NLog
(The next evolutionary step in Unity logging)

Made by AiUnity
http://aiunity.com

Table of Contents

Overview

NLog is a comprehensive Unity logging framework based upon the extremely popular open source project .NET NLog project. From the provided editor GUI you have dynamically control of logging levels, platforms, targets, content, and format. With NLog you can route your logging statements based upon class, namespace, or custom name to various targets. The available targets include Unity Console, In-Game Console, Remote Log Viewer, Log File, Email, or create your own. The In-Game Console allows you to debug inside your game while the Remote Log Viewer will have you debugging remotely in minutes.

Each target has a layout property that controls message content and format. Select from the dropdown variable list to inject content like logging level, calling method, and timestamp into log messages. Alternatively use Unity Rich Text formatting to color code messages from a particular class or namespace. The logging level of each logger can also be controlled, allowing you to increase logging verbosity of the namespace(s) under development.

With NLog you can improve performance by compiling out your logging statements on unselected platforms or logging levels. A log tester is built into to the GUI to help you verify and tweak your configurations. The available logging levels are Trace, Debug, Info, Warning, Error, Fatal, and Assert. NLog extensibility make it easy to create new targets and layout variables. Automatically instantiate NLog in new scripts using the AiUnity flagship product ScriptBuilder (Automatically generate any Unity script).

Details

The NLog Unity Editor GUI window presents you with a powerful target/rule based system. The target(s) define the log message destination, the composition of the message, and the format. It is even possible to wrap/group target(s) which enable features like filtering, batch writes, multiple targets. The rule(s) are then created to determine which messages you wish to route to a given target. To start you can simple create one target (ie. UnityConsole) and a catch-all rule that routes to that target. Perhaps during developement you add a rule that increases the debug levels for class MyClassUnderDevelopement. The GUI allows you to choose which platforms should have logging and what levels to log globally. In addition there is a Test Logger built into the GUI so you can quickly evaluate your configuration. Your configuration is stored in XML format that can be viewed and directly edited within the Unity Editor GUI.

At any time and in any environment the GUI or underlying XML can be configured to suite your needs. Drop down menus give you quick access to all available targets and content variables. The Unity console retains the ability to double click log messages to take you to GameObject and source code. With a stunning feature set NLog will lower fustration, provide insight, and speed up development tremedously. NLog is highly extendiable with full source code provided.

With a stunning feature set NLog will lower fustration, provide insight, and speed up development tremedously.

Features

- Complete logging control with a comprehensive NLog Editor GUI.
- Extended log levels are Trace, Debug, Info, Warning, Error, Fatal, and Assert.
- Logging API overloads accept formatted arguments to eliminate string.format().
- Control what messaging level is enable for each platform independently.
- Compile out logging statements on unselected platforms and levels.
- Create rules to direct log messages to various targets on a class, namespace, or platform basis.
- Use target properties to customize message content and destination features.
- The Remote Log Viewer target can send log messages over UDP to a remote viewer.
- The ugui base In-Game console target enables gameplay debug.
- Target wrappers enables advanced features like multiple targets.
- Add message content by using custom text or selectable internal variables.
- Format messages with Unity Rich Text.
- Use the comprehensive GUI to configure NLog or the underlying XML.
- Verify configuration with build in tester.
- Drop down menus give you quick access to targets and content variables.
- GUI themed for light and dark skin.
- Retain the ability to navigate to Source code and GameObject with double clicks.
- Dynamically switch between NLog source code and DLLs.
- Extend NLog with custom targets, wrappers and layout variables.
- Timeless asset in that logging will always be a fundamental debug tool.
- Works with AiUnity products ScriptBuilder, CLog, and MultipleTags.
- Dedicated website and forum available.
- Tested on .NET 3.5 and 4.6.
- Full source code provided.

Setup

1. Install NLog from the asset store.
2. Add logging statements to your code as described in Usage.
3. Configure NLog using the Unity Editor GUI.
4. Enjoy!

Usage

NLog provides a singleton instance of Log Manager locate at "LogManager.Instance". The LogManager is primarly used to get a logger instance via "LogManager.Instance.GetLogger". GetLogger can be passed the logger name, class instance, or both. The logger name uniquely identifies the logger and is what rules match against. The class instance indentifies the associated GameObject to the logger. When class instance is used alone the logger name is assigned the class name. Examples are in the installation directory "Assets/AiUnity/NLog/Examples" or in the NLog Example section below.

Example

```
private void BuildFields()
using UnityEngine;
using AiUnity.NLog.Core;
public class NLogTest : MonoBehaviour
{
    private NLogger logger;
    void Awake() {
        // Passing in the instance will result in logger name being NLogTest and
        // double clicks linked to the GameObject associated with this script.
        logger = LogManager.Instance.GetLogger(this);
    }
    void Start()
 {
        Debug.Log("Test standard Unity debug message", this.gameObject);
        logger.Trace("Test NLog TRACE message");
        logger.Info("Test NLog INFO message");
        logger.Debug("Test NLog Debug message");
        logger.Error("Test NLog ERROR message");
        logger.Assert(false, "Test NLog ASSERT message");
        logger.Trace("Test NLog DEBUG message (Explict Context)", this);
    }
}
```

GUI

The NLog GUI is a Unity Editor window that can be accessed via Tools->AiUnity->NLog->ControlPanel. NLog configuration is stored in an XML file that is maintained by the GUI. The GUI contains tooltips, selection boxes, and references making learning intuitive.

Config file

Shows fixed location "Assets/Rources/NLog.xml" of NLog XML configuration file. The XML configuration can be modified via individual GUI controls below (recommended) or directly via the XML Viewer section. The NLog framework is solely configured by reading the resulting XML file, leaving it independent to the Unity Editor. The +/- icon next to Config File allow for the creation/deletion of NLog.xml respectively. While NLog.xml is absent all log statements are dropped, which may be desired in production.

Source

Specifies if NLog DLLs (recommended) or Source Code is used durring compilation. DLLs compile faster and enable double-click of log messages to bring up corresponding IDE editor line. Source code allows you to investigate, modify, and extend the internal functionality of NLog. This magical feat is accomplished by manipulating global defines and DLL import enable flags.

Platforms

Specifies what Unity platforms will build with NLog logging. All logging statements on unselected platforms will be compiled out of existance. Note the standalone editor platform cannot be unselected.

Build Levels

Specifies active NLog logging levels in Unity builds. These globaly effect which build log levels are accepted and do not effect editor. Logging statements on unselected levels will be compiled out of existance.

Internal Levels

Specifies active NLog logging levels for NLog internal debug messages. Adding Levels Info, Debug, or Trace can be immensily helpful if NLog behaves unexpectedly.

Test Logger

The test logger is used to test your configuration. When the play icon is pressed a logger is initiated and test log statements are excuted. This replicates the procedure that would be done in your classes.

Loger Name

Name of logger to be tested. In code the logger name can be set at logger instantiation or defaults to class name.

Logger Context
GameObject associated with log message (Optional). The gameObject gains focus when console log message are selected.

Logger Levels
A test log statement is generated for each selected level.

Logger Message
The message body of the test log statement.

GUI Targets
The Targets section of the GUI provides the ablility to add/remove targets. The + icon next to Targets brings up a add selection of targets, target wrappers, and target groups. Conversly the - icon will remove the respective target. See Targets for a description of available targets and coresponding settings.

GUI Rules
The Rules section of the GUI provides the ablility to add/remove rule(s). The + icon next to Rules creates a default rule which can be customized. Conversly the - icon will remove the respective rule. See Rules for a description of rule settings.

XML Viewer
View and edit NLog XML configuration file directly. Changes will be reflected in GUI Controls once focus is lost. For brevity target attributes that match the default value are removed. See XML File for a description of XML file.

Targets
The target(s) define the log message destination, content, and the format. All target types and associated parameters provide tooltips to minimize documentation referencing. It is common practice to add the same target type mulitple times with different configurations. For additional reference see the Targets in the .NET NLog project

A target has a set of configuriable parameters in the GUI that governs its behavior. All targets have parameters Name, Type, and usually Layout. The Name is the means it which rules reference targets. The Type is a read-only parameter that indentifies the target type. The Layout determines the content and format of the log message. The Layout + icon provides a quick means of adding variables like timestamp to the log message. The most essential variable is which holds the message provided by the log statement being processed. The Layout Renders section in the open source NLog documentation described many of the available variables. In addtion Unity targets (ie. UnityConsole) accept the use of rich text markup.

File

File logger writes log messages to a designated file name. This target has some advanced features that can archive and auto flush the log file. For additional reference see the File target in the .NET NLog project

GameConsole

GameConsole logger writes log messages to a provided console assest instantiated inside your game. The Console contains a settings menu to configure Font and Log Levels. When not in use it can be disabled or minimize to icon form. Rich text markup is accepted by Game Console. The Game Console is built using the advanced Unity UI released in version 4.6. The console code can easily be modified to meet your functional and astectic needs. Alternatively the assest store has some dedicated game console products.

Mail

Mail logger sends log messages via email using SMTP protocol. In additional to standard email fields you can set SMTP username and password. For additional reference see the Mail target in the .NET NLog project

Memory

Memory logger writes log messages to an ArrayList in memory for programmatic retrieval. For additional reference see the Memory target in the .NET NLog project

MessageBox

MessageBox logger writes messages to a Unity editor dialog box.

MethodCall

MethodCall logger calls a specified static method on each log message with contextual parameters. The method must exist in the namespace/assembly as NLog framework. For additional reference see the MethodCall in the .NET NLog project

NLogViewer

NlogViewer logger sends log messages to a remote instance of an NLog viewer. In theory any viewer supporting Log4J or Log4Net should work fine with NLog. NLog Viewers is a list of viewers, but only Sentinel has been validated.

Sentinel is an open source WPF project and its use/installation is simple. First download/install Sentinel - Log Viewer executiable on any PC that wishes to monitor log files and then follow the procedure below. The procedure basically configures a new Sentinel session that will listen on a specified protocol/port.

1. Run sentinel which will bring up the "Add new logger" wizzard [Next].
2. Choose a name for this Sentinel logger (ie. UnityRemote) [Next].
3. Click "Add" to specify log source and choose "Nlog Viewer Provider"" along with a name (ie. NLogProvider) [Next].
4. Select Protocol UDP and any open port. Lets use 7230 and maybe it will become ours [Finish, Next].
5. Leave view type at "Single traditional text base log" [Finish].
6. Select File->Save Session to avoid repeating this process.

1. In Unity CLog control panel add the NLogViewer target.
2. Enter Sentinel address in the form [protocol]:[IP]:[Port] (ie. udp://[Sentinel PC IP address]:7230).
3. Enjoy!

Trouble Shooting:

- For the first attempt try running Sentinel/Unity on same machine and using IP address 127.0.0.1.
- It is possible the port is taken so try selecting an alternative port between 1024-49151.
- Try disabling the Windows firewall momentarily to see if it is blocking the traffic.

Null

Null logger sends log messages to oblivion never to be seen again. For additional reference see the Null target in the .NET NLog project

UnityConsole

UnityConsole logger writes log messages to the Unity editor console and will likely be your primary/sole target. Rich text markup is accepted by Unity Editor Console. Internally UnityConsole uses Unity Debug log APIs so the console always behaves as expected.

Target wrappers

Target wrappers perform their designated operation before passing the log events(s) to the inner target. Wrappers can only wrap a single target and are documented at [Target Wrappers]. Note the average user will probably not concern themselves with target wrappers and groups.

Async wrapper

Async Wrapper provides asynchronous, buffered execution of writes. For additional reference see the NLogViewer target in the .NET NLog project

Auto flush wrapper

Auto flush wrapper causes a flush after each write. For additional reference see the NLogViewer target in the .NET NLog project

Buffering wrapper

Buffering wrapper buffers log events and sends them in batches. For additional reference see the NLogViewer target in the .NET NLog project

Filtering wrapper

Filtering wrapper filters log entries based on a condition. For additional reference see the Filtering wrapper in the .NET NLog project

Repeating wrapper

Repeating wrapper repeats each log event a specified number of times. For additional reference see the Repeating wrapper in the .NET NLog project

Retrying wrapper

Retrying wrapper retries after write error for specified number of times. For additional reference see the Retrying wrapper in the .NET NLog project

Target groups

Groups implement a designated algorithm to determine which inner target(s) will recieve the log event. Groups must contain 2 or more targets and are documented at [Target Groups]. Note the average user will probably not concern themselves with target wrappers and groups.

Fallback group

Fallback enables the use of alternative loggers when an write error occurs. For additional reference see the NLogViewer target in the .NET NLog project

Randomize group

Randomize group sends log event to a randomly selected target. For additional reference see the Randomize in the .NET NLog project

Round robin group

Round robin group distributes log events to targets in a round-robin fashion. For additional reference see the Round robin group in the .NET NLog project

Split group

Split group writes log events to all targets. For additional reference see the Split group in the .NET NLog project

Rules

A Rule routes matching specific log statement to a designated target. A rule matches against log statement(s) using the fields below with the primary comparison being the Logger Name. Once a rule is matched the log statement is routed to the assigned target. It is permissiable to have multiple rules are a single rule.

Rule name

The specified Name is matched against the logger name and is the primary means of comparison. The logger name is established by the name argument of each GetLogger method call. Alternatively the class instance (this) can be passed to GetLogger and the classname will be used as the name. The specified Name can contain wildcards * at the beginning or end.

Rule target

Determines the target associated with the rule. A drop down list of existing targets is supplied for this field.

Rule levels

The levels matched by this rule. A multi-select drop down list of levels is provided.

Rule platforms

The platforms matched by this rule. A multi-select drop down list of platforms is provided.

Rule final

Indicates to stop rule processing if this rule matches.

Rule enable

Enable this rule.

XML File

The XML configuration file is called NLog.xml and must be located in the Assets/Resources folder. The NLog framework is configured by NLog.xml and works independent of the GUI. The GUI is the recommend method of modifying the configuration, but the XML can be edited directly. Altering the NLog.xml configuration after build/deployment could be a valuable debug feature. If NLog.xml file is not present the default behavior is to drop all log messages.