Transient Fault Handling Application Block Hands-On Lab for Enterprise Library

patterns & practices

proven practices for predictable results

Introduction

The Transient Fault Handling Application Block can apply retry policies to operations that your application performs against services that may exhibit transient faults. This makes it easier to implement consistent retry behavior for any transient faults that may affect your application.

In this hands-on lab, you will learn how to use the Transient Fault Handling Application Block to apply a retry policy in your application both by using code and through configuration settings.

This hands-on lab includes the following labs:

- Lab 1: Using the Transient Fault Handling Application Block
- Lab 2: Using the Transient Fault Handling Application Block Asynchronously

The estimated completion for these labs is **30 minutes**.

Lab Setup

Before you start this lab, you should download the Config Console v6. There are no additional lab specific setup instructions for this lab.

Authors

These Hands-On Labs were produced by the following individuals:

- Program Management: Grigori Melnik (Microsoft Corporation)
- Development/Testing: Mariano Grande (Digit Factory), Fernando Simonazzi (Clarius Consulting), Julian Dominguez (Microsoft Corporation), and Jérémi Bourgault (nVentive Inc.)
- Documentation: Mariano Grande (Digit Factory), Fernando Simonazzi (Clarius Consulting), Dominic Betts (Content Master Ltd), Nelly Delgado and RoAnn Corbisier (Microsoft Corporation)

Lab 1: Using the Transient Fault Handling Application Block

Using Code to Control the Transient Fault Handling Application Block

By definition, transient faults only occur intermittently so to simplify the lab exercises, you will use a local database running in Microsoft SQL Server® software to simulate transient faults. This makes it easier for you to observe the behavior of the Transient Fault Handling Application Block.
Before you add the Transient Fault Handling Application Block to the application, you will observe its behavior and examine the existing code.

To create the database and add the stored procedure that simulates a transient fault and to examine the application:

1. In Visual Studio, open the solution called TransientFaultHandlingHOL.sln in the **Lab01\begin** folder.
2. In Solution Explorer, in the **Scripts** folder, double-click the **CreateDatabase.sql** file to open it in the editor. This script creates the products sample database that will be used in the lab.
3. Right-click in the editor window, point to **Connection** and then click **Connect**. In the **Connect to Database Engine** dialog, select your server (typically **(LocalDB)\v11.0**) and then click **Connect**.
4. Right-click in the editor window, and then click **Execute SQL** to create the products sample database.
5. Also, under the **Scripts** folder, there is another file called **CreateDatabaseObjects.sql.** This script creates the products table, populates the table with sample data (only ten records) and then creates a stored procedure called GetProductDetails. The stored procedure will raise a transient error each time it is called by a client instance until the number of times that the client has called the stored procedure in a single session equals the value of the **@NumberOfTriesBeforeSucceeding** parameter.
6. Connect the script to the database engine (see step 3) and execute it so the objects will be created (see step 4).
7. In Solution Explorer, right-click **Main.cs**, and then click **View Code**. The **ExecuteQueryButton_Click** method contains the code to invoke the **GetProductDetails** stored procedure and display the results in the UI.
8. On the **Debug** menu, click **Start Debugging** to run the application.
9. If you click the **Execute query** button while the **Successful attempt** box contains the number **1**, the application displays a result in the UI.

10. If you click the **Execute query** button while the **Successful attempt** box contains a number greater than **1**, the application displays an error message in the UI that reports a transient error.

You can now add the Transient Fault Handling Application Block binaries to the project by using the Transient Fault Handling Application Block NuGet Package.

To add the Transient Fault Handling Application Block to the application

1. In Solution Explorer, right-click **References**, and then click **Manage NuGet Packages**.
2. In the NuGet Package Manager, click **Online**, then select NuGet official package source. In the **Search Online** box type **Topaz** to find the NuGet package that contains the Transient Fault Handling Application Block binaries. Click the **Install** button to add **Enterprise Library – Transient Fault Handling Application Block** to your project. Review the licenses and then click **I Accept**. Click **Close** to close the NuGet Package Manager. Notice how NuGet has added all the required references for the Transient Fault Handling Application Block to your project.

Because you are using SQL Server and not the SQL Database in this lab, you must create a custom Transient Fault Handling Application Block detection strategy to the application. For more information about custom detection strategies, see Implementing a Custom Detection Strategy.

> If you were using SQL Database, you would use the built-in detection strategy for SQL Database instead of adding a custom strategy.

To add the custom detection strategy to the application and implement the retry behavior

1. In Solution Explorer, right-click the **TransientFaultHAndlingHOL** project, point to **Add**, and then click **Existing Item**. In the folder **\Lab01\begin\TransientFaultHandlingHOL** select the **HolSqlTransientErrorDetectionStrategy.cs** file, and then click **Add**.
2. In Solution Explorer, double-click the **HolSqlTransientErrorDetectionStrategy.cs** file to open it in the code editor. This simple custom error detection strategy marks any SQL errors with an error number 50000 as transient.
3. In Solution Explorer, right-click **Main.cs**, and then click **View code**. Add the **using** clause highlighted below:

```
using System;
using System.Configuration;
using System.Data;
using System.Data.SqlClient;
using System.Globalization;
using System.Windows.Forms;
using Microsoft.Practices.EnterpriseLibrary.TransientFaultHandling;
```

4. Modify the code in the **ExecuteQueryButton_Click** method as shown highlighted below to create a retry policy and use the **ExecuteAction** method to invoke the stored procedure. The

retry policy specifies that the Transient Fault Handling Application Block will make two retry attempts. After the first retry, the application block will wait for five seconds before it tries – again.

```
var policy = new RetryPolicy<HolSqlTransientErrorDetectionStrategy>(2,
TimeSpan.FromSeconds(5));

// Execute the command

policy.ExecuteAction(
() =>
{
          using (var reader = command.ExecuteReader())
          {
                // For each record
                while (reader.Read())
                {
                    // Log the result
                    this.Log(string.Format("Product name : {0} ",
reader["ProductName"]));
                }
          }
});
```

5. On the **File** menu, click **Save All**.
6. On the **Debug** menu, click **Start Debugging** to run the application.
7. If you click the **Execute Query** button while the **Successful attempt** box contains the number **1**, the application displays a result in the UI.
8. If you click the **Execute Query** button while the **Successful attempt** box contains the number **2**, the application displays a result in the UI. The first call to the stored procedure fails, but the Transient Fault Handling Application Block immediately makes a second attempt (first retry) that succeeds.
9. If you click the **Execute Query** button while the **Successful attempt** box contains the number **3**, then after a pause the application displays a result in the UI. The first call to the stored procedure fails, the Transient Fault Handling Application Block immediately makes a second attempt (first retry) that also fails, the application block then waits for five seconds before it makes a third attempt (second retry) that succeeds.
10. If you click the **Execute Query** button while the **Successful attempt** box contains the number **4**, then after a pause the application displays an error message in the UI. The first call to the stored procedure fails, the Transient Fault Handling Application Block immediately makes a second attempt (first retry) that also fails, the block waits for five seconds before it makes a third attempt (second retry) that also fails. After two unsuccessful retries the block returns an error.
11. Stop the application.

12. In the code editor, make the following highlighted changes to the **ExecuteQueryButton_Click** method to add an event handler that executes whenever a retry occurs. In this lab, the event handler writes a log message.

```
var policy = new RetryPolicy<HolSqlTransientErrorDetectionStrategy>(2,
TimeSpan.FromSeconds(5));

   policy.Retrying += (s, a) =>
     this.Log(string.Format(CultureInfo.CurrentCulture,
       "Retry attempt {1} delay {2}: {0}",
       a.LastException.Message,
       a.CurrentRetryCount,
       a.Delay
     )
   );
```

13. Repeat steps 6 to 11 and observe the results.

Using Configuration to Control the Transient Fault

Handling Application Block

In the previous exercise, you added code to the application to define the retry policy. In this exercise, you will learn how you can use declarative configuration to define retry policies in your application. In order to use the transient fault handling application block from the config console v6 you will need add the NuGet Package called **Transient Fault Handling Application Block – Declarative Configuration Support.**

To add the Transient Fault Handling Application Block – Declarative Configuration Support to the application

1. In Solution Explorer, right-click **References**, and then click **Manage NuGet Packages**.
2. In the NuGet Package Manager, click **Online**, then select NuGet official package source. In the **Search Online** box type **Topaz** to find the NuGet package that contains the Transient Fault Handling Application Block binaries. Click the **Install** button to add **Enterprise Library – Transient Fault Handling Application Block – Declarative Configuration Support** to your project. Review the licenses and then click **I Accept**. Click **Close** to close the NuGet Package Manager.

To use configuration to define a retry policy

1. In Solution Explorer, in the **TransientFaultHAndlingHOL** project, right-click the **App.config** file, and then click **Edit configuration file v6**. The Enterprise Library configuration console opens.

2. In the **Blocks** menu, click **Add Transient Fault Handling Settings**. Click the **plus** symbol next to **Retry Strategies**, point to **Add Retry Strategies**, and then click **Add Fixed Interval** to add a new strategy to the configuration.
3. Change the name of the new strategy to **HOL Strategy**, set the interval to five seconds, and the maximum number of retries to two. Note that these are the same values that you defined in code in the first exercise.
4. On the **File** menu click **Save**, and then on the **File** menu click **Exit** to close the Enterprise Library configuration console.
5. In Solution Explorer, right-click **Program.cs**, and then click **View code**. Add the **using** clauses highlighted below:

```
using System;
using System.Windows.Forms;
using
Microsoft.Practices.EnterpriseLibrary.TransientFaultHandling.Configuration;
using Microsoft.Practices.EnterpriseLibrary.Common.Configuration;
using Microsoft.Practices.EnterpriseLibrary.TransientFaultHandling;
```

6. Set the Retry Manager by Adding the following lines to the main method

```
[STAThread]
static void Main()
{
    Application.EnableVisualStyles();
    Application.SetCompatibleTextRenderingDefault(false);
    var settings =
RetryPolicyConfigurationSettings.GetRetryPolicySettings(new
SystemConfigurationSource());
    RetryPolicyFactory.SetRetryManager(settings.BuildRetryManager());
    Application.Run(new Main());
}
```

7. In the code editor, open the file **Main.cs**, make the following highlighted changes to the **ExecuteQueryButton_Click** method to load the retry policy named "HOL Strategy" from the configuration file. This approach uses the static **GetRetryPolicy** method to load the retry policy.

```
command.Parameters.AddWithValue("@ProductID", productId);
command.Parameters.AddWithValue("@SessionID", sessionId);
command.Parameters.AddWithValue("@NumberOfTriesBeforeSucceeding",
attempts);

var policy =
RetryPolicyFactory.GetRetryPolicy<HolSqlTransientErrorDetectionStrategy>("H
OL Strategy");
```

8. On the **File** menu, click **Save All**.
9. On the **Debug** menu, click **Start Debugging** to run the application. You will observe the same behavior as you observed in Exercise 1 when you test the application with values of **1**, **2**, **3**, and **4** in the **Successful attempt** box.
10. Stop the application running.

Summary

To verify that you have completed this lab correctly, you can use the solution provided in the **Lab01\end** folder.

> If you choose to use the end solution, you must still create the database and add all the required objects (tables and stored procedures) to SQL Server and add the Transient Fault Handling Application Block binaries to the project as described above.

In this lab, you used a stored procedure in SQL Server to simulate a transient error in SQL Database. You learned how to define a retry policy in code and in configuration. You also learned how to instantiate a retry manager.

In this lab, the code that you used to call the stored procedure in SQL Server executed synchronously. In the next lab, you will learn how to use the Transient Fault Handling Application Block with asynchronous calls.

Lab 2: Using the Transient Fault Handling Application Block Asynchronously

Using the Transient Fault Handling Application Block with an Asynchronous Call

By definition, transient faults only occur intermittently, so to simplify the lab exercises, you will use a local database running on Microsoft SQL Server® software to simulate transient faults. This makes it easier for you to observe the behavior of the Transient Fault Handling Application Block.

To examine the application

1. If you did Lab1 you may have the required database already installed, if not, follow the instructions described in lab1.
2. In this lab, the Transient Fault Handling Application Block NuGet Package have already been added to the project along with a custom error detection strategy for SQL Server.
3. On the **Debug** menu, click **Start Debugging** to run the application.
4. If you click the **Execute query** button while the **Successful attempt** box contains the number **1**, the application displays a result in the UI.
5. If you click the **Execute query** button while the **Successful attempt** box contains a number greater than **1**, the application displays an error message in the UI that reports a transient error.
6. Stop the application.

In this lab, the client application makes an asynchronous call to SQL Server. You will learn how to use the retry policy's **ExecuteAsync** method which repeatedly executes the specified asynchronous task while it satisfies the current retry policy.

To implement the retry behavior for the asynchronous call

1. In **Main.cs**, add the **using** statement highlighted below:

```
using System;
using System.Configuration;
using System.Data;
using System.Data.SqlClient;
using System.Globalization;
using System.Windows.Forms;
using Microsoft.Practices.EnterpriseLibrary.TransientFaultHandling;
```

2. Modify the code in the **ExecuteQueryButton_Click** method as shown highlighted below to create a retry policy. The retry policy specifies that the Transient Fault Handling Application

Block will make two retry attempts. After the first retry, the application block will wait for five seconds before it tries again.

```
command.Parameters.AddWithValue("@ProductID", productId);
command.Parameters.AddWithValue("@SessionID", sessionId);
command.Parameters.AddWithValue("@NumberOfTriesBeforeSucceeding",
attempts);

var policy = new RetryPolicy<HolSqlTransientErrorDetectionStrategy>(2,
TimeSpan.FromSeconds(5));

  policy.Retrying += (s, a) =>
    this.Log(string.Format(CultureInfo.CurrentCulture,
       "Retry attempt {1} delay {2}: {0}",
       a.LastException.Message,
       a.CurrentRetryCount,
       a.Delay
      )
    );

// Execute the command
```

3. Replace the using statement with the highlighted one. So the defined policy's **ExecuteAsync** method is included

```
// Execute the command

using (var reader = await policy.ExecuteAsync(() =>
command.ExecuteReaderAsync()))
{
    // For each record
    while (reader.Read())
    {
       // Log the result
       this.Log(string.Format("Product name : {0} ",
reader["ProductName"]));
     }
}
```

4. On the **File** menu, click **Save All**.
5. On the **Debug** menu, click **Start Debugging** to run the application.
6. If you click the **Execute Query** button while the **Successful attempt** box contains the number **1**, the application displays a result in the UI.
7. If you click the **Execute Query** button while the **Successful attempt** box contains the number **2**, the application displays a result in the UI. The first call to the stored procedure fails, but the Transient Fault Handling Application Block immediately makes a second attempt (first retry) that succeeds.
8. If you click the **Execute Query** button while the **Successful attempt** box contains the number **3**, then after a pause, the application displays a result in the UI. The first call to the stored procedure fails, the Transient Fault Handling Application Block immediately makes a second

attempt (first retry) that also fails, and the block then waits for five seconds before it makes a third attempt (second retry) that succeeds.

9. If you click the **Execute Query** button while the **Successful attempt** box contains the number **4**, then after a pause the application displays an error message in the UI. The first call to the stored procedure fails, the Transient Fault Handling Application Block immediately makes a second attempt (first retry) that also fails, and the block waits for five seconds before it makes a third attempt (second retry) that also fails. After two unsuccessful retries the block returns a faulted Task.
10. Stop the application.

Summary

To verify that you have completed this lab correctly, you can use the solution provided in the **Lab02\end** folder.

If you choose to use the end solution, you must still create the database and add all the required objects (tables and stored procedures) to SQL Server.

In this lab, you used a stored procedure in SQL Server to simulate a transient error in the SQL Database. You learned how to use the Transient Fault Handling Application Block to implement retry policies for asynchronous calls.

Copyright