

Command-Line Arguments

Arguments that you can pass to an engine executable to customize how the engine runs on startup.



In Unreal Engine, **Command-line Arguments**, also called **Additional Launch Parameters**, customize how the engine runs on startup. Similar to [console commands](#), command-line arguments can be an invaluable tool for testing and optimizing your project. These settings range from high-level operations, such as forcing the **Unreal Editor** to run in game mode instead of full-editor mode, to more detailed options, such as choosing a specific map to run within your game at a particular resolution and framerate.

Pass Command-Line Arguments


There are three common methods to pass command-line arguments to your Unreal Engine project or executable. These methods correspond to different ways of running your project:

- [From the command-line.](#)
- [From the Unreal Editor.](#)
- [From an executable shortcut.](#)

From the Command-Line

The general syntax for adding command-line arguments to an executable run from the command-line is:

```
<EXECUTABLE> [URL_PARAMETERS] [ARGUMENTS]
```

 Copy full snippet

where:

- `EXECUTABLE` is the name of your executable file.
 - Examples: `UnrealEditor.exe`, `MyGame.exe`
- `URL_PARAMETERS` are any optional [URL parameters](#).
 - Examples: `MyMap`, `/Game/Maps/BonusMaps/BonusMap.umap?game=MyGameMode`
- `ARGUMENTS` include additional, optional command-line [flags](#) or [key-value pairs](#).
 - Examples: `-log`, `-game`, `-windowed`, `-ResX=400 -ResY=620`

For example, the following input runs the `MyGame` project on the `BonusMap` in the `MyGameMode` game mode fullscreen on Windows:

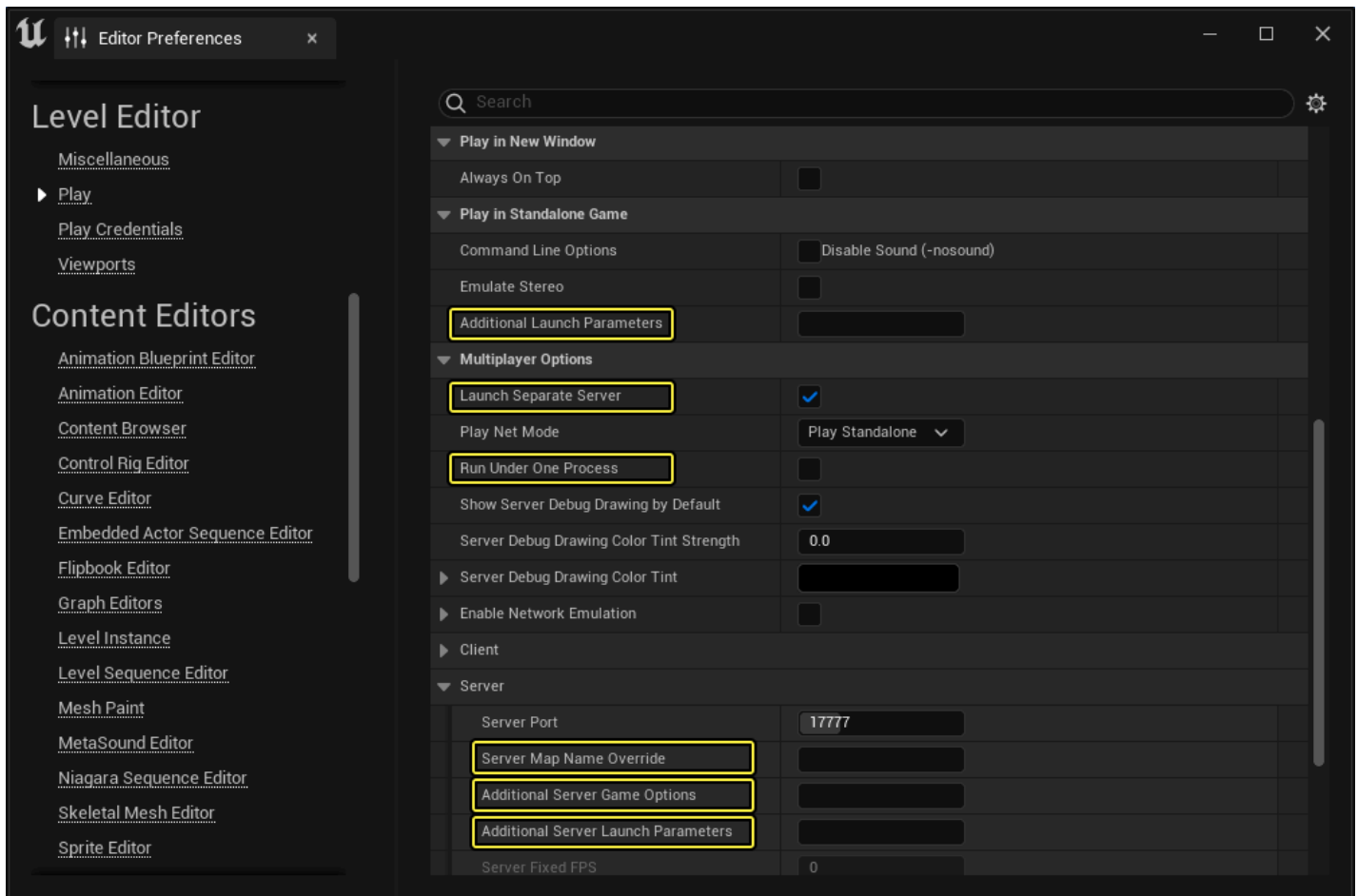
```
UnrealEditor.exe MyGame.uproject /Game/Maps/BonusMaps/BonusMap.umap?game=MyGameM
```

 Copy full snippet

From the Editor

The Unreal Editor supports customizing standalone games with command-line arguments. In the Unreal Editor, command-line arguments are referred to as *additional launch parameters*. Additional launch parameters are only supported for the **Play in Standalone Game** mode. The Unreal Editor also supports command-line arguments passed specifically to a separate, dedicated server for testing multiplayer games. Command-line arguments for a separate server are passed in three different places within the Unreal Editor:

- Server Map Name Override: this is where you can pass map name as a URL parameter.
- Additional Server Game Options: this is where you can pass additional URL parameters.
- Additional Server Launch Parameters: this is where you can pass any other additional command-line flags or key-value pairs.



Customize additional launch parameters within the Unreal Editor in the Editor Preferences.

Game Arguments

To pass command-line arguments to a standalone game launched from within the Unreal Editor, follow these steps:

1. Navigate to **Edit > Editor Preferences**. A new window pops up with a tab titled **Editor Preferences**.
2. On the left-hand side, select **Level Editor > Play**.
3. On the right-hand side, find the section titled **Play in Standalone Game**.
4. In this section, there is a textbox for **Additional Launch Parameters**. Paste your command-line arguments here. These additional parameters are then passed to the standalone game as command-line arguments.

Server Arguments

If you have checked **Launch Separate Server** and disabled **Run Under One Process**, you can specify the **Server Map Name Override**, **Additional Server Game Options**, and **Additional Server Launch Parameters**. To pass command-line arguments to a server launched from within the Unreal Editor, follow these steps:

1. Navigate to **Edit > Editor Preferences**. A new window pops up with a tab titled **Editor Preferences**.
2. On the left-hand side, select **Level Editor > Play**.
3. On the right-hand side, find the section titled **Multplayer Options**.
4. If you have not already done so, do the following from within this section:
 - a. Enable **Launch Separate Server**.
 - b. Disable **Run Under One Process**.
5. Navigate to **Multplayer Options > Server**.
6. In this section, there are three text boxes where you can specify different types of command-line arguments for your dedicated server: **Server Map Name Override**, **Additional Server Game Options**, and **Additional Server Launch Parameters**.
 - a. Use the **Server Map Name Override** textbox to pass a [map name](#) as a [URL parameter](#).
 - b. Use the **Additional Server Game Options** textbox to pass [additional URL parameters](#).
 - c. Use the **Additional Server Launch Parameters** textbox to pass any additional command-line arguments.



Additional server launch parameters are only available if you choose to **Launch Separate Server** and disable **Run Under One Process**. When Run Under One Process is disabled, your clients run slower because each client spawns a separate instance of the Unreal Editor.

From an Executable Shortcut

To pass command-line arguments to an executable shortcut, follow these steps:

1. Create a shortcut to your executable.
2. Right-click on the shortcut and select **Properties**.
3. Under the **Shortcut** section, add your command-line arguments to the end of the **Target** field.

4. When you run this shortcut, the command-line arguments are passed to the original executable that the shortcut points to.

Command-Line Arguments on Non-Desktop Platforms

To pass command-line arguments to non-desktop platforms such as consoles, mobile, and extended reality (XR); you can set the command-line by creating or editing a file titled

`UECommandLine.txt`. UE automatically reads in command-line arguments from `UECommandLine.txt` upon launch. If this file does not already exist, create the file in your project's root directory and add your command-line arguments.

Create Your Own Command-Line Arguments

Unreal Engine provides some helpful C++ functions for parsing the command-line. You can create your own command-line arguments by passing your desired flag or key-value pair to the command-line as you would any other command-line argument. To use command-line arguments that you have passed, you need to read them from the command-line within your code. If your project code does not read your custom command-line arguments and parse them, the arguments are ignored.

Flags

Flags are switches that turn a setting on or off by their presence on the command-line. For example:

```
UnrealEditor.exe MyGame.uproject -game
```

 Copy full snippet

In this example, the `-game` argument is a flag because its presence on the command-line tells the Unreal Editor executable that you want to run `MyGame` in the game mode.

Parse Flags

To parse a flag from the command-line, use the `FParse::Param` function.


For example, suppose that you want to pass a boolean flag, `-myflag`, to your executable through the command-line as:

```
UnrealEditor.exe MyGame.uproject -myflag
```

 Copy full snippet

You can check whether this flag is present with the following code in your project:

```
1 bool bMyFlag = false;
2 if (FParse::Param(FCommandLine::Get(), TEXT("myflag")))
3 {
4     bMyFlag = true;
5 }
```

 Copy full snippet

If `-myflag` is present on the command-line, the value of `bMyFlag` is `true`. If `-myflag` is not present on the command-line, the value of `bMyFlag` is `false`.

Key-Value Pairs

Key-value pairs are settings switches that specify a particular value for a switch. In addition to the presence of the switch, a setting must be provided for the switch. For example, in the following example:

```
UnrealEditor.exe MyGame.uproject -game -windowed -ResX=1080 -ResY=1920
```

 Copy full snippet

The `-ResX=1080` and `-ResY=1920` arguments are key-value pairs because each switch must be accompanied by a setting. In particular, these key-value pairs instruct the Unreal Editor executable to run at a particular resolution.

Parse Key-Value Pairs

To parse a key-value pair, use the `FParse::Value` function.

For example, suppose that you want to pass a key-value pair `-mykey=42` to your executable through the command-line:

```
UnrealEditor.exe MyGame.uproject -mykey=42
```

 Copy full snippet

You can parse this key-value pair with the following code:

```
1 int32 myKeyValue;  
2 if (FParse::Value(FCommandLine::Get(), TEXT("mykey="), myKeyValue))  
3 {  
4     // if the program enters this "if" statement, mykey was present on the  
4     command-line  
5     // myKeyValue now contains the value passed through the command-line  
6 }
```

 Copy full snippet

If `-mykey=42` is present on the command-line, the value of `myKeyValue` is `42`. If `-mykey=42` is not present on the command-line, the value of `myKeyValue` is not set.



You can find more information about what functions are available to interact with the command-line in `CommandLine.h` located in `Engine\Source\Runtime\Core\Public\Misc`.

Customize Engine Configuration from the Command-Line

Engine configuration is normally set in engine configuration `.ini` files. You can also customize engine configuration from the command-line. See the [Configuration Files](#) documentation for more information.

Customize Console Commands from the Command-Line

Console commands are normally executed from the console in the Unreal Editor. You can also customize console commands from the command-line. See the [Console Variables](#) documentation for more information.

Command-Line Arguments Reference

URL Parameters

URL parameters force your game to load a specific map upon startup. URL parameters are optional, but if you do provide them, they must immediately follow the executable name or any mode flag if one is present.

URL parameters consist of two parts:

- A [map name](#) or [server IP address](#).
- A series of optional [additional parameters](#).

Map Name

A map name can refer to any map located within the Maps directory. You can optionally include the `.umap` file extension. To load a map not found in the Maps directory, you must use an absolute path or a path relative to the maps directory. In this case, the `.umap` file extension is required.

Server IP Address

You can use a server IP address as a URL parameter to connect a game client to a dedicated server.

Additional Parameters

You can specify additional parameter options by appending them to the map name or the server IP address. Each option is prefaced by a ? (question mark) and set with = (equality or assignment). Prepending an option with - (dash) removes that option from the cached URL options.

Examples

Open Game with Map Located in Maps Directory

```
MyGame.exe MyMap
```

 Copy full snippet

Open Game with Map Located Outside Maps Directory

```
MyGame.exe /Game/Maps/BonusMaps/BonusMap.umap
```

 Copy full snippet

Open Game in Unreal Editor with Map Located Outside Maps Directory

```
UnrealEditor.exe MyGame.uproject /Game/Maps/BonusMaps/BonusMap.umap?game=MyGameM
```

 Copy full snippet

Connect a Game Client to a Dedicated Server

Supposing that you have a project, `MyGame`, that you want to run a server with the map `BonusMap.umap` located in `/Game/Maps/BonusMaps/`

You can run your dedicated server and connect your client locally as:

```
1 UnrealEditor.exe MyGame.uproject /Game/Maps/BonusMaps/BonusMap.umap -server -  
  port=7777 -log
```

```
2 UnrealEditor.exe MyGame.uproject 127.0.0.1:7777 -game -log
```

 Copy full snippet

The arguments in this example are:

- Server
 - `MyGame.uproject`: Run the `MyGame` project.
 - `/Game/Maps/BonusMaps/BonusMap.umap`: Open the `BonusMap` in your game. This is a [URL parameter](#).
 - `-server`: Run the editor as a dedicated server.
 - `-port=7777`: Use port `7777` to listen for client connections. This is the default port for servers in Unreal Engine.
 - `-log`: Display the server log so that you can monitor server activity.
- Game Client
 - `MyGame.uproject`: Run the `MyGame` project.
 - `127.0.0.1:7777`: Connect to IP address `127.0.0.1` on port `7777`. This is a [server IP address](#).
 - `-game`: Run the editor as a game client.
 - `-log`: Display the game client log so that you can monitor client activity.

Flags and Key-Value Pairs

This section provides a reference of useful command-line arguments available to use when running Unreal Engine using any of the methods outlined in the section about how to [pass command-line arguments](#).

Read Command-Line Arguments from a File

You might need to use a very large number of command-line arguments or need to reuse the same set of arguments often. You can store your command-line arguments in a text file and pass this file in the command-line for convenience. This is also useful if you find yourself hitting the Windows command-line length limit during testing.

The syntax for passing a text file containing command-line arguments is:

```
<EXECUTABLE> -CmdLineFile=ABSOLUTE\PATH\TO\FILE.txt
```

 Copy full snippet

Example

If you have a file named `MyCmdLineArgs.txt` saved in the directory `D:\UnrealEngine` and you want to pass it to the `UnrealEditor.exe`, you can do so with the following command:

```
UnrealEditor.exe -CmdLineFile=D:\UnrealEngine\MyCmdLineArgs.txt
```

 Copy full snippet

List of Available Command-Line Arguments

For a list of all available command-line arguments, see the [Command-Line Arguments Reference](#).



Command-Line Arguments Reference

List of command-line arguments available for Unreal Engine.