

Reference Documentation

Version 1.2.0

Last Updated August 4, 2007

Copyright © 2004-2006 Mark Pollack, Erich Eichinger, Bruno Baia

Copies of this document may be made for your own use and for distribution to others, provided that you do not charge any fee for such copies and further provided that each copy contains this Copyright Notice, whether distributed in print or electronically.

1. Common Logging	1
1.1. Introduction	1
1.2. Using Common.Logging API	1
1.3. Configuring Logging	2
1.3.1. Declarative Configuration	2
1.3.2. Configuring Logging in your code	3
1.4. Logging Adapters	3
1.4.1. NoOpLoggerFactoryAdapter	3
1.4.2. ConsoleOutLoggerFactoryAdapter	3
1.4.3. TraceLoggerFactoryAdapter	4
1.4.4. Log4NetLoggerFactoryAdapter	4
1.4.5. NLogLoggerFactoryAdapter	6
1.4.6. EntLibLoggingAdapter	7
1.5. Advanced Logging Tasks	8
1.5.1. Implementing a custom FactoryAdapter	8

Chapter 1. Common Logging

1.1. Introduction

There are a variety of logging implementations for .NET currently in use, log4net, Enterprise Library Logging, NLog, to name the most popular. The downside of having different implementation is that they do not share a common interface and therefore impose a particular logging implementation on the users of your library. To solve this dependency problem the Common.Logging library introduces a simple abstraction to allow you to select a specific logging implementation at runtime.

The library is based on work done by the developers of IBatis.NET and it's usage is inspired by log4net. Many thanks to the developers of those projects! The library is available for .NET 1.0, 1.1, and 2.0 with both debug and strongly signed release assemblies.

The base logging library, Common.Logging, provides the base logging interfaces that appear in your code and also include simple console and trace based logger implementations. The libraries are located under bin/net/<framework-version>/debug or release. There is one NLog implementation and two enterprise log4net implementations, one for log4net 1.2.9 and another for log4net 1.2.10. The need for two log4net versions is due to the fact that each is signed with a different strong key making assembly redirection impossible. Future releases will support Enterprise Library Logging.

Note that it is not the intention of this library to be a replacement for the many fine logging libraries that are out there. The API is incredibly minimal and will very likely stay that way. Only use this library if you truly need to support multiple logging APIs.

1.2. Using Common.Logging API

Usage of the Logging API is fairly simple. First you need to obtain a logger from the LogManager and call the appropriate logging method:

```
using Common.Logging;
...
ILog log = LogManager.GetLogger(this.GetType());
log.Debug("hello world");
```

It is also possible to obtain a logger by name

```
ILog log = LogManager.GetLogger("mylogger");
```

A logger instance provides the following methods for logging:

```
public interface ILog
{
    void Trace( object message );
    void Trace( object message, Exception exception );
    void Debug( object message );
    void Debug( object message, Exception exception );
    void Error( object message );
    void Error( object message, Exception exception );
    void Fatal( object message );
    void Fatal( object message, Exception exception );
    void Info( object message );
```

```

void Info( object message, Exception exception );
void Warn( object message );
void Warn( object message, Exception exception );

bool IsTraceEnabled { get; }
bool IsDebugEnabled { get; }
bool IsErrorEnabled { get; }
bool IsFatalEnabled { get; }
bool IsInfoEnabled { get; }
bool IsWarnEnabled { get; }
}

```

Since the ILog interface mimics that of the interface used in log4net, migration from log4net is just a matter of changing the 'using' statement.

You can get a reference to an instance of an ILog using the LogManager class. Its API is shown below:

```

public sealed class LogManager
{
    public static ILog GetLogger( Type type ) ...
    public static ILog GetLogger( string name ) ...

    public static ILoggerFactoryAdapter Adapter ...
}

```

The Adapter property is used by the framework itself.

1.3. Configuring Logging

There are 2 ways of configuring logging in your application - either declaratively or programmatically.

1.3.1. Declarative Configuration

Logging configuration can be done declaratively in your app.config

```

<configuration>
  <configSections>
    <sectionGroup name="common">
      <section name="logging" type="Common.Logging.ConfigurationSectionHandler, Common.Logging" />
    </sectionGroup>
  </configSections>

  <common>
    <logging>
      <factoryAdapter type="Common.Logging.Simple.ConsoleOutLoggerFactoryAdapter, Common.Logging">
        <arg key="level" value="DEBUG" />
        <arg key="showLogName" value="true" />
        <arg key="showDataTime" value="true" />
        <arg key="dateTimeFormat" value="yyyy/MM/dd HH:mm:ss:fff" />
      </factoryAdapter>
    </logging>
  </common>
</configuration>

```



Note

The concrete set of <arg> elements you may specify depends on the FactoryAdapter being used.

Note that if you have installed Common.Logging in the GAC, you will need to specify the fully qualified name of the assembly, i.e. add the Version, Culture, and PublicKeyToken, etc. See the log4net section for an example.

1.3.2. Configuring Logging in your code

You may manually configure logging by setting a `LoggerFactoryAdapter` in your code.

```
// create properties
NameValueCollection properties = new NameValueCollection();
properties["showDateTime"] = "true";

// set Adapter
Common.Logging.LogManager.Adapter = new Common.Logging.Simple.ConsoleOutLoggerFactoryAdapter(properties);
```



Note

The concrete set of properties you may specify depends on the `FactoryAdapter` being used.

1.4. Logging Adapters

There are simple out-of-the-box implementations coming with Common.Logging itself. For connecting to log4net, separate adapters do exist.



Note

Be sure to correctly specify the type of the `FactoryAdapter` in the common logging configuration section and to copy the logging implementation .dlls to your runtime directory. At the moment, if the specified `FactoryAdapter` type is not found or its dependent libraries, the `NoOpLoggerFactoryAdapter` is used by default and you will not see any logging output.

1.4.1. NoOpLoggerFactoryAdapter

This is the default `FactoryAdapter` if logging is not configured. It simply does nothing.

1.4.2. ConsoleOutLoggerFactoryAdapter

`ConsoleOutLoggerFactoryAdapter` uses `Console.Out` for logging output.

Table 1.1. Configuration Properties

Key	Possible Value(s)	Description
level	All Debug Info Warn Error Fatal Off	Defines the global maximum level of logging.
showDateTime	true false	output timestamp?
showLogName	true false	output logger name?

Key	Possible Value(s)	Description
dateTimeFormat	any formatstring accepted by DateTime.ToString()	defines the format to be used for output the timestamp. If no format is specified DateTime.ToString() will be used.

1.4.3. TraceLoggerFactoryAdapter

TraceLoggerFactoryAdapter uses `System.Diagnostics.Trace` for logging output. For viewing it's output you can use any tool that is capable of capturing calls to `Win32 OutputDebugString()` - e.g. the tool "DebugView" from www.sysinternals.com.

Table 1.2. Configuration Properties

Key	Possible Value(s)	Description
level	All Debug Info Warn Error Fatal Off	Defines the global maximum level of logging.
showDateTime	true false	output timestamp?
showLogName	true false	output logger name?
dateTimeFormat	any formatstring accepted by DateTime.ToString()	defines the format to be used for output the timestamp. If no format is specified DateTime.ToString() will be used.

1.4.4. Log4NetLoggerFactoryAdapter

There are two implementations, both configured similarly.

- `Common.Logging.Log4Net`
is linked against log4net 1.2.10.0
- `Common.Logging.Log4Net129`
is linked against log4net 1.2.9.0

The only difference is in the type specified to the factory adapter. Both Adapters accept the following configuration properties:

Table 1.3. Configuration Properties

Key	Possible Value(s)	Description
configType	FILE FILE-WATCH	INLINE will simply call <code>XmlConfigurator.Configure()</code>

Key	Possible Value(s)	Description
	INLINE EXTERNAL	EXTERNAL expects log4net being configured somewhere else in your code and does nothing. FILE, FILE-WATCH: see property "configFile" below.
configFile	<path to your log4net.config file>	if configType is FILE or FILE-WATCH, the value of "configFile" is passed to XmlConfigurator.Configure (FileInfo) / ConfigureAndWatch(FileInfo) method.

The example below will configure log4net 1.2.10.0 using the file `log4net.config` from your application's root directory by calling `XmlConfigurator.ConfigureAndWatch()`:

```
<common>
  <logging>
    <factoryAdapter type="Common.Logging.Log4Net.Log4NetLoggerFactoryAdapter, Common.Logging.Log4Net">
      <arg key="configType" value="FILE-WATCH" />
      <arg key="configFile" value="~/log4net.config" />
    </factoryAdapter>
  </logging>
</common>
```

For log4net 1.2.9, change the assembly name `Common.Logging.Log4Net129`.

Another example that shows the log4net configuration 'inline' with the standard application configuration file is shown below.

```
<configSections>
  <sectionGroup name="common">
    <section name="logging" type="Common.Logging.ConfigurationSectionHandler, Common.Logging" />
  </sectionGroup>
  <section name="log4net" type="log4net.Config.Log4NetConfigurationSectionHandler, log4net"/>
</configSections>
<common>
  <logging>
    <factoryAdapter type="Common.Logging.Log4Net.Log4NetLoggerFactoryAdapter, Common.Logging.Log4Net">
      <arg key="configType" value="INLINE" />
    </factoryAdapter>
  </logging>
</common>
<log4net>
  <appender name="ConsoleAppender" type="log4net.Appender.ConsoleAppender">
    <layout type="log4net.Layout.PatternLayout">
      <conversionPattern value="%date [%thread] %-5level %logger %ndc - %message%newline" />
    </layout>
  </appender>
```

```

<root>
  <level value="DEBUG" />
  <appender-ref ref="ConsoleAppender" />
</root>

<logger name="MyApp.DataAccessLayer">
  <level value="DEBUG" />
</logger>

</log4net>

```

Note that if you are using Common.Logging or any of the adapters from the GAC, you will need to specify the full type name, including version, public key token etc., as shown below

```

<configSections>
  <sectionGroup name="common">
    <section name="logging" type="Common.Logging.ConfigurationSectionHandler, Common.Logging,
      Version=1.0.2.0, Culture=neutral, PublicKeyToken=AF08829B84F0328E"/>
  </sectionGroup>
  <section name="log4net" type="log4net.Config.Log4NetConfigurationSectionHandler, log4net,
    Version=1.2.10.0, Culture=neutral, PublicKeyToken=1B44E1D426115821"/>
</configSections>
<common>
  <logging>
    <factoryAdapter type="Common.Logging.Log4Net.Log4NetLoggerFactoryAdapter, Common.Logging.Log4Net,
      Version=1.0.2.2, Culture=neutral, PublicKeyToken=AF08829B84F0328E">
      <arg key="configType" value="FILE-WATCH"/>
      <arg key="configFile" value="~/log4net.config"/>
    </factoryAdapter>
  </logging>
</common>

```

1.4.5. NLogLoggerFactoryAdapter

There is one implementation.

- Common.Logging.NLog

is linked against NLog 1.0.0.505

Table 1.4. Configuration Properties

Key	Possible Value(s)	Description
configType	INLINE FILE	INLINE FILE: see property "configFile" below.
configFile	<path to your NLog.config file>	if configType is FILE, the value of "configFile" is passed to XmlLoggingConfiguration(string) constructor.

The example below will configure NLog using the file NLog.config from your application's root directory by calling `XmlLoggingConfiguration(string)`:


```

<common>
  <logging>
    <factoryAdapter type="Common.Logging.NLog.NLogLoggerFactoryAdapter, Common.Logging.NLog">
      <arg key="configType" value="FILE" />
      <arg key="configFile" value="~/NLog.config" />
    </factoryAdapter>
  </logging>
</common>

```

Another example that shows the NLog configuration 'inline' with the standard application configuration file is shown below.

```

<configSections>
  <sectionGroup name="common">
    <section name="logging" type="Common.Logging.ConfigurationSectionHandler, Common.Logging" />
  </sectionGroup>

  <section name="nlog" type="NLog.Config.ConfigSectionHandler, NLog"/>
</configSections>

<common>
  <logging>
    <factoryAdapter type="Common.Logging.NLog.NLogLoggerFactoryAdapter, Common.Logging.NLog">
      <arg key="configType" value="INLINE" />
    </factoryAdapter>
  </logging>
</common>

<nlog xmlns="http://www.nlog-project.org/schemas/NLog.xsd" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <targets>
    <target name="console" xsi:type="Console" layout="${date:format=HH\:MM\:ss} ${logger} ${message}" />
  </targets>

  <rules>
    <logger name="*" minlevel="Debug" writeTo="console" />
  </rules>
</nlog>

```

1.4.6. EntLibLoggingAdapter

There is one implementation located in the assembly `Common.Logging.EntLib` and is linked against the Microsoft Enterprise Library v 3.1, aka EntLib 3.1. The .dlls for EntLib can not be redistributed so you will need to download EntLib separately.

There are no configuration options for the adapter. Configuration of EntLib logging is done entirely through `App.config`. The example below shows the basic configuration of the `EntLibLoggingAdapter`

```

<common>
  <logging>
    <factoryAdapter type="Common.Logging.EntLib.EntLibLoggerFactoryAdapter, Common.Logging.EntLib">
    </factoryAdapter>
  </logging>
</common>

```

Future releases may include configuration of the priority and also the format in which information about an exception is logged. The current priority used is `-1`, the default. Note that the following mapping of `Common.Logging.LogLevel` and is used.

Table 1.5. EntLib to Common.Logging log level mapping

Common.Logging.LogLevel	System.Diagnostics. TraceEventType
Trace	Verbose
Debug	Verbose
Error	Error
Fatal	Critical
Info	Information
Warn	Warning

1.5. Advanced Logging Tasks

1.5.1. Implementing a custom FactoryAdapter

If you want to plug in a new, yet unsupported logging library, you need to implement the `Common.Logging.ILoggerFactoryAdapter` interface.

Important: Any implementation *must* provide a public constructor accepting a `NameValueCollection` parameter as shown in the example below:

```
public class MyLoggingFactoryAdapter : ILoggerFactoryAdapter
{
    public MyLoggingFactoryAdapter(NameValueCollection properties)
    {
        // configure according to properties
    }

    public ILog GetLogger(Type type) { ... }

    public ILog GetLogger(string name) { ... }
}
```