Project

Reference for the Project section of the Unreal Engine Project Settings.

Description

This is where you set information about your project, such as the project name, version, company name, copyright, and so on. These settings are mainly for informational purposes and will not affect how the project runs or behaves.

About

Section	Description
Description	The project description text.
Project ID	The project's unique identifier.
Project Name	The project's non-localized name.
Project Version	The project's version number.
Project Thumbnail	The project's preview thumbnail.

Publisher

Section	Description
Company Name	The name of the company (author, provider) that created the project.
Company Distinguished Name	The Distinguished Name of the company (author, provider) that created the project, which is used by publishing tools on some platforms.
Homepage	The project's homepage URL.

Section	Description
Support Contact	The project's support contact information.

Legal

Section	Description
Copyright Notice	The project's copyright and / or trademark notices.
Licensing Terms	The project's licensing terms.
Privacy Policy	The project's privacy policy.

Displayed

Section	Description
Project Displayed Title	The project's title as displayed on the window title bar. Can include the tokens `{GameName}`, `{PlatformArchitecture}`, `{BuildConfiguration}`, or `{RHIName}`, which will be replaced with the specified text.
Project Debug Title Info	Additional data to be displayed on the window title bar in non-shipping (debug) configurations. Can include the tokens `{GameName}`, `{PlatformArchitecture}`, `{BuildConfiguration}`, or `{RHIName}`, which will be replaced with the specified text.

Settings

Section	Description
Should Window Preserve Aspect Ratio	Specifies whether the game's window should preserve its aspect ratio when resized by the user.
Use Borderless Window	Specifies whether the game should use a borderless Slate window instead of a window with system title bar and border.
Start in VR	Specifies whether the game should attempt to start in VR, regardless of whether `-vr` was set in the command line.
Allow Window Resize	Specifies whether the user should be allowed to resize the game window, when not using full screen.

Section	Description
Allow Close	Specifies whether a **Close** button should be shown for the game window, when not using full screen.
Allow Maximize	Specifies whether a **Maximize** button should be shown for the game's window, when not using full screen.
Allow Minimize	Specifies whether a **Minimize** button should be shown for the game's window, when not using full screen.

Encryption

Encryption

Section	Description
Encryption Key	The default encryption key used to protect `.pak` files.
Secondary Encryption Keys	Secondary encryption keys that can be selected for use on different assets. Games are required to make these keys available to the `.pak` platform file at runtime in order to access the data they protect.
Encrypt Pak Ini Files	Encrypts all `.ini` files in the `.pak`. Gives security to the most common sources of mineable information, with minimal runtime IO cost.
Encrypt Pak Index	Encrypts the `.pak` index, making it impossible to use UnrealPak to manipulate the `.pak` file without the encryption key.
Encrypt UAsset Files	Encrypts `.uasset` files.
Encrypt All Asset Files	Encrypts all files in the `.pak` file. Secure, but will cause some slowdown to runtime IO performance, and high entropy to packaged data which will be bad for patching.
Generate New Encryption Key	Generates a new encryption key.

Signing

Section	Description
Signing Public Exponent	The RSA key public exponent used for signing a `.pak` file.
Signing Modulus	The RSA key modulus used for signing a `.pak` file.
Signing Private Exponent	The RSA key private exponent used for signing a `.pak` file.
Enable Pak Signing	Enable signing of `.pak` files to prevent tampering of the data.
Generate New Signing Keys	Generate a new signing key.

Gameplay Tags

Gameplay Tags

Section	Description
Import Tags from Config	If true, will import tags from `.ini` files in the `Config/Tags` folder.
Warn on Invalid Tags	If true, will give load warnings when reading in saved tag references that are not in the dictionary.
Clear Invalid Tags	If true, will clear any invalid tags when reading in saved tag references that are not in the dictionary.
Invalid Tag Characters	These characters cannot be used in gameplay tags, in addition to special ones like `newline`.
Category Remapping	This allows base engine tag category metadata to remap to multiple project-specific categories.
Gameplay Tag Table List	List of data tables to load tags from.
Gameplay Tag Redirects	List of active tag redirects.
Gameplay Tag List	List of active tag redirects.

Advanced Replication

Section	Description
Fast Replication	If true, will replicate gameplay tags by index instead of name. For this to work, tags must be identical on client and server.
Commonly Replicated Tags	List of most frequently replicated tags.
Num Bits for Container Size	Numbers of bits to use for replicating container size. Set this based on how large your containers tend to be.
Net Index First Bit Segment	The length in bits of the first segment when net serializing tags. We serialize NetIndexFirstBitSegment + 1 bit to indicate "more", which is slower to replicate.

Advanced Gameplay Tags

Section	Description
Restricted Config Files	A list of `.ini` files used to store restricted gameplay tags.
Restricted Tag List	Restricted tags are intended to be top-level tags that are important for your data hierarchy and modified by very few people.

Maps & Modes

Default Modes

Section	Description
Default GameMode	`GameMode` to use if not specified in any other way (for example, per-map `DefaultGameMode` or in the URL). You can choose any `GameMode` class in your project.
Default Pawn Class	The default pawn class used by players. You can choose any pawn class in your project.
HUD Class	HUD class this game uses. You can choose any HUD class in your project.
Player Controller Class	The class of `PlayerController` to spawn for players logging in.

Section	Description
	You can choose any `PlayerController` class in your project.
Game State Class	The class of `GameState` associated with this `GameMode`. You can choose any `GameState` class in your project.
Player State Class	A `PlayerState` of this class will be associated with every player to replicate relevant player information to all clients.
Spectator Class	The pawn class used by the `PlayerController` for players when spectating.
Global Default Server Game Mode	`GameMode` to use if not specified in any other way (for example, per-map `DefaultGameMode` or in the URL). This setting is only used when running the engine as a dedicated server. You can choose any `GameMode` class in your project. If not set, the `GlobalDefaultGameMode` value will be used.
Game Mode Map Prefixes	Overrides the `GameMode` to use when loading a map that starts with a specific prefix.
Game Mode Class Aliases	List of `GameModes` to load when the game is specified in the URL (for example, **DM** could be an alias for **MyProject.MyGameModeMP_DM**).

Default Maps

Section	Description
Editor Startup Map	If set, this map will be loaded when the Editor starts up. You can choose any map available in your project.
Editor Template Map Overrides	Map templates that should show up in the new level dialog. This will completely override the default maps chosen by the default editor.
Game Default Map	The map that will be loaded by default when no other map is loaded. You can choose any map available in your project.

Section	Description
Local Map Options	The default options that will be appended to a map being loaded.
Transition Map	The map that is loaded when transitioning from one map to another. You can choose any map available in your project.
Server Default Map	The map that will be loaded by default when no other map is loaded. This setting is only used when running the engine as a dedicated server. You can choose any map available in your project.

Local Multiplayer

Section	Description
Use Splitscreen	Specifies whether the screen should be split or not when multiple local players are present.
Two Player Splitscreen Layout	The viewport layout to use if the screen should be split and there are two local players. You can choose from the following options: • **Horizontal** • **Vertical**
Three Player Splitscreen Layout	The viewport layout to use if the screen should be split and there are three local players. You can choose from the following options: • **Favor Top** • **Favor Bottom** • **Horizontal** • **Vertical**
Four Player Splitscreen Layout	The viewport layout to use if the screen should be split and there are four local players. You can choose from the following options: • **Grid** • **Horizontal**

Section	Description
	• **Vertical**
Skip Assigning Gamepad to Player 1	If enabled, this will make it so that gamepads start being assigned to the second controller ID in local multiplayer games. In PIE (Play in Editor) sessions with multiple windows, this has the same effect as enabling the **Route 1st Gamepad to 2nd Client** setting in Editor Preferences.

Game Instance

Section	Description
Game Instance Class	The class to use when instantiating the transient `GameInstance` class. You can choose any `GameInstance` class in your project.

Movies

Movies

Section	Description
Wait for Movies to Complete	If enabled, the game waits for startup movies to complete even if loading has finished.
Movies Are Skippable	If enabled, startup movies can be skipped by the user when a mouse button is pressed.
Startup Movies	Movies to play on startup. Note that these must be in your game's `Game/Content/Movies` directory.

Packaging

Packaging

Section	Description
Use Pak File	If enabled, all content will be put into one or more `.pak` files instead of many individual files. This

Section	Description
	is enabled by default.
Use Io Store	If enabled, use `.utoc` / `.ucas` container files for staged / packaged package data instead of `.pak`.
Use Zen Store	If enabled, use Zen storage server for storing and fetching cooked data instead of using the local file system.
Make Binary Config	If enabled, staging will make a binary config file for faster loading.
Generate Chunks	If enabled, will generate `.pak` file chunks. Assets can be assigned to chunks in the editor or via a delegate (see `ShooterGameDelegates.cpp`). Can be used for streaming installs (PS4 Playgo, XboxOne Streaming Install, etc).
Generate No Chunks	If enabled, no platform will generate chunks, regardless of settings in platform-specific `.ini` files.
Chunk Hard References Only	Normally, during chunk generation, all dependencies of a package in a chunk will be pulled into that package's chunk. If this is enabled, only hard dependencies are pulled in. Soft dependencies stay in their original chunk.
Build HTTP Chunk Install Data	If enabled, it will generate data for HTTP Chunk Installer. This data can be hosted on a web server to be installed at a runtime. Requires the **Generate Chunks** option to be enabled.
HTTP Chunk Install Data Directory	When **Build HTTP Chunk Install Data** is enabled, this is the directory where the data will be built to.
Package Compression Min Size to Consider DDC	Specifies the minimum (uncompressed) size for storing a compressed IoStore chunk in the

Section	Description
	Derived Data Cache (DDC).
HTTP Chunk Install Data Version	Version name for HTTP Chunk Install Data.
Share Material Shader Code	By default, shader code gets saved inline inside Material Assets. Enabling this option will store only shader code once as individual files. This will reduce overall package size, but might increase loading time.
Deterministic Shader Code Order	If this option is disabled, the shader code will be stored in the library essentially in a random order, squarely the same in which the assets were loaded by the cooker. Enabling this option will sort the shaders by their hash, which makes the shader library more similar between builds. This can help patching, but can adversely affect loading times.
Shared Material Native Libraries	By default, shader code gets saved into individual platform-agnostic files. Enabling this option will use the platform-specific library format only if one is available. This will reduce overall package size but might increase loading time.
Ini Key Denylist	List of the `.ini` file keys to strip when packaging.
Ini Section Denylist	List of `.ini` file sections to strip when packaging.
Additional Builds for This Project	A list of custom builds that will show up in the Platforms menu to allow customized builds that make sense for your project. Will show up near Package Project in the Platforms menu (in the main Level Editor toolbar).
Force One Chunk Per File	If true, individual files are only allowed to be in a single chunk and it will assign it to the lowest number requested. If false, files may end up in multiple chunks if requested by the cooker.
Max Chunk Size	If greater than 0, this sets a maximum size per chunk.

Section	Description
	Chunks larger than this size will be split into multiple `.pak` files such as `pakchunk0_s1`. This can be set in platform-specific game `.ini` files.
Create Compressed Cooked Packages	Create compressed cooked packages (decreased deployment size).
Package Compression Format	A comma-separated list of formats to use for `.pak` file and IoStore compression. If more than one format is specified, the list is in order of priority, with fallbacks to other formats in case of errors or unavailability of the format (for example, if the format's plugin is not enabled). Commonly `PackageCompressionFormat=Oodle` or `PackageCompressionFormat=None`.
Use This Compression Format Not Hardware Override	If enabled, forces the use of the specified **Package Compression Format**. This option overrides any platform-specific values set by the `HardwareCompressionFormat` parameter in `DataDrivenPlatformInfo.ini`.
Package Compression Command Line Options	A generic setting for allowing a project to control compression settings during `.pak` file and IoStore compression. For example: `PackageAdditionalCompressionOptions=-compressionblocksize=1MB -asynccompression`.
Package Compression Method	For compressors with multiple methods, select one. For example, for Oodle, you may use one of these: • **Kraken** • **Mermaid** • **Selkie** • **Leviathan** You can read the description of every method here.
Encoder Effort Level for Debug & Development	For compressors with variable levels, select the encoder level, which makes packages smaller but takes more time to encode. This does not affect decode speed.

Section	Description
	For faster iteration, use lower effort levels (for example, `1`).
Encoder Effort Level for Test & Shipping	Package compression level test shipping.
Encoder Effort Level for Distribution	Package compression level distribution.
Minimum Amount of Bytes Which Should Be Saved When Compressing a Block of Data, Otherwise Data Remains Uncompressed	A generic setting which is used to determine whether it is worth using compression for a block of data when creating IoStore or `.pak` files. If the amount of saved bytes is smaller than the specified value, then the block of data remains uncompressed. The optimal value of this setting depends on the capabilities of the target platform. For example, `PackageCompressionMinBytesSaved=1024`. Note that some compressors (for example, Oodle) do their own internal worth it check and only use this value to determine the minimal size of a block which should be compressed.
Minimum Percentage of a Block of Data Which Should Be Saved When Performing Compression, Otherwise Data Remains Uncompressed	A generic setting which is used to determine whether it is worth using compression for a block of data when creating IoStore or `.pak` files. If the saved percentage of a compressed block data is smaller than the specified value, then the block remains uncompressed. The optimal value of this setting depends on the capabilities of the target platform. For example, `PackageCompressionMinPercentSaved=5`. Note that some compressors (for example, Oodle) do their own internal worth it check and ignore this value.
Enable DDC for IoStore Compression	Specifies whether the Derived Data Cache should be used to store and retrieve compressed data when creating IoStore containers.
Include Crash Reporter	Specifies whether to include the crash reporter in the packaged project. This is included by default for Blueprint-based projects, but can optionally be disabled.
Internationalization Support	Predefined sets of cultures whose internationalization data should be packaged.

Section	Description
	You can choose from the following options: • **English** • **EFIGS**: English, French, Italian, German, Spanish • **EFIGSCJK**: English, French, Italian, German, Spanish, Chinese, Japanese, Korean • **CJK**: Chinese, Japanese, Korean • **All**
Localization Targets to Chunk	List of localization targets that should be chunked during cooking (if using chunks).
Localization Target Catch All Chunk ID	The chunk ID that should be used as the catch-all chunk for any non-asset localized strings.
Cook Everything in the Project Content Directory (Ignore List of Maps in the "List of Maps to Include in a Packaged Build" setting)	Cook all things in the project content directory.
Cook Only Maps (This Only Affects Cookall)	Cook only maps (this only affects the Cookall flag). For more information, see the Content Cooking page.
Exclude Editor Content When Cooking	Don't include content in any editor folders (anything under `Engine/Content/Editor*`) when cooking. This can cause issues with missing content in cooked games if content that the game uses is being excluded.
Exclude Movie Files When Staging (Skip Movies)	This setting tells the Editor to not include movies by default when staging or packaging. If this setting (`bSkipMovies` in the `.ini` file) is true, all movie files in `Engine\Content\Movies` and `<ProjectRoot>\Content\Movies` are skipped, and the movies listed in `UFSMovies` and `Non-UFSMovies` are staged or packaged. If this setting (`bSkipMovies` in the `.ini` file) is false, all movie files in `Engine\Content\Movies` and `<ProjectRoot>\Content\Movies` are staged or packaged.

Section	Description
Specific Movies to Package (UFSMovies)	If `bSkipMovies` is true, these specific movies will still be added to the `.pak` file (if using a `.pak` file; otherwise, they're copied as individual files). The name of the movie file should have no extension in the listing: if the filename of your movie is, for example, `Level2CinematicScene.mp4`, you should add `Level2CinematicScene` to this array. `UFSMovies` and `Non-UFSMovies` fields are read from platform `.ini` files instead of only being read from the editor `.ini` files.
Specific Movies to Copy (Non-UFSMovies)	If `SkipMovies` is true, these specific movies will be copied when packaging your project, but are not supposed to be part of the `.pak` file. The name of the movie file should have no extension in the listing: if the filename of your movie is, for example, `Level2CinematicScene.mp4`, you should add `Level2CinematicScene` to this array. `UFSMovies` and `Non-UFSMovies` fields are read from platform `.ini` files instead of only being read from the editor `.ini` files.
Compressed Chunk Wildcard	If set, only these specific `.pak` files will be compressed. This should take the form of `*pakchunk0*`. This can be set in a platform-specific `.ini` file.
List of Maps to Include in a Packaged Build	List of maps to include when no other map list is specified in the command line.
Additional Asset Directories to Cook	Directories containing `.uasset` files that should always be cooked regardless of whether they're referenced by anything in your project. These paths are stored either as a full package path (for example, `/Game/Folder`,

Section	Description
	`/Engine/Folder`, `/PluginName/Folder`) or as a relative package path from `/Game`.
Directories to Never Cook	Directories containing `.uasset` files that should never be cooked, even if they are referenced by your project. These paths are stored either as a full package path (for example, `/Game/Folder`, `/Engine/Folder`, `/PluginName/Folder`) or as a relative package path from `/Game`.
Test Directories to Not Search	Directories containing `.uasset` files that are for editor testing purposes and should not be included in enumerations of all packages in a root directory because they will cause errors on load. These paths are stored either as a full package path (for example, `/Game/Folder`, `/Engine/Folder`, `/PluginName/Folder`) or as a relative package path from `/Game`.
Additional Non-Asset Directories to Package	Directories containing files that should always be added to the `.pak` file (if using a `.pak` file; otherwise they're copied as individual files). This is used to stage additional files that you manually load via the UFS (Unreal File System) file IO API. These paths are relative to your project's `Content` directory.
Additional Non-Asset Directories to Copy	Directories containing files that should always be copied when packaging your project, but are not supposed to be part of the `.pak` file. This is used to stage additional files that you manually load without using the UFS (Unreal Files System) file IO API, for example, third-party libraries that perform their own internal file IO. These paths are relative to your project's `Content` directory.

Section	Description
Additional Non-Asset Directories to Package for Dedicated Server Only	Directories containing files that should always be added to the `.pak` file for a dedicated server (if using a `.pak` file; otherwise they're copied as individual files). This is used to stage additional files that you manually load via the UFS (Unreal File System) file IO API. These paths are relative to your project's `Content` directory.
Additional Non-Asset Directories to Copy for Dedicated Server Only	Directories containing files that should always be copied when packaging your project for a dedicated server, but are not supposed to be part of the `.pak` file. This is used to stage additional files that you manually load without using the UFS (Unreal Files System) file IO API, for example, third-party libraries that perform their own internal file IO. These paths are relative to your project's `Content` directory.
Localizations to Package	Cultures whose data should be cooked, staged, and packaged.

Project

Section	Description
Build	Specifies whether to build the game executable during packaging. You can choose from the following options: • **Always** • **Never** • **If Project Has Code, or Running a Locally Built Editor** • **If Running a Locally Built Editor**
Build Configuration	The build configuration for which the project is packaged.

Section	Description
	You can choose from the following options: • **Debug** • **DebugGame** • **Development** • **Test** • **Shipping**
Build Target	Name of the target to build.
Staging Directory	The directory to which the packaged project will be copied.
Full Rebuild	If enabled, a full rebuild will be enforced each time the project is being packaged. If disabled, only modified files will be built, which can improve iteration time. Unless you iterate on packaging, we recommend full rebuilds when packaging.
For Distribution	If enabled, a distribution build will be created using the shipping configuration. If disabled, a development build will be created. Distribution builds are for publishing to the App Store.
Include Debug Files in Shipping Builds	If enabled, debug files will be included in staged shipping builds.

Prerequisites

Section	Description
Include Prerequisites Installer	Specifies whether to include an installer for prerequisites of packaged games, such as redistributable operating system components, on platforms that support it.
Include App-Local Prerequisites	Specifies whether to include prerequisites alongside the game executable.
App-Local Prerequisites Directory	A directory containing additional prerequisite packages that should be staged in the executable directory. Can be relative to either the `Engine` folder within your Unreal Engine installation directory or the folder where your

Section	Description
	`.uproject` file is located.

Supported Platforms

Here, you can select the supported platforms for your project. Attempting to package, run, or cook your project on an unsupported platform will result in a warning.

You can choose one or more of the following options:

- **All Platforms**
- **Android**
- **IOS**
- **Linux**
- **LinuxARM64**
- **TVOS**
- **Windows**

Target Hardware

Target Hardware

Section	Description

Optimize Project Settings for	Select the hardware class and a graphical level.
	
	You can choose from the following options:
	• Class of hardware to target:
	  ◦ **Desktop:** Desktop or console
	  ◦ **Mobile:** Mobile or tablet
	• Graphical level to target:
	  ◦ **Maximum:** High-end features default to enabled.
	  ◦ **Scalable:** Some features are disabled by default but can be enabled based on the actual hardware.

Pending Changes

This section will show all pending changes to your project's settings.