



Compatible with Unity3D 3.5 & 4.x

Table of Contents

Installation of Unity Logger	3
Overview & General Use	4-7
Log API	8-10
Exporting Logs to CSV	11
Copying Selected Log Contents	11
Pause and Resume Log Viewer	11
Redirect and Intercept Console	12
Watches	13

Installation of Unity Logger

To install Unity Logger, simply import the project from the Unity Asset Store, it will compile and a new menu will appear after the scripts are ready at: Window -> Log Viewer. Select this option to bring up the Log Viewer window. You may then dock it where ever is handy, grouping it with Console is a good choice.

Next Window	Ctrl+Tab
Previous Window	Ctrl+Shift+Tab
Layouts	▶
Log Viewer	
Scene	Ctrl+1
Game	Ctrl+2
Inspector	Ctrl+3
Hierarchy	Ctrl+4
Project	Ctrl+5
Animation	Ctrl+6
Profiler	Ctrl+7
Particle Effect	Ctrl+8
Asset Store	Ctrl+9
Asset Server	Ctrl+0
Lightmapping	
Occlusion Culling	
Navigation	
Console	Ctrl+Shift+C

If you are installing this outside of the Unity Asset Store, copy the Editor and UnityLogger sub-folders into your projects Assets folder.

The Unity Logger API/Viewer have been verified working with Unity v3.5.6 and Unity 4.0.0 through 4.1.1. Use outside of these versions may not allow for all functionality to work consistently, Unity 4.x will be supported as new versions are released.

Overview and General Use

The primary purpose of Unity Logger is to allow for the following improvements over standard Console and Debug.Log usage (depending on version):

- Provides a means of logging by Category, so that you can select one of the categories provided to group related logging activity. This allows for filtering on category and selectively monitoring different aspects of your game without having unrelated log events creating too much “noise”.
- Provides for a Level, which ranges from Trivial through Critical. This also permits filtering, even within a Category, to locate log messages which are more important than others. Font colour is used to indicate Critical (red) and Important (yellow) log messages, the remaining log messages are in default font colour for your skin preferences (white or black depending on if you are using the pro skin or not).
- Allows for real-time text search of log messages to locate specific log messages, all within the context of your current Category/Level filter. Enter a full or partial keyword and the log list will be narrowed down to logs of interest only.
- Unity Logger tries to “play nice” with the Console, so you can use the ‘Intercept Console’ menu option and have Console messages get logged as well (these will be put under the Console category by default). This also makes them searchable using a keyword filter.
- If necessary, Unity Logger can be ‘disabled’ without losing running log data by selecting the ‘Redirect to Console’ option. Your current log will be left alone while the console is used for output, this can be switched back at will to re-enable logging.
- Pausing the Log Viewer will still collect log messages but allow you to read details for the page of log messages that you have paused in.
- An ‘Export to CSV’ option is included, this will dump the current log (overwriting any previous .csv exports) to the root project folder, under a file named log.csv.

One of the key goals of Unity Logger was to try and provide a solution which felt integrated to the Unity Editor instead of overlaying your graphics view. In this way, you have a dockable window that will look and feel just like the Console, with many new options. The Unity Logger is intended to just start “working” right out of the box, so you can incrementally adjust your existing logging to take advantage of the new API.

Once you have installed Unity Logger and brought up its editor window (Window -> Log Viewer), then you'll see an empty log view, which will look something like the following:



Each component of the Log Viewer is as follows:

1. **Category Selector**

Use this to select which category(ies) you wish to show, it works much like the standard layer selection widget so you can select Nothing, Everything or any combination of categories in between.

2. **Level Selector**

Use this slider to select what level of logs you want to view, as the slider is moved to the right, only logs of increasing importance will show in the list.

3. **Pagination**

This slider will allow you to scroll through the logs, when it has a red background, this means the Log Viewer is paused and will not scroll to show new log messages being added. To unpause you can use the Actions -> Resume Logging menu option, change your filter or just click on a selected log entry again. Also, the keyboard can be used to navigate the Log Viewer, pgup/down, home, end and arrow keys are all supported.

4. **Category**

This shows the category of a particular log message in the list, colour is from log level.

5. **Time**

The time this log message was created, uses log level colour.

6. **Message**

The log message text itself, specified by developer - uses log level colour.

7. **Keyword**

This field allows you to enter text free form, it will then search the message text (Category, Level, Frame and Time are all ignored) and find substring matches to filter down the logs. For instance, if you had some logging where the message included a weapon name, say 'Shotgun', you could enter 'Shotgun' to filter down to just those log messages of interest.

8. **Actions Menu**

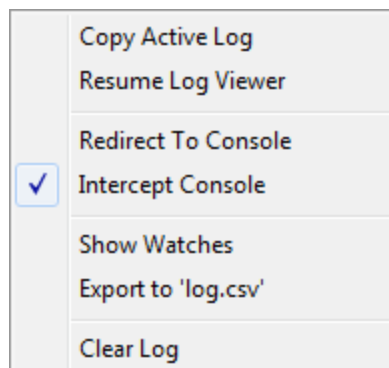
The actions menu provides options to work with the Log Viewer, for more details see the relevant section at the end of this document, the more basic features are discussed here.

Coy Active Log simply transfers a log message into your clipboard, as well as a stack trace if it exists for that log message.

Resume Log Viewer unpauses the Log Viewer, this option will also show up as Pause Log Viewer to pause the view via menu.

Clear Log will purge all log messages from the queue.

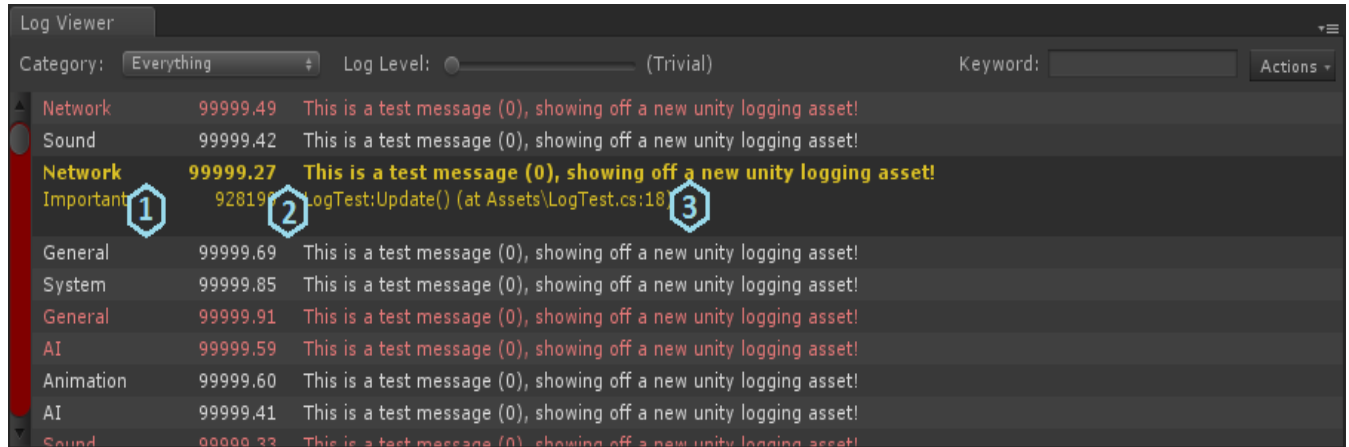
For more details on Redirect, Show Watches, Intercept and Export, see those sections.



9. **Occurrences** Tracking duplicate methods in Unity Logger is slightly different from Console, instead of grouping and losing order, the Unity Logger will only group duplicate messages which are duplicates of the message before it. So if you have alternating duplicates they will not collapse, but if you have groups they will. The number of collapsed messages will be displayed at the far right of the log message row.

When a row is clicked on, the Log Viewer will pause (only the view, logs will still flow into the log queue) so that you can read the log details without them flying by in the case of rapidly scrolling log view. Clicking on the selected row again will release the pause (or changing filter / using the Actions -> Resume Log Viewer).

This will look something like the image below:



The selected row will have a different background colour and provide some additional information, also wrapping text to allow a particularly long message to be viewed more easily.

1. **Level**

Shows the level for this log in text.

2. **Frame**

The Time.frame count when this log message was created.

3. **Stack Trace**

The stack trace for this log message, indicating where this message originated from in your codebase.

Unity Logger API

The Unity Logger API is very straight forward, since the Log class is a non-GameObject singleton which has several static methods which can be called to log new messages under a given Level and Category.

These methods take the following form, variants for an implicit Category (which is always Log.Category.Console) are included. To refer to a specific Category or Level use the enums exposed via Log, for example:

Log.Category.AI : This logs a message under the “AI” category.
Log.Category.System : A “System” category log message.
Log.Level.Minor : This logs a message at the “Minor” level.

Implicit Logging Methods:

Implicit methods can be invoked to just log as a Log.Category.Console category message, their level is the same as the method name:

```
public static void Trivial( string message );  
public static void Minor( string message );  
public static void Normal( string message );  
public static void Important( string message );  
public static void Critical( string message );
```

Log.Critical(“This is a **Critical message targeting the **Console** category.”);**

Explicit Logging Methods:

Explicit methods allow for Category to be specified, similar use to those above just with the additional Category parameter.

```
public static void Trivial( Category category, string message );  
public static void Minor( Category category, string message );  
public static void Normal( Category category, string message );  
public static void Important( Category category, string message );  
public static void Critical( Category category, string message );
```

Log.Important(Log.Category.System, “This is a **Important System message”);**

Generic Logging Methods:

The Generic logging methods allow full control over logging, they range from two convenience methods to a method that takes a LogMessage object itself, which represents a row in the log viewer or CSV export file.

The two convenience methods are:

```
public static void Record( Category category, Level level, string message );  
public static void Record( Category category, Level level, string message, string stack );
```

These methods allow you to create a log message for a given Category, Level and optionally a “stack trace”. The logger itself uses the last variant which accepts a LogMessage object simply because it affords the greatest flexibility, especially where trapping Console messages are concerned. However, you can use them as well - please remember that stack traces will be stored for you using the first method, so unless you’re decorating a stack trace or reusing it somehow, it’s not necessary to use the latter method just for obtaining stack traces.

A couple examples might be:

```
Log.Record( Log.Category.AI, Log.Level.Normal, “A Normal AI message” );  
Log.Record( Log.Category.AI, Log.Level.Normal, “A Normal AI message with stack trace  
included”, UnityEngine.StackTraceUtility.ExtractStackTrace( ) );
```

Finally, the last method for logging looks like so:

```
public static void Record( LogMessage logMessage );
```

It makes use of the LogMessage class, which has the following members:

```
public float time;  
public int frame;  
public Log.Category category;  
public Log.Level level;  
public string message;  
public string stack;  
public int occurrences;
```

In order, these members track the “Time.time” that the message was created, the “Time.frame” or frame count when the message was created, the Category and Level of the message, a user defined message string, the stack trace at the time of creation and the number of times this

message was collapsed due to duplicate messages (meaning the message, stack, Level and Category all matched a neighboring log message).

An example of using this approach might be:

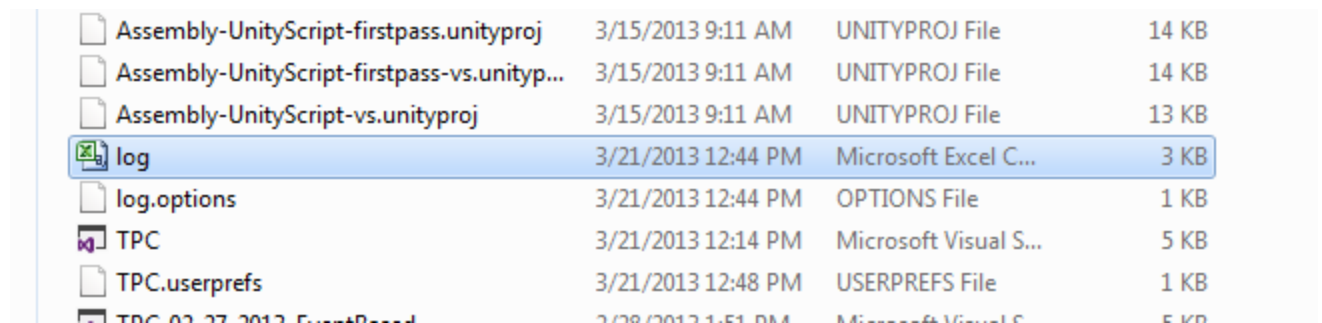
```
Log.Record( new LogMessage( Time.time, Time.frameCount,  
Log.Category.Network, Log.Level.Normal, "A message created by using LogMessage",  
UnityEngine.StackTraceUtility.ExtractStackTrace( ) ) );
```

Please read the section on Watches for more details on using Log.Watch.

Exporting Logs to CSV

When you select the Actions -> Export to CSV option, a log.csv file is created in the root of your project folder. This file is MS Excel friendly and a standard CSV file, so you can open it in Excel to view the contents like any spreadsheet. If you have multiple lines (i.e. stack traces etc), then you'll want to increase cell height to view all of those details.

For example, if my project resides in C:\dev\TPC, after exporting there will be a log.csv file present in this folder (assuming this is the active project during export request), something like this:



Assembly-UnityScript-firstpass.unityproj	3/15/2013 9:11 AM	UNITYPROJ File	14 KB
Assembly-UnityScript-firstpass-vs.unityp...	3/15/2013 9:11 AM	UNITYPROJ File	14 KB
Assembly-UnityScript-vs.unityproj	3/15/2013 9:11 AM	UNITYPROJ File	13 KB
log	3/21/2013 12:44 PM	Microsoft Excel C...	3 KB
log.options	3/21/2013 12:44 PM	OPTIONS File	1 KB
TPC	3/21/2013 12:14 PM	Microsoft Visual S...	5 KB
TPC.userprefs	3/21/2013 12:48 PM	USERPREFS File	1 KB
TPC 03-27-2013 FirstPass	3/28/2013 1:51 PM	Microsoft Visual S...	5 KB

Copying Selected Log Contents

When you select the Actions -> Copy Active Log option, if you have a currently selected log message then it's message and stack trace will be copied into the clipboard. You should then be able to paste those contents into an editor of choice.

Pause and Resume Log Viewer

Pausing and Resuming the Log Viewer happens several ways.

- You use the Actions -> Pause/Resume Log Viewer menu item, this will pause the log viewer until you trigger an unpaue (either via this menu option again or one of the actions below).
- Clicking on a log message will pause the Log Viewer, you can click to compress and unselect that log message to resume the Log Viewer.
- Adjusting filter options will unpaue the log viewer, this includes keyword, category or level.

Redirect and Intercept Console

The Actions menu includes two options for redirecting and intercepting with relation to the Console view. There is also a log.options file created at the root of your project, this is important to store your preferences between runs. It simply contains the current state for Redirect and Intercept.

Redirect to Console will take any incoming log message and redirect it to the Console via a call to Debug.Log, using just the log message as a parameter to it. This will continue until Redirect to Console is turned off or Intercept Console is turned on.

Intercept Console will grab Debug.Log(), exceptions, etc... from the Console and echo them in the logs. This will continue to occur until turned off or Redirect to Console is turned on.

You can also turn off BOTH of these options, in which case Console will be ignored and Unity Logger will handle its own logging business. This can work quite well and is how I use the Log Viewer myself right now, allowing console to catch exceptions in the application and Unity Logger to deal with application logs.

Whenever these options are changed, log.options will be updated so be sure not to open or modify log.options for reliable operation.

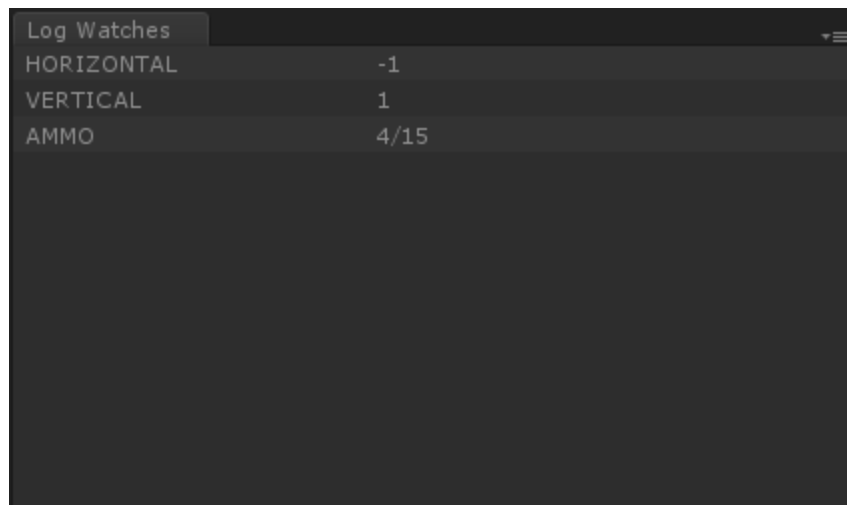
Watches

Another useful feature of Unity Logger is the 'Log Watcher' or "Watches". Basically, these are name value pairs you can set in the Log by using the following syntax:

```
Log.Watch( string nvpName, object nvpValue );
```

A watch will be added in a scrollable area and banded just like the log viewer, in this case however the watch will have its value updated whenever another watch call is made. The watch name used will always be converted to uppercase, so you need only get the same spelling for a watch value to accumulate correctly.

The watch window can be toggled on or off via Actions -> Show Watches / Hide Watches. It will come up as a floating editor window which you can dock if necessary, or just float it where ever is most convenient.



Using watches are useful for debugging current states that change rapidly, for instance the angle of a gun, velocity of an object, etc... When you log a watch, the object you pass has a ToString() call made on it, so if the default ToString() is not sufficient, you'll need to handle formatting yourself, here are a couple examples of simple and self formatted Log.Watch calls.

```
Log.Watch( "colliderheight", myCapsule.height );
```

```
Log.Watch( "playervelocity", string.Format( "{0:F2},{1:F2}", 1f, 0f ) );
```