# Using log4net for logging and audit trail activities

# Introduction

The [Apache log4net library](http://logging.apache.org/log4net/release/manual/introduction.html) is a tool to help the programmer output log statements to a variety of output targets. log4net is a port of the excellent Apache log4j™ framework to the Microsoft® .NET runtime. **It** kept the framework similar in spirit to the original log4j while taking advantage of new features in the .NET runtime.

# The Basics

There are a three parts to log4net. There is the configuration, the setup, and the call. The configuration is typically done in the app.config or web.config file. We will go over this in depth below. If you desire more flexibility through the use of a separate configuration file, see the section titled "Getting Away from app.config". Either way you choose to store the configuration information, the code setup is basically a couple of lines of housekeeping that need to be called in order to set up and instantiate a connection to the logger. Finally, the simplest part is the call itself. This, if you do it right, is very simple to do and the easiest to understand.

# Logging Levels

There are seven logging levels, five of which can be called in your code. They are as follows (with the highest being at the top of the list):

1. OFF - nothing gets logged (cannot be called)
2. FATAL
3. ERROR
4. WARN
5. INFO
6. DEBUG
7. ALL - everything gets logged (cannot be called)

These levels will be used multiple times, both in your code as well as in the config file. There are no set rules on what these levels represent (except the first and last).

# The Configuration

The standard way to set up a log4net logger is to utilize either the *app.config* file in a desktop application or the *web.config* file in a web application. There are a few pieces of information that need to be placed in the *config* file in order to make it work properly with log4net. These sections will tell log4net how to configure itself. The settings can be changed without re-compiling the application, which is the whole point of a *config* file.

**Root**

You need to have one root section to house your top-level logger references. These are the loggers that inherit information from your base logger (root). The only other thing that the root section houses is the minimum level to log. Since everything inherits from the root, no appenders will log information below that specified here. This is an easy way to quickly control the logging level in your application. Here is an example with a default level of INFO (which means DEBUG messages will be ignored) and a reference to two appenders that should be enabled under root:

<root>

<level value="INFO"/>

<appender-ref ref="FileAppender"/>

<appender-ref ref="ConsoleAppender" />

</root>

**Additional Loggers**

Sometimes you will want to know more about a particular part of your application. Log4net anticipated this by allowing you to specify additional logger references beyond just the root logger. For example, here is an additional logger that I have placed in our config file to log to the console messages that occur inside the OtherClass class object:

<logger name=”Log4NetTest.OtherClass”>

<level value=”DEBUG”/>

<appender-ref ref=”ConsoleAppender”/>

</logger>

Note that the logger name is the full name of the class including the namespace. If you wanted to monitor an entire namespace, it would be as simple as listing just the namespace you wanted to monitor. I would recommend against trying re-using appenders in multiple loggers. It can be done, but you can get some unpredictable results.

**ConfigSections**

In a config file where there will (potentially) be more information stored beyond just the log4net configuration information, you will need to specify a section to identify where the log4net configuration is housed. Here is a sample section that specifies that the configuration information will be stored under the XML tag "log4net":

<configSections>

<section name="log4net"

type="log4net.Config.Log4NetConfigurationSectionHandler, log4net"/>

</configSections>

**Appender**

An appender is the name for what logs the information. It specifies where the information will be logged, how it will be logged, and under what circumstances the information will be logged. While each appender has different parameters based upon where the data will be going, there are some common elements. The first is the name and type of the appender. Each appender must be named (anything you want) and have a type assigned to it (specific to the type of appender desired). Here is an example of an appender entry:

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

**Layout**

Inside of each appender must be a layout section. This may be a bit different depending on the type of

appender being used, but the basics are the same. You need a type that specifies how the data will be written. There are multiple options, but the one that I suggest you use is the pattern layout type. This will allow you to specify how you want your data written to the data repository. If you specify the pattern layout type, you will need a sub-tag that specifies a conversion pattern. This is the pattern by which your data should be written to the data repository. I will give a more detailed description of your options for the conversion patterns, but for now, here is an example of the layout tag with the pattern layout specified:

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

<conversionPattern value="%date [%thread] %-5level %logger [%ndc]

- %message%newline"/>

</layout>

# Filters

Filters are another big part of any appender. With a filter, you can specify which level(s) to log and you can even look for keywords in the message. Filters can be mixed and matched, but you need to be careful when doing so. When a message fits inside the criteria for a filter, it is logged and the processing of the filter is finished. This is the biggest gotcha of a filter. Therefore, ordering of the filters becomes very important if you are doing a complex filter.

**StringMatchFilter**

The string match filter looks to find a specific string inside of the information being logged. You can have multiple string match filters specified. They work like OR statements in a query. Here is an example of how to filter for entries that have *test* in their message:

<filter type="log4net.Filter.StringMatchFilter">  
<stringToMatch value="test" />  
</filter>

**LevelRangeFilter**

A level range filter tells the system to only log entries that are inside of the range specified. This range is inclusive, so in the below example, events with a level of INFO, WARN, ERROR, or FATAL will be logged, but DEBUG events will be ignored. You do not need the deny all filter after this entry since the deny is implied.

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

**LevelMatchFilter**

The level match filter works like the level range filter, only it specifies one and only one level to capture. However, it does not have the deny built into it so you will need to specify the deny all filter after listing this filter.

<filter type="log4net.Filter.LevelMatchFilter">  
<levelToMatch value="ERROR"/>  
</filter>

**DenyAllFilter**

The only purpose of this entry is to specify that no log entry should be made.

<filter type="log4net.Filter.DenyAllFilter" />

# Appenders

Appenders are noting but the defining the logger output destinations .

**Console Appender**

I use this appender for testing usually, but it can be useful in production as well. It writes to the output window, or the command window if you are using a console application.

**File Appender**

This appender will write to a text file. The big differences to note here are that we have to specify the name of

**Rolling File Appender**

This is an appender that should be used in place of the file appender whenever possible. The purpose of the rolling file appender is to perform the same functions as the file appender but with the additional option to only store a certain amount of data before starting a new log file.

**ADO.NET Appender**

Here is the tricky one. This specific example writes to SQL, but you can write to just about any database you want using this pattern. Note that the connectionType is basically a connection string, so modifying it is simple. The commandText specified is a simple query. You can modify it to any type of INSERT query that you want (or Stored Procedure)

# How to Integrate

Once you have a reference to the log4net DLL in your application, there are three lines of code that you need to know about. The first is a one-time entry that needs to be placed outside of your class. I usually put it right below my using statements in the *Program.cs* file. You can copy and paste this code since it will probably never need to change (unless you do something unusual with your *config* file). Here is the code:

[assembly: log4net.Config.XmlConfigurator(Watch = true)]

The next entry is done once per class. It creates a variable (in this case called "log") that will be used to call the log4net methods. This code is also code that you can copy and paste (unless you are using the Compact Framework). It does a System.Reflection call to get the current class information. This is useful because it allows us to use this code all over but have the specific information passed into it in each class. Here is the code:

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

The final code piece is the actual call to log some piece of information. This can be done using the following code:

log.Info("Info logging");

Notice that you can add an optional parameter at the end to include the exception that should be logged. Include the entire exception object if you want to use this option. The call is very similar, and it looks like this:

log.Error("This is my error", ex);

ex is the exception object. Remember that you need to use the *%exception* pattern variable in your appender to actually capture this exception information.

**Logging Extra Data**

Using the basic configuration in log4net usually includes enough information for a typical application. However, sometimes you want to record more information in a standard way. For example, if you use the ADO.NET appender, you may want to add a field for application user name instead of just including it in the message field. There isn't a conversion pattern that matches up with the application user name. However, you can use the Context properties to specify custom properties that can be accessed in the appenders. Here is an example of how to set it up in code:

log4net.GlobalContext.Properties["testProperty"] = "This is my test property information";

There are a couple of things to notice. First, I named the property "testProperty". I could have named it anything. However, be careful because if you use a name that is already in use, you may overwrite it. This leads into the second thing to note. I referenced the GlobalContext, but there are four contexts that can be utilized. They are based upon the threading. Global is available anywhere in the application where Thread, Logical Thread, and Event restrict the scope further and further. You can use this to store different information based upon the context of where the logger was called. However, if you have two properties with the same name, the one that is in the narrower scope will win. Looking at our first point again, we can see the issue that this might cause. If we declare a GlobalContext property that has the same property name as an existing ThreadContext, we may not see the property value we expect because of the existing value. For this reason, I would suggest developing your own naming scheme that will not conflict with anyone else's names. Here is an example of how to capture this property in our appender:

<layout type="log4net.Layout.PatternLayout">  
<conversionPattern value="%date{ABSOLUTE} [%thread] %level%logger - %message%newlineExtra Info: %property{testProperty}%newline%exception"/>  
</layout>

Tags

log4net,log,audit trail,error logging

References

<http://logging.apache.org/log4net/release/manual/introduction.html>  
<http://www.codeproject.com/Articles/140911/log4net-Tutorial?display=Print>