

Developer

/ Documentation

/ Unreal Engine ▾

/ Unreal Engine 5.4 Documentation

/ Testing and Optimizing Your Content

/ Zen Loader

# Zen Loader

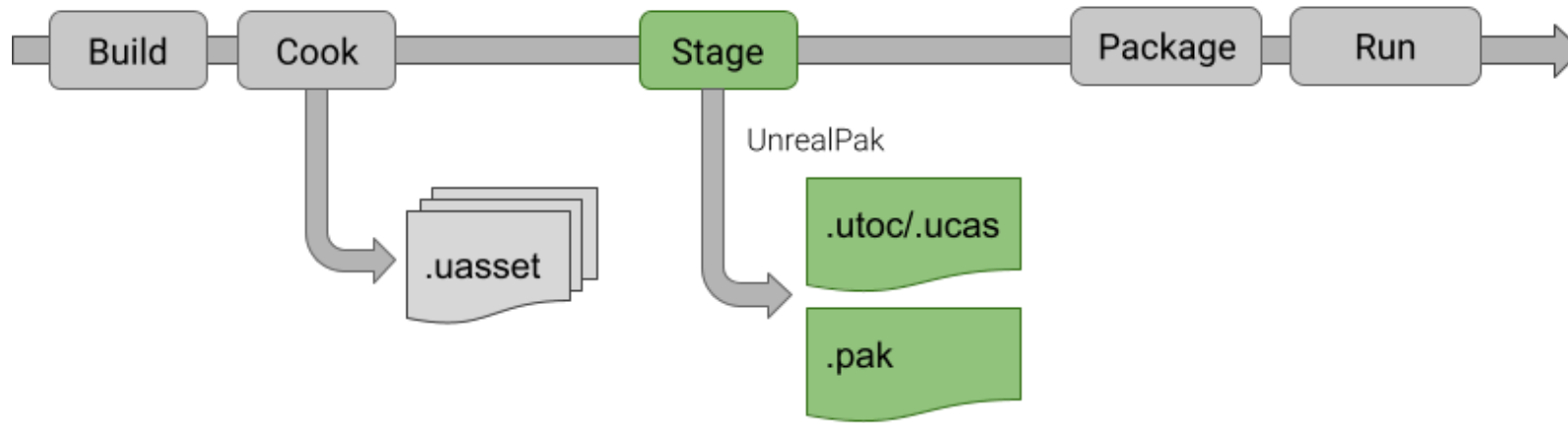
An Overview of Unreal Engine 5's runtime loader.



**Zen Loader** is the new runtime loader for **Unreal Engine 5 (UE5)**. Zen Loader reduces CPU overhead by using an optimized package and object dependency graph that is computed offline during the staging phase. Instead of packing all assets into one or more .pak file(s), Zen Loader packages all package data, bulk data and shader data into a set of `.utoc` and `.ucas` container files.

The `.pak` files are used for loose files that are accessed less frequently. When a Pak file is mounted, then the corresponding container file will be mounted too. The `.utoc` file describes the container file, including chunk size and

offset, compression format, and whether the chunks are encrypted. The `.uca` file contains the actual data. Using container files avoids file system abstractions and provides the loader with the ability to find data chunks with minimal CPU overhead using a new I/O abstraction layer called the I/O dispatcher. It uses platform specific backends to benefit from hardware capabilities and APIs.



## Using the Zen Loader

The container files are produced by default during staging with the **Unreal Automation Tool(UAT)**, and the presence of these container files will activate the Zen loader in runtime.



See the [Unreal Automation Overview](#) for additional information on how to run commands for your project.


The **Event Driven Loader(EDL)** runtime loader is deprecated and will be removed in future releases, but you can still use it with UAT during staging by using the command:

```
1 -SkipIoStore  
2
```

 Copy full snippet

Alternatively, you can navigate to your `C:\MyProject\Config\DefaultGame.ini` file, then in the [ProjectPackagingSettings] section add the following line:

```
1 [/Script/UnrealEd.ProjectPackagingSettings]  
2 bUseIoStore=False  
3
```

 Copy full snippet



Setting the `bUseIoStore` boolean to the value of `False` will tell the Zen Loader to not use IOStore.


## Limitations

The **I/O Dispatcher** is a new I/O abstraction layer that loads data addressed by a chunk ID, this means that the data can't be loaded or searched for using the standard file/directory format that is based in the I/O API. The [Asset Registry](#) supports similar capabilities for searching for assets in directories. When using the Zen loader, it is no longer possible for Bulkdata to load inline data after the initial serialization has finished, which means that once data is unloaded it cannot be

retrieved again. Any data set that requires frequent access needs to be stored as non-inline data during the cooking phase. The Zen loader performs offline preprocessing of object dependencies at stage time. This prevents any required package redirects at runtime, and systems like core redirects and delegate resolvers are ignored during loading.

# Inspecting the contents of container files

In the table below, you can view the commands supported by UnrealPak that are compatible for container files.

Command	Description
-list	Lists the chunks in a container.
-diff	Compares the contents of two containers.
-extract	Extracts the contents of a container.
IoStore -describe	<div>Provides detailed information on the container's dependencies, packages, and import and exports. The describe command takes the path to the <code>global.ucas</code> file and uses an output format similar to the one from the PkgInfo commandlet for <code>.uasset</code> files:</div> <div><pre>UnrealPak.exe IoStore -Describe=My</pre></div> <div> Copy full snippet</div>

# Troubleshooting

Below you will find some helpful troubleshooting commands to assist you in identifying errors or warnings during each stage.

## Loading


During the Loading stage, the Zen loader uses the **LogStreaming** log channel and Verbose/VeryVerbose logging can be temporarily enabled on the command line by using the command:

```
1 -LogCmds="LogStreaming veryverbose"  
2
```

 Copy full snippet

For larger projects, logging can be time consuming. You can filter the logs on specific packages by setting your selected package names or package IDs with the command:

```
1 -s.VerbosePackageNames="/Game/PackageA/Game/PackageB0xABCD1234ABCD1234"  
2
```

 Copy full snippet

If you are running with a debugger attached you can use the command:

```
1 -s.DebugPackageNames  
2
```

 Copy full snippet

This will provide an automatic breakpoint at certain load phases of the specified packages.

## Staging


During the staging phase, **UAT** will generate a command list of one or more response files that specify all the packages stored in each container by using the command:

```
1 -ScriptsForProject= "C:\MyProject\MyProject.uproject"  
2
```

 Copy full snippet

You can locate this command, and others from the **UAT** log file located in your directory path:

```
1 c:Engine\Programs\AutomationTool\Saved\Logs\Log.txt  
2
```

 Copy full snippet

## Zen Loader Internals

The Zen loader is based on the **Event Driven Loader(EDL)** and requires the same output types from the cooker such as name tables, import / export maps, and preload dependencies. The legacy EDL runtime logic has been moved offline and is produced during the staging phase.



This optimization to offline preprocessing reduces the amount of time it takes for the runtime package dependency tracking to process from seconds down to milliseconds.

Additional changes Include:

- Package summaries, export blobs, bulk data and shaders are placed in container files.
- The most important package metadata and all package interdependencies are collected into a package store that is indexed by package IDs.
- The .uasset package header is transformed to an optimized package summary.
  - Import maps with outer chains have been removed and replaced with direct references by export object hashes.
  - Name tables have a new optimized batch format.
  - Preload dependencies have been flattened into per-package export bundle nodes, each specifying a sequence of `CreateExport` and `SerializeExport` calls and dependencies to other export bundle nodes.

## Zen Store

Zen Store is an experimental feature that enables you to store cooked assets to a local storage server instead of loose files into the local filesystem. Refer to the [Zen Store](#) for additional information.