Amazon Lumberyard

Welcome Guide

Version 1.24

Amazon Lumberyard: Welcome Guide

Table of Contents

Welcome to Amazon Lumberyard

Amazon Lumberyard is a *free* high-performance *3D engine* with a professional suite of tools, editors, and libraries that enable you to create captivating real-time graphics, immersive experiences, awe-inspiring virtual worlds, and dynamic visualizations. Lumberyard brings the capabilities of Amazon Web Services and the community of Twitch to connect your ideas to players, customers, and viewers around the world.

What does "free" mean?

Lumberyard is free. There are no seat licenses, no site licenses, no subscriptions, no support contracts, no revenue share. Just free. If you choose to use AWS services in your project, you'll be responsible for fees associated with those services. Amazon Lumberyard is always free to use and you can customize the tools and source code to suit your needs.

Amazon Lumberyard's creative capabilities without compromise

A world-class engine

At its core, Lumberyard is a high-performance real-time 3D engine that produces incredible visual fidelity. Lumberyard includes all the capabilities that professional designers, artists, and developers expect, with familiar user experience patterns that enable fast adoption and development. Its robust toolset enables rapid iteration to the highest quality. Built on a modular architecture and extensible through the *Gems* framework, Lumberyard makes it easy to add new features, APIs, and assets.

Deeply integrated with Amazon Web Services

Lumberyard's online capabilities, backed by Amazon Web Services, offer staggering possibilities to developers. Common connected elements like dedicated servers, dynamic content, online economies,

and real-time stats are easily implemented and scaled, so you can focus on the creative aspects of your project. With Lumberyard's crossplay capabilities and Twitch integration, you can find new and novel ways to engage people socially. The limitless potential of AWS can deliver experiences that reach far beyond the capabilities of any single device.

Whether you're a student, hobbyist, independent developer, or major studio, Lumberyard provides the same growing toolbox to realize your creative vision.

Interested? Read on to learn what's in the box.

Welcome Guide contents

Lumberyard features

Lumberyard is packed with features that enable you to create beautiful worlds, engaging characters, and dynamic simulations. With Lumberyard's deep integration to Amazon Web Services and Twitch, you can add cloud-connected features and connect players and customers around the world.

Here are just some of Lumberyard's features:

Engage fans with Twitch

Add new levels of interactivity between Twitch streamers and viewers with Twitch ChatPlay. Give Twitch streamers dynamic real-time broadcast customization options with Twitch Metastream. Enable direct engagement between Twitch streamers and viewers with Twitch JoinIn.

Build robust multiplayer features, fast

Easily deploy, operate, and scale multiplayer game servers with Amazon GameLift. Use the Cloud Gems Framework to add dynamic content, leaderboards, and daily messages.

Create incredible worlds

Lumberyard Editor is a powerful, customizable world-building tool. Sculpt and paint expansive, multi-layered terrain. Decorate your landscape with dense, lush vegetation. Design panoramic skies with dynamic time-of-day and atmospheric effects. Create interactive infinite oceans. Populate your world with life-like entities using simple but powerful components.

Tell stories with emotive characters

Create believable, immersive characters with *EMotion FX*'s data-driven tools. Lumberyard's Animation Editor provides a visual, node-based interface for bringing characters to life with blend trees and blend spaces, visual state machines, and linear skinning. Tell engaging stories through cinematics with Lumberyard's cinematic sequence editor, Track View.

Spectacular visual effects with NVIDIA PhysX and particles

NVIDIA PhysX and NVIDIA Cloth enable you to build dynamic, real-time physical interactions with static colliders, rigid bodies, and realistic cloth simulations. With Lumberyard's Particle Editor, you can create stunning visual effects.

Develop engaging experiences with ease

You don't need to be an experienced programmer to use the power of Lumberyard. Design captivating systems and experiences with Script Canvas, a node-based visual script editor that enables easy access to Lumberyard's underlying functionality. Lumberyard also supports Lua for designers who prefer the rapid iteration that scripting languages provide.

Leverage the power of Amazon Web Services

With Lumberyard, you can incorporate AWS services such as Amazon DynamoDB, AWS Lambda, Amazon S3, Amazon Cognito, Amazon SNS, and Amazon SQS with Cloud Canvas tools and solutions. The Cloud Gem Framework Gem provides C++ classes that can execute any AWS SDK for C++ call, bringing the vast compute and storage of the cloud to your projects.

Supported platforms

You can use Lumberyard to develop games for the following platforms:

- Microsoft Windows PC
- Android
- iOS
- macOS
- PlayStation
- Xbox
- Linux (dedicated server only)

Some platforms have additional requirements.

- For console support, see the Xbox for game developers and PlayStation Partners portals.
- For mobile devices, see Creating Android and iOS projects in the User guide.
- For macOS, see Creating macOS projects in the User guide.
- For Linux dedicated servers, see Creating Lumberyard Executables for Linux in the User guide.

How Amazon Lumberyard works

Amazon Lumberyard provides a complete, end-to-end environment for developing and delivering games and simulations, and supports a wide variety of platforms, including consoles, mobile devices, virtual reality, and Microsoft Windows PCs. Since it's such a large development environment with so many different features and tools, it can be intimidating to new users, especially for those who don't have a traditional developer background. This topic covers the various parts of Lumberyard at a high level and the common ways you can work with it depending on your role or task as a new or r experienced game developer or designer.

Lumberyard has several tools, editors, and systems that work together to help you assemble a game. Central to this is the "Lumberyard game engine", which provides the following:

- Graphics rendering and output
- Game logic execution
- Messaging across game systems and components
- Memory and resource abstraction and management
- Multi-threading support for input, audio, physics simulation, user interfaces (UI), artificial intelligence (AI), networking and multiplayer, and other common game features
- Asset management and packaging

The engine itself is really just a collection of components and modules, called *Gems*. When developing, you use the Lumberyard Editor and tools to assemble the engine for your game, choosing and adding these gems and modules as you develop it. You also use the Lumberyard tools to add your assets—including textures, meshes, sounds, music, and scripts—to construct your game's unique experience and gameplay.

Think of Lumberyard as a collection of discrete elements: code, scripts, various GUI-based editors, and command line tools. When you compile a game project, Lumberyard's build scripts pull in all the pieces specified in your project's configuration to build your game. The parts of the engine that go into your game are only those you've configured your project to use, and there's very little functionality included in the final compiled code that you didn't ask to have in it. Likewise, the asset bundling and management tools make sure you only ship with the assets you actually use in your game.

You can build a game in Lumberyard just using the Lumberyard Editor, but you will be constrained to the Gems and tools provided (along with the assets and scripts you create). If your ambitions are greater—if you want to evolve Lumberyard to support features and systems we haven't provided in the box—read on.

Overview of the Lumberyard framework

The Lumberyard installation provides a sandbox of different bits to combine, including over 100 gems and modules for you to use in your game. As a high-level concept, think of Lumberyard as a framework—a conceptual structure from which you can add new features or remove anything you don't plan to use in your game. When you need to change or extend the behavior of something, you don't need to go to some massively over-architected set of code files and hack or refactor a feature in to your game's code. Rather, you just work with only the gem or module that contains the functionality you want to change.

By focusing on modularity, you can safely experiment with different feature changes without risking progress on the entire game with an unintended regression.

Central to Lumberyard is the **AzFramework** library, which relates all of the systems, modules, and Gems into the infrastructure for your game. The Event Bus (EBus) system provides request and notification messaging across the DLLs for these systems, modules, and Gems. As a developer, you write C++ code to implement methods defined in C++ API headers that define the functionality you need.

The default Lumberyard installation is to `{drive-letter}:\Amazon\Lumberyard\{lumberyard-version-number}\` on your PC. There are two folders under it: `\dev` and `\3rdParty`. We'll examine the most important contents of \dev, which is where AzFramework and all of the various Gem and module interfaces are defined, as well as the EBus interfaces. It also contains scripts, samples, and Editor assets.

- `\dev\Bin64vc142` (or, in versions prior to 1.24, `\dev\devBin64vc141`) contains the asset builder libraries, the Lumberyard editor modules (plugins) and libraries (under `\EditorPlugins`), among other things. Build and configuration logs are under `\Logs`.
- `\dev\Code` contains the C++ API headers that you'll include in any code you create to extend the core functionality of Lumberyard. They are organized under folders for each of the systems or features that you want to extend. Most of them are virtual interfaces that you'll implement so you connect that functionality to the relevant Lumberyard system or feature through EBus, or contain the expected type and template definitions to use. These include:
  - `\Code\CryEngine` contains API headers and C++ code files for CryEngine systems and features. If you need to tweak the behaviors of CryEngine, including the 3D renderer, the legacy physics engine, the GridMate networking engine, or other components of the CryEngine, start with the source files in this directory.
  - `\Code\Framework` contains the AzFramework APIs. AzFramework is the core Lumberyard framework that defines all the systems, including Gems, systems engines, and EBus.
    - `\Code\Framework\AzCore\AzCore`: Contains all of the APIs for interoperating with Lumberyard's systems, modules, and Gems. If you are extending Lumberyard's default functionality, this is where you'll find all the APIs you need.
    - `\Code\Framework\GridMate`: Contains the GridMate networking APIs.
- `\Editor` contains all the assets, scripts, and configuration files for the Lumberyard Editor as well as some of the various editors it manages, such as the Materials Editor, the UI Canvas Editor, and the Particle Editor.
- `\Engine` contains all the default asset binaries, script files (`.lua`), entity configuration files (`.ent`), material definition files (`.mtl`), shader extensions (`.ext`), and engine configuration files for Lumberyard.
- `\Gems` contains all the source and build files for the Gems that ship with Lumberyard. When you create a new Gem, you'll add the code and build sources here, and then enable the Gem through the Project Configurator.
- `\ProjectTemplates` contains the game project templates you can choose from when initially configuring Lumberyard for your game's development.
- `\SamplesProject` and `\StarterGame` contain samples and a complete game level for you to review.
- `\Tools` contains a number of tools and SDKs you'll use throughout the development of your game, including:
  - `\Tools\AWSNativeSDK`: Contains scripts for obtaining various platform-specific SDKs for working with Amazon's cloud services in their native languages.
  - `\Tools\AWSPythonSDK`: Contains Python SDKs for working with Amazon's cloud services.
  - `\Tools\AzCodeGenerator`: Contains the binaries and configuration files for AZ Code Generator, which is a command-line utility that generates source code (or any data or text) from specially tagged source code. For more details, see Automating Boilerplate with Az Code Generator.
  - `\Tools\build`: Contains scripts and tools for building your game project with Waf including the `lmbr_waf` command-line build utility.

- `\Tools\CrySCompileServer`: Contains the executable and libraries for the CryEngine shader compiler service.
- `\Tools\LuaRemoteDebugger`: Contains the executable and libraries for the Lua script remote debugging tool.
- `\Tools\Python`: Contains all basic Python libraries and the interpreter. Lumberyard supports a minimum version of 3.7.5.
- `\Tools\Redistributables`: Contains common packages and files that you can ship with your game, including DirectX and Visual Studio DLLs.
- `\Tools\RemoteConsole`: Contains the Lumberyard remote console application.

Lumberyard also provides two command-line tools, `Lmbr.exe` and `Lmbr_waf.exe`, which you use to manage your game project overall and configure the details of your game project during development.

- `Lmbr.exe`—Provides a set of commands for managing and tracking Gems, capabilities, and 3rd party tools and packages.
- `Lmbe_waf.exe`—Provides a set of commands for automating the building and packaging of game projects with the Waf build automation framework.

All of these parts—plus some not listed here— define Lumberyard, and can be used to construct your game. As you incorporate Lumberyard systems and develop your own, you will want to communicate across them. For that, we have EBus.

Working with Gems

Many of Lumberyard's capabilities are implemented through *Gems.* A Gem is a packaged extension that can be enabled in a project to add functionality or features. Gems might contain module code, assets, scripts, supporting files and references to other Gems. Using Gems enables you to choose the features you need for your project and exclude those that aren't necessary. By keeping everything modular and only using what you need, the Gem system lets you iterate faster. Asset collections, code samples, components, libraries, tools, and even entire game projects can be distributed as Gems.

You enable Gems when you create your project in Project Configurator. You can return to Project Configurator at any time to enable additional Gems and disable unused Gems. When you create your first project, be sure to click the **Enable Gems** option in Project Configurator to see the list of Gems available. Some Gems are core systems and required for all Lumberyard projects. Other Gems are extensions for existing Gems and require their dependencies be enabled for your project. Gems you enable are detected and built automatically when you build your project.

You can create your own Gems and easily reuse and distribute your own code and assets. To get a better idea what goes into creating a Gem, have a look at the Gems directory of your Lumberyard installation and examine the included Gems. The process for creating your own Gem is through Project Configurator, and is very similar to creating a project.

Messaging between systems with EBus

All of Lumberyard's Gems and systems, as well as the components in your projects need a way to communicate with each other. Lumberyard uses a general-purpose communication system called Event Bus (EBus for short).

As discussed earlier, Gems and systems are typically implemented as DLLs. EBus is used to communicate between them—and specifically, to invoke functions in one Gem or system from another. EBus provides

both request and publish/subscribe event interfaces (buses) that allow calls across those DLLs. For example, if you've created a Gem for custom physics behaviors and you'd like to provide data to the CryEngine renderer, you'd do so by implementing an EBus interface in your Gem.

We provide the interfaces for the Gems and systems DLLs we ship as headers in the default installation. To use the functionality in these DLLs, you use the interfaces in these headers to register for a single cast (Event) or broadcast (Broadcast) event, or through supplying a data request functor to a Request Bus handler.

Likewise, to expose functionality from your own gems and provide data to another system, you must inherit the virtual interface declared in the corresponding header file and implement the handlers on that interface in your Gem's classes, and then register that handler with the EBus system. Specifically, you'll register a handler you create with EBus, which will pass a pointer to your class method to the targeted system or post a notification to the systems that are subscribed to it.

Inside of your Gem code, you also manage the connection and disconnection of your implemented handler for the EBus. EBus is just a list of handlers that calls all the functors (function pointers) registered with it..

For singleton handlers where you only need one interface to communicate across DLLs, consider using AZ::Interface and AZ:Event directly, without EBus.

There are two types of EBus:

- Request bus: This EBus type registers a handler for a method that can be called by other systems.
- Notification bus: This EBus type provides a messaging interface for notifications that systems can publish or subscribe to.

EBuses have many advantages over traditional polling methods:

- Abstraction – Minimize hard dependencies between systems.
- Event-driven programming – Eliminate polling patterns for more scalable and higher performance software.
- Cleaner application code – Safely dispatch messages without concern for what is handling them or whether they are being handled at all.
- Concurrency – Queue events from various threads for safe execution on another thread or for distributed system applications.
- Predictability – Provide support for ordering of handlers on a given bus.
- Debugging – Intercept messages for reporting, profiling, and introspection purposes.

EBuses are configurable and support many different use cases:

- As a direct global function call.
- Dispatch processing to multiple handlers.
- Queue all calls, acting like a command buffer.
- As an addressable mailbox.
- For imperative delivery.
- For queued delivery.
- Automatic marshaling of a function call into a network message or other command buffer.

For details on using EBus, read:

- Working with the Event Bus (EBus) System
- Event Buses in Depth

The Component Entity system

But what about the parts of your game that are actually in the game proper? Lumberyard has a model for that as well, called the component entity system.

Understanding the component entity system is fundamental to using Lumberyard. It's conceptually simple: Every in-game object you create for your project is an *entity*, with a unique ID and container. Each entity contains *components* that provide functionality. The functionality components provide is broad and includes primitive shapes, meshes, audio, artificial intelligence behaviors, animation, visual effects, physics, scripts, and so much more. There are even components that provide tool and debugging functionality.

As an example, suppose you want to create a door entity that can be opened and closed. You made a mesh and a couple audio files. Now, consider the functionality your door must have.

- Display the door model.
- Play back the audio files when the door opens and closes.
- Prevent passing through the door when it's closed.
- Animate the door open and close.
- Trigger the door open and close by some mechanism.

Knowing the functionality your door entity needs, you add components to the entity for each aspect of the door, including its presence in the game world and the ways a player can interact with it. A *Mesh* component visually represents the door in the game world. *Audio Trigger* components provide the audio when it opens or closes. A *PhysX Collider* component prevents a player from passing through the door when it's closed. *Script Canvas* components define the behaviors, including animation and sound playback, when the door is opened or closed. Whatever behavior you need to model, each entity is going to have a collection of components to support it. The only component common to all entities is the *Transform* component, which provides the position, orientation and scale of your entity in the game world.

Entities are easy to grasp and create, but can become complex. If the entity requires a lot of functionality, the list of components grows quickly. What if you want to add a latch with its own animation and audio to the door? What if you want to add a breakable glass pane to the door? Suddenly, the entity goes from having five components to dozens of components. This is where *slices* come in.

A slice, like an entity, is a container with a transform behavior. Instead of containing components, a slice contains one or more configured entities and might contain other slices as well. To create a more complex door, you could have the initial door entity, a second entity for the latch and its components, and a third entity for the breakable glass pane and its components. These three small entities are collected into a slice that provides a reusable, fully functional door asset. You may have heard similar concepts referred to as *prefabs* in other software. In Lumberyard, a slice is a collection of entities and/or other slices in a single reusable asset.

Behaviors applied to a slice can potentially cascade down to all of the entities it contains, and then down to the components of that entity. However, the reverse is not true, as it would make no sense for a window shattering to apply to a door latch.

Once you've internalized the slice > entity > component hierarchy, consider how you would use these concepts to develop the various elements potentially populating your game levels and world.

The Lumberyard Asset Pipeline

We've discussed what's in the Lumberyard installation, the basic framework of Lumberyard and how its various parts communicate with one another, and how you represent objects in your game. But how

do you create and manage assets for your game? After all, a door doesn't truly exist in your game if it doesn't have some form of representation.

For the purposes of Lumberyard, we'll define an asset as a resource file, saved on disk, consumed by your project in some way. An asset might be a font for your user interface, a bitmap file that contains a grassy terrain texture, a rock mesh you sculpted, animations for a character, etc. Some assets might be created in Lumberyard. Specialized files called *inputbindings* that map buttons from a game pad to input events for your project and *physicsmaterials* that describe the physical properties of surfaces, for example, are both created in Lumberyard's Asset Editor.

For many reasons, the primary being performance considerations, these different assets cannot be consumed by Lumberyard without being converted to operating system specific, *game ready* data. This process, going from a source asset file on disk to game ready data is the Asset Pipeline. The processing is performed automatically by Asset Processor.

Asset Processor is a background process (you'll see its icon in the task tray when it's running) that constantly scans directories in your project for changes in files. When changes are detected, Asset Processor uses configurable rules to determine how to handle the changes. The objective is to have 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 asset directories and can be automatically rebuilt entirely from your source assets by the Asset Processor.

The asset cache contains a full image of all files (except executables and related files) needed to run your project. Asset Processor keeps the image up to date, ensuring that new files are ready to use in the project runtime and Lumberyard Editor as soon as possible. Your project runtime will only load assets from the asset cache and never directly from your asset source directories.

Projects like modern games can have thousands of assets that need to be monitored and processed for multiple target operating systems. To manage this complexity, the Asset Pipeline is completely configurable. Here are just some of the configuration options available:

- Specify what directories should be monitored for changes.
- Specify target operating systems and tailor the Asset Pipeline's behavior per target operating system.
- Set the number of concurrent processing tasks.
- Use metadata information to associate file types and process side-by-side assets.
- Add your own asset types to the Asset Pipeline.
- Batch process assets on a build server.

When you're preparing to ship, you'll need to package the assets your project uses. Even small projects can have hundreds of assets, including multiple versions of assets, many of them not required in your final project. Manually tracking and determining which assets you need to ship can be tedious, time consuming an error prone. Asset Bundler solves this for you.

Asset Bundler makes shipping the specific assets used for the release of your game more reliable and repeatable. Reliability is based on an underlying dependency system. If you make changes to your project and add, remove, or update assets, Asset Bundler uses the dependencies to automatically determine which assets to include. Repeatability is based on underlying configuration files that provide consistency each time you run Asset Bundler.

Scripting for the Lumberyard framework

Now, you've created assets and defined them in your game using slices, entities, and components. You have an empty level or world to roam, but where's the game? That's where scripting comes in.

Lumberyard includes two scripting technologies for creating logic and behaviors: Script Canvas and Lua.

Script Canvas is a visual scripting environment. In the Script Canvas editor, you create, connect, and rearrange graphical nodes that provide a visual representation of the logic flow. Script Canvas offers an approachable and easy-to-read environment to author behaviors using the same framework as Lua and C ++. You can use Script Canvas to create scripts without needing to know how to code.

To enable Script Canvas for Lumberyard, you must enable the Script Canvas Gem.

Lua is powerful, fast, lightweight, embeddable scripting language. Lua facilitates quick iteration in your project because you can run your changes immediately without needing to recompile your source code.

Lumberyard's functionality is exposed to Script Canvas and Lua by the behavior context. The behavior context reflects runtime code and makes it accessible to scripts by providing bindings to C++ classes, methods, properties, constants, and enums. The behavior context also provides bindings for Lumberyard's EBus so you can dispatch and handle events through Script Canvas and Lua.

Functionality for both Script Canvas and Lua is added to entities through components. You can have multiple script components and mix and match between Lua and Script Canvas within your entities. This approach enables you to create small, manageable modules of logic and behavior that can be reused throughout your projects.

Further learning

As you've probably suspected, there's a lot more to Lumberyard than this, but we hope this gives you a good sense of where to start your searches for greater detail in our docs and code. Here's a few additional links to help you explore further:

For some great videos on getting started with Lumberyard, check out our Getting Started video tutorial series.

For some docs tp get you started, check out the following topics:

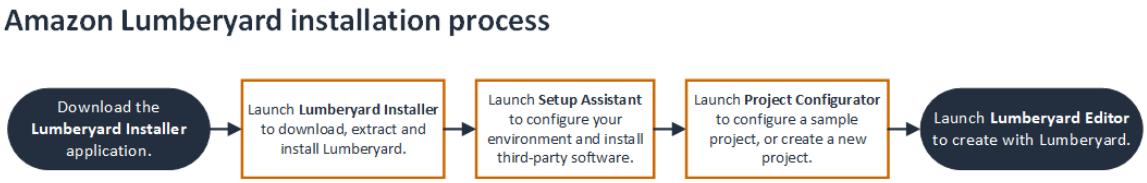
- Lumberyard Editor
- Programming Concepts
- Gems
- Component Entity System
- Component Reference
- Programmer's Guide to Entities and Components
- Emotion FX Animation Editor
- Script Canvas
- Lua Editor

Setting up Amazon Lumberyard

Amazon Lumberyard has a two-part installation process:

1. The installer application downloads, extracts, and installs Lumberyard.
2. **Setup Assistant** configures and installs additional software based on your development needs. Jump right in to the Lumberyard Editor with an **Express Install** or customize Lumberyard's features for your development needs with a **Custom Install**.

After Lumberyard is installed, use **Project Configurator** to select a project or create a new project. The following sections detail the minimum requirements for Lumberyard and guide you through the installation process.



Setup topics

Amazon Lumberyard system requirements

Lumberyard has a minimum set of system requirements for development, as outlined in the following sections. Disk space and RAM requirements are dependent on the options that you choose during installation.

System requirements

If your system is capable of running a modern real-time 3D game with good performance you should be set, however, review these detailed requirements to be certain.

Lumberyard requires Windows 10.

Lumberyard minimum hardware requirements:

- 3 GHz quad-core processor
- 8 GB RAM
- 2 GB VRAM DirectX 11 or later compatible video card
  - NVIDIA GeForce GTX 660 Ti with driver version 368.81 or later
  - AMD Radeon HD 8730M with driver version 16.15.2211 or later

- 60 GB of free disk space

> **Note**
> If you select options to build the engine, editor, or tools in **Setup Assistant**, 14 GB RAM is required for compilation.
> Some advanced graphics features require a DirectX 12 or later compatible video card.
> Required free disk space is dependent on the options that you select when installing Lumberyard.

Prerequisites

You can use the Lumberyard Editor and tools without installing additional software. To create new projects or use advanced development features in Lumberyard, you need a developer environment. One of the following versions of Microsoft Visual Studio is required:

- Microsoft Visual Studio 2019 version **16.2.4** or later.
- Microsoft Visual Studio 2017 version **15.9.14** or later.

Microsoft offers Visual Studio Community edition free to individual developers. For more information and to download and install Visual Studio Community, visit the Visual Studio Community portal.

Visual Studio 2017 and 2019 required features

The default Visual Studio installation might not include all of the features that are required by Lumberyard. Ensure that the following Visual Studio features are enabled:

1. Launch the **Visual Studio Installer** from your download directory or the **Start Menu** if you've already installed Visual Studio.
2. If you've installed Visual Studio, choose **More - Modify** on the version of Visual Studio you'll use with Lumberyard.
3. On the **Workloads** tab:
   - Select **Game development with C++**.
     - In the **Installation details** panel on the right, select at least one **Windows 10 SDK**.
4. On the **Individual components** tab, under **Compilers, build tools, and runtime**, select the **VC++ toolset** that corresponds to the installed version of Visual Studio:
   - **Visual Studio 2017**: Select at least one version of the **VC++ 2017 toolset**.
   - **Visual Studio 2019**: Select at least one version of the **MSVC v142 - VS 2019 C++ x64/x86 build tool**.
     - (Optional) To build with the Visual Studio 2017 toolset, select **MSVC v141 - VS 2017 C++ x64/x86 build tools**.

Visual C++ Redistributable Packages

Lumberyard requires Visual C++ Redistributable packages. 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** directories in the *lumberyard_version*\dev\Tools\Redistributables\ directory.
- Download and run the installers directly from Microsoft.
  - Visual C++ Redistributable for Visual Studio 2012
  - Visual C++ Redistributable for Visual Studio 2019

Note
The Visual C++ Redistributable for Visual Studio 2019 also contains redistributables for Visual Studio 2015 and 2017.

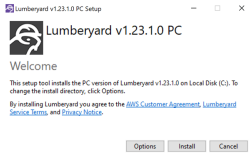
Installing Amazon Lumberyard

Use the download link below to get the latest **Lumberyard Installer** application.

Download Now

Run the Lumberyard installer

Navigate to your `Downloads` directory and run `LumberyardInstaller.exe` to download, extract, and install Lumberyard.



The default Lumberyard installation path is `C:\Amazon\Lumberyard\`. To set a different installation path, choose the **Options** button. Choose the **Install** button to begin installation. The process can take some time, depending on your internet connection speed.

The installer displays an **Installation Successfully Completed** message. Click **Launch Lumberyard Setup Assistant** to continue with setup.

Lumberyard's executables

`LumberyardInstaller.exe` creates shortcuts on the desktop and in the Start Menu for three applications:

Setup Assistant (p. 15)

Setup Assistant configures Lumberyard's environment according to your development needs, and downloads and installs additional software and SDKS. You can use Setup Assistant at any time to add development features to your Lumberyard environment.

SetupAssistant.exe is located in *`lumberyard_version`*`\dev\Tools\LmbrSetup\Win`.

Project Configurator (p. 18)

With Project Configurator, you create, configure, set, and build projects. When you run Project Configurator for the first time, you see several sample projects that are available to help you learn Lumberyard's features.

ProjectConfigurator.exe is located in *`lumberyard_version`*`\dev\Bin64`*`vc141_or_vc142`*.

Lumberyard Editor (p. 28)

Lumberyard Editor is Lumberyard's core application. In Lumberyard Editor, you create levels, assets, and interactions for your projects.

Editor.exe is located in *`lumberyard_version`*`\dev\Bin64`*`vc141_or_vc142`*.

Lumberyard's directory structure

The default Lumberyard installation location is `C:\Amazon\Lumberyard\lumberyard_version\`. The root directory contains the following directories and files:

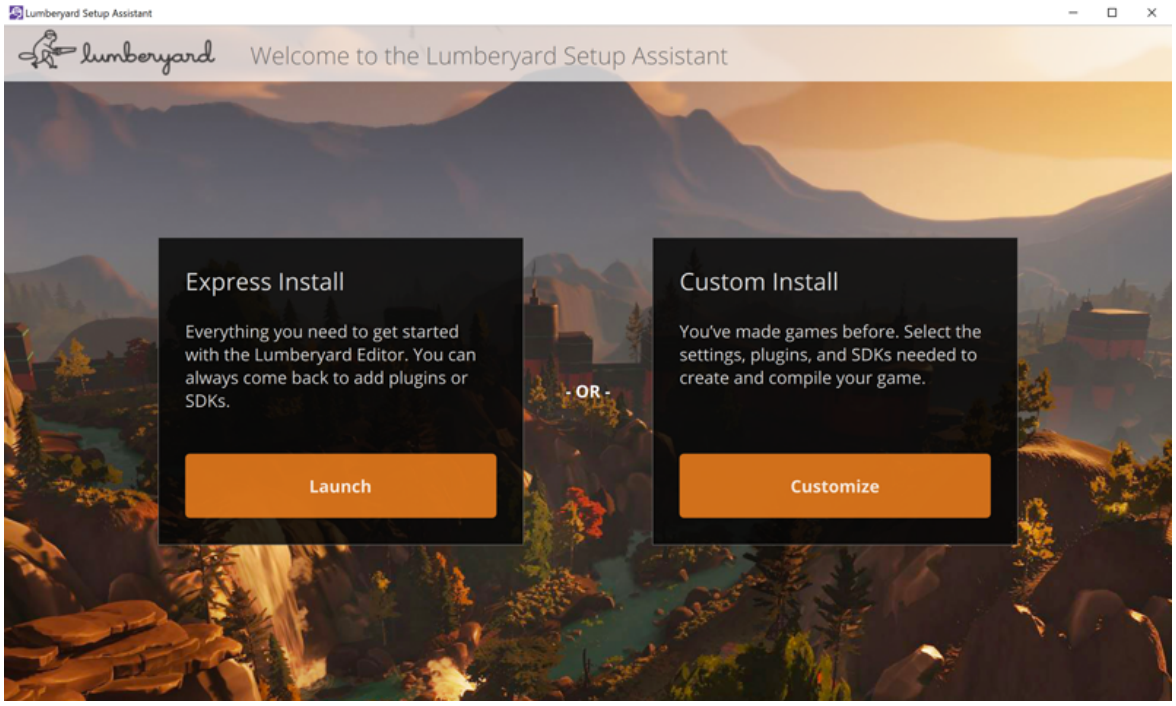
- `dev`
  - `_WAF_`: Waf build system files.
  - `Bin64`: Binaries and configuration files for the Resource Compiler.
  - `Bin64vc141`: Binaries and configuration files for Visual Studio 2017.
  - `Bin64vc142`: Binaries and configuration files for Visual Studio 2019.
  - `Code`: Source files and solution files for the Lumberyard engine and tools.
  - `Editor`: Editor assets.
  - `Engine`: Engine assets.
  - `Gems`: Modular components and assets.
  - `MultiplayerSample`: Multiplayer sample project that demonstrates how to build a multiplayer game with the component entity system.
  - `ProjectTemplates`: Configuration files, libraries, and scripts for the empty template.
  - `SamplesProject`: Sample project.
  - `StarterGame`: A full example game with 3D environments, event scripting, and basic enemy AI.
  - `Tools`: Third-party tools and plugins.
  - `engineroot.txt`: System file required by Lumberyard Setup Assistant to verify the directory.
- `3rdParty`
  - Third-party software required to use or compile Lumberyard.
  - `3rdParty.txt`: System file used by other third-party tools to verify the directory.

Configuring your Amazon Lumberyard environment with Setup Assistant

Lumberyard Setup Assistant configures and maintains your Lumberyard environment based on your development needs. If you're continuing from Installing Amazon Lumberyard (p. 14), Setup Assistant should be running on your desktop. Setup Assistant can also be launched from the Start Menu or desktop shortcuts.

Lumberyard Setup Assistant performs several important functions:

- Ensures that you have the required runtime software.
- Ensures that you have the required SDKs.
- Provides plugins for detected content applications such as Photoshop and Maya.
- Validates registry settings, paths, and libraries.



On first run, Lumberyard Setup Assistant presents two options, **Express Install** and **Custom Install**. If you're a content creator, such as a designer or artist, and won't be compiling code, or if you want to jump right in and use the Lumberyard Editor and tools, select **Express Install**. You can always revisit Setup Assistant to add development features. If you'd like to set up Lumberyard for code development, select **Custom Install**.

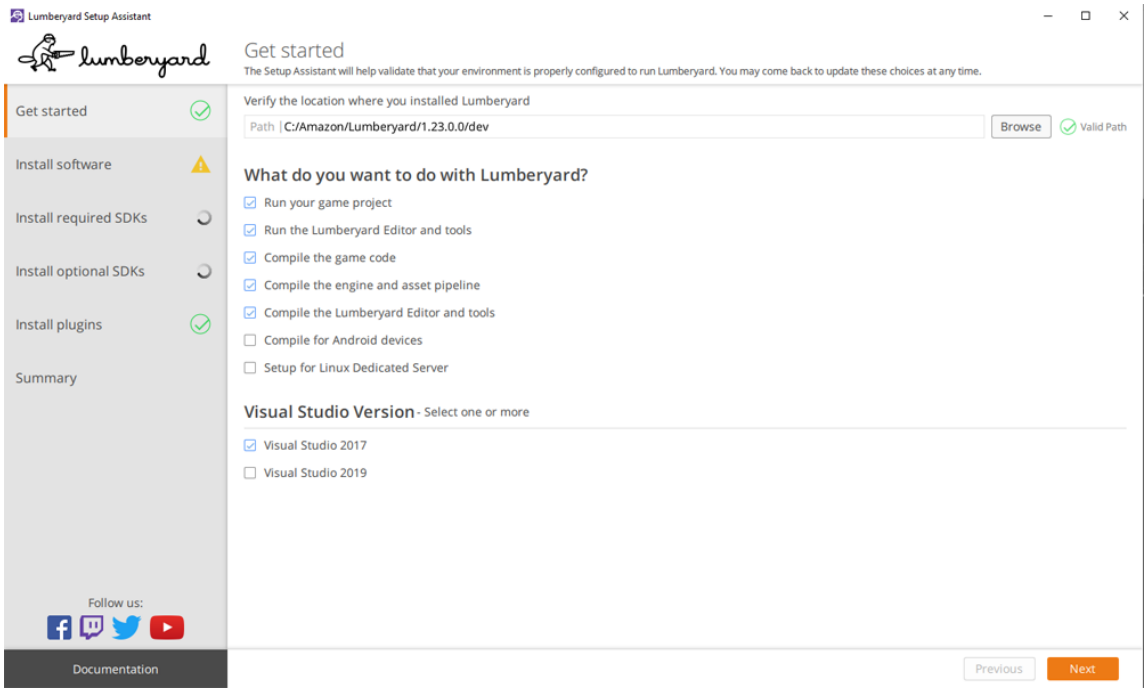
You should run Setup Assistant periodically, especially after you make changes to your environment, to validate and repair settings and paths. You can also customize Setup Assistant with a configuration file to easily integrate your specific directory structure.

> **Note**
> Some Setup Assistant options require the installation of third-party software and licenses, so make sure that you consult the terms of service before installing third-party software.

Lumberyard Setup Assistant custom installation

With **Custom Install**, you specify how you would like to use Lumberyard. Setup Assistant downloads third-party software and validates that the environment is properly configured based on your choices.

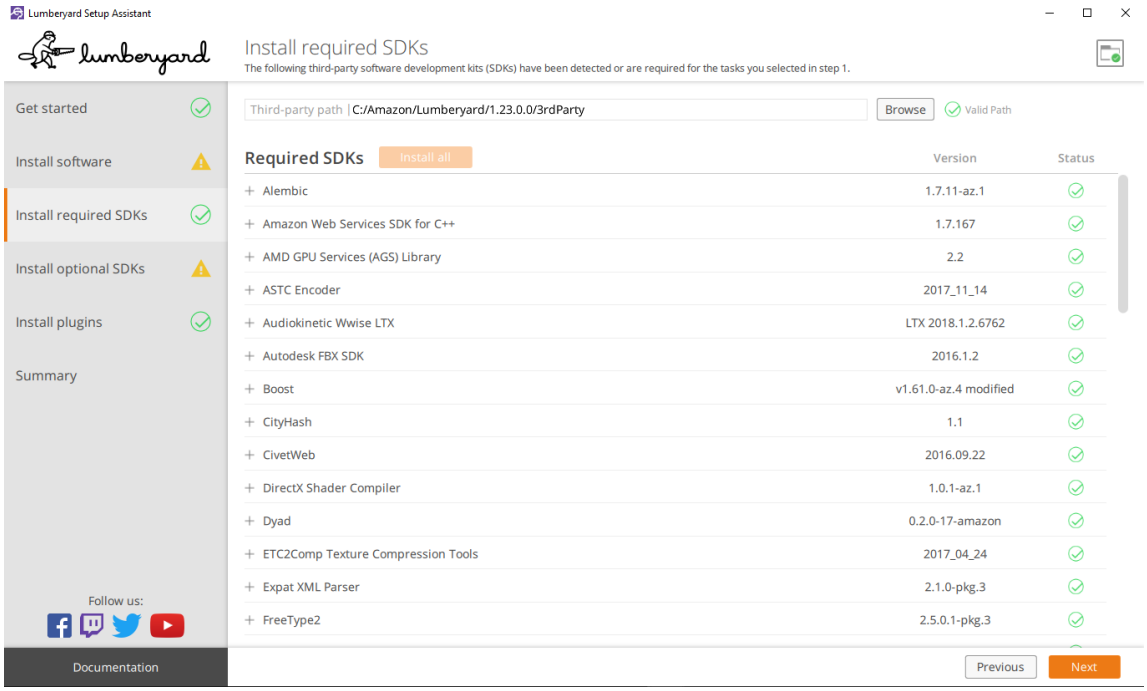
1. Custom installation begins with specifying how you intend to use Lumberyard.



2. Verify the **Path** is correct for your Lumberyard installation.
3. Select options based on your development needs:
   - **Run your game project**
   - **Run the Lumberyard Editor and tools**
   - **Compile the game code*** - Select this option if you intend to create new projects with Lumberyard.
   - **Compile the engine and asset pipeline*** - Select this option if you intend to make changes to the engine or asset pipeline.
   - **Compile the Lumberyard Editor and tools*** - Select this option if you intend to make changes to Lumberyard Editor or other tools.
   - **Compile for Android devices***
   - **Setup for Linux Dedicated Server***

     Note
     * If you select any of these options, you might need to perform additional tasks, such as installing Microsoft Visual Studio and installing additional required SDKs. These tasks display on the **Install software** and **Install required SDKs** pages.
4. Select **Visual Studio 2017**, **Visual Studio 2019**, or both.

   Note
   The version(s) of Visual Studio selected here will be enabled as build platforms, and Visual Studio Solutions will be generated.
5. Once you are satisfied with your selections, choose **Next** to install required software and SDKs.
6. The **Install software** and **Install required SDKs** pages display a red icon if a requirement cannot be found and a green checkmark for installed requirements. Missing optional software and SDKs display a yellow icon. Follow the instructions on each page to install the required software and SDKS.



Note
Ensure the **Third-party path** on the **Intall required SDKs** page is correct for your Lumberyard installation.

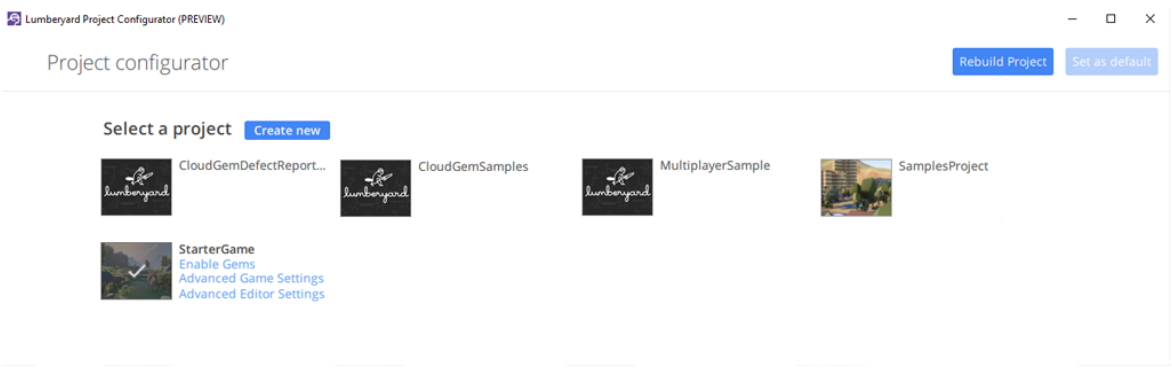
7. Install desired *optional* software and SDKs on the **Install software**, **Install optional SDKs**, and **Install plugins** pages by following the instructions on each page.
8. When you have completed installing software and SDKs, the **Summary** page displays information about your Lumberyard environment. From the **Summary** page, you can launch the Lumberyard Editor by choosing **Launch Editor**. If you'd like to choose an existing project or create a new project, choose **Configure project** to launch Project Configurator.

Manage Lumberyard projects with Project Configurator

Project Configurator configures and manages your Lumberyard projects. If you're continuing from Configuring your Amazon Lumberyard environment with Setup Assistant (p. 15), Project Configurator should be running on your desktop. Project Configurator can also be launched from the Start Menu or desktop shortcuts.

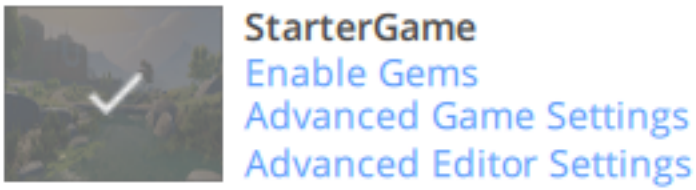
With Project Configurator, you can:

- Set a default project.
- Create new projects.
- Add code or asset features to your project by enabling Gems.
- Create new Gems that can be added to projects.
- Build projects.
- Set advanced game and editor settings per project.

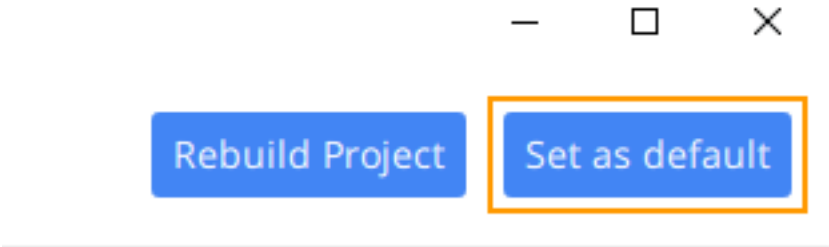


Select a project

Lumberyard has several example projects that you can work with to learn Lumberyard's features. To select a project, choose the project's icon. The selected project is highlighted with a check mark.



To load a project in Lumberyard Editor, the project must be set to default. While the project is selected, choose the **Set as default** button in the upper right-hand corner of the Project Configurator window. If the **Set as default** button is inactive, the selected project is already the default project.

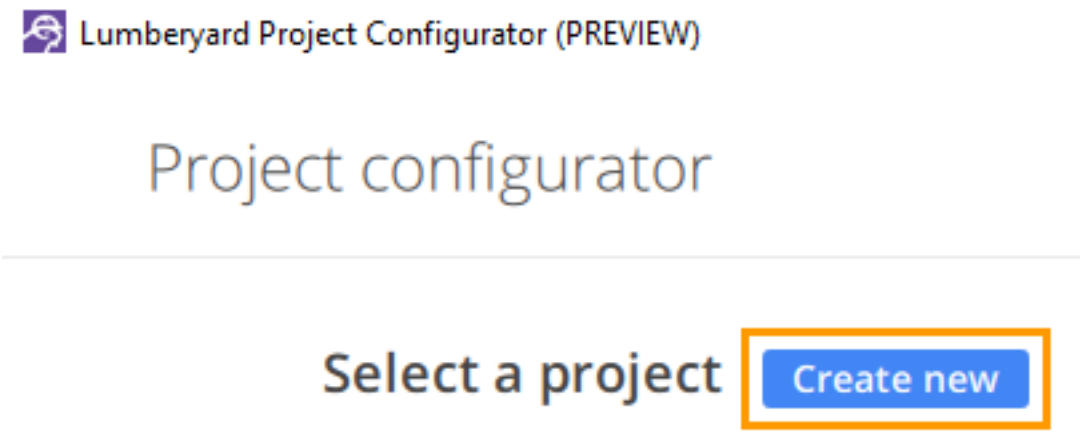


Create a new project

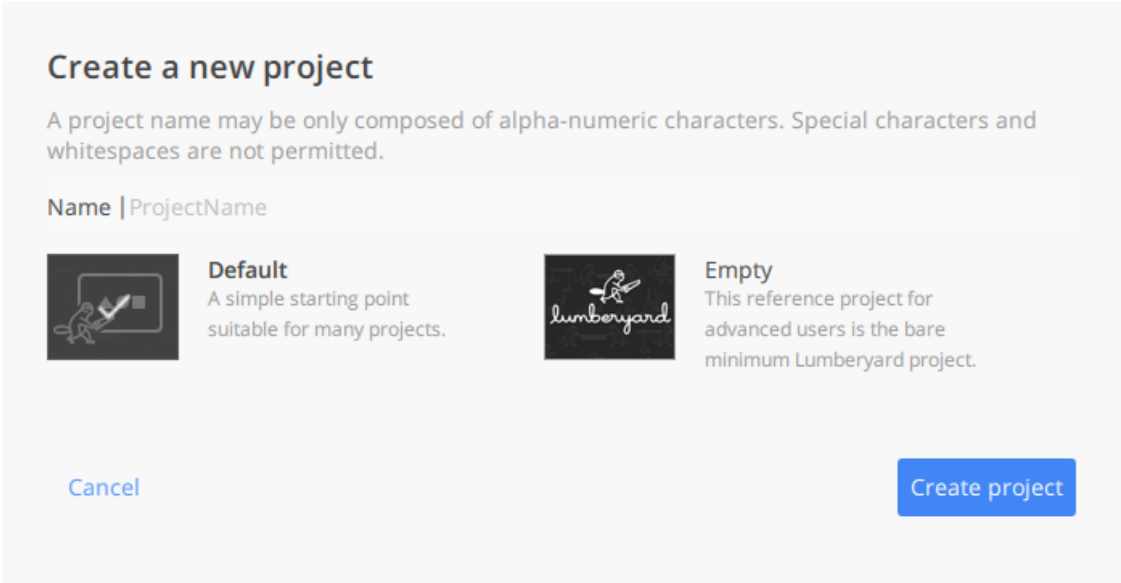
When you create a new project Project Configurator creates a new directory in *`lumberyard_version`*`\dev\` with your project's name. Any project-specific files like configuration data, assets, levels, and scripts live within this project directory. Throughout this process, Project Configurator creates the necessary files and Visual Studio solutions, then builds your project. This process can take some time.

To create a new project

1. Choose the **Create new** button in the upper left of the Project Configurator to bring up the **Create a new project** window.

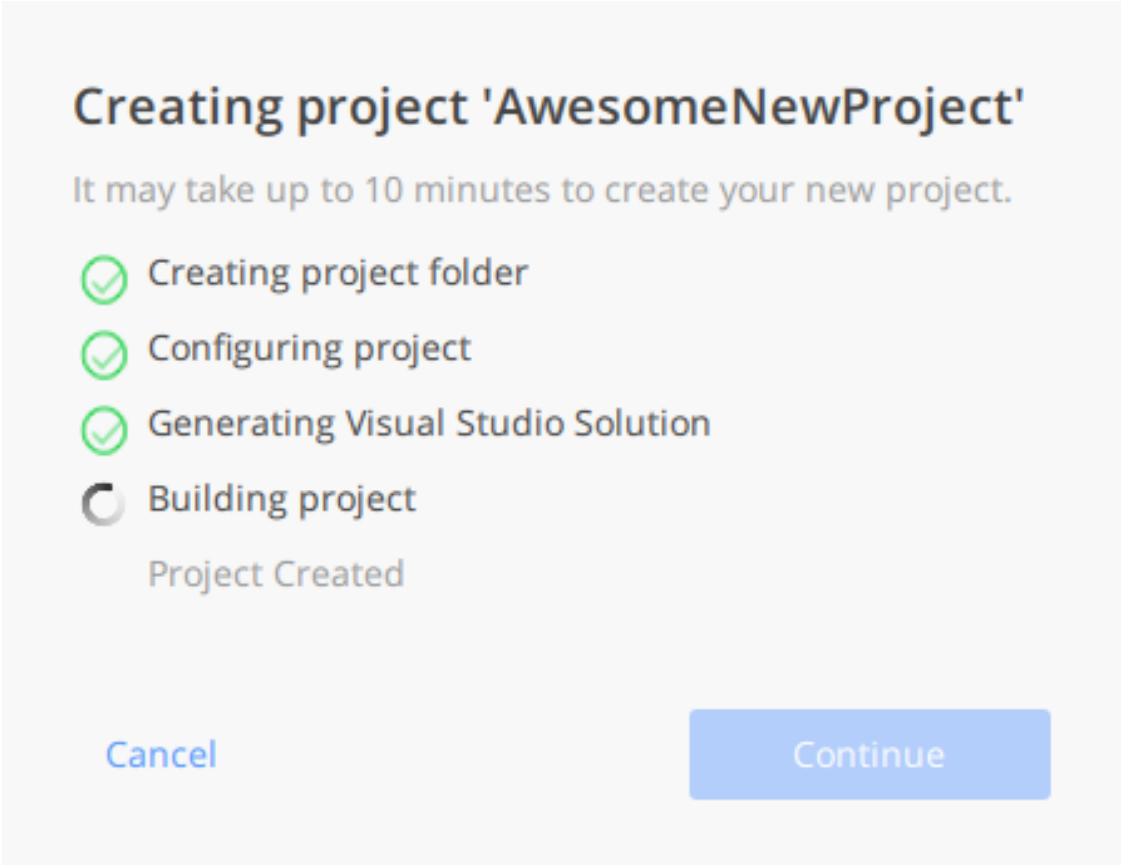


2. In **Create a new project** give your project a name in the upper left and select either the **Default** or **Empty** project template.



   Note
   Many features in Lumberyard are implemented in packaged extensions called Gems. The **Default** project template contains a basic set of commonly used Gems. The **Empty** project template contains a minimal set of required Gems. Once you have created a project, you can add or remove Gems.

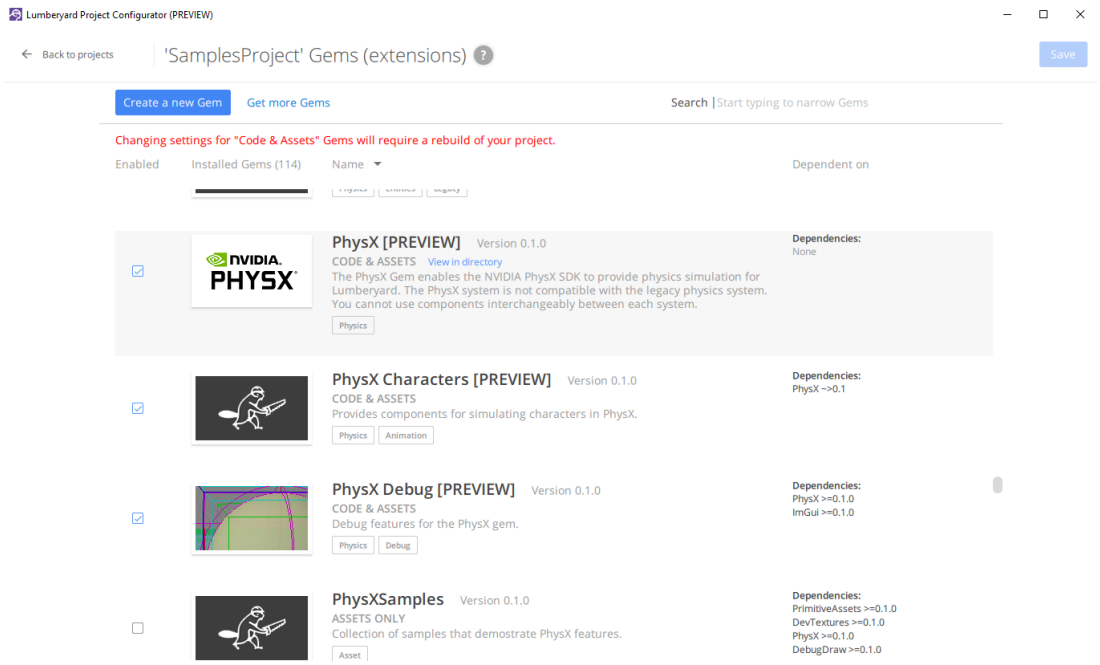
3. Choose **Create project**. A new window displays your progress as your project is created and built. The build process can take some time based on your system.



4. When your new project has been built, choose **Continue** to return to the main Project Configurator interface. Your new project should be selected. If not, choose the project icon to select it. Choose the **Set as default** button to make your project the default project.
5. Your project is now ready. You can move on to Lumberyard Editor or take the time to explore some of the advanced features of the Project Configurator before moving on. Next to your new project icon are three links:

   - **Enable Gems**

     Choose **Enable Gems** to open the Gems editor. Gems are packaged extensions that add new features and assets to your project. You can create your own Gems here, similarly to how you created a new project. Scrolling down the list of available Gems, you'll notice that some gems are enabled (their box is checked). To add or remove Gems, check or uncheck the box next to the Gem.



Note
Adding or removing Gems might require rebuilding your project, which can take some time. To rebuild your project, choose the **Rebuild** button on the main Project Configurator page.

- **Advanced Game Settings**

  Choose **Advanced Game Settings** to open the game settings view. In this view, you can modify your project's memory allocation and other settings that are exposed by Lumberyard, as well as the Gems that have been added to your project.

- **Advanced Editor Settings**

  Choose **Advanced Editor Settings** to open the settings view. In this view, you can modify Lumberyard Editor settings for your project.

Create with Amazon Lumberyard

Previously in this guide, we covered How Amazon Lumberyard works (p. 5) and Setting up Amazon Lumberyard (p. 12).

Now, let's get started creating your game! In this topic, we provide an overview of creating a game with Lumberyard, and help you identify where to focus your learning, based on your role in game development. If you're proficient in all roles, read everything!

How do I create a game with Lumberyard?

Keeping it Modular

Game development in Lumberyard is best understood if you keep a key Lumberyard design philosophy in mind as you learn: the concept of **modularity**. The Lumberyard game engine, its systems, and its environment are built as a collection of C++ modules. Choosing the right modules based on the combination of your game's design and your workflows will keep your game development process focused and likewise keep the overhead of managing your game project simpler.

Even the Lumberyard Editor – the most common tool associated with Lumberyard – uses these modules, which we call **Gems**. Lumberyard ships with well over 100 Gems as part of the installation, and you can acquire other Gems from third parties, or write new ones yourself. When you build your game, the functionality of these Gems is combined to create the systems of your game. This means that you don't have to hard-code the systems and features of your game. Instead, you can either obtain or create a Gem with the functionality you need – and even maintain the code for it independent of the game! This isolation means that a game developer can work on game-specific AI functionality in a Gem. When the Gem is updated, level designers can access the new functionality after a rebuild of the game project, and without impacting any other Gem. Gems can also add new features to the Lumberyard Editor and provide new asset processing behaviors.

You'll notice this philosophy of modularity in the individual entities you create, up through the most complex functionality you add.

For example, consider the concept of an entity in Lumberyard. An entity can represent just about anything in your game under Lumberyard's component entity system. By giving the entity components, you begin to shape their utility in your game. The components specify entity behaviors and properties.

You might have an entity that you want to see and interact with in your game world that uses all of the following components:

- The default transform component to define its position
- A mesh component to define its visual geometry
- NVIDIA PhysX components to define collision characteristics and other aspects for a realistic, rigid body simulation
- An input component to reference an input event-binding definition
- A script component to automate some sort of behavior, or process input events from the player
- A camera component to allow the entity to be used as a camera

...and many more!

Other entities that you create will be invisible to players. These entities might exist to implement triggers, spawn environmental effects, or reference assets created with tools, such as the game UI that you created in UI Editor.

A major part of assembling a game in Lumberyard revolves around using the Lumberyard Editor to do the following:

- Place and group entities
- Add components to these entities
- Configure properties on these components
- Use tools associated with the components

Tools you might use that are associated to specific components include:

- Script Canvas to create and edit scripts using a visual scripting system, then reference from the Script Canvas component.
- Emotion FX Animation Editor to animate characters, then reference from the **Anim Graph** component.
- Asset Editor to create input bindings that bind raw player input, such as keystrokes, to events, then reference from the **Input** component.
- Audio Controls Editor to setup sound effects that map to Wwise controls, then reference from the **Audio Trigger** and **Audio Switch** components.

...and many more.

Some tools can be opened directly from their associated component. Others require you to open them from the **Tools** menu in Lumberyard Editor. For a tour of Lumberyard Editor, see Introduction to the Lumberyard Editor (p. 28). For an overview of the tools provided with Lumberyard, see the following topic on Tools available in Amazon Lumberyard (p. 32).

The modular nature of Lumberyard means there are additional assets, components, and tools that you can add by enabling Gems in Project Configurator. Lumberyard comes with a library of Gems. Gems can include new code, new assets, or both! You can even write your own. In fact, this is what a game's C++ programmers will often spend their time doing, to help create the gameplay that makes your game unique.

Starting the Journey

Are you starting a new project, maybe to start a prototype of your game? Or to just play around with Lumberyard – take it for a spin and learn what it can do? You'll probably need to know how to get the minimum systems going, which typically include:

- Camera
- Rendering
- Physics
- Input

You'll use the Project Configurator tool to set the project that you're working on. You can use one of the projects that ship with Lumberyard, such as the Samples Project, if you want to learn or play around with something premade. Or, if you want to start something new, this is also where you create a brand new project, based on the template of your choice.

Then, launch the Lumberyard Editor, open one of the sample levels in the Sample Project, and start exploring! We suggest following along with one of these written or video tutorials:

- Learn Lumberyard in 20 Minutes
- Set Up Amazon Lumberyard series
- Creating a Controllable Entity

- Basics of Motion series

Joining a Team

Depending on your role in your game development team and the scope of the game project that you're working on, you might not encounter all of the tools and technologies that Lumberyard provides. In the next section, we'll take a look at how you might want to focus your learning path.

Work as an artist

Lumberyard provides all of the basic tools that you need to import, assemble, and blend animations in a AAA game, creating new worlds or environments for the player to visualize, interact with, and experience.

While much of your work as an artist might involve using tools outside Lumberyard, you will use Lumberyard tools when you want to control your character animations, and manipulate the look and feel of your assets or the environment. You might be working with designers to create the game environment.

As an example, let's take a look at the workflow for setting up an actor in your game.

1. Outside of Lumberyard, you create a character model, materials, textures, and rig for the character.
2. Lumberyard's Asset Processor automatically processes source assets into platform-specific game assets, ensuring new or modified files are ready to use in Lumberyard as soon as possible. Use **FBX Settings** if you want to modify the processor settings.
3. You import your actor file in Animation Editor and create a motion set to specify the motions that you want for your character.
4. Next, you create an animation graph using nodes.
5. Then, you build a blend tree to blend the animations together.
6. When you've built and previewed the animations and are ready to try them out in a game environment, you can switch over to Lumberyard Editor.
7. In the Lumberyard Editor, you create or open an existing test level.
8. To see your animated character, you need an entity with:
   - an **Actor** component to create a controllable character with the actor file from Animation Editor and a material linked to your actor asset.
   - an **AnimGraph** component to use the animation graph and motion set assets that you created in the Animation Editor.
9. To control your character in your level, you might want to work with a gameplay designer or programmer at this point to add an **Input** component, **PhysX** components, and script components so you can run through and playtest all of your character's animations in your game's specific environment.

We suggest that you begin your learning path by browsing the following set of Lumberyard tools and technologies, and then focusing on the ones that apply to your needs:

- Lumberyard Editor
- Asset Pipeline
- Component Entity System
- Component Reference
- Emotion FX Animation Editor

- **FBX Settings** Tool
- Gem Library
- Cinematics and the **Track View** Editor
- Shaders and Materials
- Terrain and Environment
- Vegetation Editor

Some specific tutorials you might want to look at include:

- Import Assets into Lumberyard
- Animation System and Emotion FX
- Lighting

Work as a designer

Lumberyard Editor is an important, core tool for game designers and level designers. It's where you create your levels, populate them with entities, and assign components to those entities. It also provides access to important tools such as the UI Editor for UI designers, the Audio Controls Editor for sound designers, and Script Canvas for all designers who will be working with the visual scripting system in Lumberyard.

Here's how you might start out:

1. When you start Lumberyard Editor for the first time, you can create a new level, or you might prefer to open one of the existing levels in the Samples Project to play around with. If the Samples Project is not your current project, use Project Configurator first, and set it as the default project.
2. You start populating your level in the viewport by creating entities. In Lumberyard, an entity can be just about anything, from the static objects you see, to the triggers you script, to the placeholder objects that reference game UI.
3. You add components to your entities through the Entity Inspector tool.
4. To provide player control for your entities, you'll need an input binding. Using the Asset Editor tool, you bind raw player input from keyboard, mouse, and game controllers to events that you create. Then you can listen for and respond to these events using one of the scripting tools.
5. Script Canvas and Lua are common scripting tools that are used in Lumberyard. Script Canvas provides a visual, node-based scripting system for scripting your gameplay logic, while Lua provides a more traditional scripting environment based on the Lua API. Add a Script Canvas component or Lua Script component to an entity. Develop a script in one of the script editors. Then you can add that script to the component to control that entity at runtime. For example, using a script, you can control an entity by responding to player input events, spawning dynamic entities at runtime, producing visual effects, and much more.
6. As you populate your world with entities, you'll learn that a great way to save time is to use the slice system. Slices are a type of prefab system that allows you to group and nest component entities together, save the group as a slice, and then create multiple instances of that slice throughout the levels of your project. In each instance of the slice, you have the ability to make changes to that specific instance. This change is referred to as a slice override. You can also choose to save the override to the original slice, which then pushes that change to every other instance, too.
7. When you're ready to author game UI, you use the UI Editor, where you can establish a UI canvas and layout your interface and script your workflows.
8. When you're ready to add sound to your game, you establish audio events and triggers in the Audio Controls Editor. You can then add this audio to entities through components, and script its playback using one of the scripting systems.

To get the most out of Lumberyard, browse the Components library and the Gems library to see what Lumberyard has to offer. Then talk to your programmers to see what additional components need to be authored for your game.

We suggest that you begin your learning path by browsing the following set of Lumberyard tools and technologies, and then focusing on the ones that apply to your needs:

- Lumberyard Editor
- Asset Pipeline
- Component Entity System
- Component Reference
- Gem Library
- Script Canvas
- Lua Editor
- AI Navigation
- Audio Controls Editor
- Emotion FX Animation Editor
- Cloud Canvas
- UI Editor

Some specific tutorials you might want to look at include:

- Creating a Controllable Entity
- Script Canvas series
- Basics of Motion series
- Working with Slices
- Simulate Physics
- Modify Terrain
- Import Assets into Lumberyard
- Animation System and Emotion FX

Work as a game engineer

As a Lumberyard game engineer, you will likely need to learn both how to support the designers and artists in the design of the game and environment, and how to author individual components. These new components can then be added to entities in the Lumberyard Editor to create custom gameplay. You might also learn how to develop new Script Canvas nodes. These new nodes can then be used by designers in the Script Canvas editor to handle new events that you created, or change the properties of your new components. On a larger scale, when you need to work on a system that can be distributed as a shareable container of code and assets, you can learn how to create a Gem.

We suggest that your learning path looks like this:

1. Follow the intro tutorials to learn about the component entity system and the existing library of components.
2. Browse the Gem library to see examples of the larger functionality they can add compared to individual components.
3. Learn about authoring your own components and Gems, where you will also learn about working with EBuses, Lumberyard's event bus and general-purpose messaging system; AZ Modules, a collection of C ++ code built as a static or dynamic library; and more.

Here is a basic set of Lumberyard tools and technologies to focus on:

- Lumberyard Editor
- Programming Concepts
- Gems
- Component Entity System
- Component Reference
- Programmer's Guide to Entities and Components
- Emotion FX Animation Editor
- Script Canvas
- Lua Editor

Some specific tutorials you might want to look at include:

- Creating a Controllable Entity
- Import Assets into Lumberyard
- Script Canvas series
- Working with Slices
- Working with Gems
- Simulate Physics
- Animation System and Emotion FX

Introduction to the Lumberyard Editor

Lumberyard Editor is your primary workspace. From here, you access all of the tools to design, create, test, play, and deploy your project. If you have used other professional engines or 3D animation packages, you'll find the user experience familiar and adapt to Lumberyard Editor quickly.

Lumberyard Editor can be launched from the start menu or the Lumberyard Editor desktop icon. When Lumberyard Editor launches, you're given the option to create a new level or load an existing level. If you're using one of the example Lumberyard projects, you will find example levels in the **Levels** directory of the project. If you're working with a new project, you must create a new level.

For a 20-minute crash course on navigating the **Perspective** viewport, customizing the Lumberyard Editor layout, creating entities, and working with components in Lumberyard Editor, see the following video tutorial.

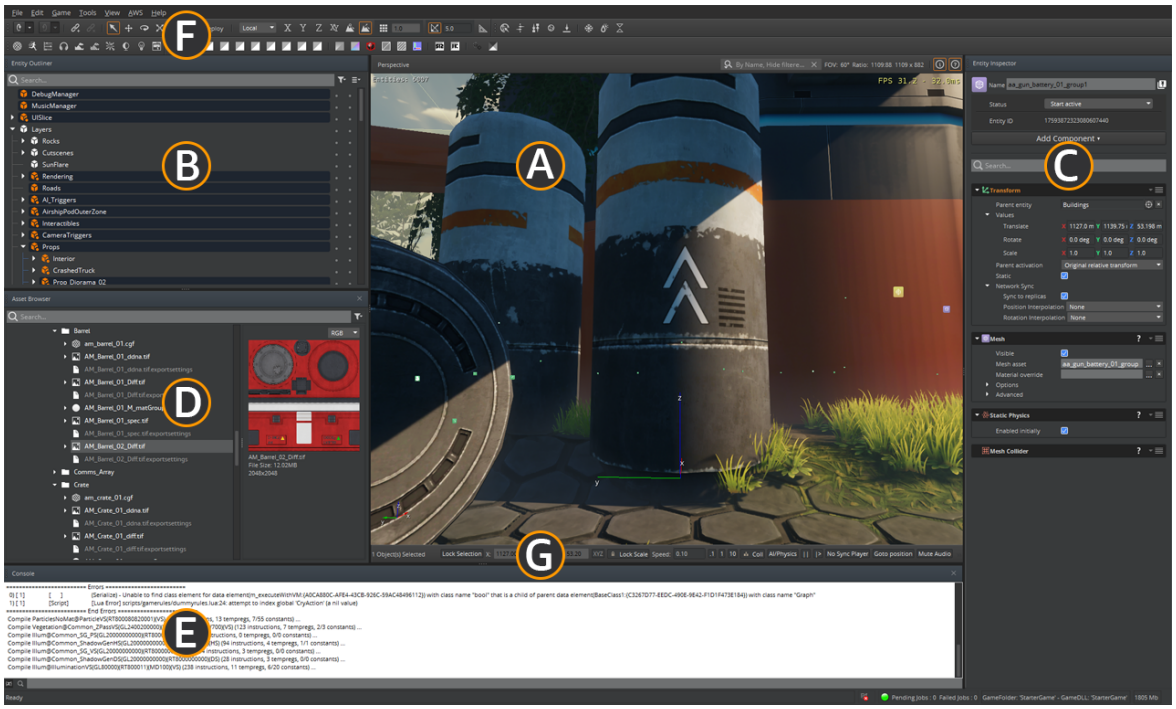
Lumberyard Fundamentals 20 Minute Editor Crash Course

The Lumberyard Editor default layout

The default layout of Lumberyard Editor contains the most commonly used tools in a configuration, similar to other content creation applications. The core workflow of Lumberyard is to create and place *entities* in a level, so the default layout contains a menu bar, two toolbars, and five tool panes focused on entity creation and placement.

You can customize the layout through drag and drop, and save to a custom layout through the **Layouts** option in the **View** menu of the main menu bar. Drag the separator bars between panes to resize the

panes. Drag the title bar of a pane to tear off the pane. The pane can be dropped anywhere in the layout or dropped outside of Lumberyard Editor as its own window. To restore the default layout, select **Default Layout** from the **Layouts** option in the **View** menu of the main menu bar.



Tools available in the default Lumberyard Editor layout

A. **Perspective** – This 3D viewport is a real-time view of your game-ready level. In **Perspective**, you create and place entities, and view and play your project.

   Click **Perspective** in the title bar of the pane to open the perspective menu. From the perspective menu, you can toggle visibility for various helpers such as the construction plane, icons, and guides. You can also select an aspect ratio, view through various cameras placed in the level, create new cameras from the current view, and split the **Perspective** pane into multiple views.

   Right-click in **Perspective** to open the context menu to create entities and *slices*, which are reusable assets that contain multiple entities. From the context menu, you can also create *layers* that you can use to organize entities and slices in your level.

B. **Entity Outliner** – The **Entity Outliner** displays a list of entities, slices, and layers in the current level.

   Right-click in the **Entity Outliner** to open the context menu to create entities, slices, and layers. Much of the functionality of the **Entity Outliner** context menu is shared with the **Perspective** context menu. The **Entity Outliner** context menu also has options to find selected entities and slices in **Perspective**, organize the list in the **Entity Outliner** and find slices in the **Asset Browser**.

C. **Entity Inspector** – The **Entity Inspector** displays the components of the currently selected entity. At the top of the **Entity Inspector** is a field for the entity **Name** and an **Add Component** button. The **Add Component** button opens a list of components, sorted by type, that can be added to the entity. Each component has its own set of properties that are displayed in the **Entity Inspector**. All entities contain a transform component that sets the position, rotation, and scale of the entity in the level.

D. **Asset Browser** – The **Asset Browser** browses your project's on-disk assets. Assets such as meshes, animations, and textures are created in third-party applications. Assets such as materials, scripts, and slices are created within Lumberyard Editor. The assets that you create are stored in your project directory. You can also browse default assets that are included with Lumberyard, as well as assets that are included with Gems that have been added to your project.

The left of the pane of the **Asset Browser** displays a directory structure that you can browse for available assets. When an asset is selected, the preview pane on the right displays a thumbnail preview and information about the asset, if available.

E. **Editor Console** – The **Editor Console** shows command and process output from Lumberyard Editor and your project. When you load a level, for example, the console displays messages about assets and configuration files as they load, and might display warnings and errors if issues are encountered. You can enter console commands such as setting console variables in the entry field at the bottom of the console. Click the **{x}** button in the lower left of the **Editor Console** to open the **Console Variables Editor**, which provides a simple interface for setting console variables.

F. **Toolbar** – The **Toolbar** provides easy access to 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. 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.

G. **Perspective Toolbar** – The **Perspective Toolbar** at the bottom of the **Perspective** pane displays position information for selected objects. You can also adjust navigation speed; mute audio; go to a specific position; toggle collisions, AI and physics and enable VR preview mode.

Navigating the Lumberyard Perspective viewport

Lumberyard's interaction model will be familiar to anyone who has played a first-person PC game, with a few minor tweaks and additions. Movement is handled by keyboard input, and view is handled by pointer device input.

Keyboard navigation

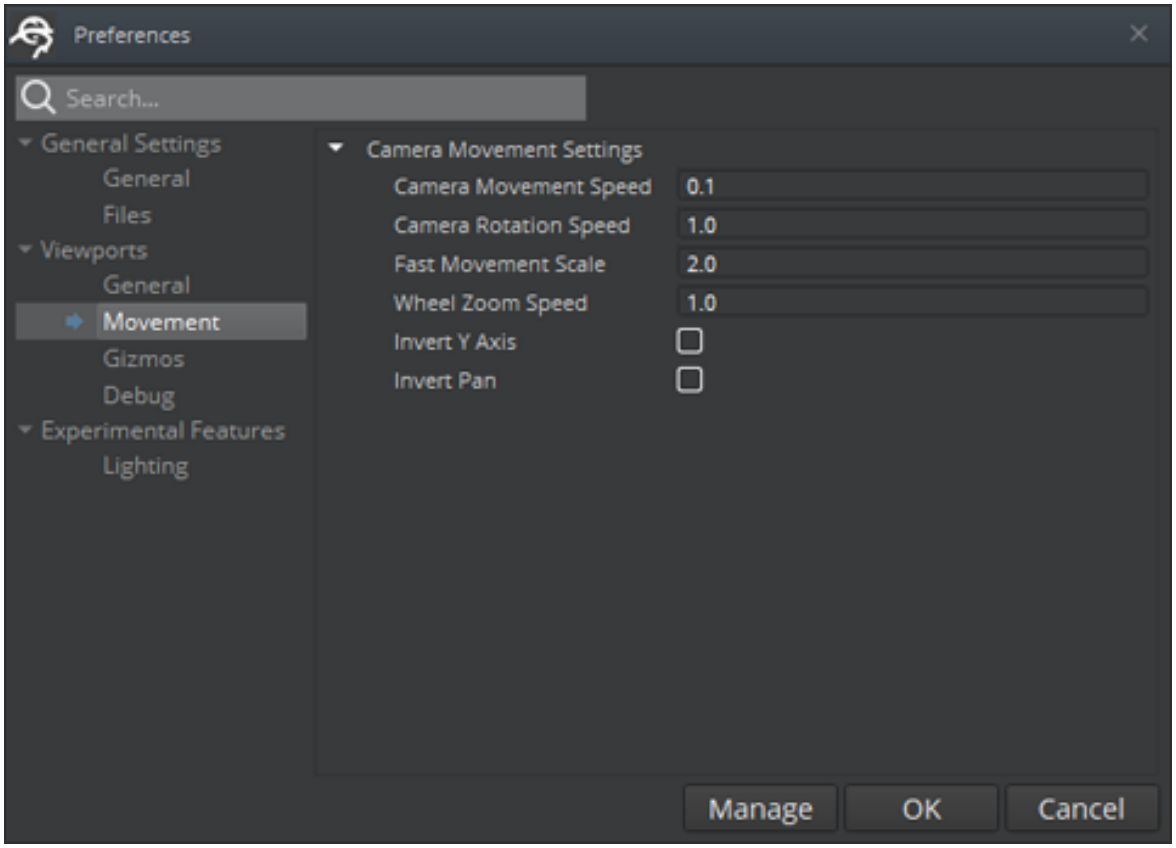
- **W** – Move forward.
- **S** – Move backward.
- **A** – Move left.
- **D** – Move right.
- **Q** – Move down.
- **E** – Move up.
- **Z** – Focus on selected.

Mouse navigation

- **Right mouse + drag** – Rotate view, known as *mouselook* in most games.
- **Mouse wheel scroll** – Zoom view.
- **Middle mouse + drag** – Pan view.
- **Left mouse** – Select entity.
- **Left mouse + drag** – Area select entities.

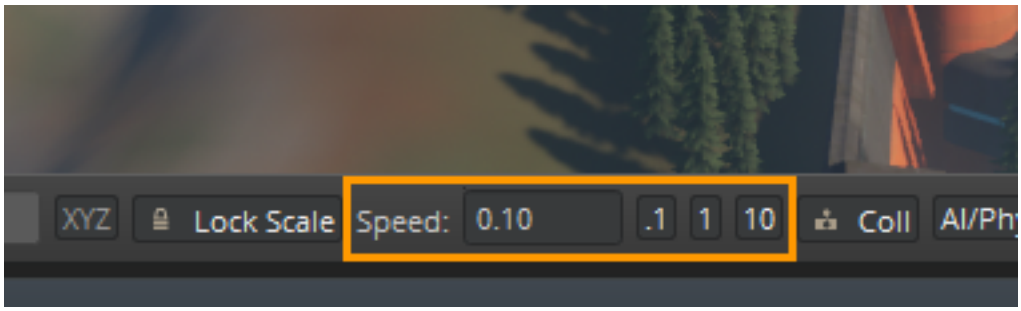
Movement preferences

You might prefer that your editor camera controls behave like a flight simulator. Or you might want to speed up or slow down the default movement or rotation of the editor camera. You can adjust the default editor camera control behavior by setting the **Movement** properties in the **Global Preferences** editor.



Choose **Global Preferences** from the **Editor Settings** group in the **Edit** menu. Select **Movement** under the **Viewports** list on the left. Here, you can invert either mouse axis and adjust the movement speed of the editor camera.

When you have your movement preferences set to your liking, you might find at times that the editor camera movement is too fast or too slow in certain situations. You can adjust the movement speed in the **Perspective Toolbar** at the bottom of the **Perspective** pane.



Enter a floating point value in the **Speed** property to set movement speed. You can also click one of the three buttons to the right of the **Speed** property to set the movement speed to **.1**, the default value **1**, or **10**. Values greater than **`1.0`** increase movement speed. Values less than **`1.0`** decrease movement speed.

Save Perspective locations

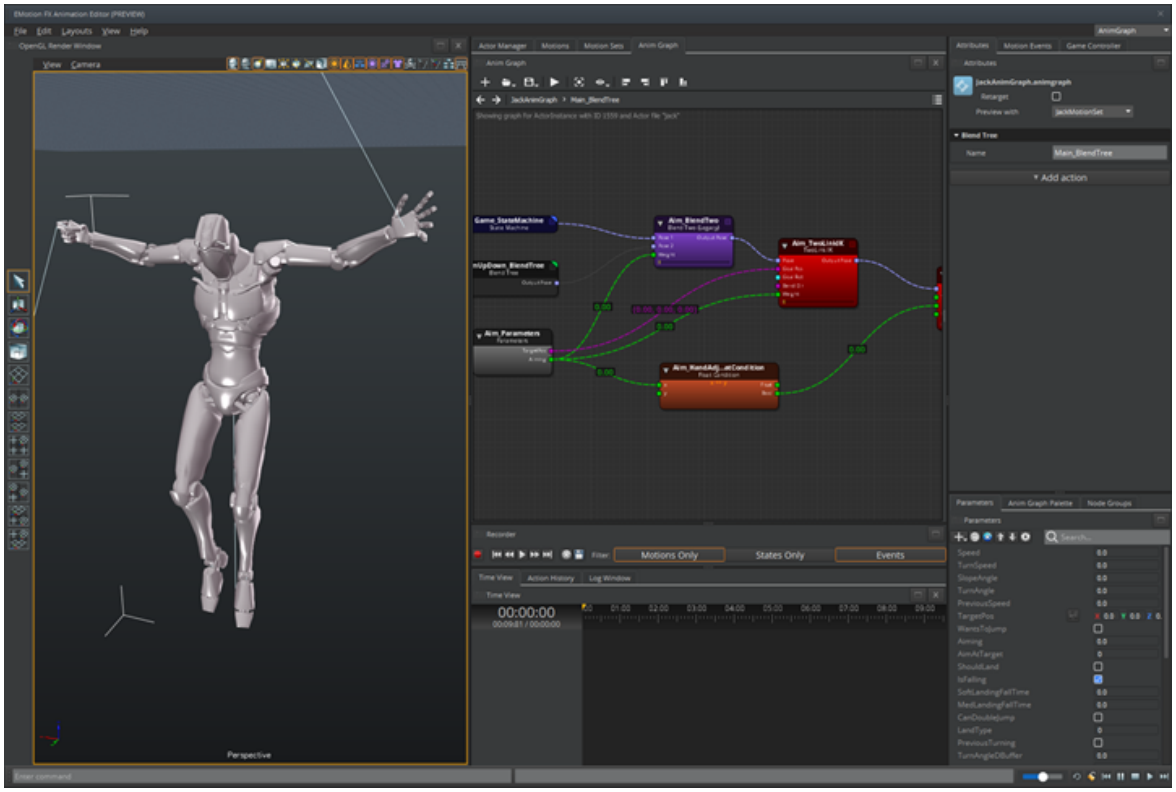
As you build a level, you might find that it's helpful to have preset **Perspective** views saved for later use. You can save the current editor camera view, assigning it to a **Function** key. To save a **Perspective** location, press **Control + Function(1-12)**. To set the **Perspective** view to a saved location, press **Shift + Function(1-12)**.

Tools available in Amazon Lumberyard

The default **Lumberyard Editor** layout contains tools common to most development workflows, but there are a lot of additional tools available. Depending on your development role, these may not always be the tools you need - or even the right tools for your job. The **Lumberyard Editor** supports custom layout options so that you can add all of the tools you need to use most frequently right where you need them. Some tools that have multiple panes, such as the **Animation Editor**, can have their own custom and preset layouts.

These tools are available in the **Tools** menu of the **Lumberyard Editor**.

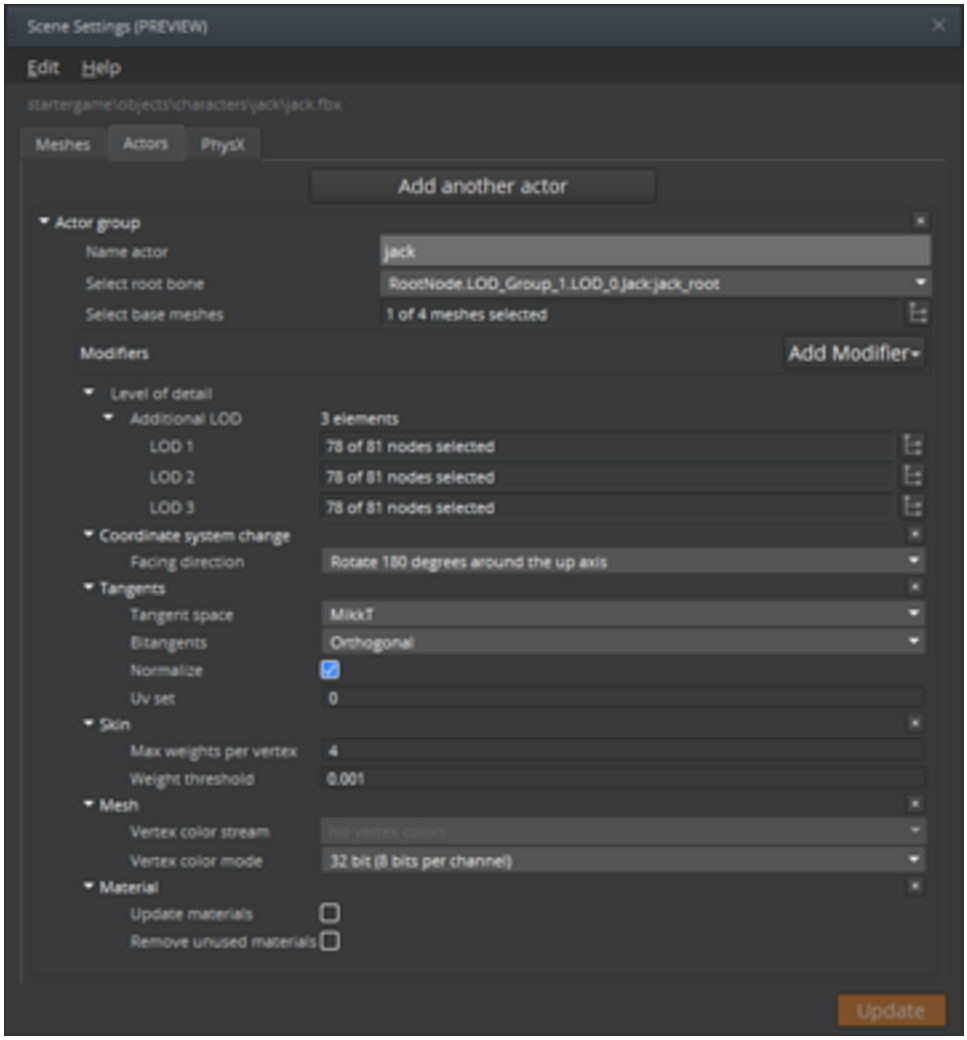
Animation Editor



With **Animation Editor**, build animated *behaviors* for your *actors*.

Use the Animation Editor to build animation loops and set up smooth transitions between them. To create behaviors, begin by loading an actor, which is a skinned mesh, and its corresponding animations. The animations are added to *motion sets*. You create *motion graphs* using motion sets that can blend animation based on states and events. The motion graphs that you create define your actor's behaviors.

FBX Settings

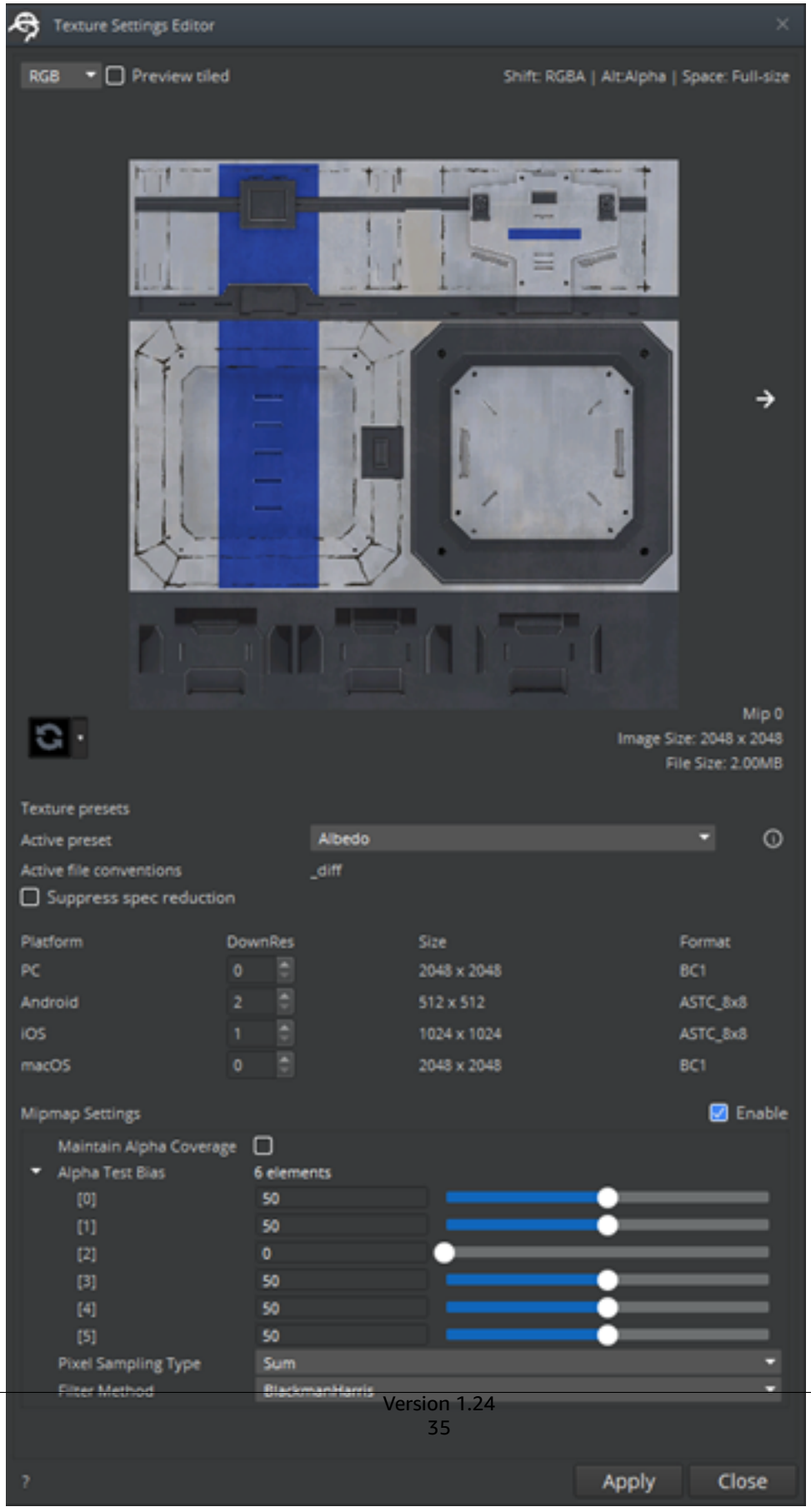


The **FBX Settings** tool converts static `.fbx` meshes, actors, PhysX meshes, and motions into Lumberyard assets.

When you export or copy `.fbx` files to a directory in your current game project, Asset Processor detects the files. Using these files as input, Lumberyard calculates default settings. These settings specify how Asset Processor converts the files into the appropriate mesh, materials, or animation assets.

To customize `.fbx` settings, find the asset in **Asset Browser**. Right-click the asset and select **Edit Settings** to open **FBX Settings**. The available options in **FBX Settings** vary depending on the contents of the `.fbx` file.

Texture Settings Editor



Texture Settings Editor converts image files into Lumberyard assets.

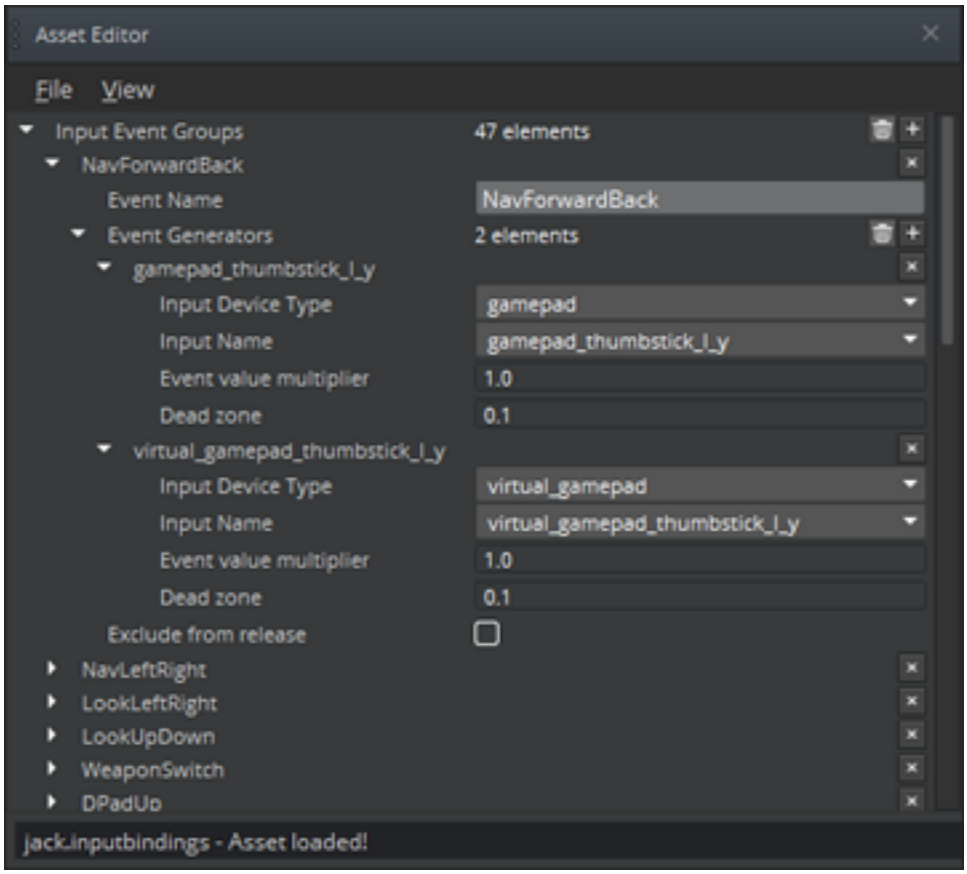
When you export or copy image files to a directory in your current game project, Asset Processor detects the files. Using these files as input, Lumberyard calculates default settings. These settings specify how Asset Processor converts the files into the appropriate textures.

To customize the texture settings, find the image asset in **Asset Browser**. Right-click the asset and select **Edit Image Settings** to open **Texture Settings Editor**. In **Texture Settings Editor**, you can choose from presets based on the texture type and specify settings for various platforms. **Texture Settings Editor** displays a preview image of the texture so you can view the settings results.

Image types supported by Lumberyard

- `.bmp`
- `.gif`
- `.jpg`
- `.jpeg`
- `.png`
- `.tga`
- `.tif`
- `.tiff`

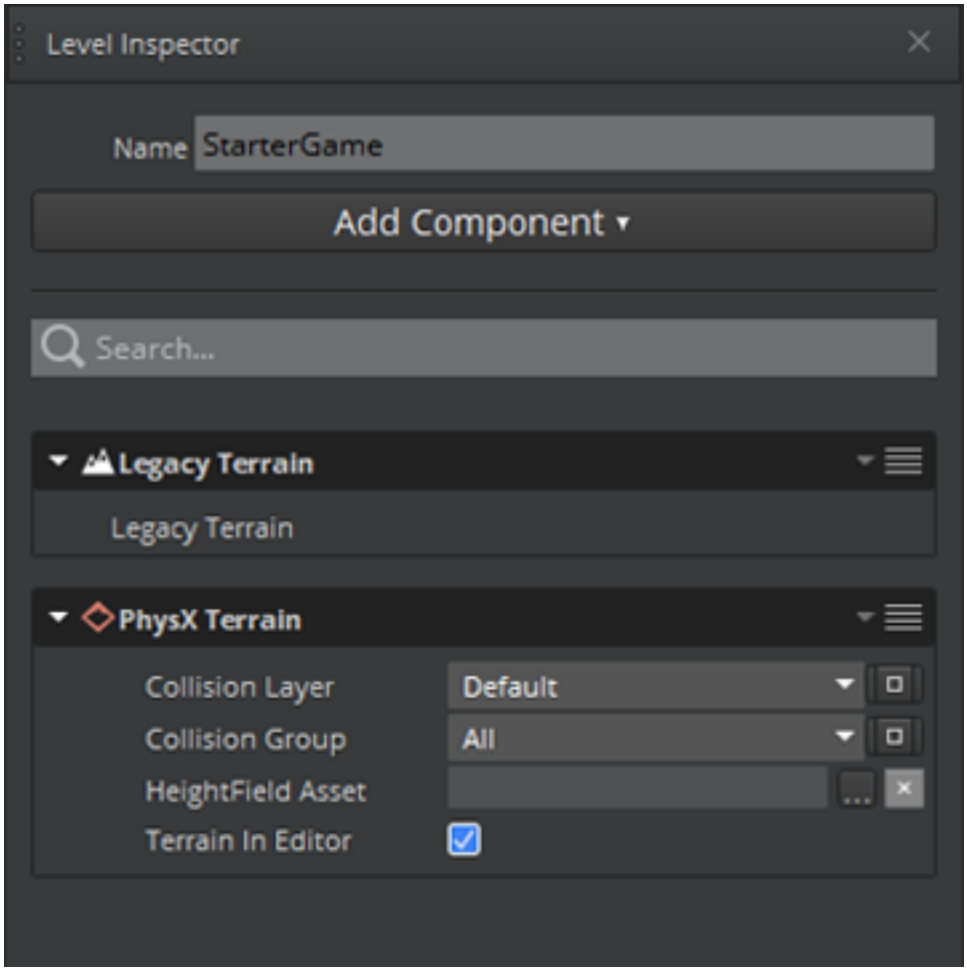
Asset Editor



Create and edit Lumberyard-specific assets in **Asset Editor**.

Lumberyard has a small number of specialized assets such as *script events* that allow scripts to communicate with each other, *physics materials* that give surfaces physical properties like friction, and *input bindings* that bind input to events. Create and edit these specialized asset types with the **Asset Editor**.

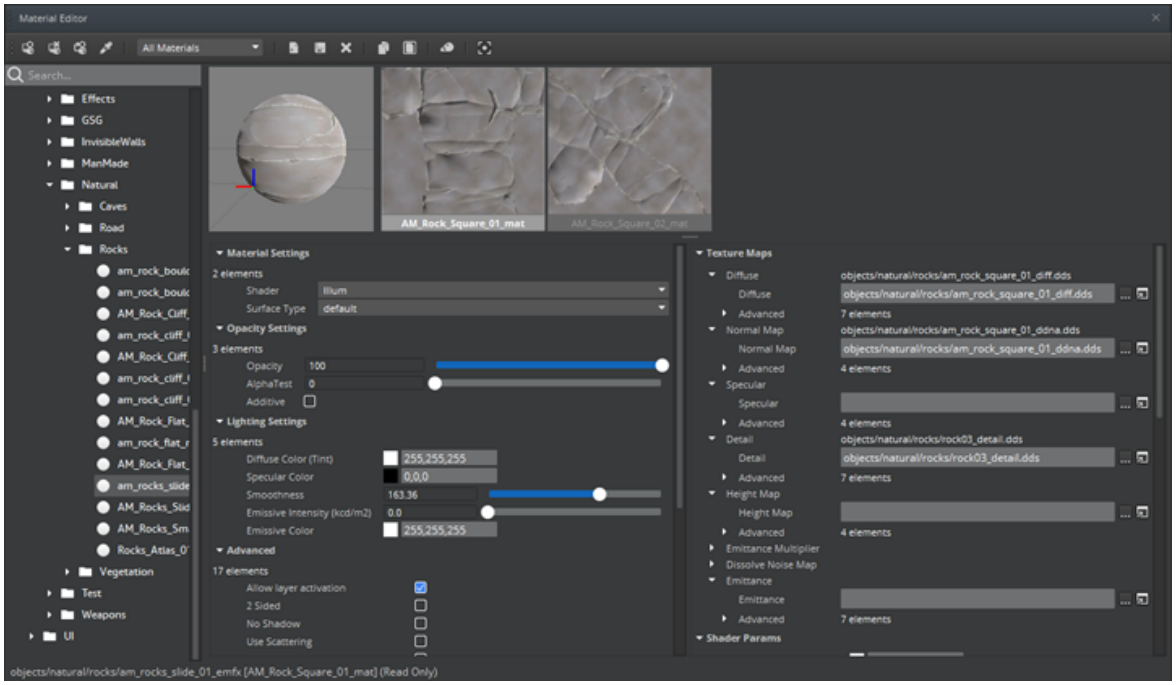
Level Inspector



Add and modify level components in **Level Inspector**.

Level Inspector allows you to add and modify level components, similar to **Entity Inspector**. In **Level Inspector**, you can add terrain components and NVIDIA PhysX terrain components for terrain collisions.

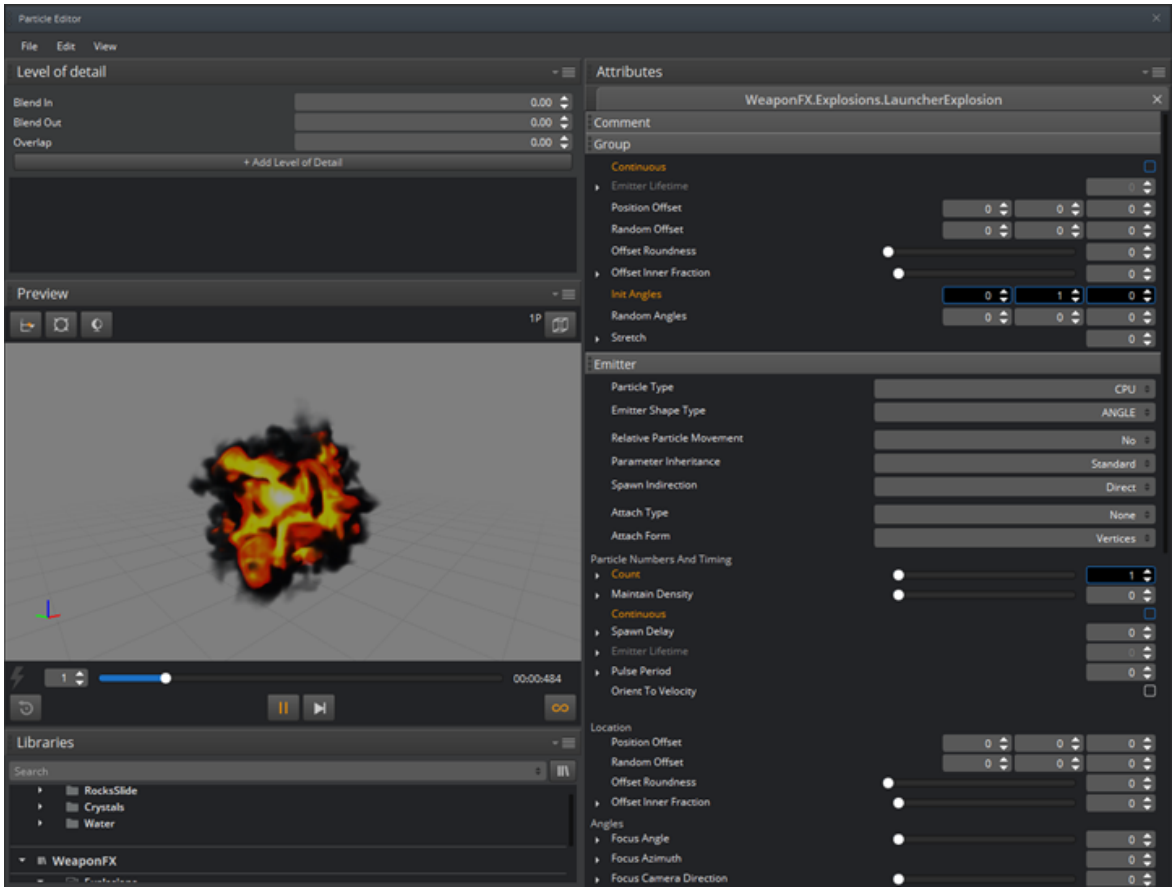
Material Editor



Create and edit the appearance of the entities in your project with **Material Editor**.

A material has a set of properties that determines how its surface reacts to physical actions, other materials, and its environment. **Material Editor** is the primary tool used to create materials and map textures, and lets you configure texture properties like opacity, lighitng effects, shader parameters, and more.

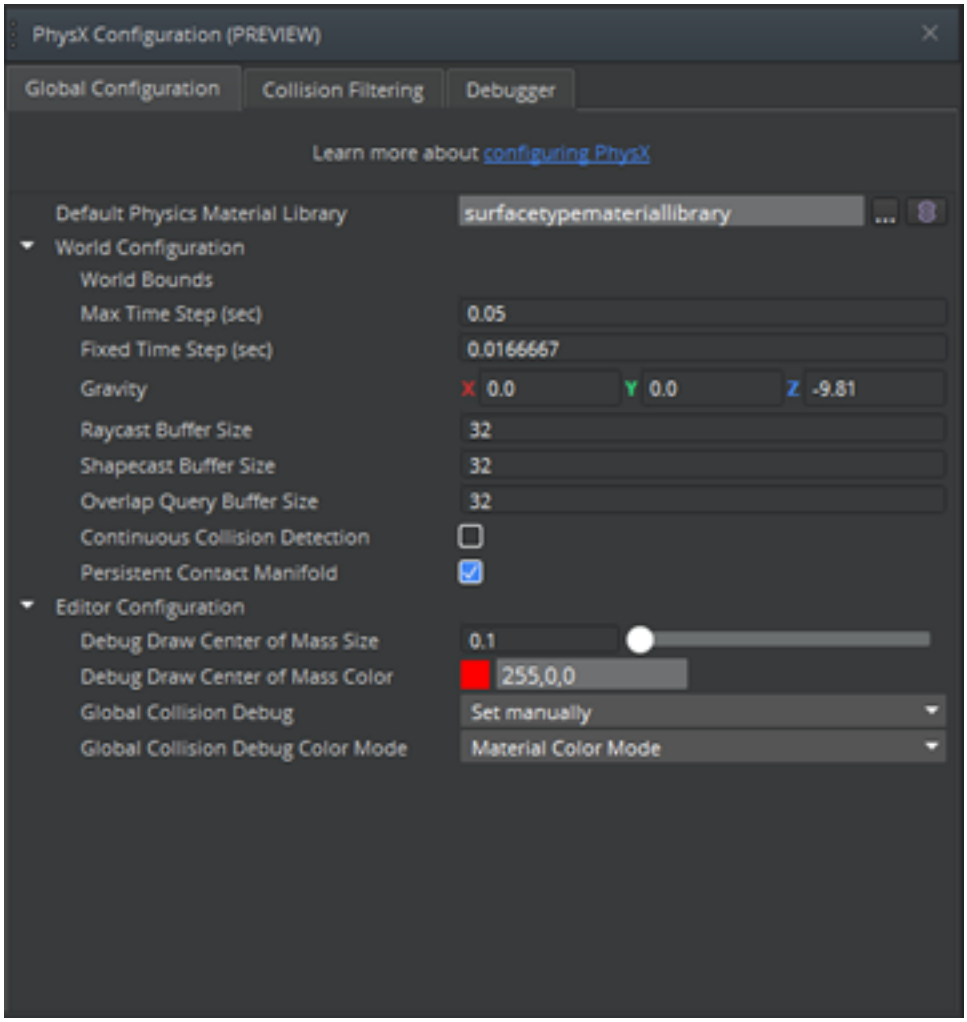
Particle Editor



Build layered, dynamic visual effects with **Particle Editor**.

Lumberyard includes an advanced particle effects system that you can use to simulate environment effects like fire and sparks, or weather effects like fog, snow, or rain. Use **Particle Editor** to create and manage libraries of particle effects in your project.

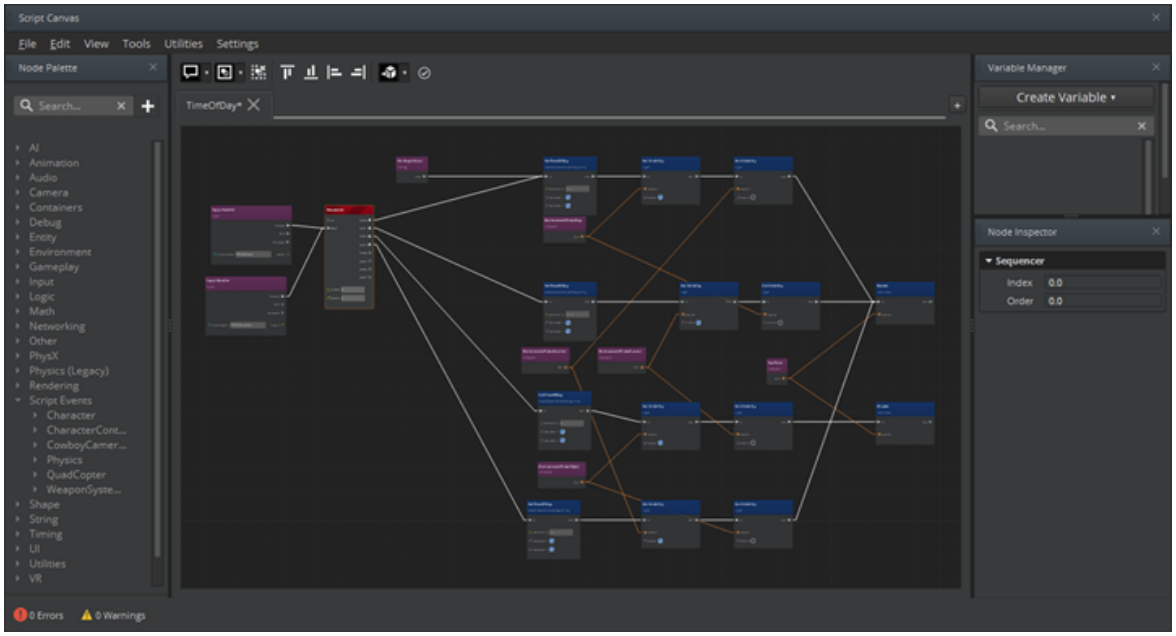
PhysX Configuration



Set the global PhysX properties for your project with **PhysX Configuration**.

Lumberyard integrates NVIDIA PhysX for real-time physics simulation. With **PhysX Configuration**, you can set global properties for NVIDIA PhysX such as gravity, balance simulation performance and accuracy, create filters with collision layers and groups, and set up a PhysX visual debugger.

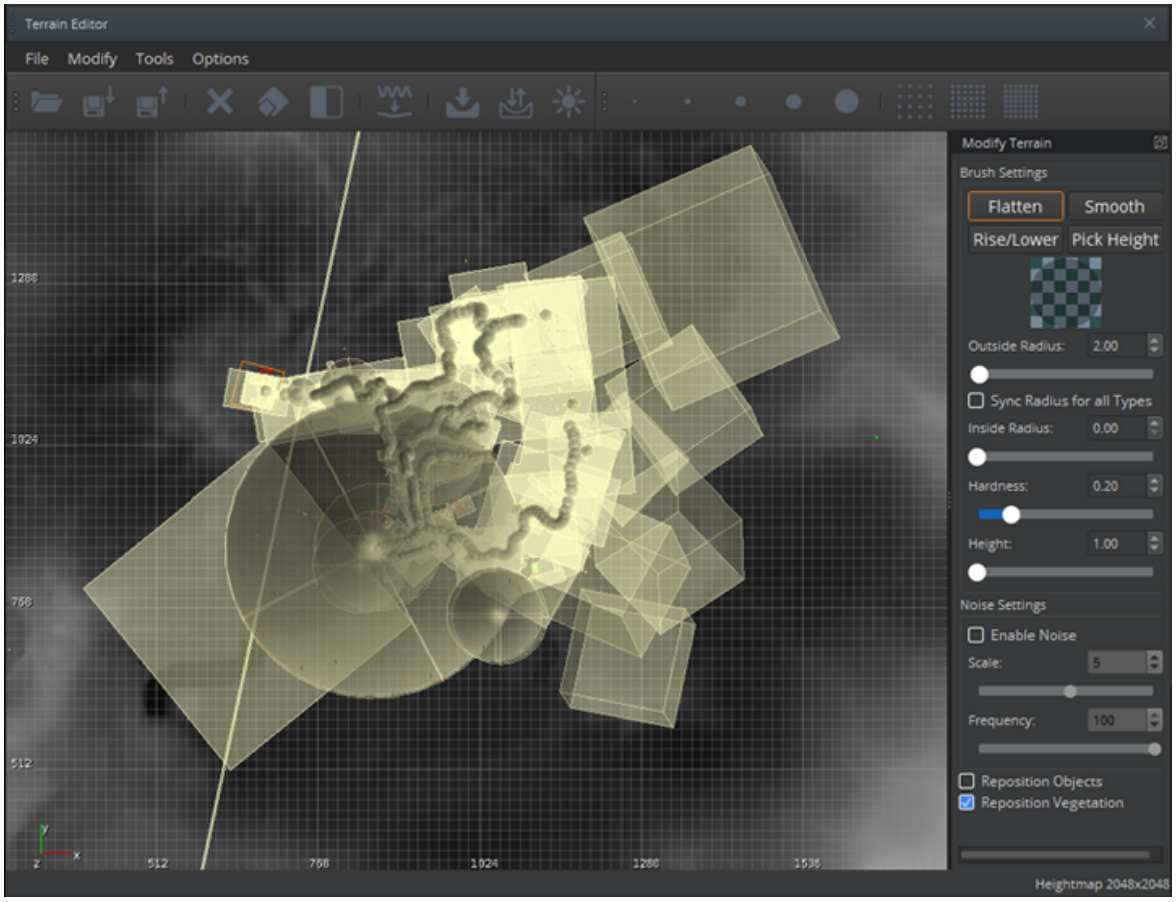
Script Canvas



Program logic and behaviors visually with **Script Canvas**.

Script Canvas is one of Lumberyard's most powerful tools. With **Script Canvas**, you can create behaviors, functions, and logic in a visual programming environment. **Script Canvas** is designed to use Lumberyard's serialization, reflection, modularization, and EBus messaging systems. It's tightly integrated with Lumberyard's component entity system and built on the `AzCore` library. This means that you can create event-driven logic and behaviors without programming experience.

Terrain Editor



Create terrain with **Terrain Editor**.

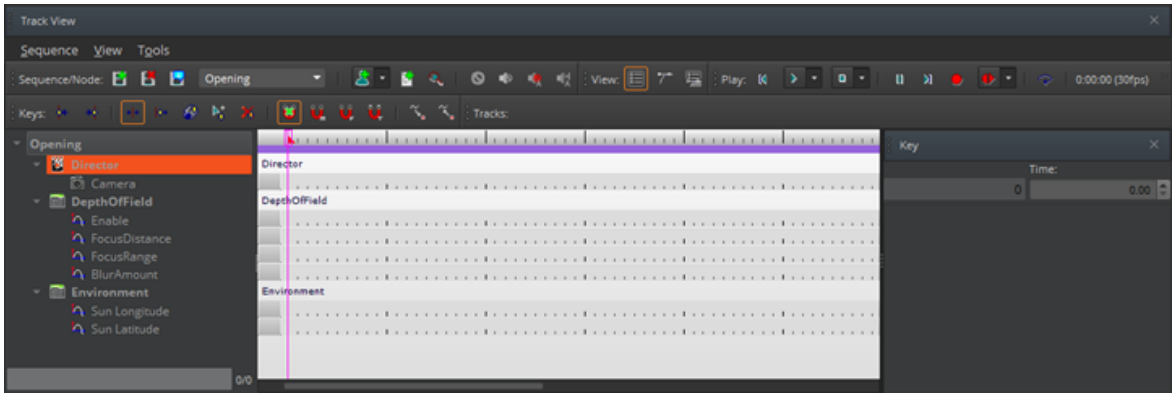
With **Terrain Editor**, you can paint heightmaps to create peaks and valleys in your terrain and add megaterrain textures for sweeping vistas.

Terrain Tool

Add fine detail to terrain with **Terrain Tool**.

With **Terrain Tool**, you can sculpt accurate details on your terrain and paint texture layers live in the **Perspective** viewport.

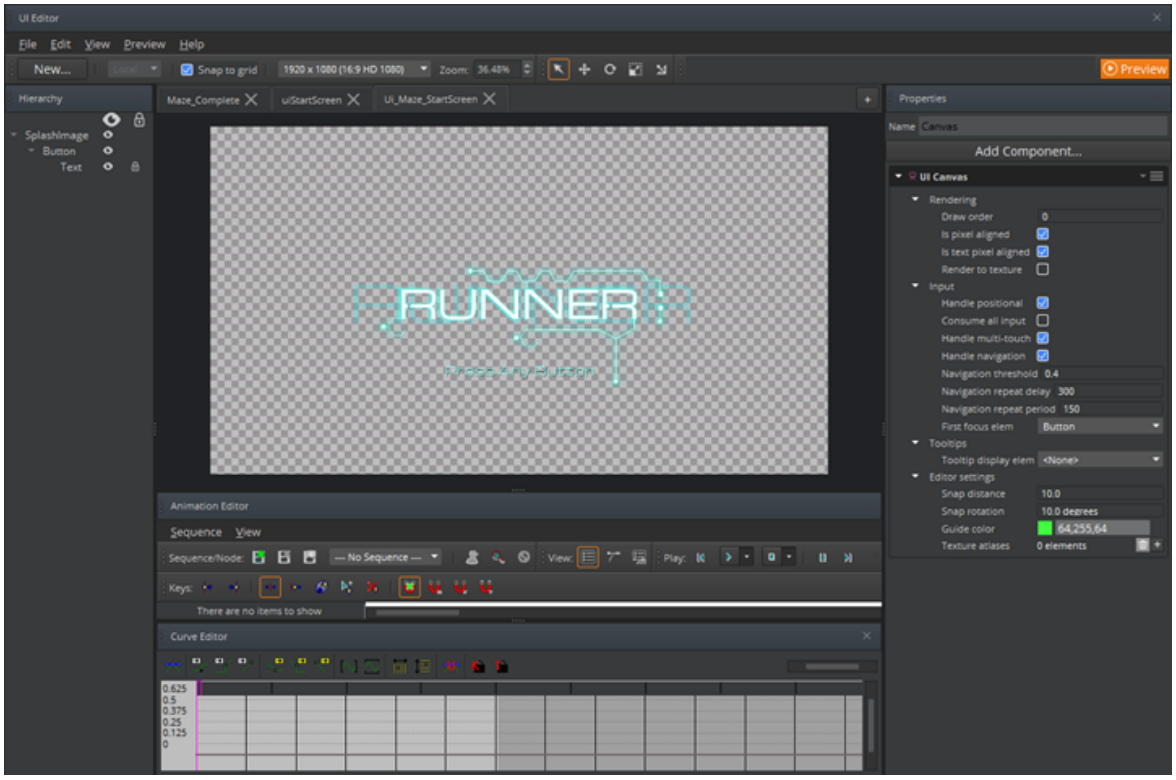
Track View



Create cinematic sequences with **Track View**.

Track View is the primary tool to create and manage cinematic sequences like cuscenes or scripted animation events.

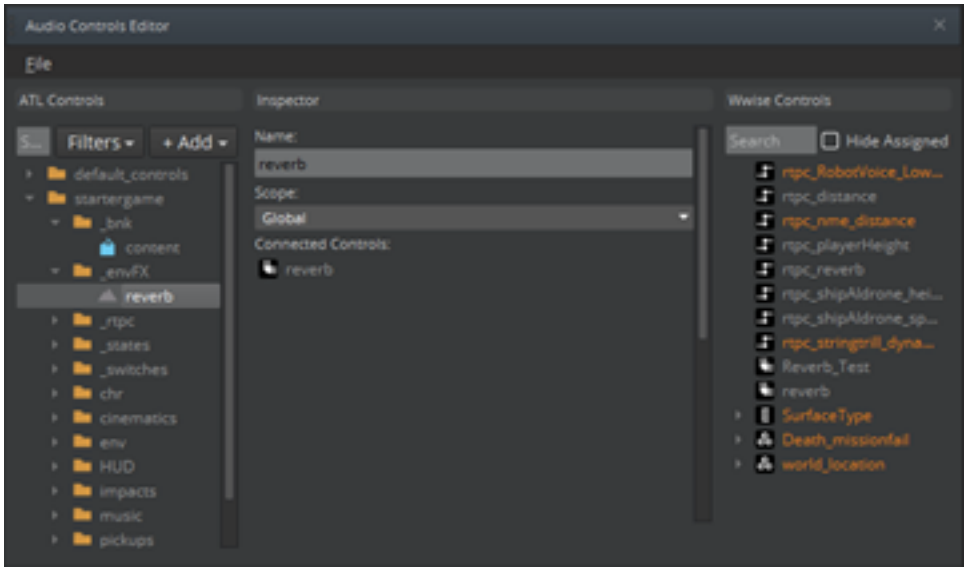
UI Editor



Design dynamic user interfaces with **UI Editor**.

You can use **UI Editor** to create, customize, and animate various game user interface elements and components such as menus, buttons, and heads-up displays.

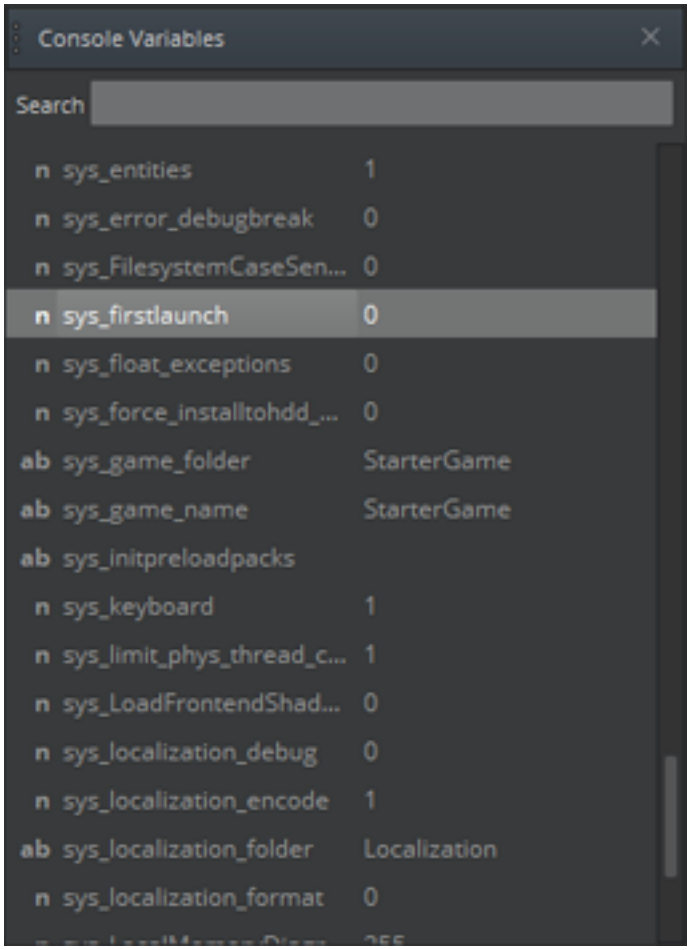
Audio Controls Editor



Map audio controls in **Audio Controls Editor**.

Your project communicates all actions, events, and parameters to the audio system with Audio Translation Layer (ATL) controls. These ATL controls are mapped to one or more controls inside your selected middleware (Wwise or Wwise LTX). With **Audio Controls Editor**, you can create controls and make connections between the ATL controls and the middleware controls.

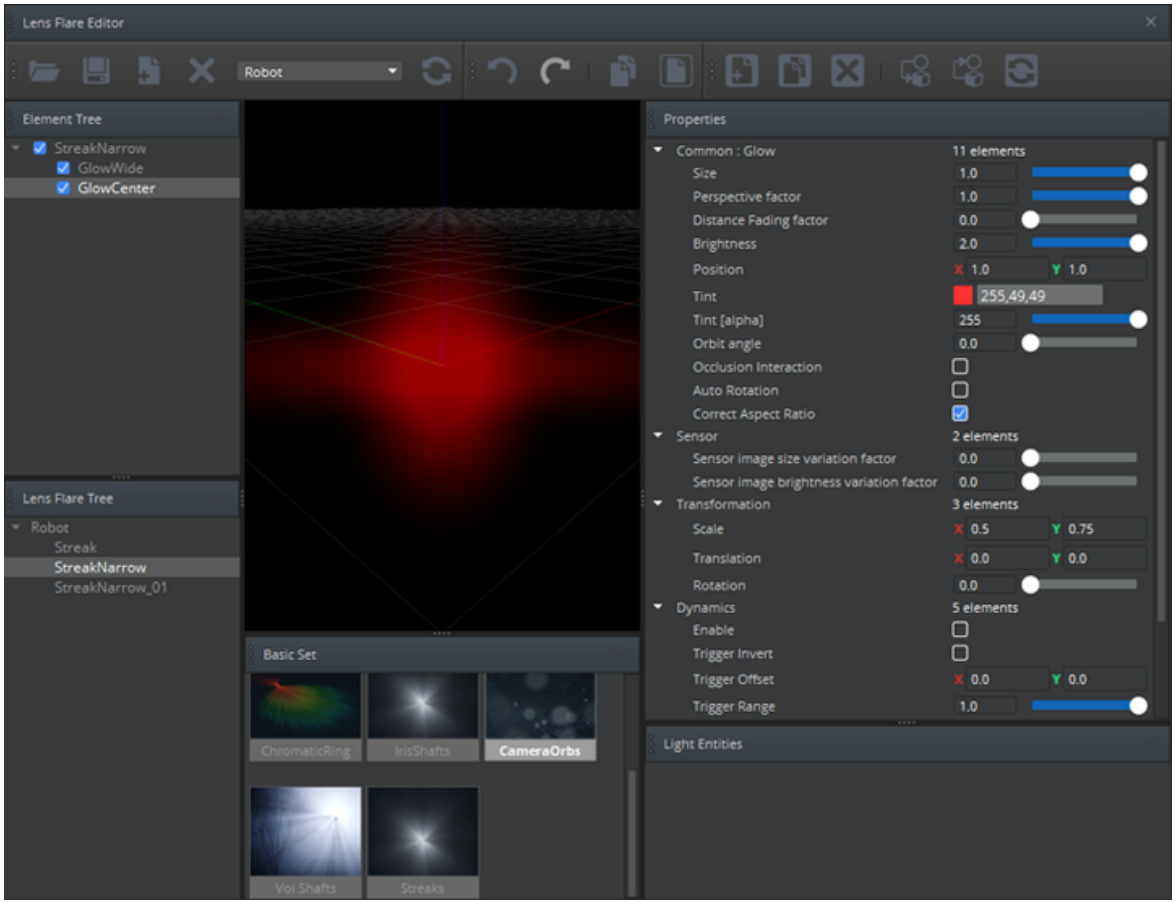
Console Variables Editor



Find and set console variables in **Console Variables Editor**.

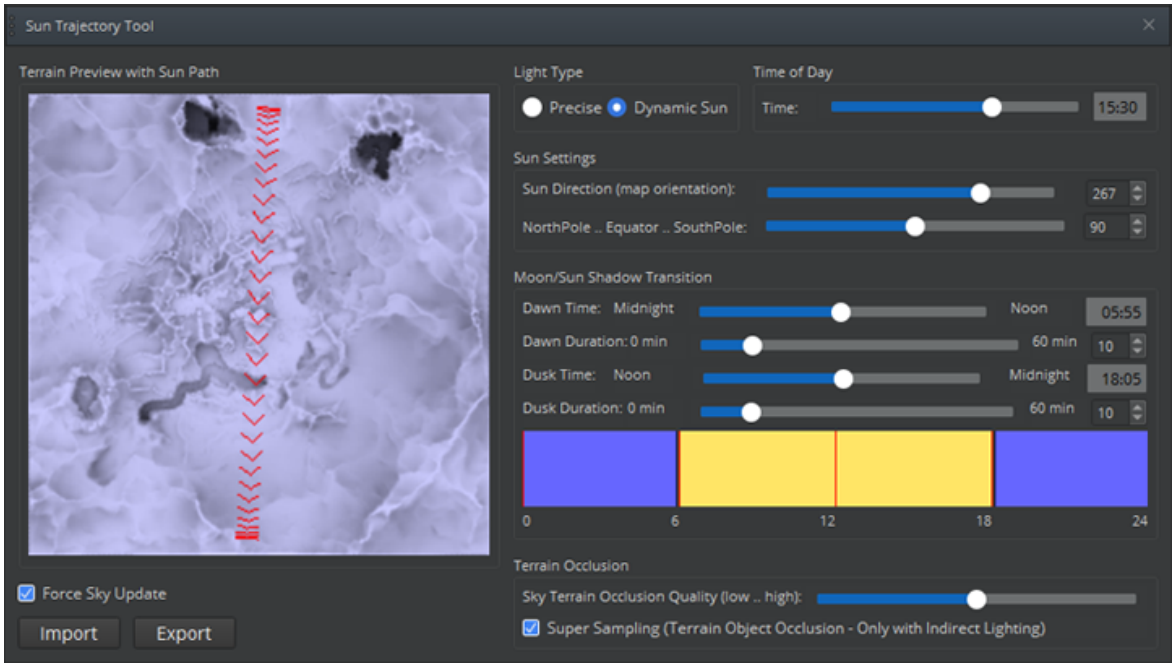
Lumberyard has many console variables, or *CVARs*, that control all aspects of the editor and your project. CVARs can enable and disable debug features, set output verbosity, modify system performance, and much more. **Console Variables Editor** presents a searchable list of available CVARs so you can modify their values.

Lens Flare Editor



Design unique lens flare effects for specific lights in **Lens Flare Editor**.

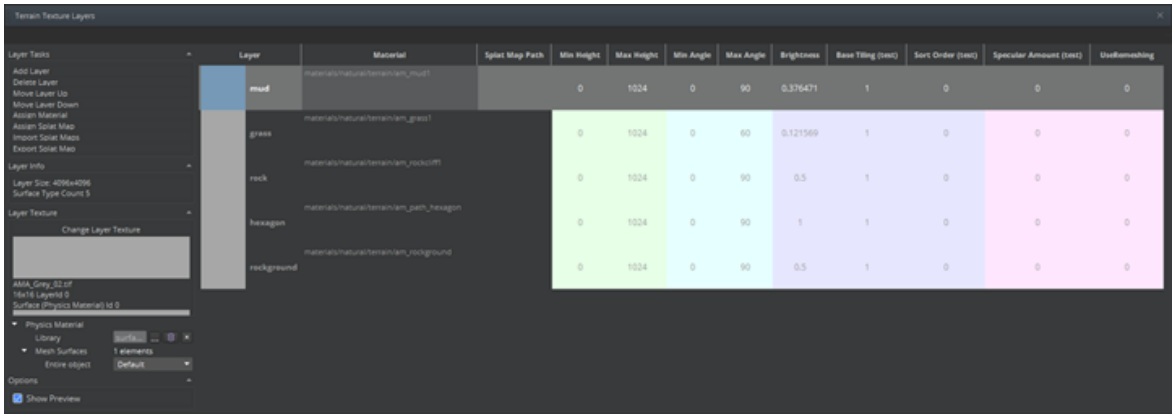
Sun Trajectory Tool



Create dynamic time-of-day lighting with the **Sun Trajectory Tool**.

With **Sun Trajectory Tool**, you define the current time, sunrise and sunset, and sun direction to create dynamic dawn, daylight, dusk, and night skies.

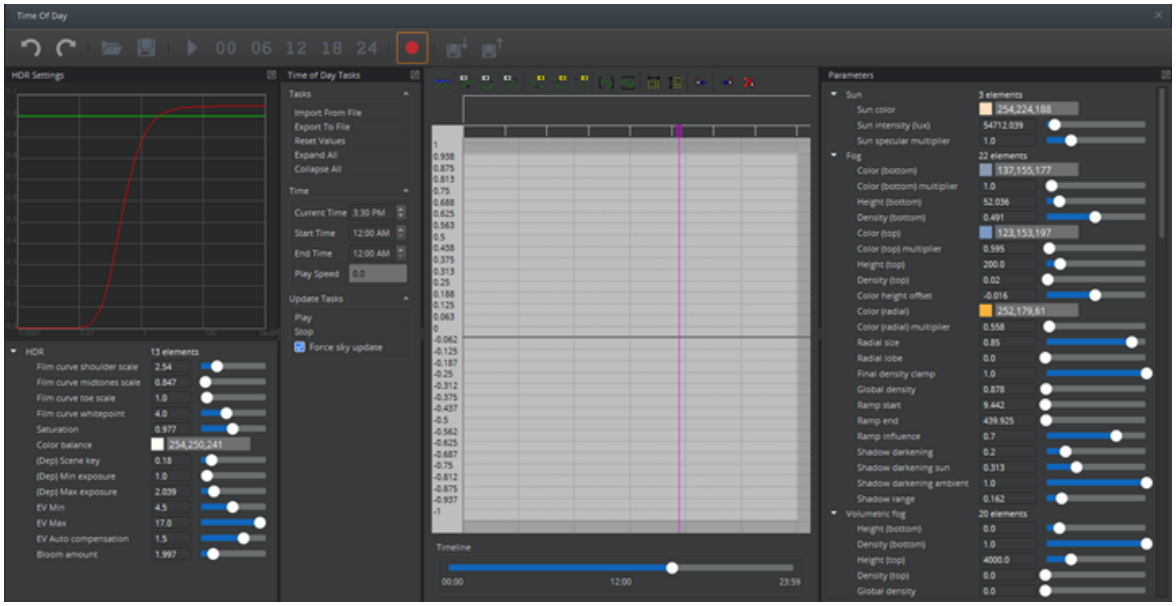
Terrain Texture Layers



Build realistic natural terrain with **Terrain Texture Layers**.

Texture layers are used to create natural-looking terrain that transitions between surface types such as sand, dirt, mud, grass, and stone. With **Terrain Texture Layers** you create and order terrain layers, and apply materials and textures that create transitions between terrain surface types.

Time of Day Editor



Create natural lighting and atmosphere transitions over time with **Time of Day Editor**.

Time of Day Editor configures changes to environment parameters over time to mimic a day-night lighting cycle. The **Time of Day Editor** uses a 24-hour time line graph and a recording function to store changing environment parameter values.

Additional Lumberyard Resources

- After setting up Amazon Lumberyard, learn how to use the platform with our Getting Started video series. This series covers all the basics of working with Lumberyard and is a terrific foundation for moving on to intermediate and advanced topics.

  Getting Started with Amazon Lumberyard
- Lumberyard also has an ever-growing YouTube channel that's loaded with additional tutorial videos and presentations. Subscribe and enable notifications so you always know when new videos are released.

  Amazon Lumberyard on YouTube
- Dive deeper into the features and usage of Lumberyard with the Amazon Lumberyard documentation.

  Amazon Lumberyard documentation home
- Join the online community in the Lumberyard forums to learn from Lumberyard staff and users, and share your projects.

  Amazon Lumberyard forums
- Lumberyard users have created an unofficial Discord. Join the conversation to share tips and get help from Lumberyard users.

  Amazon Lumberyard on Discord

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.