**Service Contract**

Service contract describes the operation that service provide. A Service can have more than one service contract but it should have at least one Service contract.

Service Contract can be define using [ServiceContract] and [OperationContract] attribute. [ServiceContract] attribute is similar to the [WebServcie] attribute in the WebService and [OpeartionContract] is similar to the [WebMethod] in WebService.

* It describes the client-callable operations (functions) exposed by the service
* It maps the interface and methods of your service to a platform-independent description
* It describes message exchange patterns that the service can have with another party. Some service operations might be one-way; others might require a request-reply pattern
* It is analogous to the element in WSDL

To create a service contract you define an interface with related methods representative of a collection of service operations, and then decorate the interface with the *ServiceContract* Attribute to indicate it is a service contract. Methods in the interface that should be included in the service contract are decorated with the *OperationContract* Attribute.

[ServiceContract()]

public interface ISimpleCalculator

{

[OperationContract()]

int Add(int num1, int num2);

}

Once we define Service contract in the interface, we can create implement class for this interface.

public class SimpleCalculator : ISimpleCalculator

{

public int Add(int num1, int num2)

{

return num1 + num2;

}

}

With out creating the interface, we can also directly created the service by placing Contract in the implemented class. But it is not good practice of creating the service

[ServiceContract()]

public class SimpleCalculator

{

[OperationContract()]

public int Add(int num1, int num2)

{

return num1 + num2;

}

}

**Data Contract**

A data contract is a formal agreement between a service and a client that abstractly describes the data to be exchanged.

Data contract can be explicit or implicit. Simple type such as int, string etc has an implicit data contract. User defined object are explicit or Complex type, for which you have to define a Data contract using [DataContract] and [DataMember] attribute.

A data contract can be defined as follows:

* It describes the external format of data passed to and from service operations
* It defines the structure and types of data exchanged in service messages
* It maps a CLR type to an XML Schema
* It defines how data types are serialized and deserialized. Through serialization, you convert an object into a sequence of bytes that can be transmitted over a network. Through deserialization, you reassemble an object from a sequence of bytes that you receive from a calling application.
* It is a versioning system that allows you to manage changes to structured data

We need to include System.Runtime.Serialization reference to the project. This assembly holds the DataContract and DataMember attribute.

Create user defined data type called Employee. This data type should be identified for serialization and deserialization by mentioning with [DataContract] and [DataMember] attribute.

[ServiceContract]

public interface IEmployeeService

{

[OperationContract]

Employee GetEmployeeDetails(int EmpId);

}

[DataContract]

public class Employee

{

private string m\_Name;

private int m\_Age;

private int m\_Salary;

private string m\_Designation;

private string m\_Manager;

[DataMember]

public string Name

{

get { return m\_Name; }

set { m\_Name = value; }

}

}

Implementation of the service class is shown below. In GetEmployee method we have created the Employee instance and return to the client. Since we have created the data contract for the Employee class, client will aware of this instance whenever he creates proxy for the service.

public class EmployeeService : IEmployeeService

{

public Employee GetEmployeeDetails(int empId)

{

Employee empDetail = new Employee();

//Do something to get employee details and assign to 'empDetail' properties

return empDetail;

}

}

**Client side**

On client side we can create the proxy for the service and make use of it. The client side code is shown below.

protected void btnGetDetails\_Click(object sender, EventArgs e)

{

EmployeeServiceClient objEmployeeClient = new EmployeeServiceClient();

Employee empDetails;

empDetails = objEmployeeClient.GetEmployeeDetails(empId);

//Do something on employee details

}

**Message Contract**

**Message**

Message is the packet of data which contains important information. WCF uses these messages to transfer information from Source to destination.

WCF uses SOAP(Simple Object Access Protocol) Message format for communication. SOAP message contain Envelope, Header and Body.SOAP envelope contails name, namespace,header and body element. SOAP Hear contain important information which are not directly related to message. SOAP body contains information which is used by the target.

**Message Pattern**

It describes how the programs will exchange message each other. There are three way of communication between source and destination

1. **Simplex** - It is one way communication. Source will send message to target, but target will not respond to the message.
2. **Request/Replay** - It is two way communications, when source send message to the target, it will resend response message to the source. But at a time only one can send a message
3. **Duplex** - It is two way communication, both source and target can send and receive message simultaniouly.

What is Message contract?

As I said earlier, WCF uses SOAP message for communication. Most of the time developer will concentrate more on developing the DataContract, Serializing the data, etc. WCF will automatically take care of message. On Some critical issue, developer will also require control over the SOAP message format. In that case WCF provides Message Contract to customize the message as per requirement.

WCF supports either RPC(Remote Procedure Call) or Message style operation model. In the RPC model, you can develop operation with Ref and out parameter. WCF will automatically create the message for operation at run time. In Message style operation WCF allows to customize the message header and define the security for header and body of the message.

**Defining Message Contract**

Message contract can be applied to type using MessageContract attribute. Custom Header and Body can be included to message using 'MessageHeader' and '*MessageBodyMember*'atttribute. Let us see the sample message contract definition.

[MessageContract]

public class EmployeeDetails

{

[MessageHeader]

public string EmpID;

[MessageBodyMember]

public string Name;

[MessageBodyMember]

public string Designation;

[MessageBodyMember]

public int Salary;

[MessageBodyMember]

public string Location;

}

When I use this EmployeeDeatils type in the service operation as parameter. WCF will add extra header call 'EmpID' to the SOAP envelope. It also add Name, Designation, Salary, Location as extra member to the SOAP Body.

Rules :

You have to follow certain rules while working with Message contract

* When using Message contract type as parameter, Only one parameter can be used in servicie Operation

[OperationContract]

void SaveEmployeeDetails(EmployeeDetails emp);

* Service operation either should return Messagecontract type or it should not return any value

[OperationContract]

EmployeeDetails GetEmployeeDetails();

* Service operation will accept and return only message contract type. Other data types are not allowed.

[OperationContract]

EmployeeDetails ModifyEmployeeDetails(EmployeeDetails emp);

**MessageHeaderArray Attribute**

Consider the Message contract type definition as shown below.

[MessageContract]

public class Department

{

[MessageHeader]

public string DepartmentID;

[MessageHeader]

public string DepartmentName;

[MessageHeader]

public Employees Employee();

}

In this we are having array of Employee type as message header. When this converted to SOAP Header it looks as shown below.

<Department>

<DepartmentID>PRO1243</DepartmentID>

<DepartmentName>Production</DepartmentName>

<Employees>

<Employee>Sam</Employee>

<Employee>Ram</Employee>

<Employee>Raja</Employee>

</Employees>

</Department>

Suppose you want to show the all employee detail in same level. We can use MessageHeaderArray attribute which will serialize the array element independently. If you use the MessageHeaderArray attribute of Employees, SOAP message will look as shown below.

<Department>

<DepartmentID>PRO1243</DepartmentID>

<DepartmentName>Production</DepartmentName>

<Employee>Sam</Employee>

<Employee>Ram</Employee>

<Employee>Raja</Employee>

</Department>

**Note:** MessageHeaderArray Attribute is applicable only for Array, not for collection.

**Message Contract Properties**

**ProtectionLevel**

You can mention the *MessageHeader* or *MessageBodyMember* to be signed or Encrypted using *ProtectionLevel* property.

**Example**

using System.Net.Security;

[MessageContract]

public class EmployeeDetails

{

[MessageHeader(ProtectionLevel=ProtectionLevel.None)]

public string EmpID;

[MessageBodyMember(ProtectionLevel = ProtectionLevel.Sign )]

public string Name;

[MessageBodyMember(ProtectionLevel = ProtectionLevel.Sign )]

public string Designation;

[MessageBodyMember(ProtectionLevel=ProtectionLevel.EncryptAndSign)]

public int Salary;

}

In the above type definition, we have made the different protection level for body. But the protection level of the body is determind by the highest *ProtectionLevel* property. By default if you are not specifying the protection level it takes 'EncryptAndSign'. So it good if you specify minimum ProtectionLevel required.

**Name and Namespace:**

SOAP representation of the message element can be change by mentioning Name and Namespace property of the Header and Body member. By default namespace is the same as the namespace of the service contract that the message is participating. In the below example, I have mention the Name property to the EmpID and Name.

[MessageContract]

public class EmployeeDetails

{

[MessageHeader(Name="ID")]

public string EmpID;

[MessageBodyMember(Name="EmployeeName")]

public string Name;

[MessageBodyMember()]

public string Designation;

[MessageBodyMember()]

public int Salary;

}

When SOAP message representation, its name is changed to ID and EmployeeName.

<EmployeeDetails>

<ID>45634</ID>

<EmployeeName>Sam</EmployeeName>

<Designation>Software Engineer</Designation>

<Salary>25000</Salary>

</EmployeeDetails>

**Order**

The order of the body elements are alpehabetical by default. But you can control the order, using *Order* property in the *MessageBody* attribute.

[MessageContract]

public class EmployeeDetails

{

[MessageHeader()]

public string EmpID;

[MessageBodyMember(Order=2)]

public string Name;

[MessageBodyMember(Order=3)]

public string Designation;

[MessageBodyMember(Order=1)]

public int Salary;

}

**Fault Contract**

Service that we develop might get error in come case. This error should be reported to the client in proper manner. Basically when we develop managed application or service, we will handle the exception using try- catch block. But these exceptions handlings are technology specific.

In order to support interoperability and client will also be interested only, what wents wrong? not on how and where cause the error.

By default when we throw any exception from service, it will not reach the client side. WCF provides the option to handle and convey the error message to client from service using SOAP Fault contract.

Suppose the service I consumed is not working in the client application. I want to know the real cause of the problem. How I can know the error? For this we are having Fault Contract. Fault Contract provides documented view for error accorded in the service to client. This help as to easy identity the what error has accord. Let us try to understand the concept using sample

example.

Step 1: I have created simple calculator service with Add operation which will throw general exception as shown below

//Service interface

[ServiceContract()]

public interface ISimpleCalculator

{

[OperationContract()]

int Add(int num1, int num2);

}

//Service implementation

public class SimpleCalculator : ISimpleCalculator

{

public int Add(int num1, int num2)

{

//Do something

throw new Exception("Error while adding number");

}

}

**Step 2:** On client side code. Exceptions are handled using try-Catch block. Even though I have capture the exception when I run the application. I got the message that exceptions are not handled properly.

try

{

MyCalculatorServiceProxy.MyCalculatorServiceProxy proxy

= new MyCalculatorServiceProxy.MyCalculatorServiceProxy();

Console.WriteLine("Client is running at " + DateTime.Now.ToString());

Console.WriteLine("Sum of two numbers... 5+5 =" + proxy.Add(5, 5));

Console.ReadLine();

}

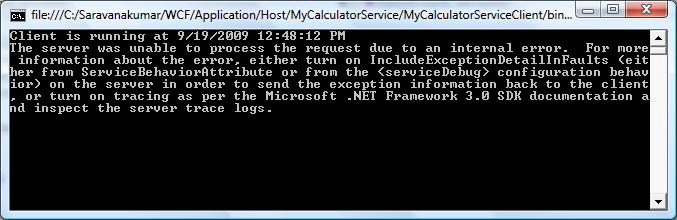
catch (Exception ex)

{

Console.WriteLine(ex.Message);

Console.ReadLine();

}



**Step 3:** Now if you want to send exception information form service to client, you have to use FaultException as shown below.

public int Add(int num1, int num2)

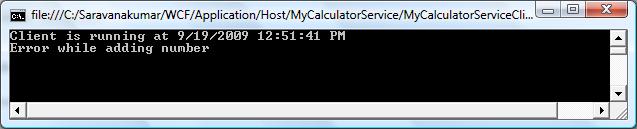
{

//Do something

throw new FaultException("Error while adding number");

}

**Step 4:** Output window on the client side is show below.



**Step 5:** You can also create your own Custom type and send the error information to the client using *FaultContract*. These are the steps to be followed to create the fault contract.

* Define a type using the data contract and specify the fields you want to return.
* Decorate the service operation with the FaultContract attribute and specify the type name.
* Raise the exception from the service by creating an instance and assigning properties of the custom exception.

**Step 6:** Defining the type using Data Contract

[DataContract()]

public class CustomException

{

[DataMember()]

public string Title;

[DataMember()]

public string ExceptionMessage;

[DataMember()]

public string InnerException;

[DataMember()]

public string StackTrace;

}

**Step 7:** Decorate the service operation with the *FaultContract*

[ServiceContract()]

public interface ISimpleCalculator

{

[OperationContract()]

[FaultContract(typeof(CustomException))]

int Add(int num1, int num2);

}

**Step 8:** Raise the exception from the service

public int Add(int num1, int num2)

{

//Do something

CustomException ex = new CustomException();

ex.Title = "Error Funtion:Add()";

ex.ExceptionMessage = "Error occur while doing add function.";

ex.InnerException = "Inner exception message from serice";

ex.StackTrace = "Stack Trace message from service.";

throw new FaultException(ex,"Reason: Testing the Fault contract") ;

}

**Step 9:** On client side, you can capture the service exception and process the information, as shown below.

try

{

MyCalculatorServiceProxy.MyCalculatorServiceProxy proxy

= new MyCalculatorServiceProxy.MyCalculatorServiceProxy();

Console.WriteLine("Client is running at " + DateTime.Now.ToString());

Console.WriteLine("Sum of two numbers... 5+5 =" + proxy.Add(5, 5));

Console.ReadLine();

}

catch (FaultException<MyCalculatorService.CustomException> ex)

{

//Process the Exception

}

**Summary**

\* **A Service** is really a **collection of endpoints**, and the endpoints implement the specific algorithms in code. They can implement high-level business functions, such as entering orders into a fulfilment system,or they can be more fine-grained, such as looking up a customer’s   
address. High-level functions typically require complex data structures,whereas targeted functions often work in more basic data types. In either case, an endpoint must specify the operations it implements and the data formats it expects. Together, these specifications make up the contract

\* There are three types of contracts in WCF:

• **Service contracts**. Service contracts describe the functional operations implemented by the service. A service contract maps the class methods of a .NET type to WSDL services, port types, and operations.Operation contracts within service contracts describe the service   
operations, which are the methods that implement functions of   
the service.

• **Data contracts**. Data contracts describe data structures that are used by the service to communicate with clients. A data contract maps CLR types to XML Schema Definitions (XSD) and defines how they are serialized and DE serialized. Data contracts describe all the data   
that is sent to or from service operations.

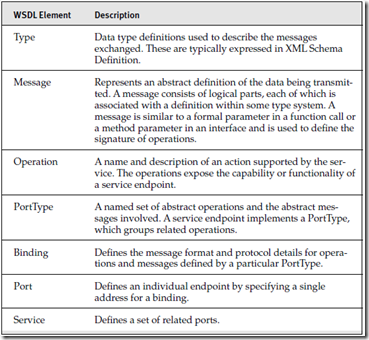
• **Message contracts**. Message contracts map CLR types to SOAP messages and describe the format of the SOAP messages and affect the WSDL and XSD definitions of those messages. Message contracts provide precise control over the SOAP headers and bodies.

**NOTE :**

To make contracts interoperable with the widest range of systems, they are expressed in [**Web Service Description Language(WSDL)**](http://en.wikipedia.org/wiki/Web_Services_Description_Language). So, before going too much further in discussing contracts, a short review of WSDL is helpful. According to the W3C, the standards body through which industry vendors (Microsoft, IBM, and so on) defined the WSDL specification.

**WSDL** :- WSDL is an XML format for describing network services as a set of endpoints operating on messages containing either document-oriented or procedure-oriented information. The operations and messages are described abstractly, and then bound to a concrete network protocol and message format to define an endpoint. Related concrete endpoints are combined into abstract endpoints (services). WSDL is extensible to allow description of endpoints and their messages regardless of what message formats or network protocols are used to communicate[;] however, the only bindings described in this document describe how to use WSDL in conjunction with SOAP 1.1, HTTP GET/POST, and MIME.]

**WSDL Elements are as follows :-**



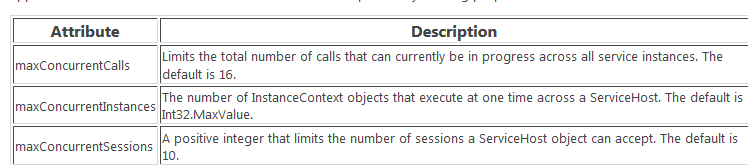
## Durable Service

Durable services are WCF services that persist service state information even after service host is restarted or Client. It means that durable services have the capability to restore their own state when they are recycled. It can use data store like SQL database for maintain instance state. It is new feature in .Net 3.5

When Durable service is created with database as data store, it will maintain all its state information in the table.

## Throttling

WCF throttling provides some properties that you can use to limit how many instances or sessions are created at the application level. Performance of the WCF service can be improved by creating proper instance.



## Request-Reply

By default all WCF will operated in the Request-Replay mode. It means that, when client make a request to the WCF service and client will wait to get response from service (till *receiveTimeout*). After getting the response it will start executing the rest of the statement. If service doesn't respond to the service within receiveTimeout, client will receive *TimeOutException*.

## One-Way

In One-Way operation mode, client will send a request to the server and does not care whether it is success or failure of service execution. There is no return from the server side, it is one-way communication.

Client will be blocked only for a moment till it dispatches its call to service. If any exception thrown by service will not reach the server.

## Callback Service

Till now we have seen that the all clients will call the service to get the things done. But WCF also provides the service to call the client. In which, service will act as client and client will act as service.

* HTTP protocols are connectionless nature, so it is not supported for callback operation. So BasicHttpBinding and WSHttpBinding cannot be used for this operation.
* WCF support WSDualHttpBinding for call back operation.
* All TCP and IPC protocols support Duplex communication. So all these binding will be used for callback operation

## What is WCF RIA service?

WCF RIA service is a framework to develop n-tier application for Rich Internet Application (RIA). It is mainly used in RIA applications like Silverlight, AJAX client, etc. It solves the major problem while developing business application like decoupling the resource access, application logic and presentation layer. WCF RIA service was introduced in Silverlight 4 with .net framework 4, and it can be developed using visual studio2010.

Main problem developer are facing while developing the n-tier RIA application will be coordinating the application logic between middle tier and presentation tier. This problem will be solved by using WCF RIA service, it will synchronize the code between middle and presentation tier.

### Buffer transfer

When the client and the service exchange messages, these messages are buffered on the receiving end and delivered only once the entire message has been received. This is true whether it is the client sending a message to the service or the service returning a message to the client. As a result, when the client calls the service, the service is invoked only after the client's message has been received in its entirety; likewise, the client is unblocked only once the returned message with the results of the invocation has been received in its entirety.

### Stream transfer

When client and Service exchange message using Streaming transfer mode, receiver can start processing the message before it is completely delivered. Streamed transfers can improve the scalability of a service by eliminating the requirement for large memory buffers. If you want to transfer large message, streaming is the best method.