# LEARNING log4net

#log4net

# Table of Contents

# About

# Chapter 1: Getting started with log4net

## Remarks

This section provides an overview of what log4net is, and why a developer might want to use it.

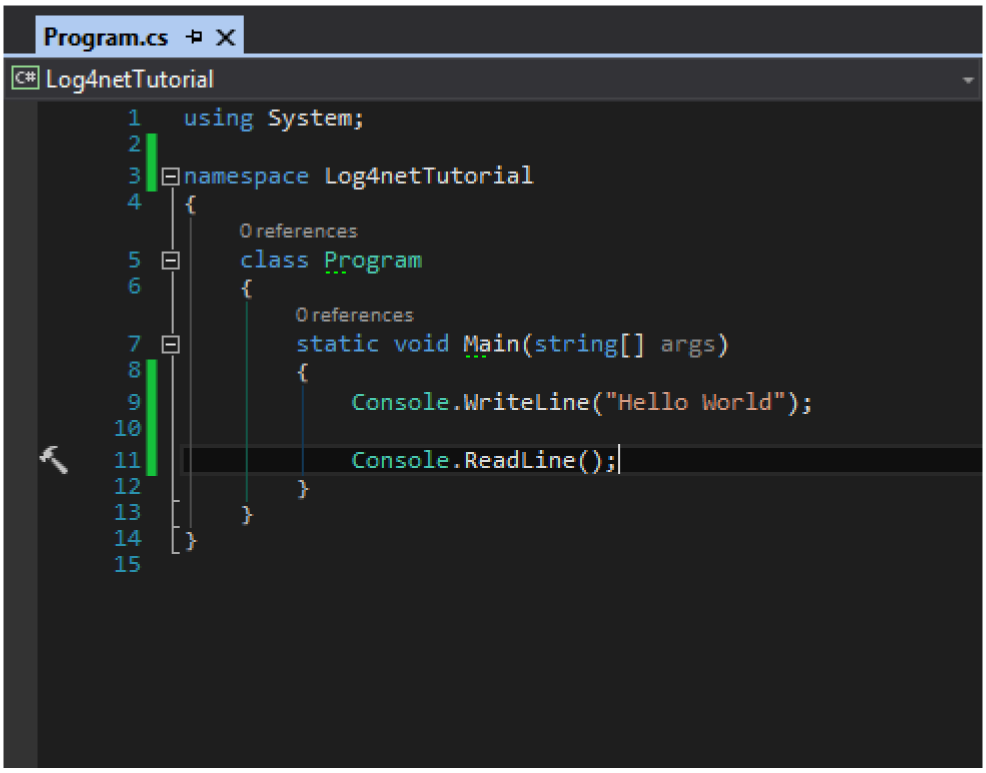
It should also mention any large subjects within log4net, and link out to the related topics. Since the Documentation for log4net is new, you may need to create initial versions of those related topics.

## Examples

### Setting up log4net

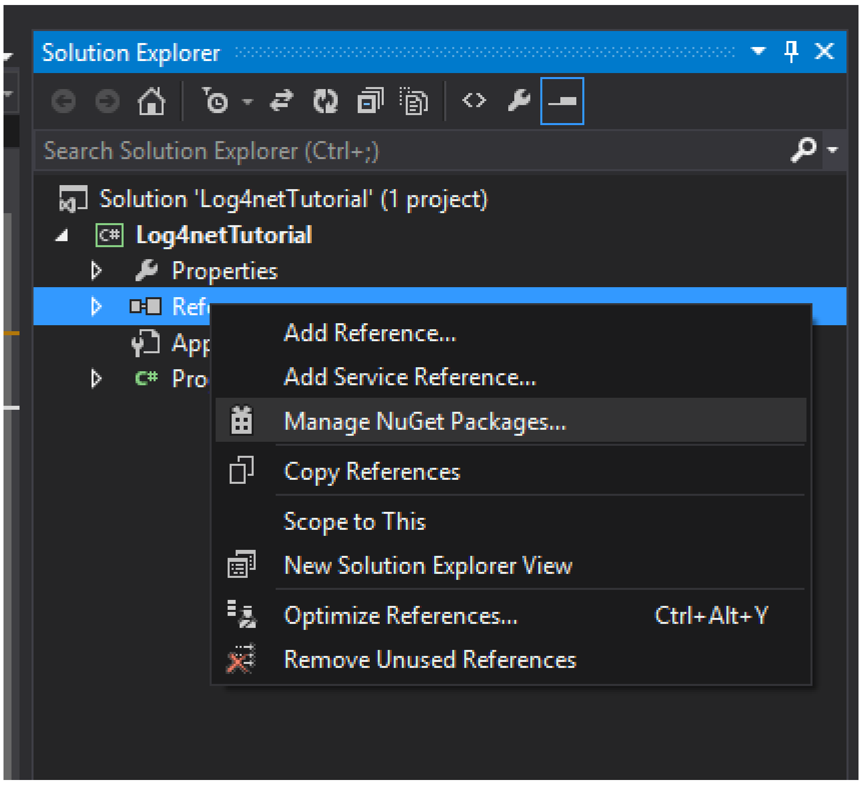
Let us create a simple *Hello World* console application and log something to the console using log4net. Once we have this running, we can scale it out to use it in real development scenarios in the following examples. Let's start small and build it up from there.

First, we need to create a simple Console Project in Visual Studio and `WriteLine("Hello World")` as shown below

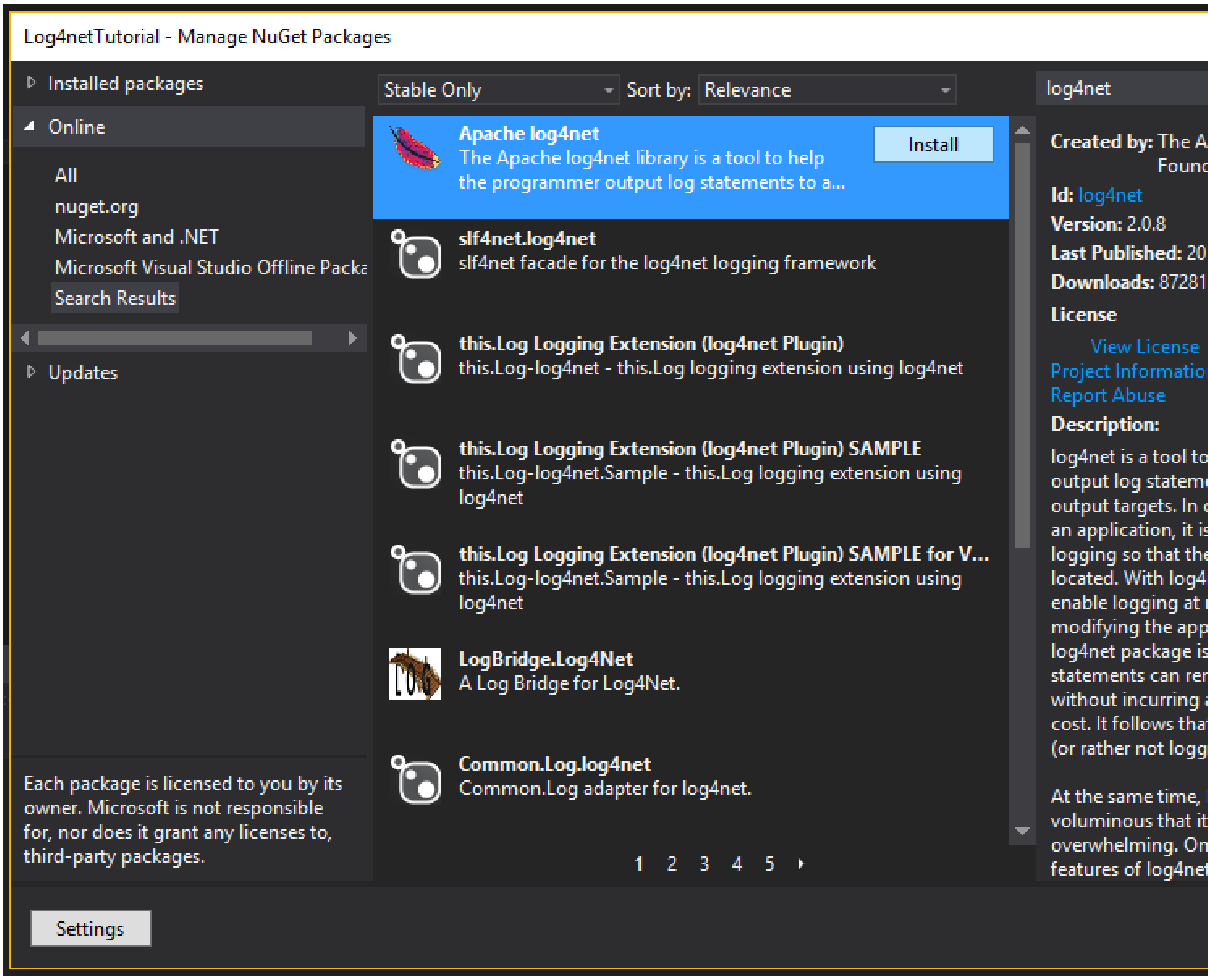


```
using System;

namespace Log4netTutorial
{
    class Program
    {
        static void Main(string[] args)
        {
            Console.WriteLine("Hello World");

            Console.ReadLine();
        }
    }
}
```

Next, we need to add the log4net library to our project. So we need to right click on *References* and select *Manage NuGet Packages*



Then search for **log4net** and install


Log4netTutorial - Manage NuGet Packages

Stable Only | Sort by: Relevance | log4net

**Apache log4net**
The Apache log4net library is a tool to help the programmer output log statements to a...

**slf4net.log4net**
slf4net facade for the log4net logging framework

**this.Log Logging Extension (log4net Plugin)**
this.Log-log4net - this.Log logging extension using log4net

**this.Log Logging Extension (log4net Plugin) SAMPLE**
this.Log-log4net.Sample - this.Log logging extension using log4net

**this.Log Logging Extension (log4net Plugin) SAMPLE for V...**
this.Log-log4net.Sample - this.Log logging extension using log4net

**LogBridge.Log4Net**
A Log Bridge for Log4Net.

**Common.Log.log4net**
Common.Log adapter for log4net.

**Created by:** The A Found
**Id:** log4net
**Version:** 2.0.8
**Last Published:** 20
**Downloads:** 87281
**License**
View License
Project Informatio
Report Abuse
**Description:**
log4net is a tool to output log stateme output targets. In c an application, it is logging so that the located. With log4 enable logging at r modifying the app log4net package is statements can ren without incurring a cost. It follows tha (or rather not logg

At the same time, l voluminous that it overwhelming. On features of log4net

Each package is licensed to you by its owner. Microsoft is not responsible for, nor does it grant any licenses to, third-party packages.

We have now successfully added log4net.dll to our project.

Now we need to configure our *app.config* (or *web.config* if it is a web application). This part is a bit tricky but don't panic. We will run through it step by step and have logging up and running.

Under the *app.config* we first need to create a config section under configuration. We need to provide a name and a type as shown below:

```
<configSections>
  <section name="log4net" type="log4net.Config.Log4NetConfigurationSectionHandler,Log4net"/>
</configSections>
```

What we have done here is we are saying that we are going to create a section in our config file called `log4net` which needs to be looked for. Everything pertaining to logging will be in here. From creating our log file, how to long and what information needs to be logged. So now let's go ahead and create the `log4net` section.

```
<?xml version="1.0" encoding="utf-8" ?>
```

```
<configuration>
  <configSections>
    <section name="log4net" type="log4net.Config.Log4NetConfigurationSectionHandler,Log4net"/>
  </configSections>
  <log4net>

  </log4net>
  <startup>
      <supportedRuntime version="v4.0" sku=".NETFramework,Version=v4.5" />
  </startup>
</configuration>
```

The first thing we are going to add under our log4net section is an appender. An appender is basically a logging tool. It is what we want to log our logs to. There are many appenders available for logging into files, database etc. In this example, we will log to our Console Window so let's create a Console Appender:

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

  </appender>
</log4net>
```

Next, we need to define a layout:

```
<log4net>
  <appender name="ConsoleAppender" type="log4net.Appender.ConsoleAppender">
    <layout type="log4net.Layout.PatternLayout">

    </layout>
  </appender>
</log4net>
```

Under layout, we will define what we want to display on the Console and how we want to display it. For this, we need a conversion pattern. There are many different type of patterns we can use, for this example we will stick to a fairly simple pattern.

```
<log4net>
  <appender name="ConsoleAppender" type="log4net.Appender.ConsoleAppender">
    <layout type="log4net.Layout.PatternLayout">
      <conversionPattern value="%date{ABSOLUTE} [%thread] %level %logger - %message%newline"/>
    </layout>
  </appender>
</log4net>
```

What we are doing above is saying that we need the following information: The absolute timestamp, the running thread which throws the exception and the logging level. There are 7 logging levels provided by log4net.

- OFF - nothing gets logged (cannot be called)
- FATAL
- ERROR
- WARN

- INFO
- DEBUG
- ALL - everything gets logged (cannot be called)

However, out the 7 we can use only 5 (DEBUG, INFO, WARN, ERROR and FATAL). Declaring the logging level to DEBUG means it will log everything, i.e. DEBUG, INFO, WARN, ERROR and FATAL. However, declaring logging level to WARN will log only WARN, ERROR and FATAL. Hope you understand the hierarchy.

The last thing we need under the *app.config* is a root section. The root section houses our top level loggers and the level to log at. It is important to understand that everything inherits from the root, so no appender will log unless it is referenced under the root. With that added, this is how your *app.config* should look like:

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <configSections>
    <section name="log4net" type="log4net.Config.Log4NetConfigurationSectionHandler,Log4net"/>
  </configSections>
  <log4net>
    <appender name="ConsoleAppender" type="log4net.Appender.ConsoleAppender">
      <layout type="log4net.Layout.PatternLayout">
        <conversionPattern value="%date{ABSOLUTE} [%thread] %level %logger -
%message%newline"/>
      </layout>
    </appender>
    <root>
      <level value="DEBUG"/>
      <appender-ref ref="ConsoleAppender" />
    </root>
  </log4net>
  <startup>
      <supportedRuntime version="v4.0" sku=".NETFramework,Version=v4.5" />
  </startup>
</configuration>
```

It is not a bad idea to copy-paste sections of the config file, however, it is important to understand what they are responsible for. Now that we are done configuring our *app.config*, we need to add a little bit of code to our Console Project. First, we need to define a one-time entry that needs to be placed outside of your class.

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

Put it right below my using statements in the Program.cs file. Next we need to create an instance of the logger to use it for logging. This is done once per class.

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

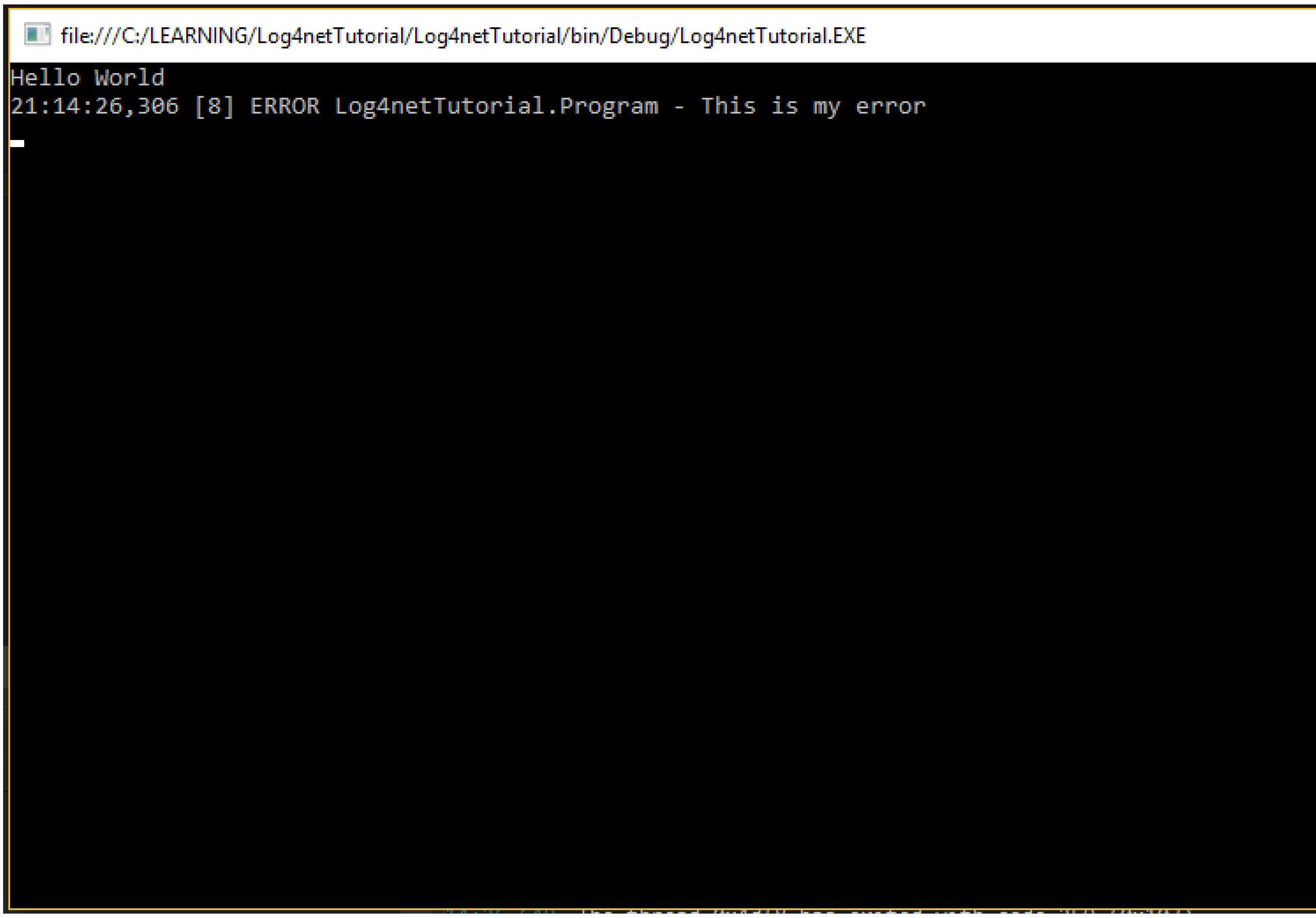
And finally, we need to log something

```
log.Error("This is my error");
```

Your Program.cs file should like something like this:

```
using System;

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

namespace Log4netTutorial
{
    class Program
    {
        private static readonly log4net.ILog log =
log4net.LogManager.GetLogger(System.Reflection.MethodBase.GetCurrentMethod().DeclaringType);

        static void Main(string[] args)
        {
            Console.WriteLine("Hello World");
            log.Error("This is my error");

            Console.ReadLine();
        }
    }
}
```

Go ahead and run the program, you should have your error logged in the console:



file:///C:/LEARNING/Log4netTutorial/Log4netTutorial/bin/Debug/Log4netTutorial.EXE

```
Hello World
21:14:26,306 [8] ERROR Log4netTutorial.Program - This is my error
```

Read Getting started with log4net online: https://riptutorial.com/log4net/topic/9982/getting-started-with-log4net

# Chapter 2: Log4Net Troubleshooting

## Introduction

Log4net is a fail-stop logging system. Fail stop means that it stops logging on an internal exception and by design does not interact with the program flow. Knowing this explains you troubleshooting log4net isn't so easy. If logging fails, your program does not notice. You see a lot of questions about: why is my log4net logging not working? This article explains the basic troubleshooting and to most common solutions.

In most cases the first step is to enable internal debugging for log4net.

## Examples

### Enable internal debugging

There are 2 way of enabling internal debugging in log4net:

- Specify the log4net.Internal.Debug option in the application's config file
- Enable log4net's internal debug programmatically

**Specify the log4net.Internal.Debug option in the application's config file**

This is the preferred method to enable internal debugging, add the log4net.Internal.Debug key to the app.config file of you application.

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
    <appSettings>
        <add key="log4net.Internal.Debug" value="true"/>
    </appSettings>
</configuration>
```

Debug logging will start immediately when the application starts.

**Enable log4net's internal debug programmatically**

A second way is to do this programmatically. Set the log4net.Util.LogLog.InternalDebugging property to true:

```
log4net.Util.LogLog.InternalDebugging = true;
```

**Internal debug log output**

Internal debugging messages are written to the console and to the System.Diagnostics.Trace. When you have to console output you can redirect the System.Console.Out. Or you can redirect the trace message to a file:

```
<configuration>
...

<system.diagnostics>
    <trace autoflush="true">
        <listeners>
            <add
                name="textWriterTraceListener"
                type="System.Diagnostics.TextWriterTraceListener"
                initializeData="C:\tmp\log4net.txt" />
        </listeners>
    </trace>
</system.diagnostics>

...
</configuration>
```

## Buffered appenders

Some of the log4net appenders are buffered appenders. These appenders will only log when a certain amount of messages are logged. Some samples are the SmtpAppender, RemotingAppender or the AdoNetAppender. These appenders have a setting BufferSize:

```
<bufferSize value="100" />
```

This means that the logger will log when there are 100 messages in the buffer. When you want to test the appender you can set the bufferSize to 1.

If you what the buffer to be flushed on an error, you can use an evaluator:

```
<evaluator type="log4net.Core.LevelEvaluator">
     <threshold value="ERROR"/>
</evaluator>
```

If the condition of the evaluator is met, the buffer will be flushed.

## Configure() not called, or called multiple times

If you do not see any logging, you can check if `Configure()` is called in your application. The easiest way is to add it as an attribute to your assembly:

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

Then you do not have to add `log4net.Config.XmlConfigurator.Configure()` anywhere in your configuration. Or you can add `log4net.Config.XmlConfigurator.Configure()` in one of your startup methods. Make sure you only call the configuration once.

## File appender does not write

If you have a file appender, make sure you are writing to a location where the user is allowed to create and update the files. If not the logging will fail. You can check this when you have internal debugging enabled.

Read Log4Net Troubleshooting online: https://riptutorial.com/log4net/topic/10686/log4net-troubleshooting

# Credits

| S. No | Chapters | Contributors |
|---|---|---|
| 1 | Getting started with log4net | Community, FortyTwo |
| 2 | Log4Net Troubleshooting | Peter |