<https://www.c-sharpcorner.com/UploadFile/b182bf/centralize-exception-handling-in-wcf-part-10/>

Before reading this article, I highly recommend that you read the previous parts of this series:

* [Introduction to WCF: Part 1](http://www.c-sharpcorner.com/UploadFile/b182bf/introduction-to-wcf/)
* [Introduction to Endpoint in WCF: Part 2](http://www.c-sharpcorner.com/UploadFile/b182bf/introduction-to-endpoint-in-wcf/)
* [How to Make Changes to WCF Service Without Breaking Client in WCF: Part 3](http://www.c-sharpcorner.com/UploadFile/b182bf/how-to-make-changes-to-wcf-service-without-breaking-client-i/)
* [Method Overloading in WCF: Part 4](http://www.c-sharpcorner.com/UploadFile/b182bf/method-overloading-in-wcf-%E2%80%93-part-4/)
* [What a DataContract and DataMember are in WCF: Part 5](http://www.c-sharpcorner.com/UploadFile/b182bf/what-a-datacontract-and-datamember-are-in-wcf-part-5/)
* [KnownType Attribute in WCF: Part 6](http://www.c-sharpcorner.com/UploadFile/b182bf/knowntype-attribute-in-wcf-part-6/)
* [MessageContract in WCF: Part 7](http://www.c-sharpcorner.com/UploadFile/b182bf/messagecontract-in-wcf-part-7/)
* [Tracing in WCF: Part 8](http://www.c-sharpcorner.com/UploadFile/b182bf/tracing-in-wcf-part-8/)
* [Exception handling in WCF: Part 9](http://www.c-sharpcorner.com/UploadFile/b182bf/exception-handling-in-wcf-part-9/)

I will use the same application as in my previous articles.

Let's add a class to the wcfService project and name it MyAppErrorHandler.cs and implement an IErrorHandler interface. The **IErrorHandle** interface is in the **System.ServiceModel.Dispatcher** namespace.

The IErrorHandler interface has the following two methods.

1. **HandleError:**The return type of this method is bool and it is called asynchronously after providing the fault method.
2. **ProvideFault:** The return type of this method is void. Whenever there is an unhandled exception, this method is called automatically. In this function will convert the .NET exception into a SOAP fault exception.

Let's understand the use of these functions. For this I will create a table and Stored Procedure.

I will create a table “**Exception\_Master**” and a procedure “**USP\_INSERTEXCEPTION**” for saving the exceptions in the database.

1. **CREATE** **TABLE** [dbo].[Exception\_Master](
2. [ID] [**int**] IDENTITY(1,1) NOT NULL,
3. [Message] [**varchar**](100) NULL,
4. [Type] [**varchar**](50) NULL,
5. [Source] [**varchar**](**max**) NULL,
6. [CreatedDate] [**date**] NULL
7. )
9. **CREATE** **PROCEDURE** [DBO].[USP\_INSERTEXCEPTION]
10. (
11. @Message [**varchar**](100) = NULL,
12. @Type [**varchar**](50) = NULL,
13. @Source [**varchar**](**max**) = NULL
14. )
15. **AS**
16. **BEGIN**
17. **INSERT** **INTO** Exception\_Master(Message,Type,Source) **VALUES**(@Message,@Type,@Source)
18. **END**

Now I will implement the IErrorHandle interface in the MyAppErrorHandler class.

**MyAppErrorHandler.cs**

1. **using** System;
2. **using** System.ServiceModel;
3. **using** System.ServiceModel.Channels;
4. **using** System.ServiceModel.Dispatcher;
5. **using** System.Data;
6. **using** System.Data.SqlClient;
8. **namespace** wcfService
9. {
10. **public** **class** MyAppErrorHandler : IErrorHandler
11. {
12. SqlConnection con = **new** SqlConnection("Data Source=.;Initial Catalog=Goods;Integrated Security=True;");
13. **public** **bool** HandleError(Exception error)
14. {
15. **string** strSQL = **string**.Empty;
16. **bool** status = **true**;
17. con.Open();
18. **try**
19. {
20. con.Open();
21. SqlCommand cmd = **new** SqlCommand("USP\_INSERTEXCEPTION",con);
22. cmd.CommandType = CommandType.StoredProcedure;
23. cmd.Parameters.AddWithValue("@Message", error.Message.ToString());
24. cmd.Parameters.AddWithValue("@Type", error.GetType().Name.ToString());
25. cmd.Parameters.AddWithValue("@Source", error.Source.ToString());
26. cmd.ExecuteNonQuery();
27. con.Close();
28. }
29. **catch** (Exception ex) {
30. status = **false**;
31. }
32. **return** status;
33. }
35. **public** **void** ProvideFault(Exception error, MessageVersion version, **ref** Message fault)
36. {
37. **if** (error **is** FaultException)
38. **return**;
40. FaultException FE = **new** FaultException("This is test Fault Exception");
41. MessageFault MF = FE.CreateMessageFault();
42. fault = Message.CreateMessage(version, MF, **null**);
43. }
44. }
45. }

Let's add one more class to our project and name it “**ErrorHandlerAttribute**”. Now I will inherit this class from the “**Attribute**” abstract class and implement it by the “**IServiceBehavior**” interface.

1. **using** System;
2. **using** System.ServiceModel.Description;
3. **using** System.ServiceModel.Dispatcher;
5. **namespace** wcfService
6. {
7. **public** **class** ErrorHandlerAttribute : Attribute,IServiceBehavior
8. {
9. **private** **readonly** Type errorHandler;
10. **public** ErrorHandlerAttribute(Type errorHandler)
11. {
12. **this**.errorHandler = errorHandler;
13. }
14. **public** **void** AddBindingParameters(ServiceDescription serviceDescription, System.ServiceModel.ServiceHostBase serviceHostBase, System.Collections.ObjectModel.Collection<ServiceEndpoint> endpoints, System.ServiceModel.Channels.BindingParameterCollection bindingParameters)
15. {
17. }
19. **public** **void** ApplyDispatchBehavior(ServiceDescription serviceDescription, System.ServiceModel.ServiceHostBase serviceHostBase)
20. {
21. IErrorHandler handler = (IErrorHandler)Activator.CreateInstance(**this**.errorHandler);
22. **foreach** (ChannelDispatcherBase channelDispatcherBase **in** serviceHostBase.ChannelDispatchers)
23. {
24. ChannelDispatcher channelDispatcher = channelDispatcherBase **as** ChannelDispatcher;
25. **if** (channelDispatcher != **null**) {
26. channelDispatcher.ErrorHandlers.Add(handler);
27. }
28. }
29. }
31. **public** **void** Validate(ServiceDescription serviceDescription, System.ServiceModel.ServiceHostBase serviceHostBase)
32. {
34. }
35. }
36. }

There is no need to provide any implementation of the Validate and AddBindingParameters methods as of now.

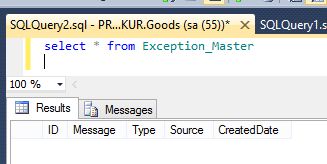
Now I will decorate the “**MathService**” class with “**ErrorHandlerAttribute**”.

**In MathService.svc.cs**

1. **using** System;
3. **namespace** wcfService
4. {
5. [ErrorHandlerAttribute(**typeof**(MyAppErrorHandler))]
6. **public** **class** MathService : IMathService
7. {
8. **public** **int** AddTwoNo(**int** FirstNo, **int** Second)
9. {
10. **return** FirstNo + Second;
11. }

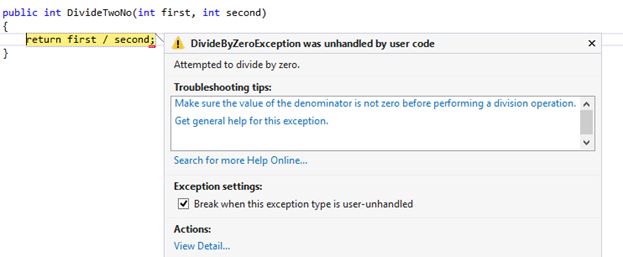
14. **public** **int** DivideTwoNo(**int** first, **int** second)
15. {
16. **return** first / second;
17. }
18. }
19. }

Let's run the SQL query and see the output.

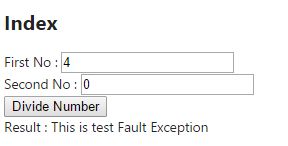


Now run the application and see the output. You can make a breakpoint on ProvideFault and HandleError for checking that the HandleError method works asynchronously or not.

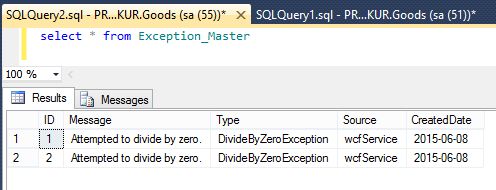
When running the application it will throw an error.



Press F5 and continue to run the application and see the output.



Let's run the SQL query and see whether or not the output is the error saved in the database.



I hope you will enjoy this article.