Developer Console 0.8.2 **Documentation**

https://github.com/anarkila/DeveloperConsole

Table of contents

Getting Started

1. Drag & Drop DeveloperConsole.prefab into your scene
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, MonoBehaviour (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

Methods with optional parameter are also supported.

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

Developer Console 0.8.2 **Documentation**

https://github.com/anarkila/DeveloperConsole

Registering new commands

You can register new commands in two ways.

1) Add [ConsoleCommand("test")] 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)

  Optional:
  - 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. Use Console.RegisterCommand() and Console.RemoveCommand for objects that are instantiated runtime.

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 warning (Editor only) and that command is ignored.

Removing commands

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

https://github.com/anarkila/DeveloperConsole

- 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);
    }
}
```

Default Commands

Developer Console comes with few commands by default. If you wish to change, modify or delete them you can find them in *DefaultCommands.cs* and *DebugRenderInfo.cs* classes.

- **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)
- **max_fps (int)** – Set Application.TargetFrameRate
- **console.style** – Toggle GUI between Large and Minimal
- **scene.loadbyindex (int)** – Load scene asynchronously by scene build index
- **scene.loadbyindexadd (int)** – Load scene asynchronously additively by scene build index
- **scene.loadbyname (string)** - Load scene asynchronously by scene name
- **scene.unloadbyname (string)** - Unload scene asynchronously by scene name
- **scene.unloadbyindex (int)** - Unload scene asynchronously by scene build index

Editor only:

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

Command list generation

To generate text file that contains all commands, open Tools/Developer console and click ‘Generate Command List’ or click ‘Generate Command List’ button on DeveloperConsole.cs inspector.

Text file will be generated to *YOUR_PATH*/DeveloperConsole/Editor/ folder.

Developer Console 0.8.2 **Documentation**

https://github.com/anarkila/DeveloperConsole

Console.cs API

Console.cs is simple static API to interact with the console programmatically.

Console.Log(): Logs message to developer console directly.
- Parameters:
  - String or System.Object

Console.RegisterCommand(): Register new console command. See Adding and removing commands section.
- Parameters:
  - MonoBehaviour script
  - String methodName
  - String commandName
  - (optional) String defaultValue
  - (optional) String info
  - (optional) Boolean debugCommandOnly
  - (optional) Boolean IsHiddenCommand
  - (optional) Boolean HiddenCommandMinimalGUI

Console.RemoveCommand(): Remove console command by command name
- Parameters:
  - String commandToRemove
  - Optional: Boolean logResult

Console.IsConsoleOpen(): returns Boolean whether console is currently open.

Console.IsConsoleInitialized(): returns Boolean whether console is fully initialized.

Console.OpenConsole(): Open Developer Console

Console.CloseConsole() : Close Developer Console

Console.SetConsoleState(): Set Developer Console state (open or closed**)**
- Parameters:
  - Boolean state

Console.AllowConsoleActivateKey(): Enable/disable listening default console activate key. If you disable listening default activate key then you must handle opening/closing console yourself (with above methods).
- Parameters:
  - Boolean enabled

Console.ClearConsoleMessages(): Clear all Developer Console messages

Console.RebindConsoleActivateKey(): Rebind console activate key (default §)
- Parameters:
  - KeyCode newKey

Console.DestroyConsole(): Destroy Developer Console

Console.SetSettings(): Set new ConsoleSettings.cs class
- Parameters:
  - ConsoleSettings newSettings

Console.GetSettings(): returns current settings (ConsoleSettings.cs class)

Console.GetGUIStyle(): returns current GUI style (ConsoleGUIStyle enum)

Console.SetGUIStyle(): Set Console style to use.
- Parameters:
  - ConsoleGUIStyle style

Console.RegisterConsoleStateChangedEvent: Register to reiceve console state change callback. See Getting Console state info section for usage.

Console.RegisterConsoleInitializedEvent: Register to receive callback when console is fully initialized.

In case your are unfamiliar with events see How to subscribe to and unsubscribe from events

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) and they can cached, so they only need to be registered once.

Additive scene un/loading

If your game relies on additive scene un/loading, consider enabling *'Register Static Command Attributes Only'* setting and use static class commands only for the best performance. Currently all MonoBehaviour commands are re-/registered again if scene is loaded or unloaded (additively and single)

Logging

Avoid unnecessary Debug.Log/LogError or Console.Log calls. C# strings generate garbage and garbage collection in Unity is quite slow, thus it's recommended to remove all unnecessary ones. Avoid calling log messages in any Update() methods unless it's for debugging purposes.

How does it work?

DeveloperConsole.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 or unloaded (additively or single), 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.

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.

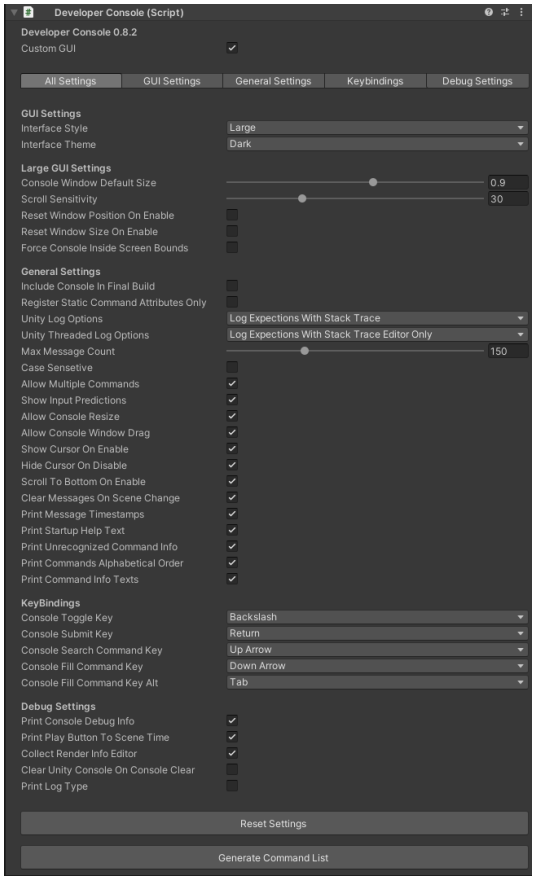
https://github.com/anarkila/DeveloperConsole

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

After commands has been registered, only one or two scripts run every frame. ConsoleKeyInputs.cs runs every frame and handles listening all key inputs. Second script that runs every frame is ListenThreadedLogs.cs (if setting 'Unity threaded log option' is set to DontPrintLogs then this script is disabled) but most of the time it will just return before even doing anything. Everything else is event based and does not run unless needed. So Developer Console should not slow your game.

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.



Custom GUI: Whether to use custom GUI (tabs) or render default inspector view (default true)

GUI settings

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

Interface theme: Select GUI them (Dark, Red, Custom) (Default Dark)

Large GUI settings

Console Window Default Size: Developer Console default window size multiplier (Applies to Large GUI only) (Default 0.9)

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

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

Reset Window Position On Enable: Whether to reset window position when console is opened (Applies to large GUI only) (Default false)

Reset Window Size On Enable: Whether to reset window size when console is opened (Applies to large GUI only) (Default false)

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)

General settings

Include Console In Final Build: Whether to include Developer Console in final release build (Default false). Be careful whether you actually want to include in final release build. This is false by default, so you don't accidentally release this into the wild. You can and have permission to publish this for release builds but be careful to not include commands you don't want users to use!

Register Static Command Attributes Only: Whether to register static commands only with [ConsoleCommand()] attributes and ignore all other [ConsoleCommand()] attributes (default false). See Adding and Removing Commands and Best Practises sections for more info.

Unity Log Option (applies to Large GUI only):

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

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

Unity Threaded Log Option (applies to Large GUI only): same as above but for logs that come from another threads.

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)

Case Sensetive: Whether calling commands must be case sensitive (Default false)

Developer Console 0.8.2 **Documentation**

https://github.com/anarkila/DeveloperConsole

Allow Multiple Commands: Whether to allow multiple commands to be executed in a row like below (default true). See Multiple Commands section for more info.

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)

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

Print Commands Alphabetical Order: Whether to print commands in alphabetical order when command 'help' is called (Default true)

Print Command Info Texts: Whether to commands info texts when 'help' command called. Command info texts are optional parameter when adding [ConsoleCommand()] attribute (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 Console Debug Info: Whether to print Console debug info like Initialization time (Editor and Debug builds only) (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 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)

https://github.com/anarkila/DeveloperConsole

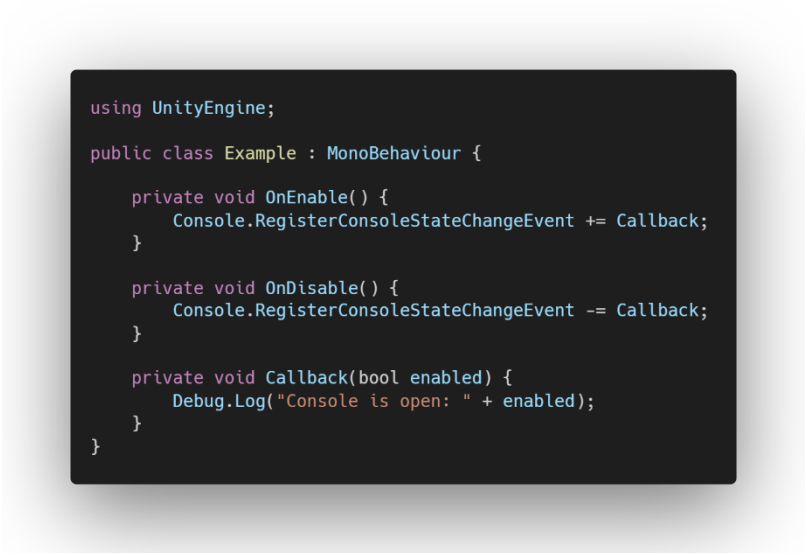
Print Log Type: Whether to print log type (Log, exception, Warning, Assert, Error) before message (Default false)

Getting Console state info

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

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.



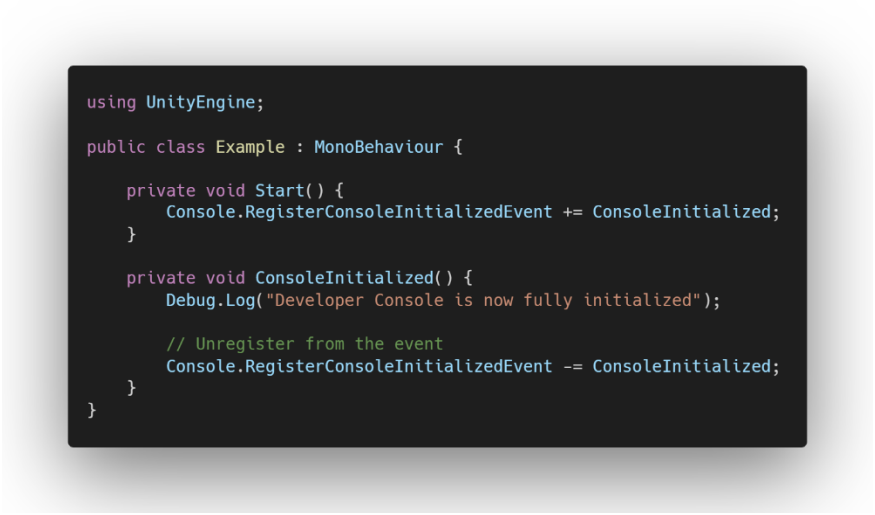
```
using UnityEngine;

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..
    }
}
```

https://github.com/anarkila/DeveloperConsole

Developer Console takes some time (approx. 1-2 seconds) to fully initialize after scene is loaded so if you need to know if it's fully Initialized, you can either register to *Console.RegisterConsoleInitializedEvent* (example below) or check by calling *Console. IsConsoleInitialized()*



```
using UnityEngine;

public class Example : MonoBehaviour {

    private void Start() {
        Console.RegisterConsoleInitializedEvent += ConsoleInitialized;
    }

    private void ConsoleInitialized() {
        Debug.Log("Developer Console is now fully initialized");

        // Unregister from the event
        Console.RegisterConsoleInitializedEvent -= ConsoleInitialized;
    }
}
```

Multiple commands

Developer Console allows multiple commands to be executed if setting 'Allow Multiple Commands' is set to true.

You can use both "&" or "&&" to separate between commands like below:

Test.int 1 & test.int 2
Test.int 1 && test.int 2

When executing multiple commands, do not use '&' as parameter. For example, in the examples scenes below command would not print right.
Test.int 1 & test.string &&

Output in console:
"Called command 'test.int' successfully with value: 1 from Console!"
"Called command 'test.string' successfully with value: ' ' from Developer Console!"

Test.string fails to print "&&" as text because this character is used to parse multiple commands. If you need to pass "&" as parameter, call that command alone like below:

test.string &&

Output in console:
"Called command 'test.string' successfully with value: ' &&' from Developer Console!"

Developer Console 0.8.2 **Documentation**

https://github.com/anarkila/DeveloperConsole

Version History

v 0.8.2

- Fixed issue where console took two button presses to open after closing console with top right X button
- Fixed issue where max message count wouldn't increase max message count
- Changed all GetComponent calls to TryGetComponent calls to reduce Editor garbage amount
- Other minor performance improvements
- Improved Console window dragging
- Added support for multiple commands
- Added support and option for threaded Debug.Log/Error messages
- Added GUI themes (Dark, Red, Custom)
- Added more default scene un/loading commands
- Added info text optional parameter to ConsoleCommand. This info string is printed when 'help' command is called and 'print command info texts' is set to true (default true).
- Added / modified settings:
    - GUI themes (Dark, Red, Custom)
    - Settings to control whether scrollrect scrollbar should auto-hide or be permanent
    - GUI colors (visible when GUI theme is selected Custom)
    - print Command info texts
    - print Unity message log type (Log, Warning, Error, Expection, Assert)
    - enable/disable multiple commands

v 0.8.1

- Fixed issue where calling coroutine which no longer existed in the scene resulted in error
- Fixed issue where coroutine was started when gameobject was inactive
- Fixed issue where multiple DeveloperConsole prefabs where open at the same time
- Fixed issue where in editor DeveloperConsole would throw NullReferenceException if scene was entered and exited before console was initialized
- Renamed namespace to avoid any conflicts with other assets
- Prediction panel texts are now buttons and can now be clicked. Clicking prediction text will input that text into console inputfield
- Minor GUI improvements
- Minor performance improvements
- Made tabbed Inspector GUI for ConsoleInitializer.cs
- New settings:
    - highlight color
    - Setting for printing console commands in alphabetical order

v 0.8.0

Initial release