QCons Developer Console

Version 1.11

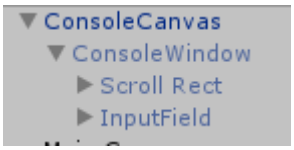
Introduction

The QCons Developer Console is a Unity asset for easier development and debugging of your games. It works in a manner similar to drop-down consoles found in Quake and many other games, usually first-person games. The console aims to maintain a feel familiar to anyone who has used such in-game consoles, or Bash.

For comments, questions or feature requests, you're welcome to email unity@capricornsoftware.se .

Getting started with the prefab

You can use the console simply by adding the `ConsoleCanvas` prefab from the Prefabs folder to your scene. Your scene also needs an EventSystem in order to process user input.



After adding the prefab, you'll be able to open the console by pressing the grave accent key (`) on your keyboard, typically the same key on which tilde (~) appears.

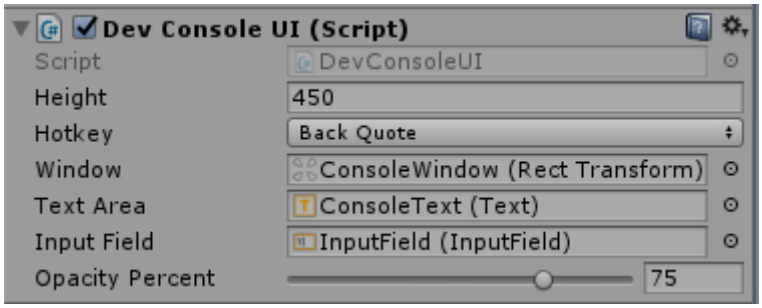
Enter a command in the input field and press Enter to execute it. You may want to first enter `help` to view the list of available commands.

Autocompletion is available by pressing Tab, and you can go back/forward in the history with the up/down arrow keys.

The console's code is in the `DevConsole` namespace, so your scripts that use the console need to use the namespace, i.e., have `using DevConsole;` at the start.

Customizing the console

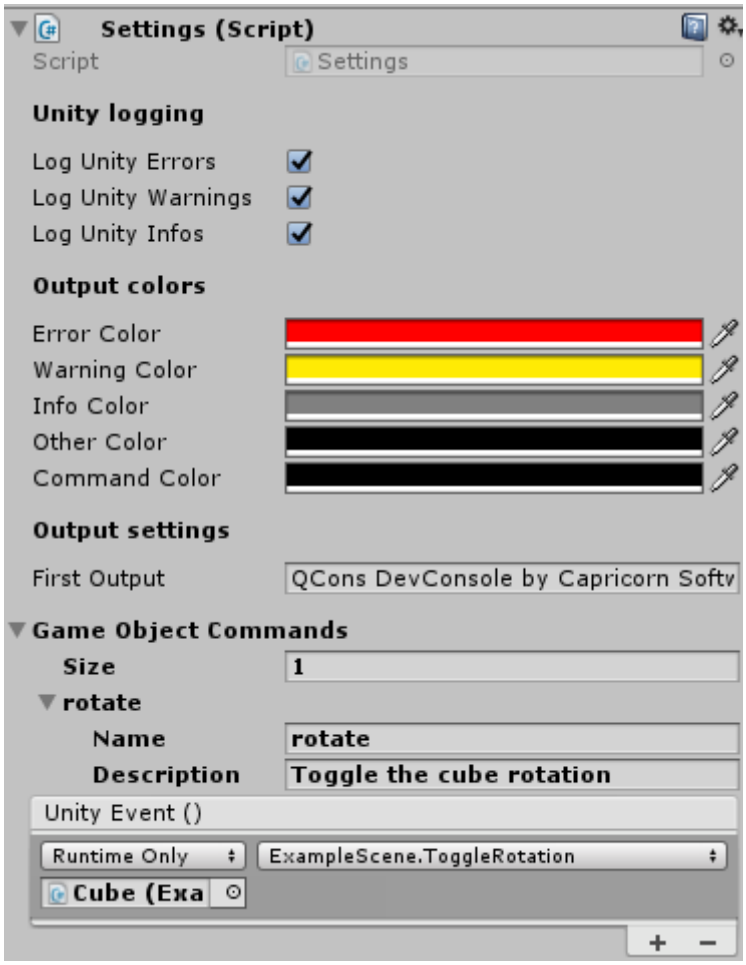
The `ConsoleCanvas` prefab instance you create is going to have two scripts in the Inspector that you can use to customze the console. The first of these is `DevConsoleUI`, responsible for the appearance of the console.



Here you have three settings that may be of interest.

- Height: this is the height of the console in pixels that it will have when activated. Values between 400-600 tend to produce a good-looking dropdown console for most common resolutions. Values as low as 200 are known to work well, if you go any lower, the console might not look good.
- Hotkey: here you can change the hotkey that opens/closes the console.
- Opacity percent: controls how transparent/opaque the console is.

The rest of the console behavior can be customized through the `Settings` script.



The developer console can optionally also log the messages of the Unity console that are written by `Debug.Log`. You can turn this off or limit by using the checkboxes in the ”Unity logging” section.

From here you can also change the colors that will be used for errors, warnings and infos printed by Unity. The ”Other color” will be used for all other output. The ”Command Color” will be used to show commands that you have entered in the console. If you will be exporting the console to HTML, it may be helpful to use a different command color to easily distinguish input from output.

The ”First Output” field sets some text that will be printed to the console the first time it is opened. If you are going to ship the console with playable builds of your game, you may want to put some information about the developers there.

The same Settings script allows you to bind console commands to GameObjects that exist in the scene.

Adding console commands

There are three ways of adding console commands: attribute annotation (recommended), runtime registration and through the Inspector.

Using the ConsoleCommand attribute

The recommended way of adding commands to the console is to use the `[ConsoleCommand]` attribute for the function you wish to call from the console. Such functions have to be public and static, and valid function signatures are:

```
public static void f();

public static void f(params string[] args);

public static string f();

public static string f(params string[] args);
```

If the function returns a string value, then the return value will be printed to the console after the command executes. To register a function with the console, do it like this:

```
[ConsoleCommand]

public static string hello() { return ”Hello world!”; }
```

The method name will be used for the console command, i.e., the above function would be triggered by typing `hello` in the console.

The `ConsoleCommand` attribute takes optional parameters:

`name` – overrides the command name used in the console

`description` – a freeform description of the command, shown in the help listing

`paramsDescription` – a freeform string indicating the format of the parameters

For example, the builtin command `savetxt` is annotated like this:

```
[ConsoleCommand(description = "Save the console to a plain text file",
paramsDescription = "(filename)")]

 public static string savetxt(params string[] args) { … }
```

It means that, in the console help, it’s displayed like this:

```
savetxt (filename)                          Save the console to a plain text file
```

Adding commands at runtime

This may be useful if you want to use the console to call some 3rd-party function that you don’t want to change by annotating, and you don’t want to write a wrapper for it. This approach can also be helpful if you wish to make some commands available only sometimes, such as when running a demo of your game.

In such cases you can use the `AddCommand` function of the `DevConsole` instance, such as:

```
DevConsole.Instance.AddCommand("dyn_example", "An example dynamically
added function", ExampleFunc);
```

You can also specify parameter descriptions for dynamically added functions, such as:

```
DevConsole.Instance.AddCommand("dyn_echo", "An example dynamic function with a parameter", "(str)", DynamicEcho);
```

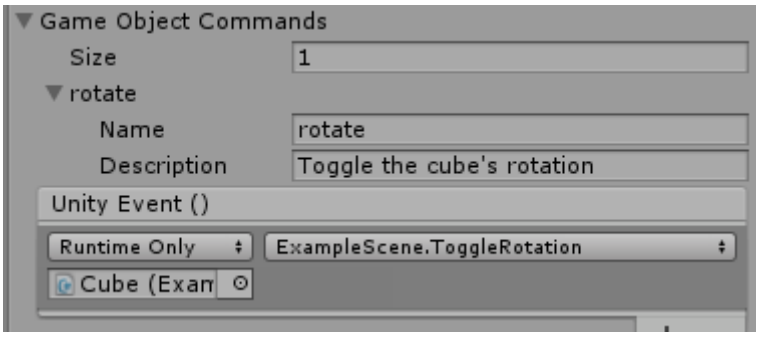
Dynamically added commands can also have variable amount of arguments, that is, have `params string[]` as the argument. Adding such functions dynamically is a bit more verbose as you need to explicitly create a delegate by calling `DevConsole.CreateDynamicCommandDelegate` for your function. Then the full call looks like this:

```
DevConsole.Instance.AddCommand("dyn_params", "An example dynamic function with params[]", "(str, ...)", DevConsole.CreateDynamicCommandDelegate(DynamicParamsFunction));
```

Examples of dynamically added commands can be seen in the exaple scene's script.

Adding commands through the Inspector

If you want a console command to operate on a GameObject that exists in your scene, you can add it through the `Settings` in the Inspector.



This creates a command `rotate` that, if entered in the console, will call the `ExampleScene.ToggleRotation` function of the `Cube` GameObject.

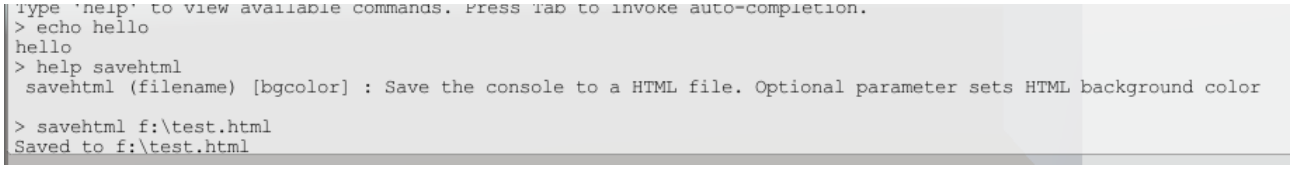
Runtime interaction

If you want to do something in your game depending on whether the console is currently open, the `DevConsoleUI` script exposes the `Active` property. The script is not a singleton, so if you wish to access the console's status, you will need to create a referens to it in your own GameObject that manages the game.

HTML output

One of the console builtins (see the Console builtins section) is `savehtml`. This command takes a filename and exports the entire console history to a HTML file with that name. The feature is primarily useful when debugging your game with someone else.

HTML output preserves the colors of input in the console. If you change the console prefab, the HTML output will try to also match the background color of the console to get more readable output. If that fails with a custom prefab or you just wish to override the background color, you can optionally specify a color string (such as `#add8e6`) as the second parameter to the command.



Doing `savehtml f:\test.html` as in the image above will produce a HTML page viewable in your browser.

Example scene

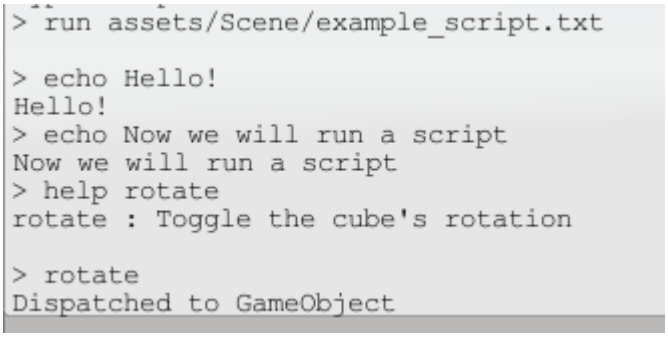
The asset package includes `Scene/ExampleScene`, a scene that demonstrates the console. It contains the console and one additional GameObject, a 3D cube with the `ExampleScene.cs` script attached to it.

You can examine the scene to see how the cube's rotation can be started or stopped from the console, and also see how the cube's script dynamically registers some commands with the console.

The scene folder also has `example_script.txt`, showing how you can save some console commands in a file and then run them like a script. To run it from the example scene, type the following in the console:

```
run Assets/Scene/example_script.txt
```

That will run all the commands in the file.



Console builtins

The console has several built-in commands.

- help [command] – displays a list of all commands. If [command] is given, shows help for that command.
- echo [params…] - will print the given parameters back to the console
- clear – clears the visible output of the console. Previous output is still retained in memory and used if you save the console with savetxt/savehtml.
- envinfo – prints information about the Unity, .NET framework and operating system versons.

- run filename – sequentially runs all console commands from a file
- savetxt filename – saves the entire console history into a file with the provided filename.
- savehtml filename [bgcolor] – saves the entire console history into a HTML file, which will maintain the colors for all text in the console. You can optionally provide a HTML color code as the second parameter, which will use that color for the HTML page's background. Usually the console correctly detects the background color you're using and this parameter is not needed unless you have customized the prefab.

Acknowledgments

The Asset Store icon is provided courtesy of Icons8.com.

For the Asset Store publishing image, graphic by Freepik Flaticon (CC BY 3.0).