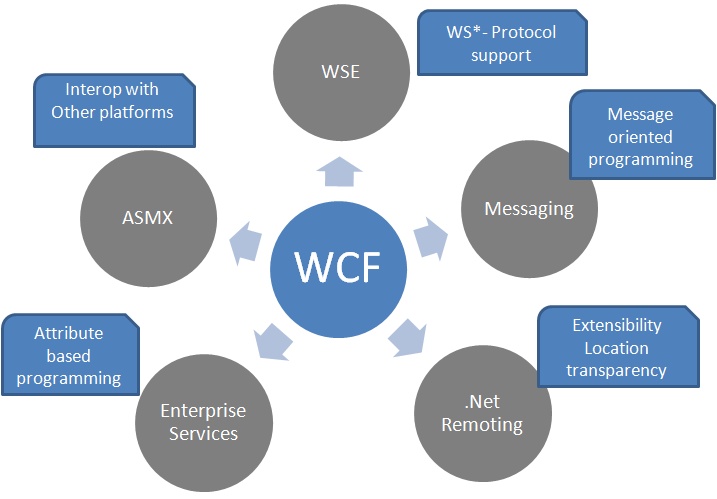
**Introduction to WCF** Windows Communication Foundation (WCF) is an SDK for developing and deploying services on Windows.

WCF provides a runtime environment for your services, enabling you to expose CLR types as services, and to consume other services as CLR types. Windows Communication Foundation (Code named Indigo) is a programming platform and runtime system for building, configuring and deploying network-distributed services. It is the latest service oriented technology; Interoperability is the fundamental characteristics of WCF.

It is unified programming model provided in .Net Framework 3.0. WCF is a combined feature of Web Service, Remoting, MSMQ and COM+. WCF provides a common platform for all .NET communication.

WCF functionality is included in a single assembly called System.ServiceModel.dll in the System.ServiceModel namespace

Below figures shows the different technology combined to form WCF.



**Advantage**

WCF is interoperable with other services when compared to .Net Remoting,where the client and service have to be .Net.

WCF services provide better reliability and security in compared to ASMX web services.

In WCF, there is no need to make much change in code for implementing the security model and changing the binding. Small changes in the configuration will make your requirements.

WCF has integrated logging mechanism, changing the configuration file settings will provide this functionality. In other technology developer has to write the code.

# Difference between WCF and Web service

Web service is a part of WCF. WCF offers much more flexibility and portability to develop a service when comparing to web service. Still we are having more advantages over Web service; following table provides detailed difference between them.

|  |  |  |
| --- | --- | --- |
| Features | Web Service | WCF |
| Hosting | It can be hosted in IIS | It can be hosted in IIS, windows activation service, Self-hosting, Windows service |
| Programming | [WebService] attribute has to be added to the class | [ServiceContraact] attribute has to be added to the class |
| Model | [WebMethod] attribute represents the method exposed to client | [OperationContract] attribute represents the method exposed to client |
| Operation | One-way, Request- Response are the different operations supported in web service | One-Way, Request-Response, Duplex are different type of operations supported in WCF |
| XML | System.Xml.serialization name space is used for serialization | System.Runtime.Serialization namespace is used for serialization |
| Encoding | XML 1.0, MTOM(Message Transmission Optimization Mechanism), DIME, Custom | XML 1.0, MTOM, Binary, Custom |
| Transports | Can be accessed through HTTP, TCP, Custom | Can be accessed through HTTP, TCP, Named pipes, MSMQ,P2P, Custom |
| Protocols | Security | Security, Reliable messaging, Transactions |

## FundaMental:

### Endpoint

Every service is associated with an address that defines where the service is, a binding that defines how to communicate with the service, and a contract that defines what the service does. This triumvirate governing the service is easy to remember as the ABC of the service. WCF formalizes this relationship in the form of an endpoint. The endpoint is the fusion of the address, contract, and binding.

WCF Service is a program that exposes a collection of Endpoints. Each Endpoint is a portal for communicating with the world.

All the WCF communications are take place through end point. End point consists of three components.

#### Address

Basically URL, specifies where this WCF service is hosted .Client will use this url to connect to the service.

In WCF, every service is associated with a unique address. The address provides two important elements:

1. the location of the service
2. Transport protocol or transport schema used to communicate with the service.

The location portion of the address indicates the name of the target machine, site, or network; a communication port, pipe, or queue; and an optional specific path or URI. A URI is a Universal Resource Identifier, and can be any unique string, such as the service name or a GUID.

WCF 1.0 supports the following transport schemas:

* HTTP
* TCP
* Peer network
* IPC (Inter-Process Communication over named pipes)
* MSMQ

Addresses always have the following format:

[base address]/[optional URI]

The base address is always in this format:

[transport]://[machine or domain][:optional port]

Here are a few sample addresses:

http://localhost:8001

http://localhost:8001/MyService

net.tcp://localhost:8002/MyService

net.pipe://localhost/MyPipe

net.msmq://localhost/private/MyService

net.msmq://localhost/MyService

The way to read an address such as

<http://localhost:8001>

<http://localhost:8090/MyService/SimpleCalculator.svc>

#### 1.3.2. HTTP Addresses

HTTP addresses use http for transport, and can also use https for secure transport. You typically use HTTP addresses with outward-facing Internet-based services, and can specify a port such as:

http://localhost:8001

When the port number is unspecified, it defaults to 80. Similar to TCP addresses, two HTTP addresses from the same host can share a port, even on the same machine.

HTTP-based addresses are also used throughout this book.

#### 1.3.3. IPC Addresses

IPC addresses use net.pipe for transport, to indicate the use of the Windows named pipe mechanism. In WCF, services that use named pipes can only accept calls from the same machine. Consequently, you must specify either the explicit local machine name or localhost for the machine name, followed by a unique string for the pipe name:

net.pipe://localhost/MyPipe

You can only open a named pipe once per machine, and therefore it is not possible for two named pipe addresses to share a pipe name on the same machine.

#### 1.3.4. MSMQ Addresses

MSMQ addresses use net.msmq for transport, to indicate the use of the Microsoft Message Queue (MSMQ). You must specify the queue name. When you're dealing with private queues, you must specify the queue type, but that can be omitted for public queues:

net.msmq://localhost/private/MyService

net.msmq://localhost/MyService

#### 1.3.5. Peer Network Address

Peer network addresses use net.p2p for transport, to indicate the use of the Windows peer network transport. You must specify the peer network name as well as a unique path and port.

### Binding

Binding will describes how client will communicate with service. There are different protocols available for the WCF to communicate to the Client. You can mention the protocol type based on your requirements.

Consider a scenario say, I am creating a service that has to be used by two type of client. One of the clients will access SOAP using http and other client will access Binary using TCP. How it can be done? With Web service it is very difficult to achieve, but in WCF it’s just we need to add extra endpoint in the .configuration file.

<system.serviceModel>

<services>

<service name="MathService"

behaviorConfiguration="MathServiceBehavior">

<endpoint address="http://localhost:8090/MyService/MathService.svc"

contract="IMathService"

binding="wsHttpBinding"/>

<endpoint address="net.tcp://localhost:8080/MyService/MathService.svc"

contract="IMathService"

binding="netTcpBinding"/>

</service>

</services>

<behaviors>

<serviceBehaviors>

<behavior name="MathServiceBehavior">

<serviceMetadata httpGetEnabled="True"/>

<serviceDebug includeExceptionDetailInFaults="true" />

</behavior>

</serviceBehaviors>

</behaviors>

</system.serviceModel>

A binding has several characteristics, including the following:

* Transport -Defines the service will be used like HTTP, Named Pipes, TCP, and MSMQ are some type of protocols.
* Encoding (Optional) - Three types of encoding are available-Text, Binary, or Message Transmission Optimization Mechanism (MTOM). MTOM is an interoperable message format that allows the effective transmission of attachments or large messages (greater than 64K).
* Protocol(Optional) - Defines information to be used in the binding such as Security, transaction or reliable messaging capability

The following table gives some list of protocols supported by WCF binding.

|  |  |
| --- | --- |
| Binding | Description |
| BasicHttpBinding | Basic Web service communication. No security by default |
| WSHttpBinding | Web services with WS-\* support. Supports transactions |
| WSDualHttpBinding | Web services with **duplex** contract and transaction support |
| WSFederationHttpBinding | Web services with federated security. Supports transactions |
| MsmqIntegrationBinding | Communication directly with MSMQ applications. Supports transactions |
| NetMsmqBinding | Communication between WCF applications by using queuing. Supports transactions |
| NetNamedPipeBinding | Communication between WCF applications on same computer. Supports **duplex** contracts and transactions |
| NetPeerTcpBinding | Communication between computers across peer-to-peer services. Supports **duplex** contracts |
| NetTcpBinding | Communication between WCF applications across computers. Supports **duplex** contracts and transactions |

#### Format and Encoding

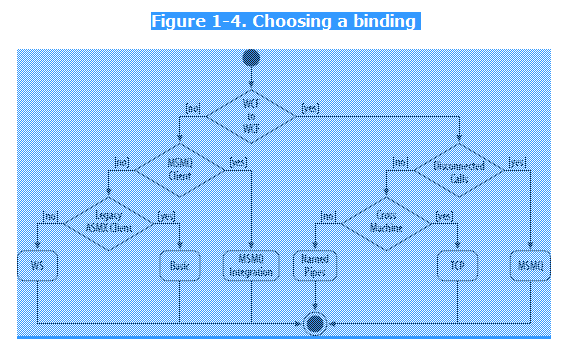
Each of the standard bindings uses different transport and encoding, as listed in [Table 1-1](mk:@MSITStore:C:\Users\kgupta\Desktop\www.free-ebooks-download.org----O'Reilly%20-%20Programming%20WCF%20Services.chm::/0596526997/orm0596526997-CHP-1-SECT-6.html#orm0596526997-CHP-1-TABLE-1).

| Table 1-1. Transport and encoding for standard bindings (default encoding is in bold) | | | |
| --- | --- | --- | --- |
| Name | Transport | Encoding | Interoperable |
| BasicHttpBinding | HTTP/HTTPS | Text, MTOM | Yes |
| NetTcpBinding | TCP | Binary | No |
| NetPeerTcpBinding | P2P | **Binary** | No |
| NetNamedPipeBinding | IPC | Binary | No |
| WSHttpBinding | HTTP/HTTPS | Text, MTOM | Yes |
| WSFederationHttpBinding | HTTP/HTTPS | Text, MTOM | Yes |
| WSDualHttpBinding | HTTP | Text, MTOM | Yes |
| NetMsmqBinding | MSMQ | Binary | No |
| MsmqIntegrationBinding | MSMQ | Binary | Yes |

Having a text-based encoding enables a WCF service (or client) to communicate over HTTP with any other service (or client) regardless of its technology. Binary encoding over TCP or IPC yields the best performance but at the expense of interoperability, by mandating WCF-to-WCF communication.

#### Choosing a Binding

Choosing a binding for your service should follow the decision-activity diagram shown in [Figure 1-4](mk:@MSITStore:C:\Users\kgupta\Desktop\www.free-ebooks-download.org----O'Reilly%20-%20Programming%20WCF%20Services.chm::/0596526997/orm0596526997-CHP-1-SECT-6.html#orm0596526997-CHP-1-FIG-4).



The first question you should ask yourself is whether your service needs to interact with non-WCF clients. If the answer is yes, and if the client is a legacy MSMQ client, choose the MsmqIntegrationBinding that enables your service to interoperate over MSMQ with such a client. If you need to interoperate with a non-WCF client and that client expects basic web service protocol (ASMX web services), choose the BasicHttpBinding, which exposes your WCF service to the outside world as if it were an ASMX web service (that is, a WSI-basic profile). The downside is that you cannot take advantage of most of the modern WS-\* protocols. However, if the non-WCF client can understand these standards, choose one of the WS bindings, such as WSHttpBinding, WSFederationBinding, or WSDualHttpBinding. If you can assume that the client is a WCF client, yet it requires offline or disconnected interaction, choose the NetMsmqBinding that uses MSMQ for transporting the messages. If the client requires connected communication, but could be calling across machine boundaries, choose the NetTcpBinding that communicates over TCP. If the client is on the same machine as the service, choose the NetNamedPipeBinding that uses named pipes to maximize performance. You may fine-tune binding selections based on additional criteria such as the need for callbacks (WSDualHttpBinding) or federated security (WSFederationBinding).

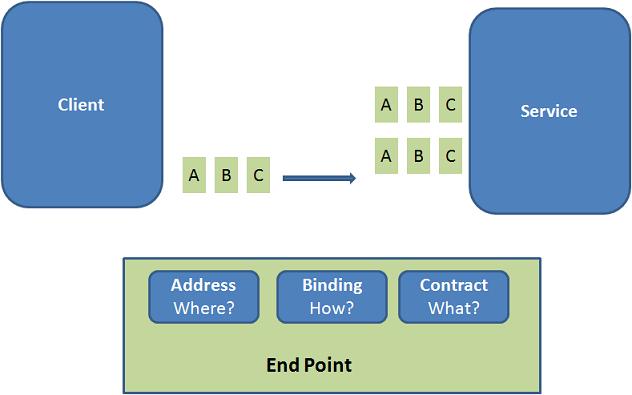
|  |  |  |
| --- | --- | --- |
| |  |  | | --- | --- | |  | Most bindings work well even outside their target scenario. For example, you could use the TCP binding for same-machine or even in-proc communication, and you could use the basic binding for Intranet WCF-to-WCF communication. However, do try to choose a binding according to [Figure 1-4](mk:@MSITStore:C:\Users\kgupta\Desktop\www.free-ebooks-download.org----O'Reilly%20-%20Programming%20WCF%20Services.chm::/0596526997/orm0596526997-CHP-1-SECT-6.html#orm0596526997-CHP-1-FIG-4). | |

### Contract

Collection of operation that specifies what the endpoint will communicate with outside world. Usually name of the Interface will be mentioned in the Contract, so the client application will be aware of the operations which are exposed to the client. Each operation is a simple exchange pattern such as one-way, duplex and request/reply.

In WCF, all services expose contracts. The contract is a platform-neutral and standard way of describing what the service does. WCF defines four types of contracts.

Below figure illustrate the functions of Endpoint



#### Example:

Endpoints will be mentioned in the web.config file on the created service.

<system.serviceModel>

<services>

<service name="MathService"

behaviorConfiguration="MathServiceBehavior">

<endpoint

**address="http://localhost:8090/MyService/MathService.svc" contract="IMathService"**

**binding="wsHttpBinding"/>**

</service>

</services>

<behaviors>

<serviceBehaviors>

<behavior name="MathServiceBehavior">

<serviceMetadata httpGetEnabled="True"/>

<serviceDebug includeExceptionDetailInFaults="true" />

</behavior>

</serviceBehaviors>

</behaviors>

</system.serviceModel>

Mainly there are four types of contracts available in WCF

### Service Contract

### Describe which operations the client can perform on the service.

### Service contracts describe the operation that service can provide. For Eg, a Service provide to know the temperature of the city based on the zip code, this service is called as Service contract. It will be created using Service and Operational Contract attribute.

### Data Contract

Data contract describes the custom data type which is exposed to the client. This defines the data types, that are passed to and from service. Data types like int, string are identified by the client because it is already mention in XML schema definition language document, but custom created class or data types cannot be identified by the client e.g. Employee data type. By using DataContract we can make client to be aware of Employee data type that are returning or passing parameter to the method.

### Message Contract

### Allow the service to interact directly with messages. Message contracts can be typed or untyped, and are useful in interoperability cases and when there is an existing message format you have to comply with. As a WCF developer, you should use message contracts only rarely

Default SOAP message format is provided by the WCF runtime for communication between Client and service. If it is not meeting your requirements then we can create our own message format. This can be achieved by using Message Contract attribute.

### 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 occurred in the service to client. This helps us to easy identity, what error has occurred.

Define which errors are raised by the service, and how the service handles and propagates errors to its clients

### Message and Channel

### Message

WCF Message is the unit of data exchange between client and service. It consists of several parts, including a body and headers.

### WCF Runtime

WCF runtime is the set of object responsible for sending and receiving message. For example formatting the message, applying security and transmitting and receiving message using various protocol.

### Channels:

Channels are the core abstraction for sending message to and receiving message from an Endpoint. Broadly we can categories channels as

### Transport Channels

- Handles sending and receiving message from network. Protocols like HTTP, TCP name pipes and MSMQ.

### Protocol Channels

- Implements SOAP based protocol by processing and possibly modifying message. e.g. WS-Security and WS-Reliability

# WCF Client and Metadata

### WCF Client

WCF client is a client application creates to expose the service operations as method. Any application can host a WCF client, including an application that host a service. Therefore it is possible to create a service that includes WCF clients of other services.

A client application is a managed application that uses a WCF client to communicate with another application. To create a client application for a WCF service requires the following steps:

1. Get the Proxy class and service end point information

Using *SvcUtil.exe* we can create proxy class for the service and configuration information for endpoints. Example type the following sentence in the Visual studio command prompt, this will generate the class file and configuration file which contain information about the endpoints.

*svcutil /language:vb /out:ClientCode.vb /config:app.config* *http://localhost:8090/MyService/SimpleCalculator.svc?wsdl*

1. Call operations.

Add this class files in the client application. Then create the object for this class and invoke the service operation. Configuration information we got from the above step has to be added to the client application configuration file. When the client application calls the first operation, WCF automatically opens the underlying channel. This underlying channel is closed, when the object is recycled.

//Creating the proxy on client side

MyCalculatorServiceProxy.MyServiceProxy proxy

= new MyCalculatorServiceProxy.MyServiceProxy();

Console.WriteLine("Counter: " + proxy.MyMethod());

1. Close the WCF client object.

After using the object created in the above steps, we have to dispose the object. Channel will be closed with the service, when the object is cleared.

### Metadata

Characteristics of the service are described by the metadata. This metadata can be exposed to the client to understand the communication with service. Metadata can be set in the service by enabling the ServiceMetadata node inside the servcieBehaviour node of the service configuration file.

<system.serviceModel>

<services>

<service name="MathService"

behaviorConfiguration="MathServiceBehavior">

<endpoint address="" contract="IMathService"

binding="wsHttpBinding"/>

</service>

</services>

<behaviors>

<serviceBehaviors>

<behavior name="MathServiceBehavior">

**<serviceMetadata httpGetEnabled="True"/>**

<serviceDebug includeExceptionDetailInFaults="true" />

</behavior>

</serviceBehaviors>

</behaviors>

</system.serviceModel>

This metadata can be viewed while creating WCF client application using SvcUtil.exe

# WCF Architecture

The following figure illustrates the major components of WCF.

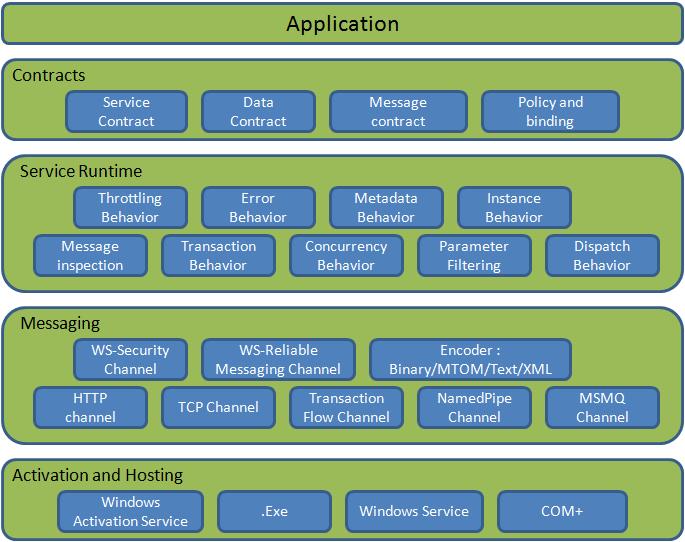


Figure 1: WCF Architecture

### Contracts

Contracts layer are next to that of Application layer. Developer will directly use this contract to develop the service. We are also going to do the same now. Let us see briefly what these contracts will do for us and we will also know that WCF is working on message system.

#### Service contracts

- Describe about the operation that service can provide. Example, Service provided to know the temperature of the city based on the zip code, this service we call as Service contract. It will be created using Service and Operational Contract attribute.

#### Data contract

- It describes the custom data type which is exposed to the client. This defines the data types, are passed to and from service. Data types like int, string are identified by the client because it is already mention in XML schema definition language document, but custom created class or datatype cannot be identified by the client e.g. Employee data type. By using DataContract we can make client aware that we are using Employee data type for returning or passing parameter to the method.

#### Message Contract

- Default SOAP message format is provided by the WCF runtime for communication between Client and service. If it is not meeting your requirements then we can create our own message format. This can be achieved by using Message Contract attribute.

#### Policies and Binding

- Specify conditions required to communicate with a service e.g security requirement to communicate with service, protocol and encoding used for binding.

### Service Runtime

- It contains the behaviors that occur during runtime of service.

* Throttling Behavior- Controls how many messages are processed.
* Error Behavior - Specifies what occurs, when internal error occurs on the service.
* Metadata Behavior - Tells how and whether metadata is available to outside world.
* Instance Behavior - Specifies how many instance of the service has to be created while running.
* Transaction Behavior - Enables the rollback of transacted operations if a failure occurs.
* Dispatch Behavior - Controls how a message is processed by the WCF Infrastructure.

### Messaging

- Messaging layer is composed of channels. A channel is a component that processes a message in some way, for example, by authenticating a message. A set of channels is also known as a channel stack. Channels are the core abstraction for sending message to and receiving message from an Endpoint. Broadly we can categories channels as

Transport Channels

Handles sending and receiving message from network. Protocols like HTTP, TCP, name pipes and MSMQ.

Protocol Channels

Implements SOAP based protocol by processing and possibly modifying message. E.g. WS-Security and WS-Reliability.

### Activation and Hosting

- Services can be hosted or executed, so that it will be available to everyone accessing from the client. WCF service can be hosted by following mechanism

* IIS

Internet information Service provides number of advantages if a Service uses Http as protocol. It does not require Host code to activate the service, it automatically activates service code.

* Windows Activation Service

(WAS) is the new process activation mechanism that ships with IIS 7.0. In addition to HTTP based communication, WCF can also use WAS to provide message-based activation over other protocols, such as TCP and named pipes.

* Self-Hosting

WCF service can be self hosted as console application, Win Forms or WPF application with graphical UI.

* Windows Service

WCF can also be hosted as a Windows Service, so that it is under control of the Service Control Manager (SCM).

# WCF Hosting

WCF service class cannot exist in a void. Every WCF service must be hosted in a Windows process called the host process. A single host process can host multiple services, and the same service type can be hosted in multiple host processes. WCF makes no demand on whether or not the host process is also the client process.

As we discussed there are mainly four different way of hosting the WCF service.

1. [IIS hosting](http://www.wcftutorial.net/WCF-IIS-Hosting.aspx)
2. [Self hosting](http://www.wcftutorial.net/WCF-Self-Hosting.aspx)
3. [Windows Activation Service](http://www.wcftutorial.net/WCF-WAS-Hosting.aspx)
4. [Windows Service](http://www.wcftutorial.net/WCF-Windows-Service-Hosting.aspx)

Microsoft has introduced the WCF concept in order to make distributed application development and deployment simple.

|  |  |
| --- | --- |
| **Hosting Environment** | **Supported protocol** |
| Windows console and form application | HTTP,net.tcp,net.pipe,net.msmq |
| Windows service application (formerly known as NT services) | HTTP,net.tcp,net.pipe,net.msmq |
| Web server IIS6 | http, wshttp |
| Web server IIS7 - Windows Process Activation Service (WAS) | HTTP,net.tcp,net.pipe,net.msmq |

A summary of hosting options and supported features.

|  |  |  |  |
| --- | --- | --- | --- |
| **Feature** | **Self-Hosting** | **IIS Hosting** | **WAS Hosting** |
| Executable Process/ App Domain | Yes | Yes | Yes |
| Configuration | App.config | Web.config | Web.config |
| Activation | Manual at startup | Message-based | Message-based |
| Idle-Time Management | No | Yes | Yes |
| Health Monitoring | No | Yes | Yes |
| Process Recycling | No | Yes | Yes |
| Management Tools | No | Yes | Yes |

Types of Binding

Let us see more detailed on predefined binding

BasicHttpBinding

* It is suitable for communicating with ASP.NET Web services (ASMX)-based services that comfort with WS-Basic Profile conformant Web services.
* This binding uses HTTP as the transport and text/XML as the default message encoding.
* Security is disabled by default
* This binding does not support WS-\* functionalities like WS- Addressing, WS-Security, WS-ReliableMessaging
* It is fairly weak on interoperability.

WSHttpBinding

* Defines a secure, reliable, interoperable binding suitable for non-duplex service contracts.
* It offers lot more functionality in the area of interoperability.
* It supports WS-\* functionality and distributed transactions with reliable and secure sessions using SOAP security.
* It uses HTTP and HTTPS transport for communication.
* Reliable sessions are disabled by default.

WSDualHttpBinding

This binding is same as that of WSHttpBinding, except it supports duplex service. Duplex service is a service which uses duplex message pattern, which allows service to communicate with client via callback.

In WSDualHttpBinding reliable sessions are enabled by default. It also supports communication via SOAP intermediaries.

WSFederationHttpBinding

This binding support federated security. It helps implementing federation which is the ability to flow and share identities across multiple enterprises or trust domains for authentication and authorization. It supports WS-Federation protocol.

NetTcpBinding

This binding provides secure and reliable binding environment for .Net to .Net cross machine communication. By default it creates communication stack using WS-ReliableMessaging protocol for reliability, TCP for message delivery and windows security for message and authentication at run time. It uses TCP protocol and provides support for security, transaction and reliability.

NetNamedPipeBinding

This binding provides secure and reliable binding environment for on-machine cross process communication. It uses NamedPipe protocol and provides full support for SOAP security, transaction and reliability. By default it creates communication stack with WS-ReliableMessaging for reliability, transport security for transfer security, named pipes for message delivery and binary encoding.

NetMsmqBinding

* This binding provides secure and reliable queued communication for cross-machine environment.
* Queuing is provided by using MSMQ as transport.
* It enables for disconnected operations, failure isolation and load leveling

NetPeerTcpBinding

* This binding provides secure binding for peer-to-peer environment and network applications.
* It uses TCP protocol for communication
* It provides full support for SOAP security, transaction and reliability.

# HTTP\_GET Enabled Metadata

We will use ServiceBehaviour to publish the metadata using HTTP-GET. This can be configures either administratively or Programmatically. Http and Https can expose by appending "?wsdl" to the end of the service address. For example service address is

*http://localhost:9090/MyCalulatorService* , HTTP-Get metadata address is given by *http://localhost:9090/MyCalulatorService?wsdl*.

### Administrative (Configuration file):

In the below mention configuration information, you can find the behavior section in the ServiceBehavior. You can expose the metadata using ServiceMetadata node with httpGetEnable='True'.

<system.serviceModel>

<services>

<service behaviorConfiguration="ServiceBehavior" name="MyService">

<endpoint address="http://localhost/IISHostedService/MyService.svc"

binding="wsHttpBinding" contract="IMyService">

<identity>

<dns value="localhost"/>

</identity>

</endpoint>

</service>

</services>

<behaviors>

<serviceBehaviors>

<behavior name="ServiceBehavior">

<!-Setting httpGetEnabled you can publish the metadata -->

**<serviceMetadata httpGetEnabled="true"/>**

</behavior>

</serviceBehaviors>

</behaviors>

</system.serviceModel>

### Progarmming Model:

Using ServiceMetadataBehavior you can enable the metadata exchange. In the following code, I have created the ServiceMetadataBehavior object and assigned HttpGetEnabled property to true. Then you have to add the behavior to host description as shown. This set of code will publish the metadata using HTTP-GET.

//Create a URI to serve as the base address

Uri httpUrl = new Uri("http://localhost:8090/MyService/SimpleCalculator");

//Create ServiceHost

ServiceHost host = new

ServiceHost(typeof(MyCalculatorService.SimpleCalculator), httpUrl);

//Add a service endpoint

host.AddServiceEndpoint

(typeof(MyCalculatorService.ISimpleCalculator), new WSHttpBinding(), "");

//Enable metadata exchange

ServiceMetadataBehavior smb = new ServiceMetadataBehavior();

//Enable metadata exchange using HTTP-GET

smb.HttpGetEnabled = true;

host.Description.Behaviors.Add(smb);

//Start the Service

host.Open();

Console.WriteLine("Service is host at " + DateTime.Now.ToString());

Console.WriteLine("Host is running... Press key to stop");

Console.ReadLine();

# Metadata Exchange Endpoint

Exposing the metadata using HTTP-GET has a disadvantage, such that there is no guarantee that other platforms you interact will support it. There is other way of exposing the using special endpoint is called as Metadata Exchange Endpoint. You can have as many metadata exchange endpoints as you want.

#### Address

It is basically Uri to identify the metadata. You can specify as address in the endpoint but append with "mex" keyword. For example "http://localhost:9090/MyCalulatorService/mex"

#### Binding

There are four types of bindings supported for metadata exchange. They are mexHttpBinding, mexHttpsBinding, mexNamedPipesBinding, mexTcpBinding.

#### Contract

IMetadataExchange is the contract used for MEX endpoint. WCF service host automatically provides the implementation for this IMetadataExcahnge while hosting the service.

You can create the Metadata Exchange Endpoint either Administrative (configuration file) or programmatically.

#### Administrative (Configuration file):

In the configuration file of the hosting application, you can add metadata exchange endpoint as shown below.

<system.serviceModel>

<services>

<service name="MyService">

<endpoint address="http://localhost/IISHostedService/MyService.svc"

binding="wsHttpBinding" contract="IMyService">

<identity>

<dns value="localhost"/>

</identity>

</endpoint>

<endpoint address="mex" binding="mexHttpBinding" contract="IMetadataExchange"/>

</service>

</services>

</system.serviceModel>

### Programming Model:

In the following code I have mention about creating the Metadata Exchange Endpoint through coding. Steps to create the metadata endpoint are

* Create the *ServiceMetadataBehavior* object and add to Service host description.
* ServiceMetadataBehavior smb = new ServiceMetadataBehavior();

host.Description.Behaviors.Add(smb);

* Create the metadata binding object using *MetadataExchangeBinding*

Binding mexBinding = MetadataExchangeBindings.CreateMexHttpBinding ();

* 3. Add the endpoint to the service host with address, binding and contract.

host.AddServiceEndpoint(typeof(IMetadataExchange), mexBinding, "mex");

Complete code for hosting the service with metadata exchange endpoint is shown below.

//Create a URI to serve as the base address

Uri httpUrl = new Uri("http://localhost:8090/MyService/SimpleCalculator");

//Create ServiceHost

ServiceHost host = new

ServiceHost(typeof(MyCalculatorService.SimpleCalculator), httpUrl);

//Add a service endpoint

host.AddServiceEndpoint

(typeof(MyCalculatorService.ISimpleCalculator), new WSHttpBinding(), "");

//Enable metadata exchange

ServiceMetadataBehavior smb = new ServiceMetadataBehavior();

host.Description.Behaviors.Add(smb);

Binding mexBinding = MetadataExchangeBindings.CreateMexHttpBinding ();

//Adding metadata exchange endpoint

host.AddServiceEndpoint(typeof(IMetadataExchange), mexBinding, "mex");

//Start the Service

host.Open();

Console.WriteLine("Service is host at " + DateTime.Now.ToString());

Console.WriteLine("Host is running... Press key to stop");

Console.ReadLine();

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;

}

}

Now you have some fundamental idea on Service contract. Next we will look into Data Contract.

**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; }

}

[DataMember]

public int Age

{

get { return m\_Age; }

set { m\_Age = value; }

}

[DataMember]

public int Salary

{

get { return m\_Salary; }

set { m\_Salary = value; }

}

[DataMember]

public string Designation

{

get { return m\_Designation; }

set { m\_Designation = value; }

}

[DataMember]

public string Manager

{

get { return m\_Manager; }

set { m\_Manager = 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.

Diagram Soap envelope

**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

1. When using Message contract type as parameter, Only one parameter can be used in servicie Operation
2. [OperationContract]
3. void SaveEmployeeDetails(EmployeeDetails emp);
4. Service operation either should return Messagecontract type or it should not return any value
5. [OperationContract]
6. EmployeeDetails GetEmployeeDetails();
7. Service operation will accept and return only message contract type. Other data types are not allowed.
8. [OperationContract]
9. EmployeeDetails ModifyEmployeeDetails(EmployeeDetails emp);

**Note: If a type has both Message and Data contract, service operation will accept only message contract.**

# 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 some 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 went 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();

}

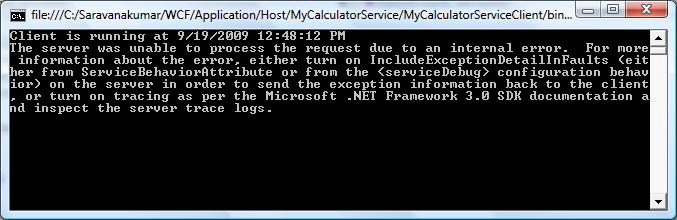
catch (Exception ex)

{

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)

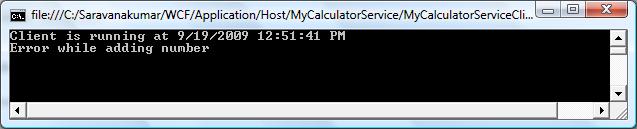
{

//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

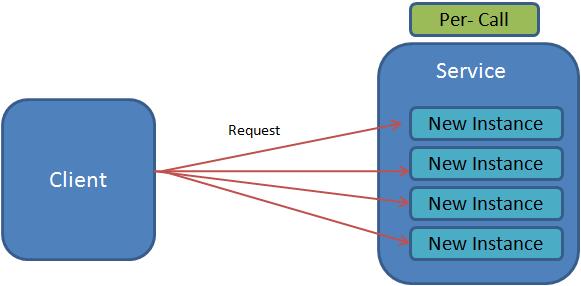
}

# Instance Management

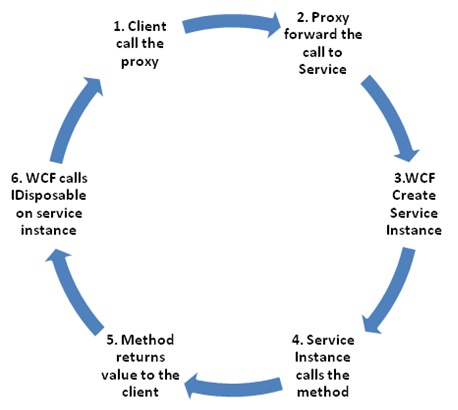
Instance management refers to the way a service handles a request from a client. Instance management is set of techniques WCF uses to bind client request to service instance, governing which service instance handles which client request. It is necessary because application will differ in their need for scalability, performance, durability, transaction and queued calls.

## Per-Call Service

When WCF service is configured for Per-Call instance mode, Service instance will be created for each client request. This Service instance will be disposed after response is sent back to client.



Following diagram represent the process of handling the request from client using Per-Call instance mode.



Let as understand the per-call instance mode using example.

**Step 1:** Create the service contract called IMyService and implement the interface. Add service behavior attribute to the service class and set the *InstanceContextMode* property to *PerCall* as show below.

[ServiceContract()]

public interface IMyService

{

[OperationContract]

int MyMethod();

}

**Step 2:** In this implementation of MyMethod operation, increment the static variable(m\_Counter). Each time while making call to the service, m\_Counter variable is incremented and return the value to the client.

[ServiceBehavior(InstanceContextMode=InstanceContextMode.PerCall)]

public class MyService:IMyService

{

static int m\_Counter = 0;

public int MyMethod()

{

m\_Counter++;

return m\_Counter;

}

}

**Step 3:** Client side, create the proxy for the service and call "myMethod" operation multiple time.

static void Main(string[] args)

{

Console.WriteLine("Service Instance mode: Per-Call");

Console.WriteLine("Client making call to service...");

//Creating the proxy on client side

MyCalculatorServiceProxy.MyServiceProxy proxy =

new MyCalculatorServiceProxy.MyServiceProxy();

Console.WriteLine("Counter: " + proxy.MyMethod());

Console.WriteLine("Counter: " + proxy.MyMethod());

Console.WriteLine("Counter: " + proxy.MyMethod());

Console.WriteLine("Counter: " + proxy.MyMethod());

Console.ReadLine();

}

Surprisingly, all requests to service return '1', because we configured the Instance mode to Per-Call. Service instance will created for each request and value of static variable will be set to one. While return back, service instance will be disposed. Output is shown below.

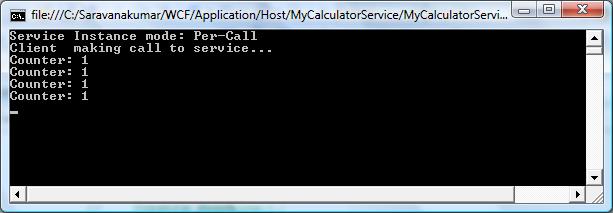
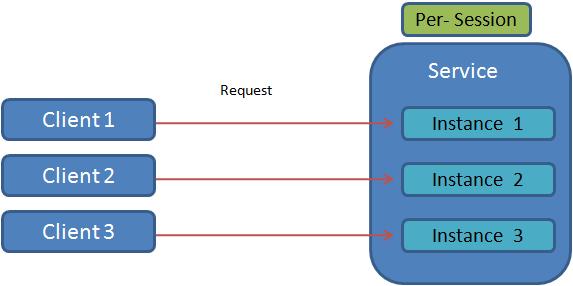


Fig: PercallOutput.

## Per-Session Service

When WCF service is configured for Per-Session instance mode, logical session between client and service will be maintained. When the client creates new proxy to particular service instance, a dedicated service instance will be provided to the client. It is independent of all other instance.

Following diagram represent the process of handling the request from client using Per-Session instance mode.



Let as understand the Per-Session instance mode using example.

**Step 1:** Create the service contract called IMyService and implement the interface. Add service behavior attribute to the service class and set the *InstanceContextMode* property to *PerSession* as show below.

[ServiceContract()]

public interface IMyService

{

[OperationContract]

int MyMethod();

}

**Step 2:** In this implementation of MyMethod operation, increment the static variable (m\_Counter). Each time while making call to the service, m\_Counter variable will be incremented and return the value to the client.

[ServiceBehavior(InstanceContextMode=InstanceContextMode.PerSession)]

public class MyService:IMyService

{

static int m\_Counter = 0;

public int MyMethod()

{

m\_Counter++;

return m\_Counter;

}

}

**Step 3:** Client side, create the proxy for the service and call "myMethod" operation multiple time.

static void Main(string[] args)

{

Console.WriteLine("Service Instance mode: Per-Session");

Console.WriteLine("Client making call to service...");

//Creating the proxy on client side

MyCalculatorServiceProxy.MyServiceProxy proxy =

new MyCalculatorServiceProxy.MyServiceProxy();

Console.WriteLine("Counter: " + proxy.MyMethod());

Console.WriteLine("Counter: " + proxy.MyMethod());

Console.WriteLine("Counter: " + proxy.MyMethod());

Console.WriteLine("Counter: " + proxy.MyMethod());

Console.ReadLine();

}

All request to service return incremented value (1, 2, 3, 4), because we configured the instance mode to Per-Session. Service instance will be created once the proxy is created at client side. So each time request is made to the service, static variable is incremented. So each call to MyMethod return incremented value. Output is shown below.

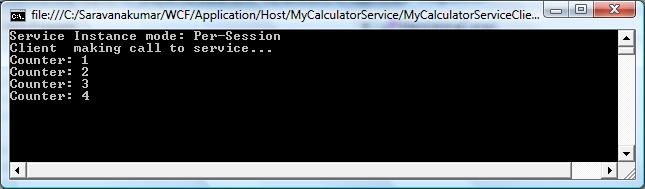
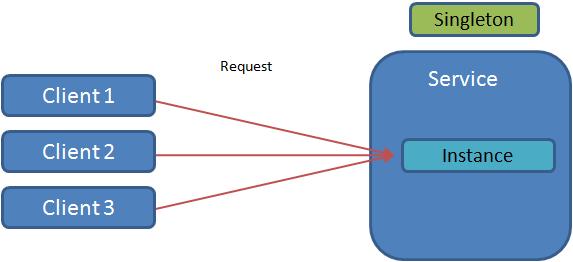


Fig: PersessionOutput.

## Singleton Service

When WCF service is configured for Singleton instance mode, all clients are independently connected to the same single instance. This singleton instance will be created when service is hosted and, it is disposed when host shuts down.

Following diagram represent the process of handling the request from client using Singleton instance mode.



Let as understand the Singleton Instance mode using example.

**Step 1:** Create the service contract called IMyService and implement the interface. Add service behavior attribute to the service class and set the *InstanceContextMode* property to *Single* as show below.

[ServiceContract()]

public interface IMyService

{

[OperationContract]

int MyMethod();

}

**Step 2:** In this implementation of MyMethod operation, increment the static variable(m\_Counter). Each time while making call to the service, m\_Counter variable is incremented and return the value to the client

[ServiceBehavior(InstanceContextMode=InstanceContextMode.Single)]

public class MyService:IMyService

{

static int m\_Counter = 0;

public int MyMethod()

{

m\_Counter++;

return m\_Counter;

}

}

**Step 3:** Client side, create the two proxies for the service and made a multiple call to MyMethod.

static void Main(string[] args)

{

Console.WriteLine("Service Instance mode: Singleton");

Console.WriteLine("Client 1 making call to service...");

//Creating the proxy on client side

MyCalculatorServiceProxy.MyServiceProxy proxy =

new MyCalculatorServiceProxy.MyServiceProxy();

Console.WriteLine("Counter: " + proxy.MyMethod());

Console.WriteLine("Counter: " + proxy.MyMethod());

Console.WriteLine("Counter: " + proxy.MyMethod());

Console.WriteLine("Client 2 making call to service...");

//Creating new proxy to act as new client

MyCalculatorServiceProxy.MyServiceProxy proxy2 =

new MyCalculatorServiceProxy.MyServiceProxy();

Console.WriteLine("Counter: " + proxy2.MyMethod());

Console.WriteLine("Counter: " + proxy2.MyMethod());

Console.ReadLine();

}

When two proxy class made a request to service, single instance at service will handle it and it return incremented value (1, 2, 3, 4), because instance mode is configured to 'Single'. Service instance is created when it is hosted. So this instance will remain till host is shutdown. Output is shown below.

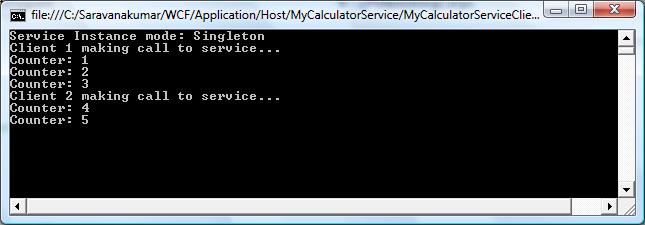


Fig: SingletonOutput.

# 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.

|  |  |
| --- | --- |
| **Attribute** | **Description** |
| maxConcurrentCalls | Limits the total number of calls that can currently be in progress across all service instances. The default is 16. |
| maxConcurrentInstances | The number of InstanceContext objects that execute at one time across a ServiceHost. The default is Int32.MaxValue. |
| maxConcurrentSessions | A positive integer that limits the number of sessions a ServiceHost object can accept. The default is 10. |

Service Throttling can be configured either Adminstractive or Programatically

### One-Way Operations and Exceptions

Suppose when we are using *BasicHttpBinding* or *WSHttpBinding*, i.e. no transport session is used, if any exception throw by service will not affect the client. Client can make a call to the service using same proxy

[ServiceContract]

interface IMyContract

{

[OperationContract(IsOneWay = true)]

void MethodWithError( );

[OperationContract]

void MethodWithoutError( );

}

//Client side without transport session

MyContractClient proxy = new MyContractClient( );

proxy.MethodWithError( ); //No exception is thrown from serivce

proxy.MethodWithoutError( ); //Operation will execute properly

proxy.Close( );

In the presence of transport session, any exception thrown by service will fault the client channel. Client will not be able to make new call using same proxy instance.

//Client side transport session

MyContractClient proxy = new MyContractClient( );

proxy.MethodWithError( );

proxy.MethodWithoutError( ); //Can not executre because channel is faulted

proxy.Close( );

### Administrative(configuration file)

Using <serviceThrottling> tag of the Service Behavior, you can configure the *maxConcurrentCalls*, *maxConcurrentInstances* , *maxConcurrentSessions* property as shown below.

<system.serviceModel>

<services >

<service behaviorConfiguration="ServiceBehavior" name="MyService">

<endpoint address="" binding="wsHttpBinding" contract="IMyService">

<identity>

<dns value="localhost"/>

</identity>

</endpoint>

<endpoint address="mex" binding="mexHttpBinding" contract="IMetadataExchange"/>

</service>

</services>

<behaviors>

<serviceBehaviors>

<behavior name="ServiceBehavior">

<serviceMetadata httpGetEnabled="true"/>

<serviceDebug includeExceptionDetailInFaults="true "/>

<serviceThrottling maxConcurrentCalls="500"

maxConcurrentInstances ="100"

maxConcurrentSessions ="200"/>

</behavior>

</serviceBehaviors>

</behaviors>

</system.serviceModel>

### Programming Model

Use ServiceThrottlingBehavior object to set concurrent calls, session and instance property.

ServiceHost host = new ServiceHost(typeof(MyService));

ServiceThrottlingBehavior throttle

= host.Description.Behaviors.Find();

if (throttle == null)

{

throttle = new ServiceThrottlingBehavior();

throttle.MaxConcurrentCalls = 500;

throttle.MaxConcurrentSessions = 200;

throttle.MaxConcurrentInstances = 100;

host.Description.Behaviors.Add(throttle);

}

host.Open();

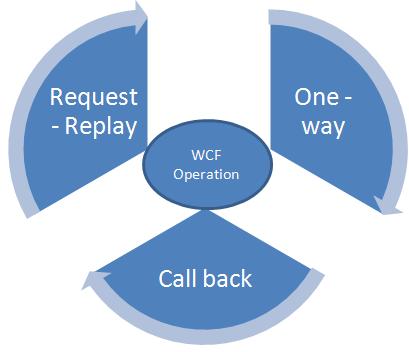
**Operations**

In classic object or component- oriented programming model offered only single way for client to call a method. Client will issue a call, block while the call was in progress, and continue executing once the method returned.

WCF will support classical Request-Replay model, along with that it also supports One-Way call(call and forget operation) and callback(service to call back the client)

Three modes of communication between client and service are

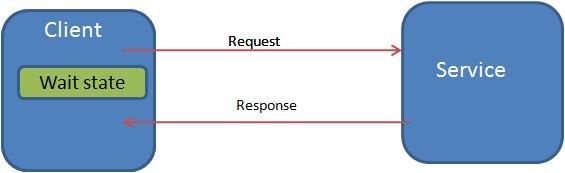
1. [Request- Replay](http://www.wcftutorial.net/Request-Reply.aspx)
2. [One-Way](http://www.wcftutorial.net/One-Way.aspx)
3. [Callback](http://www.wcftutorial.net/Callback-Service.aspx)



# 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*.

Apart from *NetPeerTcpBinding* and the *NetMsmqBinding* all other bindings will support request-reply operations.



# 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.

Client can continue to execute its statement, after making one-way call to server. There is no need to wait, till server execute. Sometime when one-way calls reach the service, they may not be dispatched all at once but may instead be queued up on the service side to be dispatched one at a time, according to the service's configured concurrency mode behavior. If the number of queued messages has exceeded the queue's capacity, the client will be blocked even if it's issued a one-way call. However, once the call is queued, the client will be unblocked and can continue executing, while the service processes the operation in the background.



### Definition :

One-way operation can be enabled by setting IsOneWay property to true in Operation contract attribute.

[ServiceContract]

public interface IMyService

{

[OperationContract(IsOneWay=true)]

void MyMethod(EmployeeDetails emp);

}

### One-Way Operations and Sessionful Services

Let us see the example, what will happen when you use the one-way communication with Sessionful service.

[ServiceContract(SessionMode = SessionMode.Required)]

interface IMyContract

{

[OperationContract(IsOneWay = true)]

void MyMethod();

}

As per above configuration, when client makes one-way call using MyMethod() operation and if it close the proxy. Client will be blocked until operation completes. It will be good practice, that one-way operation should be applied on per-call and singleton service.

Suppose If you want to make use of One-way operation in Sessionful service, use in the last operation of the service which will terminate the session. This operation should not return any value.

[ServiceContract(SessionMode = SessionMode.Required)]

interface IMyContract

{

[OperationContract]

void MyMethod1();

[OperationContract]

string MyMethod2();

[OperationContract(IsOneWay = true, IsInitiating = false,

IsTerminating = true)]

string CloseSessionService(int id);

}

**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.

**Defining and configuring a callback contract**

Callback service can be enabled by using *CallbackContract* property in the *ServiceContract* attribute. In the below example you can find the decalration of the callback contract and it is configured in the *ServiceContract* attribute.

public interface IMyContractCallback

{

[OperationContract]

void OnCallback();

}

[ServiceContract(CallbackContract = typeof(IMyContractCallback))]

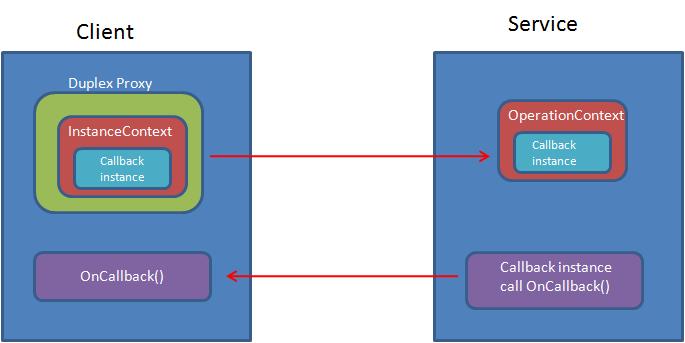
public interface IMyContract

{

[OperationContract()]

void MyMethod();

}



**Client Callback Setup**

As I said earlier, in callback operation client will act as service and service will act as client. So client has to expose a callback endpoint to the service to call. In the earlier part of the tutorial I have mention that InstanceContext is the execution scope of inner most service instance. It provides a constructor that takes the service instance to the host.

IMyContractCallback callback=new MyCallback();

InstanceContext cntx=new InstanceContext(callback);

MyServiceClient proxy = new MyServiceClient(cntx);

proxy.MyMethod();

The client must use a proxy that will set up the bidirectional communication and pass the callback endpoint reference to the service. This can be achieved by creating the proxy using *DuplexClientBase*

class MyServiceClient:DuplexClientBase,IMyContract

{

public MyServiceClient(InstanceContext callbackCntx)

: base(callbackCntx)

{

}

public void MyMethod()

{

base.Channel.MyMethod();

}

}

**Service-Side Callback Invocation**

The client-side callback endpoint reference is passed along with every call the client makes to the service, and it is part of the incoming message. The OperationContext class provides the service with easy access to the callback reference via the generic method *GetCallbackChannel*<T>( ). Service can call the client side callback method using reference e to the client side callback instance. The following code shows the callback method invocation.

IMyContractCallback

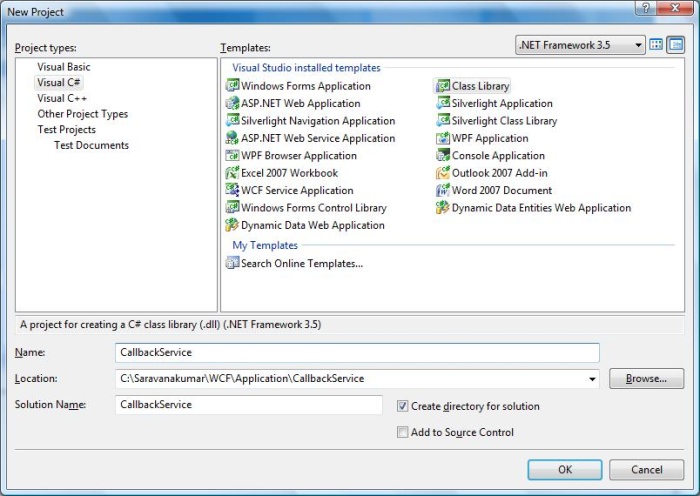
callbackInstance=OperationContext.Current.GetCallbackChannel();

callbackInstance.OnCallback();

# How to Create Callback Service in WCF

This tutorial gives hands-on to create a sample Callback service.

**Step 1:** Create the sample Classlibrary project using Visual Studio 2008 and name it as CallbackService



**Step 2 :** Add *System.ServiceModel* reference to the project

**Step 3:** Create the Callback and Service contract as shown below. You need to mention *CallbackContract* property in the *ServiceContract* attribute. Implementation of the Callback contract will be done on the client side.

**IMyContract.cs**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.ServiceModel;

namespace CallbackService

{

public interface IMyContractCallback

{

[OperationContract]

void OnCallback();

}

[ServiceContract(CallbackContract = typeof(IMyContractCallback))]

public interface IMyContract

{

[OperationContract()]

void MyMethod();

}

}

**Step 4:** Implement the Service contract as shown below. In the below code you will find using *OperationContext* is used to receive the reference to Callback instance. Using that instance we are calling the *OnCallback()* method from client side.

**MyService.cs**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.ServiceModel;

namespace CallbackService

{

[ServiceBehavior(ConcurrencyMode=ConcurrencyMode.Multiple )]

public class MyService:IMyContract

{

public void MyMethod()

{

//Do something

IMyContractCallback callbackInstance

=OperationContext.Current.GetCallbackChannel();

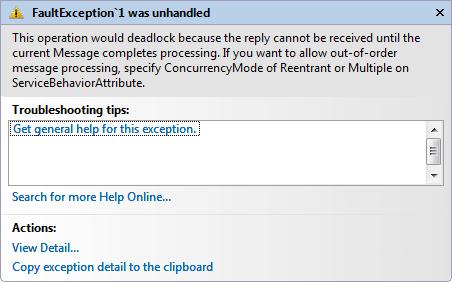
callbackInstance.OnCallback();

}

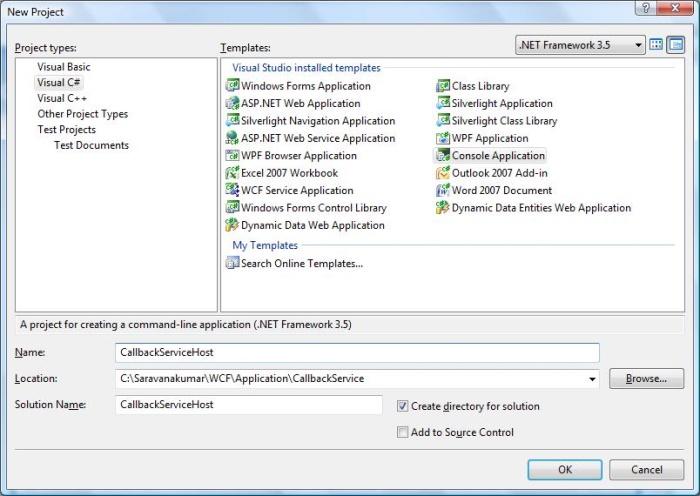
}

}

You can also note that We have set the *ConcurrencyMode* to Multile. If you are not using ConcurrencyMode to Multiple or Reentent, you will be end up with deadlock exception as shown below. This is because when a client made a call to the service, channel is created and lock by WCF service. If you are calling the Callback method inside the service method. Service will try to access the lock channel, this may leads to deadlock. So you can set *ConcurrencyMode* to *Multiple* or *Reentent* so it will release the lock silently.



**Step 5:** Create a Console application using Visual Studio 2008 and name it a CallbackServiceHost. This application is used to self-host the WCF service



**Step 6:** Main method

static void Main(string[] args)

{

Uri httpUrl = new Uri("http://localhost:8090/MyService/");

ServiceHost host = new ServiceHost(typeof(CallbackService.MyService), httpUrl);

host.Open();

Console.WriteLine("Service is Hosted at {0}", DateTime.Now.ToString());

Console.WriteLine("Host is running...Press key to stop the service.");

Console.ReadLine();

host.Close();

}

**Step 7:** Use Duplex binding to support Callback operation.

**Web.Config**

<system.serviceModel>

<services >

<service behaviorConfiguration="ServiceBehavior"

name="CallbackService.MyService">

<endpoint address="http://localhost:8090/MyService"

binding="wsDualHttpBinding" contract="CallbackService.IMyContract">

<identity>

<dns value="localhost"/>

</identity>

</endpoint>

<endpoint address="mex"

binding="mexHttpBinding" contract="IMetadataExchange"/>

</service>

</services>

<behaviors>

<serviceBehaviors>

<behavior name="ServiceBehavior">

<serviceMetadata httpGetEnabled="true"/>

<serviceDebug includeExceptionDetailInFaults="true "/>

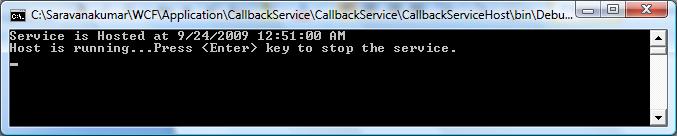
</behavior>

</serviceBehaviors>

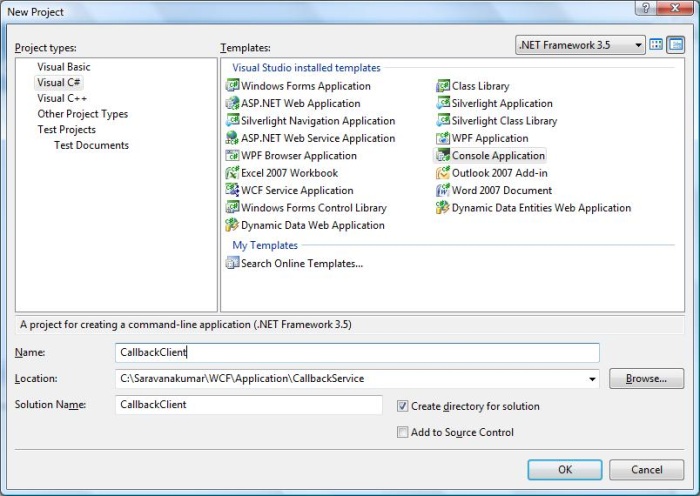
</behaviors>

</system.serviceModel>

**Step 8:** Run the host application



**Step 9:** Create Console Application using Visual Studio 2008 and name it as CallbackClient. This is the client application which contain Callback implementation.



**Step10:** Add *System.ServiceModel* and *CallbackService* as reference to the project

**Step 11:** Create the proxy class as shown below. Use *DuplexClientBase* to create the proxy, because it will support bidirectional communication. Create the contractor which will accept *InstanceContext* as parameter.

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.ServiceModel;

using CallbackService;

namespace CallbackClient

{

class MyServiceClient:DuplexClientBase<IMyContract>,IMyContract

{

public MyServiceClient(InstanceContext callbackCntx)

: base(callbackCntx)

{

}

public void MyMethod()

{

base.Channel.MyMethod();

}

}

}

**Step12:** Create the implementation for Callback Contract

class MyCallback : IMyContractCallback

{

public void OnCallback()

{

Console.WriteLine("Callback method is called from client side.");

}

}

**Step 13:** Implementation of main method

static void Main(string[] args)

{

IMyContractCallback callback=new MyCallback();

InstanceContext cntx=new InstanceContext(callback);

MyServiceClient proxy = new MyServiceClient(cntx);

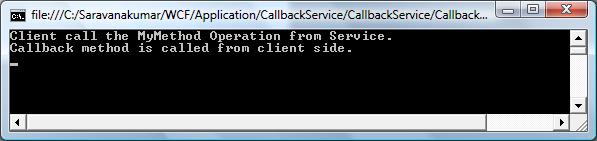
Console.WriteLine("Client call the MyMethod Operation from Service.");

proxy.MyMethod();

Console.ReadLine();

}

**Step14:** Run the client application. In the output, you can see the OnCallback method called by the service



**Introdution to WCF 4.0**

This article explains about the new features introduced in WCF 4.0.

.Net framework comes with new features and improved areas of WCF. It was mainly focused on simplifying the developer experience, enabling more communication scenario and providing rich integration with WWF.

The following items specifies the new features of WCF 4.0

**Simplified configuration**

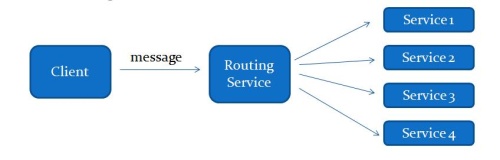
This new feature shows simplification of WCF configuration section by providing default endpoint, binding and behavior configuration. It is not mandatory to provide endpoint while hosting service. Service will automatically create new endpoint if it does find any endpoint while hosting service. These changes make it possible to host configuration-free services.

**Discovery service**

There are certain scenario in which endpoint address of the service will be keep on changing. In that kind of scenario, client who consume this service also need to change the endpoint address dynamically to identify the service. This can be achieved using WS-Discovery protocol.

**Routing service**

This new feature introduces routing service between client and actual business service. This intermediated service Act as broker or gateways to the actual business services and provides features for content based routing, protocol bridging and error handling



**REST Service**

There are few features helps while developing RESTful service.

* Automatic help page that describes REST services to consumer
* Support for declarative HTTP catching

**Workflow service**

* Improves development experience
* Entire service definition can be define in XAML
* Hosting workflow service can be done from .xamlx file, without using .svc file
* Introduce new “Context” bindings like BasicHttpContextBinding, WSHttpContextBinding, or NetTcpContextBinding
* In .Net4.0, *WorkflowServiceHost* class for hosting workflow services was redesigned and it is available in *System.ServiceModel.Activities* assembly. In .Net3.5, *WorkflowServiceHost* class is available in *System.WorkflowServices* assembly
* New messaging activities SendReply and ReceiveReply are added in .Net4.0

**Conclusion:**

This article explain new featues introduced in WCF 4.0

**Transaction**

A transaction is a collection or group of one or more units of operation executed as a whole. It provides way to logically group single piece of work and execute them as a single unit. In addition, WCF allows client applications to create transactions and to propagate transactions across service boundaries.

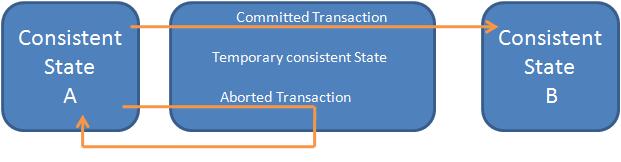
**Recovery Challenge**

Let us discuss more on challenge we will phased and how to recover from it.

1. Consider a system maintained in consistent state, when application fail to perform particular operation, you should recover from it and place the system in the consistent state.
2. While doing singe operation, there will be multiple atomic sub operation will happen. These operations might success or fail. We are not considering about sub operation which are failed. We mainly consider about the success operation. Because we have to recover all these state to its previous consistence state.
3. Productivity penalty has to be payee for all effort required for handcrafting the recovery logic
4. Performance will be decreased because you need to execute huge amount of code.

**Solution**

Best way to maintain system consistence and handling error-recovery challenge is to use transactions. Below figure gives idea about transaction.



* **Committed transaction:** Transaction that execute successfully and transfer the system from consistence state A to B.
* **Aborted transaction:** Transaction encounters an error and rollback to Consistence State A from intermediate state.
* **In-doubt transaction:** Transactions fail to either in commit or abort.

**Transaction Resources**

Transactional programming requires working with a resource that is capable of participating in a transaction, and being able to commit or roll back the changes made during the transaction. Such resources have been around in one form or another for decades. Traditionally, you had to inform a resource that you would like to perform transactional work against it. This act is called enlisting. Some resources support auto-enlisting.

**Transaction Properties**

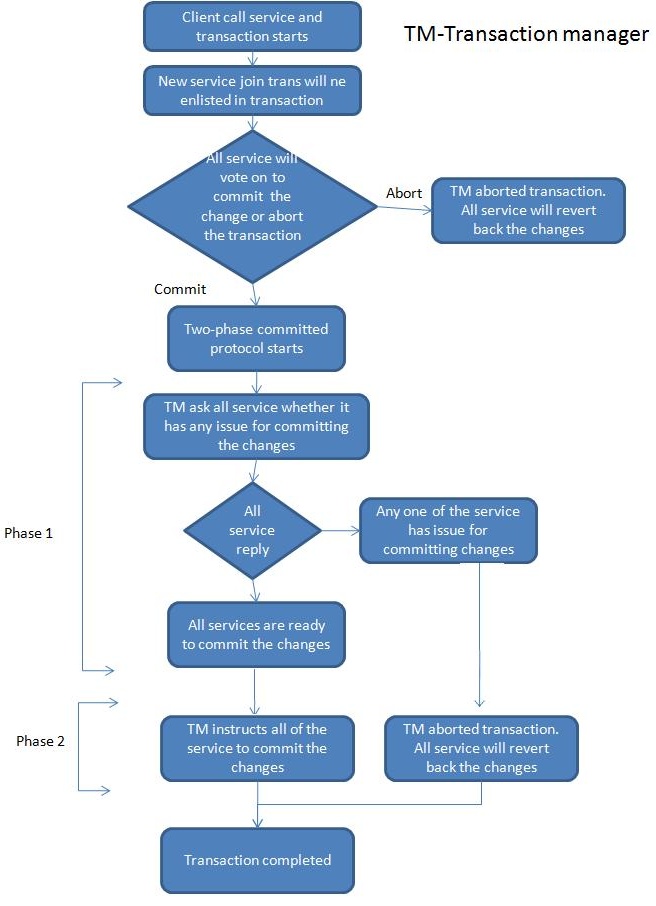
Transaction can be said as pure and successful only if meets four characteristics.

* Atomic - When transaction completes, all the individual changes made to the resource while process must be made as to they were all one atomic, indivisible operation.
* Consistent - transaction must leave the system in consistent state.
* Isolated - Resources participating in the transaction should be locked and it should not be access by other third party.
* Durable - Durable transactions must survive failures.

# Two-phase committed protocol

Consider the scenario where I am having single client which use single service for communication and interacting with single database. In which service starts and manage the transaction, now it will be easy for the service to manage the transaction.

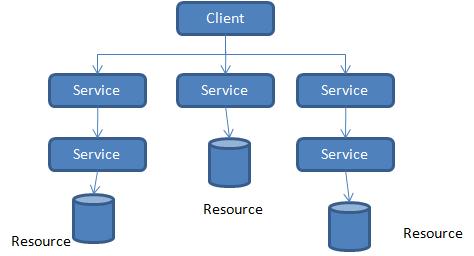
Consider for example client calling multiple service or service itself calling another service, this type of system are called as Distributed Service-oriented application. Now the questions arise that which service will begin the transaction? Which service will take responsibility of committing the transaction? How would one service know what the rest of the service feels about the transaction? Service could also be deployed in different machine and site. Any network failure or machine crash also increases the complexity for managing the transaction.



In order to overcome these situations, WCF come up with distributed transaction using two way committed protocol and dedicated transaction manager.

Transaction Manager is the third party for the service that will manage the transaction using two phase committed protocol.

Let us see how Transaction manager will manage the transaction using two-phase committed protocols.



# Transaction Propagation

In WCF, transaction can be propagated across service boundary. This enables service to participate in a client transaction and it includes multiple services in same transaction, Client itself will act as service or client.

We can specify whether or not client transaction is propagated to service by changing Binding and operational contract configuration

<bindings>

<netTcpBinding>

<binding transactionFlow="true"></binding>

</netTcpBinding>

</bindings>

Even after enabling transaction flow does not mean that the service wants to use the client’s transaction in every operation. We need to specify the “TransactionFlowAttribute” in operational contract to enable transaction flow.

[ServiceContract]

public interface IService

{

[OperationContract]

[TransactionFlow(TransactionFlowOption.Allowed)]

int Add(int a, int b);

[OperationContract]

int Subtract(int a, int b);

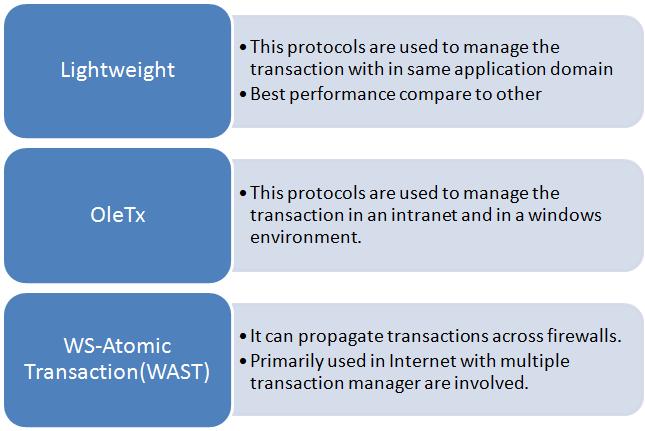
}

**Note:** TransactionFlow can be enabled only at the operation level not at the service level.

|  |  |  |
| --- | --- | --- |
| **TransactionFlowOption** | **Binding configuration** |  |
| NotAllowed | transactionFlow="true"  or transactionFlow="false" | Client cannot propagate its transaction to service even client has transaction |
| Allowed | transactionFlow="true" | Service will allow to flow client transaction. It is not necessary that service to use client transaction. |
| Allowed | transactionFlow="false" | If service disallows at binding level, client also should disable at binding level else error will be occurred. |
| Mandatory | transactionFlow="true" | Both Service and client must use transaction aware binding |
| Mandatory | transactionFlow="false" | InvalidOperationException will be throw when serice binding disables at binding level.  FaultException will be thrown when client disable at its binding level. |

# Transaction Protocols

As a developer we no need to concern about transaction protocols and transaction manager used by WCF. WCF itself will take care of what kind of transaction protocols should be used for different situation. Basically there are three different kinds of transaction protocols used by WCF.

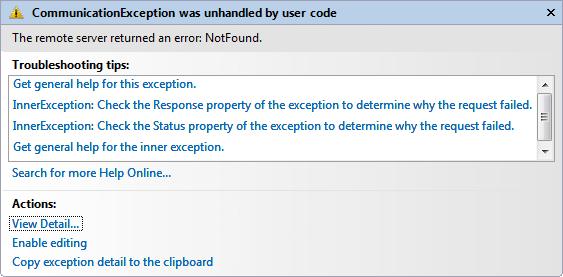


# Handling Exception in Silverlight application from WCF

### Attachment:

[Silverlight\_WCF\_Exception.zip](http://www.wcftutorial.net/Attachments/Silverlight_WCF_Exception.zip)

This article explains about handling the exception in Silverlight application from WCF. I have created the sample Silverlight application, which uses the WCF service for process the data. While testing the application I came to know that exception message thrown from WCF cannot be received at the client side(Silverlight application) even after using the FaultException. I was always getting **System.ServiceModel.CommunicationException:** The remote server returned an error: NotFound.



Later I came to know that WCF throws the HTTP 500 series Fault message but Silverlight can handle only 200 series. So we need to convert the 500 series to 200 error message for Silverlight. Here is the sample application for exception handling between WCF and Silverlight.

**Step 1:** We can customize the Endpoint behavior of the WCF service by inheriting the *Beha and implementing the IEndpointBehavior. Actual code for converting the 500 error serice to 200 serivce in BeforeSendReply method.*

*Create a ClassLibrary project and name it as “****Silverlight\_WCF\_FaultBehavior****” and name the class as “SilverlightFaultBehavior”. Copy and paste the follwing code inside the SilverlightFaultBehavior class.*

*Imports System.ServiceModel.ConfigurationrviceModel.Configuration*

*Imports System.ServiceModel.Description*

*Imports System.ServiceModel.Dispatcher*

*Imports System.ServiceModel.Channels*

*Imports System.ServiceModel*

*Public Class SilverlightFaultBehavior*

*Inherits BehaviorExtensionElement*

*Implements IEndpointBehavior*

*Public Overrides ReadOnly Property BehaviorType() As System.Type*

*Get*

*Return GetType(SilverlightFaultBehavior)*

*End Get*

*End Property*

*Protected Overrides Function CreateBehavior() As Object*

*Return New SilverlightFaultBehavior*

*End Function*

*Public Sub AddBindingParameters(ByVal endpoint As ServiceEndpoint,*

*ByVal bindingParameters As BindingParameterCollection)*

*Implements IEndpointBehavior.AddBindingParameters*

*End Sub*

*Public Sub ApplyClientBehavior(ByVal endpoint As ServiceEndpoint,*

*ByVal clientRuntime As ClientRuntime)*

*Implements IEndpointBehavior.ApplyClientBehavior*

*End Sub*

*Public Sub ApplyDispatchBehavior(ByVal endpoint As ServiceEndpoint,*

*ByVal endpointDispatcher As EndpointDispatcher)*

*Implements IEndpointBehavior.ApplyDispatchBehavior*

*Dim inspector As New SilverlightFaultMessageInspector()*

*endpointDispatcher.DispatchRuntime.MessageInspectors.Add(inspector)*

*End Sub*

*Public Sub Validate(ByVal endpoint As ServiceEndpoint)*

*Implements IEndpointBehavior.Validate*

*End Sub*

*Public Class SilverlightFaultMessageInspector*

*Implements IDispatchMessageInspector*

*Public Function AfterReceiveRequest(ByRef request As Message,*

*ByVal channel As IClientChannel,*

*ByVal instanceContext As InstanceContext) As Object*

*Implements IDispatchMessageInspector.AfterReceiveRequest*

*' Do nothing to the incoming message.*

*Return Nothing*

*End Function*

*Public Sub BeforeSendReply(ByRef reply As System.ServiceModel.Channels.Message,*

*ByVal correlationState As Object)*

*Implements IDispatchMessageInspector.BeforeSendReply*

*If reply.IsFault Then*

*Dim [property] As New HttpResponseMessageProperty()*

*' Here the response code is changed to 200.*

*[property].StatusCode = System.Net.HttpStatusCode.OK*

*reply.Properties(HttpResponseMessageProperty.Name) = [property]*

*End If*

*End Sub*

*End Class*

*End Class*

***Note:*** *Highlighted code shows the conversion for 500 serices to 200 series error code.*

***Step 2:*** *Build the project*

***Step 3:*** *Create a new WCF service with Interface and implementation class as follows*

***Interface***

*<ServiceContract()> \_*

*Public Interface IService*

*<OperationContract()> \_*

*Function Add(ByVal num1 As Integer, ByVal num2 As Integer) As Integer*

*<OperationContract()> \_*

*Function Subtract(ByVal num1 As Integer, ByVal num2 As Integer) As Integer*

*End Interface*

***Implementation***

*Public Class Service*

*Implements IService*

*Public Sub New()*

*End Sub*

*Public Function Add(ByVal num1 As Integer, ByVal num2 As Integer)*

*As Integer Implements IService.Add*

*Throw New FaultException("Error thrown by user for Add operation")*

*'Return num1 + num2*

*End Function*

*Public Function Subtract(ByVal num1 As Integer,*

*ByVal num2 As Integer) As Integer Implements IService.Subtract*

*Return num1 - num2*

*End Function*

*End Class*

*< Add the* ***Silverlight\_WCF\_FaultBehavior*** *project dll as reference to WCF Service*

***Step 5:***

***Step 5: In WCF we can extend the binding and behavior by using <extention> tag. In our case also we are extending the custom endpoint behavior as shown below. In the <behaviorExtensions> tag we need specify the fully qualified name of the cutom behaviour assembly.***

***Modify the Web.config file as shown bellow***

***<system.serviceModel>***

***<services>***

***<service name="Service" behaviorConfiguration="ServiceBehavior">***

***<!-- Service Endpoints -->***

***<endpoint address="" binding="basicHttpBinding" contract="IService"***

***behaviorConfiguration="SilverlightFaultBehavior">***

***<!--***

***Upon deployment, the following identity element should be removed or replaced***

***to reflect the identity under which the deployed service runs. If removed,***

***WCF will infer an appropriate identity automatically.-->***

***<identity>***

***<dns value="localhost"/>***

***</identity>***

***</endpoint>***

***<endpoint address="mex" binding="mexHttpBinding" contract="IMetadataExchange"/>***

***</service>***

***</services>***

***<behaviors>***

***<serviceBehaviors>***

***<behavior name="ServiceBehavior">***

***<!-- To avoid disclosing metadata information, set the value below to false and***

***remove the metadata endpoint above before deployment -->***

***<serviceMetadata httpGetEnabled="true"/>***

***<!-- To receive exception details in faults for debugging purposes, set the value***

***below to true. Set to false before deployment to avoid disclosing exception***

***information -->***

***<serviceDebug includeExceptionDetailInFaults="false"/>***

***</behavior>***

***</serviceBehaviors>***

***<endpointBehaviors>***

***<behavior name="SilverlightFaultBehavior">***

***<silverlightFaults/>***

***</behavior>***

***</endpointBehaviors>***

***</behaviorss=“BlueCode”>>***

***<extensions>***

***<behaviorExtensions>***

***<add name="silverlightFaults"***

***type="Silverlight\_WCF\_FaultBehavior.SilverlightFaultBehavior,***

***Silverlight\_WCF\_FaultBehavior, Version=1.0.0.0, Culture=neutral,***

***PublicKeyToken=null"/>***

***</behaviorExtensions>***

***</extensionss=“BlueCode”>>***

***</system.serviceModel>***

***Step 6: Create the any sample silverlight application as “Silverlight\_WCF\_Exception” and add this WCF service as Service Reference.***

***url: http://localhost/MathService/Service.svc***

***Step 7: Add a button to the MainPage.xaml and call the WCF method as shown below***

***Private Sub Button\_Click(ByVal sender As System.Object,***

***ByVal e As System.Windows.RoutedEventArgs)***

***Dim proxy As New ServiceProxy.ServiceClient***

***AddHandler proxy.AddCompleted, AddressOf AddOperationCompleted***

***proxy.AddAsync(5, 6)***

***End Sub***

***Private Sub AddOperationCompleted(ByVal sender As Object,***

***ByVal e As ServiceProxy.AddCompletedEventArgs)***

***If e.Error IsNot Nothing Then***

***MessageBox.Show(e.Error.Message)***

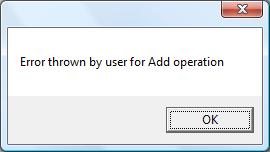
***Else***

***MessageBox.Show(e.Result)***

***End If***

***End Sub***

***Step 8: Output will look like this***

******

***Conclution: This article explains about handling the exception in Silverlight application from WCF***

# Transfer mode

In our normal day today life, we need to transfer data from one location to other location. If data transfer is taking place through WCF service, message size will play major role in performance of the data transfer. Based on the size and other condition of the data transfer, WCF supports two modes for transferring messages

### 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.

### StreamRequest

In this mode of configuration, message send from client to service will be streamed

### StreamRespone

In this mode of configuration, message send from service to client will be streamed.

### Configuration

<system.serviceModel>

<services >

<service behaviorConfiguration="ServiceBehavior" name="MyService">

<endpoint address="" binding="netTcpBinding"

bindingConfiguration="MyService.netTcpBinding" contract="IMyService">

<identity>

<dns value="localhost"/>

</identity>

</endpoint>

<endpoint address="mex" binding="mexHttpBinding"

contract="IMetadataExchange"/>

</service>

</services>

<behaviors>

<serviceBehaviors>

<behavior name="ServiceBehavior">

<serviceMetadata httpGetEnabled="true"/>

<serviceDebug includeExceptionDetailInFaults="true "/>

</behavior>

</serviceBehaviors>

</behaviors>

<bindings >

<netTcpBinding>

<binding name="MyService.netTcpBinding"

transferMode="Buffered" closeTimeout ="0:01:00" openTimeout="0:01:00"></binding>

</netTcpBinding>

</bindings>

</system.serviceModel>

### Differences between Buffered and Streamed Transfers

|  |  |
| --- | --- |
| **Buffered** | **Streamed** |
| Target can process the message once it is completely received. | Target can start processing the data when it is partially received |
| Performance will be good when message size is small | Performance will be good when message size is larger(more than 64K) |
| Native channel shape is IDuplexSessionChannel | Native channels are IRequestChannel and IReplyChannel |

**Streaming**

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.

**Supported Bindings**

* BasicHttpBinding
* NetTcpBinding
* NetNamedPipeBinding

**Restrictions**

There are some restriction, when streaming is enabled in WCF

* Digital signatures for the message body cannot be performed
* Encryption depends on digital signatures to verify that the data has been reconstructed correctly.
* Reliable sessions must buffer sent messages on the client for redelivery if a message gets lost in transfer and must hold messages on the service before handing them to the service implementation to preserve message order in case messages are received out-of-sequence.
* Streaming is not available with the Message Queuing (MSMQ) transport
* Streaming is also not available when using the Peer Channel transport

**I/O Streams**

WCF uses .Net stream class for Streaming the message. Stream in base class for streaming, all subclasses like FileStream,MemoryStream, NetworkStream are derived from it. Stream the data, you need to do is, to return or receive a Stream as an operation parameter.

[ServiceContract]

public interface IMyService

{

[OperationContract]

void SaveStreamData(Stream emp);

[OperationContract]

Stream GetStreamData();

}

**Note:**

1. Stream and it's subclass can be used for streaming, but it should be serializable
2. Stream and MemoryStream are serializable and it will support streaming
3. FileStream is non serializable, and it will not support streaming

**Streaming and Binding**

Only the TCP, IPC, and basic HTTP bindings support streaming. With all of these bindings streaming is disabled by default. *TransferMode* property should be set according to the desired streaming mode in the bindings.

public enum TransferMode

{

Buffered, //Default

Streamed,

StreamedRequest,

StreamedResponse

}

public class BasicHttpBinding : Binding,...

{

public TransferMode TransferMode

{get;set;}

//More members

}

* **StreamedRequest** - Send and accept requests in streaming mode, and accept and return responses in buffered mode
* **StreamResponse** - Send and accept requests in buffered mode, and accept and return responses in streamed mode
* **Streamed** - Send and receive requests and responses in streamed mode in both directions
* **Buffered** -Send and receive requests and responses in Buffered mode in both directions

**Streaming and Transport**

The main aim of the Streaming transfer mode is to transfer large size data, but default message size is 64K. So you can increase the message size using *maxReceivedMessageSize* attribute in the binding element as shown below.

<system.serviceModel>

<bindings >

<netTcpBinding>

<binding name="MyService.netTcpBinding"

transferMode="Buffered" maxReceivedMessageSize="1024000">

</binding>

</netTcpBinding>

</bindings>

</system.serviceModel>