User Guide

Ultimate DLC Toolkit

"A complete DLC authoring and management tool for Unity"

Trivial Interactive

Version 1.2.x

"Ultimate DLC Toolkit is a complete downloadable content creation and management package, designed to make DLC creation and loading as simple as possible".

Recommended Uses

- Additional downloadable content (Free and Paid).
- Expansion packs or add-ons for additional game content/themes.
- Additional character skins or packs.
- Additional map/scene/level packs.
- Additional game modes or gameplay (Scripting requires Desktop/Mono platforms to add new code as part of DLC).
- In game shop/store for additional game items.
- Better project organisation – Separate asset(s) into separate DLC packs to be loaded and unloaded by the game on demand. `Ship With Game` makes this sort of management and distribution trivial.
- Many more uses…

Features

- Quick and easy to setup / integrate into existing projects.
- Powerful and feature complete build pipeline for creating DLC content with full API available for custom / headless build support.
- Extensive and intuitive runtime APIs for loading, discovering, validating and evaluating DLC content files on demand.
- Simple but powerful API's for editor (Build pipeline) and runtime.
- Highly versatile with support for any types of DLC content and structure depending on project requirements.
- One click build for project DLC's to generate content for all platforms and configurations with preset options.

- DRM (Digital Rights Management) included for Steamworks and google play for ownership/install verification.
- DLC signing means that DLC content will only be loadable by your game project.
- Simple DLC creation workflow (Create DLC Profile -> Author DLC Assets -> Build DLC For Platforms).
- Platform specific folders allow you to include assets in the DLC only for certain platforms.
- Growing collection of concise and useful code samples for API usage on github. Need an additional code sample? Just let us know and we can add that ☑.
- Comprehensive .pdf documentation of the API (Runtime and Editor) for quick and easy reference.
- Fully commented C# source code included for runtime, editor and build pipeline code.

Additional Details

Built in DRM support for the following services:

Steamworks.Net | Facepunch.Steamworks | Google Play Asset Delivery | Local folder | Unity Editor DRM Simulator | Custom DRM Provider

Digital Rights Management - The above providers are included by default for validating ownership and installation of DLC content. Other providers can be added manually or by feature request.

☑ Fully commented C# source code included for both runtime and editor, including build pipeline.
☑ Script debugging support for windows platform (Mono).
☑ Long term support from a trusted publisher.
☑ Render pipeline independent - DLC content will be build using materials from your installed pipeline.
☑ Supports Unity 2022 LTS and newer.

Platforms

Windows | OSX | Linux | Android | IOS | WebGL | Console | Unity Editor (All Platforms)

Console *(Theoretical Support Only) – No built in DRM and platforms are untested due to no access to dev kits. However, there are no foreseen issues or requirements that should prevent the asset from working on console platforms.*

Scripting - Including new script content (Not available in base game) is supported on the platforms (Window, Mac, Linux) requiring Mono backend.

Getting started

To get up and running as quick as possible check out the first topic `Create New DLC`, which will walk you through the process of creating your first DLC content in the editor.

Contents

Quick Start

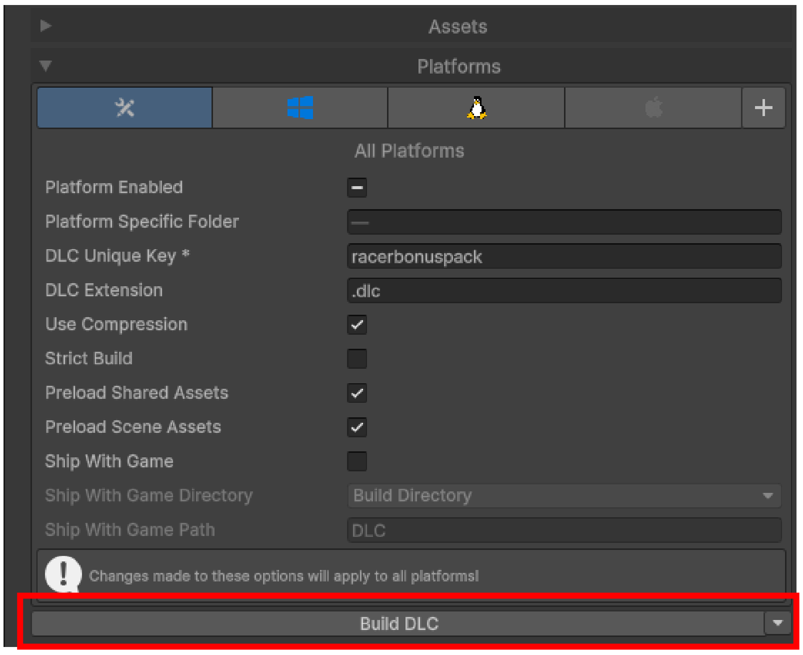
This section will show you how to setup and use the included demo racing game, which uses DLC toolkit to add additional content such as new cars and car appearances.

Build Demo DLC

The first step is to build the included demo DLC content so that it can be used or tested by the example game. To do this you can first find and select the `Racer Bonus Pack` DLC content in the project:

`Assets/Ultimate DLC Toolkit/Demo/Example DLC/Racer Bonus Pack.asset`

Once you have selected this DLC Profile asset, you can then build the DLC by simply clicking the `Build DLC` button towards the bottom of the inspector window:



Platform Enabled: –
Platform Specific Folder: —
DLC Unique Key *: racerbonuspack
DLC Extension: .dlc
Use Compression: ✓
Strict Build:
Preload Shared Assets: ✓
Preload Scene Assets: ✓
Ship With Game:
Ship With Game Directory: Build Directory
Ship With Game Path: DLC

Changes made to these options will apply to all platforms!

Note: *Building the DLC will usually take a few minutes for the first time, but subsequent builds will be much faster thanks to incremental build support.*

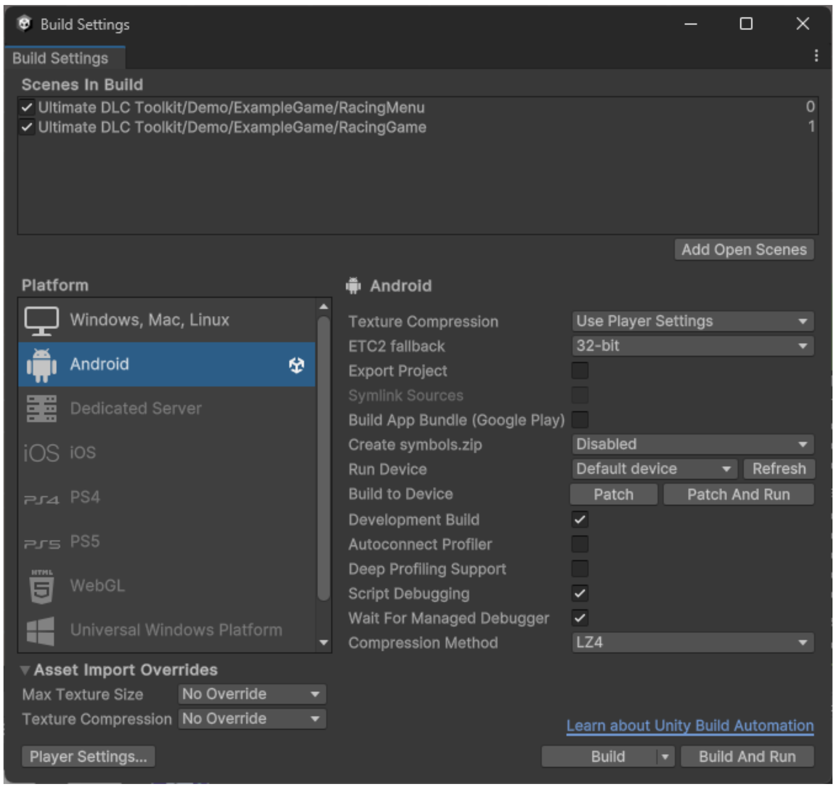
Setup Demo Scenes

The next step is to add the demo scenes listed below to the Unity build settings so that the demo game can function correctly. Open the Unity build settings window:

`File -> Build Settings`

Add the following Unity scenes to the build settings which are included in the Ultimate DLC Toolkit demo folder:

`Assets/Ultimate DLC Toolkit/Demo/ExampleGame/RacingMenu.unity`
`Assets/Ultimate DLC Toolkit/Demo/ExampleGame/RacingGame.unity`



Build Settings

Scenes In Build

- ✓ Ultimate DLC Toolkit/Demo/ExampleGame/RacingMenu 0
- ✓ Ultimate DLC Toolkit/Demo/ExampleGame/RacingGame 1

Add Open Scenes

Platform

- Windows, Mac, Linux
- Android
- Dedicated Server
- iOS
- PS4
- PS5
- WebGL
- Universal Windows Platform

Android

Texture Compression	Use Player Settings
ETC2 fallback	32-bit
Export Project	
Symlink Sources	
Build App Bundle (Google Play)	
Create symbols.zip	Disabled
Run Device	Default device / Refresh
Build to Device	Patch / Patch And Run
Development Build	✓
Autoconnect Profiler	
Deep Profiling Support	
Script Debugging	✓
Wait For Managed Debugger	✓
Compression Method	LZ4

Asset Import Overrides

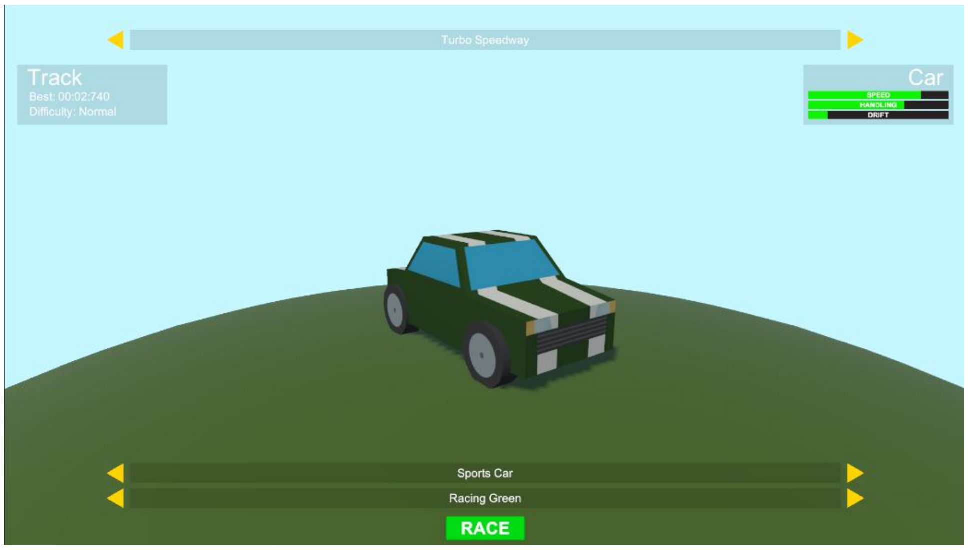
Max Texture Size	No Override
Texture Compression	No Override

Learn about Unity Build Automation

Player Settings... Build Build And Run

Run Demo Game

Finally, it is now possible to run the demo game. First you can open the `RacingMenu` scene in the Unity editor, and then simply hit the play button to enter play mode. Upon entering play mode, you will be presented with a car and track selection screen where you can choose a car from the base game, and also a car that is included as part of the DLC (All DLC content will be displayed with `(DLC)` tag at the end of the name). Make your selection and you can then click the `Race` button to start the game scene:



Congratulations – You just built and loaded your first DLC content!

Continue to the next section to begin creating your very own content from scratch

Create New DLC

The first thing you will want to do when you are authoring new DLC content is to create a new DLC Profile, and a folder where the associated DLC assets should be placed. To make this process as simple as possible, we have created a wizard which will step though all the necessary steps.

Open the create DLC wizard window via the menu:

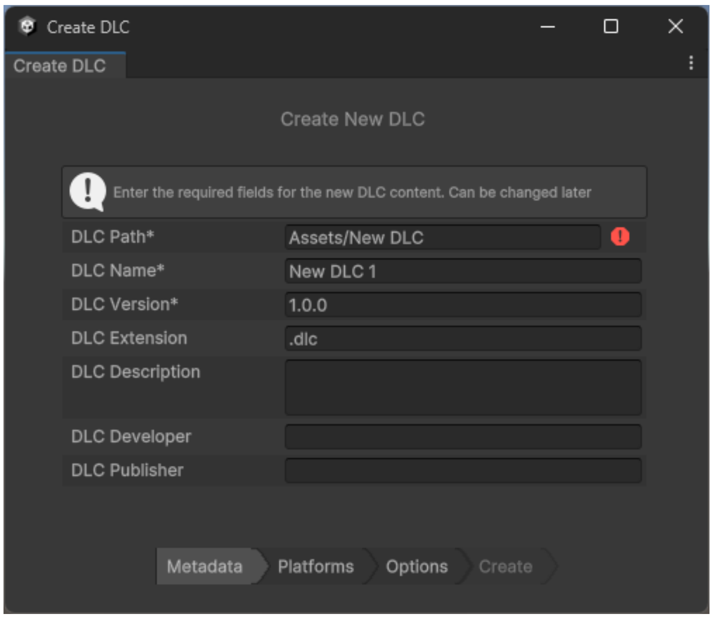
Tools -> DLC Toolkit -> Create DLC

Note: *You can use DLC content for virtually any purpose, and ultimate DLC Toolkit supports creating and loading an unlimited number of individual DLCs (Memory is usually the limiting factor). You can create very large DLCs (Many GB's in size) containing many more game items and maps in the case of expansion packs or similar. Inversely you can create thousands of small DLCs containing very few or even single main assets such as game items or skins, and Ultimate DLC Toolkit will be happy to manage that for you.*

Of course, everything in between is also possible!

Metadata

Once you have selected the menu item, you will be presented with the create new DLC wizard window which will look something like this, depending on version:



Enter the required fields for the new DLC content. Can be changed later

DLC Path*	Assets/New DLC
DLC Name*	New DLC 1
DLC Version*	1.0.0
DLC Extension	.dlc
DLC Description	
DLC Developer	
DLC Publisher	

This is the first stage of the wizard window is where you should enter all the required values (Denoted by a trailing `*`), and you can optionally provide some extra metadata now or at a later time. The required values for this section are:

- **DLC Path:** This is the path in the project `Assets` folder where the DLC will be created. The DLC content should be placed inside its own folder, so you should pick a subfolder of assets, or a nested folder if better organisation is required, for example: `Assets/DLC/New DLC 1`. The folder path should be located under the project `Assets` folder and cannot be a folder that already exists, or an error will be shown.

- **DLC Name:** The name of the DLC. The name will be used as the filename for the output DLC file, and also as part of the DLC content metadata. The name can be changed at a later time via the DLC Profile asset.
- **DLC Version:** The version of the DLC formatted as `X.X`, or `X.X.X`, or `X.X.X.X` where `X` denotes major, minor, revision, and build versions respectively. The version can be changed at a later time via the DLC Profile asset, and it is recommended that a new version is assigned every time you release an update to a DLC. Usually, the value can remain as `1.0.0` when creating a new DLC, unless an alternate version formatting is required.

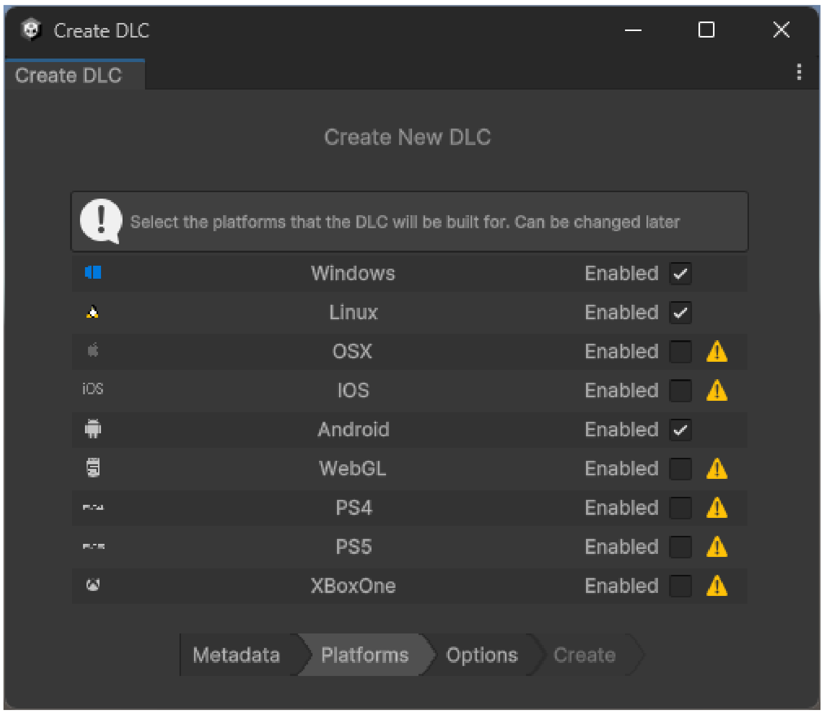
There are also some additional optional fields that you can fill in for this section, although it is not required. Note that all of these optional values can be changed at a later time via the DLC Profile asset. The optional values for this section are:

- **DLC Extension:** The file extension assigned to the DLC when built. The default is `.dlc` which can be changed to any extension to suit your game, or can be removed completely if no extension should be used. Note that this value will be assigned to all build platforms, but later you will be able to modify the file extension per platform via the DLC Profile asset.
- **DLC Description:** A short description of the DLC. Will be available at runtime as part of the metadata which can be useful to show some descriptive info as part of a UI screen for example.
- **DLC Developer:** The name of the individual or company who authored the DLC content. Will be available at runtime as part of the metadata which can be useful to show some descriptive info as part of a UI screen for example.
- **DLC Publisher:** The name of the company responsible for publishing the DLC content. Will be available at runtime as part of the metadata which can be useful to show some descriptive info as part of a UI screen for example.

Once all required values have been provided, you can select the `Platforms` section at the bottom of the wizard page to advance to the next section.

Platforms

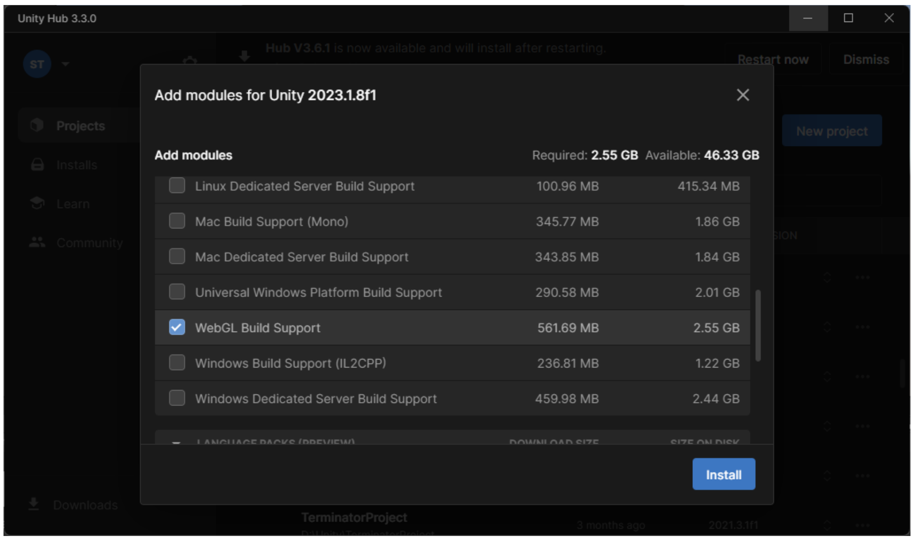
The platforms section as the name indicates allows you to setup which platforms the DLC will be built for. Usually, it is best practise to only select the platforms that you intend to distribute your game on, for example if you have a desktop game, you may select Windows, Osx, and Linux. If you have a mobile game, you may only select Ios and Android. Note that the fewer platforms you select, the quicker the DLC build time will be.



Select the platforms that the DLC will be built for. Can be changed later

Windows	Enabled	✓
Linux	Enabled	✓
OSX	Enabled	⚠
IOS	Enabled	⚠
Android	Enabled	✓
WebGL	Enabled	⚠
PS4	Enabled	⚠
PS5	Enabled	⚠
XBoxOne	Enabled	⚠

An important thing to consider is that DLC toolkit can only build DLC for a given platform if the required Unity build tools are installed. As you can see from the image above, a warning will be shown next to any platforms that don't currently have the necessary build tools installed. Note that you will still be able to enable DLC build support for that platform, but the DLC will not be built until the necessary tools are installed. Platform build tools can be installed via Unity hub simply by installing build support for that target. The below image shows WebGL build support being added which will allow DLC content to be built for WebGL platforms:



Add modules for Unity 2023.1.8f1

Add modules — Required: **2.55 GB** Available: **46.33 GB**

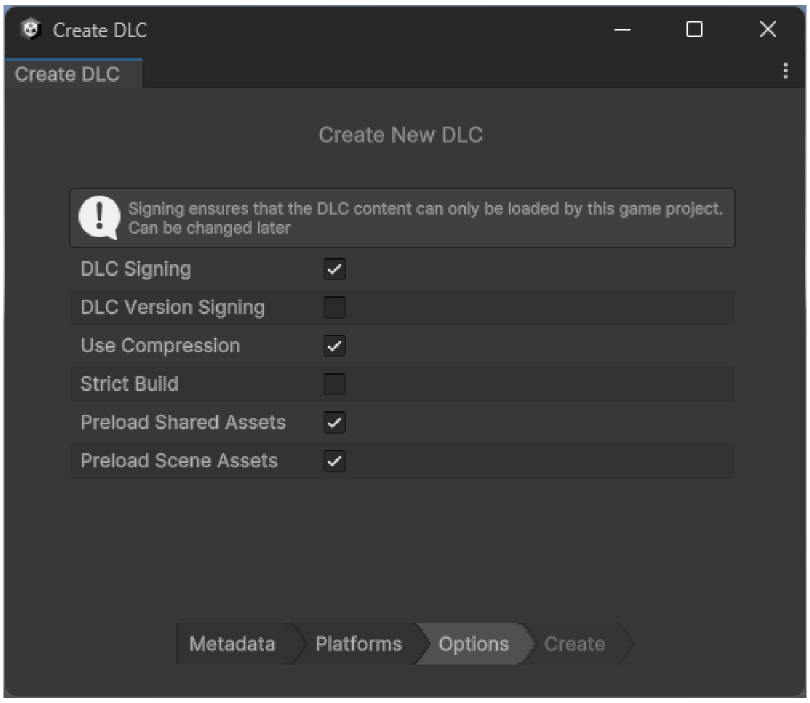
☐	Linux Dedicated Server Build Support	100.96 MB	415.34 MB
☐	Mac Build Support (Mono)	345.77 MB	1.86 GB
☐	Mac Dedicated Server Build Support	343.85 MB	1.84 GB
☐	Universal Windows Platform Build Support	290.58 MB	2.01 GB
✓	WebGL Build Support	561.69 MB	2.55 GB
☐	Windows Build Support (IL2CPP)	236.81 MB	1.22 GB
☐	Windows Dedicated Server Build Support	459.98 MB	2.44 GB

Note that enabled build platforms can be changed at a later time.

Once you have selected the platforms you would like to build for, you can continue to the next section by selecting the `Options` button at the bottom of the wizard window.

Options

The options section allows you to configure some extra settings for the DLC which will be used at build time or runtime. For most users the default values will usually be fine, but more advanced users or larger projects may require a bit more control over the DLC content to effect things like build size and load times.



Signing ensures that the DLC content can only be loaded by this game project. Can be changed later

DLC Signing	✓
DLC Version Signing	
Use Compression	✓
Strict Build	
Preload Shared Assets	✓
Preload Scene Assets	✓

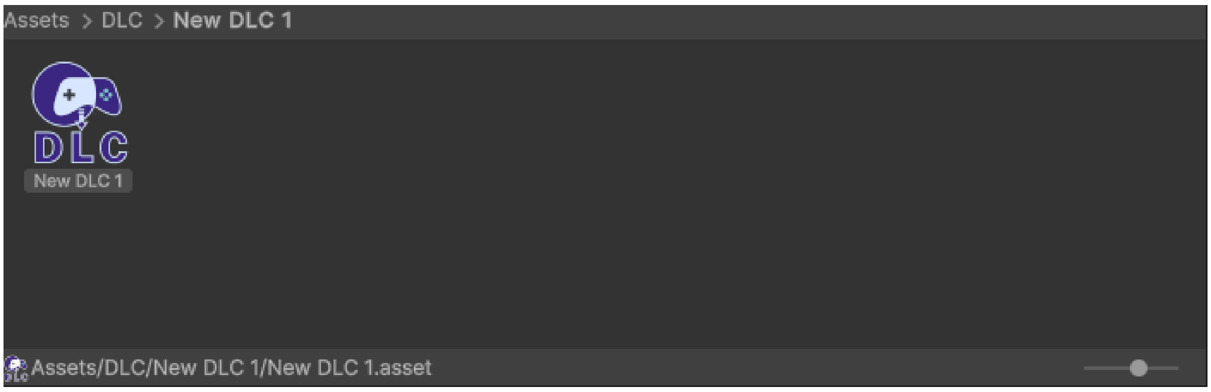
The following options can be set for all platforms during this stage, or can later be changed on a per platform basis via the DLC Profile asset:

- **DLC Signing:** Should the DLC content be signed to this game project when building. Signing means the DLC will only be loadable by the project that created it, so it won't be possible for other games also using DLC Toolkit to load content signed to a different project. It is recommended to leave this option enabled for most cases.
- **DLC Version Signing:** Should the DLC content be signed to the specific version of this game project. The DLC content will only be loadable by the specified version of the game project that created it. Useful for locking DLC content to a specific version of the game, but not recommended for most users.
- **Use Compression:** Should the DLC content be compressed to reduce file size. Compression is generally recommended for all cases, however if load time is a concern and storage space not so much, you may find that disabling compression reduces the load time required for the same DLC content at the expense of file size on disk.
- **Strict Build:** Should the DLC content be built in strict mode. When enabled, any non-fatal errors encountered will be treated as hard errors causing the build to fail.
- **Preload Shared Assets:** Should any shared assets included in the DLC be preloaded during the DLC load process so that they are quicker to access on demand. Recommended for most cases otherwise shared assets will need to be lazy loaded on demand when requested, such as when instantiating a prefab.

- **Preload Scene Assets:** Should scene assets included in the DLC be preloaded during the DLC load process. This will cause any scene asset bundles to be loaded during the DLC load phase making any subsequent DLC scene load requests quicker.

Create

After selecting the most suitable options for you DLC content, you are then ready to hit the create button to generate the new DLC. Hitting the create button will generate a new folder at the path specified in the `Metadata` section and will also create a DLC Profile asset in that folder containing all the necessary information and settings that you configured for the DLC via the wizard.



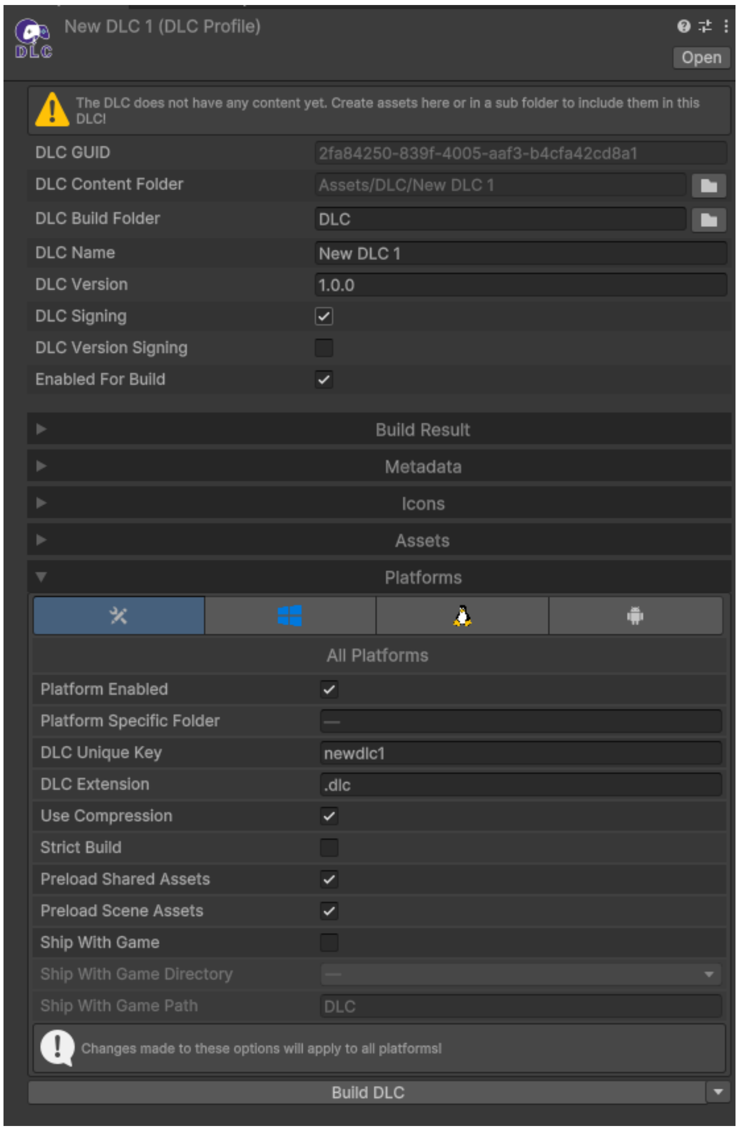
Assets/DLC/New DLC 1/New DLC 1.asset

The DLC Profile asset (`New DLC 1.asset` in the above image) can now be used to edit any metadata or settings for the DLC content at any time, and the folder is where all associated DLC assets should be placed. Of course, sub folders are fully supported for better organisation, and it is possible also to use assets and scripts from the base game when creating your content.

Next topic is adding assets to your DLC, however it is recommended that you read about the DLC Profile asset before continuing.

DLC Profile

The DLC profile is the main configuration asset for a particular DLC (One profile for each DLC you create) and stores all associated metadata and build options per platform so that DLC Toolkit knows what information to include and how to build the content for a specific platform.



New DLC 1 (DLC Profile)

Open

The DLC does not have any content yet. Create assets here or in a sub folder to include them in this DLC!

DLC GUID	2fa84250-839f-4005-aaf3-b4cfa42cd8a1
DLC Content Folder	Assets/DLC/New DLC 1
DLC Build Folder	DLC
DLC Name	New DLC 1
DLC Version	1.0.0
DLC Signing	✓
DLC Version Signing	
Enabled For Build	✓

Build Result

Metadata

Icons

Assets

Platforms

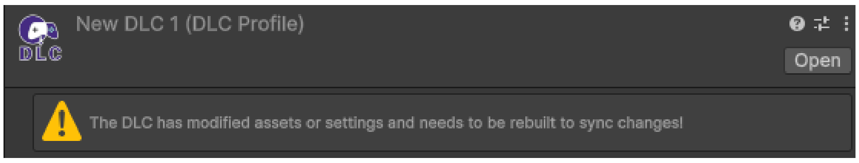
All Platforms

Platform Enabled	✓
Platform Specific Folder	—
DLC Unique Key	newdlc1
DLC Extension	.dlc
Use Compression	✓
Strict Build	
Preload Shared Assets	✓
Preload Scene Assets	✓
Ship With Game	
Ship With Game Directory	—
Ship With Game Path	DLC

Changes made to these options will apply to all platforms!

Build DLC

It will also provide useful warning and hints at the very top of the inspector window to notify you if the DLC needs to be rebuilt due to modified assets for example, if the DLC is not enabled for build, or if the DLC does not have any assets associated with it yet:



New DLC 1 (DLC Profile)

Open

The DLC has modified assets or settings and needs to be rebuilt to sync changes!

Most settings for the DLC can be changed at any time, however you will notice a couple of disabled fields that may not be edited as they will be filled out automatically by Ultimate DLC Toolkit:

- **DLC Guid:** A guid value is assigned to the DLC profile asset to uniquely identify the DLC content at both edit time and runtime. This guid value is built into the DLC metadata and is available at runtime via the metadata API. Cannot be edited manually.
- **DLC Content Folder:** The folder location inside the project `Assets` folder where associated DLC assets should be placed. This is always the folder where the DLC profile asset is located, and if the profile asset is moved, the path will also update to reflect that. Cannot be edited manually as it is auto updated based on the DLC Profile asset location.

The remaining options can all be edited at any time:

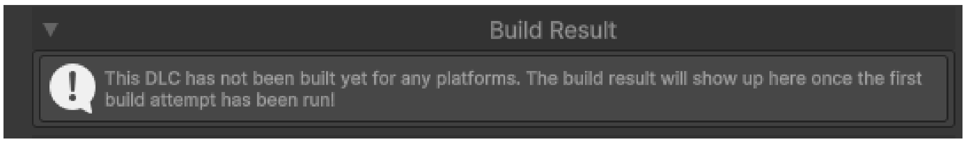
- **DLC Build Folder:** The folder location where built DLC will be saved. The default folder is `DLC` which is a path relative to the project folder (`<project_path>/DLC/`), meaning the folder will be created next to the project `Assets` folder.
  You can click the folder icon to the right and it will open the build folder in finder if it exists.
- **DLC Name:** The descriptive name for the DLC. This is not a unique identifier and is used only for display purposes, Ie displaying the DLC name in the game if required as part of the DLC metadata. Spaces and special characters are allowed.
- **DLC Version:** The version information for the DLC formatted as X.X or X.X.X or X.X.X.X. It is recommended that the version information is updated each time a release version (A version that will be distributed to users) of the DLC is built.
- **DLC Signing:** Should the DLC content be digitally signed to your game project. It is strongly recommended that this feature is enabled. Since Ultimate DLC Toolkit is a general solution, it could be possible for other games also using DLC Toolkit to be able to load the DLC created for your game if this option is disabled. Enabling it adds a digital fingerprint to the DLC which the game project must also have in order to load the DLC, making it almost impossible to load the DLC content in a different project. This process is fully automatic at build time and the digital fingerprint is created from your specific Unity project.
- **DLC Version Signing:** Adds an additional signing layer where the DLC will only be loadable if it was built with the same version of the game that it was created with. Project version information is taken from Unity player settings (Version) and is incorporated into the digital fingerprint so that if the game version is updated, the DLC will no longer be loadable. Not recommended in most cases due to the strict nature and possibility to break backwards compatibility.
- **Enabled For Build:** Convenience option that allows you to quickly and easily disable a specific DLC from the build pipeline. With the option disabled, the DLC will not be included in batch build processes started via one of the build menu commands, for example:

  Tools -> Build DLC -> All Platforms

  > **Note:** *The `Build DLC` button for the* DLC Profile *will still be usable, and this feature is related mainly to batch building.*

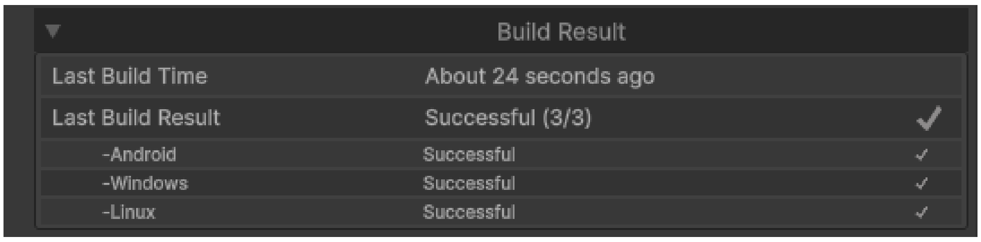
Build Result

The build result section shows useful result information about the last build attempt. It can give you a quick indication of whether the last performed build was successful, for which platforms, and can be useful if you are coming back to a project after some time has passed. If the DLC has not yet been built at all, then you will see the following information message:



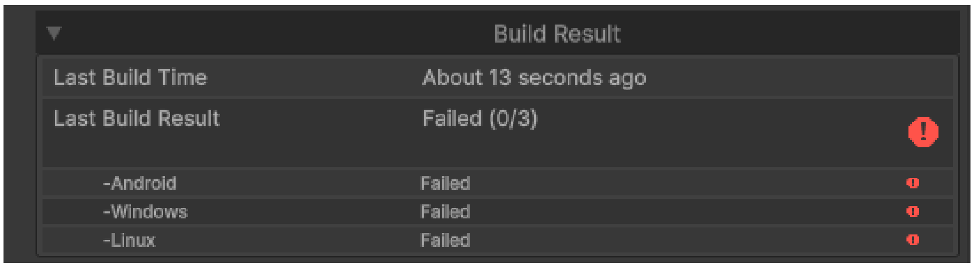
This DLC has not been built yet for any platforms. The build result will show up here once the first build attempt has been run!

Upon a successful DLC build, you can expect to see an information message similar to the following, showing the approximate time since the last build and the build result per platform. Note that this section will only show the platforms which were last built even if more platforms are enabled:

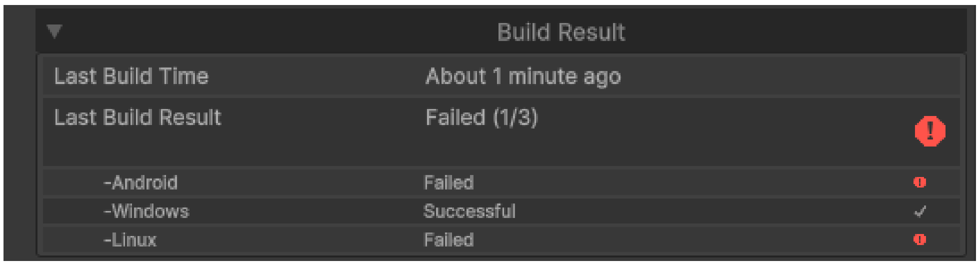


Last Build Time	About 24 seconds ago	
Last Build Result	Successful (3/3)	✓
-Android	Successful	✓
-Windows	Successful	✓
-Linux	Successful	✓

Upon a failed DLC build, you will see one or more error indications informing you of which platforms were successfully built and which were not. You should check the console for detailed error reporting as to why the build has failed:



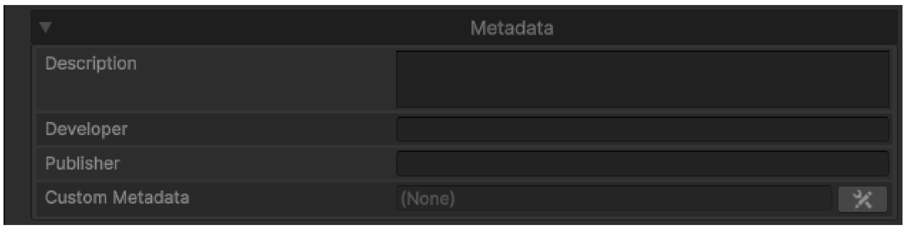
Last Build Time	About 13 seconds ago	
Last Build Result	Failed (0/3)	!
-Android	Failed	!
-Windows	Failed	!
-Linux	Failed	!



Last Build Time	About 1 minute ago	
Last Build Result	Failed (1/3)	!
-Android	Failed	!
-Windows	Successful	✓
-Linux	Failed	!

Metadata

The metadata section of the `DLC Profile` allows you to provide additional information about the DLC which will be available as quick access metadata during runtime, meaning that it can be accessed quickly by only loading the file headers of the DLC rather that performing a full load.



- **Description:** An optional short description of the DLC content.
- **Developer:** An optional developer/author name of the individual or company that created the DLC content. Usually, it will be the same as the company name set in the Unity player settings.
- **Publisher:** An optional name of the individual or company that published the DLC content. Can be the same as the developer's name if self-publishing or could be the name of a third-party publisher if required.

Custom Metadata

In addition to the built-in metadata fields that Ultimate DLC Toolkit offers, it is also possible to add extra custom user data if you need to store additional quick access data. Custom metadata is stored alongside the built-in metadata as part of the quick access headers, meaning it can be accessed very quickly from disk with only a minimal partial load of the DLC being required.

For that reason, it is perfect for storing extra information that may need to be accessed when the DLC has not yet been loaded, as a quick partial load will fetch the data on demand.

Custom Type

In order to define which custom metadata you would like to store in Ultimate DLC Toolkit, you will simply need to create a C# class that derives from `DLCToolkit.DLCCustomMetadata`. The class you create does not require adding or overriding any methods or properties, but you can simply declare any number of supported serializable field types following the traditional Unity workflow. Ie. Public fields will be serialized, also private fields using the `[SerializeField]` attribute, etc.

You can check out an example custom metadata type here:

Custom Metadata Type Code Sample

Any primitive field type will be supported, as well as arrays, lists and a few Unity types:

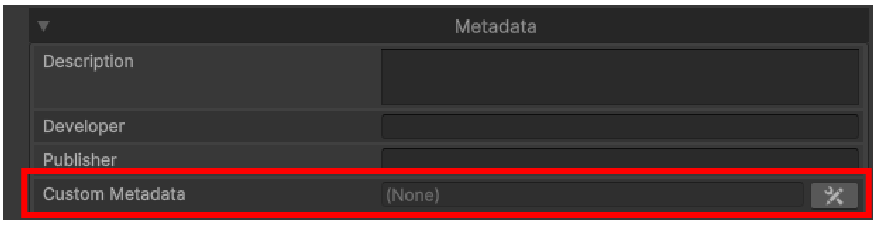
- **Integer Type:** 8, 16, 32 and 64 bit both signed and unsigned.
- **Decimal Type:** 32 and 64 bit floating point.
- **String Type:** System.String
- **Other Primitive Type:** Boolean, Char, signed and unsigned IntPtr
- **Unity Types:** Vector2/3/4, Quaternion, Color, Color32, and many other Unity types
- **Serialized classes/Structs:** Custom classes or structs not inheriting from UnityEngine.Object will also be serialized if they use the `[System.Serializable]` attribute.

- **Arrays/Lists:** Additionally arrays or generic lists where the element type is any of the above will also be serialized.

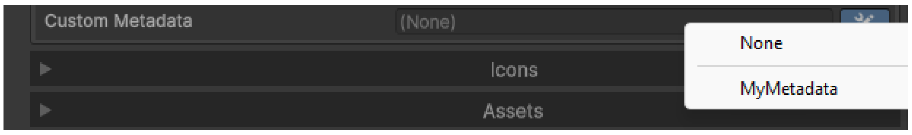
Note: *The metadata section of the DLC content is intended to be quick access. Be aware that adding large serialized fields will increase the time required to load and process the custom metadata, so you should always store the minimum amount of data possible, or profile custom metadata access times on target hardware to ensure they are suitable.*

Assign Custom Metadata

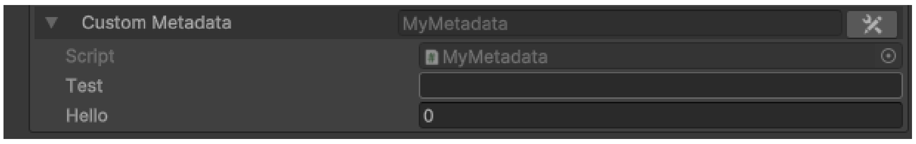
Once you have created your custom metadata script, the next step is to assign it in a DLC Profile asset so that the custom metadata can be built as part of the DLC content. You can do that via the `Custom Metadata` property of the `Metadata` section:



Click the button to the right of the field to bring up a context menu where you can select your custom metadata type. From there you should see that the type has been auto detected and listed in the menu:



After selecting your metadata type, you will see that a new foldout section is added under the `Custom Metadata` property where you can edit your custom metadata values as required. These values are persistent and will be saved when the Unity project is saved:



Note: *You can create any number of custom metadata types and each DLC you create can have their own custom metadata if required. It is however recommended to use a single custom metadata script between all DLC as it achieves better consistency and there will be no uncertainty of the custom metadata type when it comes to accessing the data at runtime.*

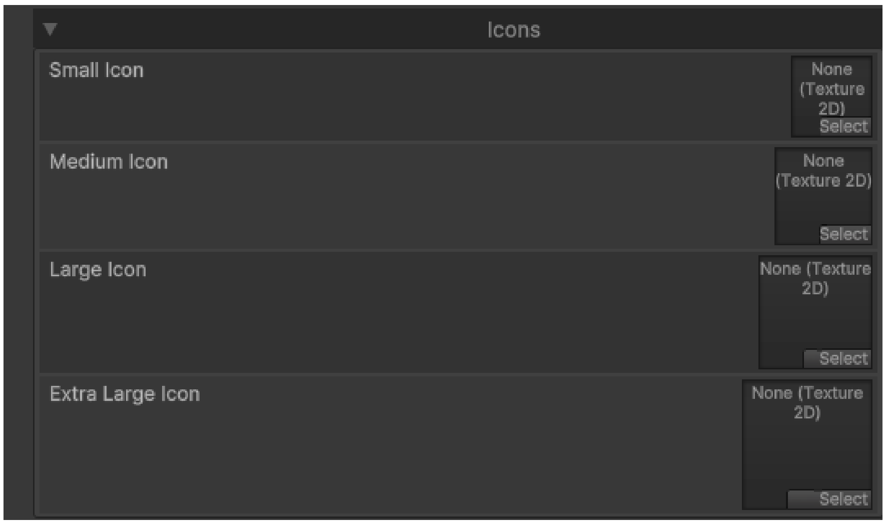
Loading Custom Metadata

Once you have input some custom metadata values and built the DLC content, you will be ready to access it via the runtime API. Check out the following code sample to see how you can access that:

Load Custom Metadata Code Sample

Icons

The icons section of the DLC profile asset allows you to add a selection of various sized icons for your DLC content which will be available at runtime with only a partial load required, meaning that they are available at relatively quick access speed without requiring a full load of the DLC content. This is useful for showing DLC icons in game UI screens such as at startup / splash or perhaps via a settings menu where you can show the user which DLC content they own/have installed.



As you can see there are preset slots available for small, medium, large and extra-large icons. There are no restrictions in place regarding the size or aspect of these icons, so you can use images that best suit the purpose, and each icon is optional. The only thing to consider is that high-res images will slow the speed at which the icons can be fetched at runtime, so the best practice is to keep them as small as possible without losing quality on the intended target hardware resolution.

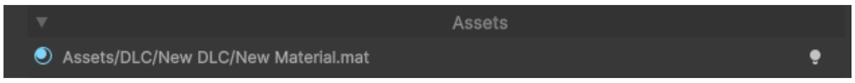
Loading Icons

If you assign any icons here, you will then be able to load and access them via the Runtime API without requiring a full load. You can find a complete code example for this at the samples repository:

Load Icons Code Sample

Assets

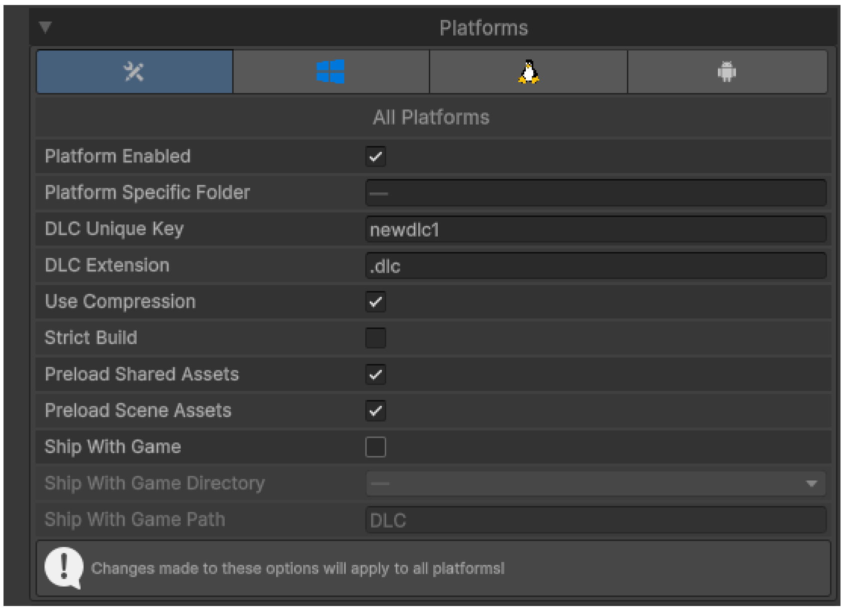
The assets section is a helpful display for quickly checking which assets will be associated with a given `DLC Profile`. This section is covered in more detail in the DLC Assets section.



Assets/DLC/New DLC/New Material.mat

Platforms

The platforms section allows you to edit additional per-platform options that affect how the DLC will be built for the build target, and it also allows you to edit platform specific options in addition, such as Android delivery type options.



Platform Enabled	✓
Platform Specific Folder	—
DLC Unique Key	newdlc1
DLC Extension	.dlc
Use Compression	✓
Strict Build	
Preload Shared Assets	✓
Preload Scene Assets	✓
Ship With Game	
Ship With Game Directory	—
Ship With Game Path	DLC

Changes made to these options will apply to all platforms!

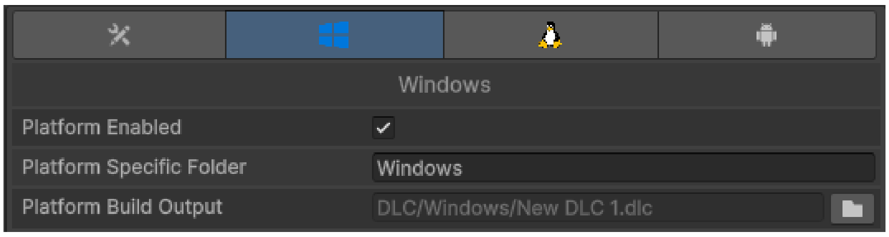
Note: *The default selected tab will edit settings for all platforms together. If you want to edit the settings only for a specific platform, then you should switch the platform tab to the correct platform logo*

- **Platform Enabled:** A utility option which allows you to quickly enable or disable building of the selected platform.
- **Platform Specific Folder:** Allows you to change the output folder for the DLC relative to the `DLC Build Folder`. When building any DLC content, Ultimate DLC Toolkit will calculate the final output path by combining the following values:

DLC Build Folder / Platform Specific Folder / DLC Name + DLC Extension

The platform specific folder is normally used to separate DLC by platform and can contain multiple sub folders if required. On the Windows platform, the default special folder is `Windows` for example, and a similar naming scheme applies for all other platforms by default. This is because the DLC produced for each platform could have the same name and extension, so we need to organize the files in this structure to prevent overwriting or clashing.

If you select a specific platform tab, then there is an additional preview field named **Platform Build Output** which will give you a preview of the build path, and will also allow you to open that file in the finder if the DLC has already been built:



Windows

Platform Enabled	✓
Platform Specific Folder	Windows
Platform Build Output	DLC/Windows/New DLC 1.dlc

- **DLC Unique Key:** This is an important field that uniquely identifies the DLC content on a specific platform. It is a required value and depending upon your DRM requirements, it may or may not need to be set to a specific value.
  For example, if you are using Steam DRM via Steamworks.Net or similar, then this unique key should be set to the steam app ID of the DLC created via the Steam dashboard.

  There is more information about this in the DRM section.

- **DLC Extension:** As you might expect, this is the file extension assigned to the DLC at build time and can be different per platform if required. Allows you to set your own file extension if require to better suit your game, otherwise the default is the generic `.dlc` extension.
  No extension is also supported if required by leaving the field blank.
- **Use Compression:** Should the DLC be built with compression enabled. Enabling compression is recommended as it can greatly reduce the output size of the DLC once built, however it can lead to longer loading times so in some cases it may be a better trade off to disable compression to aid in faster loading.
- **Strict Mode:** Should the DLC be built with strict mode enabled. Strict mode applies to the various asset bundles which Ultimate DLC Toolkit may need to build for the target platform, and string build means that warnings can be treated as errors to cause a failure.
- **Preload Shared Assets:** Modifies the loading behaviour of the DLC so that shared assets (Prefabs, Materials, Textures, etc) can be preloaded as part of the main loading phase. Disabling this option will only load assets on demand as they are accessed via the runtime API, so with that in mind, it is highly recommended that you only use async loading of assets if this option is disabled to avoid any blocking of the main thread.

  You can find complete code samples for loading shared assets via the runtime API here:

  Load Shared Assets Code Sample

- **Preload Scene Assets:** Modified the loading behaviour of the DLC so that scene assets (.unity3d) can be preloaded as part of the main loading phase. Disabling this option will only load scenes on demand as they are accessed via the runtime API. This is not as critical as the shared assets option, since typically it makes sense to show a loading screen in any case during large scene transitions, however disabling this option will shift all scene loading to on demand as the scenes are accessed.

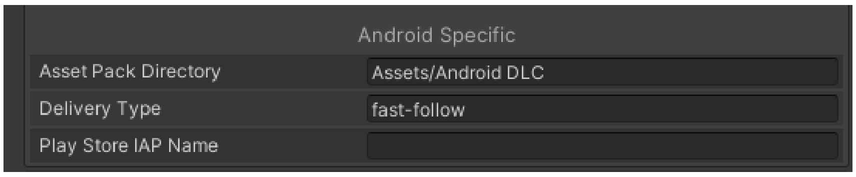
  You can find complete code samples for loading scene assets via the runtime API here:

  Load Scene Assets Code Sample

- **Ship With Game:** This and associated options are discussed in more detail in the Build DLC Section.

Android Specific

The Android platform has some additional options available that can only be edited by selecting the Android platform tab under the `Platforms` section:



Android Specific

Asset Pack Directory	Assets/Android DLC
Delivery Type	fast-follow
Play Store IAP Name	

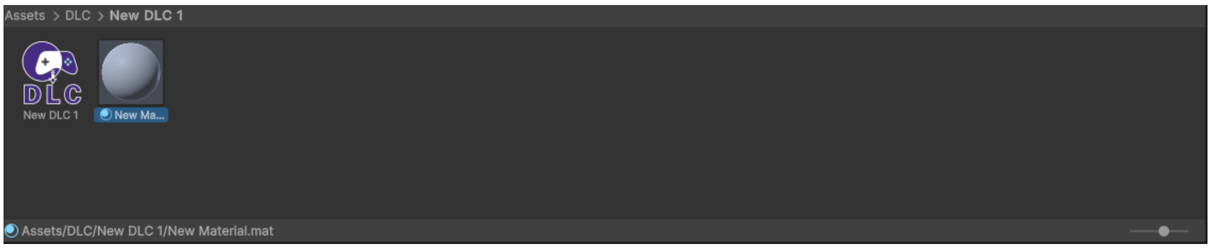
- **Asset Pack Directory:** The directory in the Unity project where DLC toolkit will package the DLC content for Android with additional necessary resources. Ultimate DLC Toolkit will place in this folder the built DLC content renamed with only the unique key, and will also generate a `build.gradle` file.
- **Delivery Type:** Represents the play store delivery mode that will determine how and when the DLC content gets installed onto the user's device. The following values are allowed:
    - **install-time:** The DLC content is installed at the same time the base app is installed.
    - **fast-follow:** The DLC content is downloaded automatically as soon as the base app is installed. The user is free to launch the base app in the meantime, or if the base app remains unlaunched, then the DLC content will still be streamed in the background until installed.
    - **On-demand**: The DLC content is only installed when requested by the runtime API, via an Android DRM provider.
- **Play Store IAP Name**: If you are creating a paid DLC, then this is the name of the Play Store IAP that is linked to the DLC. You can create an In-App Purchase separately via the Google Play dashboard for each paid DLC you wish to offer, and you can then simply input the name of that IAP into this field to create a link. Now when Ultimate DLC Toolkit attempts to access the DLC, it will first check that the user has purchased the associated IAP before granting access to the DLC content.
  Simply leave this field empty if you are not creating a paid DLC.

> **Note:** *Linking DLC content to a Play Store IAP requires Unity.Purchasing to be setup in the project, otherwise the DLC content may be accessible by all users. More info on this in a later Android DRM section.*

DLC Assets

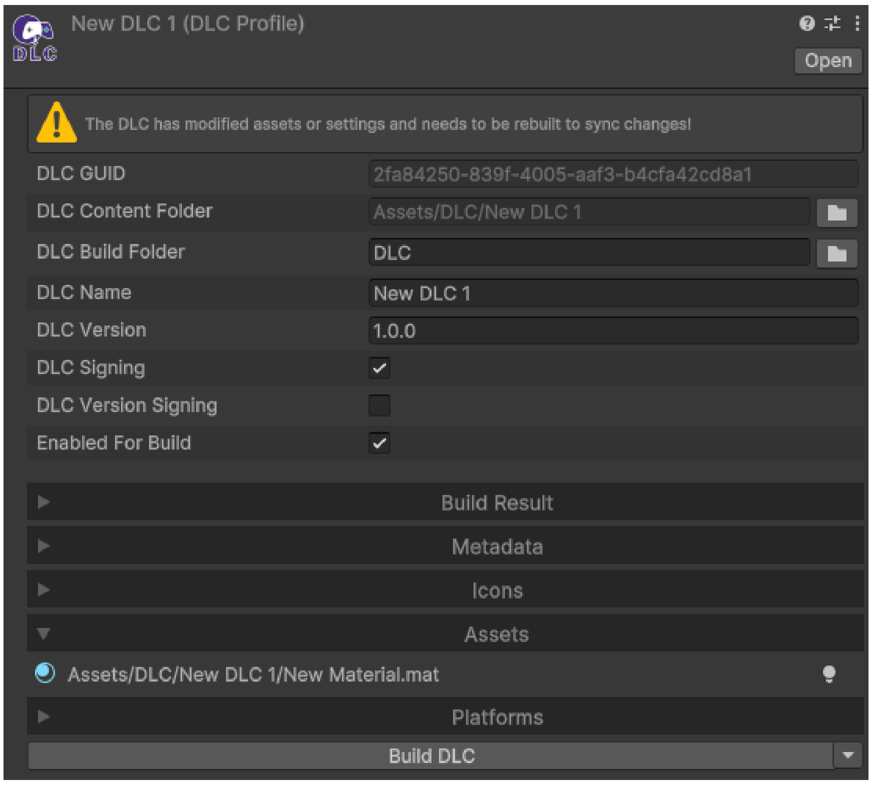
Once you have created a DLC Profile, the next step will be to create the assets that you want to include in the DLC. You can use standard Unity workflow for this, and all asset types are supported (C# scripts only supported on Mono backend).

Including assets as part of the DLC is very simple. All you need to do is place the assets inside the DLC folder (next to the DLC Profile asset or in a sub folder), and they will be automatically included by DLC Toolkit. You can verify that an asset has been included in the DLC by checking the `Assets` section of the `DLC Profile ` asset. For example, if we create a new material inside the DLC folder:



Assets/DLC/New DLC 1/New Material.mat

We can see that the new material is automatically detected and included in the DLC Profile with the asset icon and project path. You can also click the lightbulb icon on the right to ping the asset in the project window:



The DLC has modified assets or settings and needs to be rebuilt to sync changes!

DLC GUID	2fa84250-839f-4005-aaf3-b4cfa42cd8a1
DLC Content Folder	Assets/DLC/New DLC 1
DLC Build Folder	DLC
DLC Name	New DLC 1
DLC Version	1.0.0
DLC Signing	✓
DLC Version Signing	
Enabled For Build	✓

Assets/DLC/New DLC 1/New Material.mat

Note: *Some assets listed in this section may not actually be included in the built DLC, but instead this list represents assets that are associated with the DLC Profile in the editor. Special Folders are evaluated at build time and can cause some assets to be excluded or only included for certain platforms.*

Special Folders

Ultimate DLC Toolkit allows certain specially named folders to be used to have better organization and control over which assets get included in the built DLC. It is similar to some of the Unity special folders such as `Editor` or `Plugin` which change how assets are used in the editor.

Note: *All special folder names are case sensitive so be sure to use the exact name shown or the special folder will not work as intended. Special folder names can also be sub folders and do not need to be a root folder in the DLC content folder.*

Exclude Folders

Exclude folders can be used to place assets inside the DLC content folder at any depth, but any assets within will not be built as part of the DLC. This is useful for keeping DLC specific editor tools or testing content well organized with other DLC assets but keep them from bloating the output size of the DLC if they are not required for runtime, or are editor only assets.

- **Editor:** Behaves the same as the Unity special folder. Content inside this folder will be treated as editor only content and will not be built. Recommended specifically for scripts containing editor tools but can also be used for any other asset type.
- **Exclude:** A DLC specific folder. Any assets placed inside a folder with this special name will not be included in the DLC. Useful if you have runtime scripts for testing purposes or similar that cannot belong in the editor folder but should still not be included in the DLC.

Platform Folders

Additionally Ultimate DLC Toolkit supports platform specific folders that can be used if you only want to include certain assets on certain platforms. This works because the DLC must be built individually for each platform target as asset bundles are used internally for some aspects.

- **Windows:** Any assets placed inside a folder or sub folder with this special name will only be included in the DLC when building for the StandaloneWindows build target.
- **Linux:** Any assets placed inside a folder or sub folder with this special name will only be included in the DLC when building for StandaloneLinux build target.
- **OSX:** Any assets placed inside a folder or sub folder with this special name will only be included in the DLC when building for StandaloneOSX build target.
- **Android:** Any assets placed inside a folder or sub folder with this special name will only be included in the DLC when building for Android build target.
- **IOS:** Any assets placed inside a folder or sub folder with this special name will only be included in the DLC when building for IOS build target.
- **WebGL:** Any assets placed inside a folder or sub folder with this special name will only be included in the DLC when building for WebGL build target.
- **PS4:** Any assets placed inside a folder or sub folder with this special name will only be included in the DLC when building for PS4 build target.
- **PS5:** Any assets placed inside a folder or sub folder with this special name will only be included in the DLC when building for PS5 build target.
- **XboxOne:** Any assets placed inside a folder or sub folder with this special name will only be included in the DLC when building for XBoxOne build target.

Additionally, there are platform group folders which can be useful if you want to include certain content only on desktop platforms for example:

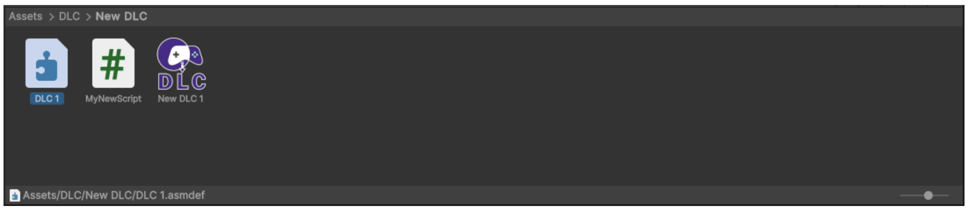
- **Desktop:** Any assets placed inside this folder will only be built for Windows, OSX and Linux platforms.
- **Mobile:** Any assets placed inside this folder will only be built for IOS and Android.
- **Console:** Any assets placed inside this folder will only be built for PS4, PS5 and XboxOne

Note: *The Unity `Resources` folder is not supported in DLC content and is indeed not required in any case since we have the option of loading any asset or scene by name or path on demand, including async loading.*

Scripts

You can create new C# script assets as part of DLC if you are targeting a Desktop platform using Mono backend. Mono is required to support scripts since IL2CPP due to it's AOT nature does not support loading code dynamically.

Creating scripts uses the exact same Unity workflow that you might expect, although it is a requirement to use assembly definition files to segregate the DLC code from your game code. You can create one main .asmdef file in the root of the DLC, or additionally any number of .asmdef files are supported if you prefer to organize code via different modules (.dll's).



For platforms where scripting is not supported, the scripts included in the DLC will simply be ignored, so usage of those scripts on unsupported platforms will be reported as missing scripts when the DLC content is loaded into game.

Note: I*f you include scripts as part of your DLC but the target platform does not support scripting, you will receive a helpful warning during the build process to inform you of this. If you are targeting both platforms that support scripting and platforms which do not, then you can disable the warning by placing your scripts inside platform specific folders.*

Finally, you can of course use any scripts you have created as part of the base game on any platform. This will be the most common workflow because typically you will have created behaviour scripts as part of the base game, and they can be reused in the DLC with no issues. For example: If you are creating a racing game, you may have a car controller script or third-party asset that you are using for the car physics. Now when you want to add a new car model via DLC, you can simply attach that script/plugin to your new car model prefab inside your DLC folder and everything will work just as you would expect.

Next topic is setting up and using DRM for distribution and management purposes.

DRM

DRM or Digital Rights Management can be a vital aspect in managing DLC content owned by the user, especially if the DLC is paid content and requires an extra transaction to own. There are many online game services that have built in DRM for this or similar purposes such as Steam, and Ultimate DLC Toolkit tries to make it as easy as possible to integrate with some of those services.

> **Note:** *DRM is not a requirement for Ultimate DLC Toolkit to function, and it will happily load DLC content from any location, even dropped into a certain game folder by the user if required. It is however recommended if you are distributing paid DLC via an online store such as Steam or Google Play so that the ownership of the DLC can be verified before it is ever installed or loaded.*

DRM in Ultimate DLC Toolkit uses the DLC Unique Key to uniquely identify the content, and to connect it with the store version of the DLC where the content will be sold or distributed if using such as service such as Steam store. The unique key you enter into Ultimate DLC Toolkit will usually be some number, guid or product identifier that is created via the DRM service tools (Steam partner in this example). Check out the section for a particular DRM provider for more information about which DLC unique key should be used.

DRM Providers

Ultimate DLC Toolkit includes a few built in DRM providers for common DLC distribution/management services that you might be using for your project. The following section will go though each built in provider and the necessary setup that is required to enable DRM for that particular service:

Steamworks (Desktop)

Ultimate DLC Toolkit has built in DRM support for Steam via a couple of popular plugins for Unity. There is a dedicated guide included with the Ultimate DLC Tookit asset named `Steamworks DRM` which you should check out, as it covers this topic in detail.

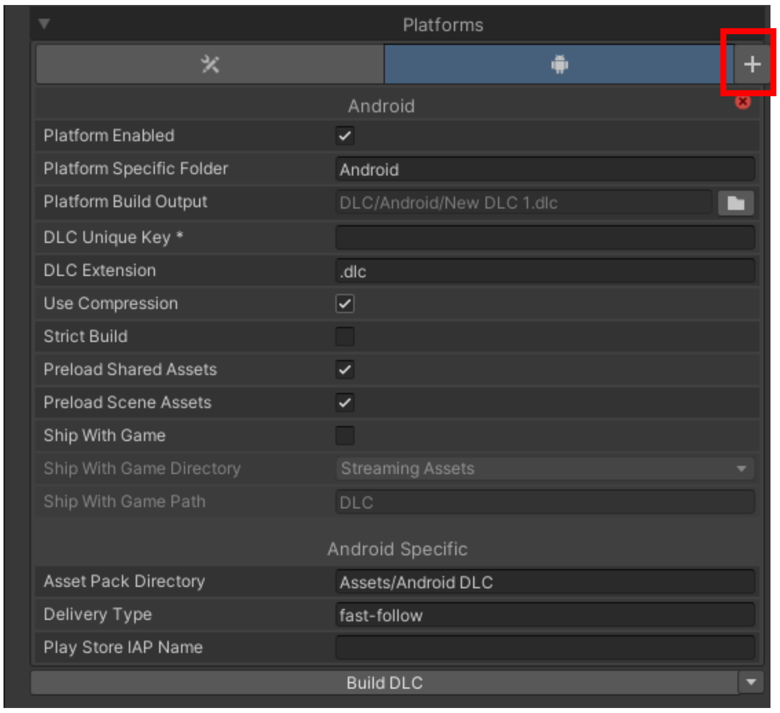
Google Play (Android)

Ultimate DLC Toolkit has built in DRM support for Google Play content delivery for DLC purposes. All the build tasks and necessary API calls can be handled automatically by Ultimate DLC Toolkit once the Google Play DRM has been setup.

1. The first step is to setup your Unity project for Android if you have not done already. You check the Unity docs for Android setup to install the necessary packages and development kits.
2. (Optional): Ultimate DLC Toolkit uses the Google Play asset delivery API for managing DLC content on Google Play store. This does not allow for ownership verification by default, and you will need to setup IAP in addition if you want to offer paid DLC on Google Play with checks in place to ensure that the user has actually purchased the content. For now all you need to know is that if you intend to offer paid DLC on Google play store, then you will need to setup Unity.Purchasing which can be found here.
3. Add the following preprocessor directive to the Unity player settings scripting define symbols and allow Unity to recompile:

```
DLCTOOLKIT_DRM_GOOGLEPLAY
```

Add an Android platform to your DLC content if you have not already by clicking the plus (Add) icon under the platforms section:



Platform Enabled	✓
Platform Specific Folder	Android
Platform Build Output	DLC/Android/New DLC 1.dlc
DLC Unique Key *	
DLC Extension	.dlc
Use Compression	✓
Strict Build	
Preload Shared Assets	✓
Preload Scene Assets	✓
Ship With Game	
Ship With Game Directory	Streaming Assets
Ship With Game Path	DLC
Asset Pack Directory	Assets/Android DLC
Delivery Type	fast-follow
Play Store IAP Name	

You will see that there are some Android specific options towards the bottom of the platform sections which have been covered in a previous section.

Paid DLC
If you are creating paid DLC that needs to be purchased before it can be accessed, then you will first need to setup an In-App Purchase that can be linked to the DLC content for ownership verification. You can find Unity documentation for setting up an Google Play IAP here under the `Add In-App Purchases` topic.

Once you have created your IAP, take note of the product name created via the Google Play console, and insert that into the `Play Store IAP Name` field of the DLC as shown above.

Android Google Play DRM is now setup and ready for use. Building your project will now also build all DLC setup for the platform and will package it with `build.gradle`.

Local Directory (Desktop)

If DRM verification is not required to check whether a user owns the content, then Ultimate DLC Toolkit includes local directory DRM emulation. It means that the standard DLC API can be used at runtime for loading DLC content that is simply placed inside a specific folder of the game installation. It is mainly intended for easier testing of DLC content in the editor environment but can additionally be used at runtime by mounting a specific folder as a `DLC` folder.

Todo – folder setup

Custom DRM Provider

If the built in DRM providers are not suitable for your use case/intend distribution service, then it is possible to create your own DRM provider so that you can still use the standard DLC API as normal. It is just a case of implementing a C# interface, and you can find a code sample for that here:

[Custom DRM Provider Code Sample]

List Available DLC

The DRM API can be used to fetch all available DLC content from the service provider such as Steam. Using this approach, it is possible to add new DLC content via the online service (Steam Partner in this example) without requiring an update to the base game. This is possible because all available DLC unique keys can be provided in real time when the game requests it, and Ultimate DLC Toolkit offers a unified API for loading and initializing game assets into your game, so the same code can be used to iterate over all installed DLC content.

Listing remote available DLC is only available as an async operation, due to the fact that it will usually require and API call to a service which may take some time to complete. You can check out the code sample linked below which will fetch a list of DLC unique keys (Includes all DLC whether owned by the user or not).

Ultimate DLC Toolkit also allows you to list all local DLC content that is available. This will only list content that is known to the game at build time (Ie. DLC content that was built before the game was built), but this has the benefit of being able to list the DLC unique keys instantly without waiting for an async request.

List Available DLC Code Sample

Check Ownership

The DRM API can be used at runtime to check whether the user owns a specific DLC so that your game logic can act accordingly. It is likely that you will only want to load DLC that the user has actually purchased, or even if the DLC is free it is still good practice to check via DRM that the user has added it to their account/checked out that free content via the associated online store such as Steam.

Checking for ownership is only available as an async operation, due to the fact that it will usually require and API call to a service which may take some time to compete. You can check out the following code sample to check for ownership, assuming you know the DLC unique key ahead of time.

Check Ownership Code Sample

Next topic is building DLC content in the editor.

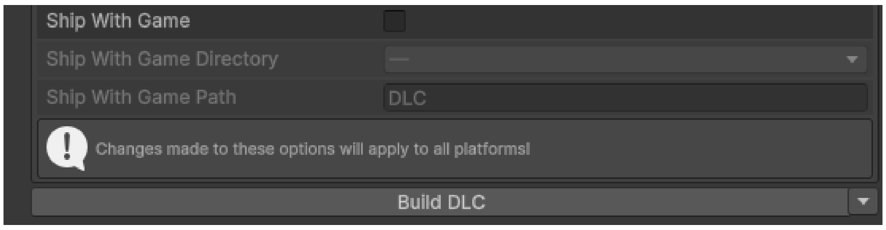
Build DLC

Building DLC content in Ultimate DLC Toolkit is very simple in most cases, but you have a couple of options to consider. Ultimate DLC Toolkit supports individual building of DLC Profiles, and batch building of DLC Profiles as a single operation, making it easy to build all DLC in the project via a single click. There is also a fully documented build API that can be called from any editor script if more advanced integration is required.

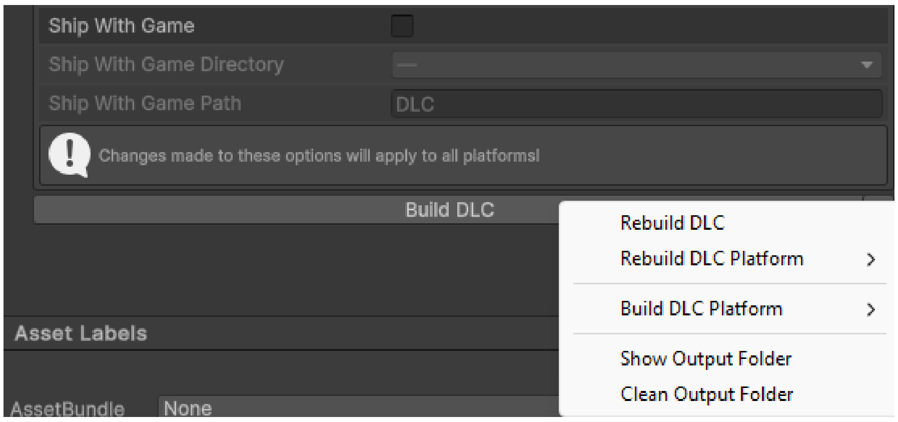
Note: *The first DLC build performed will usually take significantly longer than any subsequent builds. This is correct behaviour since Ultimate DLC Toolkit uses incremental building to speed up subsequent build requests.*

Build DLC Profile

With a DLC Profile asset selected in the project window, you can start a build of the DLC content for all enabled platforms by clicking the `Build DLC` button towards the bottom of the inspector window:


Ship With Game
Ship With Game Directory —
Ship With Game Path DLC
Changes made to these options will apply to all platforms!

Additionally, you can have more build and utility options available by selecting the drop-down arrow to the right of the `Build DLC` button. This will bring up a context menu where you can choose a more concise build option if you only wanted to build for a specific platform, and you can also choose a `Rebuild DLC` option which will forcefully rebuild the DLC content from scratch, disregarding any incremental build data. Additionally, you can quickly show and clean the output folder where the DLC content would be created.


Ship With Game
Ship With Game Directory —
Ship With Game Path DLC
Changes made to these options will apply to all platforms!

Build DLC Batch

For batch building of DLC you can use the `Tools` menu to start a build depending upon the option selected. The following build options are available via the menu:

Tools -> DLC Toolkit -> Build DLC -> (Platforms)*

Tools -> DLC Toolkit -> Build Selected DLC -> (Platforms)*

- **Build DLC Menu Group:** This menu group will start building all enabled DLC Profiles in the project. From this group you can select `All Platforms` to build all DLC content for all enabled platforms, or you can choose only a specific platform if you only want to build all DLC in the project for a particular build target such as Windows only, in order to speed up build time. This option allows you to quickly and easily build all applicable DLC Profiles in the project as a single operation.
- **Build Selected DLC Menu Group:** Similar to the above option however this selection is contextual and will take the current Unity editor selection into account, meaning that only DLC Profile assets that are selected in the project window will be built. Similarly, you can select whether you would like to build for `All Platforms` or alternatively you can choose a specific platform if required.

Build Settings

Ultimate DLC Toolkit includes a few global settings that can change the build behaviour as required.

There are a few configuration options which can be toggled on or off via the tool's menu, the first of which is the `Force Rebuild` option. This option will determine if DLC content is forcefully rebuilt from scratch when batch building via the tool's menu. It is recommended that this option remains disabled in most cases for faster build times:

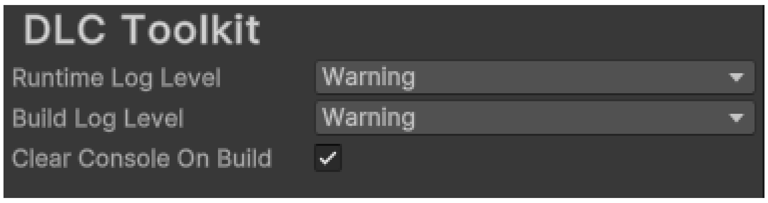
Tools -> DLC Toolkit -> Build Config -> Force Rebuild -> (Enabled/Disable)*

The next option `Compilation` is only related to scripting, so if your DLC content does not include any C# scripts then it has no relevance. From this option you can choose whether the scripts are compiled in `Debug` or `Release` mode:

Tools -> DLC Toolkit -> Build Config -> Compilation -> (Debug/Release)*

Additionally, there are some utility options which can be edited from the project settings window, or by selecting the following menu:

Tools -> DLC Toolkit -> Settings


DLC Toolkit

Runtime Log Level	Warning
Build Log Level	Warning
Clear Console On Build	✓

- **Build Log Level:** Determines how much information will be logged to the Console during the build. Available options are:
  - **Error:** Only errors or exceptions generated during the build will be logged to the console.
  - **Warning:** Exceptions, errors and warnings will be logged to the console during the build.
  - **Info:** Exceptions, errors, warnings and verbose information messages will be logged to the console during the build.
- **Clear Console On Build:** When enabled, DLC Toolkit will clear the Unity console when starting to build any DLC content. It means that only information related to the current build will be shown in the console, making it easier to see any issues.

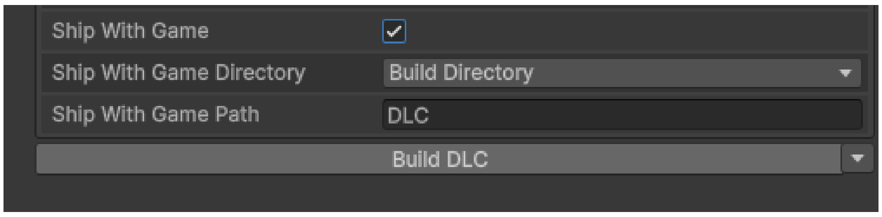
Ship With Game

Ship with game is a Unity player build integration step that will attempt to include any DLC content with the `Ship With Game` option enabled as part of the built game. It works by detecting when the Unity standalone build pipeline is launched to build your game, and then it will automatically build any DLC with this option enabled as part of the same process. If both the player build and DLC build are successful, then Ultimate DLC Toolkit will then package the built DLC into a preset location in the game build directory. This is all fully automatic and there is no longer a requirement to manually build your DLC if you intend to ship the DLC content with your base game.

Note: *The `Ship With Game` feature can be used to distribute DLC content with the base game if distribution via an online service is not required. You can however still enforce DRM for this type of DLC content so that it can only be accessed if the user has added or purchased the content via a DRM provider such as Steam, despite the fact that the DLC content is distributed with the base game. Check out the Load DLC section for DRM management which applies to both dynamically installed and pre-installed DLC equally.*

There are a few options relating to the `Ship With Game` feature that can be edited via the DLC Profile asset on a per platform basis. You will first need to enable the feature to be able to edit the additional options.

Note: *The `Ship With Game` setup can be configured differently for each build platform the DLC is targeting.*


Ship With Game ✓
Ship With Game Directory: Build Directory
Ship With Game Path: DLC

Available options are:

- **Ship With Game:** Enable the ship with game build step, so that the DLC will be built and packaged with the base game automatically when building a Unity player.
- **Ship With Game Directory:** Allows you to select the directory relative to the build base game directory where the DLC content will be placed. Available options are:
    - **Build Directory:** The DLC content will be placed in the root build directory in the `Ship With Game Path` sub folder, for example:

      ```
      <build_directory>/DLC/New DLC 1.dlc
      ```

    - **Streaming Assets:** The DLC content will be placed in the streaming assets build directory in the `Ship With Game Path` sub folder, for example:

      ```
      <build_directory>/<project_name_Data>/StreamingAssets/DLC/New DLC 1.dlc
      ```

      Note: *This example is specific to desktop builds, and the streaming asset's location for other platforms will vary.*

- **Ship With Game Path:** The sub folder relative to the `Ship With Game Directory` where the DLC content will be placed. It is possible to create any number of sub folders using the `/` separator if required for better organisation.

Build API

If you are an advanced user or require more control over the DLC build pipeline, then there is a fully documented public API available which can allow you full control for building DLC content. There is also the option of registering to receive various build events at different stages of the DLC build process from an editor script. This can be useful if you want to run custom editor code when DLC is built.

To call the DLC build pipeline manually offering you full control over the process, check out the following code sample:

Call DLC Build Pipeline Code Sample

To subscribe to various build events that are called during the DLC build process, check out the following code samples:

Receive Pre-Build Event Code Sample

Receive Post-Build Event Code Sample

Receive Pre-Build Platform Event Code Sample

Receive Post-Build Platform Event Code Sample

For further reference there is an additional dedicated scripting reference guide included with the asset in .pdf format. This document contains a detailed overview of all public facing API's that are part of Ultimate DLC Toolkit, for both runtime and editor aspects. You can find it in the main asset directory:

Assets/DLC Toolkit/ScriptingReference.pdf

Next topic is loading DLC content into your game.

Load DLC

Once you have created and built some DLC content, your next step will be to load it into your game so that the assets included can be used while in play mode or as part of a standalone game. There are may ways you can load DLC content into your game which will be covered in this section.

Note: *Async loading of DLC content is recommended in most cases to avoid blocking the game thread, especially for larger DLC which may take more time to be loaded into memory.*

Load Locally

The most basic but probably least useful option in a real-world application would be to load DLC content from a location known at build time. For example, the DLC content is expected to exist in a certain folder of the game install directory.

In that case then you can make use of the various `LoadFrom` methods which accept the file path location of the DLC content. Check out the following samples for loading the DLC in this manner:

Load Local DLC Code Sample

Load Local DLC Async Code Sample

Load DRM

Loading DLC via the DRM API is highly recommended as it allows for proper ownership verification via a service such as Steam API if setup, also supports listing available DLC, and loading DLC without specifying a path as it will be determined by the DRM service in use as to where the DLC will be installed.

All DRM load methods require the DLC unique key rather than a path, and it is highly recommended that async loading is used since the DRM in use may need to make requests to service APIs as part of the load request which could take some time to complete. You can specify the DLC unique key manually when issuing the load request, however it is recommended that you use the DRM API to list all DLC contents that are available to the game, or alternatively use the locally listed approach instead:

List Available DLC

Note: *When attempting to load DLC via the DRM API, ownership checks are incorporated into the load request, so the DLC will only be loadable if the user has purchased or added the DLC to their account. If the DLC is not available to the user, then the load request will simply fail. Additionally, all async versions accept an extra parameter `installOnDemand` which when enabled will incorporate the installation of the DLC into the load request if it is owned but not yet installed (Only available for async methods due to the fact that on demand installation can take any amount of time to complete).*

Check out the following samples for loading DLC using the DRM API:

Load DLC Code Sample

Load DLC Async Code Sample

Editor Test Mode

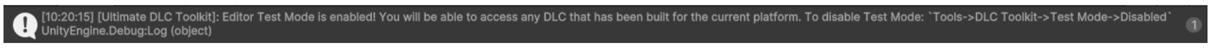
To make testing DLC content in the editor easier, Ultimate DLC Toolkit is able to simulate DRM management using the unified API. With test mode enabled, using the DRM API will redirect calls to the Unity project which will be able to perform the following actions:

- **List DLC content that is available in the project**: Requirements are that the DLC profile is enabled, is setup to build for the equivalent platform (Windows on Windows Editor for example).
  If you are targeting mobile, web or console, then you may not have a platform profile setup to match your editor platform. It may still be worthwhile settings one up for quick testing of the DLC in editor even if you will not be intending to target desktop platforms, otherwise testing will only be possible by building the project.

  Note: *All DLC content will be listed if it meets the above requirements, even if the DLC has not yet been built for the equivalent platform.*

- **Check availability**: The DLC will always be considered available and owned as long as it has been built (The built DLC exists at the output location). If the DLC has not yet been built, then it will simply be considered as `Not Owned`.
- **Stream content**: The DLC will be available for loading if both the above points are true. You can use the standard loading API (Any load method accepting a DLC unique key) to load the DLC using DRM as if it was setup via any other DRM service such as Steam.

Editor test mode is enabled by default, and you will always see a helpful message logged to the Unity editor console when entering play mode if that is the case:



```
[10:20:15] [Ultimate DLC Toolkit]: Editor Test Mode is enabled! You will be able to access any DLC that has been built for the current platform. To disable Test Mode: `Tools->DLC Toolkit->Test Mode->Disabled`
UnityEngine.Debug:Log (object)
```

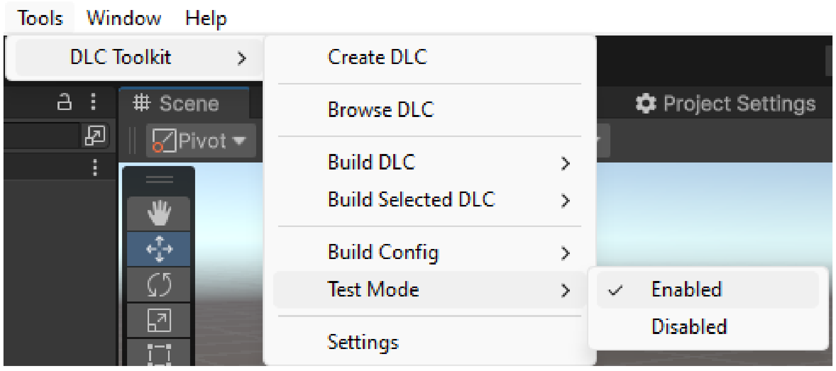
Editor test mode as the name suggests is only available in the Unity editor, and at runtime Ultimate DLC Toolkit will fallback to use the default setup DRM provider for the platform if available. You can however easily enable or disable test mode at any time if you want to test runtime DRM functionality in the editor.

Enabling or disabling test mode is done via the tools menu:

`Tools -> DLC Toolkit -> Test Mode -> Enabled`

`Tools -> DLC Toolkit -> Test Mode -> Disabled`

The menu option will also have a check mark placed next to the active selection, so you can easily see whether test mode is on or off:



In addition, you can easily check whether test mode is active from code if required, so that you can adjust your game logic accordingly.

Check Test Mode Code Sample

Batch Loading

If you have multiple DLC available for your game, then a normal use case is to load all DLC owned by the user when the game starts. To aid with this and similar scenarios, Ultimate DLC Toolkit supports batch loading which allows any number of DLC content load requests to be processed in parallel as part of the same async request.

Batch loading has the benefit of allowing all load requests to run in parallel, reducing the overall load time significantly when compared with sequential loading. Additionally, each load request can be tracked by a single async operation, meaning that overall progress can be calculated as a total value. It also provides a means to easily determine which load operations were successful and which were not.

Note: *Batch loading is only available in async variants, as generally the load times will take longer overall. Usually, the overall load time will be approximately the time taken to load the largest DLC content, plus some additional minor overhead for async management and batch management.*

Check out the following sample for batch loading:

Batch Load Code Sample

Install On Demand

Often the DLC content will be automatically installed to the user's device as soon as they have purchased or added it to their account. Some services however such as Steam and Google play can support installing on demand at the request of the application.

Ultimate DLC Toolkit has support for requesting on demand installation to have finer control over when DLC content is available on the user's device. You will need to consult the documentation for the DRM provider (Steam partner / Google Play / etc.) about how to achieve this using their services, but the actual installation request can be made via the unified API once configured.

Note: *Many of the DRM async load methods include an additional `installOnDemand` parameter. You can pass `true` to those requests and the install on demand request will be incorporated into the overall load request if the DLC is owned by the user but has not yet been downloaded or installed.*

Check out the following sample to see how you can request installation at any time, and how you can run code after the installation has completed or failed:

Install On Demand Code Sample

Access Metadata

Once you have loaded DLC content into memory, you will be able to access the metadata associated with that DLC. This is mainly user data that you input into the DLC Profile asset, but also contains versioning information, and information about included content.

Check out the following example to see which metadata you can have access to at runtime:

Access Metadata Code Sample

Additionally, custom metadata can be included in the DLC if required. The topic of custom metadata is covered in detail in a previous section:

Custom Metadata In DLC

Load Assets

Once DLC content is loaded into memory, you have the option of loading the `Shared Assets` that are included as part of the DLC at any time. Shared assets are simply any asset type other than Scenes or Scripts. For example, Prefabs, Textures, Materials, Audio Clips and so on are all considered shared assets.

Ultimate DLC Toolkit offers a complete API for performing asset reflection, which means listing asset contents included in various ways, such as assets in a particular folder, assets of a specified type, assets with a certain extension and so on, plus this reflection can be performed without requiring any assets to be loaded. That way you can use the API to find only the assets you are interested in, and then load those assets on demand when required.

Check out the following example for loading DLC assets at runtime:

Load DLC Assets Code Sample

Load Scenes

Ultimate DLC Toolkit offers access to scene assets separately, as internally there are differences between shared and scene assets. Scenes can be found again using a similar metadata reflection system for listing scenes included in a specific folder for example, so the differences are mainly organisations from the perspective of the runtime API.

Scenes are also loaded slightly differently than shared assets, since there is the possibility of additive loading, plus only activating the loaded scene on demand at a later time, useful if you intend to show a loading screen for example.

Check out the following example for loading DLC scenes at runtime:

Load DLC Scenes Code Sample

Unload DLC

Once your game has finished using a particular DLC, it can be unloaded at any time in order to free up memory and other resources. Depending upon the type of DLC and the intended usage, this may or may not be useful, as in many cases it is perfectly valid for the DLC to be loaded at the start of the game and survive the whole session if assets are in constant use.

As an example, if you are creating a character dresser system where each available clothing item is part of its own DLC (Perfectly achievable using Ultimate DLC Toolkit, and a valid way to manage items and resources), then it would be ideal to only keep the clothing items applied to the character in memory, and all other items can be unloaded until required.

Check out the following example for unloading any given DLC content at runtime:

Unload DLC Code Sample

Batch Unloading

Ultimate DLC Toolkit supports batch unloading in a similar way as batch loading. It allows you to specify multiple DLC contents that should be unloaded from memory, and Ultimate DLC Toolkit will unload all contents as part of the same operation. Additionally, it is possible to batch unloads asynchronously, meaning that each unload can occur in parallel taking less time overall.

Check out the following example for batch unloading multiple DLC contents at runtime:

Batch Unload DLC Code Sample

Report a Bug

At Trivial Interactive we test our assets thoroughly to ensure that they are fit for purpose and ready for use in games, but it is often inevitable that a bug may sneak into a release version and only expose itself under a strict set of conditions.

If you feel you have exposed a bug within the asset and want to get it fixed, then please let us know and we will do our best to resolve it. We would ask that you describe the scenario in which the bug occurs along with instructions on how to reproduce the bug so that we have the best possible chance of resolving the issue and releasing a patch update.

http://trivialinteractive.co.uk/bug-report/

Request a Feature

Ultimate DLC Toolkit was designed as a complete DLC content authoring and management tool, however if you feel that it should contain a feature that is not currently incorporated then you can request to have it added into the next release. If there is enough demand for a specific feature, then we will do our best to add it into a future version. Please note, requested features should fall within the scope of the asset and unrelated or overreaching features will not be added.

http://trivialinteractive.co.uk/feature-request/

Contact Us

Feel free to contact us if you are having trouble with the asset and need assistance. Contact can either be made by the contact options on the asset store or via the link below.

Please attempt to describe the problem as best you can so we can fully understand the issue you are facing and help you come to a resolution. Help us to help you :-)

http://trivialinteractive.co.uk/contact-us/

Alternatively, you can join us on Discord!