<https://www.c-sharpcorner.com/UploadFile/00a8b7/exception-handling-in-wcf-service/>

Exception Handling in WCF Service

.NET provides a very simple way of dealing with exceptions using try-catch-throw to provide a user an understandable message for a system generated exception.

But a WCF service is consumed by some other application, not an end user. If something goes wrong in a service, the developer of the client application should know what's going wrong in the service. Again, the client application developer shouldn't know each and every exception in detail, it may lead to some security issues. Also, sometimes a client should be instructed using exceptions.

Again, the Exception class is .NET specific and a WCF Service is consumed by an application that may be in some other language like Java, PHP, ROR and so on. So exceptions may or may not be understood by the client applications.

So, things should be under the control of the service end. For this we have FaultException and FaultContract. Now, let's dive into exception handling in WCF.

Let's create a service that will perform a division operation. We will use the “DivideByZeroException” to demonstrate Exception handling in a WCF service.

I believe you know how to create a WCF service and consume it. You can download the democode for a better understanding on “Exception handling in WCF service”.

**IArithmaticService.cs**

1. **using** System.Runtime.Serialization;
2. **using** System.ServiceModel;
4. **namespace** FaultContractDemo
5. {
6. [ServiceContract]
7. **public** **interface** IArithmaticService
8. {
9. [OperationContract]
10. **int** Divide(**int** Divident, **int** Divisor);
11. }
12. }

**ArithmaticService.svc.cs**

1. **using** System;
2. **using** System.ServiceModel;
4. **namespace** FaultContractDemo
5. {
6. **public** **class** ArithmaticService : IArithmaticService
7. {
8. **public** **int** Divide(**int** Divident, **int** Divisor)
9. {
10. **int** Quotient;
11. **try**
12. {
13. Quotient = Divident / Divisor;
14. }
15. **catch** (Exception Ex)
16. {
17. **throw** Ex;
18. }
19. **return** Quotient;
20. }
21. }
22. }

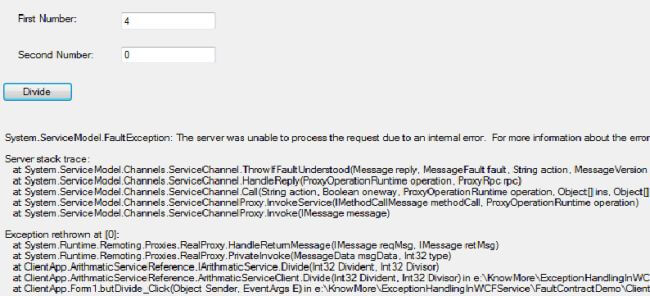
Now, let's create a client application to consume this service.

**Form1.cs**

1. **using** System;
2. **using** System.ServiceModel;
3. **using** System.Windows.Forms;
5. **namespace** ClientApp
6. {
7. **public** partial **class** Form1 : Form
8. {
9. **public** Form1()
10. {
11. InitializeComponent();
12. }
14. **private** **void** butDivide\_Click(**object** Sender, EventArgs E)
15. {
16. **try**
17. {
18. **int** Divident, Divisor;
19. **int**.TryParse(txtFirstNum.Text, **out** Divident);
20. **int**.TryParse(txtSecondNum.Text, **out** Divisor);
21. var ObjProxy = **new** ArithmaticServiceReference.ArithmaticServiceClient();
22. lblQuotient.Text = ObjProxy.Divide(Divident, Divisor).ToString();
23. }
24. **catch** (Exception Ex)
25. {
26. lblException.Text = Ex.Message;
27. }
29. }
30. }
31. }

2 non-zero numbers, you should get their quotient.

Now, zero for the divisor. You will get an exception as follows.



This exception message indicates something went wrong in the service. But, this is not giving any clear message of what went wrong.

Since the service and client applications are interacting with each other using SOAP, we need to exception details in SOAP format. We can use FaultException &/or FaultContract (in System.ServiceModel namespace) to these exception details to the client.

FaultException

Let's see how to use a FaultException.

* You need to create an instance of the FaultException class with FaultReason/FaultCode/MessageFault and throw it.
* In the client application add a catch block with an instance of FaultException and use it's properties.

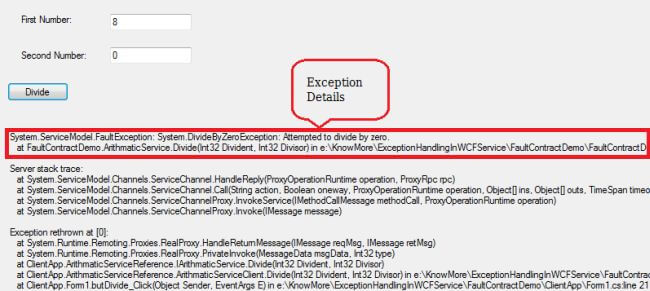
**ArithmaticService.svc.cs**

1. **using** System;
2. **using** System.ServiceModel;
4. **namespace** FaultContractDemo
5. {
6. **public** **class** ArithmaticService : IArithmaticService
7. {
8. **public** **int** Divide(**int** Divident, **int** Divisor)
9. {
10. **int** Quotient;
11. **try**
12. {
13. Quotient = Divident / Divisor;
14. }
15. **catch** (Exception Ex)
16. {
17. **throw** **new** FaultException(Ex.ToString());
18. }
19. **return** Quotient;
20. }
21. }
22. }

**Form1.cs**

1. **using** System;
2. **using** System.ServiceModel;
3. **using** System.Windows.Forms;
5. **namespace** ClientApp
6. {
7. **public** partial **class** Form1 : Form
8. {
9. **public** Form1()
10. {
11. InitializeComponent();
12. }
14. **private** **void** butDivide\_Click(**object** Sender, EventArgs E)
15. {
16. **try**
17. {
18. **int** Divident, Divisor;
19. **int**.TryParse(txtFirstNum.Text, **out** Divident);
20. **int**.TryParse(txtSecondNum.Text, **out** Divisor);
21. var ObjProxy = **new** ArithmaticServiceReference.ArithmaticServiceClient();
22. lblQuotient.Text = ObjProxy.Divide(Divident, Divisor).ToString();
23. }
24. **catch** (FaultException Ex)
25. {
26. lblException.Text = Ex.Message;
27. }
28. }
29. }
30. }

Now, if you zero for the divisor then you will get an exception like:



FaultContract

If you want to send some additional details (maybe to describe some business requirements) to the client then you can use a FaultContract.

* Create a custom class with properties for holding the details you want to send to the client application.
* Decorate this custom class with a DataContract attribute and it's properties with a DataMember attribute.
* Decorate the method with a FaultContract attribute with the type of object you want to throw.
* In the client application, create a catch block with an instance of the custom fault class and use it's properties.

Add a class to the IArithmaticService interface in IArithmaticService.cs as in the following:

1. **using** System.Runtime.Serialization;
2. **using** System.ServiceModel;
4. **namespace** FaultContractDemo
5. {
6. [ServiceContract]
7. **public** **interface** IArithmaticService
8. {
9. [OperationContract]
10. [FaultContract(**typeof**(CustomExceptionDetails))]
11. **int** Divide(**int** Divident, **int** Divisor);
12. }
14. /// <summary>
15. /// Custom class for exception details
16. /// </summary>
17. [DataContract]
18. **public** **class** CustomExceptionDetails
19. {
20. [DataMember]
21. **public** **int** Id { **get**; **set**; }
23. [DataMember]
24. **public** **string** Message { **get**; **set**; }
26. [DataMember]
27. **public** **string** Details { **get**; **set**; }
28. }
29. }

**ArithmaticService.svc.cs**

1. **using** System;
2. **using** System.ServiceModel;
4. **namespace** FaultContractDemo
5. {
6. **public** **class** ArithmaticService : IArithmaticService
7. {
8. **public** **int** Divide(**int** Divident, **int** Divisor)
9. {
10. **int** Quotient;
11. **try**
12. {
13. **if** (Divisor == 0)
14. {
15. var ObjCustom = **new** CustomExceptionDetails
16. {
17. Message = "Divisor should be non-zero.",
18. Details = @"Dividing a number by 0 will always give you infinity,                           so enter a non-zero number for divisor.",
19. Id = 1
20. };
21. **throw** **new** FaultException<CustomExceptionDetails>(ObjCustom);
22. }
23. Quotient = Divident / Divisor;
24. }
25. **catch** (Exception Ex)
26. {
27. **throw** Ex;
28. }
29. **return** Quotient;
30. }
31. }
32. }

**Form1.cs**

1. **using** System;
2. **using** System.ServiceModel;
3. **using** System.Windows.Forms;
5. **namespace** ClientApp
6. {
7. **public** partial **class** Form1 : Form
8. {
9. **public** Form1()
10. {
11. InitializeComponent();
12. }
14. **private** **void** butDivide\_Click(**object** Sender, EventArgs E)
15. {
16. **try**
17. {
18. **int** Divident, Divisor;
19. **int**.TryParse(txtFirstNum.Text, **out** Divident);
20. **int**.TryParse(txtSecondNum.Text, **out** Divisor);
21. var ObjProxy = **new** ArithmaticServiceReference.ArithmaticServiceClient();
22. lblQuotient.Text = ObjProxy.Divide(Divident, Divisor).ToString();
23. }
24. **catch** (FaultException<ArithmaticServiceReference.CustomExceptionDetails> Ex)
25. {
26. lblException.Text = "Id: " + Ex.Detail.Id + "\nMessage: " + Ex.Detail.Message +
27. "\nDetails: " + Ex.Detail.Details;
28. }
29. **catch** (FaultException Ex)
30. {
31. lblException.Text = Ex.Message;
32. }
33. **catch** (Exception Ex)
34. {
35. lblException.Text = Ex.Message;
36. }
37. }
38. }