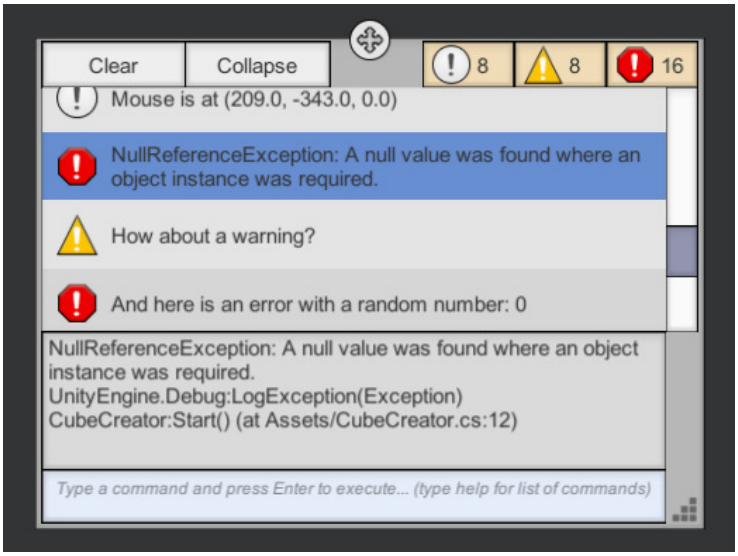
In-game Debug Console *by Suleyman Yasir Kula*

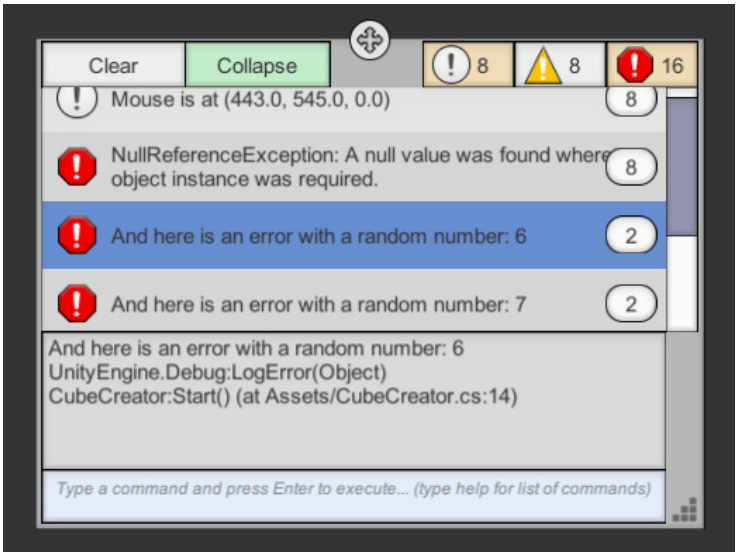


Thank you for downloading this asset. This asset is a means to see debug messages (logs, warnings, errors, exceptions) in-game in a build (also assertions in editor). It also has a built-in console that allows you to enter commands and execute pre-defined functions in-game.

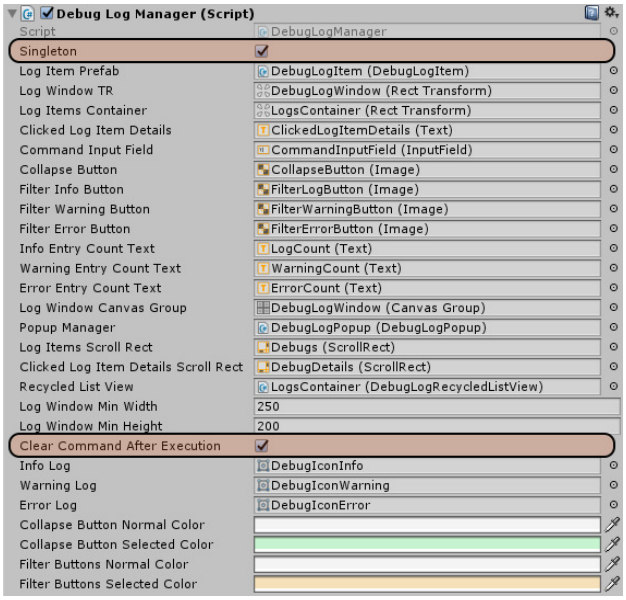
Here, I will briefly talk about how to use these features. Enjoy the show!

Contact: mailto:yasirkula@yahoo.com **Web:** http://yasirkula.net

▪ Viewing Debug Messages



Console handles interception of log messages automatically. It is sufficient to just drag & drop the **DebugLogCanvas** prefab at **Plugins/DebugLog** into your scene. However, there are 2 important variables that you may want to tweak:

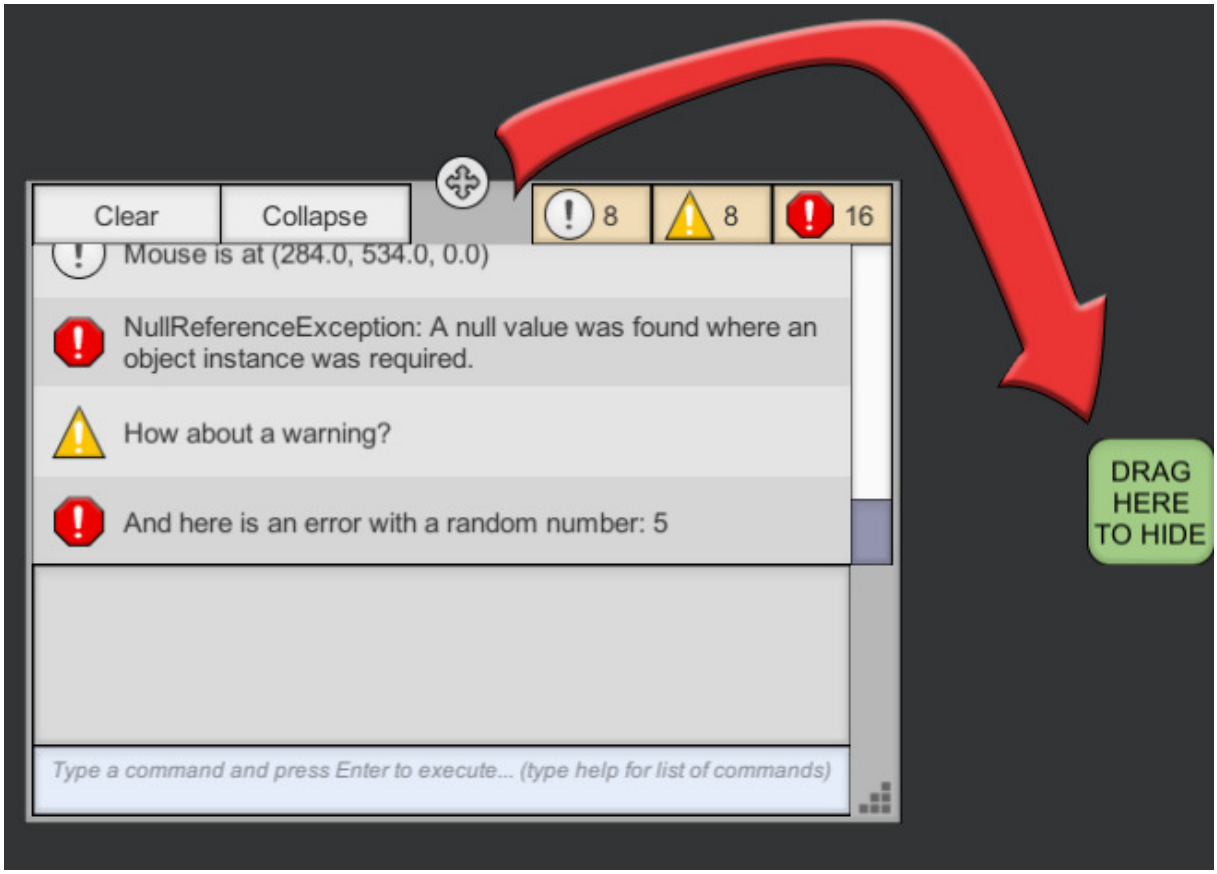


Singleton: if selected, console window will persist between scenes (recommended). If, however, you don't want the console on each scene, then deselect this option and manually drag & drop the **DebugLogCanvas** prefab to the scenes you want.

Clear Command After Execution: if selected, the command input field at the bottom of the console window will automatically be cleared after entering a command. If you want to spam a command, or make small tweaks to the previous command and don't want to write the whole command again, then deselect this option.

Food For Thought: This asset uses **uGUI** with packed sprites and costs only **1 SetPass Call** (and 6 to 10 batches). Number of Instantiate/Destroy calls are heavily optimized using a customized recycled list view.

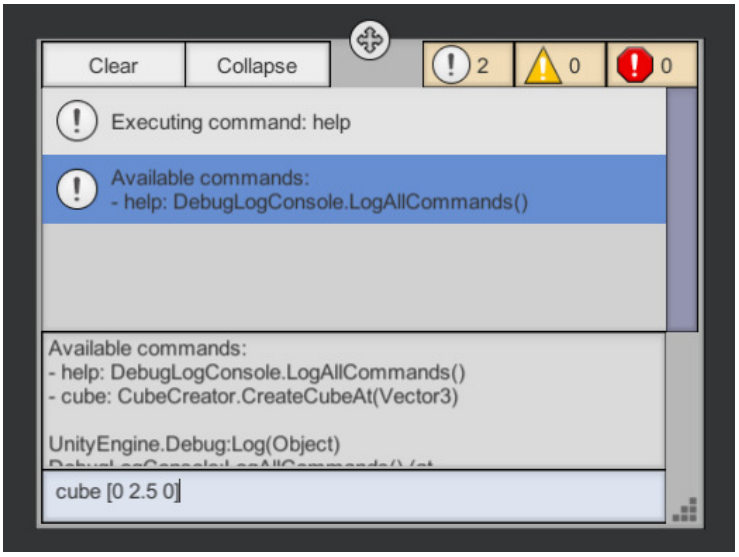
- **Hiding Console Window**



If you don't want the console window to obstruct a big portion of the screen, you can scale it down using the gizmo at the bottom right of the window. To hide the console window completely, drag it using the gizmo at the top-center and drop it onto the little popup at the edge of the screen.

Now, this small popup will stay on screen to notify you of new debug entries. You can drag the popup to reposition it, or simply click it to show the console window again.

▪ Calling Commands



You can enter commands using the input field at the bottom of the console. Initially, only the "*help*" command is available, which lists all the valid commands registered to the console.

A command is basically a function that can be called from the console by typing the corresponding signature into the command input field. This function can be **static** or an **instance function** (non static), in which case, a living instance is required to call the function. The return type of the function can be anything (including void); it does not matter as the console simply ignores the returned object. The function can also take any number of parameters; the only restriction applies to the types of these parameters. Supported parameter types are:

int, float, bool, string, Vector2, Vector3, Vector4

To call a registered command, simply write down the function code and then provide the necessary parameters: "*cube [0 2.5 0]*".

To see the syntax of a command, see the help log: "- *cube: CubeCreator.CreateCubeAt(Vector3)*". Here, function code is "*cube*" and the only necessary parameter is a Vector3. This command calls the CreateCubeAt function in the CubeCreator script (this is a demo script, not available in the asset itself).

The console uses a simple algorithm to parse the command input and has some strict restrictions:

- Put exactly one space character in-between parameters
- Don't put an f character after a float parameter
- Wrap strings with quotation marks (")
- Wrap vectors with square brackets ([]) or normal brackets (())
- Don't put space character after vector opening ([ ) and/or before vector closing ( ])

However, there is some flexibility in the syntax, as well:

- You can provide an empty vector to represent Vector_.zero: []
- You can enter a float as input to an int parameter. The fractional part will automatically be discarded
- You can enter 1 instead of true, or 0 instead of false

▪ Registering Custom Commands

If the parameters of the function you'd like to register meets the criteria mentioned above, you can register it into the console in 2 different ways:

- **Static Functions**

Use "`DebugLogConsole.AddCommandStatic( string command, string methodName, System.Type ownerType )`". Here, command is the function code, methodName is the name of the method in string format, and ownerType is the type of the owner class. It may seem strange to provide the method name in string and/or provide the type of the class; however, after hours of research, I have found it the best way to register any function with any number of parameters and parameter types into the system without knowing the signature of the method.

```
using UnityEngine;

public class CubeCreator : MonoBehaviour
{
    void Start()
    {
            DebugLogConsole.AddCommandStatic( "cube", "CreateCubeAt", typeof(
CubeCreator ) );
    }

    public static void CreateCubeAt( Vector3 pos )
    {
            GameObject obj = GameObject.CreatePrimitive( PrimitiveType.Cube );
            obj.transform.position = pos;
    }
}
```

Food For Thought: Console uses reflection to register and call commands.

- **Instance Functions**

Use "`DebugLogConsole.AddCommandInstance( string command, string methodName, object instance )`":

```
using UnityEngine;

public class CubeCreator : MonoBehaviour
{
    void Start()
    {
            DebugLogConsole.AddCommandInstance( "cube", "CreateCubeAt", this );
    }

    void CreateCubeAt( Vector3 pos )
    {
            GameObject obj = GameObject.CreatePrimitive( PrimitiveType.Cube );
            obj.transform.position = pos;
    }
}
```

The only difference is that, you have to provide an actual instance of the class that owns the function, instead of the type of the class.