Developer Console **Documentation**

https://github.com/anarkila/DeveloperConsole

Table of contents

Getting Started

1. Drag & Drop DeveloperConsole.prefab into your scene (root hierarchy)
2. Add [ConsoleCommand] to your methods like below.
3. Play your scene and press toggle console key (default § or ½) to toggle Console on/off.

```
using UnityEngine;

public class ExampleScript : MonoBehaviour {

    [ConsoleCommand("test")]
    public void Test() {
        Debug.Log("calling 'test' from developer console!");
    }

    [ConsoleCommand("test.int")]
    public void TestInt(int i) {
       Debug.Log(string.Format("Calling 'test.int' with value: {0} from Developer Console!", i));
    }
}
```

Adding and removing commands

Developer Console supports static, instance, and Unity Coroutines methods (both public and private). Methods can contain either no parameters or single parameter with following types:

int, float, string, string[], bool, double, char, Vector2, Vector3, Vector4, Quaternion

Multiple parameters are currently not supported. If you try to register command which takes multiple parameters or parameter type that is not supported, Developer Console will log warning (Editor only) and the command will not be registered.

Registering new commands

You can register new commands in two ways.

https://github.com/anarkila/DeveloperConsole

1) Add [ConsoleCommand("command.name")] attribute to your methods like above example.

2) Register new command with Console.RegisterCommand() method like below (see notes below!).

- Console.RegisterCommand parameters:
  - Script reference (MonoBehaviour)
  - Method name (string)
  - Command name (string)

  Overloads:
  - default value (string) (default "")
    - would show as "{command} {default value}", example: "test.int 42"
  - debugCommandOnly (bool) (default false)
    - if set to true command will only register in Editor and Development build
  - isHiddenCommand (bool) (default false)
    - if set to true would not show in predictions
  - hiddenCommandMinimalGUI (bool) (default false)
    - if set to true would not show in predictions (Minimal GUI only)

```
using UnityEngine;

public class Example : MonoBehaviour {

    private void Start() {
        Console.RegisterCommand(this, "ExampleRegister", "example.register");
    }

    public void ExampleRegister() {
        Debug.Log("Example Register called!");
    }
}
```

Notes:

Instantiated objects with MonoBehaviour script(s) that use [ConsoleCommand()] attribute(s) will not be registered. In that case use Console.RegisterCommand() and Console.RemoveCommand instead.

Use [ConsoleCommand()] attribute for all scripts that do not inherit from MonoBehaviour as Console.RegisterCommand() does not support non-MonoBehaviour command registering!

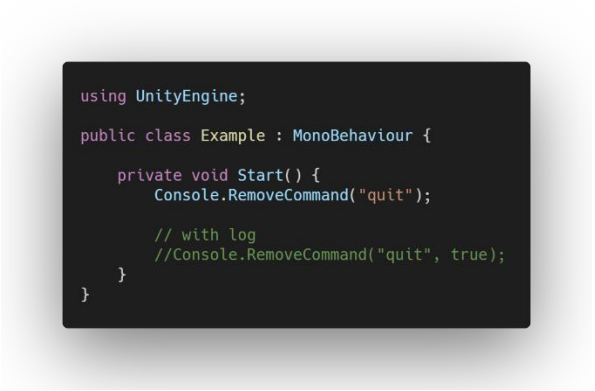
Multiple instances of same command is allowed, like in the example scenes all 3 cubes are disabled/enabled when command *'cube.enabled (bool)'* is called, but same command but in different class or method is not allowed. If same command is found in different class or method, Developer Console logs an warning (Editor only) and that command is ignored.

Removing commands

To remove any command during runtime, you can call Console.RemoveCommand() method. Just provide command name into the method and command will be removed if it exists. Additionally, you can log whether command was found and was removed successfully (Editor only).

- Console.RegisterCommand parameters:
  - Command to remove (string)
  - Log to console (bool)

https://github.com/anarkila/DeveloperConsole



```
using UnityEngine;

public class Example : MonoBehaviour {

    private void Start() {
        Console.RemoveCommand("quit");

        // with log
        //Console.RemoveCommand("quit", true);
    }
}
```

Note:
If Console.RemoveCommand() is called before Console is fully initialized, they will be removed after console is initialized.

Default Commands

Developer Console comes with few commands by default. If you wish to modify or delete them, modify DefaultCommands.cs. Default commands:

- 'help' - Print list of available commands
- 'quit' – Quit the application
- 'Close' – Close console
- 'clear' – clear all Console messages
- 'reset' – Reset Console window position and size (Large GUI only)
- 'console.style' – Toggle GUI between Large and Minimal
- 'scene.loadbyindex (int)' – Load scene asynchronously by scene build index
- 'scene.loadbyname (string)' - Load scene asynchronously by scene name
- 'max_fps (int)' – Set Application.TargetFrameRate

Editor only:

- 'debug.renderinfo' – Print rendering information: High and Avg FPS, highest draw call, batches, triangle and vertices count.

Best Practices

Performance

For the best performance, prefer static class commands over MonoBehaviour instance commands.

This is because registering static class commands can be done in the background (expect in WebGL build). Static commands are also cached on first scene load, so they only need to be registered once.

MonoBehaviour Commands

It's recommended to only use ConsoleCommand()] attributes for MonoBehaviour scripts that will not be Destroyed when playing scene. If you know GameObjects with ConsoleCommand()] attributes will be destroyed runtime (apart from scene change to another scene), consider using Console.RegisterCommand() and Console.RemoveCommand() instead. Calling MonoBehaviour script command which no longer exists is handled by Developer Console, but for best practises consider manually adding if you know they will be deleted.

Developer Console **Documentation**

https://github.com/anarkila/DeveloperConsole

How does it work?

ConsoleInitializer.cs (attached to DeveloperConsole prefab root gameobject) *Awake()* function calls ConsoleManager.cs initialization method. After this ConsoleManager.cs handles everything.

Every time new scene is loaded, ConsoleManager.cs handles looking for all [ConsoleCommand()] attributes in the C# CurrentDomain Assembly. This work happens in two parts.

1) First part runs in the background (unless it's WebGL build), meaning this part should not slow down your game in any way. In this part all [ConsoleCommand()] attributes are found and all static commands are registered and cached.

2) Second part of the attribute registration does run in Unity main thread. This part only register MonoBehaviour commands and this done by looking for all the different MonoBehaviour scripts that have [ConsoleCommand()] attributes. This is done to get MonoBehaviour class instance references, otherwise calling class instance methods is not possible (as far as I know).

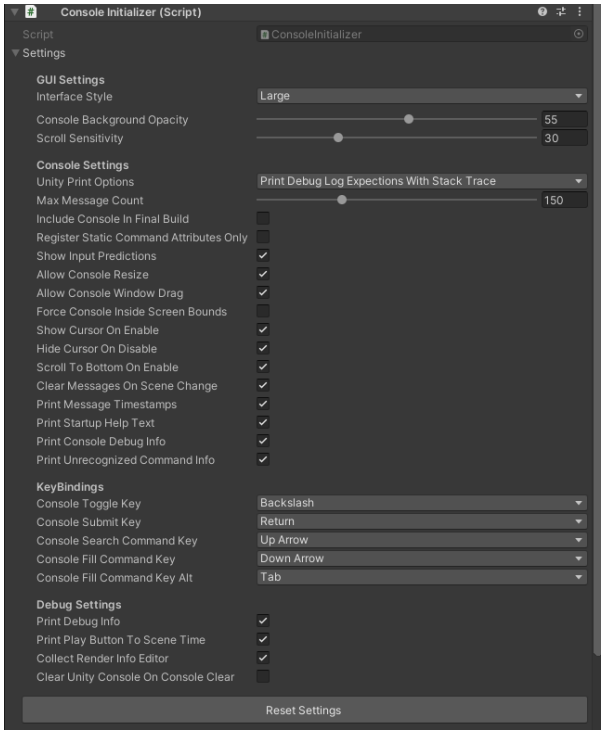
After all commands has been registered, final command lists are built, and all subscribers are notified that console has been fully initialized. If setting *'Print Debug Info'* is enabled and build is Debug (Editor or Development build) then initialization times are printed to console and it should look something like this:

> *Console Initialized. Threaded work took: 1273.1 ms and non-threaded work took: 1.1 ms.*

Where threaded work time means first part and non-threaded work means second part of the attribute registration process.

Developer Console settings

Developer Console comes with few settings you can tweak. To change settings, modify DeveloperConsoleInitilizer.cs in the inspector which is attached to DeveloperConsole prefab gameobject. Some settings can be changed during runtime through static Console.cs class.



Developer Console **Documentation**

https://github.com/anarkila/DeveloperConsole

GUI settings

Interface style: Modify GUI style, Large / Minimal (Default Large)

Console Background Opacity: Developer Console message area background opacity (Applies to Large GUI only) (Default 55)

Scroll Sensitivity: Developer Console scroll sensitivity (Default: 30)

Print options (applies to Large GUI only):

- **Don't print debug logs:**
  - Don't print any Unity Debug.Log/LogError messages into Developer Console
- **Print Debug logs with expections Editor only:**
  - Print Debug.Log/LogError messages into Developer Console window with all expections but only in Editor.
- **Print Debug Logs Expections With Stack Trace Editor Only**
  - Print Debug.Log/LogError messages into Developer Console window with all expections and stack traces but only in Editor. Stack traces will print detailed error logs with script names and all.
- **Print Debug Logs Without Expections**
  - Print Debug.Log/LogError messages into Developer Console window with all expections (Editor and Build)

- **Print Debug Logs Without Expections With Stack Trace (Default)**
  - Print Debug.Log/LogError messages into Developer Console window with all expections and stack traces (Editor and Build). Stack traces will print detailed error logs with script names and all.

Max Message count: how many messages will be shown in the Developer Console window (Large GUI only) before messages will start to recycle from the beginning (Default 150)

Include Console In Final Build: Whether to include Developer Console in final release build. Be careful whether you can actually want to include in final release build! Console is still included in Development builds. (Default false)

Register Static Command Attributes Only: Whether to register static commands only with [ConsoleCommand()] attributes and ignore all other [ConsoleCommand()] attributes. To still register MonoBehaviour commands, use Console.RegisterCommand() method when this setting is set to true. (Default false)

Show Input Predictions: Whether to show inputfield predictions (Default true)

Allow Console Resize: Allow Console to be resized (Applies to large GUI only) (Default true)

Allow Console Window Drag: Allow Console to be dragged (Applies to large GUI only) (Default true)

Force Console Inside Screen Bounds: Whether to force Console to be inside screen bounds when dragging and resizing console. If set to false and console was moved outside screen bounds, Console position and size will reset next time console is opened. (Applies to large GUI only) (Default false)

Show Cursor On Enable: Whether to force cursor on when Console is opened. (Default true)

Hide Cursor On Disable: Whether to force cursor off when Console is closed. (Default true)

https://github.com/anarkila/DeveloperConsole

Scroll to bottom On Enable: Whether to scroll bottom when console is opened (Default true)

Clear Messages On Scene Change: Whether clear all console messages when scene is changed (Default true)

Print Message Timestamps: Whether to print message timestamp (Applies Large GUI only) (Default true)

Print Startup Help Text: Whether to show print *'type help and press enter to.."* text on startup. (Default true)

Print Console Debug Info: Whether to print Console debug info like Initialization time (Editor and Debug builds only) (Default true)

Print Unrecognized Command Info: Whether to print unrecognized command info to console: "Command [command name] was not recognized." (Default true)

Key Bindings

Console Toggle key: Keycode which activates/deactivates Developer Console

Console Submit key: Keycode which submits command

Console Search Command key: Keycode which searches previously (successfully) executed command

Console Fill Command key: Keycode which fills command from prediction

Console Fill Command alt key: Alternative/second keycode which fills command from prediction

Debug Settings

Print Debug Info: Whether to print debug info such as Console initialization times (Default true)

Print Play Button To Scene Time: Whether to print play button click to playable scene time (Default true)

Collect Render Info Editor: Whether to collect Unity rendering information such as highest draw call / batches count. This can be printed to console with command: `debug.print.renderinfo` This is happens in DebugRenderInfo.cs class (Default true)

Clear Unity Console On Console Clear: Whether to Clear Unity console messages too when Developer Console 'clear' event called (Editor only) (Default false)

Getting console state info

Developer Console provides simple static API you can call everywhere to interact with console. This is intended if you wish to change Console settings runtime or to know console state.

If you need to know when Developer Console is opened or closed (for turning scripts on/off for example), you can do it two ways.

1) Register to console event like below. Just remember to unregister your event either in OnDisable or OnDestroy.

https://github.com/anarkila/DeveloperConsole

```
using UnityEngine;

public class Example : MonoBehaviour {

    private void OnEnable() {
        Console.RegisterConsoleStateChangeEvent += Callback;
    }

    private void OnDisable() {
        Console.RegisterConsoleStateChangeEvent -= Callback;
    }

    private void Callback(bool enabled) {
        Debug.Log("Console is open: " + enabled);
    }
}
```

2) If you want to check Developer Console state in Update() function.

```
using UnityEngine;

public class Example : MonoBehaviour {

    private void Update() {
        if (Console.IsConsoleOpen()) {
            return;
        }

        // Else do your work..
    }
}
```

TODO

- Improve GUI
- Explore ways to have multiple parameters
- Explore ways to register commands faster
- Way to register commands in Editor
- Improve Garbage Collection
- Ability to generate grid or list of buttons that can fire commands

Developer Console **Documentation**

https://github.com/anarkila/DeveloperConsole

Version History

v0.7.0

First public release