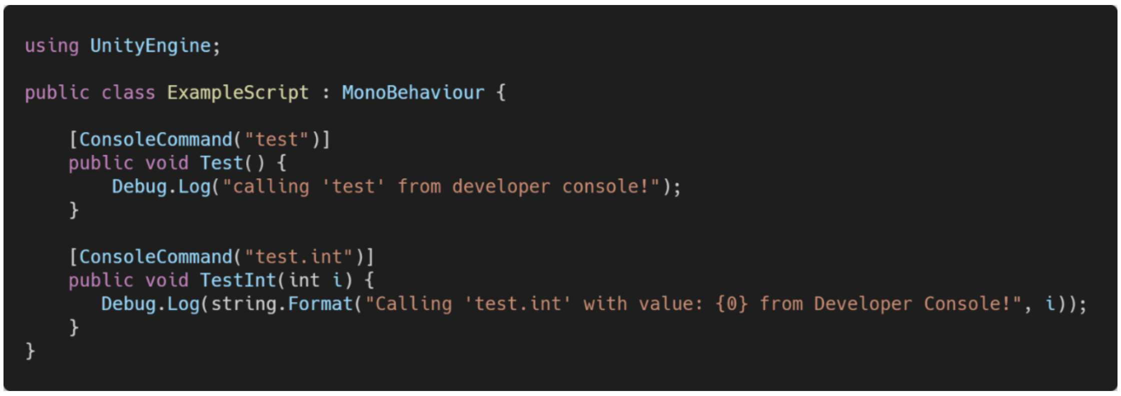
Developer Console **Documentation**

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));
    }
}
```

Handling 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, byte, char, Vector2, Vector3, Vector4, Quaternion

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

Registering new commands

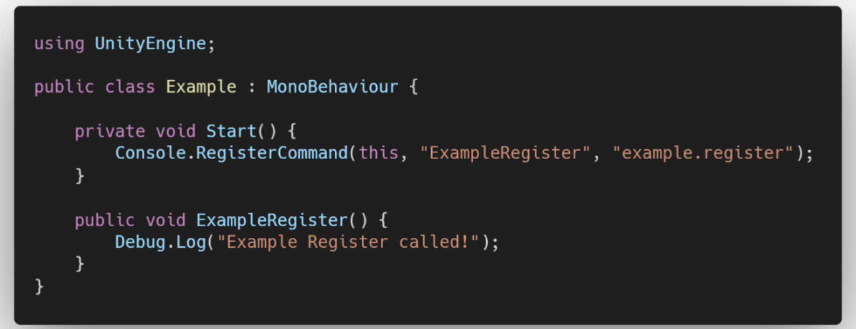
You can register new commands in two ways.

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

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

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

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



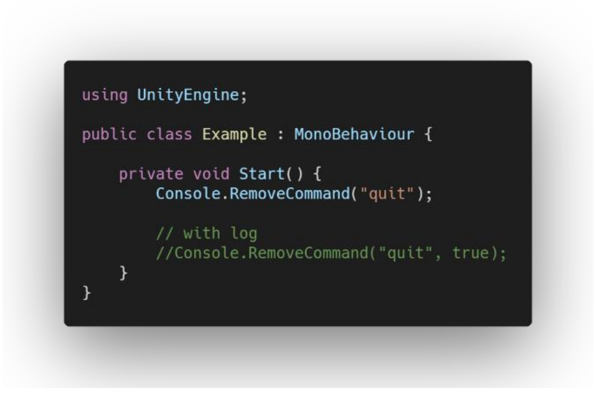
```
using UnityEngine;

public class Example : MonoBehaviour {

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

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

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

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)



```
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.

Best Practices

For the best performance, prefer adding new commands in this order.

1) static class commands with [ConsoleCommand()] attribute

   - This is fast and is done in the background (expect in WebGL build). Static commands are also cached on first scene load.

2) Register MonoBehaviour commands through Console.RegisterCommand()

   - Fastest way to Register MonoBehaviour commands. Note. when scene has been changed all non-static commands will be removed. If you have script that lives through multiple scenes, you have register the command again.

3) Register MonoBehaviour commands with [ConsoleCommand()] attribute

   - While this is the easiest and most convenient way to register MonoBehaviour commands, it's not the fastest. To get MonoBehaviour script references all gameobjects in the scene must be looped and checked if they contain the script(s). As you can imagine this is not very efficient and can be rather slow for big scenes. If setting 'Print Console Debug Info' is enabled, this part of the registration will be printed to console. In DeveloperConsole example scenes this process takes ~25 ms in Editor. This process takes longer, the more gameobjects scene have and the more [ConsoleCommand()] attribute you add.

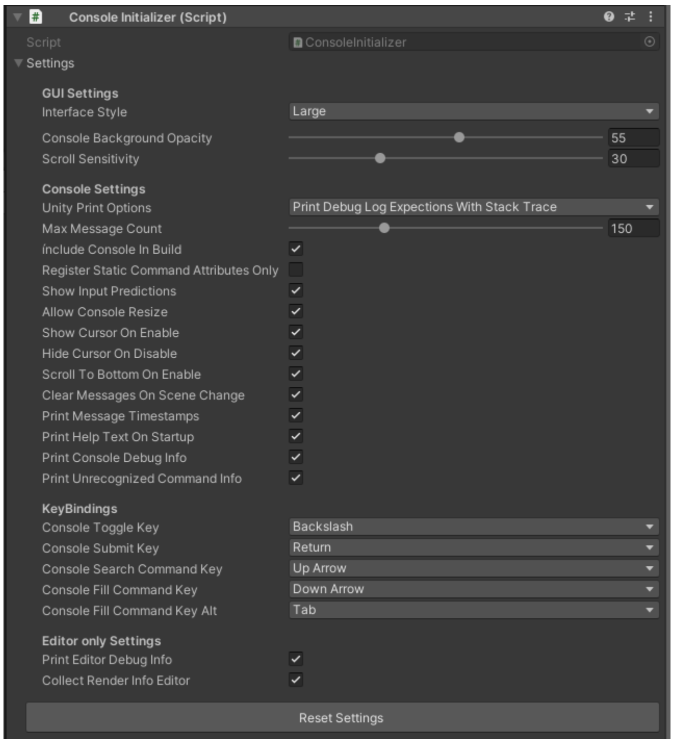
How does it work?

Everytime scene loads ConsoleManager.cs looks for all [ConsoleCommand()] attributes in C# CurrentDomain Assembly (1). This work happens in two parts. First part does not run in Unity main thread (unless it's WebGL build), meaning the first part should not slow down your game load time. Second part of the attribute registration does run in main thread. This part loops through all GameObjects in your current scene and finds MonoBehaviour references so Monobehaviour methods can be properly called. This *can be* rather slow operation for large scenes, but don't worry, this only happens once after scene is loaded.

1. Static commands are only registered once, and then they are cached. So when when scene changes static commands do not need to be re-registered.

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.



Console Initializer (Script)

Script	ConsoleInitializer
Settings	
GUI Settings	
Interface Style	Large
Console Background Opacity	55
Scroll Sensitivity	30
Console Settings	
Unity Print Options	Print Debug Log Expections With Stack Trace
Max Message Count	150
Include Console In Build	✓
Register Static Command Attributes Only	
Show Input Predictions	✓
Allow Console Resize	✓
Show Cursor On Enable	✓
Hide Cursor On Disable	✓
Scroll To Bottom On Enable	✓
Clear Messages On Scene Change	✓
Print Message Timestamps	✓
Print Help Text On Startup	✓
Print Console Debug Info	✓
Print Unrecognized Command Info	✓
KeyBindings	
Console Toggle Key	Backslash
Console Submit Key	Return
Console Search Command Key	Up Arrow
Console Fill Command Key	Down Arrow
Console Fill Command Key Alt	Tab
Editor only Settings	
Print Editor Debug Info	✓
Collect Render Info Editor	✓

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 Build: Whether to include Console in build. If set to false tag will change to EditorOnly. (Default true)

Register Static Command Attributes Only: Whether to register static commands only with [ConsoleCommand()] attributes and ignore all other [ConsoleCommand()] attributes. Registering static commands is fast and efficient while registering MonoBehaviour commands with [ConsoleCommand()] can be rather slow for big scenes as all gameobjects must be looped through to find MonoBehaviour references. To 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)

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)

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

Print help info on Startup: 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)

KeyBindings

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

Editor only Settings:

PrintEditorDebugInfo: Whether to print Editor debug info - Play button click to playable scene time (Default true)

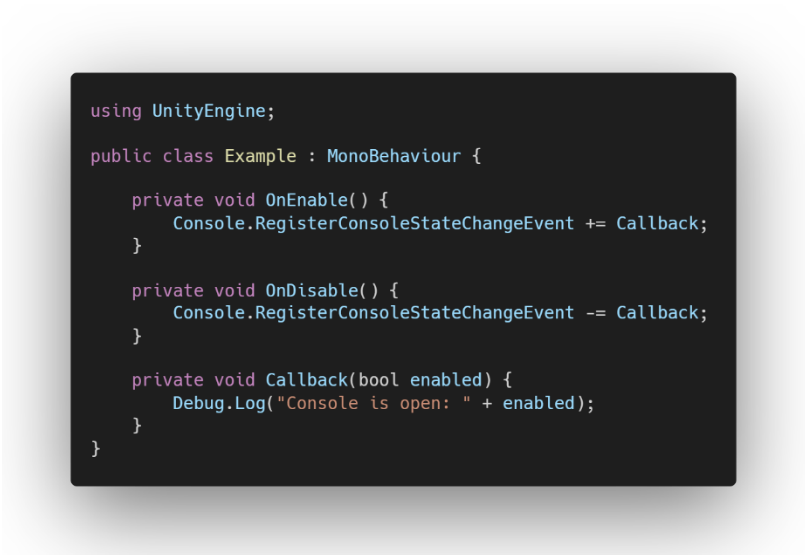
Collect Render Info Editor: Whether to collect Unity rendering information such as highest draw call / batches count. This can be printed to console with command: `debug.print.renderinfo` This is happens in DebugRenderInfo.cs class (Default true)

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.

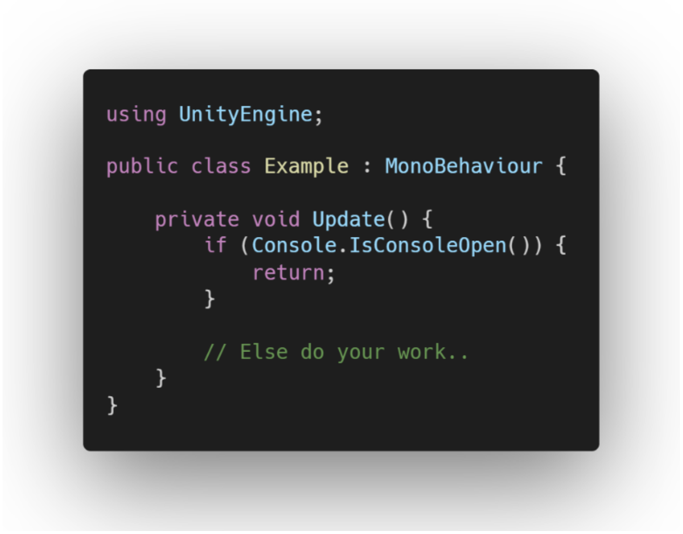
If you need to know when Developer Console is opened or closed (for turning scripts on/off for example), you can do it two ways.

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



```
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 Monobehaviour commands faster
- Explore way to register commands in Editor
- Improve Garbage Collection
- Ability to generate grid or list of buttons that can fire commands

Version History

v0.9.0

First public release