

# Console Variables and Commands

Overview of the Console Manager and implementation details for creating console variables.



A **Console Command** is a string sent to the engine, often typed in by the user at the in-game console, that the engine recognizes and can react to in some way. For example, a console commands can trigger a console / log response, change a variables internal state, and so on. A **Console Variable** can be used to store state information that can be viewed or changed through the console. The in-game console supports auto-completion and enumeration for console commands and console variables that have been registered with the **Console Manager**. The console manager also provides a registry for you to see all registered console commands and console variables as well as information about their purpose. For these reasons, you should avoid the old `Exec` interface and instead define and register your console variables and console commands through the console manager.

## What is a Console Variable?

A Console Variable is a variable of a simple data type (for example, `float`, `int32`, `FString`) that has an engine-wide state. The user can read and write to the state. The Console Variable is identified by a unique name, and the in-game console will assist the user with auto-completion while typing into the console. Some examples:

User console input	Console output	Description
<code>MyConsoleVar</code>	<code>MyConsoleVar = 0</code>	The current state of the variable is printed into the console.
<code>MyConsoleVar</code> <code>123</code>	<code>MyConsoleVar = 123 LastSetBy:</code> <code>Constructor</code>	The state of the variable is changed and the new state printed into the console.
<code>MyConsoleVar ?</code>	<code>Possibly multi line help text.</code>	Print the console variable help text into the console.

## Console Variable and Console Command Reference

For a list of available console variables, see the [Console Variable Reference](#) page.

For a list of available console commands, see the [Console Command Reference](#) page.

## Creating / Registering a Console Variable

All Console Variables are registered when the engine is created. The following example has been taken from engine code:

```

1 static TAutoConsoleVariable<int32> CVarRefractionQuality(
2 TEXT("r.RefractionQuality"),
3 2,
4 TEXT("Defines the distortion/refraction quality, adjust for quality or
   performance.\n")
5 TEXT("<=0: off (fastest)\n")
6 TEXT(" 1: low quality (not yet implemented)\n")
7 TEXT(" 2: normal quality (default)\n")
8 TEXT(" 3: high quality (e.g. color fringe, not yet implemented)"),
9 ECVF_Scalability | ECVF_RenderThreadSafe
10 );

```

 Copy full snippet

Here we register a Console Variable of the type `int32`, with the name `r.RefractionQuality`, the default value of `2` and some multi-line help text and flags. The help text is shown when the user types "?" after the Console Variable's name. The names and descriptions of flags available (from the `EConsoleVariableFlags` enumerated type) can be found on the [EConsoleVariableFlags API Reference page](#).



There are other entries in the `EConsoleVariableFlags` enumerated type, but all other flags are intended for internal use only.

If needed, you also can generate a console variable inside a function:

```
1 IConsoleManager::Get().RegisterConsoleVariable(  
2 TEXT("r.RefractionQuality"),  
3 2,  
4 TEXT("Defines the distortion/refraction quality, adjust for quality or  
   performance.\n")  
5 TEXT("<=0: off (fastest)\n")  
6 TEXT(" 1: low quality (not yet implemented)\n")  
7 TEXT(" 2: normal quality (default)\n")  
8 TEXT(" 3: high quality (e.g. color fringe, not yet implemented)"),  
9 ECVF_Scalability | ECVF_RenderThreadSafe  
10 );
```

 Copy full snippet

`IConsoleManager::Get()` is the global access point. There you can register a Console Variable or find an existing one. The first parameter is the name of the Console Variable. The second parameter is the default value, and depending on the type of this constant, a different Console Variable type is created: int, float, or string (!FString). The next parameter defines the Console Variable help text.

It is also possible to register a reference to an existing variable. This is convenient and fast but bypasses multiple features (for example, thread safety, callback, sink, cheat) so we suggest to avoiding this method. Here is an example:

```

1 FAutoConsoleVariableRef CVarVisualizeGPUSimulation(
2 TEXT("FX.VisualizeGPUSimulation"),
3 VisualizeGPUSimulation,
4 TEXT("Visualize the current state of GPU simulation.\n")
5 TEXT("0 = off\n")
6 TEXT("1 = visualize particle state\n")
7 TEXT("2 = visualize curve texture"),
8 ECVF_Cheat
9 );
10

```

 Copy full snippet

Here they type is deducted from the variable type.


## Getting the State of a Console Variable

Getting the state of a Console Variables created with **RegisterConsoleVariableRef** can be done efficiently by using the variable that it was registered with. For example:

```

1 // only needed if you are not in the same cpp file
2 extern TAutoConsoleVariable<int32> CVarRefractionQuality;
3
4 // get the value on the game thread
5 int32 MyVar = CVarRefractionQuality.GetValueOnGameThread();

```

 Copy full snippet

Using Getter functions (that is, `GetInt()`, `GetFloat()`, `GetString()`) to determine a Console Variable's state results in a slightly slower implementation (virtual function call, possibly cache miss, and so on). For best performance you should use same type the variable was registered with. To get the pointer to the variable, you can either store the return argument of the register function or call **FindConsoleVariable** just before you need the variable. For example:

```

1 static const auto CVar =
    IConsoleManager::Get().FindConsoleVariable(TEXT("TonemapperType"));
2 int32 Value = CVar->GetInt();

```

 Copy full snippet


The static there ensures the name search (implemented as map) is only done the first time this code is called. This is correct as the variable will never move and only gets destructed on engine shutdown.

## How to Track Console Variable Changes

If you want to execute some custom code if a Console Variable changes, you have 3 methods you can choose from.

Often the simplest method is the best: You can store the old state in your subsystem and check each frame if they differ. Here you control when this happens very freely; for example, on the render thread, game thread, or streaming thread, before / after tick or rendering. When you detect the difference, you copy the console variable state and do your custom code. For example:

```
1 void MyFunc()
2 {
3     int GBufferFormat = CVarGBufferFormat.GetValueOnRenderThread();
4
5     if(CurrentGBufferFormat != GBufferFormat)
6     {
7         CurrentGBufferFormat = GBufferFormat;
8
9         // custom code
10    }
11 }
```

 Copy full snippet

You also can register a console variable sink. For example:

```
1 static void MySinkFunction()
2 {
3     bool bNewAtmosphere = CVarAtmosphereRender.GetValueOnGameThread() != 0;
4
5     // by default we assume the state is true
```

```

6 static bool GAtmosphere = true;
7
8 if (GAtmosphere != bNewAtmosphere)
9 {
10 GAtmosphere = bNewAtmosphere;
11
12 // custom code
13 }
14 }
15
16 FAutoConsoleVariableSink
    CMyVarSink(FConsoleCommandDelegate::CreateStatic(&MySinkFunction));

```

 Copy full snippet

The sink is called at a specific point on the main thread before rendering. The function does not get the console variable name/pointer as this often would lead to the wrong behavior. If multiple console variables (for example, `r.SceneColorFormat`, `r.GBufferFormat`) should all trigger the change, it is best to call the code after all have been changed, not one after another.

The last method, using the callback, you should avoid as it can cause problems if not used carefully:

- A cycle can cause a deadlock (we could prevent the deadlock but which callback to favour is not clear).
- The callback can come back at any point in time whenever **!Set()** is getting called. Your code has to work in all cases (during init, during serialization). You can assume it is always on the main thread side.

We do not recommend using this method unless you cannot solve it with the other methods mentioned.

For example:

```

1 void OnChangeResQuality(IConsoleVariable* Var)
2 {
3     SetResQualityLevel(Var->GetInt());
4 }
5
6 CVarResQuality.AsVariable()

```

```
7 -  
>SetOnChangedCallback(FConsoleVariableDelegate::CreateStatic(&OnChangeResQuali  
8
```

 Copy full snippet

## Intended Console Variable Behavior and Style

- Console variable should reflect the user input, not necessarily the state of the system (e.g. !MotionBlur 0/1, some platforms might not support it). The variable state should not be changed by code. Otherwise the user might wonder if they mistyped because the variable does not have the state they specified or they might not be able to change a console variable because of the state of some other variable.
- Always provide a good help explaining what the variable is used for and what values make sense to specify.
- Most console variables are intended for development only so specifying the `ECVF_Cheat` flag early would be a good idea. Even better might be to compile out the feature using defines (for example, `#if !(UE_BUILD_SHIPPING || UE_BUILD_TEST)`).
- The variable name should be as minimal as possible while being descriptive, negating meaning should be avoided (for example, bad names would be !EnableMotionBlur, !MotionBlurDisable, MBlur, !HideMotionBlur). Use upper and lower case to make the name easier to read and consistent (for example, !MotionBlur).
- For indentation, you can assume fixed width font (non proportional) output.
- It is important to register the variable during engine initialization so that auto completion and !DumpConsoleCommands and !Help can work.

Please read `IConsoleManager.h` for find more details on this.

## Loading Console Variables

On engine startup, the state of Console Variables can be loaded from the file **Engine/Config/ConsoleVariables.ini**. This place is reserved for the local developer - it should not be used for project settings. More details can be found in the file itself:

```
1 ; This file allows you to set console variables on engine startup (In
   undefined order).
2 ; Currently there is no other file overriding this one.
3 ; This file should be in the source control database (for the comments and
   to know where to find it)
4 ; but kept empty from variables.
5 ; A developer can change it locally to save time not having to type
   repetitive
6 ; console variable settings. The variables need to be in the section called
   [Startup].
7 ; Later on we might have multiple named sections referenced by the section
   name.
8 ; This would allow platform specific or level specific overrides.
9 ; The name comparison is not case sensitive and if the variable does not
   exist, it is ignored.
10 ;
11 ; Example file content:
12 ;
13 ; [Startup]
14 ; FogDensity = 0.9
15 ; ImageGrain = 0.5
16 ; FreezeAtPosition = 2819.5520 416.2633 75.1500 65378 -25879 0
17
18 [Startup]
19
```

 Copy full snippet

You also can put the settings in **Engine/Config/BasEngine.ini**, for example:

```
1 [SystemSettings]
2 r.MyCvar = 2
3
4 [SystemSettingsEditor]
5 r.MyCvar = 3
6
```

 Copy full snippet

Setting can also come from **Script/Engine.RendererSettings**. These project settings are exposed like this:



```
1 UPROPERTY(config, EditAnywhere, Category=Optimizations, meta=(
2 ConsoleVariable="r.EarlyZPassMovable",DisplayName="Movables in early Z-pass",
3 ToolTip="Whether to render movable objects in the early Z pass. Need to
  reload the level!"))
4 uint32 bEarlyZPassMovable:1;
5
```

 Copy full snippet

Those settings can be changed in the editor UI. Project settings should not intermix with scalability settings (to prevent priority issues).

Other settings can come from the Scalability feature. Look at **Config/BaseScalability.ini** or the Scalability documentation for more info.

## Command line

The command line allows to set console variables, call console commands or exec commands. For example:

```
1 UE4Editor.exe GAMENAME -ExecCmds="r.BloomQuality 12,vis 21,Quit"
2
```

 Copy full snippet

Here we execute 3 commands. Note that setting a console variable this way requires you to omit the '=' you would need in an ini file.

## Priority

Console variables can be overridden from various sources, for example user / editor / project settings, command line, consolevariables.ini, and so on. To be able to reapply some settings (for example, project settings can be changed in the editor UI) while keeping the specified overrides (for example, from the command line), we introduced a priority. Now all settings can be applied in any order.

see IConsoleManager.h:

```

1 // lowest priority (default after console variable creation)
2 ECVF_SetByConstructor = 0x00000000,
3 // from Scalability.ini
4 ECVF_SetByScalability = 0x01000000,
5 // (in game UI or from file)
6 ECVF_SetByGameSetting = 0x02000000,
7 // project settings
8 ECVF_SetByProjectSetting = 0x03000000,
9 // per device setting
10 ECVF_SetByDeviceProfile = 0x04000000,
11 // per project setting
12 ECVF_SetBySystemSettingsIni = 0x05000000,
13 // consolevariables.ini (for multiple projects)
14 ECVF_SetByConsoleVariablesIni = 0x06000000,
15 // a minus command e.g. -VSync
16 ECVF_SetByCommandline = 0x07000000,
17 // least useful, likely a hack, maybe better to find the correct SetBy...
18 ECVF_SetByCode = 0x08000000,
19 // editor UI or console in game or editor
20 ECVF_SetByConsole = 0x09000000,
21

```

 Copy full snippet

In some cases, you might see this log printout:

```

1 Console variable 'r.MyVar' wasn't set (Priority SetByDeviceProfile <
  SetByCommandline)
2
3

```

 Copy full snippet

It might be intended (for example, the command line forces a user setting) or caused by some code issue. The priority is also helpful to see who set the variable the last time. You can get this information when getting the console variable state. e.g.

```

1 > r.GBuffer
2
3 r.GBuffer = "1" LastSetBy: Constructor

```

# Unregistering Console Variables

The **UnregisterConsoleVariable** method allows you to remove the Console Variable. At least, this is what is happening from the user's perspective. The variable is still kept (with the unregistered flags) to not crash when pointers access the data. If a new variable is registered with the same name, the old variable is restored and flags get copied from the new variable. This way DLL loading and unloading can work even without losing the variable state. This will not work for console variable references.