WCF(INDIGO) -> Windows Communication Foundation

- It is used for building service-oriented , distributed and interoperable applications.
- **Service Oriented applications:** Applications that are largely composed of services.
- **Distributed application or Connected applications:** Application whose parts runs in 2 or more computer nodes.
- **Interoperable Applications:** An application that can communicate with any other application that is built on any platform.
- **NOTE:** a) Web services are interoperable.
  b) .net remoting services are not interoperable.
- **WCF** is a tool mainly used to implement and deploy a service oriented architecture.
- **Service-Oriented Architecture (SOA)** is a style of software design where services are provided to the other components by application components, through a communication protocol over a network.In this architecture, number of services communicate with each other, in one of two ways: through passing data or through two or more services coordinating an activity.
- To create a wcf service select wcf service from add menu.
- We use system.servicemodel
- To make a contract available publicly we use HTTP protocol else we use TCP protocol.

Serialization in WCF

- Serialization is the process of converting an object into an XML representation.
- The reverse process, that is reconstructing the same object from the XML is called as Deserialization.
- By default, **WCF** uses **DataContractSerializer.**
- For a complex type like class to be serialized, the complex type can either be decorated with
  1. SerializableAttribute or
  2. DataContractAttribute
- We don't have to explicitly use **DataContract or DataMember** attributes. The **Data Contract Serializer** will serialize all public properties of your complex type in an alphabetical order. By default private field and properties are not serialized.
- We can serialize data members in 2 ways

1) Using **Seriailzable**
2) Using **Data Contract**

- if we decorate a complex type, with **[Serializable]** attribute the **DataContractSerializer** serializes all fields. With **[Serializable]** attribute we don't have explicit control on what fields to include and exclude in serialized data.
- If we decorate a complex type with **[Datacontract]** attribute, the **DataContractSerializer** serializes the fields marked with

the **[DataMember]** attribute. The fields that are not marked with **[DataMember]** attribute are excluded from serialization.

- The **[DataMember]** attribute can be applied either on the **private fields** or **public properties.**
- In WCF, the most common way of serialization is to mark the type with the **DataContract** attribute and mark each member that needs to be serialized with the **DataMember** attribute.
- If you want to have explicit control on what fields and properties get serialized then use DataContract and DataMember attributes.
  1. Using DataContractAttribute, you can define an XML namespace for your data
  2. Using DataMemberAttribute, you can
    a) Define Name, Order, and whether if a property or field IsRequired
    b) Also, serialize private fields and properties.

WCF components	**Description**
Service Contract	Used for defining a wcf service
Operation Contract	Used for exposing a method to the client
Data Contact	used for data members or properties for serialization

Known Type Attributes

- If we have classes related by inheritance, the wcf service generally accepts and returns the base type. If you expect the service to accept and return inherited types, then use KnownType attribute.
- **There are 4 different ways to associate KnownTypes**

1. Use **KnownType** attribute on the base type. This option is global, that is all service contracts and all operation contracts will respect the known types.

```
[KnownType(typeof(FullTimeEmployee))]
[KnownType(typeof(PartTimeEmployee))]
[DataContract]
public class Employee
{
}

[DataContract]
public class FullTimeEmployee : Employee
{
   public int AnnualSalary { get; set; }
}

[DataContract]
public class PartTimeEmployee : Employee
{
   public int HourlyPay { get; set; }
   public int HoursWorked { get; set; }
}
```

2. Apply **ServiceKnownType** attribute on the service contract. With this option the known types are respected by all operation contracts with in this service contract only.

```
[ServiceKnownType(typeof(PartTimeEmployee))]
[ServiceKnownType(typeof(FullTimeEmployee))]
[ServiceContract]
public interface IEmployeeService
{
    [OperationContract]
    Employee GetEmployee(int Id);

    [OperationContract]
    void SaveEmployee(Employee Employee);
}
```

3. If you want even more granular control, then apply **ServiceKnownType** attribute on specific operation contracts. With this option, only the operation contracts that are decorated with ServiceKnownType attribute respect known types.

```
[ServiceContract]
public interface IEmployeeService
{
    [ServiceKnownType(typeof(PartTimeEmployee))]
    [ServiceKnownType(typeof(FullTimeEmployee))]
    [OperationContract]
    Employee GetEmployee(int Id);

    [OperationContract]
    void SaveEmployee(Employee Employee);
}
```

4. You can also specify known types in the configuration file. This is equivalent to applying **KnownType** attribute on the base type, in the sense that it is applicable globally. All service contracts and operation contracts respect the known types.

```
<system.runtime.serialization>
  <dataContractSerializer>
    <declaredTypes>
      <add type="EmployeeService.Employee, EmployeeService, Version=1.0.0.0,
          Culture=Neutral, PublicKeyToken=null">
        <knownType type="EmployeeService.FullTimeEmployee, EmployeeService,
                Version=1.0.0.0, Culture=Neutral, PublicKeyToken=null"/>
        <knownType type="EmployeeService.PartTimeEmployee, EmployeeService,
                Version=1.0.0.0, Culture=Neutral, PublicKeyToken=null"/>
      </add>
    </declaredTypes>
  </dataContractSerializer>
</system.runtime.serialization>
```

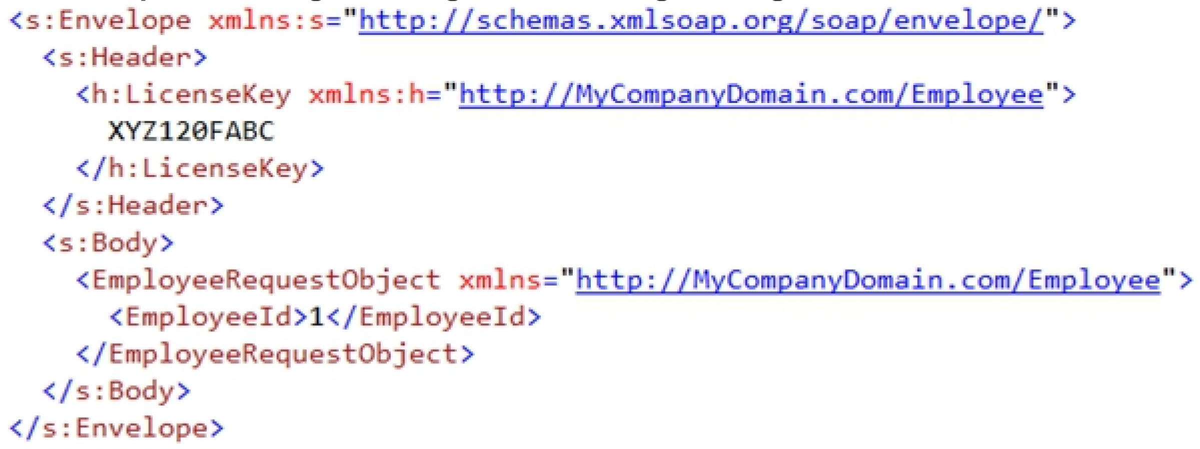
Message Contract (mainly used for easy logging and tracing)

- Use **Message Contracts,** if you want to have full control over the generated XML SOAP messages.
- **When to use Message Contract:**

1. Include some custom data in the SOAP header. In general SOAP headers are used to pass user credentials, license keys, session keys etc.
2. Change the name of the wrapper element in the SOAP message or to remove it altogether.

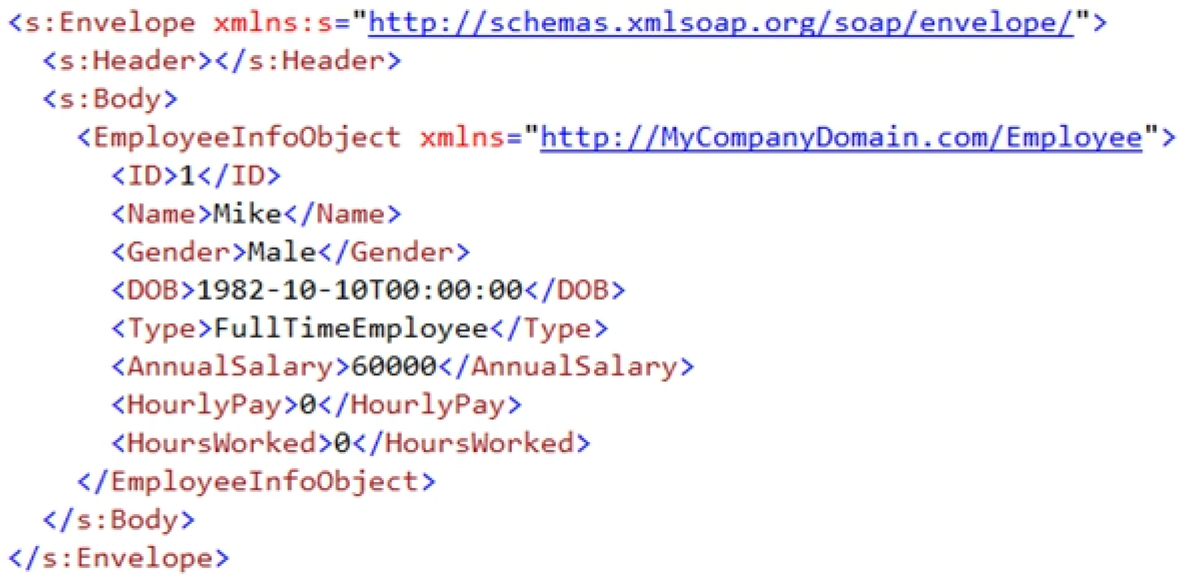
- **Example:**

SOAP Request message that is generated using Message Contracts



```xml
<s:Envelope xmlns:s="http://schemas.xmlsoap.org/soap/envelope/">
  <s:Header>
    <h:LicenseKey xmlns:h="http://MyCompanyDomain.com/Employee">
      XYZ120FABC
    </h:LicenseKey>
  </s:Header>
  <s:Body>
    <EmployeeRequestObject xmlns="http://MyCompanyDomain.com/Employee">
      <EmployeeId>1</EmployeeId>
    </EmployeeRequestObject>
  </s:Body>
</s:Envelope>
```

SOAP Response message that is generated using Message Contracts



```xml
<s:Envelope xmlns:s="http://schemas.xmlsoap.org/soap/envelope/">
  <s:Header></s:Header>
  <s:Body>
    <EmployeeInfoObject xmlns="http://MyCompanyDomain.com/Employee">
      <ID>1</ID>
      <Name>Mike</Name>
      <Gender>Male</Gender>
      <DOB>1982-10-10T00:00:00</DOB>
      <Type>FullTimeEmployee</Type>
      <AnnualSalary>60000</AnnualSalary>
      <HourlyPay>0</HourlyPay>
      <HoursWorked>0</HoursWorked>
    </EmployeeInfoObject>
  </s:Body>
</s:Envelope>
```

- Decorate a class with **MessageContract a**ttribute, and then use that class as an operation contract parameter or return type. MessageContract attribute has the following parameters.
  1. IsWrapped
  2. WrapperName
  3. WrapperNamespace
  4. ProtectionLevel

- **MessageHeader** attribute is applied on a property of the class that you want to include in the SOAP header**.** MessageHeader attribute has the following parameters.
  1. Name
  2. Namespace
  3. ProtectionLevel

4. Actor
5. MustUnderstand
6. Relay

- **MessageBodyMember a**ttribute is applied on a property of the class that you want to include in the SOAP body section. MessageBodyMember attribute has the following parameters.
  1. Name
  2. Namespace
  3. Order
  4. ProtectionLevel

Note: Use MessageContract only if there is a reason to tweak the structure of the soap XML message.

<u>Difference b/w Data Contract and message Contract</u>

Data Contract	**Message Contract**
Very limited control over the SOAP messages	Full control over the SOAP messages. .
Only allows us to control the Name and Order of XML elements in the body section of the SOAP message.	Provides access to the SOAP header and body sections using **MessageHeader** and **MessageBody Member** attributes.

1. **Why do use MessageContract in WCF?**
MessageContract gives full control over the SOAP messages. For example, it allows us to include custom information in the SOAP header.

2. What kind of custom information?
User credentials to invoke the service.

3. Why do you need to pass user credentials in the header? Can't you pass them as method parameters?
We can, but user credentials are periphery to what the method has to do. So, it would make more sense to pass them out of band in the header, rather than as additional parameters.

4. SOAP messages are in xml format, so anyone can read the credentials? How you do you protect sensitive data?
Using MessageContract we can sign and encrypt messages. Use ProtectionLevel named parameter.

<u>Backward compatible WCF contract changes (not important)</u>

<u>Figure 1: Service contracts and backward compatibility</u>

WCF CONTRACTS AND BACKWARD COMPATIBI	
Service Contract Changes	Impact to Existing Clients
Adding new parameters to an	Client unaffected. New parameters initialized to default values at the servic

Service Contract Changes	Impact to Existing Clients
operation signature	
Removing parameters from an operation signature	Client unaffected. Superfluous parameters pass by clients are ignored, data at the service.
Modifying parameter types	An exception will occur if the incoming type from the client cannot be converted to the parameter data type.
Modifying return value types	An exception will occur if the return value from the service cannot be converted to the expected data type in the client version of the operation signature.
Adding new operations	Client unaffected. Will not invoke operations it knows nothing about.
Removing operations	An exception will occur. Messages sent by the client to the service are considered to be using an unknown action header.

Figure 2: Data contracts and backward compatibility

Data Contract Changes	Impact to Existing Clients
Add new non-required members	Client unaffected. Missing values are initialized to defaults.
Add new required members	An exception is thrown for missing values.
Remove non-required members	Data lost at the service. Unable to return the full data set back to the client, f example. No exceptions.
Remove required members	An exception is thrown when client receives responses from the service wit missing values.
Modify existing member data types	If types are compatible no exception but may receive unexpected results.

IExtensibleDataObject

- Use IExtensibleDataObject to preserve unknown elements during serialization and deserialization of DataContracts.
- On the service side, at the time of deserialization the unknown elements from the client are stored in ExtensionDataObject. To send data to the client, the service has to serialize data into XML. During this serialization process the data from ExtensionDataObject is serialized into XML as it was provided at the time of service call.
- The downside of implementing **IExtensibleDataObject** interface is the risk of **Denial of Service attack**. Since, the extension data is stored in memory, the attacker may flood the server with requests that contains large number of unknown elements which can lead to system out of memory and DoS.

NOTE: When **IExtensibleDataObject** feature is turned off, the deserializer will not populate the ExtensionData property.

Exception handling in WCF or Soap Faults

What happens when an exception occurs in a WCF service?
OR
What is a SOAP fault?
OR
How are WCF service exceptions reported to client applications?

- When an exception occurs in a WCF service, the service serializes the exception into a SOAP fault, and then sends the SOAP fault to the client.
- By default unhandled exception details are not included in SOAP faults that are propagated to client applications for security reasons. Instead a generic SOAP fault is returned to the client.
- If we want to include exception details in SOAP faults, enable **IncludeExceptionDetailInFaults** setting. This can be done in 2 ways as shown below.

  1. In the config file using service behavior configuration

  ```
  <behaviors>
    <serviceBehaviors>
      <behavior name="inculdeExceptionDetails">
        <serviceDebug includeExceptionDetailInFaults="true"/>
      </behavior>
    </serviceBehaviors>
  </behaviors>
  ```

  2. In code using ServiceBehavior attribute

  ```
  [ServiceBehavior(IncludeExceptionDetailInFaults=true)]
  public class CalculatorService : ICalculatorService
  {
      public int Divide(int Numerator, int Denominator)
      {
          return Numerator / Denominator;
      }
  }
  ```

NOTE: WCF serializes exceptions to SOAP faults before reporting the exception information to the client. This is because exceptions are not allowed to be passed through a wcf service channel.

- SOAP faults are in XML format and are platform independent. A typical SOAP fault contains
  1. FaultCode
  2. FaultReason
  3. Detail elements etc.
- The Detail element can be used to include any custom xml.
- SOAP faults are formatted based on **SOAP 1.1** or **SOAP 1.2** specifications. SOAP format depends on the binding. BasicHttpBinding uses SOAP 1.1 and wsHttpBinding with security code = none uses SOAP 1.2.

NOTE: Soap faults are included under soap body xml representation.

Unhandled exceptions in WCF

- An unhandled exception in the WCF service, will cause the communication channel to fault and the session will be lost. Once the communication channel is in faulted state, we cannot use the same instance of the proxy class any more. We will have to create a new instance of the proxy class.
- **BasicHttpBinding does not have sessions.** So when there is an unhandled exception, it only faults the server channel. The client proxy is still OK, because with BasicHttpBinding the channel is not maintaining sessions, and when the client calls again it is not expecting the channel to maintain any session.
- **WSHttpBinding have secure sessions.** So when there is an unhandled exception, it faults the server channel. At this point the existing client proxy is useless as it is also faulted, because with wsHttpBinding the channel is maintaining a secure session, and when the client calls again it expects the channel to maintain the same session. The same session does not exist at the server channel anymore, as the unhandled exception has already torn down the channel and the session along with it.

Throwing fault exceptions from a WCF service

- A WCF service should be throwing a **FaultException** or **FaultException<T>** instead of **Dot Net exceptions.** This is because of the following 2 reasons.
  1. An unhandled .NET exception will cause the channel between the client and the server to fault. Once the channel is in a faulted state we cannot use the client proxy anymore. We will have to re-create the proxy. On the other hand fault exceptions will not cause the communication channel to fault.

  2. As .NET exceptions are platform specific, they can only be understood by a client that is also .NET. If you want the WCF service to be interoperable, then the service should be throwing FaultExceptions.

  Note: FaultException<T> allows us to create strongly typed SOAP faults.
- Example of throwing a fault exception:

```
        throw new FaultException("Denomintor cannot be ZERO", new
FaultCode("DivideByZeroFault"));
```

Syntax: throw new FaultException(FaultReason, FaultCode);

- To create a strongly typed SOAP fault
  1. Create a class that represents your SOAP fault. Decorate the class with **DataContract** attribute and the properties with **DataMember** attribute.
  2. In the service data contract, use **FaultContractAttribute** to specify which operations can throw which SOAP faults.
  3. In the service implementation create an instance of the strongly typed SOAP fault and throw it using **FaultException<T>**.

Centralized exception handling in WCF by implementing IErrorHandler interface

- How to handle all WCF service exceptions in one central location?
  In an ASP .NET web applications we can use Application_Error() event handler method in Global.asax to log all the exceptions and redirect the user to a custom error page.

- In WCF, to centralize exception handling and to return a general fault reason to the client, we implement IErrorHandler interface. Let's now look at the 3 steps involved in centralizing exception handling in WCF,

Step 1: Implement IErrorHandler interface.
IErrorHandler interface has 2 methods for which we need to provide implementation.
1. ProvideFault() - This method gets called automatically when there is an unhandled exception or a fault. In this method we have the opportunity to write code to convert the unhandled exception into a generic fault that can be returned to the client. ProvideFault() gets called before HandleError() method.

2. HandleError() - This method gets called asynchronously after ProvideFault() method is called and the error message is returned to the client. This means that this method allows us to write code to log the exception without blocking the client call.

Step 2: Create a custom Service Behaviour Attribute to let WCF know that we want to use the GlobalErrorHandler class whenever an unhandled exception occurs.

Notice that the GlobalErrorHandlerBehaviourAttribute class
1. Inherits from **Attribute** abstract class.

2. Implements **IServiceBehavior** interface. This interface has 3 methods (Validate(), AddBindingParameters(), ApplyDispatchBehavior()). The implementation for Validate() and AddBindingParameters() method can be left blank. In the ApplyDispatchBehavior() method, we create an instance of the GlobalErrorHandler class and associate the instance with each channelDispatcher.

3. Has a constructor that contains one Type parameter. We will use this constructor in Step 3.

Step 3: Decorate **CalculatorService** class in CalculatorService.cs file with **GlobalErrorHandlerBehaviourAttribute.** Notice that this attribute has one constructor that expects a single Type parameter. Pass GlobalErrorHandler class created in Step 1 as the argument.

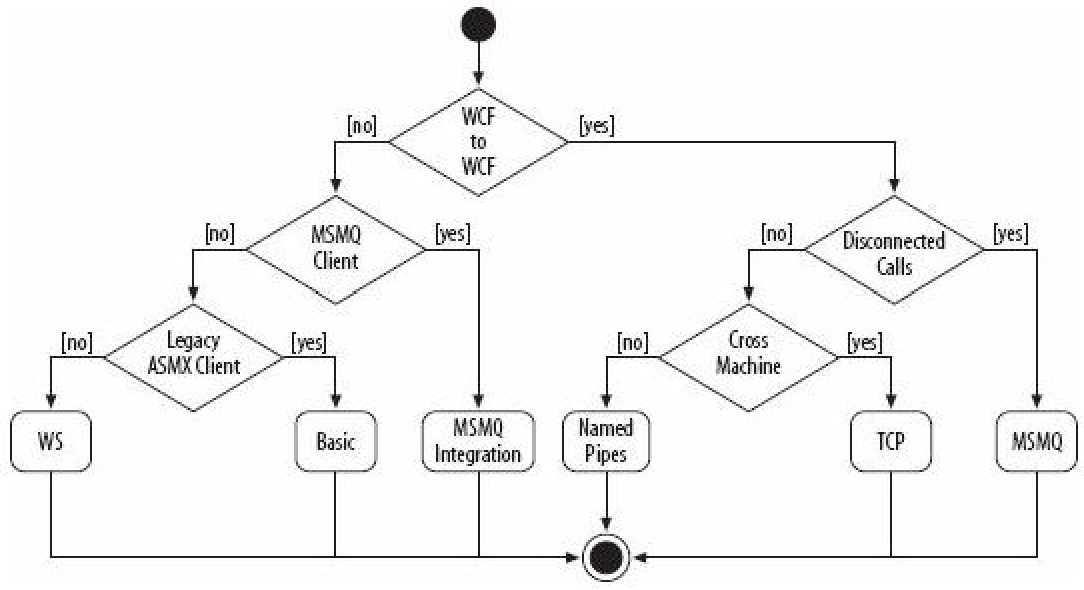
```
[GlobalErrorHandlerBehaviour(typeof(GlobalErrorHandler))]
public class CalculatorService : ICalculatorService
{
   public int Divide(int Numerator, int Denominator)
   {
   }
}
```

Bindings in WCF

- WCF service endpoint consists of 3 things
  A - Address (Defines where the WCF Service is available)
  B - Binding (Defines how the client needs to communicate with the service)
  C - Contract (Specifies what the service can do. For example, the service contract describes which operations the client can perform on the service)
- WCF binding that you choose determines the following for the communication between the client and the service.
  Transport Protocol (for example, HTTP, TCP, NamedPipe, MSMQ)
  Message Encoding (for example, text/XML, binary, or (MTOM) Message Transmission Optimization Mechanism)
  Protocols (for example, reliable messaging, transaction support)
- Supported wcf bindings are available in the below link

https://docs.microsoft.com/en-us/dotnet/framework/wcf/system-provided-bindings?redirectedfrom=MSDN

- Choosing the right WCF binding



- As WCF is very extensible, you can also create a custom binding and use it, if none of the system provided bindings suit your needs.

Configure WCF service endpoint

We can configure an endpoint for WCF service in 2 ways
1. Declaratively using the configuration file
2. Dynamically in code

First let's look at configuring an **endpoint** for the **HelloService declaratively** using the config file.

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <system.serviceModel>
    <behaviors>
```

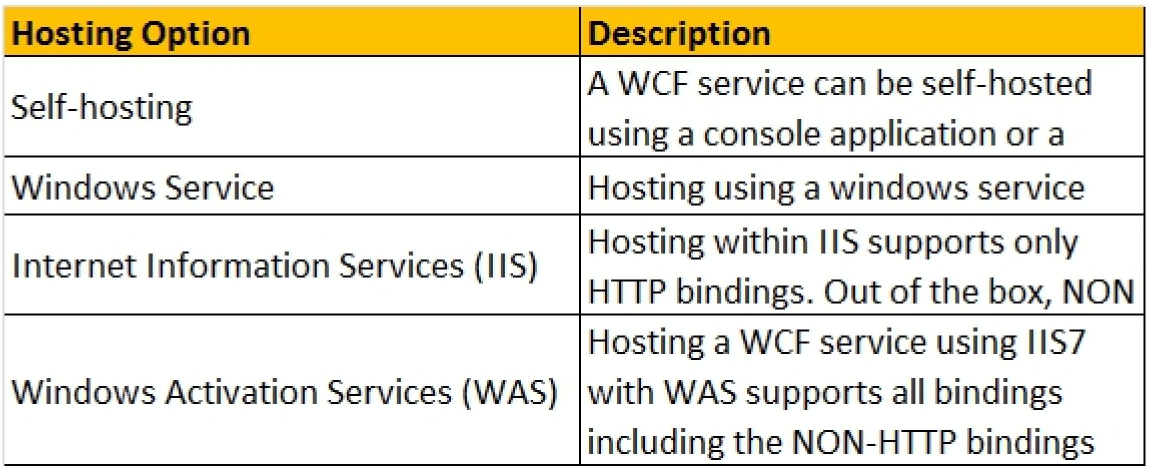
```
      <serviceBehaviors>
        <behavior name="mexBehaviour">
          <serviceMetadata httpGetEnabled="true" />
        </behavior>
      </serviceBehaviors>
    </behaviors>
    <services>
      <service name="HelloService.HelloService" behav
iorConfiguration="mexBehaviour">
        <endpoint address="HelloService" binding="net
TcpBinding" contract="HelloService.IHelloService">
        </endpoint>
        <host>
          <baseAddresses>
            <add baseAddress="http://localhost:8080/"
 />
            <add baseAddress="net.tcp://localhost:809
0"/>
          </baseAddresses>
        </host>
      </service>
    </services>
  </system.serviceModel>

</configuration>
```

Dynamically in code

In the **Main()** method in **Program.cs** file(in the Host project), include the lines highlighted with yellow color.

```
public static void Main()
{
   using (System.ServiceModel.ServiceHost host = new
      System.ServiceModel.ServiceHost(typeof(HelloService.HelloService)))
   {
      host.Description.Behaviors.Add(new ServiceMetadataBehavior { HttpGetEnable
d = true });
      host.AddServiceEndpoint(typeof(HelloService.IHelloService),
         new NetTcpBinding(), "HelloService");
      host.Open();
      Console.WriteLine("Host started @ " + DateTime.Now.ToString());
      Console.ReadLine();
   }
}
```

Hosting WCF service

- For a WCF service to be available for the clients to consume, we need to host the WCF service. The following are the different WCF service hosting options.
- 

Hosting Option	Description
Self-hosting	A WCF service can be self-hosted using a console application or a
Windows Service	Hosting using a windows service
Internet Information Services (IIS)	Hosting within IIS supports only HTTP bindings. Out of the box, NON
Windows Activation Services (WAS)	Hosting a WCF service using IIS7 with WAS supports all bindings including the NON-HTTP bindings

Self hosting a wcf service

- Hosting a wcf service in any managed .net application is called as self hosting.
- Console applications, WPF applications, WinForms applications are all examples of managed .net applications.
- **Advantages of self hosting a wcf service**
  1. Very easy to setup. Specify the configuration in app.config file and with a few lines of code we have the service up and running.
  2. Easy to debug as we don't have to attach a separate process that hosts the wcf service.
  3. Supports all bindings and transport protocols.
  4. Very flexible to control the lifetime of the services through the Open() and Close() methods of ServiceHost.
- **Disadvantages of self hosting a wcf service**
  1. The service is available for the clients only when the service host is running.
  2. Self hosting does not support automatic message based activation that we get when hosted within IIS.
  3. Custom code required.

NOTE: Self-hosting is only suitable during the development and demonstration phase and not for hosting live wcf services.

Hosting a WCF service using windows services

- **What is a windows service and how do they differ from regular applications and programs**
  A windows service is similar to any other program or application running on a windows machine. The following are the differences between a windows service and a regular application
  1. Windows service runs in the background
  2. They can be configured to start automatically when the system starts
  3. They don't have user interface.
- **What is the use of windows services**
  Windows services provide core operating system features such as

Event Logging - Windows Event Log Service
Providing Security - Windows Firewall Service
Error reporting - Windows Error Reporting Service

- **When would an asp.net developer use a windows service**
  In general, we should create a Windows Service to run code in the background all the time, without any sort of user interaction. An asp.net developer can use a windows service to host a wcf service. We can then configure the windows service to start automatically when the computer starts. This makes our WCF service always available for clients to consume, even if no-one is logged on, on that computer.
- Use windows service from add template.
- To install the service run the following command on visual studio commander

```
installutil -I [Path of the program]
```

- **Advantages of hosting a wcf service in a windows service**
  1. The Windows Service can be configured to start automatically when the system starts without having the need for any user to logon on the machine. This means that, the WCF service that the windows service hosts also starts automatically.
  2. The Windows Service can be configured to automatically restart and recover when failures occur.

  3. Supports all bindings and transport protocols
- **Disadvantages of hosting a wcf service in a windows service**
  1. Involves writing custom code to create a windows service.
  2. Windows service that hosts the wcf service must be deployed to the production server.
  3. Difficult to debug the wcf service, as we need to attach the process within which the windows service is running.

Hosting wcf service in IIS

- **To host a wcf service in IIS, create a file with .svc extension.** This file contains ServiceHost directive. The Service attribute of ServiceHost directive, specifies which service this file points to. The service code can reside in
  1. The .svc file
  2. A separate assembly
  3. A file in App_Code folder
- The configuration for the wcf service goes in web.config file.
- The ServiceHost directive in .svc file is responsible for creating an instance of ServiceHost when required. There is no need to write code to instantiate and start ServiceHost, as we did with self hosting.
- **Advantages:**

1. No code required to host the service
2. Automatic message based activation
3. Automatic process recycling

- **Disadvantages:**
  Hosting WCF service in IIS 5.1 and IIS 6.0 is limited to HTTP communication only. This means we can only use HTTP related bindings.

Hosting wcf service with WAS

- If you want to use NON-HTTP protocols like TCP, we need to install **"Windows Communication Foundation Non-HTTP Activation"** component and WAS **(Windows Process Activation Service)** component.
- To support NON-HTTP protocols in IIS, we need to do the following
  1. Install WAS (Windows Process Activation Service) and "Windows Communication Foundation Non-HTTP Activation component"
  2. Enable NON-HTTP protocol support in IIS for your application
- Advantages
  1. No code required to host the service
  2. IIS provides process recycling, automatic message based activation, idle time management etc
  3. Supports all transport protocols including the NON-HTTP protocols like TCP, named pipes etc

Message Exchange Patterns in WCF

- Message Exchange Pattern describes how the client and the wcf service exchange messages.
- WCF supports the following 3 Message Exchange Patterns

1. Request-Reply (Default)
2. One-Way
3. Duplex

Request-Reply Message Exchange Pattern

1. This is the default Message Exchange Pattern.
2. Client sends a message to a WCF Service and then waits for a reply. During this time the client client stops processing until a response is received from the wcf service.
3. The client waits for the service call to complete even if the operation return type is void.
4. All WCF bindings except the MSMQ-based bindings support the Request-Reply Message Exchange Pattern.

OneWay Message Exchange Pattern

- In **One-Way** operation, only one message is exchanged between the client and the service. The client makes a call to the service method, but does not wait for a response message. So, in short, the receiver of the message does not send a reply message, nor does the sender of the message expects one.
- As messages are exchanged only in one way, faults if any while processing the request does not get reported.
- Clients are unaware of the server channel faults until a subsequent call is made.
- An exception will be thrown, if operations marked with IsOneWay=true declares output parameters, by-reference parameters or return value.
- **Are OneWay calls same as asynchronous calls?**
  No, they are not. When a oneway call is received at the service, and if the service is busy serving other requests, then the call gets queued and the client is unblocked and can continue executing while the service processes the operation in the background. One-way calls can still block the client, if the number of messages waiting to be processed has exceeded the server queue limit. So, OneWay calls are not asynchronous calls, they just appear to be asynchronous.

Duplex message exchange pattern in WCF

- Duplex messaging pattern can be implemented using Request/Reply or OneWay operations.
- For more details, go through below link

https://csharp-video-tutorials.blogspot.com/2014/02/part-34-duplex-message-exchange-pattern_19.html

Sending large messages in WCF using MTOM

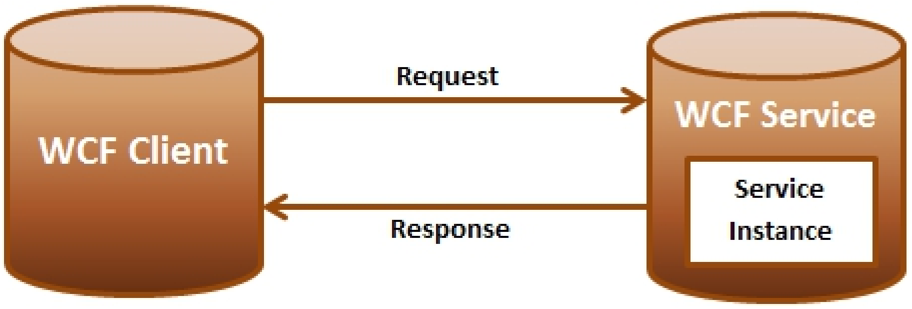
- The default message encoding mechanism in WCF is Text, which base64 encodes data. This has the following 2 disadvantages
  1. Base64 encoding bloats the message size by approximately 33%.
  2. Involves additional processing overhead to base64 encode and decode.
- The preferred approach to send large binary messages in WCF is to use MTOM message encoding.
- MTOM is an interoperable standard and stands for Message Transmission Optimization Mechanism.
- MTOM does not base64 encode data. This also means, the additional processing overhead to base64 encode and decode data is removed. Hence, MTOM can significantly improve the overall message transfer performance.
- With Text Message encoding, the binary data is base64 encoded and it is embedded in the SOAP envelop. With MTOM, binary data is included as a MIME (Multipurpose Internet Mail Extensions) attachment.

NOTE: To use MTOM set message encoding attribute to Ntom both at client side and server side

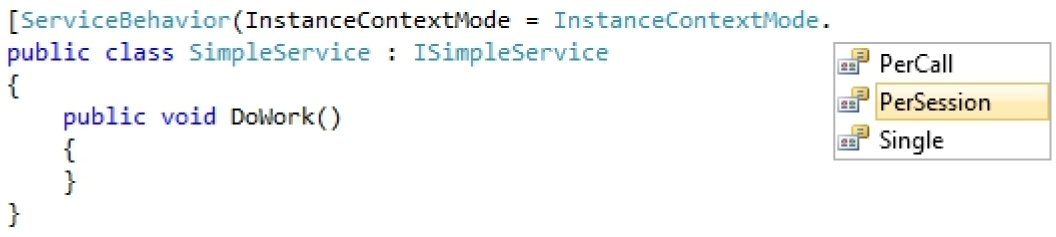
Example:
```
<bindings>
  <wsHttpBinding>
    <binding name="wsHttp" messageEncoding="Mtom"
            maxReceivedMessageSize="700000">
      <readerQuotas maxArrayLength="700000"/>
    </binding>
  </wsHttpBinding>

</bindings>
```

Instancing modes in WCF

- Instancing modes are also called as instance context modes.
- Instance context mode dictates, how long the service instance remains on the server.



- There are 3 instancing modes
  1. PerCall - A new instance of service object is created for every request, irrespective of whether the request comes from the same client or a different client.
  2. PerSession - A new instance of the service object is created for each new client session and maintained for the duration of that session.
  3. Single - A single instance of the service object is created and handles all requests for the lifetime of the application, irrespective of whether the request comes from the same client or a different client.
- **How do you specify what instancing mode you want to use?**
  Use ServiceBehavior attribute and specify InstanceContextMode.



```
[ServiceBehavior(InstanceContextMode = InstanceContextMode.
public class SimpleService : ISimpleService
{
    public void DoWork()
    {
    }
}
```

PerCall instance context mode in WCF

- What are the implications or overview of a PerCall WCF service?
  1. Better memory usage as service objects are freed immediately after the method call returns
  2. Concurrency not an issue
  3. Application scalability is better
  4. State not maintained between calls.
  5. Performance could be an issue as there is overhead involved in reconstructing the service instance state on each and every method call.

PerSession instance context mode in WCF

- What are the implications of a PerSession WCF service?
  1. State maintained between calls.
  2. Greater memory consumption as service objects remain in memory until the client session times out. This negatively affects application scalability.
  3. Concurrency is an issue for multi-threaded clients
- How do you design a WCF service? Would you design it as a PerCall service or PerSession service?
  1. PerCall and PerSession services have different strengths and weaknesses.
  2. If you prefer using object oriented programming style, then PerSession is your choice. On the other hand if you prefer SOA (Service Oriented Arhcitecture) style, then PerCall is your choice.
  3. In general, all things being equal, the trade-off is performance v/s scalability. PerSession services perform better because the service object does not have to be instantiated on subsequent requests. PerCall services scale better because the service objects are destroyed immediately after the method call returns.
  So the decision really depends on the **application architecture, performance & scalability needs.**
- Does all bindings support sessions?
  No, not all bindings support sessions. For example basicHttpBinding does not support session. If the binding does not support session, then the service behaves as a PerCall service.

- **How to control the WCF service session timeout?**
  The default session timeout is 10 minutes. If you want to increase or decrease the default timeout value,
  Step 1: Set **receiveTimeout** attribute of the respective binding element as shown below. In the example below, we have configured the session timout for 10 seconds.

```
<bindings>
  <netTcpBinding>
    <binding name="netTCP" receiveTimeout="00:00:10"></binding>
  </netTcpBinding>
</bindings>
```

Step 2: Associate the binding element with the endpoint
using **bindingConfiguration** attribute as shown below.

```
<endpoint address="SimpleService"
          binding="netTcpBinding"
          contract="SimpleService.ISimpleService"
          bindingConfiguration="netTCP"/>
```

- **What happens when the session timeout is reached**
  When the session timeout is reached, the connection to the wcf service is closed. As a result, the communication channel gets faulted and the client can no longer use the same proxy instance to communicate with the service. This also means that along with the session, the data in the service object is also lost.

  After the session has timed out,
  1. On the first attempt to invoke the service using the same proxy instance would result in the following exception.
  The socket connection was aborted. This could be caused by an error processing your message or a receive timeout being exceeded by the remote host, or an underlying network resource issue. Local socket timeout was '00:00:59.9355444'.

  2. On the second attempt, the following exception will be thrown
  The communication object, System.ServiceModel.Channels.ServiceChannel, cannot be used for communication because it is in the Faulted state.

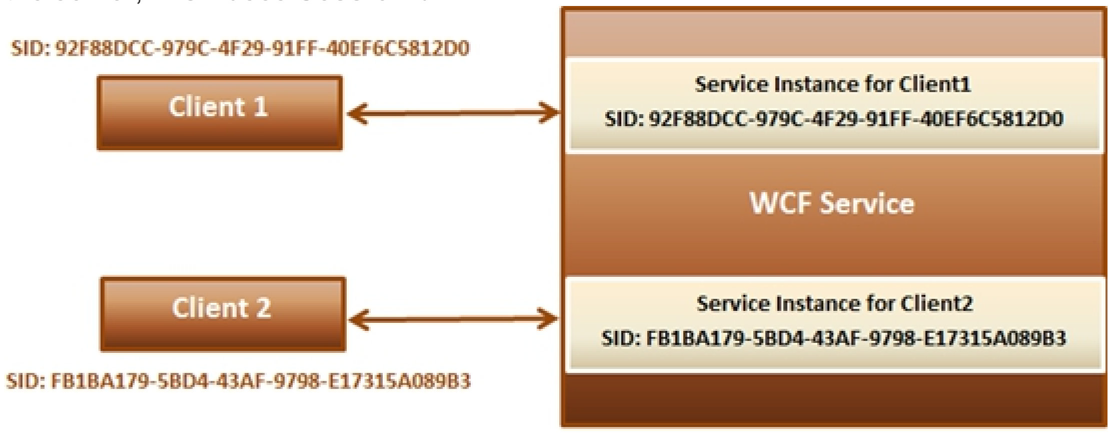
How to fix, The communication channel is in a faulted state exception
1. Put the line that calls the service should in the try block
2. Catch the CommunicationException
3. Check if the communication channel is in a faulted state and create a new instance of the proxy class.

Example:

```
try
{
    MessageBox.Show("Number = " + client.IncrementNumber().ToString());
}
catch (System.ServiceModel.CommunicationException)
{
    if (client.State == System.ServiceModel.CommunicationState.Faulted)
    {
        MessageBox.Show("Session timed out. Your existing session will be lost. A new
session will now be created");
        client = new SimpleService.SimpleServiceClient();
    }
}
```

- How to retrieve the sessionid in WCF service and in the client application?

In order to send messages from a particular client to a particular service instance on the server, WCF uses **SessionId**.


SID: 92F88DCC-979C-4F29-91FF-40EF6C5812D0
Client 1
Client 2
SID: FB1BA179-5BD4-43AF-9798-E17315A089B3
Service Instance for Client1
SID: 92F88DCC-979C-4F29-91FF-40EF6C5812D0
WCF Service
Service Instance for Client2
SID: FB1BA179-5BD4-43AF-9798-E17315A089B3

NOTE:There are different types of sessions in WCF.
To retrieve SessionId from the client application use procyClassInstance.InnerChannel.SessionId property

To retrieve SessionId from the WCF service use
OperationContext.Current.SessionId

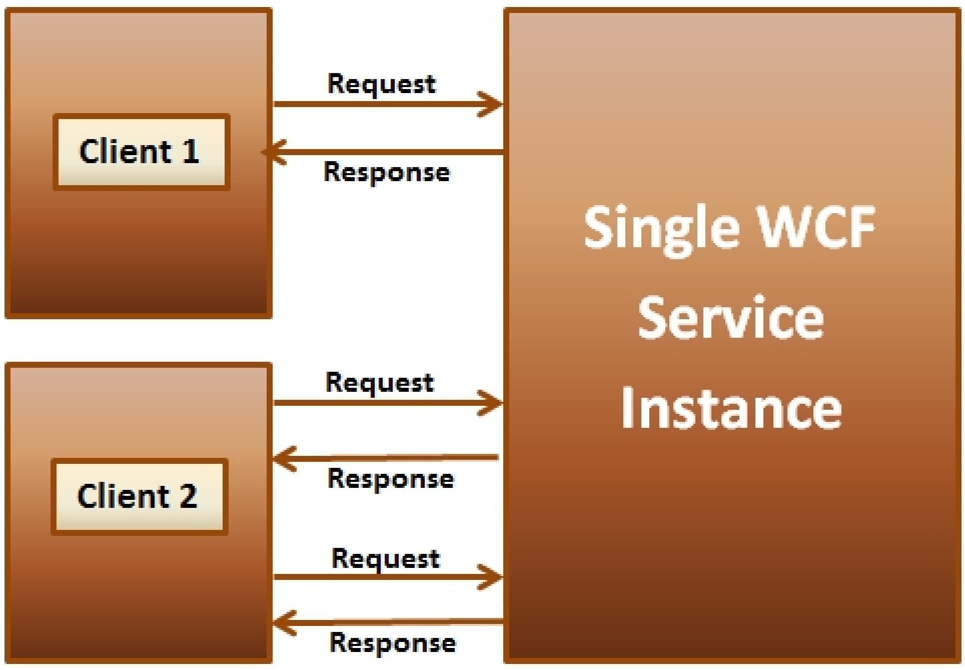
The client-side and service-side session IDs are corelated using the reliable session id. So, if TCP binding is being used with reliable sessions disabled then the client and server session id's will be different. On the other hand, if reliable sessions are enabled, the session id's will be same.

Enable reliable sessions using the binding element as shown below
```
<bindings>
  <netTcpBinding>
    <binding name="netTCP" receiveTimeout="00:00:10">
      <reliableSession enabled="true"/>
    </binding>
  </netTcpBinding>
</bindings>
```

With wsHttpBinding, irrespective of whether reliable sessions are enabled or not, the session id's will be same.

Single instance context mode in WCF

- When the instance context mode for a wcf service is set to Single, only one instance of the wcf service class is created to handle all requests, from all clients.



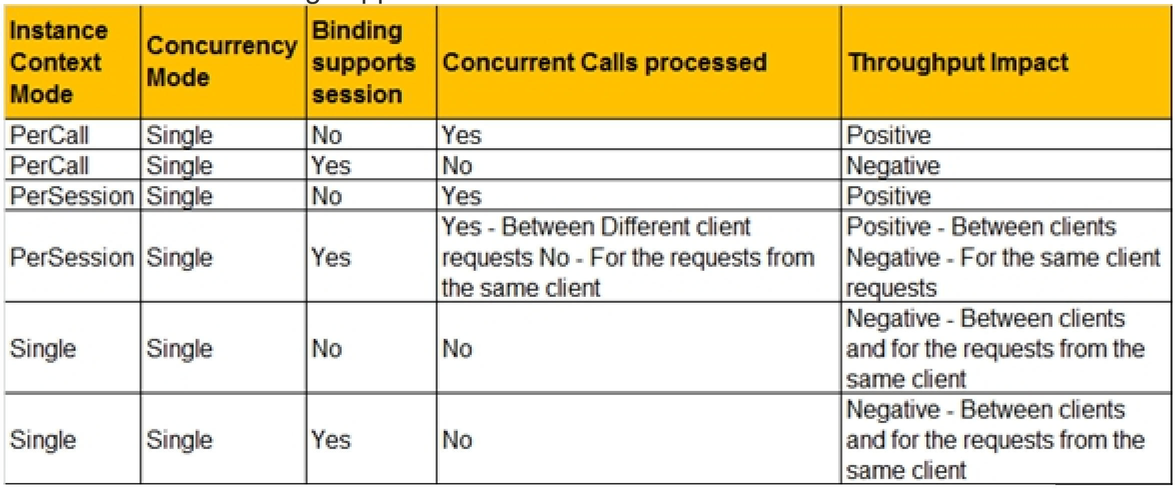
- **Set InstanceContextMode to Single**, to create a single instance of a WCF service that handles all requests from all clients.
  [ServiceBehavior(InstanceContextMode = InstanceContextMode.Single)]
  public class SimpleService : ISimpleService
- **Implications of creating a wcf service with Single instance context mode:**
  1. Since a single service instance is serving all client requests, state is maintained and shared not only between requests from the same client but also between different clients.
  2. Concurrency is an issue
  3. Throughput can be an issue. To fix the concurrency issue, we can configure the service to allow only a single thread to access the service instance. But the moment we do it throughput becomes an issue as other requests queue up and wait for the current thread to finish it's work.

SessionMode Enumeration in WCF

- Use **SessionMode** enumeration with the Service Contract to require, allow, or prohibit bindings to use sessions.
- SessionMode enum has the following members:
  Allowed : Service contract supports sessions if the binding supports them. This is the default if we have not explicitly specified the SessionMode on Service contract.
  NotAllowed : Service contract does not support bindings that initiate sessions.
  Required : Service contract requires a binding that supports session.
- **Example 1:** Set the service **InstanceContextMode to Single** and **SessionMode to Allowed**.
  If we use **basicHttpBinding** that does not support sessions, the service still works but without session.
  On the other hand if we use, **netTcpBinding** that support sessions, the service gets a session, and continue to work as a singleton service.
- **Example 2:** Now change **SessionMode** to **Required.**
  If we use, **netTcpBinding** that support sessions, the service gets a session, and continue to work as a singleton service.
  On the other hand, if we use **basicHttpBinding** that does not support sessions, an exception is thrown.

Single concurrency mode in WCF

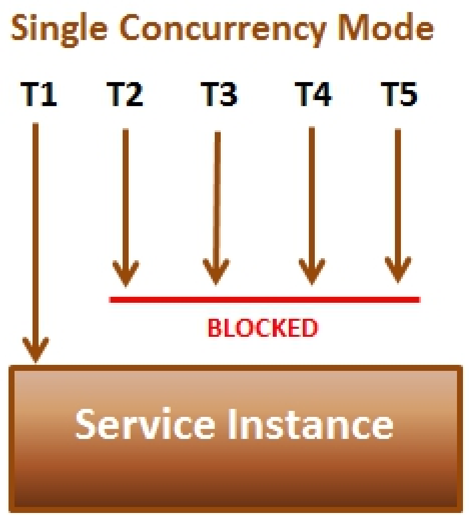
- **Throughput** is the amount of work done in a specified amount of time.
- Multiple threads executing the application code simultaneously is called as **concurrency.**
- In general with concurrency we get better throughput. However, concurrency issues can occur when multiple threads access the same resource. In WCF, Service Instance is a shared resource.
- **ConcurrencyMode** can be
  Single
  Reentrant
  Multiple
- The default concurrency mode in WCF is Single. This means only a single thread can access the service instance at any given point in time. While a request is being processed by the service instance, an exclusive lock is acquired and all the other threads will have to wait until the current request completes and the lock is released. Upon relasing the lock, next thread in the queue can access the service instance.
- ConcurrencyMode parameter of the ServiceBehavior attribute controls the concurrency setting
  [ServiceBehavior(ConcurrencyMode = ConcurrencyMode.Single)]
- Whether a WCF service handles client requests concurrently or not depends on 3 things
  1. Service Instance Context Mode
  2. Service Concurrency Mode and
  3. Whether if the binding supports session or not

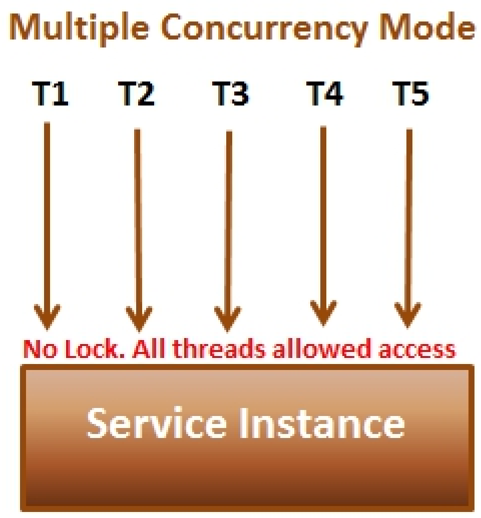


Instance Context Mode	Concurrency Mode	Binding supports session	Concurrent Calls processed	Throughput Impact
PerCall	Single	No	Yes	Positive
PerCall	Single	Yes	No	Negative
PerSession	Single	No	Yes	Positive
PerSession	Single	Yes	Yes - Between Different client requests No - For the requests from the same client	Positive - Between clients Negative - For the same client requests
Single	Single	No	No	Negative - Between clients and for the requests from the same client
Single	Single	Yes	No	Negative - Between clients and for the requests from the same client

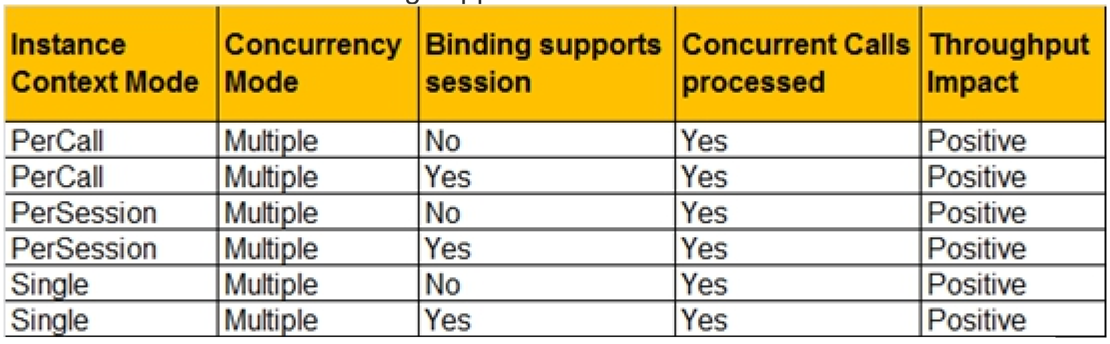
Multiple concurrency mode in WCF

- With Multiple concurrency mode an exclusive lock is not acquired on the service instance. This means multiple threads are allowed to access the service instance simultaneously and we get better throughput. However, shared resources if any must be protected from concurrent access by multiple threads to avoid concurrency issues.




Multiple Concurrency Mode
T1
T2
T3
T4
T5
No Lock. All threads allowed access
Service Instance

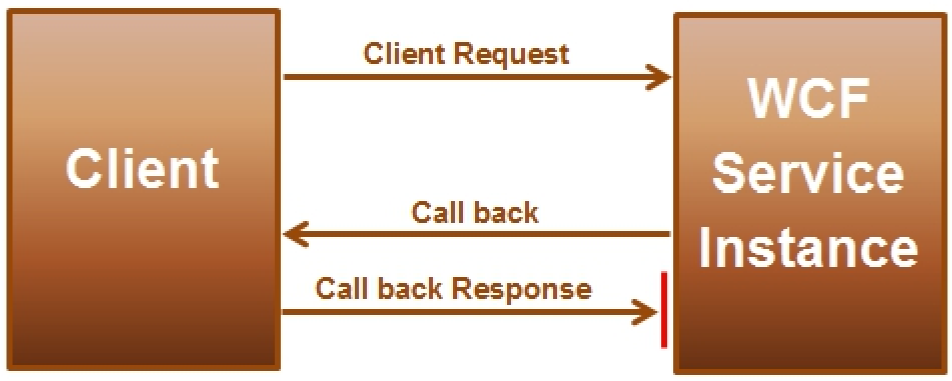
- **When concurrency mode is set to Multiple**, requests are processed concurrently by the service instance irrespective of the Service Instance Context Mode and whether if the binding supports session or not.



Instance Context Mode	Concurrency Mode	Binding supports session	Concurrent Calls processed	Throughput Impact
PerCall	Multiple	No	Yes	Positive
PerCall	Multiple	Yes	Yes	Positive
PerSession	Multiple	No	Yes	Positive
PerSession	Multiple	Yes	Yes	Positive
Single	Multiple	No	Yes	Positive
Single	Multiple	Yes	Yes	Positive

Reentrant concurrency mode in WCF

- Reentrant mode allows the WCF service to issue callbacks to the client application.



1. The WCF service concurrency mode is set to Single. This means only one thread is allowed to access the service instance.
2. Client makes a request to the WCF Service. The service instance gets locked by the thread that is executing the client request. At this point no other thread can access the service instance, until the current thread has completed and

released the lock.
3. While the service instance is processing the client request, the service wants to callback the client. The callback operation is not one way. This means the response for the callback from the client needs to get back to the same service instance, but the service instance is locked and the response from the client cannot reenter and access the service instance. This situation leads to a deadlock.

There are 2 ways to solve this problem.
1. Set the concurrency mode of the WCF service to Reentrant.

```
[ServiceBehavior(ConcurrencyMode = ConcurrencyMode.Reentrant)]
public class ReportService : IReportService
```

OR

2. Make the callback operation oneway. When the callback operation is made oneway, the service is not expecting a response for the callback from the client, so locking will not be an issue.

```
public interface IReportServiceCallback
{
    [OperationContract(IsOneWay = true)]
    void Progress(int percentageComplete);
}
```

WCF throttling

- **What is throughput?**
  Throughput is the amount of work done in a given time. In addition to Service Instance Context Mode and Concurrency Mode, Throttling settings also influence the throughput of a WCF service.
- Throttling settings can be specified either in the config file or in code

Throttling settings in config file

```
<behaviors>
  <serviceBehaviors>
    <behavior name="throttlingBehavior">
      <serviceThrottling
        maxConcurrentCalls="3"
        maxConcurrentInstances="3"
        maxConcurrentSessions="100"/>
    </behavior>
  </serviceBehaviors>
</behaviors>
```

Throttling settings in code

```
using (ServiceHost host = new ServiceHost(typeof(SimpleService.SimpleService)))
{
    ServiceThrottlingBehavior throttlingBehavior = new ServiceThrottlingBehavior
    {
        MaxConcurrentCalls = 3,
        MaxConcurrentInstances = 3,
        MaxConcurrentSessions = 100
    };
    host.Description.Behaviors.Add(throttlingBehavior);
    host.Open();
    Console.WriteLine("Host started @ " + DateTime.Now.ToString());
    Console.ReadLine();
}
```

Note: ServiceThrottlingBehavior class is present in System.ServiceModel.Description namespace

- With the above throttling settings a maximum of 3 concurrent calls are processed.
- In addition to **maxConcurrentCalls** property, **maxConcurrentInstances** and **maxConcurrentSessions** may also impact the number of calls processed concurrently.
- For example, if we set maxConcurrentCalls="3", maxConcurrentInstances="2", and maxConcurrentSessions="100",
  With a PerCall instance context mode, only 2 calls are processed concurrently. This is because every call requires a new service instance, and here we are allowing only a maximum of 2 concurrent instances to be created.

  With a Single instance context mode, 3 calls are processed concurrently. This is because with a singleton service there is only one service instance which handles all the requests from all the clients. With singleton maxConcurrentInstances property does not have any influence.

  Note: maxConcurrentSessions is the sum of all types of sessions, that is
  1. Application sessions
  2. Transport sessions
  3. Reliable sessions
  4. Secure sessions
- **Defaults:**
  Before WCF 4.0
  MaxConcurrentCalls: 16
  MaxConcurrentSessions: 10
  MaxConcurrentInstances: MaxConcurrentCalls + MaxConcurrentSessions(26)

  WCF 4.0 and later
  MaxConcurrentCalls: 16 * processor count
  MaxConcurrentSessions: 100 * processor count
  MaxConcurrentInstances: MaxConcurrentCalls + MaxConcurrentSessions

WCF security

- **Authentication -** The process of identifying the sender and recipient of the message.
- **Authorization -** The process of determining what rights the authenticated user has.
- **Confidentiality -** The process of ensuring that only the intended recipient of the message can view the message as it is being transmitted from the sender to the receiver. We achieve confidentiality by encrypting the message.
- **Integrity -** The process of ensuring that the message is not tampered with by a malicious user as it is being transmitted from the sender to the receiver. We achieve Integrity by signing the messages.
- **Bindings in WCF determine the security scheme.** The following MSDN link contains all the system provided bindings and their respective security defaults.
  http://msdn.microsoft.com/en-us/library/ms731092(v=vs.110).aspx
- The default security scheme for **NetTcpBinding** is **Transport** and for **WSHttpBinding** it is **Message.**

- From a security perspective, when sending a message between a client and a WCF service, there are 2 things to consider
  1. The WCF Message itself
  2. The medium or protocol (HTTP, TCP, MSMQ) over which the message is sent
- **Securing the transport channel is called transport security.**
- Each of the protocols (HTTP, TCP, MSMQ etc) have their own way of providing transport security.
- For example, TCP provides transport security, by implementing Transport Layer Security (TLS). The TLS implementation is provided by the operating system.
- HTTP provides transport security by using Secure Sockets Layer (SSL) over HTTP. Transport security provides only point-to-point channel security. It means if there is an intermediary (Load balancer, proxy etc) between, then that intermediary has direct access to content of the message.
- Securing the message itself by encapsulating the security credentials with every SOAP message is called message security. As the message itself is protected, it provides end to end security.

  The following MSDN article explains all the differences between message and transport security and when to use one over the other.
  http://msdn.microsoft.com/en-us/library/ms733137.aspx
- By default for secure bindings WCF messages are signed and encrypted.

Controlling WCF message protection using ProtectionLevel parameter

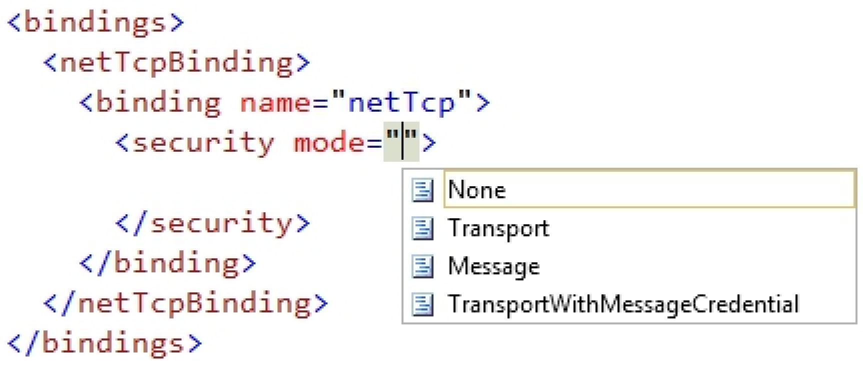
- **The following 6 attributes has the ProtectionLevel named parameter,** which implies that ProtectionLevel can be specified using any of the below 6 attributes. They are specified in the order of precedence.
- For example ProtectionLevel specified at an operation contract level overrides the ProtectionLevel specified at the service contract level.
  ServiceContractAttribute
  - **OperationContractAttribute**
    - **FaultContractAttribute**
    - **MessageContractAttribute**
      - **MessageHeaderAttribute**
      - **MessageBodyMemberAttribute**
- ProtectionLevel specified on the ServiceContract will be applicable for all the operation contracts within that service contract.
- ProtectionLevel specified on the OperationContract will be applicable for just that OperationContract.
- If we specify ProtectionLevel on a MessageContract, then that protection level will be applicable for all messages of that MessageContract type.
- **When we use wsHttpBinding,** by default the messages are encrypted and signed. Encryption provides confidentiality and signing provides integrity. To customize the level of message protection use ProtectionLevel parameter.

  Note: ProtectionLevel enum is present in System.Net.Security namespace.
- **ProtectionLevel enum has the following values**
  None - No protection. Message is not signed and not encrypted. Provides only authentication.
  Sign - No encryption but is digitally signed to ensure the integrity of the message
  EncryptAndSign - Message is encrypted and then signed to ensure confidentiality and integrity of the message.

- What happens if the binding does not provide security, and you have explicitly set ProtectionLevel other than None?
  An exception will be thrown.
- In general ProtectionLevel parameter is used to enforce the minimum level of protection required. If the binding does not provide that minimum level of protection then an exception will be thrown.

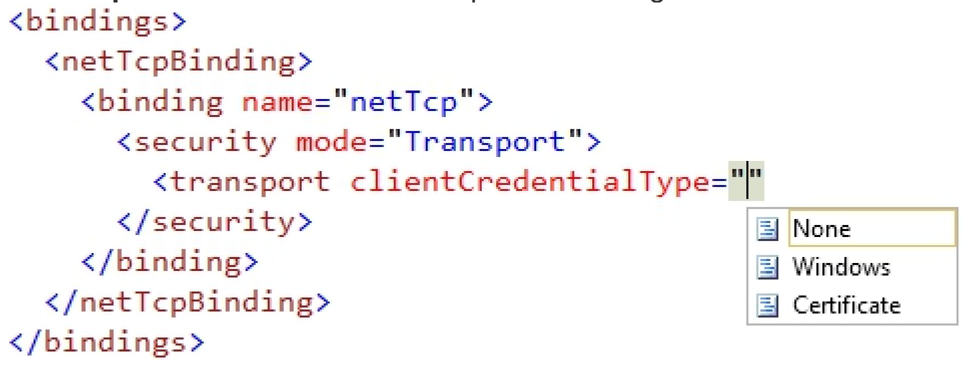
Authentication in WCF

- Both **wsHttpBinding** and **netTcpBinding** provides windows authentication.
- **Note:** If you have another windows user configured on the machine and if you run the application as a different user, you should see the user name of that user.
- To customize the security mode for a binding, use **mode** attribute of **security** element with in the respective binding



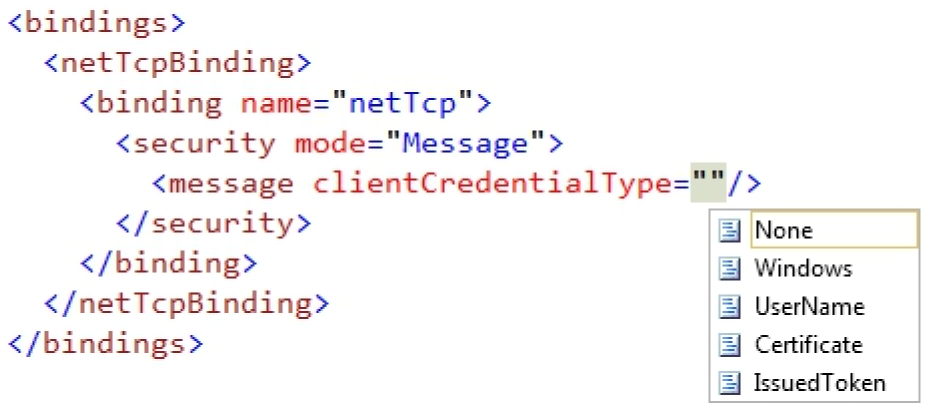
```
<bindings>
  <netTcpBinding>
    <binding name="netTcp">
      <security mode="">

      </security>
    </binding>
  </netTcpBinding>
</bindings>
```

To customize the authentication mode for a binding, use **clientCredentialType** attribute of either the **message** element or **transport** element with in the respective binding



```
<bindings>
  <netTcpBinding>
    <binding name="netTcp">
      <security mode="Transport">
        <transport clientCredentialType=""
      </security>
    </binding>
  </netTcpBinding>
</bindings>
```

OR



```
<bindings>
  <netTcpBinding>
    <binding name="netTcp">
      <security mode="Message">
        <message clientCredentialType=""/>
      </security>
    </binding>
  </netTcpBinding>
</bindings>
```

Message confidentiality and integrity with transport security

- netTcpBinding provides transport security.
- With transport security, all messages are encrypted and signed.

Configure wsHttpBinding to use transport security

- The following are the defaults of wsHttpBinding
  Security Mode - Message
  ClientCredentialType - Windows

  The following MSDN article contains all the system provided bindings and their security defaults.
  http://msdn.microsoft.com/en-us/library/ms731092(v=vs.110).aspx
- For more info go through below link

https://csharp-video-tutorials.blogspot.com/2014/04/part-52-configure-wshttpbinding-to-use_20.html

Configure netTcpBinding to use message security

- Default binding for netTcpBinding is Transport security

The following MSDN article contains all the system provided bindings and their security defaults. t.
http://msdn.microsoft.com/en-us/library/ms731092(v=vs.110).aspx