Developer Console 0.8.4 **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. See *ConsoleExamples.cs* for all examples.
3. Play your scene and press console key (default § or ½) to toggle Console on/off.

```
using UnityEngine;

public class ExampleScript : MonoBehaviour {

    [ConsoleCommand("test")]
    private void Test() {
        Debug.Log("Called 'test' from Console!");
    }

    [ConsoleCommand("test_int")]
    private void TestInt(int i) {
       // single parameter allowed types:
       // int, float, string, bool, double, char, string[], Vector2, Vector3, Vector4, Quaternion
       Debug.Log(string.Format("Called 'test_int' with value: {0} from Console!", i));
    }

     [ConsoleCommand("test_multi_int")]
    private void TestMultiInt(int i, int j) {
       // multi parameter allowed types:
       // int, float, string, bool, double, char
       Debug.Log(string.Format("Called 'test_multi_int' with value: {0} and {1} from Console!", i, j));
    }
}
```

Developer Console 0.8.4 **Documentation**

https://github.com/anarkila/DeveloperConsole

Adding and removing commands

Developer Console supports static, MonoBehaviour (instance), and Unity Coroutines methods (both public and private). Methods can contain either no parameters, single parameter or multiple parameters with certain rules.

Single parameter type support:

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

Multi parameter supported types:

int, float, string, bool, double, char

Unity coroutines cannot contain multiple parameters because coroutines are started by calling method by its name and you can only pass one argument into it.

Methods with optional parameter are also supported.

Methods that contain multiple parameters with Vector2, Vector3, Vector4, Quaternion or string[] are currently not supported because character ‘,’ (comma) is used to parse multiple parameters as well as types above. Developer Console will log warning (Editor only) and reason if the command cannot be registered.

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)

Developer Console 0.8.4 **Documentation**

https://github.com/anarkila/DeveloperConsole

```
using UnityEngine;

public class Example : MonoBehaviour {

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

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

Notes:

Command name cannot contain characters whitespaces, '&' or ',' because these characters are reserved for parsing.

Instantiated objects with MonoBehaviour script 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).

- 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);
    }
}
```

Developer Console 0.8.4 **Documentation**

https://github.com/anarkila/DeveloperConsole

Calling commands

To call added Console commands certain rules are applied due parsing.

To call command(s) that take no parameters:

{command_name}

Example: test_method

Note. Empty space(s) before {command_name} is not allowed.

To call command(s) that take one parameter:

{command_name} {empty space} {parameter}

Example: test_int 5

To call command(s) that take two or more parameters:

{command_name} {empty space} {parameter} {comma} {parameter}...

Example: test_multi_int 5, 8

To call multiple commands in a row, add character '&' or '&&' with above rules.

{command_name} {empty space} {& or &&} {command_name} {empty space} {parameter}

Example: test_method && test_int 9

Note. Option 'Allow Multiple Commands' must be enabled this to work (enabled by default).

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*, *DebugRenderInfo.cs* and *DeveloperConsole.cs* classes.

- **help** - Print list of available commands
- **quit** – Quit the application (works in Editor and Build)
- **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. This is only registered if option 'Allow GUI Change runtime' is set to true.
- **scene_loadindex (int)** – Load scene asynchronously by scene build index

- **scene_addloadindex (int)** – Load scene asynchronously additively by scene build index
- **scene_loadname (string)** - Load scene asynchronously by scene name
- **scene_unloadname (string)** - Unload scene asynchronously by scene name
- **scene_unloadindex (int)** - Unload scene asynchronously by scene build index
- **empty** – Log empty line to console. This command is Editor only.
- **debug_renderinfo** – Print rendering information: High and Avg FPS, highest draw call, batches, triangle and vertices count. This command is Editor only. The API (UnityStats.cs) only works in Editor.

Command list generation

To generate text file that contains all [ConsoleCommand] attribute 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.

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
  - (optional) Boolean forceIgnoreTimestamp

Console.LogEmpty(): Logs empty message to Developer Console without timestamp

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 receive 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

Logging

By default, *Unity.Log* and *Debug.LogError* messages are redirected to Developer Console. This can be disabled and the level of logging of *Debug.LogError* can be controlled through settings.

To log messages directly to Console, you can use Console.Log() method from anywhere:

Console.Log(“hello”);

To log messages with certain Color, you can add optional Color parameter.

Console.log(“hello”, Color.Red);

There’s also third optional parameter to force ignore timestamp even when option *‘print message timestamps’* is enabled:

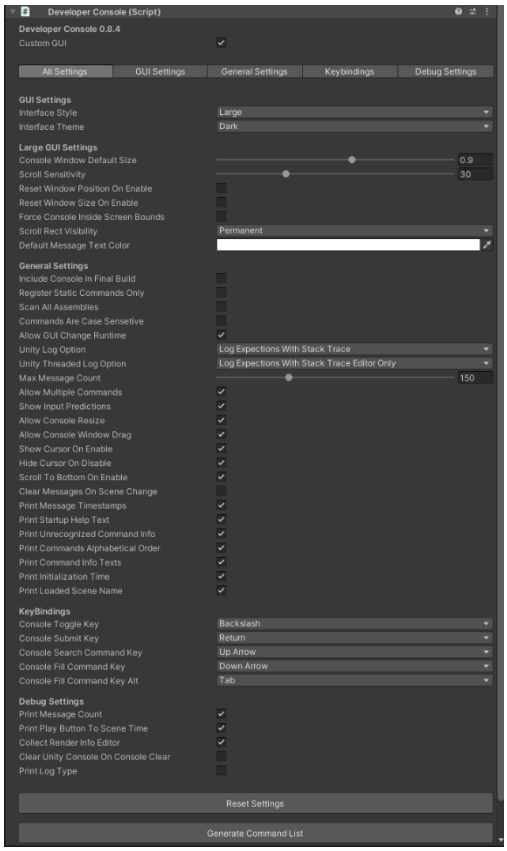
Console.Log("*hello* ", Color.red, true);

Developer Console 0.8.4 **Documentation**

https://github.com/anarkila/DeveloperConsole

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.



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 theme (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)

Scroll Rect Visibility: Whether to show Scroll Rect permanent or hidden when there's not enough messages to scroll through (default permanent)

Default Message Text Color: Default message text color (default white)

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

Scan All Assemblies: Whether to scan all C# Assemblies when registering commands. Enabling this will increase command registration time significantly. You shouldn't have to enable this. If this option is enabled a warning is displayed (Editor only) (default false!)

Allow GUI Change runtime: Whether to allow changing between Large GUI and Minimal GUI runtime (default true)

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 options but for logs that come from other than main threads. *ListenThreadedLogs.cs* script handles listening threaded messages and safely 'moves' them into main thread so messages can printed to Developer Console.

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

https://github.com/anarkila/DeveloperConsole

Commands Are Case Sensetive: Whether calling commands are case sensitive (Default false).

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

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 false)

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)

Print Initialization Time: Whether to print Console Initialization time when scene is loaded (default true).

Print Loaded Scene Name: Whether to print loaded scene name.

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 Message Count: Whether to print Debug.Log/LogError and Console.Log message count after exiting play mode (Editor only) (default true)

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

https://github.com/anarkila/DeveloperConsole

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)

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

Getting Console state info

Developer Console comes with 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. 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..
    }
}
```

If you need to know if Developer Console is fully initialized, you can either register to *Console.RegisterConsoleInitializedEvent* (like in the example below) or check by calling *Console.IsConsoleInitialized()* function.

```
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;
    }
}
```

Note.

This event fires every time scene loads as all Console commands (expect static commands) must be re-registered.

Since version 0.8.3 Developer Console should initialize relatively fast and should be fully initialized before any *Start()* function fires.

Multiple commands

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

You can use character "&" 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!"

Note. Input predictions are not shown if console inputfield contains character '&'.

How does it work?

DeveloperConsole.cs which is 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 looks for all [ConsoleCommand()] attributes in the Unity C# Runtime assembly (Assembly-CSharp) and all static class commands are registered and cached. Unity Editor and all other assemblies are ignored unless setting *'Scan All Assemblies'* is set to true.

2) Second part handles registering MonoBehaviour instance commands and this done by looking for all the different MonoBehaviour scripts that have [ConsoleCommand()] attribute(s).

After all commands has been registered, final command lists are built, and all subscribers are notified that console is now fully initialized. If option *'Print Initialization Time'* is enabled then initialization times are printed to console, and it should look something like this:

Console Initialized. Initialization took 15 ms.

After commands has been registered, only one or two scripts run every frame.

- *ConsoleKeyInputs.cs* runs every frame and handles listening all key inputs.
- *ListenThreadedLogs.cs* also runs every frame (if option '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.

https://github.com/anarkila/DeveloperConsole

Best Practices

Performance

If your game relies heavily on additive scene un/loading or you change scenes often, consider enabling *'Register Static Commands Only'* option and use static class commands only for the best performance. Currently all MonoBehaviour commands are re-/registered again when new scene is loaded or unloaded while all static commands are cached on first load.

Consider keeping option *'Max Message Count'* as small as you need. The default 150 is great start. After 150 messages has been reached, messages will start to recycle from the beginning. You can increase this as big as you want but consider Console might get slow'*ish* with large number of messages (1K+).

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 Late/Fixed/Update() methods unless it's for short debugging purposes.

If you are only using Minimal GUI style, consider changing settings 'UnityPrintOption' and 'UnityThreadedPrintOption' ConsoleLogOptions enum to DontPrintLogs. By default, Debug.Log/LogError and Console.Log messages are still printed to Large GUI console even when it's not selected. Why? So you can toggle between them during runtime with command 'console.style'.

Developer Console 0.8.4 **Documentation**

https://github.com/anarkila/DeveloperConsole

Version History

v 0.8.4

- Added support for no domain/scene reload
- Added warnings if Console.cs methods were called from another thread.
- Added Ability to log message with color
- Added partial multiple parameter support
    - Only methods that don't contain Vector2, Vector3, Vector3 or String[] are supported.
- Fixed issue where command prediction would not show in minimal GUI
- Optimized console message pooling

v 0.8.3

- Improved ConsoleCommand attribute registration performance (from ~1500 ms to ~15 ms)
- Other minor performance improvements
- Logging and warning improvements
- Changed default command names
- Changed few settings defaults
- Added new settings

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 setting 'max message count' wouldn't increase max message count
- Reduced garbage collection in editor
- Minor performance improvements
- Improved Console window dragging
- Improved input prediction
- Added support for multiple commands (separated by '&' or '&&')
- Added support and option for threaded Debug.Log and Debug.LogError messages
- Added GUI themes (Dark, Red, Custom)
- Added more default scene un/loading commands
- Added optional info text parameter to ConsoleCommand. This info string is printed ´with 'help' command if setting 'print command info texts' is set to true (default true).
- Added more settings to tweak

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

Developer Console 0.8.4 **Documentation**

https://github.com/anarkila/DeveloperConsole

v 0.8.0

Initial release