Shader Debugging Workflows

A workflow and properties reference for shader debugging.

The primary route to debugging shaders is to enable the console variable `r.Shaders.Symbols`. This will trigger the engine to prepare shaders for debugging by platform-specific programs like RenderDoc and PIX.

You can override this console variable, along with the others listed below, on a per-platform basis. This is important to keep extra shader compilation under control.

Shader Debugging Workflows

Three shader debugging workflow examples are described below. In each case, you need to edit the engine **configuration file** for the platform you want to debug shaders on. For example, if you are debugging shaders for Android, you need to add console variables to the **AndroidEngine.ini** file.

Example: Using a Platform-Specific Shader Debugger

You want to use a platform-specific shader debugger to debug the shaders in a cooked game on that platform.

Add the following text to your **[Platform]Engine.ini** file:

```
[ShaderCompiler]
r.Shaders.Symbols=1

```

Copy full snippet

Cook your game for that platform using the Editor or Unreal Build Tool (UBT). Create a GPU capture on that platform. If asked for a shader symbol path, direct it at the folder `Path/To/My/Project/Saved/ShaderSymbols/Platform`.

Example: Have Only the Build Machine Write Symbols to a Zip File

You want to always build shader symbols for a platform and only have the build machine write them to a `.zip` file.

Add the following text to your **[Platform]Engine.ini** file:

```
[ShaderCompiler]
r.Shaders.GenerateSymbols=1

[ShaderCompiler_BuildMachine]
r.Shaders.WriteSymbols=1
r.Shaders.WriteSymbols.Zip=1

```

Copy full snippet

Now the build machine generates the following `.zip` file: `Path/To/My/Project/Saved/ShaderSymbols/Platform/ShaderSymbols.zip`.

Example: Debugging Shaders Locally

You have a graphics programmer working on a project with the same setup as Example 2, and they want to debug shaders locally.

In this case, the user should edit the **[Platform]Engine.ini** locally and add the following:

```
[ShaderCompiler]
r.Shaders.WriteSymbols=1

```

Copy full snippet

When they cook your project, the shaders will all be written loosely to `Path/To/My/Project/Saved/ShaderSymbols/Platform`.

Console Variable Summary

These are the console variables available for shader debugging.

Console Variable	Description
`r.Shaders.Symbols`	Enables debugging of shaders by generating symbols. If the platform requires external symbols, they are written to disk; otherwise, they are stored inside the shader data loaded at runtime. Can be overridden on a per-platform basis.
`r.Shaders.ExtraData`	Generates shader names and any other per-platform extra shader data. You can override this variable on a per-platform basis.
`r.Shaders.GenerateSymbols`	Generates symbols but does not write them to disk. You can override this variable on a per-platform basis.
`r.Shaders.WriteSymbols`	If the platform supports external symbols, write them to disk if they were generated. You can override this variable on a per-platform basis.

Console Variable	Description
`r.Shaders.SymbolPathOverride`	If the platform supports external symbols, you can use this console variable to override the location they are written to.
`r.Shaders.WriteSymbols.Zip`	If the platform supports external symbols and they need to be written to disk, write them out into a single `.zip` file rather than individual files.
`r.Shaders.AllowUniqueSymbols`	If the platform supports external symbols, generate the symbol file names from the resulting shader and not its source files. We do not recommend enabling this as it can dramatically increase the size of your symbols.

Platform Overrides

You can override shader symbol console variables on a per-platform basis by adding special sections to the **[Platform]Engine.ini** files.

For example, if you want to override shader symbol console variables on the Android platform, add the following text to your **AndroidEngine.ini**:

```
[ShaderCompiler_BuildMachine]
Console variables go here.

```

Copy full snippet

Changes from UE4

In Unreal Engine 5, the console variables used for debugging shaders have changed. The table below highlights the old UE4 console variable and the new name being used in UE5. You need to update your configuration files that use these console variables when migrating your project to UE5 to continue using the generated data and debug shaders.

Old Name	New Name	Description
`r.Shaders.KeepDebugInfo`	`r.Shaders.Symbols`	Enables debugging of shaders by generating symbols and writing them to disk for consoles, PC symbols are still stored inline.
See note.	`r.Shaders.ExtraData`	Generates shader names and any other "extra" shader data.
`r.Shaders.PrepareExportedDebugInfo`	`r.Shaders.GenerateSymbol`	Generates symbols but does not write them to disk (note: symbols are stored in the DDC)
`r.Shaders.ExportDebugInfo`	`r.Shaders.WriteSymbols`	Writes symbols to disk if they have been generated.
`r.Shaders.AllowUniqueDebugInfo`	`r.Shaders.AllowUniqueSymbols`	Generates symbol associations based on shader source (off by default).
`r.Shaders.ExportDebugInfo.Zip`	`.Shaders.WriteSymbols.Zip`	Enables writing all symbols to disk as a single `.zip` file.

`r.Shaders.KeepDebugInfo` was split into `r.Shaders.Symbols` and `r.Shaders.ExtraData` to remove the changes to runtime shader data when only symbols are required. This is particularly useful for platforms that support exported debug info since this allows you to generate symbols for shipping builds without changing the final shader data.