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 Developer 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));
    }
}
```

Developer Console **Documentation**
https://github.com/anarkila/DeveloperConsole

Registering new commands

You can register new commands two ways.

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

2) Or manually registering new command with ConsoleAPI.RegisterCommand() method like below.

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

     Overloads:
     - default value (string)
     - isHiddenCommand (bool) – Would not show in predictions
     - hiddenCommandMinimalGUI (bool) – Would not show in predictions (Minimal GUI only)

```
using UnityEngine;

public class Example : MonoBehaviour {

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

    private void ExampleRegister() {
        Debug.Log("ExampleRegister called");
    }
}
```

For all examples see DeveloperConsoleExamples.cs script.

Allowed parameter types:

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

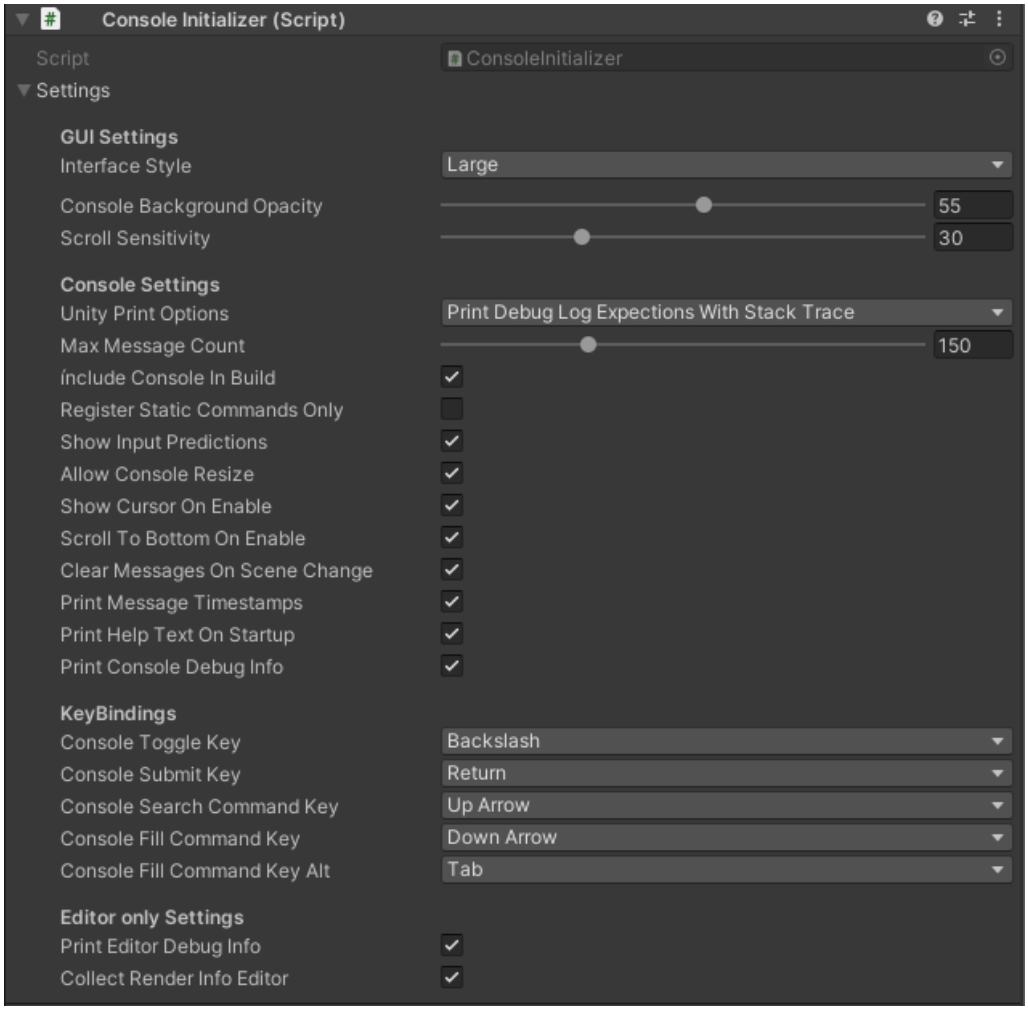
How does it work?

Everytime scene loads ConsoleManager.cs looks for all [ConsoleCommand("command")] 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 ConsoleAPI class.



Settings

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 Commands Only: Whether to register static commands only and ignore all MonoBehaviour ConsoleCommand attributes. Registering static commands is fast and efficient. They are also cached so it only needs to be done once during the entire application runtime. (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)

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 (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 (ConsoleAPI)

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() {
        ConsoleAPI.RegisterConsoleStateChangeEvent += Callback;
    }

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

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

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

```
using UnityEngine;

public class Example : MonoBehaviour {

    private void Update() {

        if (!ConsoleAPI.IsConsoleOpen()) {
            return;
        }

        // else do your work..
    }
}
```

Developer Console **Documentation**
https://github.com/anarkila/DeveloperConsole

TODO

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

Version History

v0.9

First public release