** pane of the **UI Editor** (p. 1539), select the element whose anchors you want to modify.
2. In the **Properties** pane, under **Transform2D**, choose from the selection of commonly used anchor placements.

    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).



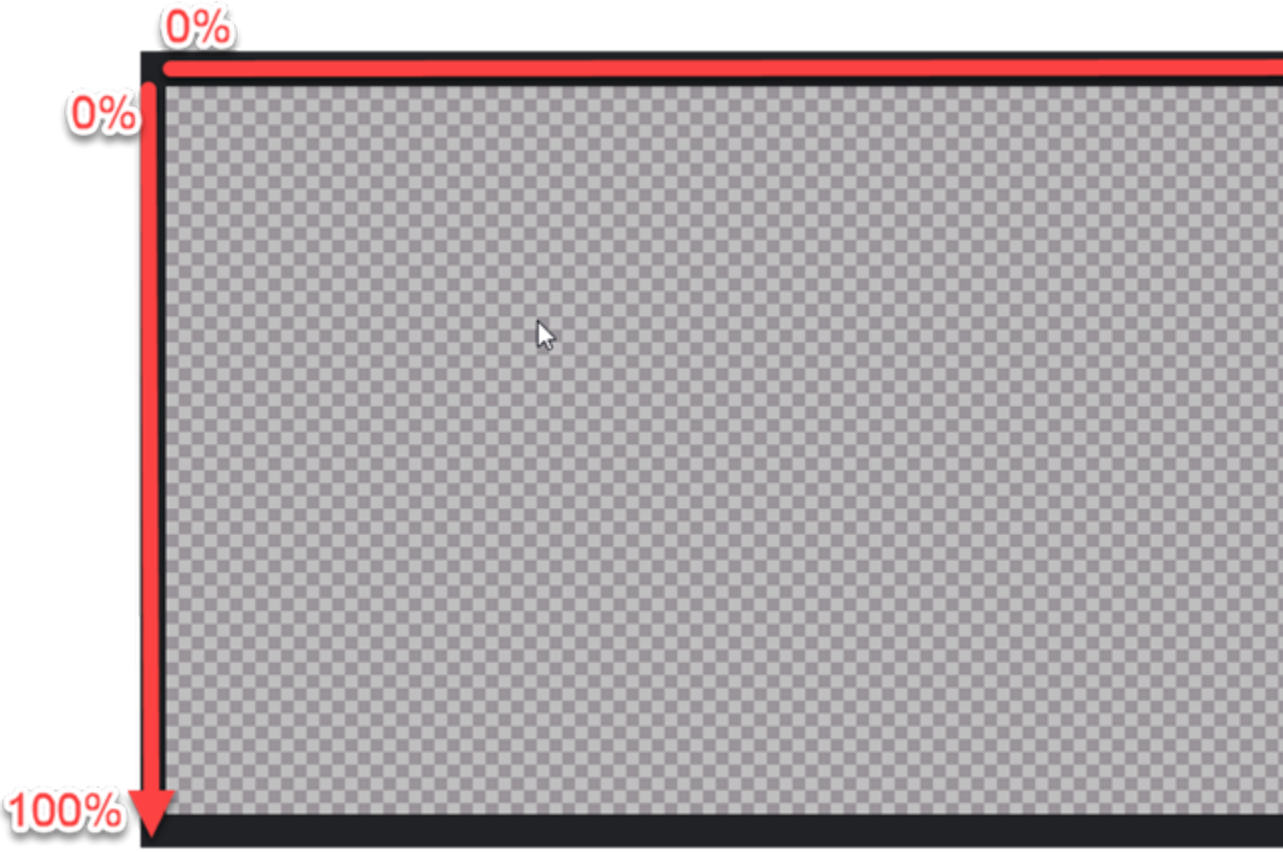
To further edit (fine-tune) an element's anchors

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 in-game examples of completed canvases with image components, open the level UiFeatures in the project SamplesProject. Press **Ctrl+G** to play the game, and then choose **Components**, **Visual Components**, **Image**. You can view examples of image types such as sliced, stretched, fixed, tiled, stretched to fit, and stretched to fill. Press **Esc** to exit the game.

To view these same canvases in the **UI Editor**, navigate to the `\Gems\LyShineExamples\Assets\UI\Canvases\LyShineExamples\Comp\Image` directory.

You can open the following canvases:

- `ColorTest.uicanvas`
- `ImageTypes.uicanvas`

To edit an image component

In the **Properties** pane of the **UI Editor** (p. 1539), expand **Image** and do the following, as appropriate:

SpriteType

Select one of the following:

- **Sprite/Texture asset** – Image displays the asset specified for **Sprite path**.
- **Render target** – Image displays the render target specified in **Render target name**.

Sprite path

Click the browse (**...**) 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 in-game examples of completed canvases with text components, open the level UiFeatures in the project SamplesProject. Press **Ctrl+G** to play the game, and then choose **Components**, **Visual Components**, **Text**. You can view examples of text alignment, color and alpha, styling markup, overflow and wrapping, and character and line spacing. Press **Esc** to exit the game.

To view these same canvases in the **UI Editor**, navigate to the `\Gems\LyShineExamples\Assets\UI\Canvases\LyShineExamples\Comp\Text` directory.

You can open the following canvases:

- `Alignment.uicanvas` – Examples of aligning the text within its element rectangle
- `ColorAlpha.uicanvas` – Examples of setting different colors and transparency levels

- `Markup.uicanvas` – Examples of customizing the appearance of text using different stylings, colors, and font within a single string
- `OverflowWrap.uicanvas` – Examples of different wrap and overflow options
- `Text.uicanvas` – Examples of different wrap and overflow options

To edit a text component

In the **UI Editor** (p. 1539) **Properties** pane, expand **Text** and do the following, as appropriate:

Text

Type the preferred text string and press **Enter**. Here, you can apply text styling markup (p. 1565).

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 `.font` file. For more information, see Adding New Fonts (p. 1612).

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 `.font` 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 type 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 `.font` asset file). For more information about adding font families to your projects, see Implementing New Fonts (p. 1612).

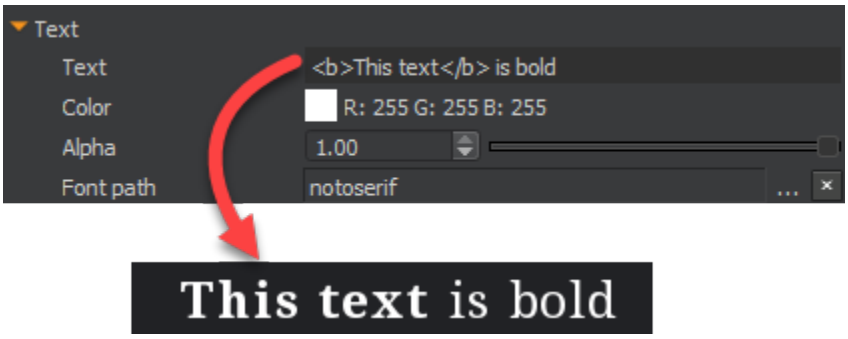
To use text styling markup

1. In the **UI Editor** (p. 1539), 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. Type a string with markup styling in the **Text** box. See the next section for examples.

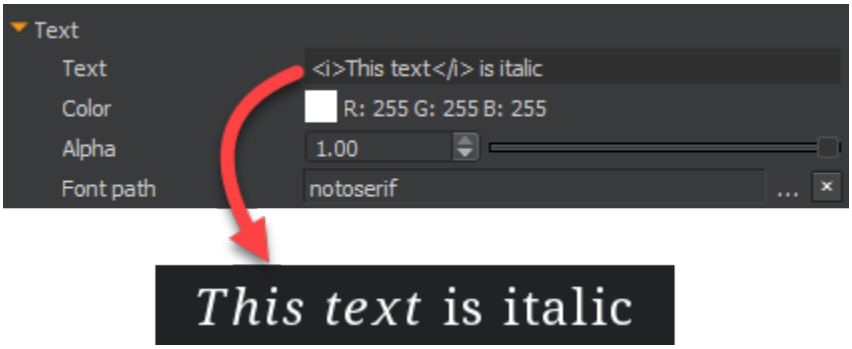
Tags and Attributes

You can use the following tags and attributes when styling text with markup:

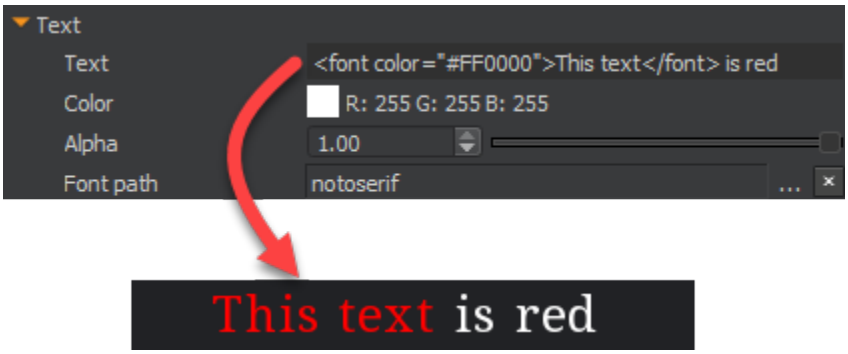
Bold tag: <b>



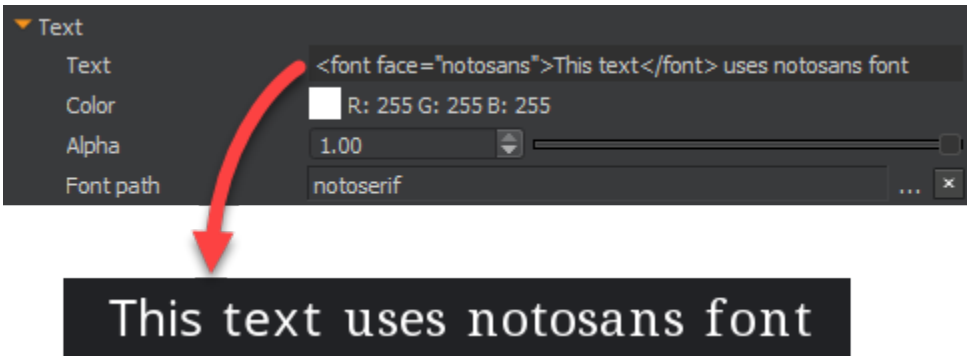
Italic tag: <i>



Font color tag: <font color>



Font face tag: <font face>



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. 1550) or flow graphs (p. 1626) to link the component response to an action.

An interactive element is defined as an element that has an interactive component applied.

Topics

Properties

All of the interactive components share a common set of properties. These properties are grouped into the following categories:

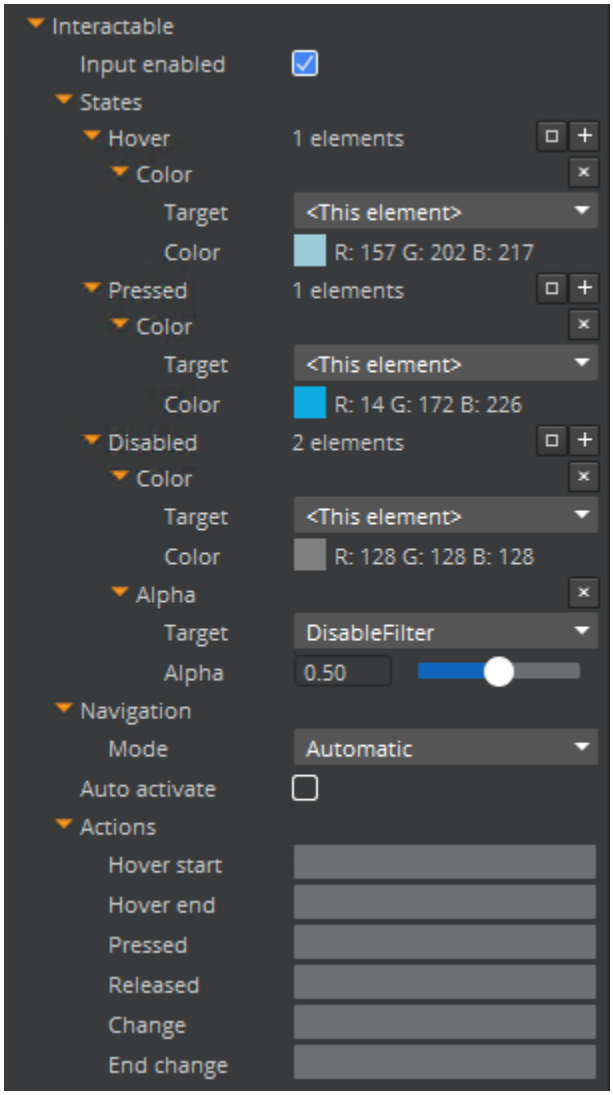
- **Input Enabled** (p. 1568) – Check box or flag that determines whether the element can be interacted with.

- **States** (p. 1569) – Settings that determine the appearance of the element when in the **Hover**, **Pressed**, or **Disabled** states.

- **Navigation** (p. 1570) – Settings that determine how the gamepad or arrow keys navigate between interactive elements (p. 1570).

- **Auto Activate** (p. 1571) – Check box or flag that determines whether the element is automatically activated when the player pauses on the element.

- **Actions** (p. 1571) – 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. 1542) 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. 1724).

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.

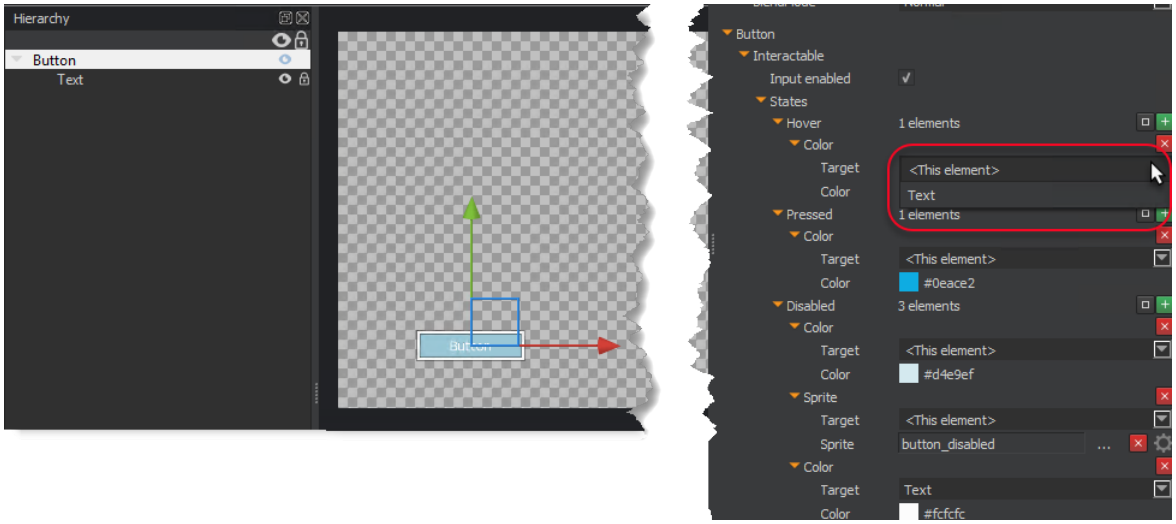
The normal appearance of a visual element (defined as an element with a visual component, such as image or text) is defined by the properties of that visual component. Some of the visual component's properties, however, can be overridden by an interactive component that is in the **Hover**, **Pressed**, 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 of the visual component:

- **Color** – RGB color tint
- **Alpha** – Opacity
- **Sprite** – Texture
- **Font** – Text font and font effect (of 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. 1558) has a top element named **Button** that has a visual component to define its color. It also has a child element with a text component to define the text (and its color) of the button. The top element (**Button**) also has the **Interactable** component. 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 select from the list.

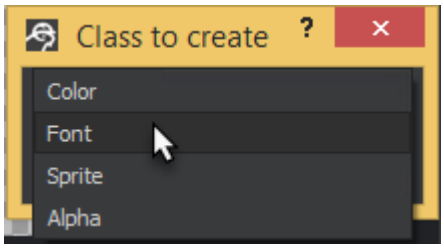


When you first add an interactive component to an element, no **state actions** are added by default. You must add state actions to the states that you want to use and modify.

To add a state action to a state

In the **UI Editor** (p. 1539), 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

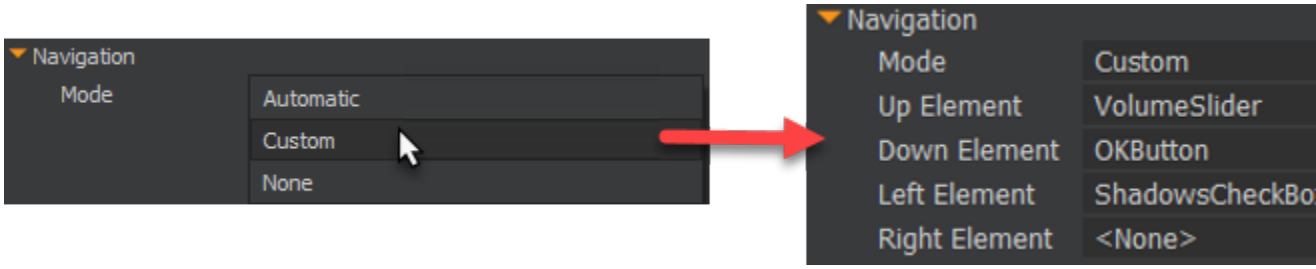
- Click **Clear container** (box icon) next to the state from which you want to clear all the state actions.

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. 1545). 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 has focus until the user provides direction input from a keyboard or mouse. At that point, the element closest to the top left corner of the canvas receives focus.

Once an element has focus, navigation to other elements is controlled by the navigation properties (p. 1570) defined for each interactive component.

Interactive Element Controls

To interact with a focused element, press **Enter** on the keyboard).

When an element is interactive, use the following controls:

- **Button** and **Checkbox** – **Enter** presses the button or selects or clears the check box, and then returns to navigation automatically. The control does not remain active after action.

- **Slider** and **Scrollbox** – 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.
  - Alphanumeric keys type text at the 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.

Auto Activate

The **Auto Activate** setting applies to interactive elements that remain active after they are pressed. These include the slider, scroll bar, scroll box, and text input.

When Auto Activate is set, an element automatically activates when navigated to with the directional keys, and deactivates when navigated away from with the directional keys. The user does not need to press the **Enter** and **Back** keys to activate or deactivate the element.

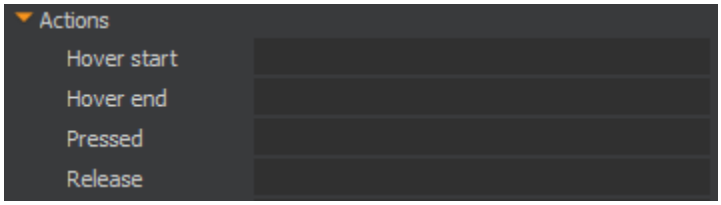
The UI must be designed so that a user cannot get stuck on an activated element. For example, a horizontal slider that is set to automatically activate should not have another interactive element to the left or right of it. This is because pressing the left or right key on an active slider moves the slider handle.

Actions

You can use the **Actions** properties to trigger a particular event when one of the listed actions occur. Type a string for one of the actions. When the listed action occurs (for example, when a game player starts to pause on the element), the listed string is sent as an action. You can set up the UIe:Canvas:ActionListener (p. 1627) flow graph node to listen for this action.

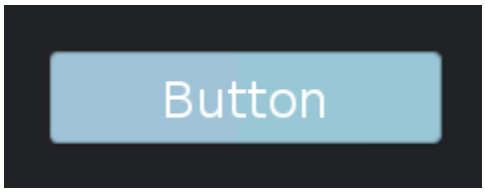
You can type strings for the following actions:

- **Hover start**
- **Hover end**
- **Pressed**
- **Release**



Button

You can use a **Button** component to make an element behave like a button.



To see an in-game example of a completed canvas with the **Button** component, open the level UiFeatures in the project SamplesProject. Press **Ctrl+G** to play the game, and then choose **Components**, **Interactable Components**, **Button**. You can view the different types of buttons you can create. Press **Esc** to exit the game.

To view this same canvas in the **UI Editor**, open `\Gems\LyShineExamples\Assets\UI\Canvases\LyShineExamples\Comp\Button\Styles.uicanvas`.

Note the following:

- This component is typically applied to an element with an image component; if no visual or image component is present, many of the button's 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.

You can add a prebuilt **Button** element from the slice library. When you do this, a basic button with the text string "Button" is automatically created in your **Hierarchy** pane.

To add a Button element from the slice library

- In the **UI Editor** (p. 1539), choose **New**, **Element from Slice Library**, **Button**.

To edit a button component

In the **UI Editor** (p. 1539) **Properties** pane, expand **Button** and do the following, as appropriate:

Interactable

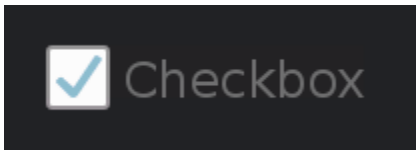
See Properties (p. 1567) to edit the common interactive component settings.

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. 1702).

Checkbox

You can use the **Checkbox** component to make an element behave like a check box. This component is typically applied to an element with two visual child elements: One element appears when the check box is selected and another appears when the check box is cleared.



To see in-game examples of completed canvases with the **Checkbox** component, open the level UiFeatures in the project SamplesProject. Press **Ctrl+G** to play the game, and then choose **Components**, **Interactable Components**, **CheckBox**. You can view examples of color targets, check box interaction areas, and on and off elements. Press **Esc** to exit the game.

To view these same canvases in the **UI Editor**, navigate to the `\Gems\LyShineExamples\Assets\UI\Canvases\LyShineExamples\Comp\CheckBox` directory. You can open the following canvases:

- `Area.uicanvas` – Examples of different check box interaction areas
- `ColorTargets.uicanvas` – Examples of different color targets when interacting with the check box
- `OnOff.uicanvas` – Examples of different check box on and off elements

You can add a prebuilt **Checkbox** element from the slice library. When you do this, a basic check box with the text string "Checkbox" and a check image for the box is automatically created in your **Hierarchy** pane.

To add a Checkbox element from the slice library

- In the **UI Editor** (p. 1539), choose **New**, **Element from Slice Library**, **Checkbox**.

To edit a check box component

In the **Properties** pane of the **UI Editor** (p. 1539), expand **Checkbox** and do the following, as appropriate:

Interactable

See Properties (p. 1567) to edit the common interactive component settings.

Elements, On

Select an element from the list to specify the entity that appears when the check box state is `on` (selected).

Elements, Off

Select an element from the list to specify the entity that appears when the check box state is `off` (cleared).

Value, Checked

Select 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 the UI:Canvas:ActionListener Node (p. 1702).

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 the UI:Canvas:ActionListener Node (p. 1702).

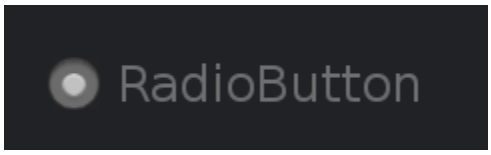
Actions, Off

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 the UI:Canvas:ActionListener Node (p. 1702).

RadioButton

You can use the **RadioButton** component to make an element behave like a radio button. This component is typically used on an element with two visual child elements—one to display when the radio button is selected and another to display when the radio button is cleared.

Use this component in conjunction with the **RadioButtonGroup** component. The **RadioButtonGroup** component handles selecting and clearing the radio buttons in the group and makes surethat only one radio button is ever selected.



To see in-game examples of completed canvases with the **RadioButton** component, open the level UiFeatures in the project SamplesProject. Press **Ctrl+G** to play the game, and then choose **Components**, **Interactable Components**, **RadioButton**. You can view examples of different behaviors, default settings, and groups for radio buttons Press **Esc** to exit the game.

To view these same canvases in the **UI Editor**, navigate to the `\Gems\LyShineExamples\Assets\UI\Canvases\LyShineExamples\Comp\RadioButton` directory. You can open the following canvases:

- `Groups.uicanvas` – Examples of different radio button groupings
- `RadioButton.uicanvas` – Examples of different behaviors and default settings

To edit a radio button component

In the **Properties** pane of the **UI Editor** (p. 1539), expand **RadioButton** and do the following, as appropriate:

Interactable

See Properties (p. 1567) to edit the common interactive component settings.

Elements, On

Select an element from the list to specify the entity that appears when the radio button state is on (selected).

Elements, Off

Select an element from the list to specify the entity that appears when the radio button state is off (cleared).

Elements, Group

Select an element from the list to specify the group the radio button belongs to.

Value, Checked

Select the box to change the initial state of the radio button.

Actions, Change

Type a text string. This string is sent as an action on the UI canvas when the radio button has any state changes. You can listen for this action in the flow graph using the UI:Canvas:ActionListener Node (p. 1702).

Actions, On

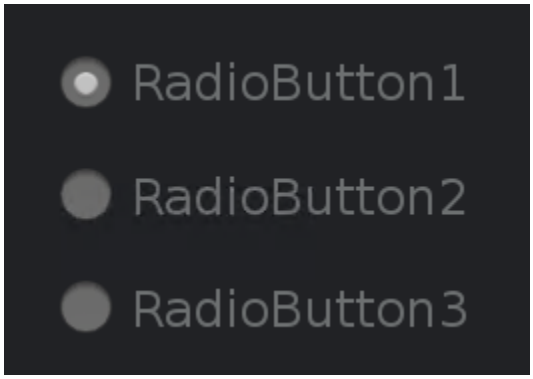
Type a text string. This string is sent as an action on the UI canvas when the radio button state changes to on (selected). You can listen for this action in the flow graph using the UI:Canvas:ActionListener Node (p. 1702).

Actions, Off

Type a text string. This string is sent as an action on the UI canvas when the radio button state changes to off (cleared). You can listen for this action in the flow graph using the UI:Canvas:ActionListener Node (p. 1702).

RadioButtonGroup

You can use the **RadioButtonGroup** component to manage radio buttons. This component handles selecting and clearing the radio buttons in the group when appropriate. It also ensures that only one radio button is selected at one time. You typically use this component on an element with children radio buttons, which are part of the radio button group.



To see in-game examples of completed canvases with the **RadioButtonGroup** component, open the level UiFeatures in the project SamplesProject. Press **Ctrl+G** to play the game, and then choose **Components**, **Interactable Components**, **RadioButton**. You can view examples of different behaviors, default settings, and groups for radio buttons Press **Esc** to exit the game.

To view these same canvases in the **UI Editor**, navigate to the `\Gems\LyShineExamples\Assets\UI\Canvases\LyShineExamples\Comp\RadioButton` directory. You can open the following canvases:

- `Groups.uicanvas` – Examples of different radio button groupings
- `RadioButton.uicanvas` – Examples of different behaviors and default settings

You can add a prebuilt **RadioButtonGroup** element from the slice library. When you do this, a group of three radio buttons is automatically created in your **Hierarchy** pane.

To add a RadioButtonGroup element from the slice library

- In the **UI Editor** (p. 1539), choose **New**, **Element from Slice Library**, **RadioButtonGroup**.

To edit a RadioButtonGroup component

In the **Properties** pane of the **UI Editor** (p. 1539), expand **RadioButtonGroup** and do the following, as appropriate:

Settings, Allow uncheck

Select to enable the clearing or unchecking of selected radio buttons.

Actions, Change

Type a text string. This string is sent as an action on the UI canvas when the radio button group has any state changes. You can listen for this action in the flow graph using the UI:Canvas:ActionListener Node (p. 1702).

Slider

You can use the **Slider** component to make an element behave like a slider. This component is typically applied to 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 in-game examples of completed canvases with the **Slider** component, open the level UiFeatures in the project SamplesProject. Press **Ctrl+G** to play the game, and then choose **Components**, **Interactable Components**, **Slider**. You can view examples of different slider behavior and positioning. Press **Esc** to exit the game.

To view these same canvases in the **UI Editor**, navigate to the `\Gems\LyShineExamples\Assets\UI\Canvases\LyShineExamples\Comp\Slider` directory. You can open the following canvases:

- `Behavior.uicanvas` – Examples of slider behaviors such as fill type, handles, and buttons to control the slider
- `Rotation.uicanvas` – Example of slider positioning

You can add a prebuilt **Slider** element from the slice library. When you do this, the slider's track, fill, and handle are automatically created in your **Hierarchy** pane.

To add a Slider element from the slice library

- In the **UI Editor** (p. 1539), choose **New**, **Element from Slice Library**, **Slider**.

To edit a slider component

In the **Properties** pane of the **UI Editor** (p. 1539), expand **Slider** and do the following, as appropriate:

Interactable

See Properties (p. 1567) to edit the common interactive component settings.

Elements, Track

Select an element from the list to provide the background of the slider and to limit the movement of the manipulator.

Elements, Fill

Select an element from the list to provide 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 movable knob of the slider.

Value, Value

Type the initial value of the slider.

Value, Min

Type the lower limit of the slider.

Value, Max

Type the upper limit of the slider.

Value, Stepping

Type the step value. For example, use `1` to only permit whole integer values.

Actions, Change

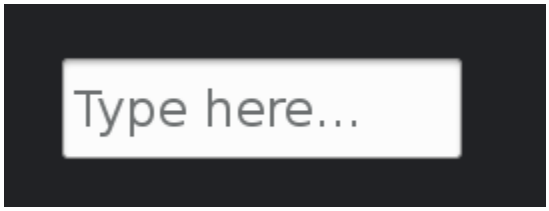
Type a text string. This string is sent as an action on the UI canvas when the slider has finished changing values. You can listen for this action in the flow graph using the UI:Canvas:ActionListener Node (p. 1702).

Actions, End Change

Type a text string. This string is sent as an action on the UI canvas when the slider has changing values. You can listen for this action in the flow graph using the UI:Canvas:ActionListener Node (p. 1702).

TextInput

You can use a **TextInput** component to make an element offer player input. This component is typically applied to an element with an image component and two child elements with text components (one for placeholder text and one for input text).



To see in-game examples of completed canvases with the **TextInput** component, open the level UiFeatures in the project SamplesProject. Press **Ctrl+G** to play the game, and then choose **Components**, **Interactable Components**, **TextInput**. You can view examples of different types of text input behavior on single lines and on multiple lines. Press **Esc** to exit the game.

To view these same canvases in the **UI Editor**, navigate to the `\Gems\LyShineExamples\Assets\UI\Canvases\LyShineExamples\Comp\TextInput` directory. You can open the following canvases:

- `Multiline.uicanvas` – Examples of editing multiline text strings
- `SingleLine.uicanvas` – Example of editing single-line text strings

You can add a prebuilt **TextInput** element from the slice library. When you do this, a text input box, pause-on state, and placeholder text "Type here..." are automatically created in your **Hierarchy** pane.

To add a TextInput element from the slice library

- In the **UI Editor** (p. 1539), choose **New**, **Element from Slice Library**, **TextInput**.

To edit a text input component

In the **Properties** pane of the **UI Editor** (p. 1539), expand **TextInput** and do the following, as appropriate:

Interactable

See Properties (p. 1567) 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 the selected text.

Text editing, Cursor color

Click the color swatch to select a different color for the cursor.

Text editing, Max char count

Type the maximum number of characters allowed in the text input box. Type `-1` for no character limit.

Text editing, Cursor blink time

Type 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

Select 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**.

ScrollBar

You can use a **ScrollBar** component to add a scrollable bar, or handle, for manipulating settings or scrolling within a scroll box.

To see in-game examples of completed canvases with the **ScrollBar** component, open the level UiFeatures in the project SamplesProject. Press **Ctrl+G** to play the game, and then choose **Components**, **Interactable Components**, **ScrollBar**. You can view examples of different types of scroll bar positioning and handles, scroll bars paired with scroll boxes, and visibility options. Press **Esc** to exit the game.

To view these same canvases in the **UI Editor**, navigate to the `\Gems\LyShineExamples\Assets\UI\Canvases\LyShineExamples\Comp\ScrollBar` directory. You can open the following canvases:

- `ScrollBoxes.uicanvas` – Examples of scroll bars paired with scroll boxes
- `Simple.uicanvas` – Examples of scroll bar with simple logic
- `Visibility.uicanvas` – Examples of scroll bar visibility options when paired with scroll boxes

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.

You can add a prebuilt **ScrollBarHorizontal** or **ScrollBarVertical** element from the slice library. When you do this, the scroll bar and its handle is automatically created in your **Hierarchy** pane.

To add a ScrollBar element from the slice library

- In the **UI Editor** (p. 1539), choose **New**, **Element from Slice Library**, **ScrollBarHorizontal** or **ScrollBarVertical**.

To edit a scroll bar component

In the **Properties** pane of the **UI Editor** (p. 1539), expand **ScrollBar** and do the following, as appropriate:

Interactable

See Properties (p. 1567) to edit the common interactive component settings.

Elements, Handle

Select an element from the list to provide the movable handle of the scroll bar.

Values, Orientation

Select the scroll bar's orientation:

- **Horizontal** – Scrollbar's handle moves left and right.
- **Vertical** – Scrollbar's handle moves up and down.

Values, Value

Type the initial value of the scroll bar (`0.0` to `1.0`).

Values, Handle size

Type the size of the handle relative to the scroll bar (`0.0` to `1.0`).

Values, Min handle size

Type 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 the UIe:Canvas:ActionListener Node (p. 1627).

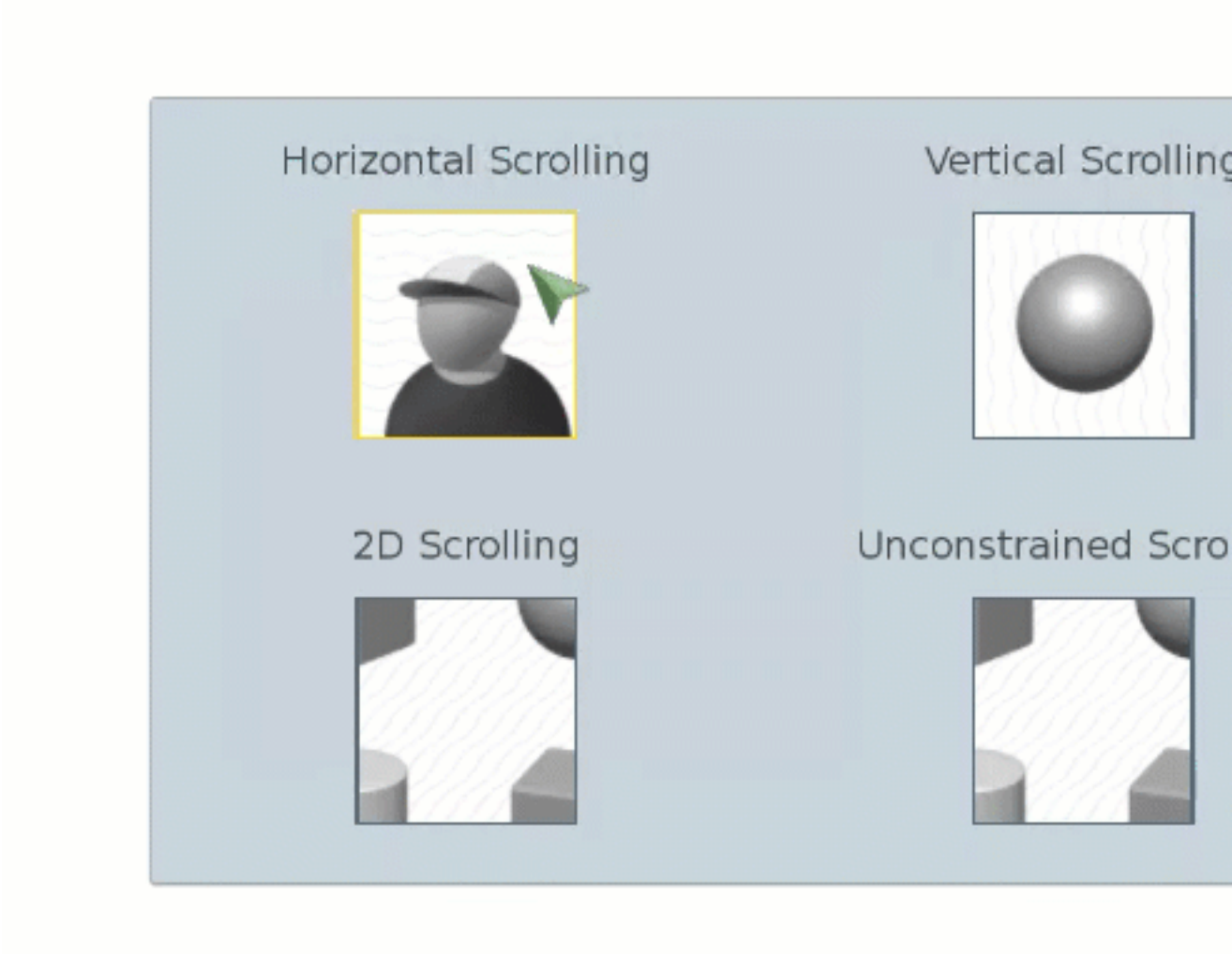
Actions, End Change

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 the UIe:Canvas:ActionListener Node (p. 1627).

ScrollBox

You can use a **ScrollBox** component to present content, such as images or text, within a scrollable area.

This component is typically used with a mask component, which hides the content outside of the masked area.



To see in-game examples of completed canvases with the **ScrollBox** component, open the level UiFeatures in the project SamplesProject. Press **Ctrl+G** to play the game, and then choose **Components**, **Interactable Components**, **ScrollBox**. You can view examples of different scrolling options, snapping options, interactions between scroll boxes and other components, and nested scroll boxes. Press **Esc** to exit the game.

To view these same canvases in the **UI Editor**, navigate to the `\Gems\LyShineExamples\Assets\UI\Canvases\LyShineExamples\Comp\ScrollBox` directory. You can open the following canvases:

- `Interactions.uicanvas` – Examples of interactions between scroll boxes and other interactive components
- `Nested.uicanvas` – Examples of nesting scroll boxes
- `Scrolling.uicanvas` – Examples of different scrolling options such as horizontal, vertical, 2D, and unconstrained
- `Snapping.uicanvas` – Examples of different snapping options

You can add a prebuilt **ScrollBox** element from the slice library. 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 the slice library

- In the **UI Editor** (p. 1539), choose **New**, **Element from Slice Library**, **Scrollbox**.

The element named **ScrollBox** (1) has the **ScrollBox** component (2) applied to 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. 1560) component.

The element named **Mask** has a **Mask** (p. 1611) component applied to 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 of the **UI Editor** (p. 1539), expand **ScrollBox** and do the following, as appropriate:

Interactable

See Properties (p. 1567) 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

Type 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 the dragging stops.
- **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 scroll bar associated with the scroll box.

Horizontal scrolling, Scrollbar visibility

Select the visibility behavior of the horizontal scroll bar:

- **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 scroll bar associated with the scroll box.

Vertical scrolling, Scrollbar visibility

Select the visibility behavior of the vertical scroll bar:

- **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 dragging each time the position changes.

Actions, End change

Set the action that is triggered when a drag motion is completed.

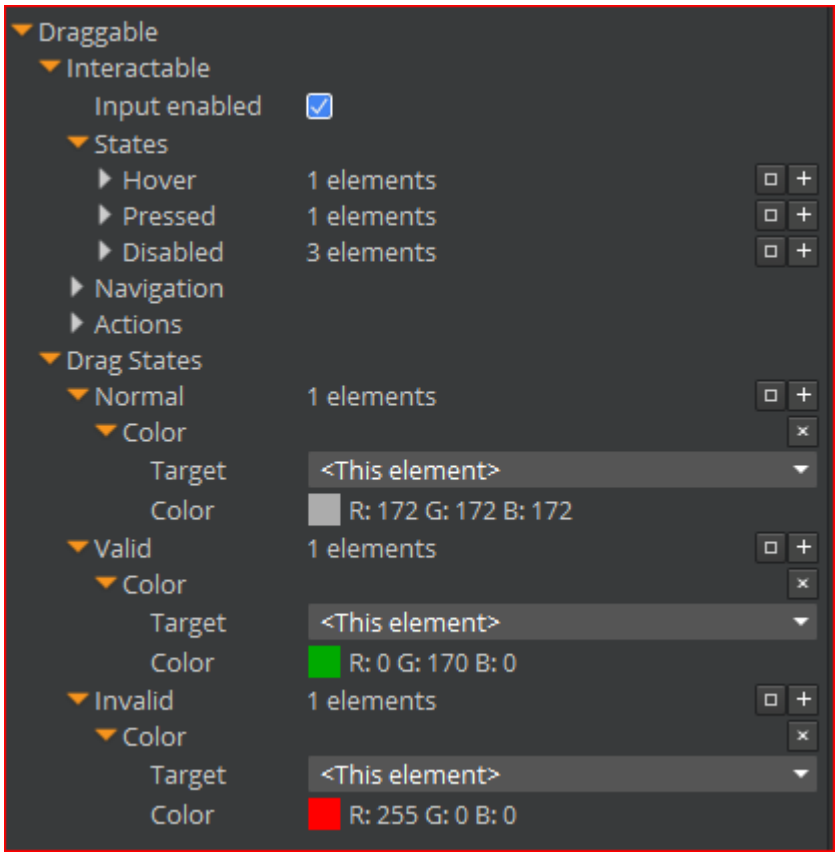
Draggable

You can use the **Draggable** component to make a UI element movable from one location to another on screen. Use the **Draggable** component in combination with the **DropTarget** component so that dragging can start on the draggable element and end on the drop target element. Drag-and-drop is a common operation in UI screens, such as an inventory system.

Because drag-and-drop behavior is game specific, the **Draggable** and **DropTarget** components are designed to be used with scripting or C++ to define actions that result from the drag and the drop.

To add the draggable component to a UI element, use the **Add Component** menu in the **Properties** pane.

The following picture shows an example of a **Draggable** component, where color has been added to the state actions for **Drag States**.



The **Draggable** component is an interactive component. It has the standard interactive properties (p. 1567).

To edit a Draggable component

In the **Properties** pane of the **UI Editor** (p. 1539), expand **Draggable** and do the following, as appropriate:

Interactable

See Properties (p. 1567) to edit the common interactive component settings.

Drag States

Define the color, alpha, sprite, or font of this element and its child elements for a particular drag state.

When it is not being dragged, an element with a draggable component uses the interactable states (pause on, pressed, and disabled).

When being dragged, however, the draggable component has an additional three states:

- **Normal** – Automatic state when a drag state begins.

- **Valid** – Typically the state used when the draggable component pauses on a valid drop target. This state is determined by a script that you write or C++ code that connects to the `UiDropTargetNotificationBus` and listens for the `OnDropHoverStart` method.

- **Invalid** – Typically the state used when a draggable component pauses over an invalid drop target. This state is determined by a script that you write or C++ code. When a valid drag state is triggered, a notification is automatically sent using the `UiDropTargetNotificationBus`.

The script or C++ can use the `UiDraggableBus` to set the drag state of the **Draggable** component. It can also set the drop state of the **DropTarget** to indicate valid drop targets to the user.

To see an example of simple drag Lua script, open `DraggableElement.lua` in `Gems\LyShineExamples\Assets\UI\Scripts\LyShineExamples\DragAndDrop`.

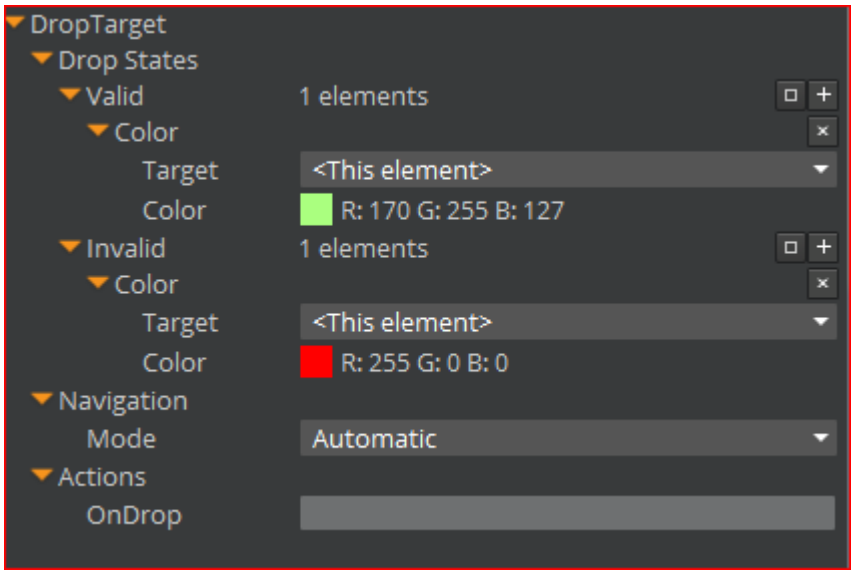
DropTarget

You can use the **DropTarget** component to implement drag-and-drop behavior with the **Draggable** component.

Because drag-and-drop behavior is game specific, the **Draggable** and **DropTarget** components are designed to be used with scripting or C++ to define actions that result from the drag and the drop.

To add the **DropTarget** component to a UI element, use the **Add Component** menu in the **Properties** pane.

The following picture shows an example of a **DropTarget** component, where color has been added to the state actions for **Drop States**.



The **DropTarget** component shares properties with interactive components, such as state actions and navigation settings.

To edit a DropTarget component

In the **Properties** pane of the **UI Editor** (p. 1539), expand **DropTarget** and do the following, as appropriate:

Drop States

Define the color, alpha, sprite, or font of this element and its child elements in a valid or invalid drop state. By default, the drop state of a drop target is normal, which means that there are no visual overrides.

During a drop, the drop target component can be **Valid** or **Invalid**. Because the drop target component has no knowledge of what is a valid drag-and-drop operation, you use a script or C++ to switch the drop target into the **Normal**, **Valid**, and **Invalid** states. This is usually accomplished by connecting to the `UiDropTargetNotifications` bus and listening for the `OnDropHoverStart` and `OnDropHoverEnd` notifications.

Navigation

Navigation settings control how keyboard or gamepad navigation works during a drag-and-drop operation. When using the keyboard, you can press **Enter** on a draggable element to enter drag mode. Then you can use the arrow keys to move the element from one drop target to another using the navigation settings specified here.

Actions, On Drop

Type a text string. This string is sent as an action on the UI canvas whenever a draggable is dropped on this drop target. For better control, we recommend that you use the `UiDropTargetNotifications` bus instead.

To see an example of simple drop target Lua script, open `DropTarget.lua` in `Gems\LyShineExamples\Assets\UI\Scripts\LyShineExamples\DragAndDrop`.

Dropdown

You can use the **Dropdown** component to make an element behave like a drop-down menu. Use this component with child elements that contain content. The child elements provide the contents of the drop-down menu.

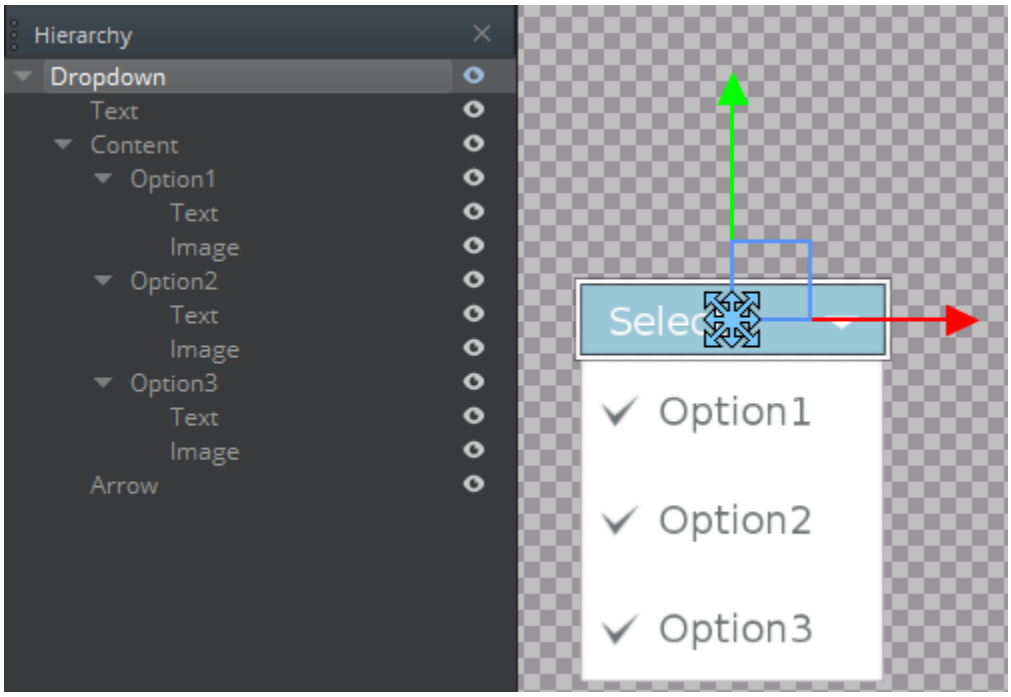
You can use the **Dropdown** component with the **DropdownOption** component. With the **DropdownOption** component, you can configure options to change the menu text and its icon after it is selected.

To see in-game examples of completed canvases with the **Dropdown** component, open the level UiFeatures in the project SamplesProject. Press **Ctrl+G** to play the game, and then choose **Components**, **Interactable Components**, **Dropdown**. You can view **Simple Dropdowns**, **Nested Dropdowns**, **Multiple Select & Functionality**, and **Using UI Components and Dropdowns**. Press **Esc** to exit the game.

To view these same canvases in the **UI Editor**, navigate to the `\Gems\LyShineExamples\Assets\UI\Canvases\LyShineExamples\Comp\Dropdown` directory. You can open the following canvases:

- `MultipleFunc.uicanvas` – Multiple selection drop-down menu and functional drop-down menu (perform actions on a ball such as create, destroy, move, and change color)
- `Nested.uicanvas` – Two levels and multilevel with siblings submenus
- `Simple.uicanvas` – Simple selection drop-down menu, selection drop-down menu with icons, expand on pause drop-down menu, and expand on pause with icons
- `UsingUi.uicanvas` – Drop-down menus with a scroll box, image, check box, slider, and radio buttons

You can add a prebuilt **Dropdown** element from the slice library. When you do this, a drop-down menu, three options, and their image elements are automatically created in your **Hierarchy** pane.



To add a Dropdown element from the slice library

- In the **UI Editor** (p. 1539), choose **New**, **Element from Slice Library**, **Dropdown**.

To edit a Dropdown component

In the **Properties** pane of the **UI Editor** (p. 1539), expand **Dropdown** and do the following, as appropriate:

Interactable

See Properties (p. 1567) to edit the common interactive component settings.

Elements, Content

Select an element from the list. This specifies the entity that appears when the drop-down menu is expanded.

Elements, Expanded Parent

Drag an element onto the **Expanded Parent** box. This specifies the entity that serves as the parent when the drop-down menu is expanded. This is used to layer drop-down menus.

Elements, Text Element

Select an element from the list to specify the entity to display the text corresponding to the selected option.

Elements, Icon Element

Select an element from the list to specify the entity to display the icon corresponding to which option is selected.

Options, Expand on Hover

Select to enable drop-down behavior upon pause and collapse upon exit.

Options, Wait Time

Enter a number of seconds that the drop-down menu waits before expanding on pause or collapsing on exit.

Options, Collapse on Outside Click

Select to enable drop-down menu collapse upon clicking outside the menu.

Dropdown States, Expanded

Click plus to add states to the drop-down menu when it is expanded.

Actions, Expanded

Type a text string to be sent as an action on the UI canvas when the drop-down menu is expanded.

Actions, Collapsed

Type a text string to be sent as an action on the UI canvas when the drop-down menu is collapsed.

Actions, Option Selected

Type a text string to be sent as an action on the UI canvas when a drop-down option is selected.

DropdownOption

You can use the **DropdownOption** component to make an element an option in a drop-down menu. When using the **DropdownOption** component, note the following:

- The **DropdownOption** is used along with the **Dropdown** component on an element.
- The **DropdownOption** requires an interactive component, typically a **RadioButton** component so that only one option can be selected at any time in the drop-down menu.
- Its child elements typically contain the **Text** component, which appears when the option is selected.

To see an in-game example of a completed canvas with the **DropdownOption** component, open the level UiFeatures in the project SamplesProject. Press **Ctrl+G** to play the game, and then choose **Components**, **Interactable Components**, **Dropdown**, **Using UI Components and Dropdowns**. Press **Esc** to exit the game.

To view this same canvas in the **UI Editor**, navigate to the `\Gems\LyShineExamples\Assets\UI\Canvases\LyShineExamples\Comp\Dropdown` directory and open the `UsingUi.uicanvas` file. This canvas features a drop-down menu with a scroll box, image, check box, slider, and radio buttons.

To edit a DropdownOption component

In the **Properties** pane of the **UI Editor** (p. 1539), expand **DropdownOption** and do the following, as appropriate:

Interactable

See Properties (p. 1567) to edit the common interactive component settings.

Elements, Owning Dropdown

Select an element from the list that has the **DropdWown** component. This is the element to be modified when this option is selected.

Elements, Text Element

Select an element from the list that displays the text corresponding to this option. This text is displayed on the owning drop-down menu when this option is selected (as long as the drop-down menu has a **Text Element** configured).

Elements, Icon Element

Select an element from the list that displays the icon corresponding to this option. This icon is displayed on the owning drop-down menu when this option is selected (as long as the drop-down menu has an **Icon Element** configured).

Layout Components

Like other UI components, layout components define the layout properties of your game's interface. The UI system features four layout components to organize your elements: **LayoutColumn**, **LayoutRow**, **LayoutGrid**, and **LayoutCell**. You can also nest layout components.

Topics

LayoutColumn

Add the **LayoutColumn** component to an element to make it a layout column. When you add child elements to the layout column, the layout column assigns each child element a layout cell. The layout column adjusts the size of the layout cells depending on how many child elements you add as well as the values provided by the child elements' layout cells.

LayoutRow

Add the **LayoutRow** component to an element to make it a layout row. Like the layout column, the layout row assigns each child elements a layout cell. The layout row adjusts the size of the layout cells depending on how many child elements you add and the values provided by the child elements' layout cells.

LayoutGrid

Add the **LayoutGrid** component to an element to make it a layout grid. The layout grid places child elements into a grid. Unlike the layout row and layout column, however, the layout grid does not use layout cells. The **LayoutGrid** component's properties determine the size of its children.

LayoutCell

Add the **LayoutCell** component to a layout row or layout column's children to customize how a layout cell's size is determined. A layout cell is a programmatic concept whose properties define the area of a child element. Anytime that you add a child element to a layout row or layout column, that child element receives layout cell properties (not visible in the **UI Editor**), which determine the size of the child's space. You can override the layout cell's calculated properties by adding the **LayoutCell** component to the child. For more information, see LayoutCell (p. 1595).

LayoutFitter

Add the **LayoutFitter** component to an element to make the element resize itself to fit its content. Use the layout fitter component with other components that provide cell sizing information, such as text, image (with **ImageType** set to **`Fixed`**), or layout components (cell, row, column, grid). For more information, see LayoutFitter (p. 1600).

LayoutColumn

You can use a **LayoutColumn** component to organize child elements into a column. To use this feature, add the **LayoutColumn** component to an element and then add child elements. The UI system positions the child elements within the column, from top to bottom or bottom to top, depending on the order you choose in the component properties. The child elements can contain a texture or image, a button, a check box, text, columns, rows, grids, and so on.

To see an in-game example of a completed canvas with the **Layout Column** component, open the level UiFeatures in the project SamplesProject. Press **Ctrl+G** to play the game, and then choose **Components**, **Layout Components**, **Layout Column**. You can view examples of different child sizes within a column. Press **Esc** to exit the game.

To view this same canvas in the **UI Editor**, navigate to the `\Gems\LyShineExamples\Assets\UI\Canvases\LyShineExamples\Comp\Layout` directory and open the `SimpleColumn.uicanvas` file.

You can add a prebuilt **Layout Column** element from the slice library. When you do this, a simple layout column is automatically created and nested in your **Hierarchy** pane.

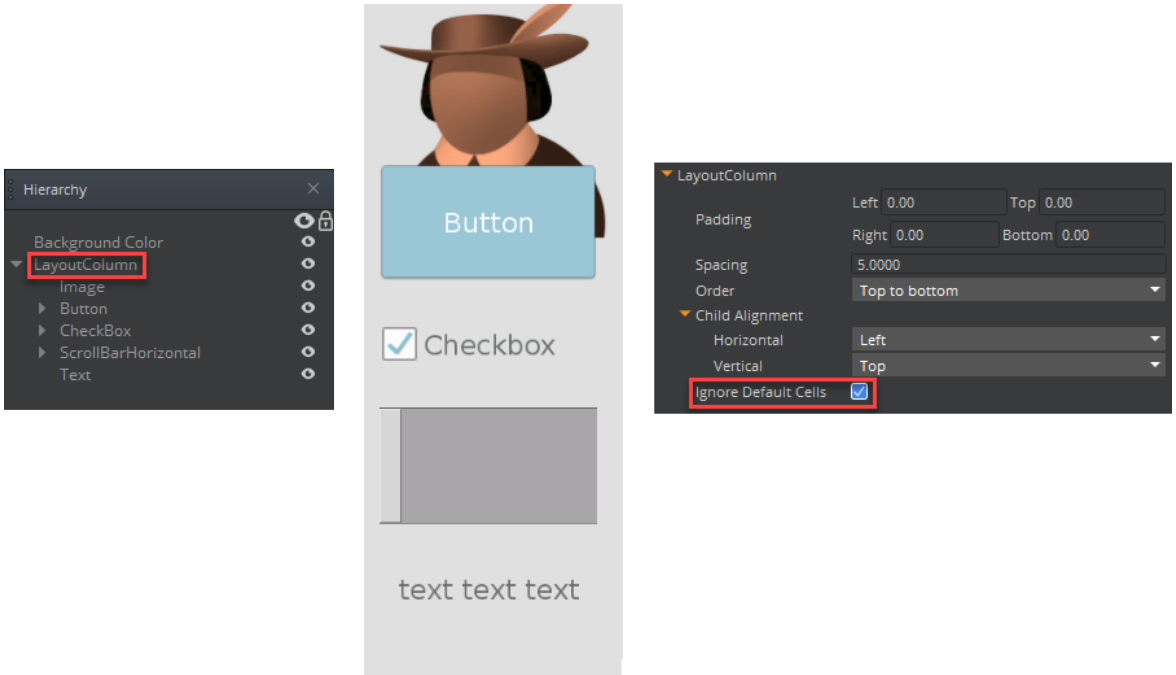
To add a Layout Column element from the slice library

- In the **UI Editor** (p. 1539), choose **New**, **Element from Slice Library**, **LayoutColumn**.

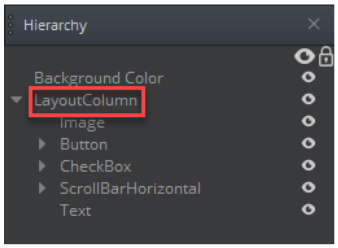
By default, the layout column gives every child the same amount of space, regardless of its content. You can, however, manipulate the sizes of each child by adding the layout cell (p. 1595) component to each or specific children.

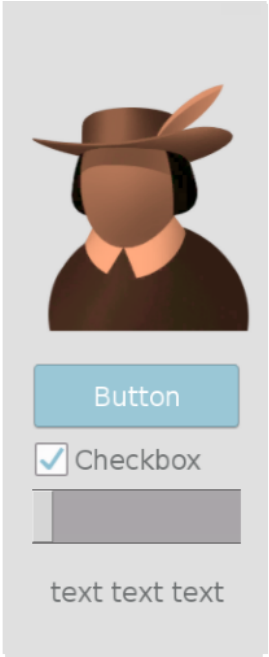
The layout column can also give varying space to each child depending on its content. To enable the layout column to do this, clear the **Ignore Default Cell** option in the **LayoutColumn** component's properties.

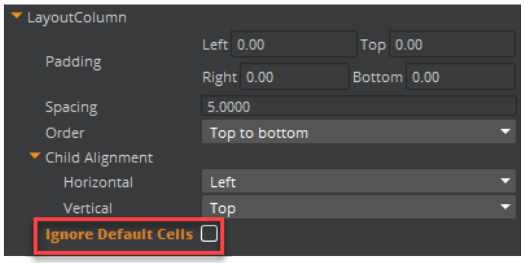
In the first image, **Ignore Default Cells** is selected. The layout column gives each child the same amount of space regardless of their contents.



In the second image, **Ignore Default Cells** is cleared. The layout column calculates its children's space based on their contents.







To see an example of a completed canvas with the layout column component, open `SimpleColumn.uicanvas` in the LyShineExamples Gem (`\dev\Gems\LyShineExamples\Assets\UI\Canvases\LyShineExamples\Comp\`).

To edit a layout column component

In the **Properties** pane of the **UI Editor** (p. 1539), expand **LayoutColumn** and do the following, as appropriate:

Padding

Type values in pixels, relative to the element's borders.

Spacing

Type values in pixels to adjust spacing between elements.

Order

Select **Top-to-Bottom** or **Bottom-to-Top** to specify the order in which the child elements appear in the column.

Child Alignment

If the layout's children don't occupy all the available layout space, use this setting to determine how the children are aligned.

For **Horizontal**, select **Left**, **Center**, or **Right** to determine how the children are aligned horizontally.

For **Vertical**, select **Top**, **Center**, or **Bottom** to determine how the children are aligned vertically.

Ignore Default Cell

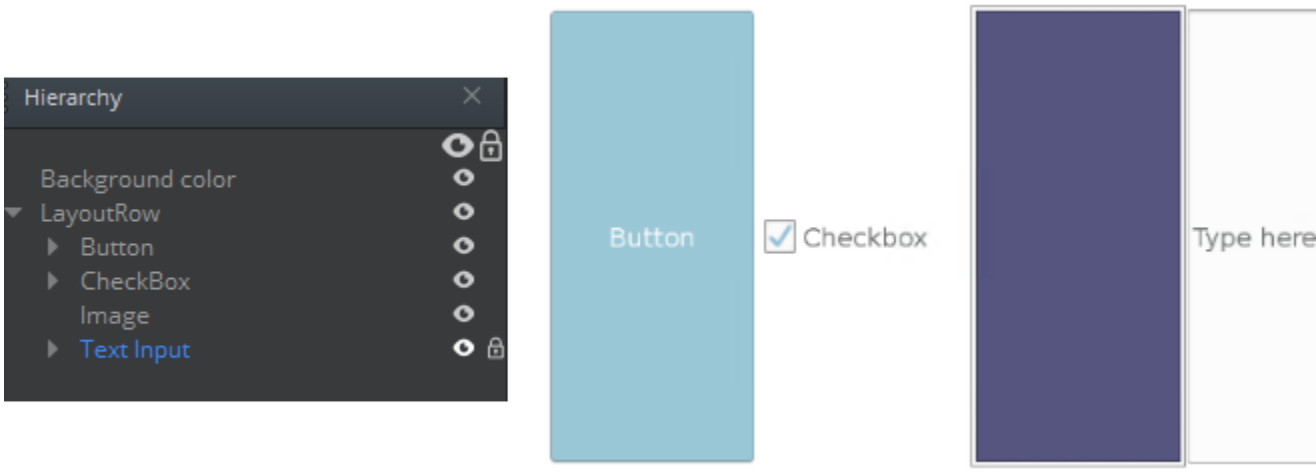
Selected by default, this property causes the layout column to give each child an equal amount of space regardless of their contents (unless the child has a **LayoutCell** (p. 1595) component). The layout column ignores the layout cell's content-based default calculations.

When you clear this option, the layout column uses the children's layout cell calculated values to determine how much space to give each child based on its contents. For more information, see LayoutCell (p. 1595).

LayoutRow

You can use a **LayoutRow** component to organize child elements into a row. To use this feature, you add the **LayoutRow** component to an element and then add child elements. The UI system positions the child elements within the row, from left to right or right to left, depending on the order you choose. The child elements can contain a texture or image, a piece of text, a button, a check box, more columns, rows, grids, and so on. To control the sizes of specific, or all, children, add the layout cell (p. 1595) component to those children.

Similar to the **LayoutColumn** component, the **LayoutRow** component has an **Ignore Default Cell** property. For more information, see LayoutColumn (p. 1591).



To see an in-game example of a completed canvas with the **Layout Row** component, open the level UiFeatures in the project SamplesProject. Press **Ctrl+G** to play the game, and then choose **Components**, **Layout Components**, **Layout Row**. You can view examples of different child sizes within a row. Press **Esc** to exit the game.

To view this same canvas in the **UI Editor**, navigate to the `\Gems\LyShineExamples\Assets\UI\Canvases\LyShineExamples\Comp\Layout` directory and open the `SimpleRow.uicanvas` file.

You can add a prebuilt **Layout Row** element from the slice library. When you do this, a simple layout row is automatically created and nested in your **Hierarchy** pane.

To add a Layout Row element from the slice library

- In the **UI Editor** (p. 1539), choose **New**, **Element from Slice Library**, **LayoutRow**.

To edit a layout row component

In the **Properties** pane of the **UI Editor** (p. 1539), expand **LayoutRow** and do the following, as appropriate:

Padding

Type values in pixels, relative to the element's borders.

Spacing

Type values in pixels to adjust spacing between elements.

Order

Select **Left-to-Right** or **Right-to-Left** to specify the order in which the child elements appear in the row.

Child Alignment

If the layout's children don't occupy all the available layout space, use this setting to determine how the children are aligned.

For **Horizontal**, select **Left**, **Center**, or **Right** to determine how the children are aligned horizontally.

For **Vertical**, select **Top**, **Center**, or **Bottom** to determine how the children are aligned vertically.

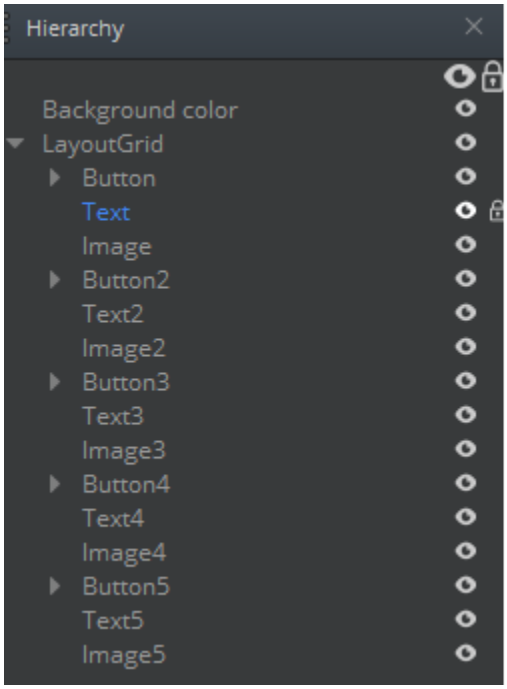
Ignore Default Cell

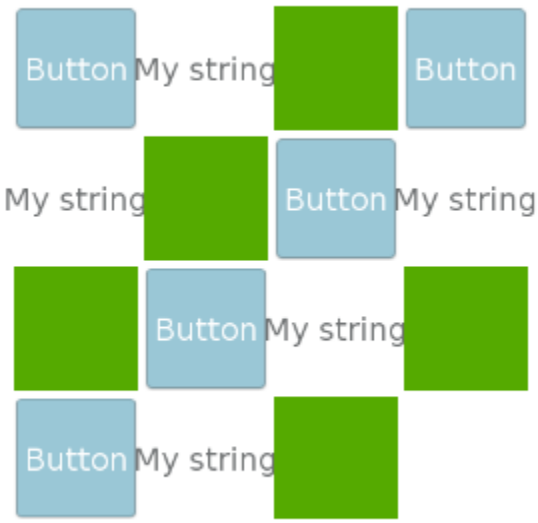
Selected by default, this property causes the layout row to give each child an equal amount of space regardless of their contents (unless the child has a **LayoutCell** (p. 1595) component). The layout row ignores the layout cell's content-based default calculations.

When you clear this option, the layout row uses the children's layout cell calculated values to determine how much space to give each child based on its contents. For more information, see LayoutCell (p. 1595).

LayoutGrid

You can use a layout grid component to organize child elements into a uniform grid. To use this feature, you add the layout grid component to an element and then add child elements. The UI system positions the child elements in a grid pattern. You can choose whether the child elements are positioned left to right or right to left, and bottom to top or top to bottom. The child elements can contain a texture or image, a piece of text, a button, a check box, more columns, rows, grids, and so on. The size of each child is determined by the **Cell Size** property and is independent of each child's content.





To see an in-game example of a completed canvas with the **LayoutGrid** component, open the level UiFeatures in the project SamplesProject. Press **Ctrl+G** to play the game, and then choose **Components**,

Layout Components, **Layout Grid**. You can view examples of different fill patterns. Press **Esc** to exit the game.

To view this same canvas in the **UI Editor**, navigate to the `\Gems\LyShineExamples\Assets\UI\Canvases\LyShineExamples\Comp\Layout` directory and open the `SimpleGrid.uicanvas` file.

You can add a prebuilt **Layout Grid** element from the slice library. When you do this, a simple layout grid is automatically created and nested in your **Hierarchy** pane.

To add a Layout Grid element from the slice library

- In the **UI Editor** (p. 1539), choose **New**, **Element from Slice Library**, **LayoutGrid**.

To edit a layout grid component

In the **Properties** pane of the **UI Editor** (p. 1539), expand **LayoutGrid** and do the following, as appropriate:

Padding

Type values in pixels, relative to the element's borders.

Spacing

Type values in pixels to adjust spacing among elements.

Cell size

Type 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.

Child Alignment

If the layout's children don't occupy all the available layout space, this setting determines how the children are aligned.

For **Horizontal**, select **Left**, **Center**, or **Right** to determine how the children are aligned horizontally.

For **Vertical**, select **Top**, **Center**, or **Bottom** to determine how the children are aligned vertically.

LayoutCell

When working with cell contents, it's important to understand the difference between a layout cell and the **LayoutCell** component. A layout cell represents a set of values that determine the space or area allocated to a child in a layout row or column. The **LayoutCell** component, on the other hand, manipulates the way a layout cell is sized. A layout cell exists on a child of a layout row or layout column whether or not it has a **LayoutCell** component. The **LayoutCell** component simply provides a way to manipulate and override the default calculations of a layout cell.

A layout cell's properties consist of a minimum size, a target size, and an extra size ratio. These properties are not directly modifiable in the UI Editor, but are determined in several ways:

- Components – The following components can affect the layout cell size:
  - Image or text – The image's default size is the layout cell's target size. The length and size of a string in a text component is the layout cell's target size.
  - Layout row or layout column (added or nested as children) – The default values of a layout row or layout column, added as a child, determines the layout cell's minimum and target size. The default value is calculated by the sum of its own children plus padding and spacing.

    Note
    The **LayoutColumn** and **LayoutRow** components contain a property called **Ignore Default Cells**. Selecting this property causes the above calculations to be ignored and simply allocate equal space to all children regardless of content. Clear this property to calculate layout cell values by components. For more information, see LayoutColumn (p. 1591).
- Fixed default layout cell values – If the child doesn't have any components that calculate their own layout cell values, then the layout cell is assigned a minimum and target size of 0 and an extra size ratio of 1. This typically means equal spacing for the children that do not have a component affecting the layout cell's size. Each layout cell grows at the same rate to fill the available space (hence the extra size ratio of 1).
- **LayoutCell** component – Add the **LayoutCell** component to specify values for the minimum and target sizes, and the extra size ratio. Any values you specify here override the values calculated by all other methods.

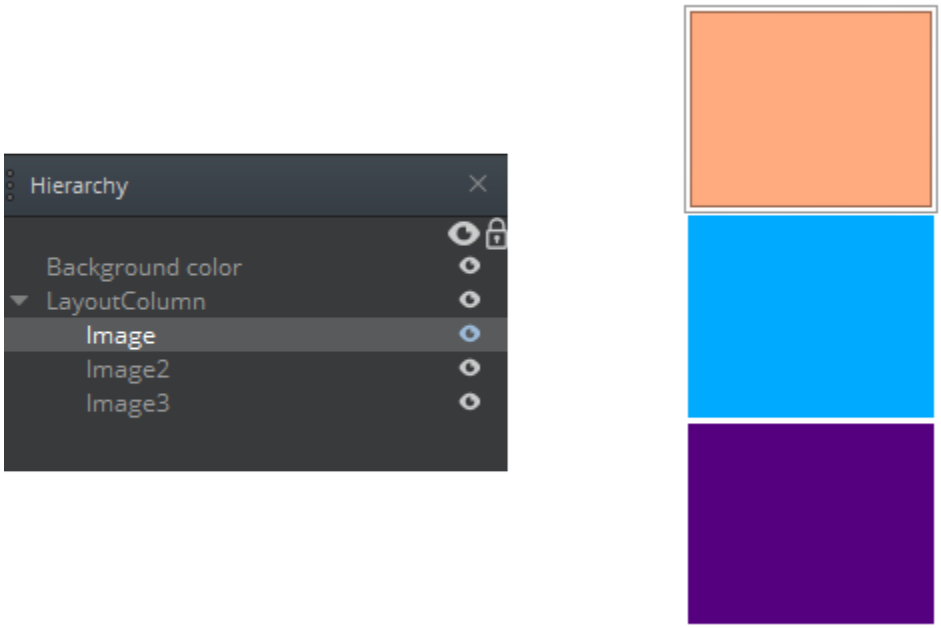
After layout cell values are calculated, layout cell space is allocated by the following:

1. First, each child receives its minimum size (**Min Height** or **Min Width**).
2. If space is available, each child receives its target size (**Target Height** or **Target Width**).
3. If space is still available after that, then the **Extra Size Ratio** value is used to determine how to allocate the remaining space. This ratio is relative to the child's siblings. For example, if one child's extra size ratio is 1, and another child's is 2, then the second child gains twice as much extra space as the first child. An extra size ratio of 0 means that no more space is allocated once the target size is reached.

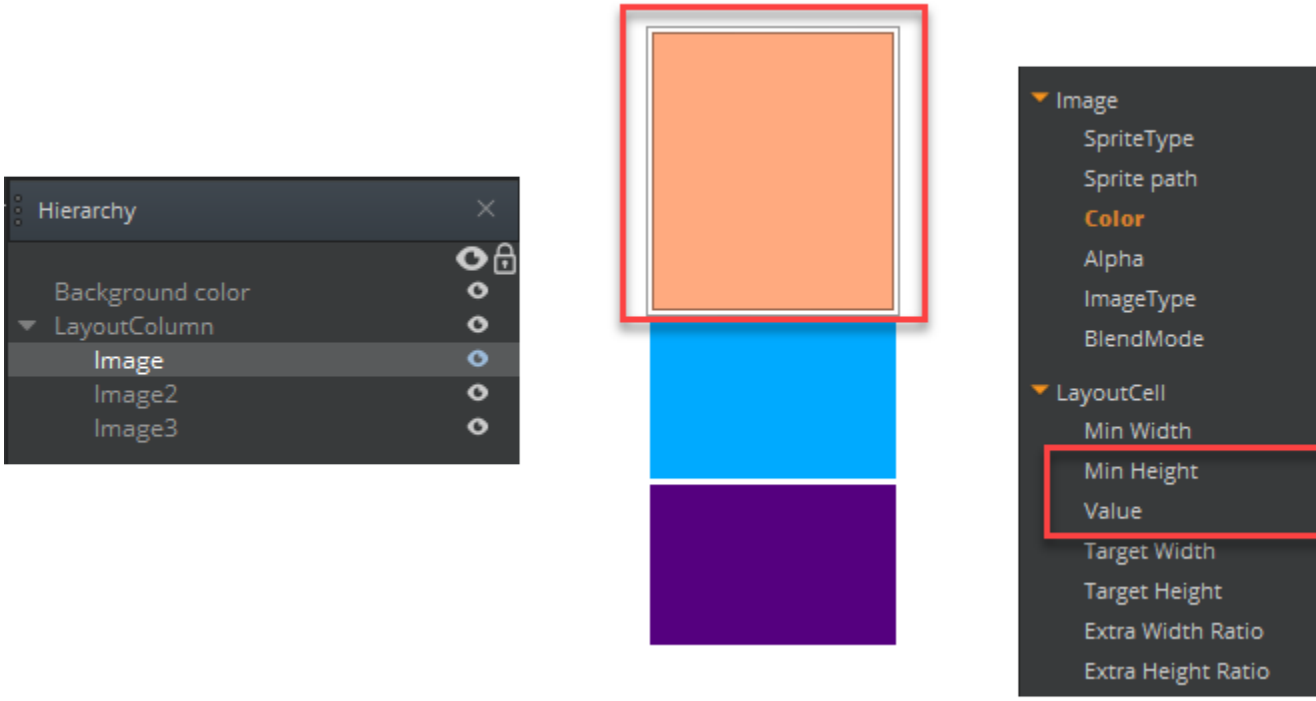
Using the LayoutCell Component

You can apply the **LayoutCell** component to the children of a layout row or column to override the layout cell's default calculations.

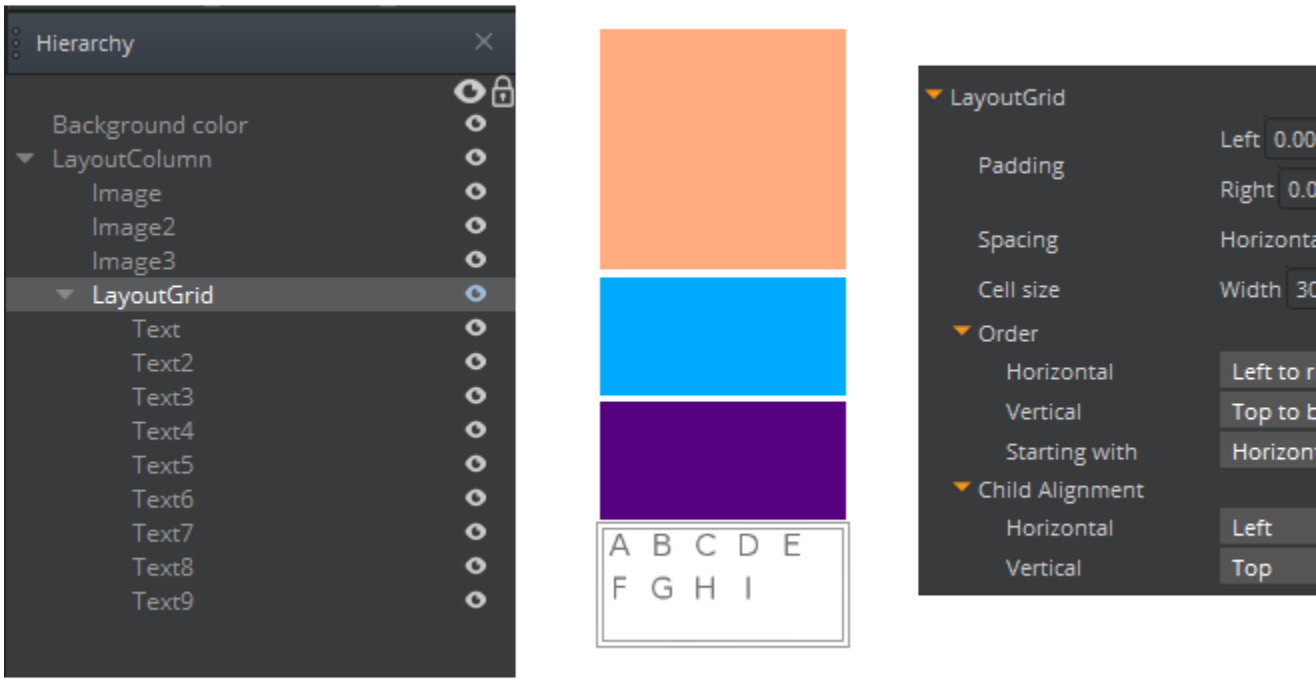
In the following example, the layout column has three images as its children. The images each occupy equal space in the column.



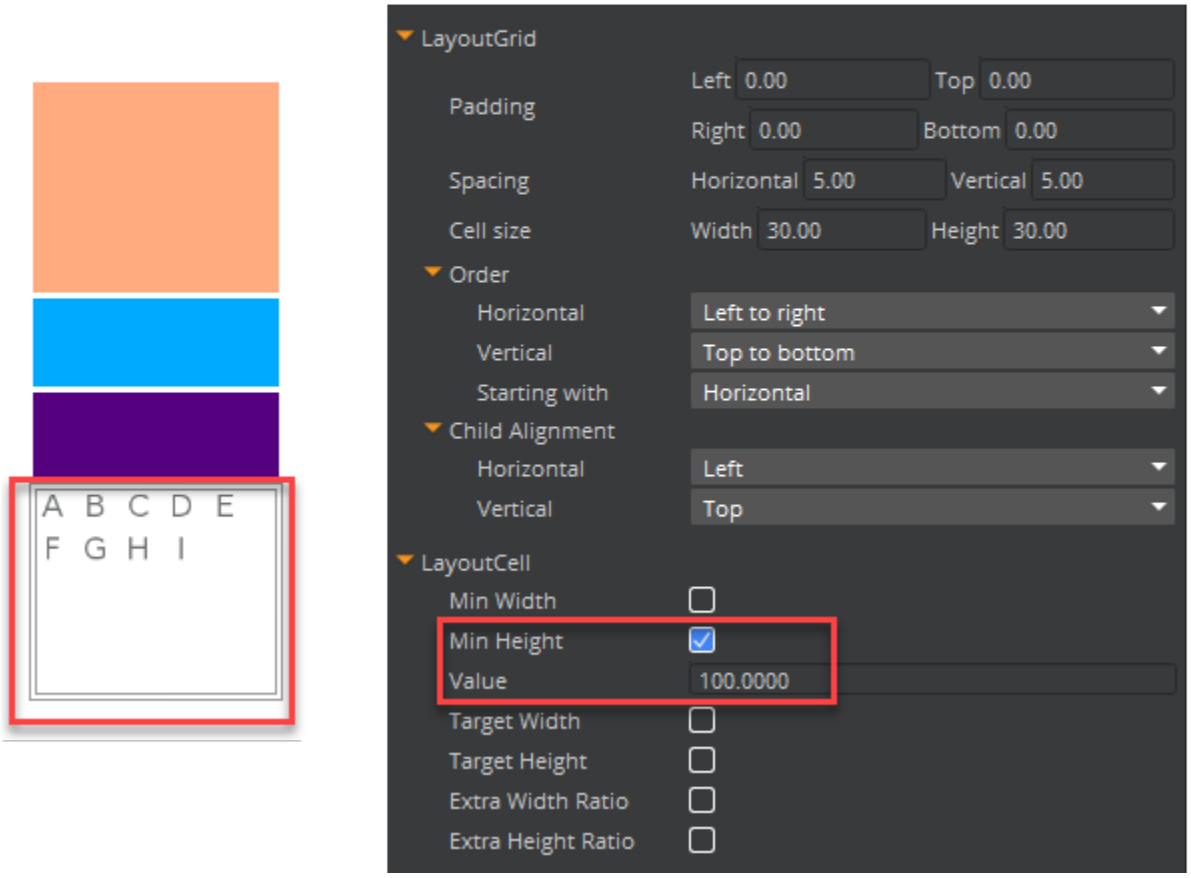
If you add a **LayoutCell** component to the first image, and then select **Min Height** and assign a value of 100, then the UI system overrides that child's default calculated value, and gives the top image more height than its siblings, whose values are recalculated to adjust to the remaining column space.



In the next example, a layout grid was added as a child. Its calculated size is the same as its two siblings above it.



However, when you add a **LayoutCell** component to the grid, and then specify a **Min Height** of 100, then the grid,as a whole is granted that amount of space. If you add the**LayoutCell** component to the children of a layout grid, however, it has no effect. That's because individual grid spaces are always uniform and are controlled by the grid parent.



To edit a LayoutCell component

In the **Properties** pane of the **UI Editor** (p. 1539), expand **LayoutCell** and do the following, as appropriate:

Min Width

Select to define the layout cell's minimum width. Type a value in the box that appears.

Min Height

Select to define the layout cell's minimum height. Type a value in the box that appears.

Target Width

Select to define the layout cell's target width. Type a value in the box that appears. If space is available, this target width is allocated to the layout cell.

Target Height

Select to define the target height. Type a value in the box that appears. If space is available, this target height is allocated to the layout cell.

Extra Width Ratio

Select to define the layout cell's extra width ratio. Type a value in the box that appears. This value is a ratio that is relative to the other elements. If space remains after the layout cell reaches its target size, the **Extra Size Ratio** value is used to allocate the rest of the space.

Because this **Extra Size Ratio** value is relative to the other children, if one child's extra size ratio is 1, and another child's is 2, then the second child gains twice as much extra space as the first child. An extra size ratio of 0 means that no more space is allocated once the target size is reached.

Extra Height Ratio

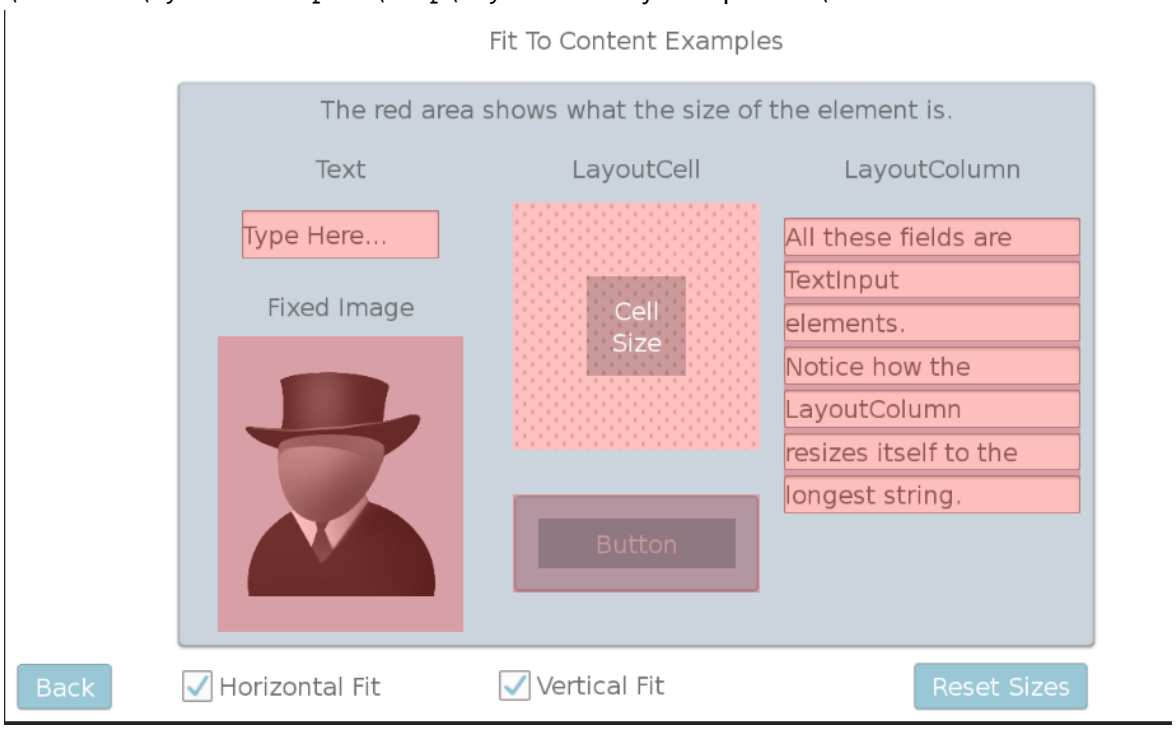
Select to define the layout cell's extra height ratio. Type a value in the box that appears.

LayoutFitter

You can use the **LayoutFitter** component to make an element resize itself to fit its content. Use this component with other components that provide cell sizing information, such as the **Text** component, the **Image** component (with **ImageType** set to **`Fixed`**), or the **Layout** components (**Cell**, **Row**, **Column**, **Grid**).

To see an in-game example of a completed canvas with the **LayoutFitter** component, open the level UiFeatures in the project SamplesProject. Press **Ctrl+G** to play the game, and then choose **Components**, **Layout Components**, **Layout Fitter**. Press **Esc** to exit the game.

To view that same canvas in the **UI Editor**, navigate to the `\Gems\LyShineExamples\Assets\UI\Canvases\LyShineExamples\Comp\Layout` directory and open the `\fitter.uicanvas` file.



To edit a layout fitter component

In the **Properties** pane of the **UI Editor** (p. 1539), expand **LayoutFitter** and do the following, as appropriate:

Horizontal Fit

Select the check box to resize an element's width to fit its content.

Vertical Fit

Select the check box to resize an element's height to fit its content.

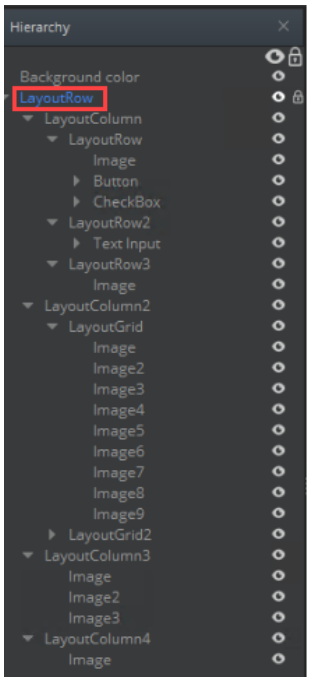
Nesting Layout Components

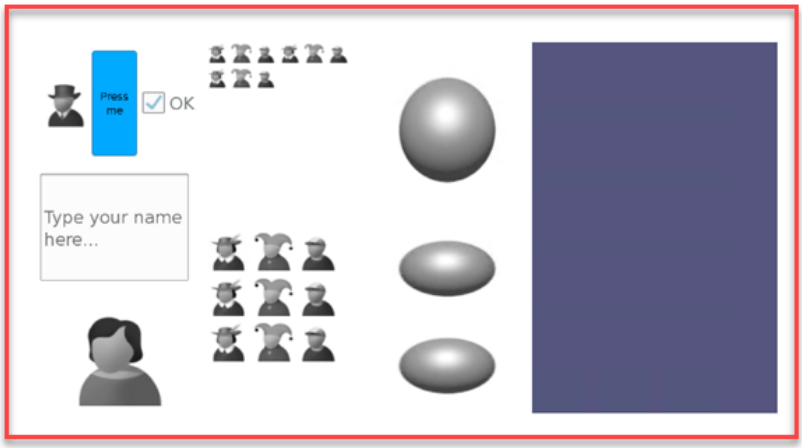
You can nest layout components within other layout components.

To see an in-game example of a completed canvas with a nested layout, open the level UiFeatures in the project SamplesProject. Press **Ctrl+G** to play the game, and then choose **Components**, **Layout Components**, **Nested Layout**. Press **Esc** to exit the game.

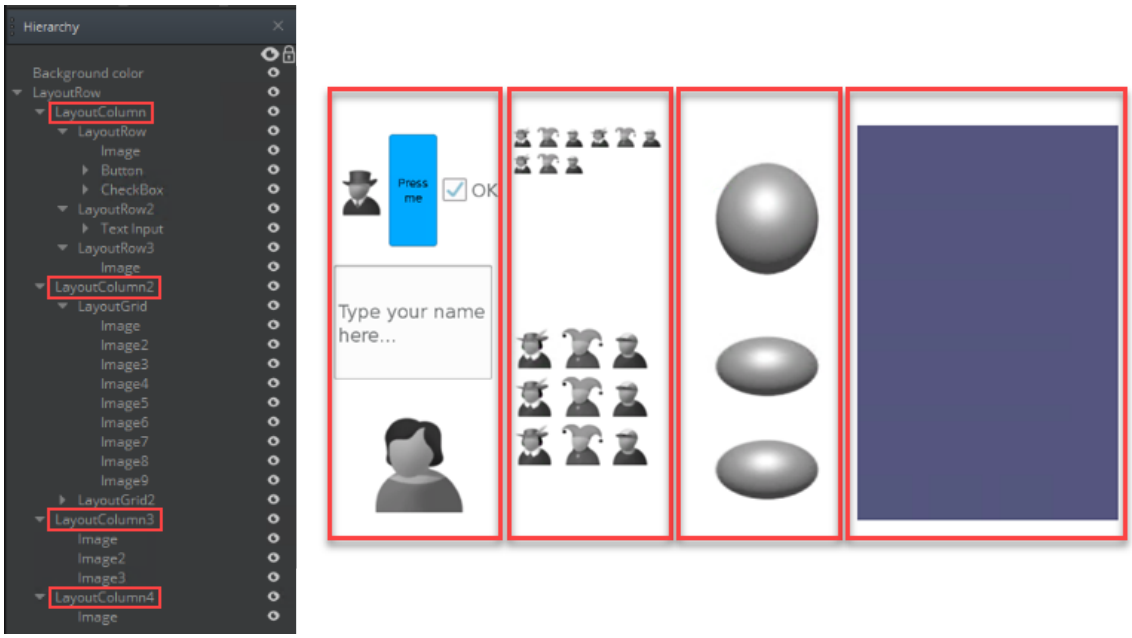
To view that same canvas in the **UI Editor**, navigate to the `\Gems\LyShineExamples\Assets\UI\Canvases\LyShineExamples\Comp\Layout` directory and open the `\NestedLayout.uicanvas`.

The following examples shows one large layout row.

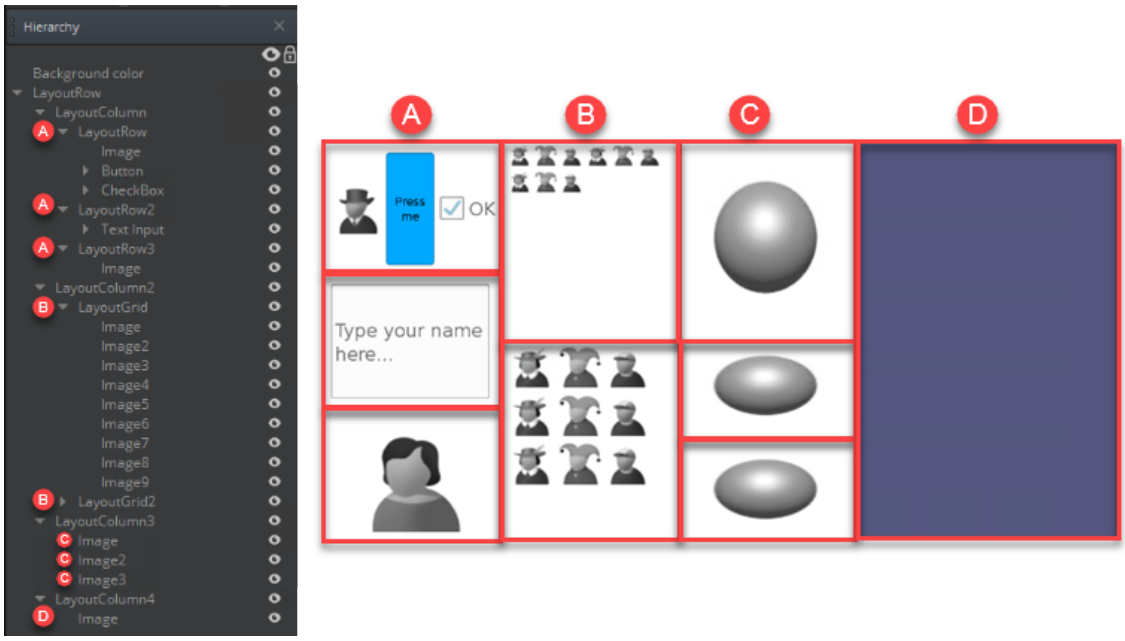




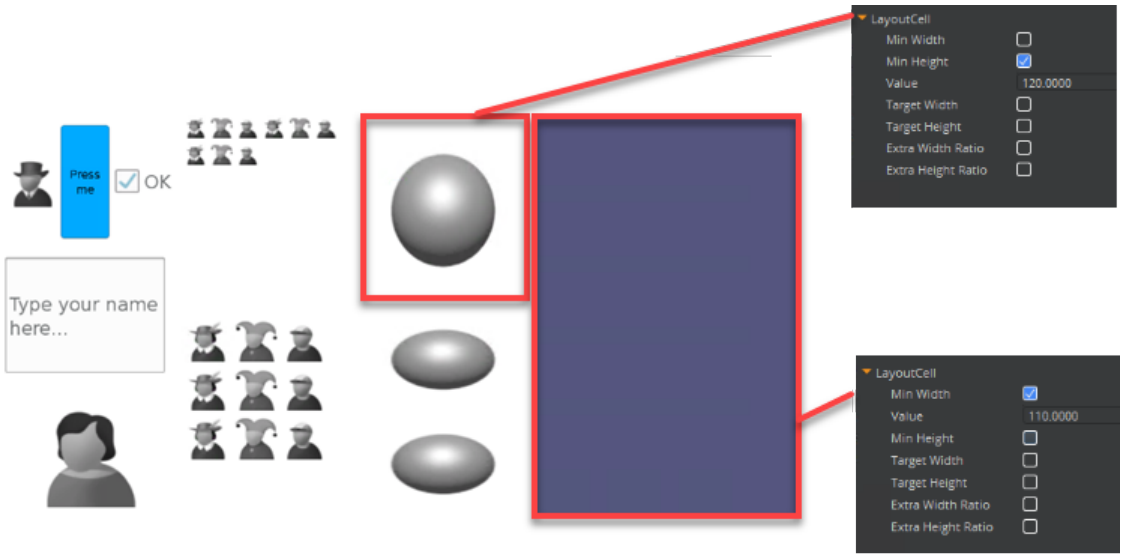
Within the layout row are four columns.



Within column A, there are three layout rows. Column B has two layout grids. Column C has three images. Column D has one large image consisting of a color.



The first image in column C has a layout cell component with a minimum height set at 120. This gives it a larger space than its two siblings below it, which do not have **LayoutCelll** components. Layout column D also has a **LayoutCell** component, with a minimum width of 110, giving it more space than the other three columns, which do not have **LayoutCell** components.



Tooltip Components

You can add either a **Tooltip** component or a **TooltipDisplay** component to an element. With these components, you can display a tooltip when hovering over an interactive element.

To see in-game examples of completed canvases with **Tooltip** components, open the level UiFeatures in the project SamplesProject. Press **Ctrl+G** to play the game, and then choose **Components**, **Other Components**, **Tooltips**. You can view examples of tooltip text options and display styles. Press **Esc** to exit the game.

To view these same canvases in the **UI Editor**, navigate to the `\Gems\LyShineExamples\Assets\UI\Canvases\LyShineExamples\Comp\Tooltips` directory.

You can open the following canvases:

- `TextOptions.uicanvas`
- `Tooltips.uicanvas`

Tooltip

You can use a **Tooltip** component to provide the text of the tooltip. Add a tooltip component to any interactive element that is to display a tooltip in the pause state.

To edit a tooltip component

1. In the **Properties** pane of the **UI Editor** (p. 1539), expand **Tooltip**.
2. Type a text string.

TooltipDisplay

The **TooltipDisplay** component defines the tooltip's display behavior. Add a **TooltipDisplay** component to the element that is to visually represent the tooltip. You must also set the **Tooltip display element** (p. 1546) property of the canvas to this element. For more information, see Configuring Canvas Properties (p. 1545).

To edit a TooltipDisplay component

- In the **Properties** pane of the **UI Editor** (p. 1539), expand **TooltipDisplay** and use the following settings, as appropriate:

  Auto position

  Automatically positions the element based on the positioning mode. The positioning mode is specified in the **Positioning** property.

  Positioning

  Select a positioning mode:

  - **Offset from mouse** – Position the element so that its pivot is a certain distance from the pointer. The distance is specified in the **Offset** property.
  - **Offset from element** – Position the element so that its pivot is a certain distance from the pivot of the element that triggered the tooltip display.

  Offset

  The offset to use when automatically positioning the element.

  Auto size

  Automatically resizes the element to match the tooltip string's size. The text element is a child of the element, and its text is specified in the **Text** property. If **Auto size** is selected, then the text element's anchors should be apart so that the text element can grow and shrink with its parent.

  Text

  The child element that is to display a tooltip string.

  Delay time

  The amount of time to wait before displaying the element.

  Display time

  The amount of time the element is to be displayed.

  Show sequence

  The animation sequence to be played when the element is about to appear.

  Hide sequence

  The animation sequence to be played when the element is about to disappear.

Dynamic Components

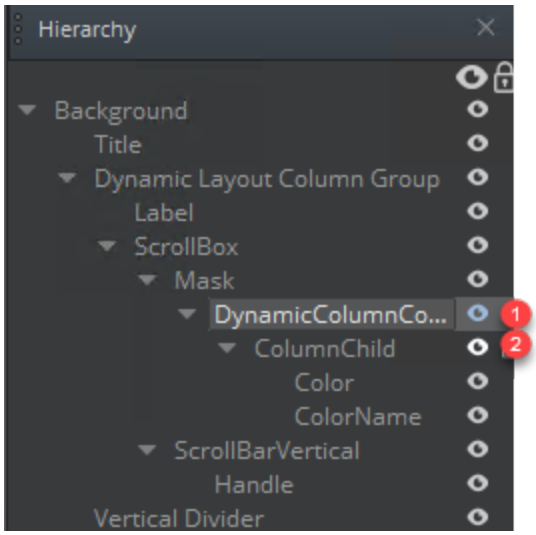
Dynamic components work together with Layout (p. 1590) components and the Scroll Box (p. 1580) component to display dynamic content in the user interface.

To see in-game examples of completed canvases with dynamic components, open the level **UiFeatures** in the project **SamplesProject**. Press **Ctrl+G** to play the game, and then choose **Components**, **Dynamic Components**. You can view examples of different types of dynamic layouts and scroll boxes. Press **Esc** to exit the game.

DynamicLayout

To use the **DynamicLayout** component, you place it on an element that also has a **LayoutColumn** (p. 1591), **LayoutRow** (p. 1593), or **LayoutGrid** (p. 1594) component. With the **DynamicLayout** component, you can change the number of children of the layout element at run time.

The layout element (1) dynamically resizes to fit its child elements. The first child (2) of the layout element acts as the prototype element. At run time, the UI system clones the prototype element to achieve the specified number of children in the layout.



The automatic resizing of the layout element depends on the layout type.

For **LayoutColumn** (p. 1591) and **LayoutRow** (p. 1593) elements, the layout element resizes in order to keep all of the child elements the same size as the prototype element.

For a **LayoutGrid** (p. 1594) element, the cell size of the **LayoutGrid** component determines the size of the child elements. The **LayoutGrid** element's initial size determines the number of children that can fit in each row or each column, depending on fill direction or **Order** settings. If the **Starting with** fill direction is **`horizontal`**, the UI system uses the **LayoutGrid** element's initial width to determine how many children fit in each row. If set to **`vertical`**, the initial height is used to determine how many children fit in each column.

You can use flow graph to set the number of child elements in the layout, and to set up each child element with dynamic content.

To use a dynamic layout component

1. In the **UI Editor** (p. 1539), add a **LayoutRow**, **LayoutColumn**, or **LayoutGrid** prefab. To do this, choose **New**, **Element from Prefab**. Then select one of the layout elements.

   This serves as the structure or framework to hold your dynamic content.
2. Add a **DynamicLayout** component to your layout component. To do this, in the **Properties** pane choose **Add Component**, **DynamicLayout**.

   For **Num Cloned Elements**, type the initial number of children to be created.
3. Create a child entity that has an **Image** component. To do this, right-click your layout component in the **Hierarchy** pane and choose **New**, **Element from Prefab**, **Image**.

   This image serves as the prototype element that will be cloned and filled with dynamic content.

DynamicScrollBox

To use the **DynamicScrollBox** component, you place it on an element that also has a **ScrollBox** component. With the **DynamicScrollBox** component, you can change the number of children of the scroll box's element at run time.

The content element dynamically resizes to fit its child elements. The first child of the content element acts as the prototype element. At run time, the UI system clones the prototype element to achieve the specified number of children in the layout.

With the **DynamicScrollBox** component, only the minimum number of child elements are actually created for display. This is different from the **DynamicLayout** component, where all child elements are created at run time and can consume a large amount of resources. The **DynamicScrollBox**'s elements are reused as the user scrolls; therefore, a scroll box can simulate a large number of children while maintaining good performance.

The **DynamicScrollBox** component automatically positions its children and resizes the content element to match the bounding box of its children. Each child's size is the same as the prototype element. By default, the children are positioned in a row from left to right. If vertical scrolling is enabled, the children are positioned in a column from top to bottom.

To use a dynamic scroll box component

1. In the **UI Editor** (p. 1539), add a **Scrollbox** prefab. To do this, click **New**, **Element from Prefab**, **ScrollBox**.

   This serves as the structure or framework to hold your dynamic content.
2. Add a **DynamicScrollbox** component to your scroll box component. To do this, in the **Properties** pane choose **Add Component**, **DynamicScrollbox**.

   For **Default Num Elements**, type the initial number of children to be created. This is mainly for previewing a canvas in **Preview** mode since the number of children ultimately comes from a custom component that implements the `UiDynamicScrollBoxDataBus`.
3. Create a child entity that has an **Image** component. To do this, right-click on your scroll box component in the **Hierarchy** pane, choose **New**, **Element from Prefab**, **Image**.

   This image serves as the prototype element that will be cloned and filled with dynamic content.

The **DynamicScrollBox** component uses a bus called `UiDynamicScrollBoxDataBus` to retrieve the number of children that the content element should have. It also uses a bus called `Ui DynamicScrollBoxElementNotificationBus` to notify when a child element is about to become

visible. This is where you set up the child element with dynamic content. To do this, you must create and add to the scroll box element a custom component that implements these two buses.

EBus 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) in the *Amazon Lumberyard Developer Guide*.

UiDynamicScrollBoxDataBus:GetNumElements

Implement this bus to provide a dynamic scroll box the number of children it should clone.

Returns the number of children that the dynamic scroll box should clone.

Parameters

None

Return

Number of children to clone.

Ui DynamicScrollBoxElementNotificationBus:OnElementBecomingVisible

Implement this bus to receive notifications when elements of a dynamic scroll box are about to become visible.

Sends a signal when an element of a dynamic scroll box is about to become visible.

Parameters

`entityID` – The entity Id of the element that is about to become visible.

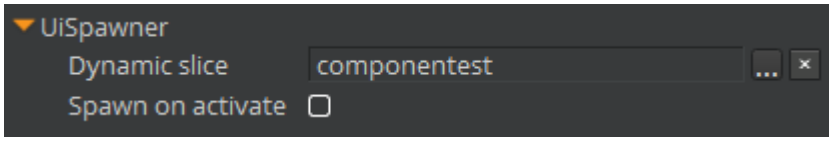
`index` – The index of the element that is about to become visible.

Return

None

UISpawner Component

Use the **UISpawner** component to spawn a run-time dynamic slice (p. 1554) (`*.dynamicslice`) at an entity's location with an optional offset. In combination with scripting, you can use the **UISpawner** component to spawn any dynamic slice at any time and to spawn multiple instances of the same dynamic slice.



UISpawner Component Properties

The **UISpawner** component has the following properties:

Dynamic Slice

Select the slice asset to spawn.

Spawn on Activate

If selected, spawns the selected slice upon activation.

EBus Request Bus Interface

Use the following request functions with the **UiSpawnerBus** EBus interface to communicate with other components of your game.

For more information about using the event bus (EBus) interface, see Event Bus (EBus) in the *Amazon Lumberyard Developer Guide*.

Spawn

Spawns the UI slice specified in the component at the entity's location.

Parameters

None

Return

The slice instantiation ticket for this spawn

Scriptable

Yes

SpawnRelative

Spawns the UI slice specified in the component at the entity's location with the specified relative offset.

Parameters

`Relative` – The offset position from the entity with the spawner component

Return

The slice instantiation ticket for this spawn

Scriptable

Yes

SpawnViewport

Spawns the slice specified in the component at the specified viewport position.

Parameters

`Pos` – The viewport position at which to spawn the slice

Return

The slice instantiation ticket for this spawn

Scriptable

Yes

SpawnSlice

Spawns the specified slice at the entity's location.

Parameters

`slice` – Specifies the slice asset to be spawned

Return

The slice instantiation ticket for this spawn

Scriptable

No

SpawnSliceRelative

Spawns the given slice at the entity's location with the relative offset.

Parameters

`slice` – Specifies the slice asset to be spawned

`relative` – The offset position from the entity with the spawner component

Return

The slice instantiation ticket for this spawn

Scriptable

No

SpawnSliceViewport

Spawns the specified slice at the specified viewport position.

Parameters

`slice` – Specifies the slice asset to be spawned

`pos` – The viewport position at which to spawn the slice

Return

The slice instantiation ticket for this spawn

Scriptable

Yes

EBus Notification Bus Interface

Use the following notification functions with the UiSpawnerNotificationBus EBus interface to communicate with other components of your game.

For more information about using the event bus (EBus) interface, see Event Bus (EBus) in the *Amazon Lumberyard Developer Guide*.

OnSpawnBegin

Announces that the slice has been spawned, but entities have not yet been activated. `OnEntitySpawned` events are about to be dispatched.

Parameters

`ticket` – The slice instantiation ticket returned by the spawn function. These can be compared in order to know which spawn request it relates to.

Scriptable

Yes

OnSpawnEnd

Announces that a slice has been spawned. This function is called once for each spawn request. All `OnEntitySpawned` events have been dispatched.

Parameters

`ticket` – The slice instantiation ticket returned by the spawn function. These can be compared in order to know which spawn request it relates to.

Scriptable

Yes

OnEntitySpawned

Announces that an entity has been created during a spawn. This function is called once for each entity created while spawning a slice.

Parameters

`ticket` – The slice instantiation ticket returned by the spawn function. These can be compared in order to know which spawn request it relates to.

`spawnedEntity` – Specifies the ID of the spawned entity

Scriptable

Yes

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 in-game examples of completed canvases with a **Fader** component, open the level UiFeatures in the project SamplesProject. Press **Ctrl+G** to play the game, and then choose **Components**, **Other**

Components, **Fader**. You can view an example of a direct fade and an animated fade. Press **Esc** to exit the game.

To view these same canvases in the **UI Editor**, navigate to the `\Gems\LyShineExamples\Assets\UI\Canvases\LyShineExamples\Comp\Fader` directory.

You can open the following canvases:

- `AnimFade.uicanvas`
- `DirectFade.uicanvas`

To edit a fader component

1. In the **Properties** pane of the **UI Editor** (p. 1539), 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 in-game examples of completed canvases with a **Mask** component, open the level UiFeatures in the project SamplesProject. Press **Ctrl+G** to play the game, and then choose **Components**, **Other Components**, **Mask**. Press **Esc** to exit the game.

To view these same canvases in the **UI Editor**, navigate to the `\Gems\LyShineExamples\Assets\UI\Canvases\LyShineExamples\Comp\Mask` directory.

You can open the following canvases:

- `AlphaMask.uicanvas` – Using a texture with alpha as a mask
- `Basic.uicanvas` – Basic mask example
- `ImageMode.uicanvas` – Examples of using different image modes on the image used for the mask
- `MaskingInteractables.uicanvas` – Example of how masks interact with interactable components
- `NestedMasks.uicanvas` – Example of nesting masks
- `TextMask.uicanvas` – Example of using text as the visual component of the mask

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. 1854), 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 with a ScrollBox prefab element (p. 1580).

To add an image to be used as a custom mask

1. In the **UI Editor** (p. 1539) toolbar, create a new empty element by choosing **New**, **Empty Element**. This is the parent element.
2. In the **Properties** pane, add an image component by choosing **Add Component**, **Image**.
3. Add a mask component by choosing **Add Component**, **Mask**.
4. Add a child element by right-clicking the parent element and then choosing **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 of the **UI Editor** (p. 1539), 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. 1565).

Using the procedures in this section, you can:

- Add new fonts to your game UI (p. 1612)
- Create font families (p. 1613)
- Configure font rendering quality (p. 1615)

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 `.font` 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 `.font` file into your game project `Font` directory. The following directories (included in the Lumberyard project) contain font `.font` files for reference:
   - `dev\Engine\Fonts\`
   - `dev\SamplesProject\Fonts\`
3. Change the `.font` file name (leave the `.font` extension unchanged). Use any file name that is descriptive and appropriate for your purposes.
4. Open the `.font` 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 `.font` file (p. 1564) 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.

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. 1565).

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 in a project must be unique, and only one `fontfamily` tag may be specified per `.fontfamily` file. You can, however, reuse the same font XML files (defined by the file tag) in multiple font families.

Tag: `font`

Container tag for the `file` tag.

Attribute: `lang`

Language that the font files should be associated with. The font files are loaded only if the listed language is being used. This allows a single font family to use different fonts and styling depending on the language being used.

Tag: `file`

Attribute: `path`

Path to the font XML, a TTF or OTF file. The path is relative to the font family file. The same font asset can be referenced multiple times for a given font family and across multiple font families.

Attribute: `tags`

This tag is optional. If omitted, this font file is used when no styling is applied.

Values:

- **b** – indicates <b> bold tag
- **i** – indicates <i> italic tag

- **b,i** – indicates when both <b> 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, the number of character slots in the font texture, 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

Fonts are defined in XML by `*.font` files. The XML in a `.font` file defines various parameters, such as the path to the source TTF/OTF asset and important rendering properties. The font file `Engine/Fonts/default-ui.font` 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 (along with the number of character slots) is an important value for determining font rendering quality.

Character Slots

In Lumberyard, a font texture is logically divided into equally sized slots. In each slot, there is a uniform amount of space for each character (glyph). By default (without additional configuration), there are 128 unique characters (16 rows * 8 columns).

If you support a language with many unique characters, such as Chinese, Japanese, or Korean, the default number of slots (128) may not be adequate for your needs; you will need further configuration. Otherwise, 128 unique characters may be adequate for most languages. The following information about character slots describes in further detail the font rendering pipeline in Lumberyard.

When rendering a string of characters, the number of *unique* characters in a string is different than the number of characters in a string (its length). The number of character slots in a font texture imposes a limitation only on the number of unique characters that can be rendered in a single frame.

For example, the following is a font definition which defines a font texture with `1 (1x1)` texture slot.

```
<font path="Vera.ttf" w="512" h="512" widthslots="1" heightslots="1"/>
```

The default values for `widthslots` and `heightslots` is *16* and *8*, respectively. However, as shown in the above example, you can configure the number of character slots (`1`). This font is capable of rendering a single unique character to the screen, any number of times, such as the following string:

AAAA

The number of unique characters in `AAAA` is 1, while the length of the string is 4. This font texture configuration can render this character an unlimited number of times (that is, a string of variable length) as long as the string contains only a single character. However, this font could not render the following string:

AABB

Because only one character slot exists in the texture, it cannot store both the glyphs for upper-case 'A' and upper-case 'B' and render them both in a frame. In order to render this string, you would need to increase the number of slots using the `widthslots` and `heightslots` parameters.

Here is another example:

```
<font path="NotoSansSC-Regular.otf" w="4096" h="4096" widthslots="128" heightslots="128"/>
```

In this example, the font texture size is `4096x4096` and there are a total number of `128x128` (16,384) character slots. To determine the available size for each character, divide the texture size (4096x4096) by the number of slots (128x128) to yield a 32x32 pixel space per character. This configuration allows you to render over 16K unique characters at a 32px size in a single frame.

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 / `widthslots` = Slot width
- Font texture height / `heightslots` = Slot height

Where `widthslots` is the number of character slots across the width (x-axis) of the font texture, and `heightslots` is the number of character slots across the height (y-axis) of the font texture.

In the `default-ui.font` example given in the previous section, the font texture size was 512x256. Assuming the character slots are at their default values (16x8):

- 512 / 16 = 32 (slot width)
- 256 / 8 = 32 (slot height)

For a 512x256 sized font texture, you can render pixel-perfect characters at 32x32 pixels.

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 `.font` file (p. 1613) 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 elements that have text components that use your font `.font` 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 * `widthslots`
   - Texture height = Font size * `heightslots`

   Note
   The default value for `widthslots` and `heightslots` is *16* and *8*, respectively, which gives 128 total character slots. If you need to render more than 128 unique characters to the screen in a single frame—for example, if your game supports Chinese, Japanese, or Korean text—then adjust these values accordingly.
6. Edit your font `.font` 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 `widthslots` (the default value is 16), and the height must be a multiple of `heightslots` (the default value is 8).
- You can have multiple font `.font` files that reference the same TTF/OTF file 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. You can achieve this with separate `.font` files for each use case, with font texture settings adjusted for ideal rendering quality.

Using the Animation Editor

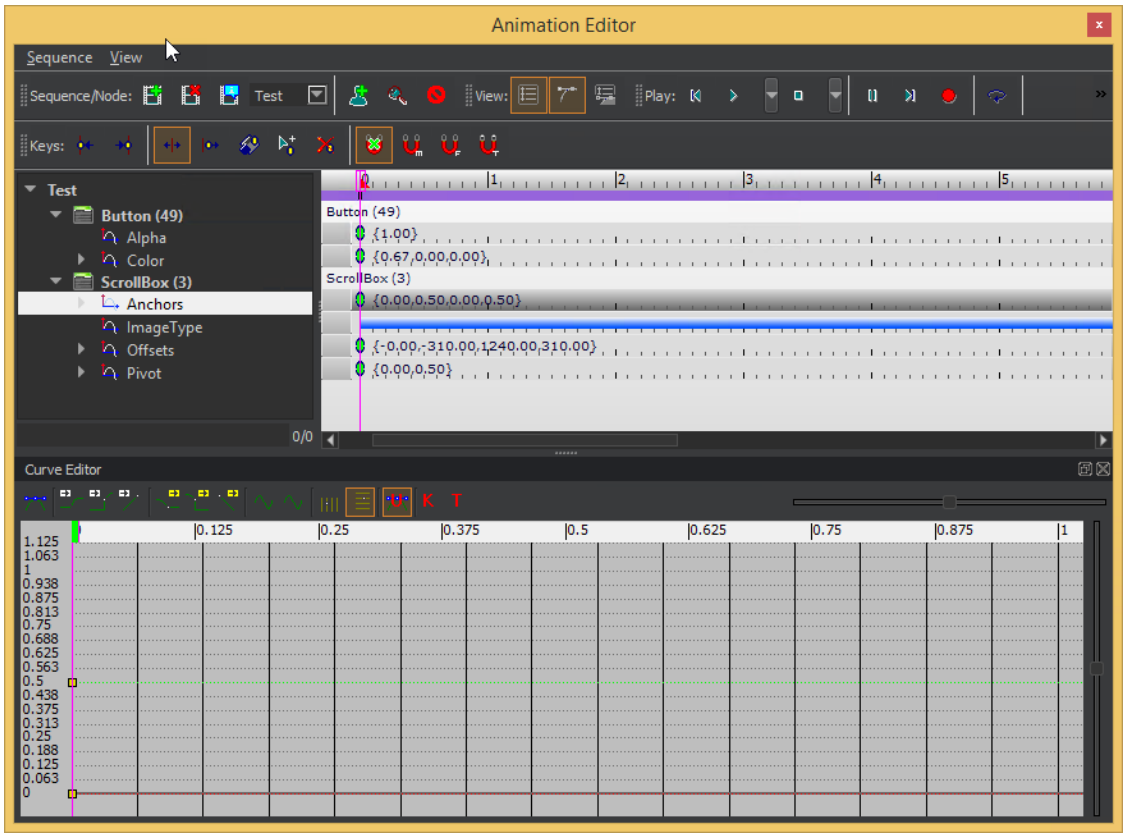
You can use animation sequences to animate UI elements. A UI canvas can contain many named animation sequences.

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. 1539) 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. 1620).

You can create an animation sequence from the **Animation Editor** menu or toolbar.

To create a new animation sequence

In the **Animation Editor** (p. 1617), 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. 1539), 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. 1620)

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. 1780).

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

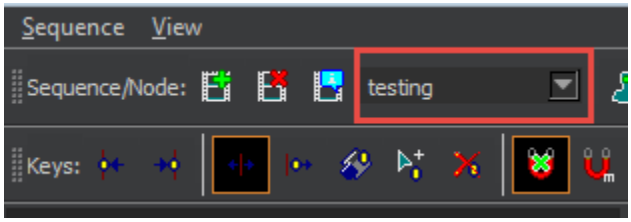
After you create your sequence(s) and record animation data to them, you can use the **Node Pane**, **Track Editor**, and **Curve Editor** in the **Animation Editor** to modify your sequences.

- 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. 1620).
- 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. 1622).
- 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. 1625).

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

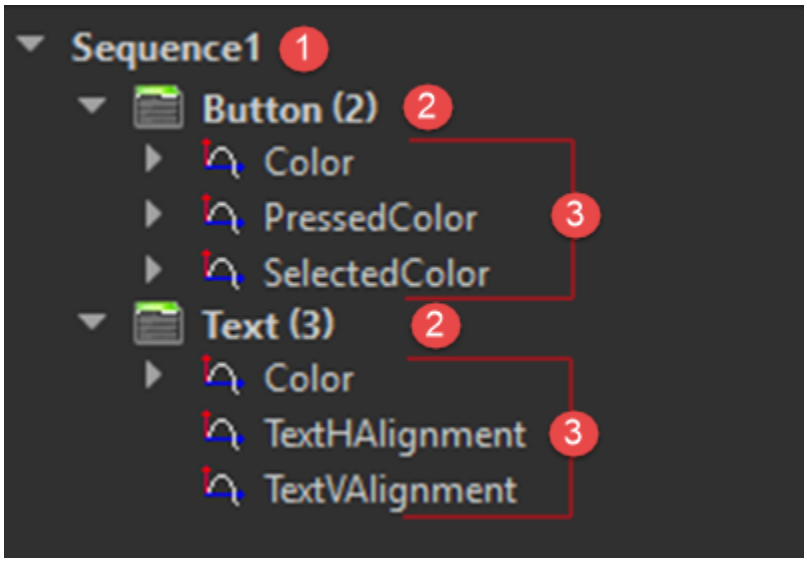
- Using the Node Pane (p. 1620)
- Using the Track Editor (p. 1622)
- Using the Curve Editor (p. 1625)

Using the Node Pane

The **Node Pane** in the Animation Editor (p. 1617) 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. 1539), 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**.

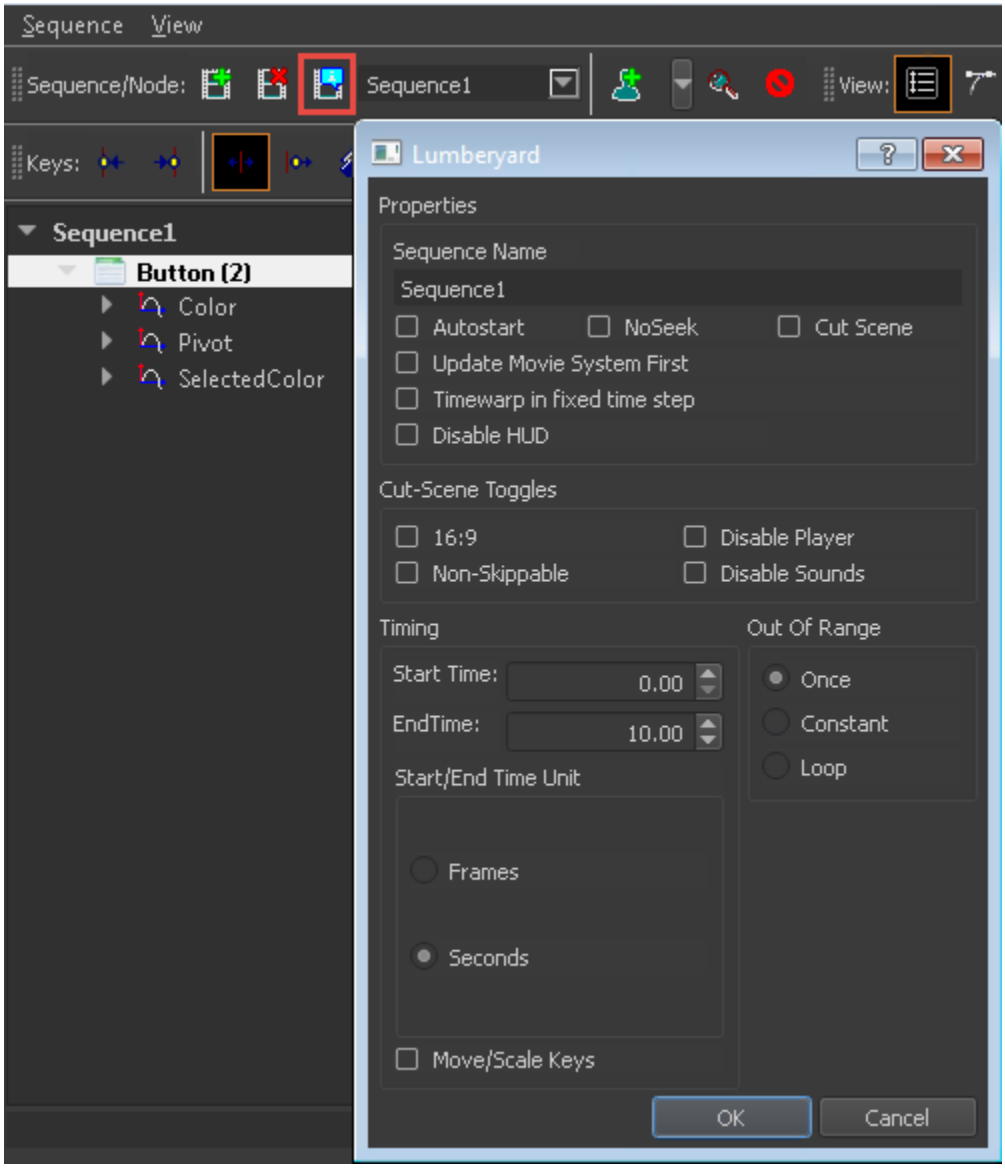
To edit a track

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. 1617), 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

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. 1617) 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, ½, ¼, or ⅛. 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, offfset, 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

Flow Graph is deprecated and will be removed in a future version of Lumberyard. Learn about Script Canvas (p. 682), Lumberyard's new visual scripting environment.

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: **UIe** and **UI**. The improved **UIe** nodes supersede the original (and now legacy) **UI** flow graph nodes. For best results when creating new flow graph nodes, use the **UIe** flow graph node set.

For more information on flow graphs, see Flow Graph System (p. 754).

Topics

UIe Flow Graph Nodes

Flow Graph is deprecated and will be removed in a future version of Lumberyard. Learn about Script Canvas (p. 682), Lumberyard's new visual scripting environment.

The **UIe** flow graph node set supersedes the original **UI** flow graph node set (now legacy). The **UIe** 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 **UIe** 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. 1546).

Topics

UIe Canvas Nodes

Flow Graph is deprecated and will be removed in a future version of Lumberyard. Learn about Script Canvas (p. 682), Lumberyard's new visual scripting environment.

You can use these flow graph nodes to perform actions on a UI canvas.

UIe: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.

ElementId

The element Id of the fader element. Used if `ElementName` is empty.

UIe: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.

UIe: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.

UIe: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:GetIsPixelAligned 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.

UIe: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.

UIe: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.

UIe: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

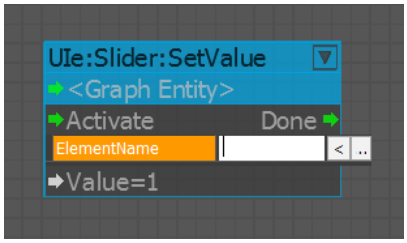
Sends a signal when the node's action is finished.

UIe Component Nodes

Flow Graph is deprecated and will be removed in a future version of Lumberyard. Learn about Script Canvas (p. 682), Lumberyard's new visual scripting environment.

The **UIe** component flow graph node set supersedes the original **UI** component flow graph node set (now legacy). The **UIe** 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 **UIe** 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

UIe Button Component Nodes

Flow Graph is deprecated and will be removed in a future version of Lumberyard. Learn about Script Canvas (p. 682), Lumberyard's new visual scripting environment.

Use the following flow graph nodes to perform actions on the button component.

UIe: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.

UIe: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.

UIe Checkbox Component Nodes

Flow Graph is deprecated and will be removed in a future version of Lumberyard. Learn about Script Canvas (p. 682), Lumberyard's new visual scripting environment.

Use the following flow graph nodes to perform actions on the check box component.

UIe: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.

UIe: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).

UIe: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 check box is selected (in the `on` state).

Node Output

Done

Sends a signal when the node's action is finished.

UIe: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).

UIe: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.

UIe: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).

UIe: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.

UIe: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.

UIe: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.

UIe DynamicLayout Component Nodes

Flow Graph is deprecated and will be removed in a future version of Lumberyard. Learn about Script Canvas (p. 682), Lumberyard's new visual scripting environment.

Use the following flow graph nodes to perform actions on the dynamic layout component.

UIe:DynamicLayout:SetNumChildElements Node

Sets the number of child elements of the layout element. The child elements are cloned from a prototype element.

Node Inputs

Activate

Sets the number of child elements.

ElementName

Name of the layout element.

ElementId

The element Id of the layout element. Used if `ElementName` is empty.

NumChildElements

The number of child elements.

Node Output

Done

Sends a signal when the node's action is finished.

UIe DynamicScrollbox Component Nodes

Flow Graph is deprecated and will be removed in a future version of Lumberyard. Learn about Script Canvas (p. 682), Lumberyard's new visual scripting environment.

Use the following flow graph nodes to perform actions on the dynamic scroll box component.

UIe:DynamicScrollBox:GetLocationIndexOfChild Node

Gets the index of the specified child of the dynamic scroll box's content element. Use this node rather than `UIe:Element:GetIndexOfChild`, since dynamic scroll boxes create only the minimum number of children for display.

Node Inputs

Activate

Updates the output.

ElementName

Name of the dynamic scroll box element.

ElementId

The element Id of the dynamic scroll box element. Used if `ElementName` is empty.

ChildElementId

The element Id of the child.

Node Output

Index

The index of the child.

UIe:DynamicScrollBox:RefreshContent Node

Refreshes the dynamic scroll box by retrieving the number of the content element's children, setting up the content element according to the number of children, and notifying listeners of the child elements that are visible.

Node Inputs

Activate

Refreshes the content.

ElementName

Name of the dynamic scroll box element.

ElementId

The element Id of the dynamic scroll box element. Used if `ElementName` is empty.

Node Output

Done

Sends a signal when the node's action is finished.

UIe Element Node

Flow Graph is deprecated and will be removed in a future version of Lumberyard. Learn about Script Canvas (p. 682), Lumberyard's new visual scripting environment.

Use the following flow graph nodes to perform actions on an element.

UIe:Element:IsEnabled Node

Gets whether the element is enabled.

Node Inputs

Activate

Updates the outputs.

CanvasID

Unique identifier of the element's canvas.

ElementID

The element Id. Used if `ElementName` is empty

Node Output

State

The enabled state of the element.

UIe: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.

ElementID

The element Id. Used if `ElementName` is empty

State

The Boolean enabled state of the element.

Node Output

Done

Sends a signal when the node's action is finished.

UIe:Element:GetChildAtIndex Node

Gets the child of an element at the specified index.

Node Inputs

Activate

Updates the outputs.

ElementName

Name of the element.

ElementID

The element Id. Used if ElementName is empty.

ChildIndex

The index of the child.

Node Output

ChildElementName

The name of the child element.

ChildElementId

The element Id of the child element.

UIe:Element:GetChildByName

Gets the element Id of a child by its name.

Node Inputs

Activate

Updates the outputs.

ElementName

Name of the element.

ElementID

The element Id. Used if ElementName is empty.

ChildElementName

The name of the child element.

Node Output

ChildElementId

The element Id of the child element.

UIe:Element:GetIndexOfChild Node

Gets the index of the specified child.

Node Inputs

Activate

Updates the outputs.

ElementName

Name of the element.

ElementID

The element Id. Used if ElementName is empty.

ChildElementName

The name of the child element.

ChildElementId

The child element Id. Used if `ChildElementName` is empty.

Node Output

IndexOfChild

The index of the child element.

UIe:Element:GetNumChildElements Node

Gets the number of children of an element.

Node Inputs

Activate

Updates the outputs.

ElementName

Name of the element.

ElementID

The element Id. Used if ElementName is empty.

Node Output

NumChildElements

The number of child elements.

UIe:Element:GetParent Node

Gets the parent of an element.

Node Inputs

Activate

Updates the outputs.

ElementName

Name of the element.

ElementID

The element Id. Used if `ElementName` is empty.

Node Output

ParentElementName

The name of the parent element.

ParentElementId

The element Id of the parent element.

UIe Fader Component Nodes

Flow Graph is deprecated and will be removed in a future version of Lumberyard. Learn about Script Canvas (p. 682), Lumberyard's new visual scripting environment.

Use the following flow graph nodes to perform actions on the fader component.

UIe:Fader:Animation Node

Animates the fader component on the specified element.

Node Inputs

Activate

Starts a fade animation.

ElementName

Name of the fader element.

ElementId

The element Id of the fader element. Used if `ElementName` is empty.

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.

UIe:Fader:GetFadeValue Node

Gets the floating-point fade value of an element.

Node Inputs

Activate

Updates the output.

ElementName

Name of the fader element.

ElementId

The element Id of the fader element. Used if `ElementName` is empty.

Node Output

Value

The floating-point fade value of the element (**ElementID**).

UIe: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.

ElementId

The element Id of the fader element. Used if `ElementName` is empty.

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.

UIe Image Component Nodes

Flow Graph is deprecated and will be removed in a future version of Lumberyard. Learn about Script Canvas (p. 682), Lumberyard's new visual scripting environment.

Use the following flow graph nodes to perform actions on the image component.

UIe:Image:GetImageSource Node

Replaced by UIe:Image:GetSprite Node (p. 1644).

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.

UIe:Image:SetImageSource Node

Replaced by UIe:Image:SetSprite Node (p. 1644).

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.

UIe: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.

UIe: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.

UIe: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

UIe: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.

UIe: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**).

UIe: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.

UIe Interactable Component Nodes

Flow Graph is deprecated and will be removed in a future version of Lumberyard. Learn about Script Canvas (p. 682), Lumberyard's new visual scripting environment.

Use the following flow graph node for the **Interactable** component.

UIe: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.

UIe Layout Column Component Nodes

Flow Graph is deprecated and will be removed in a future version of Lumberyard. Learn about Script Canvas (p. 682), Lumberyard's new visual scripting environment.

Use the following flow graph nodes to perform actions on the layout column component.

UIe: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

UIe: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.

UIe: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.

UIe: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.

UIe: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**).

UIe: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.

UIe Layout Grid Component Nodes

Flow Graph is deprecated and will be removed in a future version of Lumberyard. Learn about Script Canvas (p. 682), Lumberyard's new visual scripting environment.

Use the following flow graph nodes to perform actions on the layout grid component.

UIe: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**).

UIe: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.

UIe: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

UIe: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.

UIe: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.

UIe: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.

UIe: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**).

UIe: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.

UIe: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

UIe: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.

UIe: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

UIe: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.

UIe Layout Row Component Nodes

> Flow Graph is deprecated and will be removed in a future version of Lumberyard. Learn about Script Canvas (p. 682), Lumberyard's new visual scripting environment.

Use the following flow graph nodes to perform actions on the layout row component.

UIe: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

UIe: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.

UIe: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.

UIe: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.

UIe: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**).

UIe: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

Flow Graph is deprecated and will be removed in a future version of Lumberyard. Learn about Script Canvas (p. 682), Lumberyard's new visual scripting environment.

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.

UIe: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.

UIe: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.

UIe: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.

UIe: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.

UIe: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.

UIe ScrollBox Component Nodes

Flow Graph is deprecated and will be removed in a future version of Lumberyard. Learn about Script Canvas (p. 682), Lumberyard's new visual scripting environment.

Use the following flow graph nodes to perform actions on the **ScrollBox** component.

UIe: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.

UIe: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 **ScrollBox** scrolls.

UIe:ScrollBox:SetContentEntity Node

Sets the content element for the **ScrollBox**.

Node Inputs

Activate

Sets the content element for the **ScrollBox**.

ElementName

Name of the element.

Content

The element that the **ScrollBox** scrolls.

Node Output

Done

Sends a signal when the node's action is finished.

UIe:ScrollBox:GetIsHorizontalScrollingEnabled Node

Gets whether the **ScrollBox** allows horizontal scrolling.

Node Inputs

Activate

Updates the outputs.

ElementName

Name of the element.

Node Output

Enabled

Indicates whether horizontal scrolling is enabled.

UIe:ScrollBox:SetIsHorizontalScrollingEnabled Node

Sets whether the **ScrollBox** allows horizontal scrolling.

Node Inputs

Activate

Sets whether the **ScrollBox** 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.

UIe:ScrollBox:SetIsVerticalScrollingEnabled Node

Sets whether the **ScrollBox** allows vertical scrolling.

Node Inputs

Activate

Sets whether the **ScrollBox** 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.

UIe:ScrollBox:GetScrollOffset Node

Gets the scroll offset of the **ScrollBox**.

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**.

UIe:ScrollBox:SetScrollOffset Node

Sets the scroll offset of the **ScrollBox**.

Node Inputs

Activate

Sets the scroll offset of the **ScrollBox**.

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.

UIe:ScrollBox:GetScrollOffsetChangedActionName Node

Gets the action triggered when the **ScrollBox** drag is completed.

Node Inputs

Activate

Updates the output.

ElementName

Name of the element.

Node Output

ChangedAction

The action name.

UIe: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.

ElementName

Name of the element.

ChangedAction

The action name.

Node Output

Done

Sends a signal when the node's action is finished.

UIe:ScrollBox:GetScrollOffsetChangingActionName Node

Gets the action triggered while the **ScrollBox** is being dragged.

Node Inputs

Activate

Updates the output.

ElementName

Name of the element.

Node Output

ChangingAction

The action name.

UIe: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.

ElementName

Name of the element.

ChangingAction

The action name.

Node Output

Done

Sends a signal when the node's action is finished.

UIe:ScrollBox:GetSnapGrid Node

Gets the snapping grid of the **ScrollBox**.

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**.

UIe:ScrollBox:SetSnapGrid Node

Sets the snapping grid of the **ScrollBox**.

Node Inputs

Activate

Sets the snapping grid of the **ScrollBox**.

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.

UIe:ScrollBox:GetSnapMode Node

Gets the snap mode for the **ScrollBox**.

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

UIe:ScrollBox:SetSnapMode Node

Sets the snap mode for the **ScrollBox**.

Node Inputs

Activate

Sets the snap mode for the **ScrollBox**.

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.

UIe:ScrollBox:GetHorizontalScrollBarVisibility Node

Gets horizontal scroll bar 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 scroll bar visibility behavior.

Valid values: **0** = AlwaysVisible | **1** = AutoHide | **2** = AutoHideAndResizeViewArea

UIe:ScrollBox:SetHorizontalScrollBarVisibility Node

Sets horizontal scroll bar 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.

ScollBarVisibility

An integer representing the scroll bar visibility behavior.

0 = AlwaysVisible | **1** = AutoHide | **2** = AutoHideAndResizeViewArea.

Node Output

Done

Sends a signal when the node's action is finished.

UIe:ScrollBox:GetVerticalScrollBarVisibility Node

Gets vertical scroll bar 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 scroll bar visibility behavior.

Valid values: **0** = AlwaysVisible | **1** = AutoHide | **2** = AutoHideAndResizeViewArea

UIe:ScrollBox:SetVerticalScrollBarVisibility Node

Sets vertical scroll bar 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.

ScollBarVisibility

An integer representing the scroll bar visibility behavior.

0 = AlwaysVisible | **1** = AutoHide | **2** = AutoHideAndResizeViewArea.

Node Output

Done

Sends a signal when the node's action is finished.

UIe:ScrollBox:GetHorizontalScrollBarEntity Node

Gets the horizontal scroll bar 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.

UIe:ScrollBox:SetHorizontalScrollBarEntity Node

Sets the horizontal scroll bar element for the **ScrollBox**.

Node Inputs

Activate

Sets the horizontal scroll bar 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.

UIe:ScrollBox:GetVerticalScrollBarEntity Node

Gets the vertical scroll bar 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.

UIe:ScrollBox:SetVerticalScrollBarEntity Node

Sets the vertical scroll bar element for the **ScrollBox**.

Node Inputs

Activate

Sets the vertical scroll bar 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.

UIe ScrollBar Component Nodes

Flow Graph is deprecated and will be removed in a future version of Lumberyard. Learn about Script Canvas (p. 682), Lumberyard's new visual scripting environment.

Use the following flow graph nodes to perform actions on the **Scrollbar** component.

UIe: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.

UIe: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.

UIe: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`.

UIe: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.

UIe: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`.

UIe: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.

UIe: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`.

UIe: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.

UIe: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.

UIe: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.

UIe: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.

UIe: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.

UIe Slider Component Nodes

Flow Graph is deprecated and will be removed in a future version of Lumberyard. Learn about Script Canvas (p. 682), Lumberyard's new visual scripting environment.

Use the following flow graph nodes to perform actions on the slider component.

UIe:Slider:GetFillEntity Node

Gets the fill element.

Node Inputs

Activate

Updates the output.

ElementName

Name of the element.

Node Output

FillElement

The fill element.

UIe: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.

UIe: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**.

UIe: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.

UIe: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.

UIe: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.

UIe:Slider:GetTrackEntity Node

Gets the track element.

Node Inputs

Activate

Updates the output.

ElementName

Name of the element.

Node Output

Track

The track element.

UIe: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.

UIe: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**.

UIe: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.

UIe: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.

UIe: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.

UIe: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.

UIe: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.

UIe Text Component Nodes

Flow Graph is deprecated and will be removed in a future version of Lumberyard. Learn about Script Canvas (p. 682), Lumberyard's new visual scripting environment.

Use the following flow graph nodes to perform actions on the text component.

UIe: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**.

UIe: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.

UIe: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.

UIe: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.

UIe: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**.

UIe: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.

UIe: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

UIe: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.

UIe: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**.

UIe: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.

UIe: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 ines are handled.

Valid values: `0` = No wrap | `1` = Wrap

UIe: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 ines are handled.

Valid values: `0` = No wrap | `1` = Wrap

Node Output

Done

Sends a signal when the node's action is finished.

UIe Text Input Component Nodes

Flow Graph is deprecated and will be removed in a future version of Lumberyard. Learn about Script Canvas (p. 682), Lumberyard's new visual scripting environment.

Use the following flow graph nodes to perform actions on the text input component.

UIe: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.

UIe: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.

UIe: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**.

UIe: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.

UIe: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.

UIe: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.

UIe: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

UIe: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.

UIe: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.

UIe: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.

UIe: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

UIe: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.

UIe: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**.

UIe: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.

UIe:TextInput:GetTextEntity Node

Gets the text element.

Node Inputs

Activate

Updates the output.

ElementName

Name of the element.

Node Output

TextElement

The text element.

UIe: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.

UIe: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**.

UIe: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.

UIe Transform Component Nodes

Flow Graph is deprecated and will be removed in a future version of Lumberyard. Learn about Script Canvas (p. 682), Lumberyard's new visual scripting environment.

Use the following flow graph nodes to perform actions on the transform component.

UIe:Transform:GetCanvasPosition Node

Gets the position of an element in canvas space.

Node Inputs

Activate

Updates the outputs.

ElementName

Name of the element.

ElementID

The element Id. Used if `ElementName` is empty.

Node Outputs

XPosition

The x position of the element in canvas space.

YPosition

The y position of the element in canvas space.

UIe:Transform:SetCanvasPosition Node

Sets the position of an element in canvas space.

Node Inputs

Activate

Sets the position of the element in canvas space.

ElementName

Name of the element.

ElementID

The element Id. Used if `ElementName` is empty.

XPosition

The x position of the element in canvas space.

YPosition

The y position of the element in canvas space.

Node Outputs

Done

Sends a signal when the node's action is finished.

UIe:Transform:GetLocalPosition Node

Gets the relative position of an element from the center of the element's anchors.

Node Inputs

Activate

Updates the output.

ElementName

Name of the element.

ElementID

The element Id. Used if `ElementName` is empty.

Node Outputs

XPosition

The relative x position of the element.

YPosition

The relative y position of the element.

UIe:Transform:SetLocalPosition Node

Sets the relative position of an element from the center of the element's anchors.

Node Inputs

Activate

Sets the relative position of the element.

ElementName

Name of the element.

ElementID

The element Id. Used if `ElementName` is empty.

XPosition

The relative x position of the element.

YPosition

The relative y position of the element.

Node Outputs

Done

Sends a signal when the node's action is finished.

UIe:Transform:GetZRotation Node

Gets the z rotation of an element.

Node Inputs

Activate

Updates the output.

ElementName

Name of the element.

ElementID

The element Id. Used if `ElementName` is empty.

Node Outputs

Value

The z rotation of the element.

UIe:Transform:SetZRotation Node

Sets the z rotation of an element.

Node Inputs

Activate

Sets the z rotation of the element.

ElementName

Name of the element.

ElementID

The element Id. Used if `ElementName` is empty.

Node Outputs

Done

Sends a signal when the node's action is finished.

UIe:Transform:MoveCanvasPositionBy Node

Moves an element in canvas space.

Node Inputs

Activate

Moves the element.

ElementName

Name of the element.

ElementID

The element Id. Used if `ElementName` is empty.

XOffset

The x offset value.

YOffset

The y offset value.

Node Outputs

Done

Sends a signal when the node's action is finished.

UIe:Transform:MoveLocalPositionBy Node

Moves an element relative to the center of the element’s anchors.

Node Inputs

Activate

Moves the element.

ElementName

Name of the element.

ElementID

The element Id. Used if `ElementName` is empty.

XOffset

The x offset value.

YOffset

The y offset value.

Node Outputs

Done

Sends a signal when the node's action is finished.

UIe Animation Node

The UI animation node consists of the following node inputs and outputs:

UIe: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

Flow Graph is deprecated and will be removed in a future version of Lumberyard. Learn about Script Canvas (p. 682), Lumberyard's new visual scripting environment.

The **UI** flow graph node set is the original, and now legacy, version of the **UI** flow graph nodes.

Use the **UIe** (p. 1626) flow graph node set for best results when creating new flow graph nodes for your user interface.

Topics

UI Canvas Nodes

Flow Graph is deprecated and will be removed in a future version of Lumberyard. Learn about Script Canvas (p. 682), Lumberyard's new visual scripting environment.

The **UI Canvas** flow graph nodes have been superseded by the **UIe** Canvas (p. 1626) flow graph nodes. For best results, use the **UIe** Canvas (p. 1626) 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:GetIsPixelAligned 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

Flow Graph is deprecated and will be removed in a future version of Lumberyard. Learn about Script Canvas (p. 682), Lumberyard's new visual scripting environment.

The **UI** component flow graph nodes have been superseded by the **UIe** Component (p. 1630) flow graph nodes. For best results, use the **UIe** Component (p. 1630) flow graph nodes.

These flow graph nodes perform actions on UI elements through their components.

Topics

UI Button Component Nodes

Flow Graph is deprecated and will be removed in a future version of Lumberyard. Learn about Script Canvas (p. 682), Lumberyard's new visual scripting environment.

The **UI** Button component flow graph nodes have been superseded by the **UIe** Button (p. 1631) flow graph nodes. For best results, use the **UIe** Button (p. 1631) 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

Flow Graph is deprecated and will be removed in a future version of Lumberyard. Learn about Script Canvas (p. 682), Lumberyard's new visual scripting environment.

The **UI** component flow graph nodes have been superseded by the **UIe** Checkbox (p. 1632) component flow graph nodes. For best results, use the **UIe** Checkbox (p. 1632) 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 check box 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 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.

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 DynamicLayout Component Nodes

> Flow Graph is deprecated and will be removed in a future version of Lumberyard. Learn about Script Canvas (p. 682), Lumberyard's new visual scripting environment.

The UI DynamicLayout flow graph nodes have been superseded by the UIe DynamicLayout (p. 1636) flow graph nodes. For best results, use the UIe Dynamic Layout flow graph nodes.

Use the following flow graph nodes to perform actions on the dynamic layout component.

UI:DynamicLayout:SetNumChildElements Node

Sets the number of child elements of the layout element. The child elements are cloned from a prototype element.

Node Inputs

Activate

Sets the number of child elements.

ElementName

Name of the layout element.

ElementId

The element Id of the layout element. Used if `ElementName` is empty.

NumChildElements

The number of child elements.

Node Output

Done

Sends a signal when the node's action is finished.

UI DynamicScrollbox Component Nodes

Flow Graph is deprecated and will be removed in a future version of Lumberyard. Learn about Script Canvas (p. 682), Lumberyard's new visual scripting environment.

The **UI DynamicScrollBox** flow graph nodes have been superseded by the UIe DynamicScrollBox (p. 1637) flow graph nodes. For best results, use the **UIe DynamicScrollBox** flow graph nodes.

Use the following flow graph nodes to perform actions on the dynamic scroll box component.

UI:DynamicScrollBox:GetLocationIndexOfChild Node

Gets the index of the specified child of the dynamic scroll box's content element. Use this node rather than `UI:Element:GetIndexOfChild`, since dynamic scroll boxes create only the minimum number of children for display.

Node Inputs

Activate

Updates the output.

ElementName

Name of the dynamic scroll box element.

ElementId

The element Id of the dynamic scroll box element. Used if `ElementName` is empty.

ChildElementId

The element Id of the child.

Node Output

Index

The index of the child.

UI:DynamicScrollBox:RefreshContent Node

Refreshes the dynamic scroll box by retrieving the number of the content element's children, setting up the content element according to the number of children, and notifying listeners of the child elements that are visible.

Node Inputs

Activate

Refreshes the content.

ElementName

Name of the dynamic scroll box element.

ElementId

The element Id of the dynamic scroll box element. Used if `ElementName` is empty.

Node Output

Done

Sends a signal when the node's action is finished.

UI Element Node

Flow Graph is deprecated and will be removed in a future version of Lumberyard. Learn about Script Canvas (p. 682), Lumberyard's new visual scripting environment.

The **UI** Element flow graph nodes have been superseded by the **UIe** Element (p. 1638) flow graph nodes. For best results, use the **UIe** Element (p. 1638) flow graph nodes.

Use the following flow graph nodes to perform actions on an element.

UI:Element:IsEnabled Node

Gets whether the element is enabled.

Node Inputs

Activate

Updates the outputs.

CanvasID

Unique identifier of the element's canvas.

ElementID

Unique identifier of the element.

Node Output

State

The enabled state of the 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:Element:GetChildAtIndex Node

Gets the child of an element at the specified index.

Node Inputs

Activate

Updates the outputs.

CanvasID

Unique identifier of the element's canvas.

ElementID

Unique identifier of the element.

ChildIndex

The index of the child.

Node Output

ChildElementId

The element Id of the child element.

UI:Element:GetIndexOfChild Node

Gets the index of the specified child.

Node Inputs

Activate

Updates the outputs.

CanvasID

Unique identifier of the element's canvas.

ElementID

Unique identifier of the element.

ChildElementId

Unique identifier of the child element.

Node Output

IndexOfChild

The index of the child element.

UI:Element:GetNumChildElements Node

Gets the number of children of an element.

Node Inputs

Activate

Updates the outputs.

CanvasID

Unique identifier of the element's canvas.

ElementID

Unique identifier of the element.

Node Output

NumChildElements

The number of child elements.

UI:Element:GetParent Node

Gets the parent of an element.

Node Inputs

Activate

Updates the outputs.

CanvasID

Unique identifier of the element's canvas.

ElementID

Unique identifier of the element.

Node Output

ParentElementId

The element Id of the parent element.

UI Fader Component Nodes

Flow Graph is deprecated and will be removed in a future version of Lumberyard. Learn about Script Canvas (p. 682), Lumberyard's new visual scripting environment.

The **UI** Fader component flow graph nodes have been superseded by the **UIe** Fader (p. 1641) component flow graph nodes. For best results, use the **UIe** Fader (p. 1641) 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

Flow Graph is deprecated and will be removed in a future version of Lumberyard. Learn about Script Canvas (p. 682), Lumberyard's new visual scripting environment.

The **UI** Image component flow graph nodes have been superseded by the **UIe** Image (p. 1643) component flow graph nodes. For best results, use the **UIe** Image (p. 1643) 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. 1721).

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. 1721).

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

Flow Graph is deprecated and will be removed in a future version of Lumberyard. Learn about Script Canvas (p. 682), Lumberyard's new visual scripting environment.

The **UI** Interactable component flow graph nodes have been superseded by the **UIe** Interactable (p. 1647) component flow graph nodes. For best results, use the **UIe** Interactable (p. 1647) 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

> Flow Graph is deprecated and will be removed in a future version of Lumberyard. Learn about Script Canvas (p. 682), Lumberyard's new visual scripting environment.

The **UI** Layout column component flow graph nodes have been superseded by the **UIe** Layout column (p. 1647) component flow graph nodes. For best results, use the **UIe** Layout column (p. 1647) 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

Flow Graph is deprecated and will be removed in a future version of Lumberyard. Learn about Script Canvas (p. 682), Lumberyard's new visual scripting environment.

The **UI** Layout grid component flow graph nodes have been superseded by the **UIe** Layout grid (p. 1650) component flow graph nodes. For best results, use the **UIe** Layout grid (p. 1650) 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

Flow Graph is deprecated and will be removed in a future version of Lumberyard. Learn about Script Canvas (p. 682), Lumberyard's new visual scripting environment.

The **UI** Layout row component flow graph nodes have been superseded by the **UIe** Layout row (p. 1655) component flow graph nodes. For best results, use the **UIe** Layout row (p. 1655) 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

Flow Graph is deprecated and will be removed in a future version of Lumberyard. Learn about Script Canvas (p. 682), Lumberyard's new visual scripting environment.

The **UI** Mask component flow graph nodes have been superseded by the **UIe** Mask (p. 1658) component flow graph nodes. For best results, use the **UIe** Mask (p. 1658) 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:GetIsMaskingEnabled 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

Flow Graph is deprecated and will be removed in a future version of Lumberyard. Learn about Script Canvas (p. 682), Lumberyard's new visual scripting environment.

The **UI** Scrollbox component flow graph nodes have been superseded by the **UIe** Scrollbox (p. 1661) component flow graph nodes. For best results, use the **UIe** Scrollbox (p. 1661) 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:GetIsHorizontalScrollingEnabled 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:SetIsHorizontalScrollingEnabled 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:GetIsScrollingConstrained 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:GetIsVerticalScrollingEnabled 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 scroll bar 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 scroll bar visibility behavior.

Valid values: **0** = AlwaysVisible | **1** = AutoHide | **2** = AutoHideAndResizeViewArea

UI:ScrollBox:SetHorizontalScrollBarVisibility Node

Sets horizontal scroll bar 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.

ScollBarVisibility

An integer representing the scroll bar 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 scroll bar 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 scroll bar visibility behavior.

Valid values: **0** = AlwaysVisible | **1** = AutoHide | **2** = AutoHideAndResizeViewArea

UI:ScrollBox:SetVerticalScrollBarVisibility Node

Sets vertical scroll bar 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.

ScollBarVisibility

An integer representing the scroll bar 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 scroll bar 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 scroll bar element for the **ScrollBox**.

Node Inputs

Activate

Sets the horizontal scroll bar 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 scroll bar 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 scroll bar element for the **ScrollBox**.

Node Inputs

Activate

Sets the vertical scroll bar 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

Flow Graph is deprecated and will be removed in a future version of Lumberyard. Learn about Script Canvas (p. 682), Lumberyard's new visual scripting environment.

The **UI** ScrollBar component flow graph nodes have been superseded by the **UIe** ScrollBar (p. 1672) component flow graph nodes. For best results, use the **UIe** ScrollBar (p. 1672) 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 scroll bar.

Node Inputs

Activate

Updates the output.

CanvasID

Unique identifier of the element's canvas.

ElementID

Unique identifier of the element.

Node Output

Value

The scroll bar 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 scroll bar.

CanvasID

Unique identifier of the element's canvas.

ElementID

Unique identifier of the element.

Value

The scroll bar 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

Flow Graph is deprecated and will be removed in a future version of Lumberyard. Learn about Script Canvas (p. 682), Lumberyard's new visual scripting environment.

The **UI** Slider component flow graph nodes have been superseded by the **UIe** Slider (p. 1677) component flow graph nodes. For best results, use the **UIe** Slider (p. 1677) 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

Flow Graph is deprecated and will be removed in a future version of Lumberyard. Learn about Script Canvas (p. 682), Lumberyard's new visual scripting environment.

The **UI** Text component flow graph nodes have been superseded by the **UIe** Text (p. 1684) component flow graph nodes. For best results, use the **UIe** Text (p. 1684) 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 ines 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 ines 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

Flow Graph is deprecated and will be removed in a future version of Lumberyard. Learn about Script Canvas (p. 682), Lumberyard's new visual scripting environment.

The **UI** Text Input component flow graph nodes have been superseded by the **UIe** Text Input (p. 1688) component flow graph nodes. For best results, use the **UIe** Text Input (p. 1688) 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. 1701) flow graph node. For best results, use the **UIe** Animation (p. 1701) 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

Unique ID of UI canvas that contains the animation sequence.

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).

Virtual Reality

Lumberyard's virtual reality (p. 1858) 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. 1782) in the Project Configurator and then rebuild your project (p. 1060). 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. 1060) for other head-mounted devices.

Use console variables (CVARs) (p. 1784) 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.

For information on Lua scripting functions for VR, see VR Lua Functions.

Topics

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.

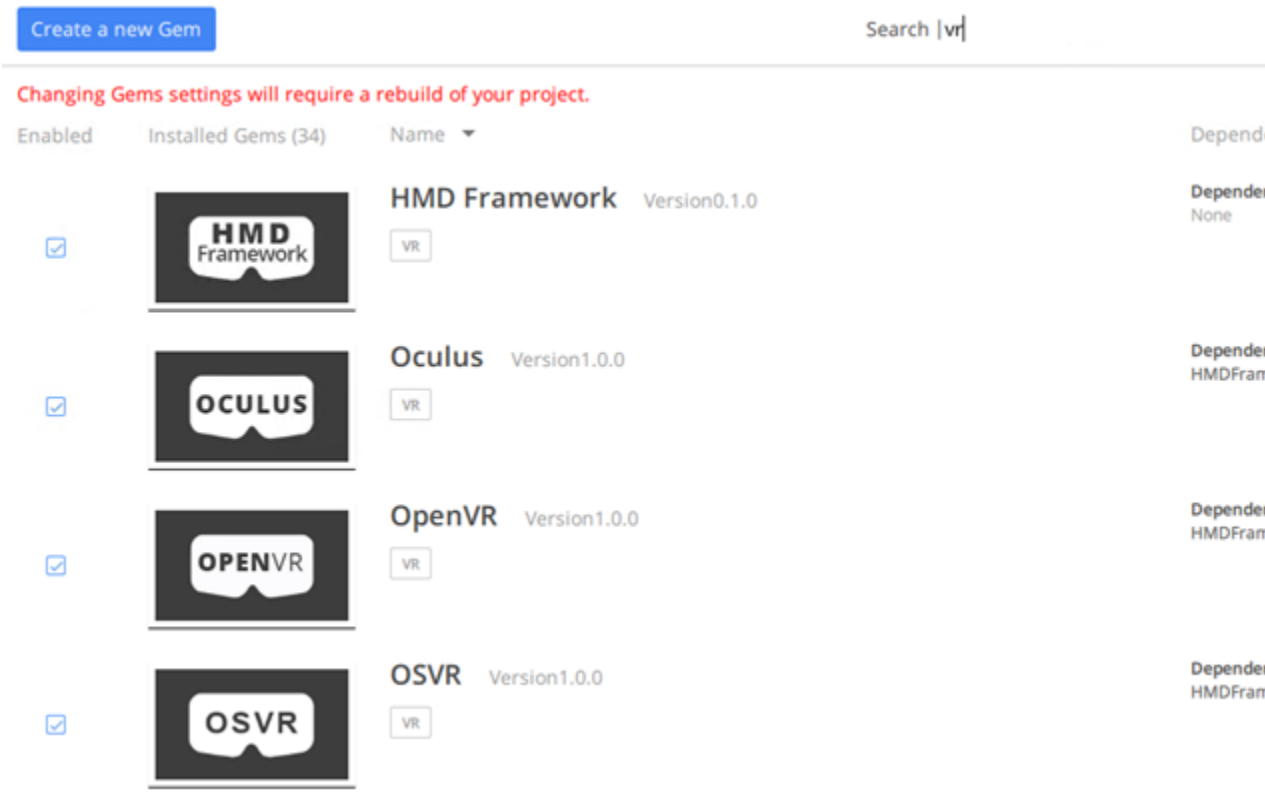
Supported HMDs include:

- **Oculus** – Oculus Rift HMD
- **OpenVR** – HTC Vive HMD
- **OSVR** (p. 1783) – 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. Type VR into the search tool. Enable the **HMD Framework** Gem and 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. 1060) before the gem(s) will function in Lumberyard Editor.

Topics

- Configuring OSVR (p. 1783)

Configuring OSVR

If you are using an OSVR HDK, use the following procedure to ensure that Lumberyard correctly detects and identifies your head-mounted display (HMD).

To configure Lumberyard to use the OSVR head-mounted display

1. If you have not already done so, download and install Lumberyard (p. 14).
2. Connect and set up your OSVR HDK.
3. Open http://www.osvr.org/getting-started.html.
4. Download and run the HDK Windows Installer.

   The tool is installed on the system tray.
5. Right-click the tool in your system tray, and then choose **Launch Firmware Utility**.

   Update your OSVR HDK firmware and your IR camera's firmware, if necessary.
6. Follow the steps in Configuring your Project for Virtual Reality (p. 1782) to set up gems in your project or Virtual Reality Project Sample (p. 1500) to launch a virtual reality sample project.

Configuring Required Console Variables

Console variables – Console variables (CVARs) are a type of variable that you can manipulate in Lumberyard's console interface. For more information, see Using the Console Window (p. 93).

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.

r_ssdoHalfRes = 3

Applies screen space directional occlusion (SSDO) (p. 1858) 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.

Known Issue

There is a known issue in the VR system where having CVar `e_CheckOcclusion` set to `1` (which is its default) can cause occasional flickering of the Entity geometry.

The `e_CheckOcclusion` CVar performs a visible check in check occlusion job.

The VR sample levels have been updated with `e_CheckOcclusion = 0` to prevent the flickering issue. We recommend you set this CVar to `0` in other projects as well.

Using the InstantVR Slice

The **instantVR** slice is a set of entities, scripts, and assets that provide basic pieces of VR functionality to give you a starting point from which you can build your own VR application.

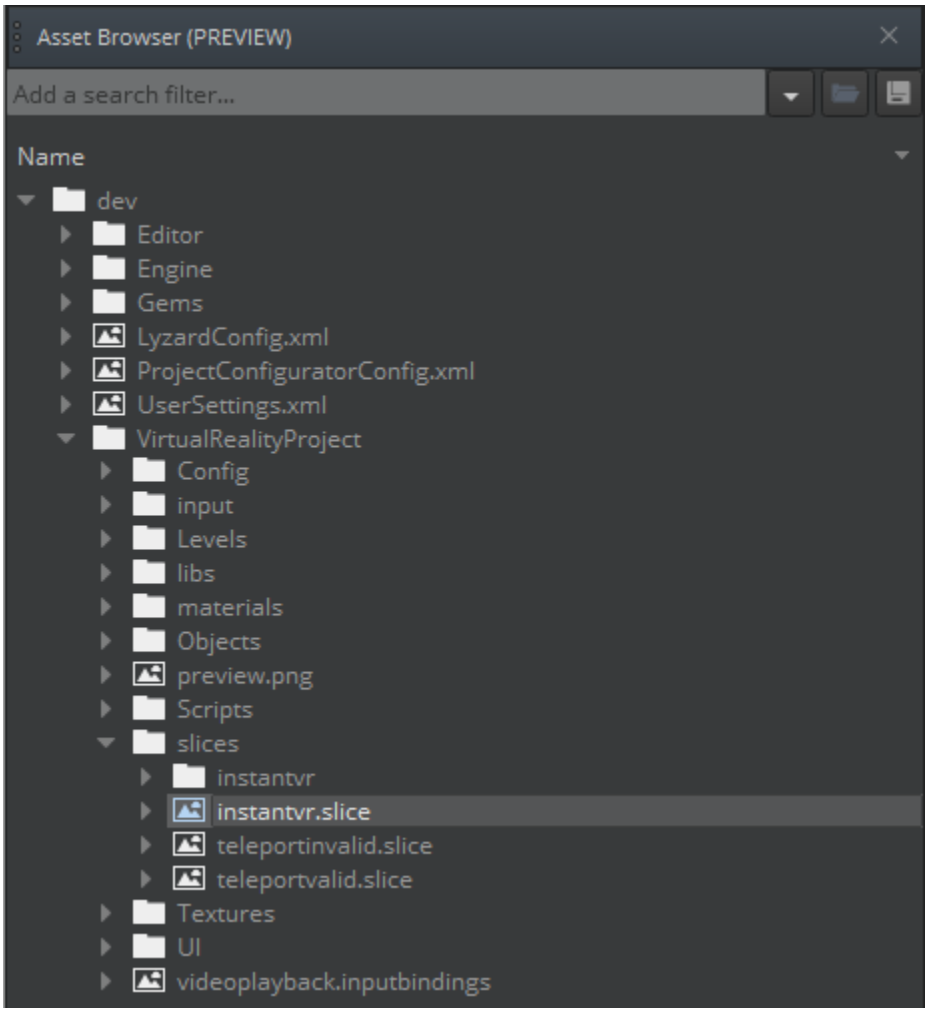
The VR functionality provided in this slice includes:

- Implementation of models-tracked controllers
- Teleportation using a navigation mesh to define a valid area
- Generation of a starting navigation area

The **instantVR** slice is part of the Virtual Reality Project sample level. Follow the instructions in VirtualRealityProject (p. 1500) to download, install, and select the Virtual Reality Project before performing the following procedure.

To use the instantVR slice

1. In the **Asset Browser** (p. 179), navigate to `dev\VirtualRealityProject\slices\`.



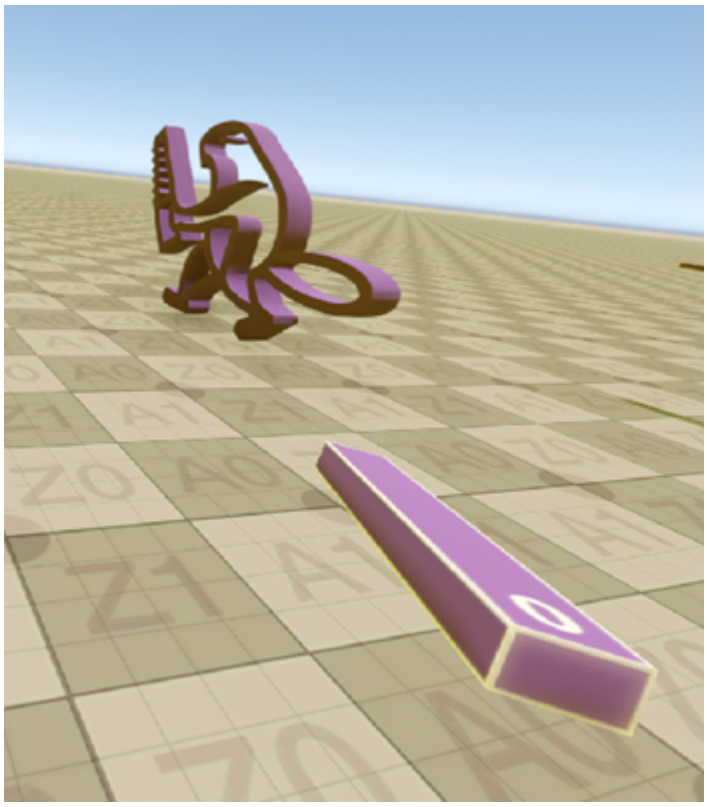
2. Drag `instantvr.slice` into the viewport.

   The Lumberyard beaver is the starting location in this slice. Two controllers and a navigation area also appear.

3. Click the **VR Preview** button to enable VR preview, and then press **Ctrl+G** to run your level.

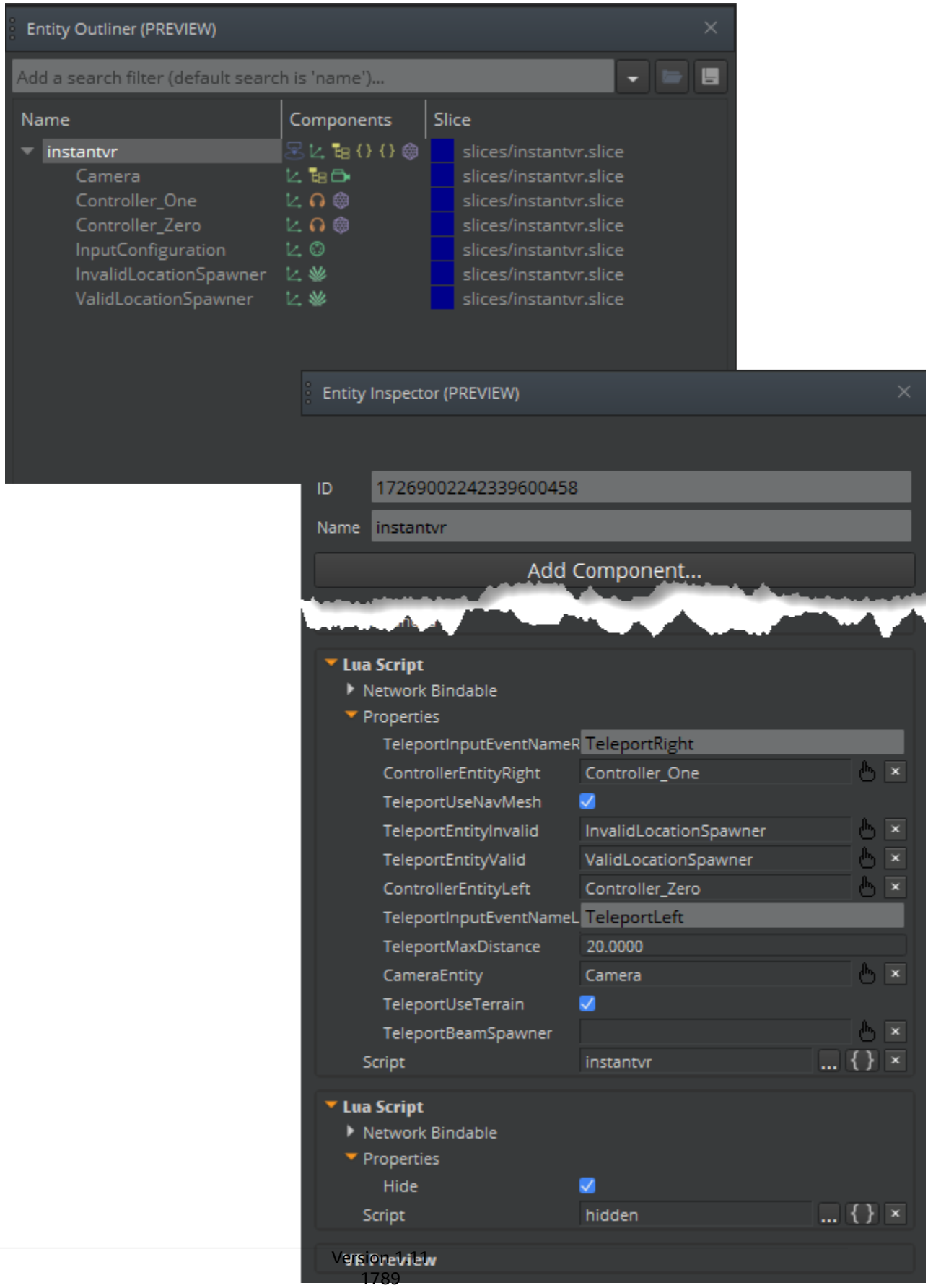
   You can use the trigger buttons on your controllers to teleport around the space.


Z: 32.00
XYZ
Lock Sca
Speed: 1.00
.1
1
10
Coll
AI/Physics
|>
Sync Play
oto positio
lute Audic
VR Preview



InstantVR Lua Script Properties

You can view **instantVR**'s assets in the **Entity Outliner**. The **instantVR** slice contains a Lua script called `instantVR`, which includes many of the child entities that add controller tracking and provide teleport support.


Entity Outliner (PREVIEW)
Add a search filter (default search is 'name')...
Name
Components
Slice
instantvr
Camera
Controller_One
Controller_Zero
InputConfiguration
InvalidLocationSpawner
ValidLocationSpawner
slices/instantvr.slice
Entity Inspector (PREVIEW)
ID
1726900224233960045 8
Name
instantvr
Add Component...
Lua Script
Network Bindable
Properties
TeleportInputEventNameR
TeleportRight
ControllerEntityRight
Controller_One
TeleportUseNavMesh
TeleportEntityInvalid
InvalidLocationSpawner
TeleportEntityValid
ValidLocationSpawner
ControllerEntityLeft
Controller_Zero
TeleportInputEventNameL
TeleportLeft
TeleportMaxDistance
20.0000
CameraEntity
Camera
TeleportUseTerrain
TeleportBeamSpawner
Script
instantvr
Lua Script
Network Bindable
Properties
Hide
Script
hidden

The `instantVR` Lua script has the following properties:

TeleportInputEventNameRight

Name of the input event that triggers the right controller teleport.

ControllerEntityRight

Name of the right controller entity.

TeleportUseNavMesh

If selected, teleport function uses the navigation area for validation.

> **Note**
> If you select **TeleportUseNavMesh** without selecting **TeleportUseTerrain**, then teleport ray cast uses only the nav mesh to determine the valid area. If it finds no nav mesh, it displays an 'invalid' indicator on terrain.
> If both are selected, then teleport validation uses a ray cast to the terrain and to the navigation mesh. If both beams hit terrain and nav mesh, then the teleport location is valid, otherwise it is invalid.

TeleportEntityInvalid

Name of the entity that spawns the invalid teleport location entity.

TeleportEntityValid

Name of the entity that spawns the valid teleport location entity.

TeleportInputEventNameLeft

Name of the input event that triggers the left controller teleport.

TeleportMaxDistance

Maximum distance for teleporting.

CameraEntity

Name of the camera entity, which determines the transform of the HMD and controllers.

TeleportUseTerrain

If selected, the teleport function uses the terrain for validation.

> **Note**
> If you select **TeleportUseTerrain** without selecting **TeleportUseNavMesh**, then the teleport ray cast uses the terrain for validation only. Teleport will be valid as long as it is in contact with terrain.
> If both are selected, then teleport validation uses a ray cast to the terrain and to the navigation mesh. If both beams hit terrain and nav mesh, then the teleport location is valid, otherwise it is invalid.

TeleportBeamSpawner

The name of the teleport beam arc entity along which to spawn entities, though this does not follow the ray cast line exactly.

To customize the `instantVR` Lua script, try the following modifications:

- Change the controllers to a different model by changing the **Static asset** of the **Static Mesh** component.
- Change the **TeleportMaxDistance** to a large value.
- Deselect **TeleportUseNavMesh** and climb the walls.
- Open the **InvalidLocationSpawner** and **ValidLocationSpawner** dynamic slices and change the entities they spawn.

InstantVR Known Issue

If you modify the size and position of the navigation area in the Lumberyard Editor, you must manually rebuild by clicking **Game**, **AI**, **Generate Triangulation**. Restarting the Lumberyard Editor or reloading the level also automatically rebuilds the navigation area.

Setting Up Virtual Reality with Flow Graph

You can use flow graph modules to set up or script your virtual reality game.

The Virtual Reality (VR) flow graph nodes (p. 1034) are available for any attached head-mounted display.

You can also use Force Feedback Nodes (p. 869) to manage haptic feedback for motion controllers. You can select the device to send force feedback to and enable it to send events to the motion controllers. The Oculus and OpenVR APIs are configured to respond to force feedback events.

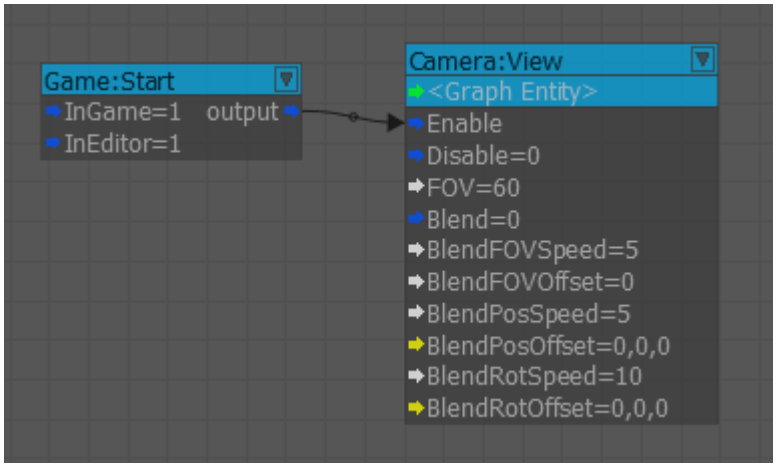
Setting Up a Basic Virtual Reality Flow Graph

The following flow graph (p. 754) 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. 412) at a specific location and enabling it with flow graph.

To specify a custom starting point

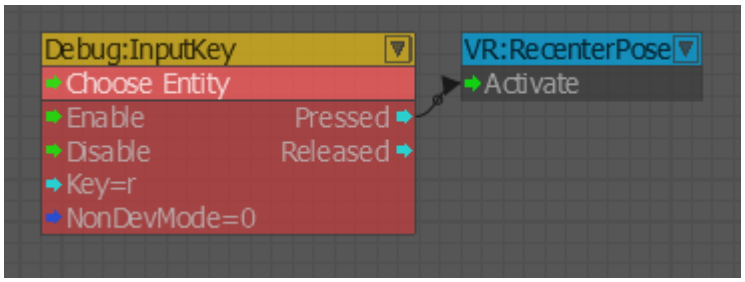
1. Place a game play camera (p. 412) at the desired location in your level. This is your default camera.
2. Right-click the camera entity and click **Create Flow Graph** to open the **Flow Graph** editor.
3. Drag a **Game:Start** node and **Camera:View** node onto your flow graph canvas.
4. Connect the **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. 754) 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.



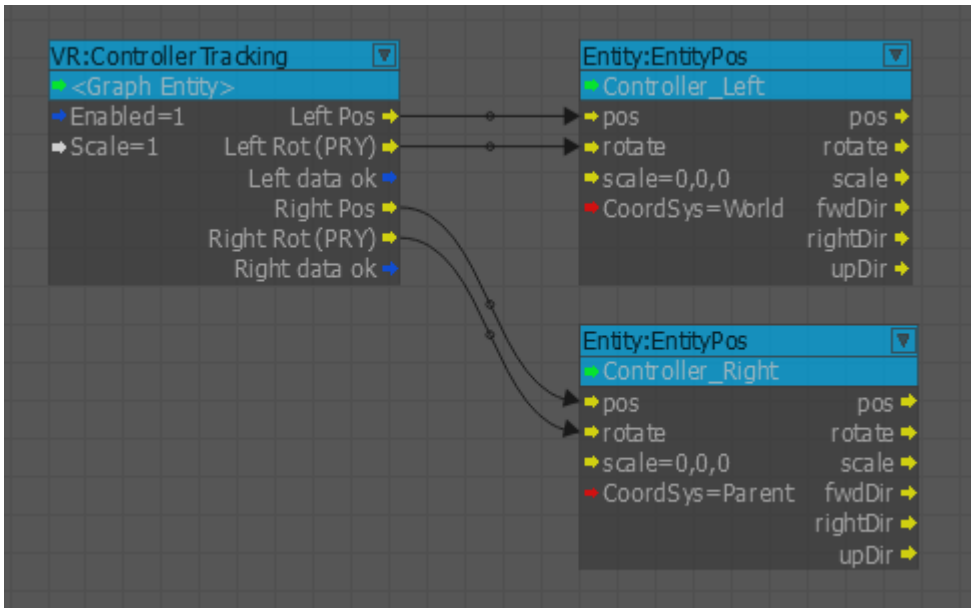
For your virtual reality game, you may want to place a graphical, virtual controller to represent where a physical controller is within the 3D space. You can use flow graph to add this graphical representation of a controller (for example, hands, weapons, and so on).

For this procedure, the default camera is the game play camera that you placed in the custom starting point (p. 1791) 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 **Left 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

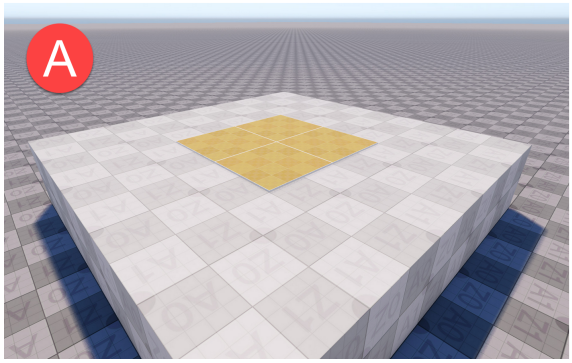
You can use flow graph to set up a custom playspace for OpenVR (Vive HTC head-mounted display [HMD]).

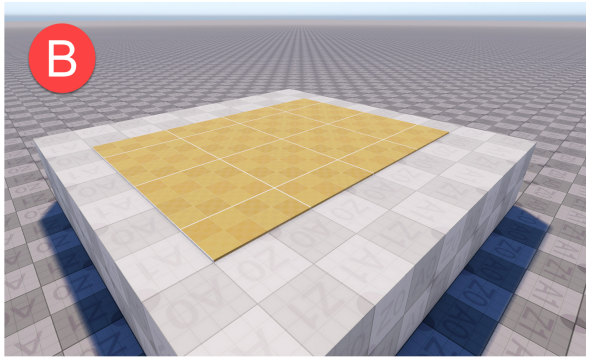
When a user first sets up their Vive HTC HMD, they must configure their playspace, or play area, according to their available space and room configuration. You can set up a flow graph to enable your game to access this information in order to spawn game objects within the user's reach and to create a 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. 1472). You may copy or modify this sample level for your own uses.

The following pictures show examples of:

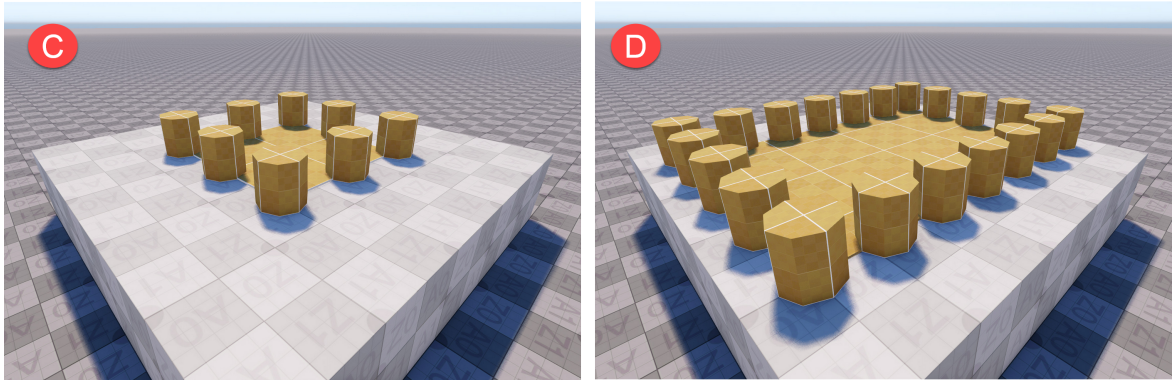
- **A** – 2m x 2m playspace
- **B** – 4m x 3m playspace



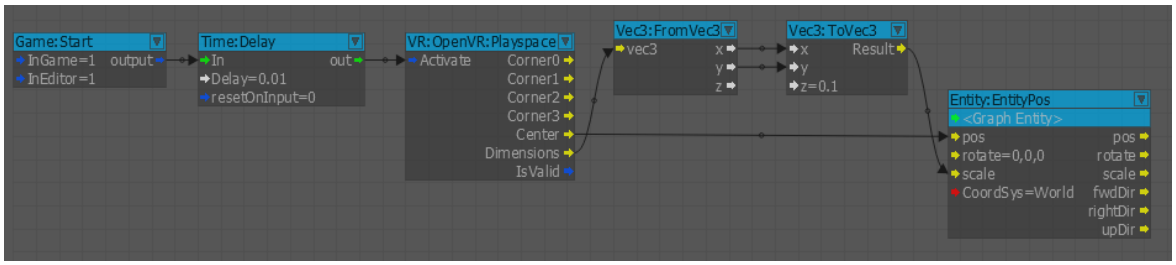


The following pictures show examples of spawning posts around the perimeter of a playspace with an area of:

- **C** – 2m x 2m
- **D** – 4m x 3m



The following flow graph from **VR_BoxGarden_Sample** in the Samples Project (p. 1472) 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.



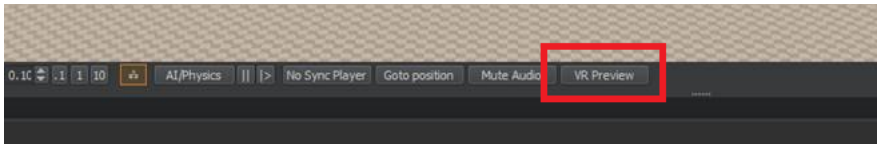
Previewing your Virtual Reality Project

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. 1848) 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**, **Play 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.

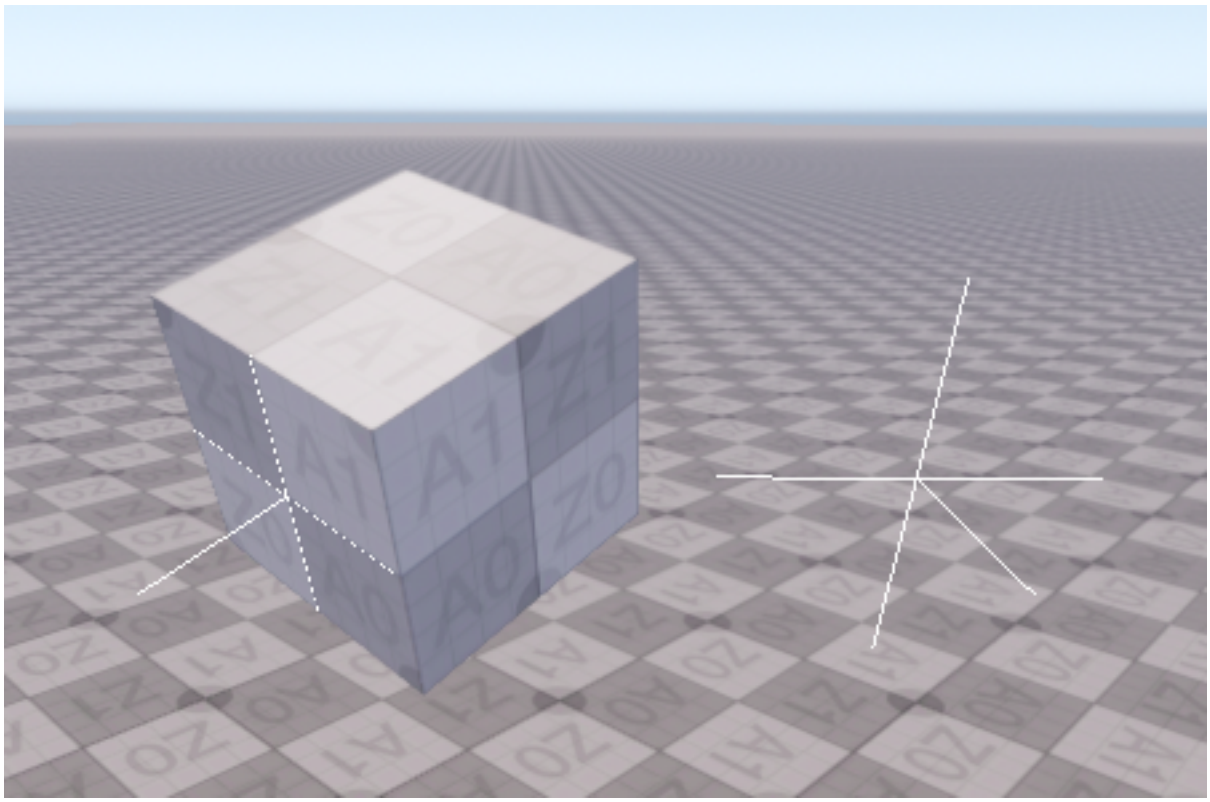
Debugging your Virtual Reality Project

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

- Enable one or both of the following console variable (p. 1784)s:
  - Set **hmd_debug_info** to **1** (enabled) – Enables display of debug information provided by the associated HMD SDK.
  - Set **hmd_debug_camera** to **1** (enabled) – Tests an editor-style debug camera at runtime. With this setting, players can use WASD keys to control the camera relative to the camera's facing direction, including the HMD. In regular (non-VR) mode, hold down the right mouse button to manipulate the camera's rotation.

When in debug mode, motion controllers appear as white crosshairs. That is, if you assigned an object or entity to represent the motion controller in the game play world, you will see it rendered with white crosshairs. The following picture shows two controllers, one with render geometry assigned, and the other without.



Using EBus Request Bus Interface for Virtual Reality

Use the following request function with the StereoRendererBus event bus (EBus) interface to communicate with other components of your game.

For more information about using the EBus interface, see Event Bus (EBus) in the *Amazon Lumberyard Developer Guide*.

IsRenderingToHMD

Returns true if the renderer is rendering to the HMD.

Parameters

None

Response

`True` or `False`

Scriptable

Yes

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 2.7 or later.

Lumberyard includes the Project Configurator (p. 28), 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

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`, `AppleTV`, `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)

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 operating systems 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', '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 dlls.
```

```
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
ADDITIONAL_COPYRIGHT_TABLE	Optional additional table of copyrights. To add a company specific copyright, add a name value pair to define the desired copyright statement for generated binaries and add the 'copyright_org' in your wscript definition
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:

```
    "project_name"      : "SamplesProject",
    "product_name"      : "Samples Project",
    "executable_name"   : "SamplesProjectLauncher",
    "code_folder"       : "Code/SamplesProject",
    "modules"           : ["SamplesProject"],
    "project_id": "{D882E365-54D6-586E-BD78-2650F3057D49}",
```

You can configure the following settings in the `project.json` file:

General settings

Value	Description
executable_name	Name of the built executable file: • **Dedicated server executables** – '_Server' is appended to the name • **Unit test executables** – '_UnitTest' is appended to the name
modules	(List) Base modules for the game
product_name	Externally-facing name of the product

The following values are only valid under the **android_settings** key:

Android settings

Value	Description
package_name	Android application package identifier. Used for generating the project-specific Java activity class and in `AndroidManifest.xml`. Must be in dot separated format. Default: "com.lumberyard.sdk"
version_number	Internal application version number. Used to set the `android:versionCode` tag in `AndroidManifest.xml`. Default: 1
version_name	Human readable version number. Used to set the `android: versionName` tag in the `AndroidManifest.xml`. Example: "1.2.3-beta" Default: "1.0.0.0"
orientation	Desired orientation of the Android application. Used to set the `android:screenOrientation` tag in `AndroidManifest.xml`. Expectable values can be found here: http://developer.android.com/guide/topics/manifest/activity-element.html#screen Default: "landscape"
icons	A map of icon override path(s) for each screen DPI type. All entries require a path that is either relative to <engine>\Code\<project>\Resources or an absolute resource path to the PNG image. Available sub-options: • default

Value	Description

	• Default image used if a specific DPI is not specified
	• mdpi
	  • Medium ~160 dpi
	• hdpi
	  • High ~240 dpi
	• xhdp”
	  • Extra high ~320 dpi
	• xxhdpi
	  • Extra-extra high ~480 dpi
	• xxxhdpi
	  • Extra-extra-extra high ~640 dpi
	
	For more information on Android screen DPI settings, consult the official Android documentation page https://developer.android.com/guide/practices/screens_support.html
	
	Example:
	
	`"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" }`
	
	Default: Empty

Value	Description

splash_screen	A map of splash screen override path(s) for each orientation and screen DPI type. All entries require a path that is either relative to <engine>\Code \<project>\Resources or an absolute resource path to the PNG image.
	
	Available sub-options (orientation type):
	
	• land
	  • Map of overrides for the landscape orientation
	• port
	  • Map of overrides for the portrait orientation
	
	Available sub-options for each orientation type:
	
	• default
	  • Default image used if a specific DPI is not specified
	• mdpi
	  • Medium ~160 dpi
	• hdpi
	  • High ~240 dpi
	• xhdp”
	  • Extra high ~320 dpi
	• xxhdpi
	  • Extra-extra high ~480 dpi
	
	For more information on Android screen DPI settings, consult the official Android documentation page https://developer.android.com/guide/practices/screens_support.html
	
	Example:
	
	`"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" :`

Value	Description
	`{ "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" } }` Default: Empty
place_assets_in_apk	Forces the assets to be packed in the APK in non-release builds. Must be either a 1 (Yes) or 0 (No). Default: 0
app_public_key	The application license key provided by Google Play. Required for using APK Expansion files or other Google Play Services. Example: `"MIIBIjANBgkqhkiG9w0BALuMbErYaRdAMIIBCgKCAQEAjvkl +K7rVASNkLExAmPlEoWwsxCX1vx7 uV3IIH5CQIZBRGT8KeYr6ThWlIPhSMKMImj7KxjdYcil8J0rwrVL3cm +02ntnBEemGvRVOKx kDaFc5Fw6tJVv3SJ6UyjVtehB7tJupaYdfFe9SVhW0xJZu2YsZLuMbE HJ7jG7YFVrQgxou1W/71QnExAmPlExi6mlsUJBFN4xADikNWZDlI70i +D5 oS3KeY/ KKd5WOU6BB8NmTY5VZVdUOd4VPRXrYMnRY7FjZJMPujLNvlrAJs5H/ G+0wUTR4SI61AiGJ iQIDAQAB"` Default: "NoKey"
app_obfuscator_salt	Application specific salt value for (un)obfuscation when using APK Expansion files Example: `"8d87473f5b24852836d6512abbd9e9b9869c208"` Default: ""
enable_obb_in_dev	Forces APK Expansion file mode in non-release builds. Value must be either "true" or "false". Default: "false"

Value	Description
use_main_obb	Specify if the "Main" APK Expansion file should be used. This option toggles APK Expansion file mode in release builds. Value must be either "true" or "false". Default: "false"
use_patch_obb	Specify if the "Patch" APK Expansion file should be used. This option toggles APK Expansion file mode in release builds. Value must be either "true" or "false". Default: "false"
enable_key_screen_on	Enabled or disable the screen wake lock (device won't go to sleep while the application is running). Value must be either "true" or "false". Default: "false"
rc_pak_job	Path to the RC job XML file used to override the normal PAK files generation used in release builds. Path must be relative to *<lumberyard_version>*\Bin64\rc Default: "RcJob_Generic_MakePaks.xml"
rc_obb_job	Path to the RC job XML file used to override the normal APK Expansion file(s) generation used in release builds. Path must be relative to *<lumberyard_version>*\Bin64\rc Default: " RCJob_Generic_Android_MakeObb.xml"

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 the my game",
    "visual_studio_name"    : "My Game",
    "comment"               : "This is meant to only compile tools on Windows.",
    "disable_game_projects" : true,
```

```
    "platforms" : ["win"],
    "configurations" : ["debug","profile"],

    "modules" :
    [
        "AzCore",
        "AzFramework",
        "AzToolsFramework",
        "GridMate",
        "LuaIDE",
        "Profiler"
    ]
}
```

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 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.

Overlapping lists are combined into a single list based on the build command.

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 `project.json`) 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.

Keyword Value	Description
*platform*_*configuration*_defines	• *platform* and *configuration* are optional values. • Possible values for *platform* and *configuration* can be determined from the `waf_branch_spec.py` 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 `lmbr_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

Build Module	Description	Consumers	Project Type
CryDedicatedServer	Build module for dedicated (server) game project launchers	FeaturesTestsDedicatedLauncher	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, CryAudioImplNoSound, CryAudioImplSDLMixer, CryAudioImplWwise, CrySystem, CryRenderD3D11, CryRenderOpenGL, CryRenderNULL, CryD3DCompilerStub	Shared Library (non-release), Static Library (performance, release)
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	FeaturesTestsLauncher	Executable
CryPipelineModule	Build module for pipeline components	CryTIFPluginCS4_11, CryExport2014, CryExport2015, CryExport2016, MayaCryExport22014, MayaCryExport22015, MayaCryExport22016	Custom

Build Module	Description	Consumers	Project Type
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
CryStandAlonePlugin	Build module for Lumberyard Editor standalone plugins (does not link to any engine shared libraries)	EditorAudioControlsBrowser, EditorLibrary, EditorNoSound,EditorWwise, FBXPlugin, FFMPEGPlugin, MetricsPlugin, PrototypeEditorPlugin, StateMachineEditorPlugin, UiEditor	Shared Library
CryUnitTestLauncher	Build module for unit test launchers	UnitTestLauncher	Executable

Waf Default Settings (default_settings.json)

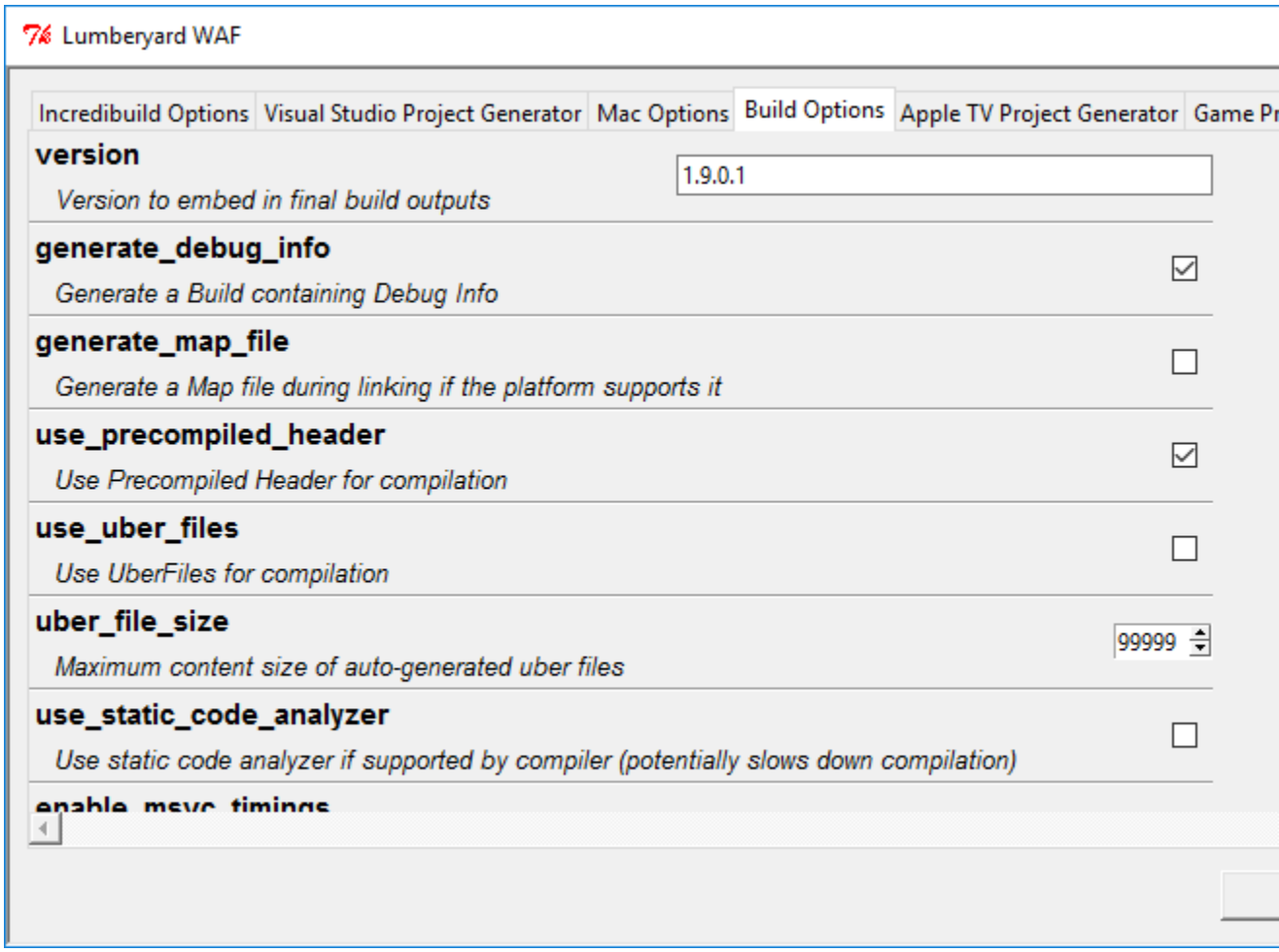
The Waf default settings file includes the default values for configurable Waf options. These values are used if custom values are not entered in the command line or `user_settings.options` cache file.

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:

```
lmbr_waf show_option_dialog
```



The tabs shown represent each section in the `user_settings.option` file.

Attribute	Override Parameter	Description	Default
Game Projects			
enabled_game_projects	-enabled-game-projects	Comma-separated list of game projects to enable for compiling	GameSDK, SamplesProject, MultiplayerSample
Incredibuild Options			
use_incredibuild	-i --use-incredibuild	Use Incredibuild if available.	False
Use Incredibuild for Windows PC builds. This requires at a minimum the	False		

Attribute	Override Parameter	Description	Default
Make and Build tools package.			
use_incredibuild_android	--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	--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 located 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

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_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. A value of 0 indicates that as many cores as needed will be used. Set a specific value to limit the number of cores used.	0
bootstrap_tool_param	--bootstrap-tool-param	Optional parameters to pass to SetupAssistantBatch.exe as part of the bootstrap process.	
bootstrap_third_party_3rdparty	--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_automatically	--generate-vs-projects-automatically	Automatically generate Visual Studio solutions.	True
visual_studio_solution_name	--visual-studio-solution-name	Name of the generated Visual Studio solution.	LumberyardSDK
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	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).	14
Android Project Generator			
generate_android_projects_automatically	--generate-android-projects-automatically	Automatically generates Android projects.	False
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	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	Deploys the executable .apk to the Android device.	True
deploy_android_replace_apk	--deploy-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	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

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 *`lumberyard_version`*`\dev\` of your project:
`lmbr_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 `lmbr_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:
[ ] rungame - 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_vs2015 - [debug, profile, performance, release, debug_dedicated,
 profile_dedicated, performance_dedicated, release_dedicated]"
[INFO] Configure "win_x64_vs2013 - [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] Removing the following Android target platforms due to "Compile For Android" not
 checked in Setup Assistant.
        -> android_armv7_gcc, android_armv7_clang
[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: `lmbr_waf 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: `lmbr_waf build_`*`platform_configuration`* `-p spec`

The following commands and options are vailable:

- `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 using Visual Studio 2015: `lmbr_waf build_win_x64_vs2015_release -p all`

Here's an example of how to clean the build release for Windows x64 using Visual Studio 2013: `lmbr_waf clean_win_x64_vs2013_release -p all`

Note
Combining the `clean_*` and `build_*` commands is the equivalent of performing a rebuild.

Configure command options

Command	Command Option	Description
build_*, clean_*	-p *spec name* --project-spec=*spec name*	The spec name to use to build or clean a project.
	--targets = *target1*,*target2*,...	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.
build_*, clean_*, configure	--profile-execution = (True|False)	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.
build_*	--execsolution=*VS_solution_path*	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 `.vcxproj` files themselves.
build_*	--file-filter=*source_files*	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.
build_*	--show-includes=(True|False)	Option to show the `#include` tree that a file uses during compilation. This option is only valid when `--file-filter` is specified.
build_*	--show-preprocessed-file = (True|False)	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 `\bintemp` based on the location of the source file. This option is only valid when `--file-filter` is specified.
build_*	--show-disassembly = (True|False)	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 `\bintemp` based on the location of the source file. This option is only valid when `--file-filter` is specified.
configure	--update-settings = (True|False)	Option to update the `user_settings.options` file with any values that are modified from the command line. For instance, if you want to modify the value of `use_uber_files` in `user_settings.options`, set `--use-uber-files=True` in the command line for `configure` and add `--update-settings=True` to apply the changes to `user_settings.options`.

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. 1809).

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

Configuration	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_vs2015, win_x64_vs2013, darwin_x64	Debug, profile, performance, release	Configuration to build the engine, editor, plugins, and tools
game_and_engine	win_x64_vs2015, win_x64_vs2013, darwin_x64, linux_x64	Debug, profile, performance, release	Configuration to build the engine and game project
pipeline	win_x64_vs2015, win_x64_vs2013	Debug, profile	Configuration to build tools for the asset pipeline
resource_compiler	win_x64_vs2015, win_x64_vs2013	Debug, profile	Configuration to build the Resource Compiler only

Build configuration options

Option	Description
--progress	Shows the build progress and updates in real time.

Option	Description
--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:

```
lmbr_waf build_win_x64_vs2015_profile_dedicated -p dedicated_server
```

> **Note**
> If you use Visual Studio 2015, type `vs2015`. If you use Visual Studio 2013, type `vs2013`.

Waf Supported Operating Systems and Compilers

This topic provides information about the operating systems and compilers that Waf supports. For more information about supported configurations, see Waf Commands and Options (p. 1814)

Supported operating systems

Operating system	Build environment	Waf short name
64-bit Windows	MSBuild / Visual Studio 2015	win_x64_vs2015
64-bit Windows	MSBuild / Visual Studio 2013	win_x64_vs2013

The following compilers are supported based on the build operating system.

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)
MSVC 14.0 (Visual Studio 2015)	Yes (except for CryExport2014 and CryExport2015)
GCC	No

Compiler	Windows 64-Bit
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	Container that groups compile settings and acts upon them recursively; useful for specifying options for particular files in a project
	
	For example, you can disable precompiled headers for a specific file using the following:
	
	`...`
	`additional_settings = Settings ( files = 'my_file.cpp', disable_pch=True )`
		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
cxxflags	Additional CXX flags to pass to the compiler	Y
defines	List of additional pre-processor defines for the project	Y
enable_rtti	Enable RTTI for a particular module	Y
export_defines	List of definitions to add to the module that has a 'use' dependency on the current module	Y
export_definitions	List of export definitions to export using the /DEF: compiler option	Y
export_includes	List of Additional include paths to add to the module that has a 'use' dependency on the current module	Y

Attribute	Description	Target to Platform or Configuration
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
frameworkpath	(Darwin) Specifies additional paths to search for frameworks	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
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	N
priority_includes	Same as the includes paths, except this include list is added prior to the ones defined in the includes paths	N
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

Attribute	Description	Target to Platform or Configuration
uselib	Add a dependency on a 3rd party library by name	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 be 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
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
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

Feature	Description
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 8.0.1 or later, and allows for distributed network builds for compiling larger projects.

You must have the appropriate package for your operating system:

- **Windows** or **Android** – IncrediBuild for Make and Build or Accelerated Dev Tools

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	`use_incredibuild` Instructs Waf to use incredibuild to distribute and parallelize the build, if possible. You need to specify the type of incredibuild package based on the platform.
Adjust the maximum number of parallel tasks	`incredibuild_max_cores`
Determine which IncrediBuild package is configured for your machine	`xgConsole.exe /QUERYLICENSE`

Waf requires certain packages and the Windows registry key settings below to run IncrediBuild. Run `lmbr_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
MaxConcurrentPDBs	0	Controls how many files can be processed in parallel. This optimization is also useful for MSBuild.

To enable IncrediBuild

1. Open the `user_settings.options` file located in `/_WAF_`.
2. In the `user_settings.options` file, under **[Incredibuild Options]**, do the following:
   - Set the **use_incredibuild** 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 2015 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 2015.

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 2015.

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 2015

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. 1824)
- Adding a Spec (p. 1827)
- Adding a Build Module (p. 1829)
- Launching the Game Project (p. 1837)

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 Using the Project Configurator (p. 28).

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"      : "SamplesProject",
    "product_name"      : "Samples Project",
    "executable_name"   : "SamplesProjectLauncher",
    "code_folder"       : "Code/SamplesProject",
    "modules"           : ["SamplesProject"],
    "project_id": "{D882E365-54D6-586E-BD78-2650F3057D49}",
    "sys_game_folder"   : "SamplesProject",
    "sys_dll_game"      : "SamplesProject",
```

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"
                         ]
                     }
                 }
```

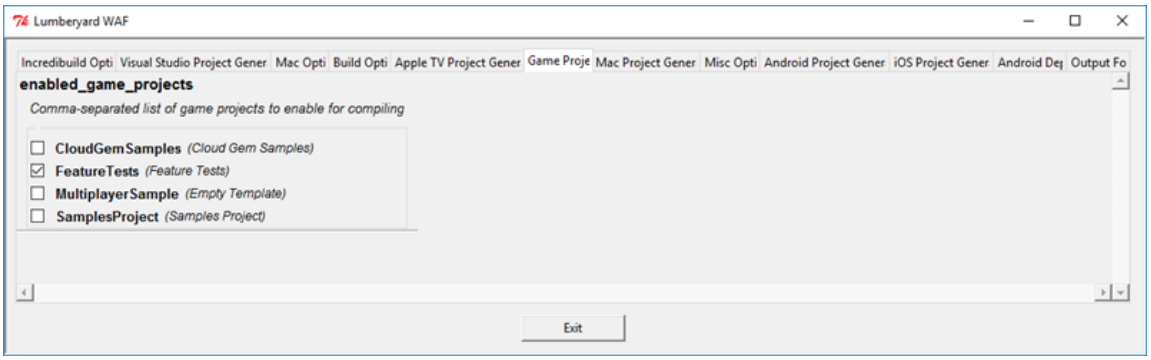
Updating the User Settings

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 (in `\dev\_WAF_\`) 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. To run the GUI, type `lmbr_waf show_option_dialog` from a command window in the `\dev\` directory. 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 build_win_x64_vs2015_debug -p game_and_engine --enabled-game-projects=MyGame
```

> **Note**
> If you use Visual Studio 2015, type `vs2015`. If you use Visual Studio 2013, type `vs2013`.

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. 1829).

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. 1827)
- Adding the Spec File to the Visual Studio Solution Generator (p. 1828)
- Building the Spec (p. 1829)

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 the Woodpecker",
    "visual_studio_name"    : "Woodpecker",
    "comment"               : "This is meant to only compile tools on Windows.",
    "disable_game_projects" : true,
    "platforms" : ["win"],
    "configurations" : ["debug","profile"],

    "modules" :
    [
        "AzCore",
        "AzFramework",
        "AzToolsFramework",
        "GridMate",
        "LuaIDE",
        "Profiler"
    ]
}
```

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. 1805).

Spec	Description
all.json	Configuration for all targets.
dedicated_server.json	Configuration to build dedicated servers for the enabled projects.
external_sdks.json	Configuration to build externally distributed binary-only libraries.
game_and_engine.json	Configuration to build the engine and game projects.
pipeline.json	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)
resource_compiler.json	Configuration to build only the Resource Compiler.
shadercachegen.json	Configuration to build only the shadercache generator.
tools.json	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: `lmbr_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: `lmbr_waf build_win_x64_vs2015_profile -p MySpec`

> **Note**
> If you use Visual Studio 2015, type `vs2015`. If you use Visual Studio 2013, type `vs2013`.

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. There `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	Modules that are dynamically loaded at runtime as part of the lumberyard engine module	Shared Library (Non-Release), Static Library Performance (Performance, Release)

Build Module	Description	Project Type
	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. For Debug and Profile configurations, these modules are built as shared libraries. For the Windows platform, versioning information is injected as defined in the waf_branch_spec.py file located in the root folder. As such, a Windows resource (`.rc`) file as needed as part of the waf_files content.	
CryEngineSharedLibrary	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 `use` keyword.	Shared Library
CryEngineStaticLibrary	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 `use` keyword.	Static Library
CryLauncher	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\Launcher` subfolder. If an additional platform is included, a new launcher project would be added in this subfolder and use the CryLauncher build module.	Executable

Build Module	Description	Project Type
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 `build_utilities` 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 `CryEditorUI_QT` plugin.	
CryPlugin	Used by Lumberyard Editor plugin projects. It is automatically placed in the `\EditorPlugins` subfolder and automatically loaded by Lumberyard Editor at runtime.	Shared Library
CryStandAlonePlugin	Used by Lumberyard Editor plugin projects. The difference between this module and CryPlugin is that it does not import any `SANDBOX` or `EDITOR_COMMON` imports, RTTI is enabled, and `nodefaultlib:/` is set to `libcmt`.	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 `use` feature of Waf.	Shared Library

Build Module	Description	Project Type
CryResourceCompiler	Used by the Resource Compiler to implicitly set the target name to `rc` and the subfolder to `\rc` under the `\configure` output folder.	Executable
CryResourceCompilerModule	Used by the Resource Compiler to implicitly set the target name to `rc` and the subfolder to `rc` under the `\configure` 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 verboseness of `wscript` files:

`<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.	

Keyword	Description	Targetable?
platforms	The list of platforms to restrict this module to. If not specified, then defaults to `all`, which assumes all supported target platforms on the current host.	No
configurations	The list of configurations to restrict this module to. If not specified, then defaults to `all`. In addition to the standard configurations (debug, profile, and release), 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 debug and profile for the iOS platform, then the configuration list would include the values `ios:debug` and `ios:profile`.	
file_list	The `.waf_files` 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
stlibpath	Lib path for static libs (generally the same for any lib).	Yes

Keyword	Description	Targetable?
linkflags	Additional link flags during the linker phase.	Yes
export_definitions	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')]
```

```
    )
)
```

Launching the Game Project

Building your game project produces an `.exe` file in Lumberyard's file system.

To launch your project, open the executable file. The directory and file name are named according to the following conventions:

`/dev/Bin64<compiler>.<build mode>/<executable name>.exe`

Where:

- `<compiler>`
  - `vc120` – Visual Studio 2013.
  - `vc140` – Visual Studio 2015.
- `<build mode>` (p. 1847)
  - `Debug` – Built in debug mode.
  - (None) – Built in profile mode.
  - `Performance` – Built in performance mode.
  - `Release` – Built in release mode.
- `<executable name>` – The `executable_name` specified in your `project.json` (p. 1800) file. The file name typically mirrors the project name.

For example, if your project was configured with the following:

- Compiler: Visual Studio 2015
- Build mode: debug
- Executable name: `SampleProjectLauncher`

Then your directory and file name would be the following:

- `/dev/Bin64vc140.Debug/SampleProjectLauncher.exe`

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
- **Hinter** (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: `lmbr_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.

To enable Qt for a particular module, you must define it as a feature. Each module's configuration is in a `wscript` file that is in each module's directory. For example, the `EditorUI_QT` module has a `wscript` file located at `dev\Code\Sandbox\Plugins\EditorUI_QT\`. To enable Qt, edit this file and add `['qt5']` to the `features` line, as shown in the following example.

```
bld.CryEditorUiQt(
        # Settings
        target              = 'EditorUI_QT',
        vs_filter           = 'Sandbox/Plugins',
        file_list           = 'editorui_qt.waf_files',
        platforms           = ['win'],
        configurations      = ['debug', 'debug_test', 'profile', 'profile_test'],
        features            = ['qt5'],
```

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

...
..

#incude <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.

The following is an example of the contents in an `ObjectPanel.qrc` file:

```
<RCC>
    <qresource prefix="/Panels/ObjectPanel">
        <file alias="icon_layers.png">res/icon_layers.png</file>
    </qresource>
</RCC>
```

For more information, see The Qt Resource System documentation.

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
  - 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:

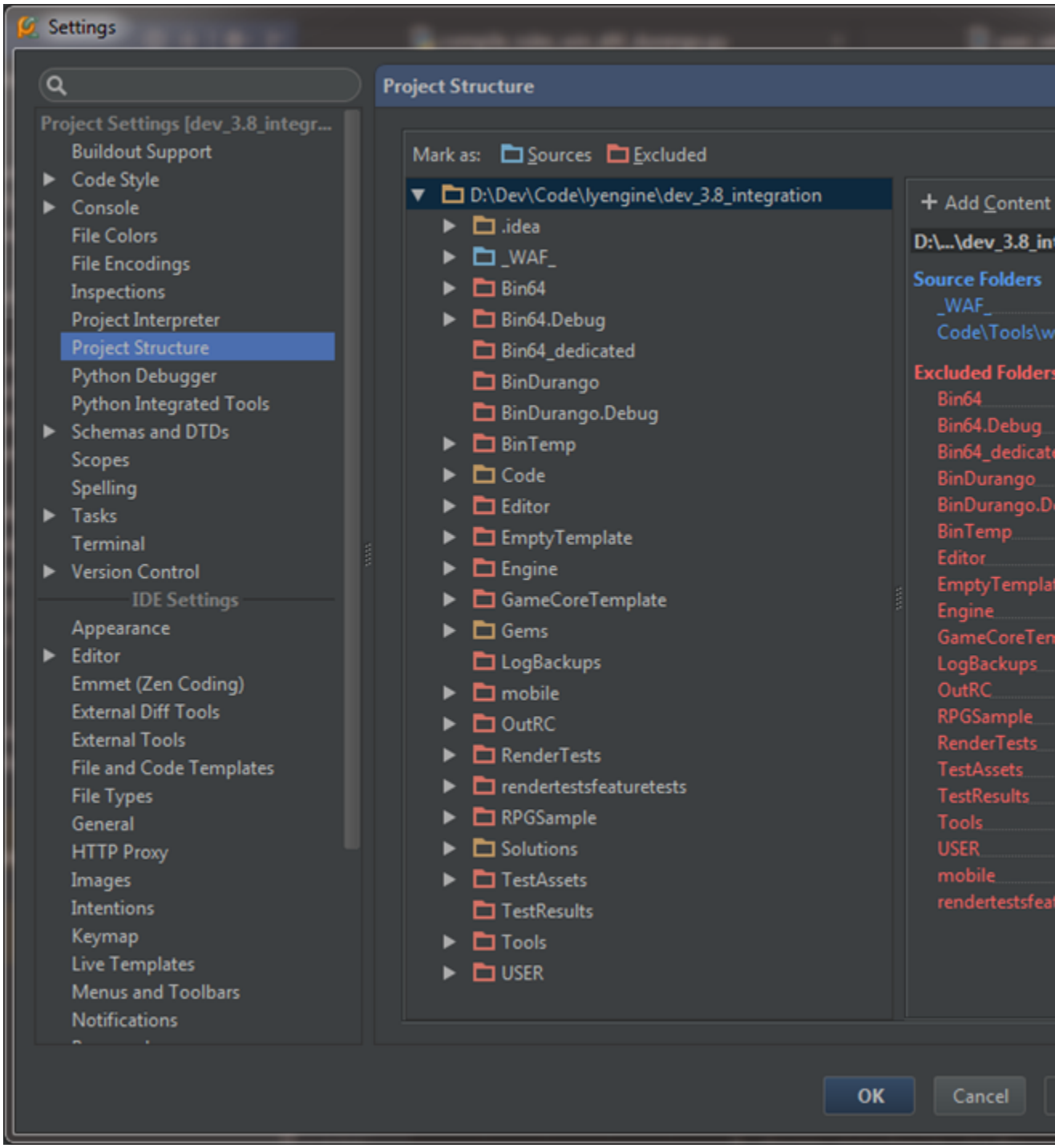
- **use_uber_files** = `True`
- **max_parallel_link** = `4`
- **use_incredibuild** = `True`
- **incredibuild_max_cores** = `128`

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 must debug either in `\dev\Tools\Build\waf-#.#.## \waflib` or `\dev\Tools\Build\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 that the _WAF_ project is outdated but your project is already up-to-date, click **No** to build.

Troubleshooting

When using multiple jobs (for example, `--jobs=12`), Waf can be difficult to debug. Try using `--jobs=1` to disable multi-threading.

When using IncrediBuild, the debugger won't properly execute all break points. Try disabling IncrediBuild when debugging Waf.

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

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. 20).
2. On the **Summary** page, click **Configure project** to create your game project using Project Configurator. For more information, see Using the Project Configurator (p. 28).
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, type one of the following:
     - If you are using Visual Studio 2015: `lmbr_waf build_win_x64_vs2015_profile -p game`
     - If you are using Visual Studio 2013: `lmbr_waf build_win_x64_vs2013_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. 1851).

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 one of the following:
   - If you are using Visual Studio 2015: **lmbr_waf build_win_x64_vs2015_profile -p all**
   - If you are using Visual Studio 2013: **lmbr_waf build_win_x64_vs2013_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 one of the following:
   - If you are using Visual Studio 2015: **lmbr_waf build_win_x64_vs2015_release -p game_and_engine**
   - If you are using Visual Studio 2013: **lmbr_waf build_win_x64_vs2013_release -p game_and_engine**

Note
If you use Visual Studio 2015, type `vs2015`. If you use Visual Studio 2013, type `vs2013`.

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_<Platform>.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.

     Note
     If the status of the remote shader compiler does not change from **Ready to Build**, the `r_ShadersRemoteCompiler` setting in the `dev\system_windows_pc.cfg` file is likely still set to the default of `0`. To enable the remote shader compiler, type `r_ShadersRemoteCompiler = 1` in the Lumberyard Editor console and then relaunch the remote shader compiler.
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_<Platform>.txt`.
   - If you use the `ShaderList_<Platform>.txt` file from the shader compiler server, you must specify the path as the first parameter in the batch file. For example, `BuildShaderPak_DX11 f:\shader_compiler_server\ShaderList_<Platform>.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_packs` 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 or batch 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_version`*`\samplesproject_pc_paks\Bin64.release\SamplesProjectLauncher.exe`
- **Command Arguments** to `+map` *`(MAPNAME)`*
- **Working Directory** to *`lumberyard_version`*`\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. 1848) 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 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. 1355).
- **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. 173).

- **BuildShaderPak_DX11.bat** – Batch file used to generate the `ShaderCache.pak` files. For more information, see ShaderCache.pak File Generation (p. 173).

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.

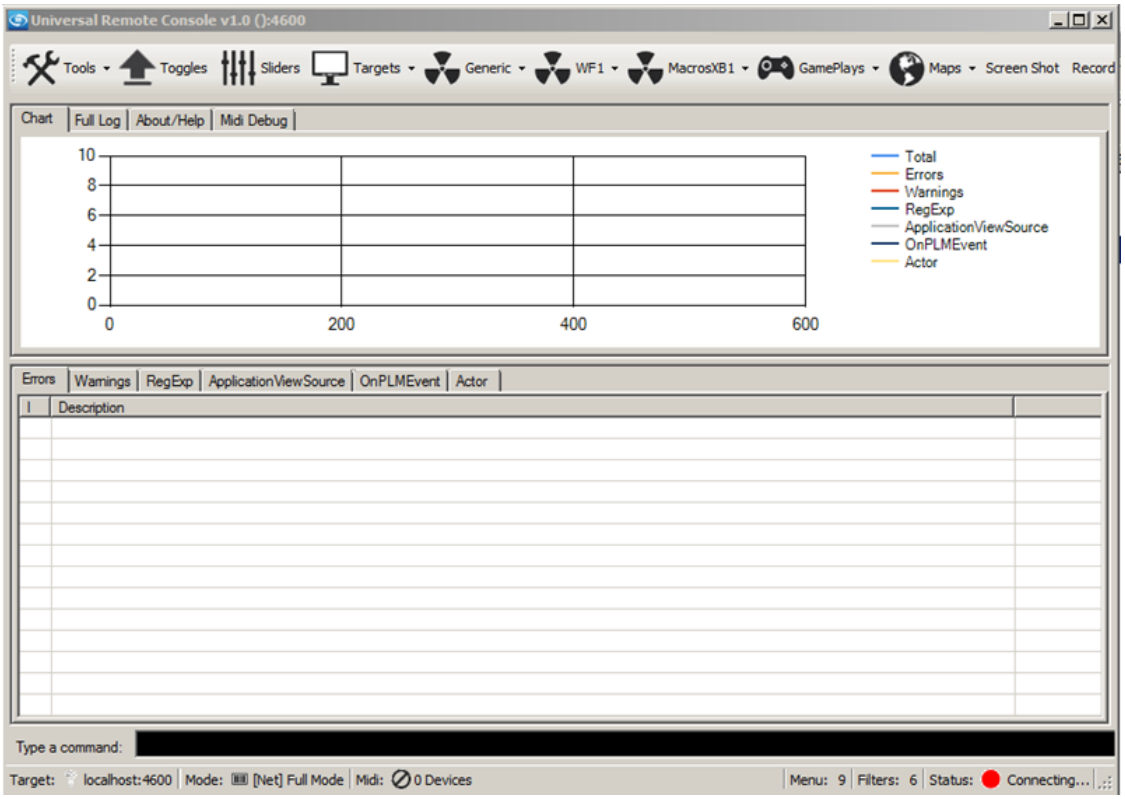
Universal Remote Console

You can use **Console** commands to modify and configure the Lumberyard runtime 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_version\dev\Tools\RemoteConsole\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. 1856) 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. 1854) service for deploying, operating, and scaling session-based multiplayer game servers in the cloud.
archetype entity	A special type of entity (p. 1856) 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 *prebaked*.
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 *morph target animation* or *per-vertex animation*.

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 *bspace*.
bloom	Effect that reproduces an imaging artifact of real-world cameras. The effect appears as fringes or feathers of illumination bleeding from the border of a bright area in an image, which gives the illusion of a bright light overwhelming the camera.
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. 1856) 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 modular message dispatch system that enables components, entities, and other types of objects to communicate with one another with few or no interdependencies. Because Ebuses are decoupled from each other, you can more easily build high performing and modular game systems. Also known as *event bus*.

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.
legacy	A designation for Lumberyard tools that are no longer being advanced and will eventually be removed.
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. 1856) 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.
Preview	A designation for Lumberyard tools that may be missing key features but are still stable and usable. The user experience is high quality, functional, and consistent where it exists but may be unfinished. APIs are subject to change.
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. Similar to the capability of prefabs, slices are reusable, easily updatable component entity templates.
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. 1857), 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.
z-fighting	Phenomenon in 3D rendering that occurs when two or more primitives have similar or identical values in the z-buffer. Also called stitching.

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

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\CloudGemSamples`
- `\dev\Code\CloudGemSamples`
- `\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\Editor`
- `\dev\Engine`
- `\dev\FeatureTests`
- `\dev\Gems`
- `\dev\MultiplayerProject`
- `\dev\ProjectTemplates`
- `\dev\SamplesProject`
- `\dev\Tools\Build\waf-1.7.13`
- `\dev\Tools\lmbr_aws\AWSResourceManager\default-project-content`
- `\dev\AssetProcessorPlatformConfig.ini`
- `\dev\bootstrap.cfg`
- `\dev\editor.cfg`
- `\dev\engineroot.txt`
- `\dev\lmbr_aws.cmd`
- `\dev\lmbr_waf.bat`
- `\dev\lmbr_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

Starter Game

- 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.