Log4net

Purpose: Log4net is a logging framework for the .NET platform. It’s definitely not the only one, but it’s one of the most popular frameworks out there.

A logging framework is a tool that can dramatically reduce the burden of dealing with logs.

By adopting a logging framework, it becomes easy to write your logs to different places by simply changing your configuration. You can write your .NET logs to a file on disk, a database, a log management system or potentially dozens of other places, all without changing your code.

**Getting Started:** How to Install log4net Using Nuget

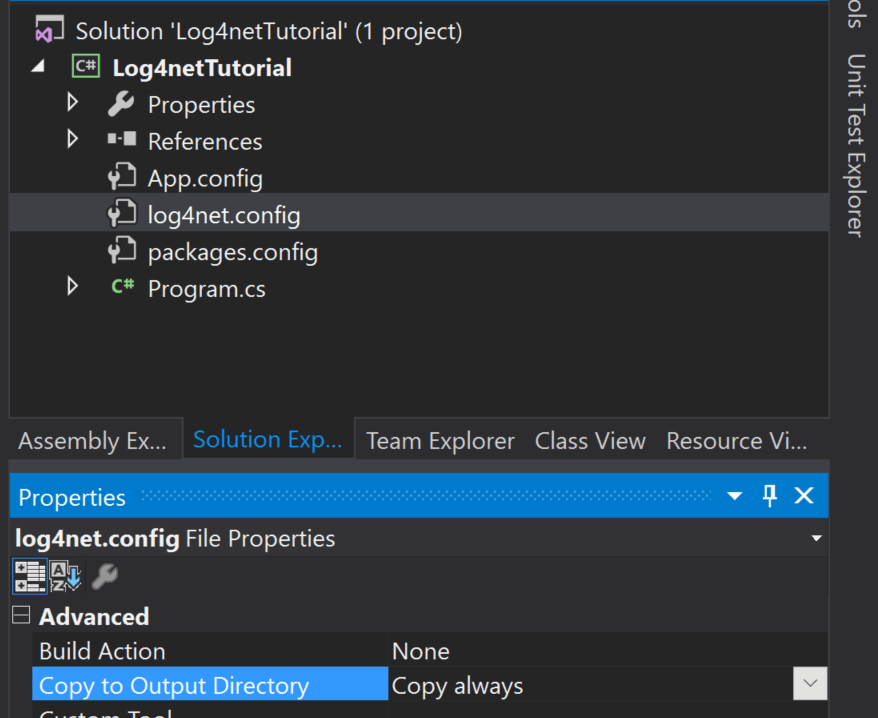
**1. Add log4net Package**

Starting with log4net is as easy as installing a Nuget package. You can use the Visual Studio UI to search for it and install it, or just run this quick command from the Package Manager Console:

PM> Install-Package log4net

**2. Add log4net.config File**

Add a new file to your project in Visual Studio called log4net.config and be sure to set a property for the file. Set Copy to Output Directory to Copy Always. This is important because we need the log4net.config file to be copied to the bin folder when you build and run your app.



To get you started quickly, copy this log4net config and put it in your new log4net.config file. This will log messages to the console and a log file both. We will discuss more about logging appenders further down.

<log4net>

<root>

<level value="ALL" />

<appender-ref ref="console" />

<appender-ref ref="file" />

</root>

<appender name="console" type="log4net.Appender.ConsoleAppender">

<layout type="log4net.Layout.PatternLayout">

<conversionPattern value="%date %level %logger - %message%newline" />

</layout>

</appender>

<appender name="file" type="log4net.Appender.RollingFileAppender">

<file value="myapp.log" />

<appendToFile value="true" />

<rollingStyle value="Size" />

<maxSizeRollBackups value="5" />

<maximumFileSize value="10MB" />

<staticLogFileName value="true" />

<layout type="log4net.Layout.PatternLayout">

<conversionPattern value="%date [%thread] %level %logger - %message%newline" />

</layout>

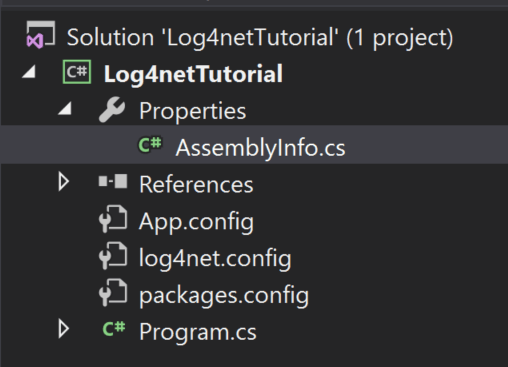
</appender>

</log4net>

**3. Tell log4net to Load Your Config**

The next thing we need to do is tell log4net where to load it’s configuration from so that it actually works. I suggest putting this in your AssemblyInfo.cs file.

You can find it under the Properties section in your project:



Add this to the bottom of your AssemblyInfo file.

[assembly: log4net.Config.XmlConfigurator(ConfigFile = "log4net.config")]

**Advanced Topics & log4net Best Practices**

**1. Define Your LogManager Object as Static**

Declaring any variable in your code has overhead. When I have been doing some profiling sessions in the past to optimize code, I have noticed that the constructors on the LogManager object can use a lot of CPU.

Declare it as static and use this little trick so you don’t have to hard code the class type.

private static readonly log4net.ILog log = log4net.LogManager.GetLogger(System.Reflection.MethodBase.GetCurrentMethod().DeclaringType);

**2. How to Enable log4net’s Own Internal Debug Logging**

From time to time, you may have problems with a specific appender, or issues working with it. To help resolve these issues, enable internal log4net logging via your web.config file.

<configuration>

<appSettings>

<add key="log4net.Internal.Debug" value="true"/>

</appSettings>

</configuration>

You can then specify where the logging is written to.

<configuration>

...

<system.diagnostics>

<trace autoflush="true">

<listeners>

<add

name="textWriterTraceListener"

type="System.Diagnostics.TextWriterTraceListener"

initializeData="C:\tmp\log4net.txt" />

</listeners>

</trace>

</system.diagnostics>

...

</configuration>

**3. Do Not Send Your Logs to a Database Table with the AdoAppender**

Trying to query logs in SQL is very difficult if you log any real volume. You are much better off sending your logs to Elasticsearch or a log management service that can provide full-text indexing and more functionality with your logs.

**4. Do Not Send Emails on Every Exception**

The last thing you want to do is send any sort of emails from an appender. They either get ignored over time or something starts throwing a lot of exceptions and then your app starts sending thousands of errors. Although, there is an SmtpAppender if you really want to do this.

**5. How to Send Alerts for Exceptions**

If you want to send alerts about exceptions, send your exceptions to an error tracking product, like Retrace, which is designed for this.

They can also de-dupe your errors so you can figure out when an error is truly new, track its history, and track error rates.

**6. Send Your Logs to a Log Management System to View Them Across Servers**

Capturing logs and logging them to a file on disk is great. But if you want to search your logs across multiple servers and applications, you need to send all of your logs to a central repository.

There are a lot of log management solutions that can help you with this, or you can even setup your own elasticsearch cluster for it.

If you want to query all the files on disk, consider using VisualLogParser.

**7. Use Filters to Suppress Certain Logging Statements**

Filters can be configured to suppress specific log messages. Take a look at these examples.

Here’s how you can filter by the text on the log messages.

<filter type="log4net.Filter.StringMatchFilter"> <stringToMatch value="test" /> <!-- Can filter by string or regex --></filter>

And here, you can filter by the log level:

<filter type="log4net.Filter.LevelRangeFilter"> <levelMin value="INFO" /> <levelMax value="FATAL" /></filter>

**8. You Can Make Your Own Custom log4net Appenders**

If you want to do something that the standard appenders do not support, you can search online for one or write your own.

One example could be an appender for writing to Azure Storage. Once upon a time, we wrote one to send our logs to Azure Table storage to centralize them. We couldn’t really query them due to the lack of full-text indexing, but we could view them.

As an example of a custom appender, you can review the source code for our appender for sending logs to Retrace.

**9. Customize Your Layout in Your Logs with log4net Pattern Layouts**

You can modify your configuration to change what fields are outputting and in what format using pattern layouts.

<appender name="LogFileAppender" type="log4net.Appender.RollingFileAppender">

<param name="File" value="stackify.log" />

<param name="AppendToFile" value="true" />

<rollingStyle value="Size" />

<maxSizeRollBackups value="10" />

<maximumFileSize value="10MB" />

<staticLogFileName value="true" />

<layout type="log4net.Layout.PatternLayout">

<param name="ConversionPattern" value="%-5p %d{MM-dd hh:mm:ss.ffff} [%thread] %m%n" />

</layout>

</appender>

Using the layout above, write the level (%p), current date time (%d), thread # (%thread), the message (%m) and a new line (%n). The -5 in the %-5p is to set the width of the field to 5 characters.

Here are some other notable fields you can log, although they can have a big performance impact to your app and would not be recommended for high volume logging on a production application.

%method: name of the method where the log message was written

%stacktrace{level}: output a stack trace to show where the log message was written

%type: type of the caller issuing the log request. Mostly likely your class name

%line: the line number from where your logging statement was logged

A layout like this:

<layout type="log4net.Layout.PatternLayout">

<param name="ConversionPattern" value="%-5p%d{ yyyy-MM-dd HH:mm:ss} – [%thread] %m method:%method %n stacktrace:%stacktrace{5} %n type:%type %n line: %line %n" /></layout>

Produces a log message like this:

ERROR 2017-02-06 09:38:10 – [10] Error downloading web request method:ThrowWebException stacktrace:Microsoft.VisualStudio.HostingProcess.HostProc.RunUsersAssembly > System.AppDomain.ExecuteAssembly > System.AppDomain.\_nExecuteAssembly > ConsoleApplication1.Program.Main > ConsoleApplication1.Class1.ThrowWebException type:ConsoleApplication1.Class1 line: 26

**10. Use the Diagnostic Contexts to Log Additional Fields**

You can also log custom fields to help give some more context about the user, customer or transaction related to the log statements.

The below example sets a custom property called customer. You can then modify your log4net pattern layout to include %property{customer} to output it in your logs.

**log4net.ThreadContext.Properties["customer"] = "My Customer Name";**

**log.Debug("We are going to try and do a web request");**

**try**

**{**

**Class1.ThrowWebException();**

**}**

**catch (Exception ex)**

**{**

**log.Error("Error trying to do something", ex);**

**}**

**log.Debug("We are done with the web request");**

**11. How to Correlate Log Messages by Web Request Transaction**

Additionally, you can assign objects in contexts to use what it calls “active property values.” When the log message is written, the ToString() method will be called which can dynamically do something.

This can be used to write transaction IDs to help correlate messages to a web request or transaction!

//Create our little helper class

public class ActivityIdHelper

{

public override string ToString()

{

if (Trace.CorrelationManager.ActivityId == Guid.Empty)

{

Trace.CorrelationManager.ActivityId = Guid.NewGuid();

}

return Trace.CorrelationManager.ActivityId.ToString();

}

}

In your global.asax or Startup.cs class, subscribe to an event for when a request first starts

public override void Init()

{

base.Init();

this.Error += WebApiApplication\_Error;

this.BeginRequest += WebApiApplication\_BeginRequest;

this.EndRequest += WebApiApplication\_EndRequest;

}

void WebApiApplication\_BeginRequest(object sender, EventArgs e)

{ //set the property to our new object

log4net.LogicalThreadContext.Properties"activityid"] = new ActivityIdHelper(); log.Debug("WebApiApplication\_BeginRequest"); }