

Log4Net Overview

Introduction

Log4Net is an open source a .NET based set of assemblies, ported from the Apache Log4j java classes, to provide a configurable logging framework. The classes allow for different levels of logging based upon a hierarchical set of methods. Each module that utilise Log4Net can allow separate areas of code to be configured for different levels of logging. In addition there is a flexibility to allow class and namespace level logging if required. Log4Net is configurable so that the format, type of log, log levels etc are not hard coded and can be changed in a configuration file at run time (i.e. no program restart is required).

Installation

Copy the assembly files to a location on your hard drive and in Visual Studio, add a reference to the log4net.dll assembly.

Code Changes

In order to utilise these classes the following needs adding to each class that requires logging:

```
using log4net;
using log4net.Config;

namespace LoggerTest
{
    public class Form1 : System.Windows.Forms.Form
    {
        private static readonly ILog log =
            LogManager.GetLogger (typeof(Form1));
        .
        .
        .
    }
}
```

The `LogManager.GetLogger` line tells Log4Net that you want to create a new logger and that it identified by the `typeof(Form1)` parameter. Which in this case is `LoggerTest.Form1`, this tag will be used later during Log4Net configuration.

Each application will also need an additional line during startup that defines the configuration of Log4Net which is as follows:

```
static void Main()
{
    XmlConfigurator.ConfigureAndWatch(new
        System.IO.FileInfo(@"logConfig.XML"));
    Application.Run(new Form1());
}
```

The line `XmlConfigurator.ConfigureAndWatch`, configures Log4Net by using the `logConfig.xml` file and will watch the file for changes so that if the configuration file is changed Log4Net will reconfigure itself based upon the new configuration information without the need for a restart.

It is also useful to turn off logging when the application closes by calling the log4Net shutdown methods. This allows all Appenders to be shut down correctly.

```
protected override void Dispose( bool disposing )
{
    log.Warn(string.Format("Application ended {0}"
        ,System.DateTime.Now.ToString("dd-MMM-yyy hh:mm:ss")));
    log4net.LogManager.Shutdown();
}
```

Using Log4Net is now fairly easy to use. There are 5 levels of logging:

- Debug
- Info
- Warn
- Error
- Fatal

To log at these levels call the relevant functions e.g.

```
log.Info("Form1 button1_Click Start");
try
{
    if(checkBox1 == null)
    {
        log.Fatal("checkBox1 is null");
        return;
    }

    if(!checkBox1.Checked)
    {
        log.Debug("Form1 button1_Click : Checkbox not checked");
    }
    else
    {
        log.Warn("Form1 button1_Click : Checkbox checked");
    }
}
catch (Exception ex)
{
    log.Error(ex.Message);
}
log.Info("Form1 button1_Click End");
```

Configuration File

The Log4Net configuration file is XML based and is used to configure the Level of logging, the areas to log, the types of logging required. A sample configuration is as follows:

```
<log4net>
  <!-- A1 is set to be an OutputDebugStringAppender -->
  <appender name="A1"
    type="log4net.Appender.OutputDebugStringAppender">

    <!-- A1 uses PatternLayout -->
    <layout type="log4net.Layout.PatternLayout">
      <conversionPattern value="%d [%t] %-5p %C.%M - %m%n" />
    </layout>
  </appender>
```

```

<!-- Set root logger level to DEBUG and its appender to A1 -->
<root>
  <level value="DEBUG" />
  <appender-ref ref="A1" />
</root>
<log4net>

```

There are 2 parts to this file: The appender and the log level. In this example the appender is set to be an [OutputDebugStringAppender](#), which will log to DebugView. The log level is set to `DEBUG`, which is the lowest level of logging and will also log all other levels. If the log level was set to `FATAL`, only logs at the fatal level will be logged.

This log will produce a log in the following format:

```
DEBUG 28064 LoggerTest.Form1 - Form1 button1_Click : Checkbox checked
```

This format is defined by the `conversionPattern` parameter. The format options are as follows:

Type	Description
%c	Used to output the category of the logging event. The category conversion specifier can be optionally followed by precision specifier, that is a decimal constant in brackets. If a precision specifier is given, then only the corresponding number of right most components of the category name will be printed. By default the category name is printed in full. For example, for the category name "a.b.c" the pattern %c{2} will output "b.c".
%C	Used to output the fully qualified class name of the caller issuing the logging request. This conversion specifier can be optionally followed by precision specifier, that is a decimal constant in brackets. If a precision specifier is given, then only the corresponding number of right most components of the class name will be printed. By default the class name is output in fully qualified form. For example, for the class name "org.apache.xyz.SomeClass", the pattern %C{1} will output "SomeClass". WARNING Generating the caller class information is slow. Thus, it's use should be avoided unless execution speed is not an issue.
%d	Used to output the date of the logging event. The date conversion specifier may be followed by a date format specifier enclosed between braces. For example, %d{HH:mm:ss,SSS} or %d{dd MMM yyyy HH:mm:ss,SSS}. If no date format specifier is given then ISO8601 format is assumed.
%F	Used to output the file name where the logging request was issued. WARNING Generating caller location information is extremely slow. It's use should be avoided unless execution speed is not an issue.
%l	Used to output the line number from where the logging request was issued. WARNING Generating caller location information is extremely slow. It's use should be avoided unless execution speed is not an issue.
%m	Used to output the application supplied message associated with the logging event.
%M	Used to output the method name where the logging request was issued. WARNING Generating caller location information is extremely slow. It's use should be avoided unless execution speed is not an issue.
%n	Outputs the platform dependent line separator character or characters. This conversion character offers practically the same performance as using non-portable line separator strings such as "\n", or "\r\n". Thus, it is the preferred way of specifying a line separator.
%p	Used to output the priority of the logging event.

%r	Used to output the number of milliseconds elapsed since the start of the application until the creation of the logging event.
%t	Used to output the name of the thread that generated the logging event.
%%	The sequence %% outputs a single percent sign.

The conversion pattern "%d{dd-MM-yyyy HH:mm:ss} [%t] %-5p %c:%m%n" will be displayed as:

24-02-2005 15:47:24 [5984] INFO LoggingCheck.Form2:Form2 Constructor

Multiple appenders can be attached so that data can be logged to files, debug view, databases etc. For a list of Appenders see later. To use multiple appenders see the following:

```
<log4net>
  <!-- A1 is set to be a ConsoleAppender -->
  <appender name="A1"
    type="log4net.Appender.OutputDebugStringAppender">

    <!-- A1 uses PatternLayout -->
    <layout type="log4net.Layout.PatternLayout">
      <conversionPattern
        value="%d{dd-MM-yyyy HH:mm:ss} [%t] %-5p %c:%m%n" />
    </layout>
  </appender>
  <appender name="RollingFile"
    type="log4net.Appender.RollingFileAppender">
    <file value="example.log" />
    <rollingStyle value="Size" />
    <appendToFile value="true" />
    <maximumFileSize value="100KB" />
    <maxSizeRollBackups value="2" />

    <layout type="log4net.Layout.PatternLayout">
      <conversionPattern
        value="%d{dd-MM-yyyy HH:mm:ss} [%t] %-5p %c:%m%n" />
    </layout>
  </appender>

  <!-- Set root logger level to DEBUG and its only appender to A1 -->
  <root>
    <level value="DEBUG" />
    <appender-ref ref="A1" />
    <appender-ref ref="RollingFile" />
  </root>
</log4net>
```

This example will log to DebugView and to a file called example.log which is a rolling file of max size 100K

Logging to a class/namespace level can be done as follows:

```
<root>
  <level value="FATAL" />
  <appender-ref ref="A1" />
  <appender-ref ref="RollingFile" />
</root>
<logger name = "LoggingCheck.Form2">
  <level value="INFO" />
</logger>
```

This will log the `LoggingCheck.Form2` class at `INFO` level and everything else at `FATAL`.

To log at the namespace level change `LoggingCheck.Form2` to `LoggingCheck`.

Logging can also be directed to different appenders depending upon the logger used.

```
<logger name="LoggingCheck.Form1">
  <level value="Warn" />
  <appender-ref ref="RollingFile2" />
</logger>
```

Adding this logger will log `LoggingCheck.Form1` to the `RollingFile2` appender, which will also need defining in your config file.

Note:

Logging as `DEBUG` will log all logs.

Logging as `INFO` will log `INFO`, `WARN`, `ERROR` and `FATAL`.

Logging as `WARN` will log `WARN`, `ERROR` and `FATAL`.

Logging as `ERROR` will log `ERROR` and `FATAL`.

Logging as `FATAL` will log `FATAL` only.

Logging as `ALL` will log all logs.

Logging as `OFF` will log turn off logs.

Configuration in App.Config/Web.Config

Using a separate config file for log4net has its advantages, especially if log4net is configured to watch a config file. Using a separate config file means that log4net will only be reconfigured when the log4net config file changes and not when other config is changed. The log4net configuration can also be added to the web.config or app.config files.

Modify `AssemblyInfo.cs` and add

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

in `web.config/app.config` specify a new `configSection`

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <configSections>
    <section name="log4net"
      type="log4net.Config.Log4NetConfigurationSectionHandler, log4net"
    />
  </configSections>
  .
  .
  <log4net>
    <!-- Add Log4Net Config Here -->
  </ log4net >
</configuration>
```

If you are using log4net on a web application then the folder that contains the log files needs to be have aspnet user write permissions.

Compact Framework

Log4Net supports the compact framework as a separate build. When a Smart Device Application is created in Visual Studio, add in the reference to the netcf version of the dll.

The compact framework version does not support the following:

`XmlConfigurator.ConfigureAndWatch` use `XmlConfigurator.Configure` instead. This means that the configuration file can not be changed on the fly and requires the application to be restarted before any changes will be implemented.

Types of Appender

Below is a list of Appenders that are available

(from <http://logging.apache.org/log4net/release/manual/introduction.html>)

Type	Description
<code>log4net.Appender.ADONetAppender</code>	Writes logging events to a database using either prepared statements or stored procedures.
<code>log4net.Appender.ASPNetTraceAppender</code>	Writes logging events to the ASP trace context. These can then be rendered at the end of the ASP page or on the ASP trace page.
<code>log4net.Appender.BufferingForwardingAppender</code>	Buffers logging events before forwarding them to child appenders.
<code>log4net.Appender.ColoredConsoleAppender</code>	Writes logging events to the application's Console. The events may go to either the standard out stream or the standard error stream. The events may have configurable text and background colors defined for each level.
<code>log4net.Appender.ConsoleAppender</code>	Writes logging events to the application's Console. The events may go to either the standard out stream or the standard error stream.
<code>log4net.Appender.EventLogAppender</code>	Writes logging events to the Windows Event Log.
<code>log4net.Appender.FileAppender</code>	Writes logging events to a file in the file system.
<code>log4net.Appender.ForwardingAppender</code>	Forwards logging events to child appenders.
<code>log4net.Appender.MemoryAppender</code>	Stores logging events in an in memory buffer.
<code>log4net.Appender.NetSendAppender</code>	Writes logging events to the Windows Messenger service. These messages are displayed in a dialog on a users terminal.
<code>log4net.Appender.OutputDebugStringAppender</code>	Writes logging events to the debugger. If the application has no debugger, the system debugger displays the string. If the application has no debugger and the system debugger is not active, the message is ignored.
<code>log4net.Appender.RemotingAppender</code>	Writes logging events to a remoting sink using .NET remoting.
<code>log4net.Appender.RollingFileAppender</code>	Writes logging events to a file in the file system. The RollingFileAppender can be configured to log to multiple files based upon date or file size constraints.

log4net.Appender.SMTPAppender	Sends logging events to an email address.
log4net.Appender.SmtpPickupDirAppender	Writes SMTP messages as files into a pickup directory. These files can then be read and sent by an SMTP agent such as the IIS SMTP agent.
log4net.Appender.TraceAppender	Writes logging events to the .NET trace system.
log4net.Appender.UdpAppender	Sends logging events as connectionless UDP datagrams to a remote host or a multicast group using a UdpClient.

Writing your own appender

Using the log4Net SDK you can create your own appender. Add a reference to the log4net dll and create a class that derives from `log4net.Appender.IAppender`

```
public class MyAppender : log4net.Appender.IAppender
{
    public MyAppender ()
    {
        // TODO: put your construction code here
    }

    // The IAppender interface methods
    #region IAppender Members
    public void Close()
    {
        // TODO : put your code here to stop the appender
        // e.g. stop any threads running etc.
    }

    public void DoAppend(log4net.spi.LoggingEvent loggingEvent)
    {
        if (loggingEvent != null)
        {
            // TODO: put your code here to log
        }
    }

    public string Name
    {
        get { return _name; }
        set { _name = value; }
    }
    private string _name = "MyAppendersAppenderDefaultName";
    #endregion

    // The following properties are set automatically by
    // log4net using the XML configuration properties.
    // These have been copied from the
    // Recneps.Logging.WebServiceAppender.
    // Download from http://www.recneps.co.uk.

    /// <summary>
    /// <param name="Url" value="http://www.yourwebservice/wsdlfile/" />
    /// </summary>

```

```

public string Url
{
    get { return _url; }
    set { _url = value; }
}
private string _url =
    "http://www.recneps.co.uk/webservices/logging/log.php";

/// <summary>
/// <param name="ApplicationName" value="YourAppName" />
/// </summary>
public string ApplicationName
{
    get { return _ApplicationName; }
    set { _ApplicationName = value; }
}
private string _ApplicationName = "Default";

/// <summary>
/// <param name="Threshold" value="ERROR" />
/// </summary>
public string Threshold
{
    get { return _Threshold; }
    set { _Threshold = value.ToUpper(); }
}
private string _Threshold = "OFF";

// TODO : Add in the rest of your appender code
}

```

To configure your web service add the following to your config file.

```

<appender name="WebServiceAppender"
    type="Recneps.Logging.WebServiceAppender, Recneps.Logging">
    <param name="Url"
        value="http://www.recneps.biz/logging/loggingservice.php" />
    <param name="Threshold" value="Error" />
    <param name="ApplicationName" value="MyTestApplicaion" />
</appender>

```

The `type` property is a comma-separated list as follows:

```
type="<appender class name>, <assembly name without dll extension>"
```

In the configuration example above the `appender class name` is `Recneps.Logging.WebServiceAppender` and the `assembly name without dll extension` is `Recneps.Logging`

Further Information

See <http://sourceforge.net/projects/log4net>
<http://logging.apache.org/log4net/>

Original Java implementation
<http://logging.apache.org/log4j/docs/index.html>

All ports
<http://logging.apache.org/>