Exception Handling Application Block In Microsoft Enterprise Library 6.0

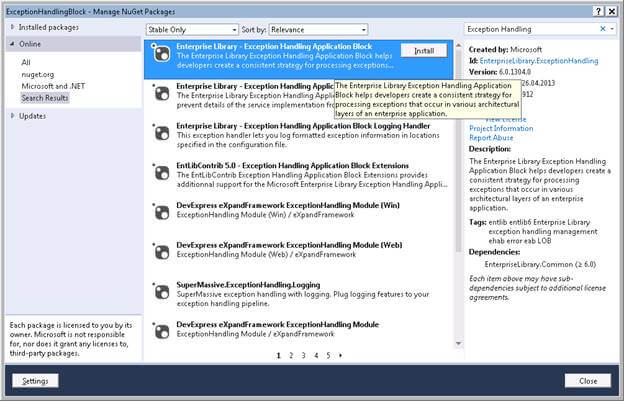
Introduction

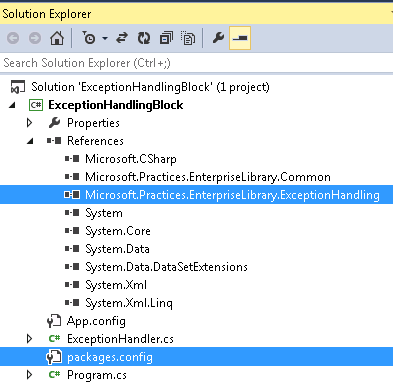
Exception handling is one of the most important tasks in any application. Many applications either do not handle applications or they handle it in an Adhoc manner.

In this section, we will see how we can use the readymade exception handling block so that we need not code, and build error handling routines from scratch.

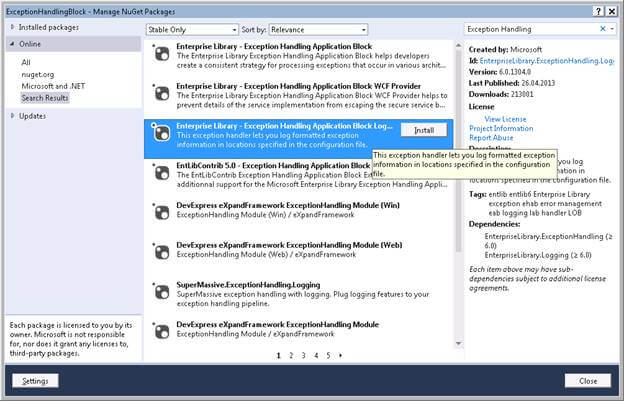
**How**

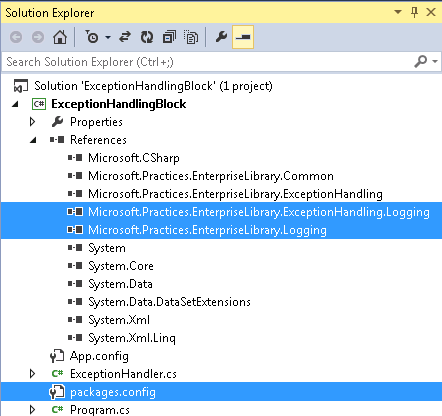
We have to install Enterprise Library – Exception Handling Application Block from NuGet Packages.





Install Enterprise Library – Exception Handling Application Block Logging Handler from NuGet Packages to log the

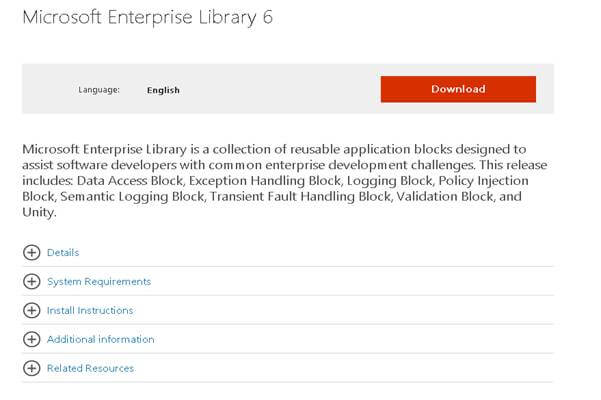




**Microsoft Enterprise Library Configuration Tool**

This tool will be used to configure different blocks for the application. Now, we need to install the Microsoft Enterprise Library from the below URL by following the installation instructions.

[Microsoft Enterprise Library 6](https://www.microsoft.com/en-in/download/details.aspx?id=38789)



**This version of Exception application block is removed by Microsoft.**

**Please download the previous version :**

**https://www.microsoft.com/en-in/download/confirmation.aspx?id=15104**

As in this article, we are just going to discuss the Exception Application Block, we will now configure the exception handling configuration details for our application.

**Please follow the configuration steps below**

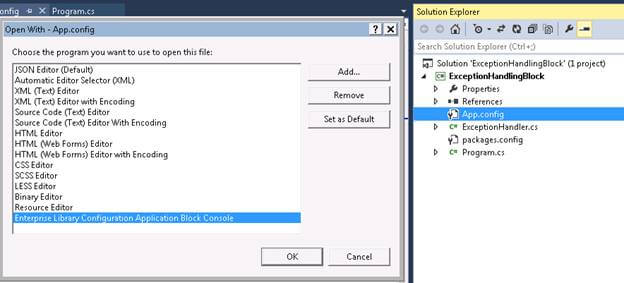
Right-click on the App.config >> Open with----- Enterprise Library Application Block Console. If it is not available in the list, you can add it manually by clicking on the "Add" button.

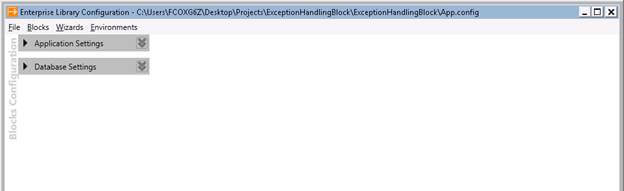
To get the Console option with App.config ---Open with ,

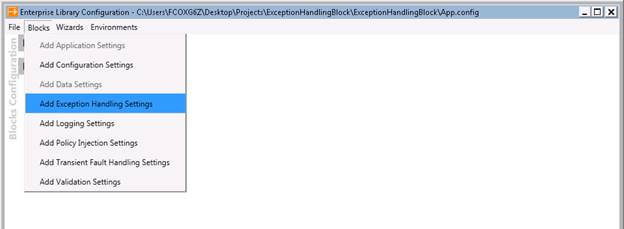
Rt click on App.config =🡺 Add new item =🡺 EnterLibrary.exe

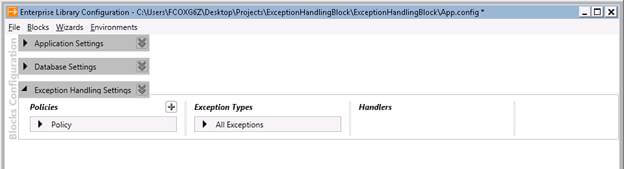
Graphical user interface, application, Word

Description automatically generated

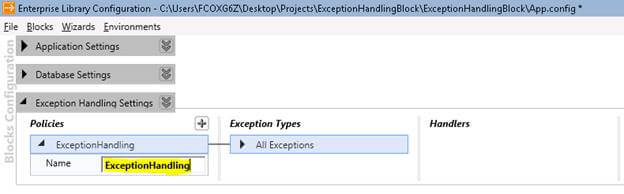


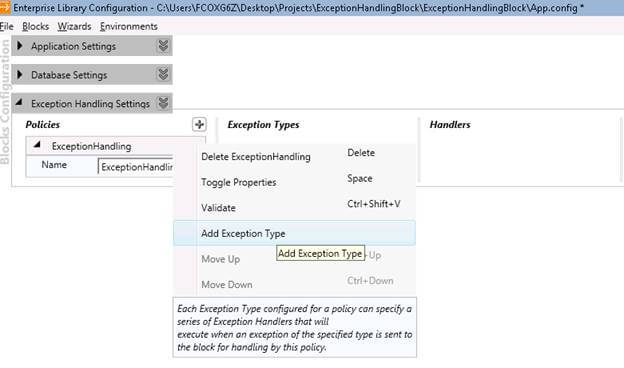


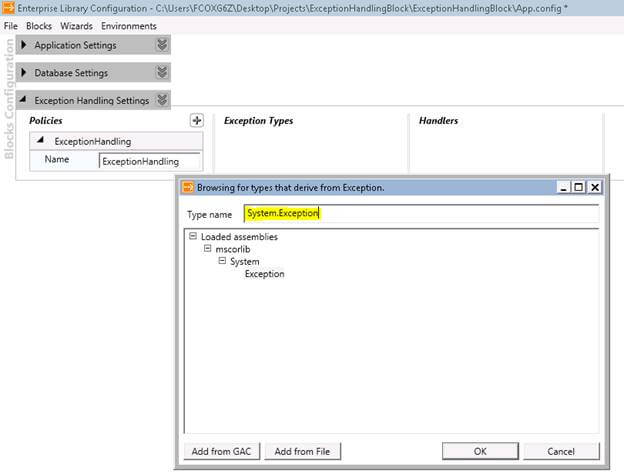












**Using Exception Handlers**

The Exception Handling Application Block is designed to support the typical code contained in catch statements in application components. Instead of repeating this code (such as - logging exception information), throughout the identical catch blocks in an application component, the application block allows developers to encapsulate this logic as reusable exception handlers.

Exception handlers are .NET classes that encapsulate exception handling logic and implement the Exception Handling Application Block interface named IExceptionHandler.

The Exception Handling Application Block includes five exception handlers:

* *Wrap Handler*

 This exception handler wraps one exception around another.

* *Replace Handler*

This exception handler replaces one exception with another.

* *Logging Exception Handler*

 This exception handler formats exception information, such as the message and the stack trace. Then the logging handler gives this information to the Enterprise Library Logging Application Block so that it can be published.

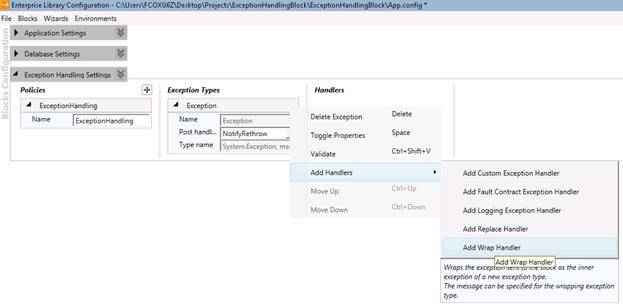
* *Fault Contract Exception Handler*

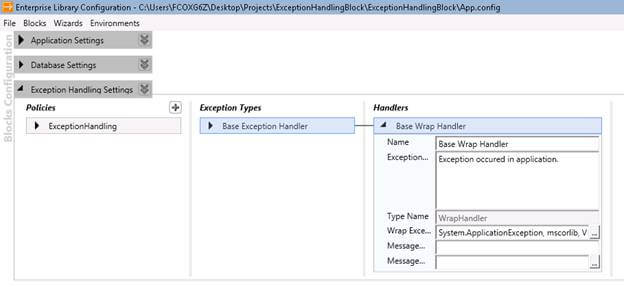
 This exception handler is designed for use at Windows Communication Foundation (WCF) service boundaries and generates a new Fault Contract from the exception.

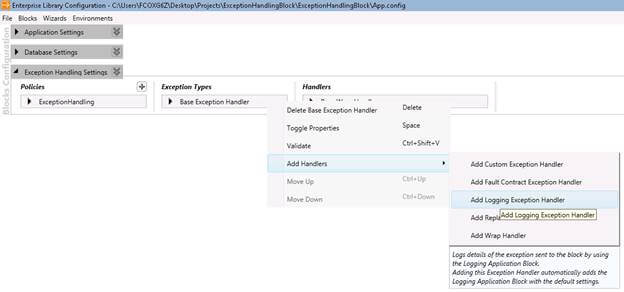
* *Custom Exception Handler*

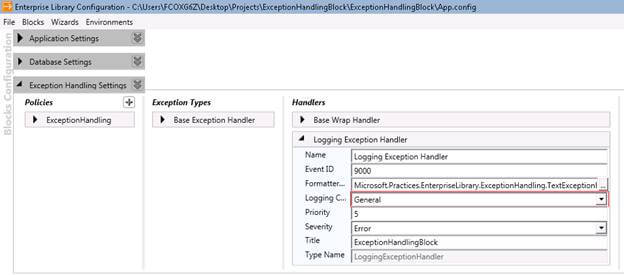
 This exception handler is designed to create our own custom exception handlers.

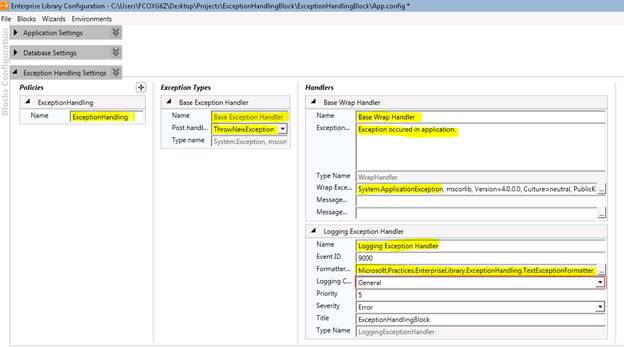
In this article, we are going to cover Wrap, Replace, and Logging Exception Handler.



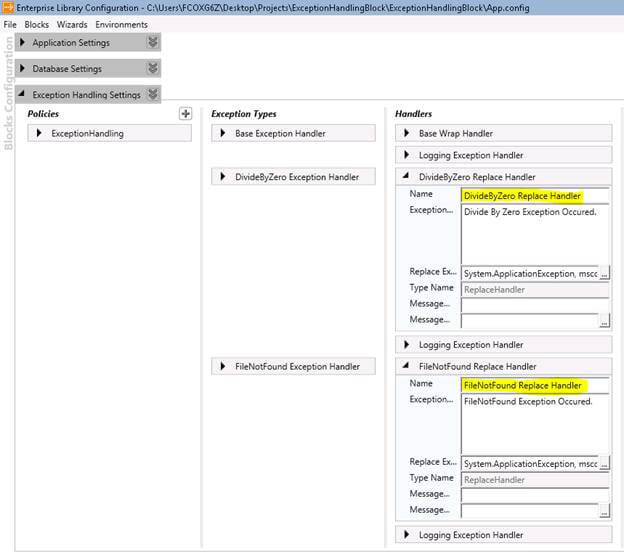








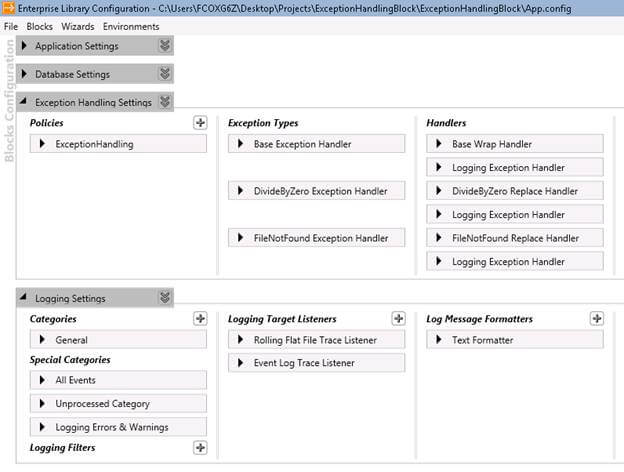
In the same way, we can configure the settings for Replace Handler also. The total configuration will look something like the image shown below:



Now, we are done with our configuration for the Exception Handling Block; and, we are going to configure the Logging Application Block here to log the exceptions that are going to be thrown by our application.

I am not going to explain regarding configuring Logging Application Block here as we have already covered that in my previous article. [Click here](https://www.c-sharpcorner.com/article/logging-block-in-microsoft-enterprise-library-6-0/) to have a look at the Logging Application Block.

After configuring the Exception Handling and Logging Block, the configuration tool will look as below.



All the above configuration will generate the below sections in our App.config.

You can directly add the below configuration blocks to your App.config under <configuration> section if you don’t want to configure through the configuration console.

1. <configSections>
2. <section name="loggingConfiguration" type="Microsoft.Practices.EnterpriseLibrary.Logging.Configuration.LoggingSettings, Microsoft.Practices.EnterpriseLibrary.Logging, Version=6.0.0.0, Culture=neutral, PublicKeyToken=31bf3856ad364e35" requirePermission="true" />
3. <section name="exceptionHandling" type="Microsoft.Practices.EnterpriseLibrary.ExceptionHandling.Configuration.ExceptionHandlingSettings, Microsoft.Practices.EnterpriseLibrary.ExceptionHandling, Version=6.0.0.0, Culture=neutral, PublicKeyToken=31bf3856ad364e35" requirePermission="true" />
4. </configSections>
5. <loggingConfiguration name="" tracingEnabled="false" defaultCategory="General" logWarningsWhenNoCategoriesMatch="false">
6. <listeners>
7. <add name="Rolling Flat File Trace Listener" type="Microsoft.Practices.EnterpriseLibrary.Logging.TraceListeners.RollingFlatFileTraceListener, Microsoft.Practices.EnterpriseLibrary.Logging, Version=6.0.0.0, Culture=neutral, PublicKeyToken=31bf3856ad364e35" listenerDataType="Microsoft.Practices.EnterpriseLibrary.Logging.Configuration.RollingFlatFileTraceListenerData, Microsoft.Practices.EnterpriseLibrary.Logging, Version=6.0.0.0, Culture=neutral, PublicKeyToken=31bf3856ad364e35" fileName="RollingFlatFile.log" footer="----------------------------------" formatter="Text Formatter" header="" rollInterval="Day" traceOutputOptions="DateTime, Timestamp" filter="All" />
8. <add name="Event Log Trace Listener" type="Microsoft.Practices.EnterpriseLibrary.Logging.TraceListeners.FormattedEventLogTraceListener, Microsoft.Practices.EnterpriseLibrary.Logging, Version=6.0.0.0, Culture=neutral, PublicKeyToken=31bf3856ad364e35" listenerDataType="Microsoft.Practices.EnterpriseLibrary.Logging.Configuration.FormattedEventLogTraceListenerData, Microsoft.Practices.EnterpriseLibrary.Logging, Version=6.0.0.0, Culture=neutral, PublicKeyToken=31bf3856ad364e35" source="Application" formatter="Text Formatter" log="Application" machineName="." traceOutputOptions="None" />
9. </listeners>
10. <formatters>
11. <add type="Microsoft.Practices.EnterpriseLibrary.Logging.Formatters.TextFormatter, Microsoft.Practices.EnterpriseLibrary.Logging, Version=6.0.0.0, Culture=neutral, PublicKeyToken=31bf3856ad364e35" template="Timestamp: {timestamp(local)}{newline}
12. Message: {message}{newline}
13. Category: {category}{newline}
14. Priority: {priority}{newline}
15. Severity: {severity}" name="Text Formatter" />
16. </formatters>
17. <categorySources>
18. <add switchValue="All" autoFlush="true" name="General">
19. <listeners>
20. <add name="Rolling Flat File Trace Listener" />
21. <add name="Event Log Trace Listener" />
22. </listeners>
23. </add>
24. </categorySources>
25. <specialSources>
26. <allEvents switchValue="All" name="All Events">
27. <listeners>
28. <add name="Rolling Flat File Trace Listener" />
29. </listeners>
30. </allEvents>
31. <notProcessed switchValue="All" name="Unprocessed Category">
32. <listeners>
33. <add name="Rolling Flat File Trace Listener" />
34. </listeners>
35. </notProcessed>
36. <errors switchValue="All" name="Logging Errors & Warnings">
37. <listeners>
38. <add name="Rolling Flat File Trace Listener" />
39. </listeners>
40. </errors>
41. </specialSources>
42. </loggingConfiguration>
43. <exceptionHandling>
44. <exceptionPolicies>
45. <add name="ExceptionHandling">
46. <exceptionTypes>
47. <add name="Base Exception Handler" type="System.Exception, mscorlib,   Version=4.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089" postHandlingAction="ThrowNewException">
48. <exceptionHandlers>
49. <add name="Base Wrap Handler" type="Microsoft.Practices.EnterpriseLibrary.ExceptionHandling.WrapHandler, Microsoft.Practices.EnterpriseLibrary.ExceptionHandling" exceptionMessage="Exception occurred in application." wrapExceptionType="System.ApplicationException, mscorlib, Version=4.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089" />
50. <add name="Logging Exception Handler" type="Microsoft.Practices.EnterpriseLibrary.ExceptionHandling.Logging.LoggingExceptionHandler, Microsoft.Practices.EnterpriseLibrary.ExceptionHandling.Logging, Version=6.0.0.0, Culture=neutral, PublicKeyToken=31bf3856ad364e35" logCategory="General" eventId="9000" severity="Error" title="ExceptionHandlingBlock" formatterType="Microsoft.Practices.EnterpriseLibrary.ExceptionHandling.TextExceptionFormatter, Microsoft.Practices.EnterpriseLibrary.ExceptionHandling, Version=6.0.0.0, Culture=neutral, PublicKeyToken=31bf3856ad364e35" priority="5" />
51. </exceptionHandlers>
52. </add>
53. <add name="DivideByZero Exception Handler" type="System.DivideByZeroException, mscorlib, Version=4.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089" postHandlingAction="ThrowNewException">
54. <exceptionHandlers>
55. <add name="DivideByZero Replace Handler" type="Microsoft.Practices.EnterpriseLibrary.ExceptionHandling.ReplaceHandler, Microsoft.Practices.EnterpriseLibrary.ExceptionHandling" exceptionMessage="Divide By Zero Exception Occurred." replaceExceptionType="System.ApplicationException, mscorlib, Version=4.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089" />
56. <add name="Logging Exception Handler" type="Microsoft.Practices.EnterpriseLibrary.ExceptionHandling.Logging.LoggingExceptionHandler, Microsoft.Practices.EnterpriseLibrary.ExceptionHandling.Logging, Version=6.0.0.0, Culture=neutral, PublicKeyToken=31bf3856ad364e35" logCategory="General" eventId="9000" severity="Error" title="ExceptionHandlingBlock" formatterType="Microsoft.Practices.EnterpriseLibrary.ExceptionHandling.TextExceptionFormatter, Microsoft.Practices.EnterpriseLibrary.ExceptionHandling, Version=6.0.0.0, Culture=neutral, PublicKeyToken=31bf3856ad364e35" priority="5" />
57. </exceptionHandlers>
58. </add>
59. <add name="FileNotFound Exception Handler" type="System.IO.FileNotFoundException, mscorlib, Version=4.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089" postHandlingAction="ThrowNewException">
60. <exceptionHandlers>
61. <add name="FileNotFound Replace Handler" type="Microsoft.Practices.EnterpriseLibrary.ExceptionHandling.ReplaceHandler, Microsoft.Practices.EnterpriseLibrary.ExceptionHandling" exceptionMessage="FileNotFound Exception Occurred." replaceExceptionType="System.ApplicationException, mscorlib, Version=4.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089" />
62. <add name="Logging Exception Handler" type="Microsoft.Practices.EnterpriseLibrary.ExceptionHandling.Logging.LoggingExceptionHandler, Microsoft.Practices.EnterpriseLibrary.ExceptionHandling.Logging, Version=6.0.0.0, Culture=neutral, PublicKeyToken=31bf3856ad364e35" logCategory="General" eventId="9000" severity="Error" title="ExceptionHandlingBlock" formatterType="Microsoft.Practices.EnterpriseLibrary.ExceptionHandling.TextExceptionFormatter, Microsoft.Practices.EnterpriseLibrary.ExceptionHandling, Version=6.0.0.0, Culture=neutral, PublicKeyToken=31bf3856ad364e35" priority="5" />
63. </exceptionHandlers>
64. </add>
65. </exceptionTypes>
66. </add>
67. </exceptionPolicies>
68. </exceptionHandling>

Now, the configuration part is done for the Exception Handling block, which can be used throughout the application level.

Let’s build a sample console application to experience the functionality.

**ExceptionHandler Class**

using Microsoft.Practices.EnterpriseLibrary.ExceptionHandling;

using Microsoft.Practices.EnterpriseLibrary.Logging;

using System;

namespace ExceptionHandlingBlock

{

/// <summary>

/// Purpose for this type is to ease the handling of the exception handling.

/// </summary>

**public** **static** **class** ExceptionHandler

{

#region Static Variables

**private** **static** LogWriter logWriter;

**private** **static** readonly ExceptionPolicyFactory \_exceptionPolicyFactory;

**private** **static** readonly ExceptionManager \_exceptionManager;

#endregion

#region Constructor

**static** ExceptionHandler()

{

logWriter = **new** LogWriterFactory().Create();

Logger.SetLogWriter(logWriter, **false**);

\_exceptionPolicyFactory = **new** ExceptionPolicyFactory();

\_exceptionManager = \_exceptionPolicyFactory.CreateManager();

}

#endregion

#region Static Methods

/// <summary>

/// Performs the handling of an application towards the configured policy.

/// <param name="ex">Exception to handle.</param>

**public** **static** **void** HandleException(Exception ex)

{

Exception exceptionToThrow = **null**;

//The HandleException method will return true if the exception should be (re-)thrown.

**if** (\_exceptionManager.HandleException(ex, "ExceptionHandling", out exceptionToThrow))

{

**if** (exceptionToThrow == **null**)

{

Console.WriteLine(ex.Message);

Console.ReadKey();

}

**else**

{

Console.WriteLine(exceptionToThrow.Message);

Console.ReadKey();

}

}

}

#endregion

}

}

**Executer Class**

using System;

using System.IO;

namespace ExceptionHandlingBlock

{

**class** Executer

    {

**static** **void** Main(string[] args)

        {

            BaseException();

            //DivideByZeroException();

            //FileNotFoundException();

        }

**public** **static** **void** BaseException()

        {

**try**

            {

                string x = **null**;

**int** a = 10;

                Console.WriteLine(**int**.Parse(x) + a);

            }

**catch** (Exception ex)

            {

                ExceptionHandler.HandleException(ex);

            }

        }

**public** **static** **void** DivideByZeroException()

        {

**try**

            {

**int** x = 5;

**int** y = 0;

**int** z = x / y;

            }

**catch** (DivideByZeroException ex)

            {

                ExceptionHandler.HandleException(ex);

            }

        }

**public** **static** **void** FileNotFoundException()

        {

**try**

            {

                // Read in non-existent file.

                using (StreamReader reader = **new** StreamReader("TextFile1.txt"))

                {

                    reader.ReadToEnd();

                }

            }

**catch** (FileNotFoundException ex)

            {

                ExceptionHandler.HandleException(ex);

            }

        }

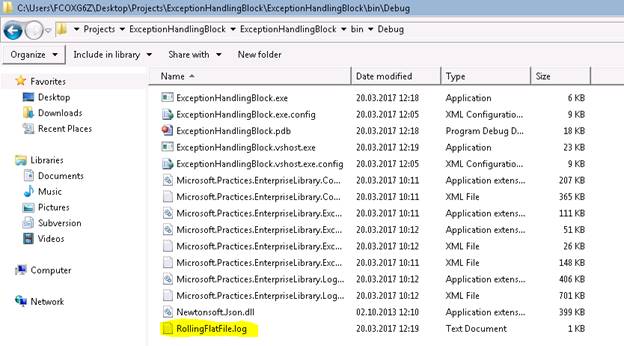
    }

}

Let me explain a bit about the above code blocks.

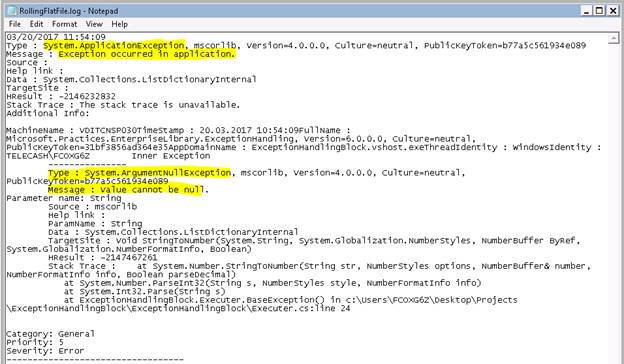
* The ExceptionHandler class will handle the exceptions for all types of exceptions as we have defined here DivideByZeroException, FileNotFoundException, and the base Exception class will deal with the remaining exceptions.
* The ExceptionHandler class will also log the messages for each exception occurred in the application.
* The Executer class contains three methods to test the various exception handlers that we have configured in the application.

Once you execute the application, it will create the Log file in the debug folder as no specific path is mentioned for the file in the configuration and log the message in Windows Event Log.

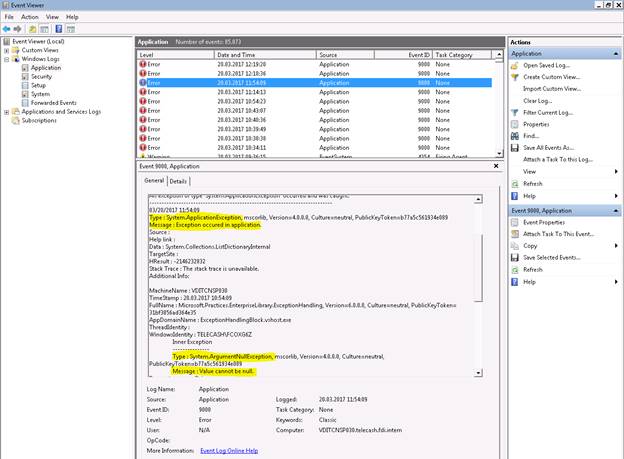


**Executing the BaseException Method**

* As we know, it will throw an ArgumentNullException, but as we are not handling this in our configuration, it will come under Base Exception Handler which uses WrapHandler.
* We can see in the below log message it wrapped the exception with our custom message in the configuration as “Exception occurred in application”



**Windows Event Log**

****

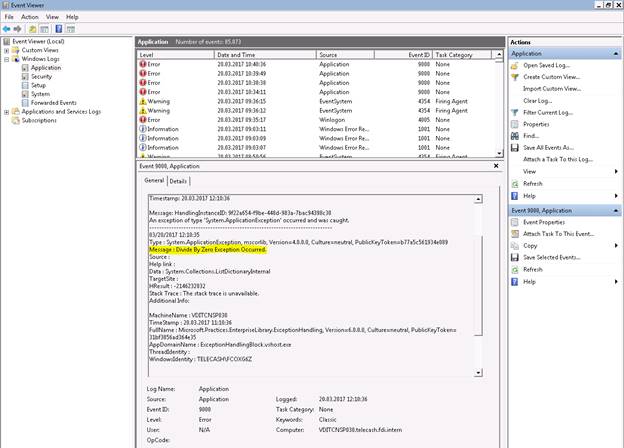
**Executing the DivideByZeroException Method**

* As we know, it will throw a DivideByZeroException and we are handling it in our configuration, it will come under DivideByZero Exception Handler, which uses ReplaceHandler.

* We can see in the below log message it replaced the exception with our custom message in the configuration as “Divide By Zero Exception Occurred”, whereas the actual message should have been “*Attempted to divide by zero*”.



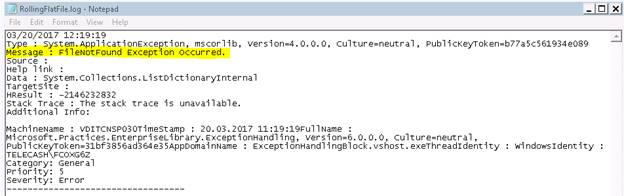
**Windows Event Log**

****

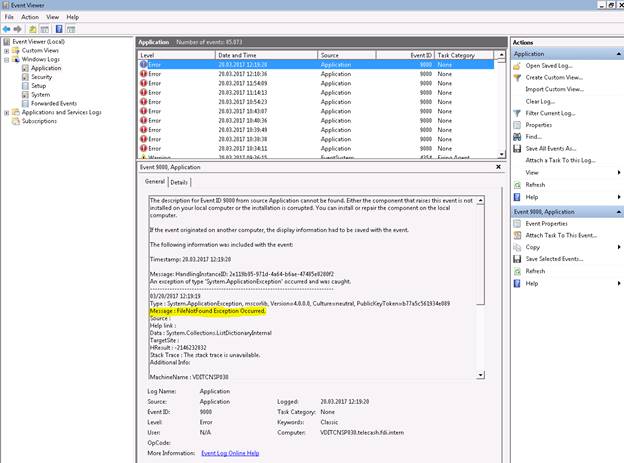
**Executing the FileNotFoundException Method**

* As we know, it will throw a FileNotFoundException and we are handling it in our configuration, it will come under FileNotFound Exception Handler, which uses ReplaceHandler.

* We can see in the below log message it replaced the exception with our custom message in the configuration as “FileNotFound Exception Occurred”, whereas the actual message should have been “*Could not find file 'C:\Users\FCOXG6Z\Desktop\Projects\ExceptionHandlingBlock\ExceptionHandlingBlock\bin\Debug\TextFile1.txt'.*”



**Windows Event Log**

****

Resources

* [Developer's Guide to Microsoft Enterprise Library 6](https://www.microsoft.com/en-us/download/details.aspx?id=41145)
* [Microsoft Enterprise Library 6](https://www.microsoft.com/en-in/download/details.aspx?id=38789)
* [MSDN](https://msdn.microsoft.com/en-us/library/dn169621.aspx)

Conclusion

The best part of the Exception Handling Application block is that you can change policies and handlers on the fly without compiling the code. You can also change the error logging source from event viewer to file or email. I have attached the sample application. Just download it and experience the Exception Handling functionality in Microsoft Enterprise Library 6.0.

I hope you have enjoyed this article and I am sure if you use this block properly, you can have a very stable, efficient, and flexible error handling framework.

In the next article, we will cover the Data Access Application Block in Microsoft Enterprise Library 6.0.

Happy Coding!!!

# Logging Application Block In Microsoft Enterprise Library 6.0

## Introduction

The Microsoft Enterprise Library is a collection of reusable software components (application blocks) designed to assist software developers with common enterprise development cross-cutting concerns, such as logging, validation, data access, exception handling, and many others.

Enterprise Library comes with an easy configuration tool that makes it easier to plug required application blocks into your application. Most developers, though, plug in these application blocks programmatically since this approach is also easy.

## What are the application blocks?

The definition we use is “pluggable and reusable software compo­nents designed to assist developers with common enterprise development challenges.” Application blocks help address the kinds of problems developers commonly face from one line-of-business project to the next.

Their design encapsulates the Microsoft recommended practices for Microsoft .NET Framework-based applications, and developers can add them to .NET based applications so as to configure them quickly and easily.

Diagram

Description automatically generated

In this article, we are going to explore the Logging block of the Microsoft Enterprise Library.

**How**

We have to install Enterprise Library – Logging Application Block from NuGet Packages.

Graphical user interface, text, application, email

Description automatically generated

Graphical user interface, text, application

Description automatically generated

Graphical user interface, text, application

Description automatically generated

Once it is installed successfully, the references and packages will be added to the application.

Graphical user interface, text, application, email

Description automatically generated

## Microsoft Enterprise Library Configuration Tool

This tool will be used to configure different blocks for the application. Now, we need to install the Microsoft Enterprise Library from the below URL by following the installation instructions.

[Microsoft Enterprise Library 6](https://www.microsoft.com/en-in/download/details.aspx?id=38789)

*Graphical user interface, text, application

Description automatically generated*

As in this article, we are just going to discuss the Logging Application Block, we will now configure the logging configure details for our application.

**Please follow the configuration steps below:**

Right click on the App.config >> Open with Enterprise Library Application Block Console.

If it is not available in the list, you can add it manually by clicking on the Add button.

Graphical user interface, application

Description automatically generated

Graphical user interface, application

Description automatically generated

Graphical user interface, application, Word

Description automatically generated

Graphical user interface, application

Description automatically generated

We can log the messages in different file types. In this article, we are going to log in a simple text file (Rolling Flat File Listener) and Windows Event Log (Event Log Listener).

Graphical user interface

Description automatically generated

**Basic Field Information**

* File Name - Log File name and path of the file.
* Formatter Name - Formatter of the log.
* Template - Template of the log; we can set as per our requirement.
* Roll Interval - Interval of the Log; as we have selected Day, it will create log files on daily basis.

Graphical user interface, text, application

Description automatically generated

Graphical user interface, application

Description automatically generated

Now, we are done with our configuration. All the above configuration will generate the below sections in our App.config.

You can directly add the below configuration blocks to your App.config under <configuration> section if you don’t want to configure through the console.

1. <configSections>
2. <section name="loggingConfiguration" type="Microsoft.Practices.EnterpriseLibrary.Logging.Configuration.LoggingSettings, Microsoft.Practices.EnterpriseLibrary.Logging, Version=6.0.0.0, Culture=neutral, PublicKeyToken=31bf3856ad364e35" requirePermission="true" />
3. </configSections>
4. <loggingConfiguration name="" tracingEnabled="false" defaultCategory="General" logWarningsWhenNoCategoriesMatch="false">
5. <listeners>
6. <add name="Rolling Flat File Trace Listener" type="Microsoft.Practices.EnterpriseLibrary.Logging.TraceListeners.RollingFlatFileTraceListener, Microsoft.Practices.EnterpriseLibrary.Logging, Version=6.0.0.0, Culture=neutral, PublicKeyToken=31bf3856ad364e35" listenerDataType="Microsoft.Practices.EnterpriseLibrary.Logging.Configuration.RollingFlatFileTraceListenerData, Microsoft.Practices.EnterpriseLibrary.Logging, Version=6.0.0.0, Culture=neutral, PublicKeyToken=31bf3856ad364e35" fileName="RollingFlatFile.log" footer="----------------------------------" formatter="Text Formatter" header="" rollInterval="Day" traceOutputOptions="DateTime, Timestamp" filter="All" />
7. <add name="Event Log Trace Listener" type="Microsoft.Practices.EnterpriseLibrary.Logging.TraceListeners.FormattedEventLogTraceListener, Microsoft.Practices.EnterpriseLibrary.Logging, Version=6.0.0.0, Culture=neutral, PublicKeyToken=31bf3856ad364e35" listenerDataType="Microsoft.Practices.EnterpriseLibrary.Logging.Configuration.FormattedEventLogTraceListenerData, Microsoft.Practices.EnterpriseLibrary.Logging, Version=6.0.0.0, Culture=neutral, PublicKeyToken=31bf3856ad364e35" source="Application" formatter="Text Formatter" log="Application" machineName="." traceOutputOptions="None" />
8. </listeners>
9. <formatters>
10. <add type="Microsoft.Practices.EnterpriseLibrary.Logging.Formatters.TextFormatter, Microsoft.Practices.EnterpriseLibrary.Logging, Version=6.0.0.0, Culture=neutral, PublicKeyToken=31bf3856ad364e35" template="Timestamp: {timestamp(local)}{newline}
11. Message: {message}{newline}
12. Category: {category}{newline}
13. Priority: {priority}{newline}
14. Severity: {severity}" name="Text Formatter" />
15. </formatters>
16. <categorySources>
17. <add switchValue="All" autoFlush="true" name="General">
18. <listeners>
19. <add name="Rolling Flat File Trace Listener" />
20. <add name="Event Log Trace Listener" />
21. </listeners>
22. </add>
23. </categorySources>
24. <specialSources>
25. <allEvents switchValue="All" name="All Events">
26. <listeners>
27. <add name="Rolling Flat File Trace Listener" />
28. </listeners>
29. </allEvents>
30. <notProcessed switchValue="All" name="Unprocessed Category">
31. <listeners>
32. <add name="Rolling Flat File Trace Listener" />
33. </listeners>
34. </notProcessed>
35. <errors switchValue="All" name="Logging Errors & Warnings">
36. <listeners>
37. <add name="Rolling Flat File Trace Listener" />
38. </listeners>
39. </errors>
40. </specialSources>
41. </loggingConfiguration>

Now, the configuration part is done for the Logging block which can be used throughout the application level.

Let’s build a sample Console application to log some messages.

### LoggerBlock Class

**using** Microsoft.Practices.EnterpriseLibrary.Logging;

**namespace** LoggingApplicationBlock {

**public** **class** LoggerBlock {

**protected** LogWriter logWriter;

**public** LoggerBlock() {

            InitLogging();

        }

**private** **void** InitLogging() {

            logWriter = **new** LogWriterFactory().Create();

            Logger.SetLogWriter(logWriter, **false**);

        }

**public** LogWriter LogWriter {

**get** {

**return** logWriter;

            }

        }

    }

} 

### Executer Class

**using** System;

**using** System.Collections.Generic;

**using** System.Diagnostics;

**using** System.IO;

**using** System.Text;

**namespace** LoggingApplicationBlock {

**class** Executer {

        LoggerBlock loggerBlock = **new** LoggerBlock();

**public** **static** **void** Main() {

**new** Executer().ReadFile();

        }

**public** **void** ReadFile() {

            WriteTraceLog("Application Started!!!");

**string**[] lines;

            var list = **new** List < **string** > ();

            var fileStream = **new** FileStream(@ "sample.txt", FileMode.Open, FileAccess.Read);

**using**(var streamReader = **new** StreamReader(fileStream, Encoding.UTF8)) {

**string** line;

**while** ((line = streamReader.ReadLine()) != **null**) {

                    WriteTraceLog(line);

                    list.Add(line);

                }

            }

            lines = list.ToArray();

            WriteTraceLog("Application Stopped!!!");

        }

**public** **void** WriteTraceLog(String message) {

            loggerBlock.LogWriter.Write(message, "General", 5, 2000, TraceEventType.Information);

        }

    }

} 

Once you execute the application, it will create the Log file in the debug folder as I have not mentioned any specific path for the file in the configuration.

Graphical user interface, text, application, email

Description automatically generated

You can see here, the messages are logged as per the template format.

Also, you can see that the same messages have been logged in the Windows event log too.

Graphical user interface, table

Description automatically generated

**Resources**

* [Developer's Guide to Microsoft Enterprise Library 6](https://www.microsoft.com/en-us/download/details.aspx?id=41145)
* [Microsoft Enterprise Library 6](https://www.microsoft.com/en-in/download/details.aspx?id=38789)
* [MSDN](https://msdn.microsoft.com/en-us/library/dn169621.aspx)

## Conclusion

The best part of the Logging Application Block is that you can change log template format, roll interval, and log file path on the fly without compiling the code. You can also change the error logging source from the event viewer to file or email. I have attached the sample application. Just download it and experience the Logging Application Block functionality in Microsoft Enterprise Library 6.0.

I hope you have enjoyed this article and I am sure if you use this block properly, you can have a very stable, efficient, and flexible logging framework.

In the next article, we will cover Exception Handling Application Block in Microsoft Enterprise Library 6.0.

Happy Coding!!!

=================================================================

It seems to me that you are not referencing the right version of one of the following dll.

Microsoft.Practices.EnterpriseLibrary.Common.dll

Microsoft.Practices.EnterpriseLibrary.Logging.dll

Microsoft.Practices.ServiceLocation.dll