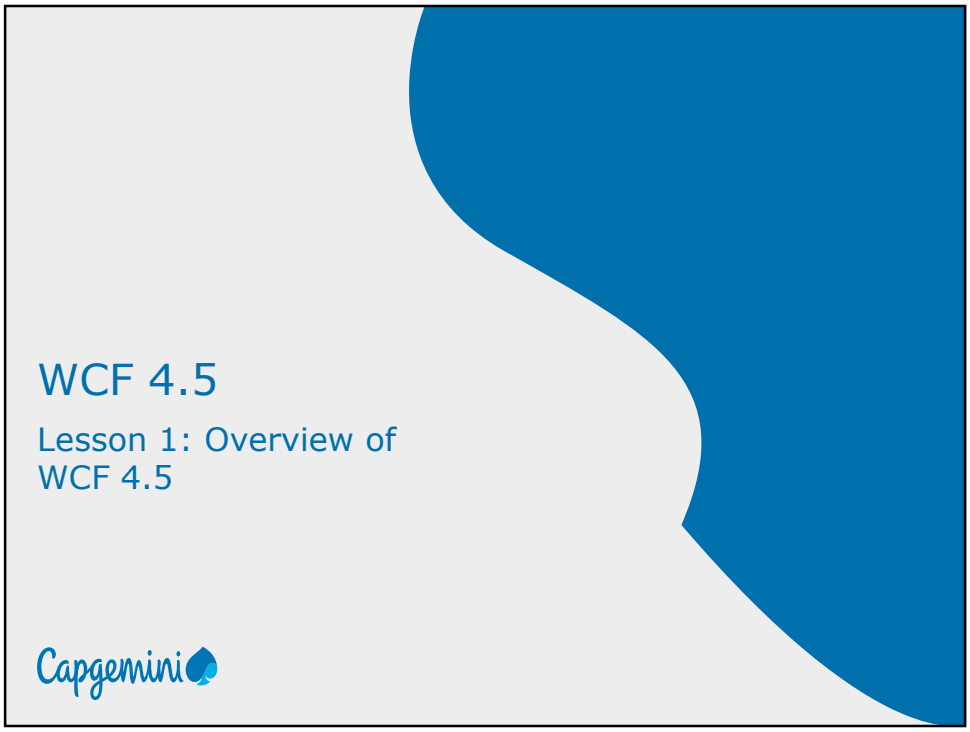
Instructor Notes:



Instructor Notes:

.NET 4.0

Microsoft started the development on the .NET Framework in the late 1990s originally under the name of Next Generation Windows Services (NGWS). By late 2000 the first beta versions of .NET 1.0 were released.

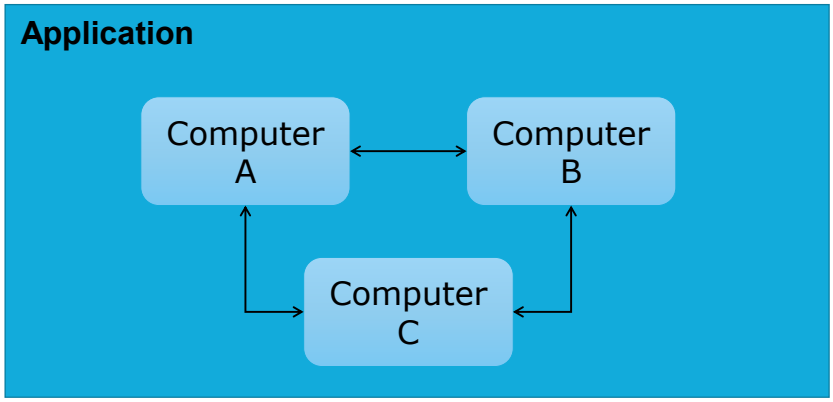
Version 3.0 of the .NET Framework is included with Windows Server 2008 and Windows Vista. Version 3.5 is included with Windows 7, and can also be installed on Windows XP and the Windows Server 2003 family of operating systems.

On 12 April 2010, .NET Framework 4 was released alongside Visual Studio 2010.

Instructor Notes:

Distributed Applications

- An application that is distributed / deployed which shares the business logic across multiple computer nodes.



Distributed Applications are software systems running on two or more computers in a computer network.

Instructor Notes:

Distributed application on Windows

- Microsoft shipped many communication frameworks to achieve connected system environment.

DCOM/COM+
Component Oriented(RPC). Distributed transactions. Requires DCOM infrastructure

.NET Remoting
Component Oriented(RPC). Simple & highly extensible. Requires CLR infrastructure

MSMQ
Message Oriented. Asynchronous. Requires MSMQ infrastructure

Disadvantages:
Each framework comes up with a unique programming model.
Each of these framework can be used only in windows platform.

DCOM

Distributed Component Object Model, an extension of the Component Object Model (COM) that allows COM components to communicate across network boundaries. Traditional COM components can only perform interprocess communication across process boundaries on the same machine. DCOM uses the RPC mechanism to transparently send and receive information between COM components (i.e., clients and servers) on the same network.

.NET Remoting

A .NET technology that allows objects residing in different application domains to communicate. Objects in different application domains are said to be separated by a remoting boundary. Objects using remoting may be on the same computer, or on different computers connected by a network. Remoting is the .NET replacement for DCOM.

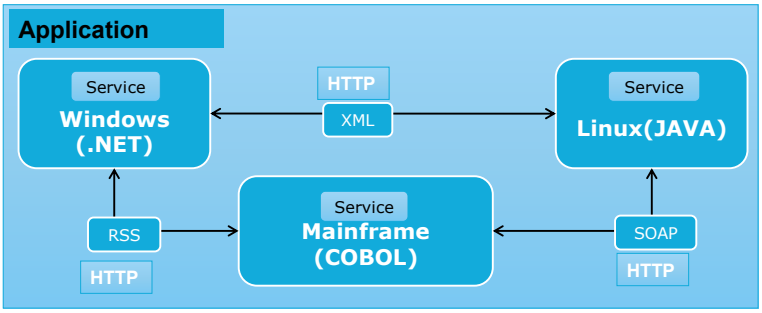
MSMQ

Microsoft Message Queuing, or MSMQ, is technology for asynchronous messaging. Whenever there's need for two or more applications (processes) to send messages to each other without having to immediately know results, MSMQ can be used. MSMQ can communicate between remote machines, even over internet.

Instructor Notes:

Services

- Demand for technology freedom, interoperability shorter development cycle were the key factors for the need of services.
- Services were used to expose the units of functionality via messages.



Web services extend the World Wide Web infrastructure to provide the means for software to connect to other software applications. Applications access Web services via various Web protocols and data formats such as HTTP, XML, and SOAP, with no need to worry about how each Web service is implemented. Web services combine the best aspects of component-based development and the Web.

A Web Service is a collection of functions that are packaged as a single entity and published to the network for use by other programs

A Web Service exposes a number of methods to provide functions

These functionalities can be used by one or more applications regardless of the following used to develop them:

- Programming languages
- Operating systems
- Hardware platforms

Instructor Notes:

SOAP & REST

- To make a service available over http can be done by exposing a SOAP based service or using a REST(non SOAP) based service.
- SOAP
  - XML Messaging using SOAP as the format, enhanced with the WS-* protocols(can be used with any transport protocol). Typically used in enterpise.
- REST
  - Which operates on resources through a unified interface(HTTP). Typically used in public facing web scenarios.

SOAP is a simple XML-based protocol to let applications exchange information over HTTP. SOAP is a protocol for accessing a Web Service.

SOAP stands for Simple Object Access Protocol

SOAP is a communication protocol

SOAP is a format for sending messages

SOAP is platform independent

SOAP is language independent

SOAP is based on XML

SOAP allows you to get around firewalls

SOAP is a W3C recommendation

REST is an "architectural style" that basically exploits the existing technology and protocols of the Web, including HTTP (Hypertext Transfer Protocol) and XML. REST is simpler to use than the well-known SOAP (Simple Object Access Protocol) approach, which requires writing or using a provided server program (to serve data) and a client program (to request data). SOAP, however, offers potentially more capability.

Instructor Notes:

Microsoft Web service frameworks

- Microsoft provides the following web service frameworks
  - ASP.NET Web Services : Basic XML over HTTP. Simple support for SOAP 1.1/1.2
  - Web Service Enhancements(WSE) : Added support for WS-Security and TCP Services.
- Limitations
  - Each framework comes up with a unique programming model.
  - Doesn't support transport neutrality
  - Doesn't support RESTful services

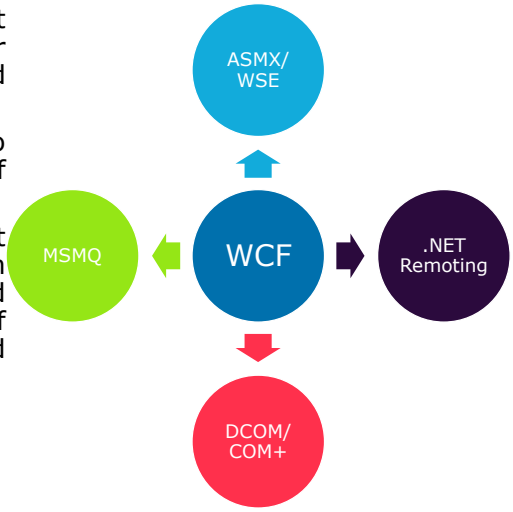
ASMX, also called ASP.NET Web Services, would be an option for communicating with the J2EE-based existing applications and with the applications across the Internet. Given that basic Web services are supported today on most platforms, this would likely be the most direct way to achieve cross-vendor interoperability.

Web Services Enhancements (WSE) could be used along with ASMX to communicate with the J2EE-based application and with the internet based applications. Because it implements more recently defined Web services agreements, known collectively as the WS-* specifications, WSE can allow better security and more, as long as all applications involved support compatible versions of these new specifications.

Instructor Notes:

What Is WCF?

- WCF is the Microsoft next generation technology for developing distributed applications
- WCF has been built to facilitate the development of service-oriented applications
- It is a fusion of current distributed system technologies designed and developed with the goal of achieving SOA based development



Windows Communication Foundation (WCF) is a platform, a framework, for creating and distributing connected applications.

WCF is a programming model that enables developers to build service solutions that are reliable and secure, and even transacted.

Building connected systems on the Microsoft platform usually means using multiple technologies and as such different programming models. WCF comes with a unified programming model that makes your life as a developer much easier. Some of the benefits are as follows:

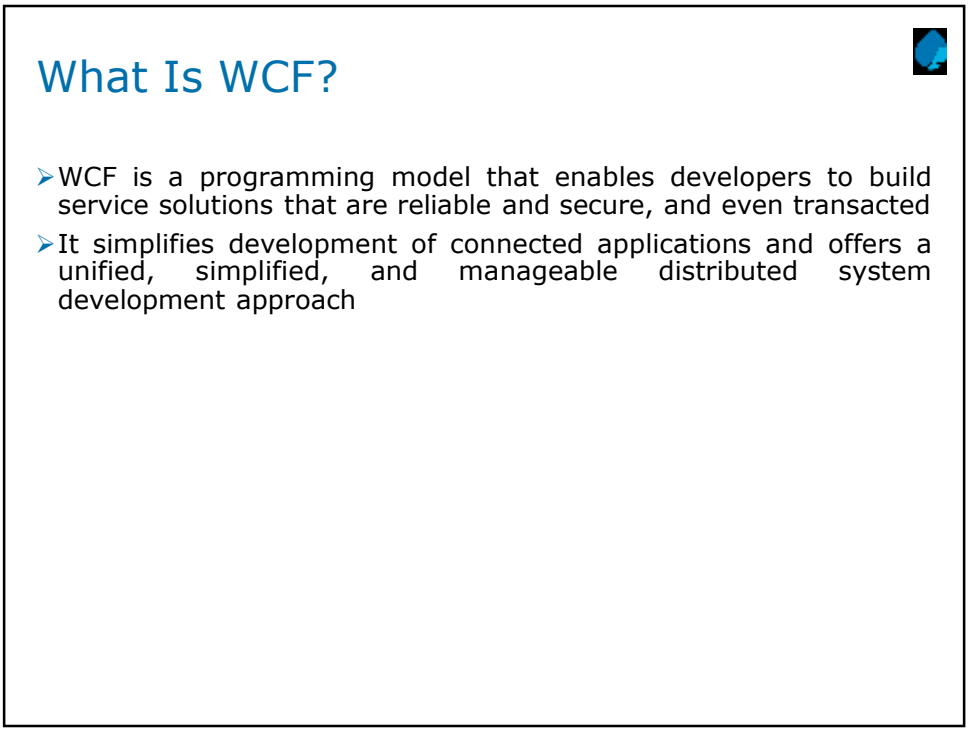
• Built-in support for a broad set of web service protocols

• Implicit use of service-oriented development principles

• A single API for building connected systems

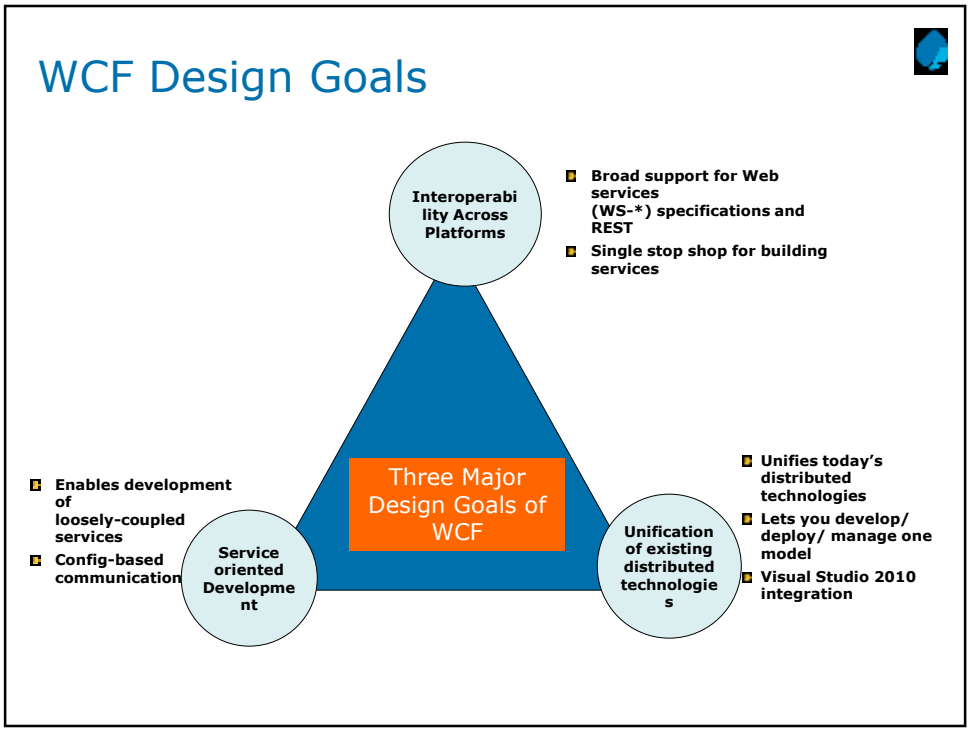
Built on top of the .NET Framework CLR (Common Language Runtime), the Windows Communication Foundation is a set of classes that allow developers to build service-oriented applications in .NET environment (VB.NET or C#).

Instructor Notes:



WCF provides a number of benefits that will make developers more productive, reduce the number of bugs, speed up application development, and simplify deployment. WCF helps achieve all those challenges using a unified programming model that provides for a unifying message style that is not only based on open standards but also is optimized for binary communication where applicable. WCF's unified programming model also provides for building secure, reliable, and interoperable applications.

Instructor Notes:



WCF is the first programming model built from the ground up to provide explicit service-oriented application development and future-ready business orientation. Service orientation is not a technology, but a design concept. Service orientation uses the best practices for building today's distributed applications.

The current world of enterprise computing has many distributed technologies, each of which has a notion to perform a specific task and have a distinct role in the space. Apart from that, these distributed

technologies are based on different programming models. For example, if you are building an application that communicates over HTTP, you will be required to change your programming model if

you want to switch to using TCP. If you are used to building XML web services today, you don't have the ability to support transactions with message queuing enabled without changing your programming model. This has created problems for developers, who have to keep learning different APIs for different ways of building distributed components.

WCF is Microsoft's solution to distributed application development for enterprise applications. It avoids confusion by taking all the capabilities of the existing distributed systems' technology stacks and enables you to use one clean and simple API

Instructor Notes:

Unification of distributed technologies

	ASMX	.NET Remoting	Enterprise Services	WSE	MSMQ	WCF
Interoperable Web Services	X					X
.NET–.NET Communication		X				X
Distributed Transactions, etc.			X			X
Support for WS-* Specifications				X		X
Queued Messaging					X	X

Most of the existing distributed technologies are based on the same concept. However, all of them provide specific services that are unique to the product (e.g., if you want to do queuing, you need to use MSMQ or System.Messaging; if you want to do transactions, you need to use System.EnterpriseServices; if you want to do security, you need to use WSE). As a programmer, you are constantly forced to switch between these programming models.

Therefore, we need one distributed technology to gather the capabilities of the existing stack and provide a solution with a simple and flexible programming model. WCF provides a unified programming model wherein you can compose all these different functionalities into your application without having to do a lot of context switching.

If you want queuing, you just add an attribute to your WCF service contract that makes it queued. If you want to secure the communication with your WCF service, you just add the appropriate security attributes for authentication and privacy. If you are after transaction support, you just add the transaction attribute. This new programming paradigm should help you concentrate more on the business logic.

Instructor Notes:

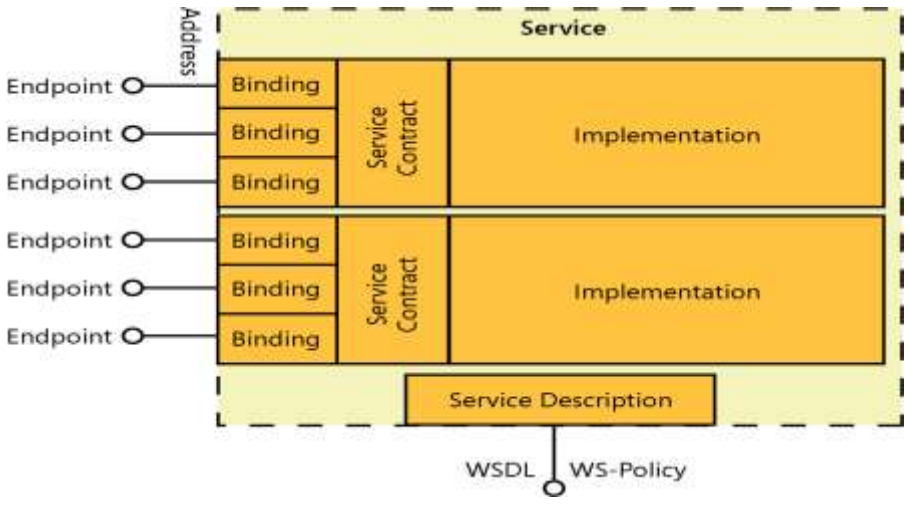
WCF Service from the outside

- A service is a consists of Business Logic and one or more endpoints
- Service implementation defines business logic
- Endpoint defines the communication options
- Each endpoint has its very own address, which makes it distinguishable from the other endpoints on the service

A service exposes a service description and a collection of endpoints. The service description tells what the service can do and how it can be accessed. The endpoints are the access points to the service. A service description provides essential information about how the service can be used. Service descriptions are expressed and communicated using several standards. This keeps service descriptions compatible with first-generation Web services while allowing more sophisticated descriptions for modern services.

Instructor Notes:

WCF Service from the inside



Services contain endpoints, bindings, contracts, and implementation code. Contracts describe a service's behavior, structures, or message format. Bindings describe how a service is accessed. Endpoints associate a contract and a binding with an address.

There are three kinds of contracts in Indigo services: service contracts, data contracts, and message contracts.

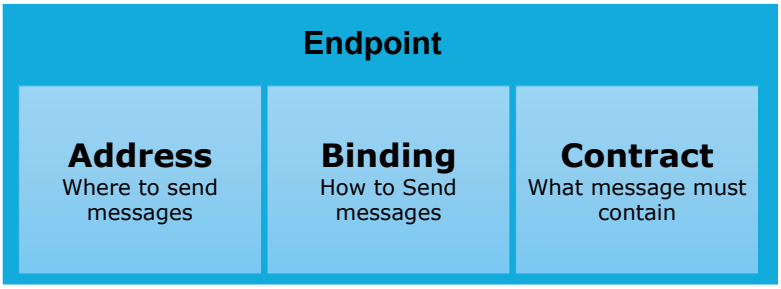
Bindings describe how a service communicates with the outside world. They specify such details as transport method, encoding format, reliability requirements, and security requirements. Indigo creates a channel stack to satisfy the binding you want to use. A service contract must be related to at least one binding in order to be accessible and can be related to multiple bindings.

Endpoints describe where a service is, associating an address with a service contract and binding. A service must provide at least one endpoint in order to be accessible and can have multiple endpoints.

Instructor Notes:

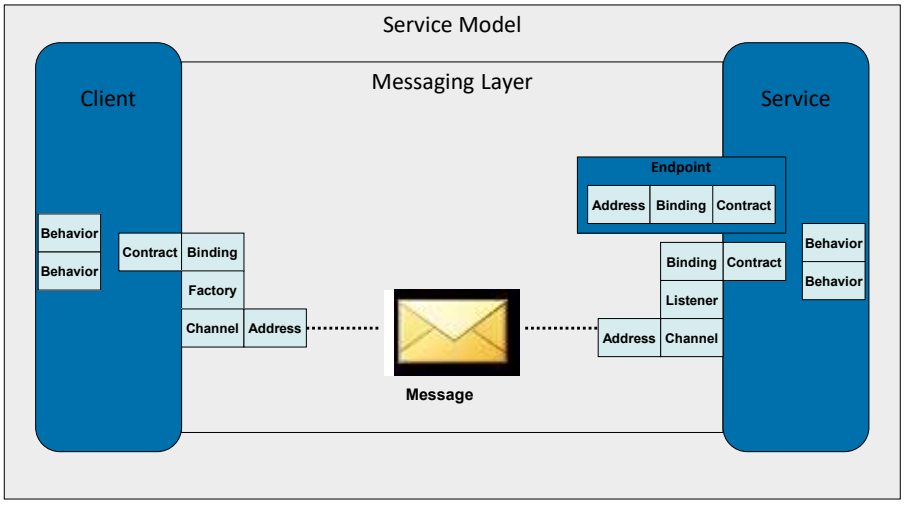
Endpoint

- The ABC's of a Service



Instructor Notes:

WCF Programming Model



Together, the address, binding, and contract are commonly referred to as an endpoint. On the client side, there is only one endpoint. Think of the client as a piece inside your program that is able to communicate with a service—not your entire application. This is commonly referred to as a proxy for that servic

The service side can have multiple endpoints. A service just sits around and waits for messages to come in on its endpoints. The service side has the same behaviors as the client side with regard to local configuration and operations.

Instructor Notes:

Addresses

- Addressing a service is essential to use a service
- Addresses in WCF are URLs that define the protocol used, the machine where the service is running, and the path to the service.
- Service endpoints are exposed through addresses
- WCF lets you use several types of addresses to associate with each endpoint, and it is through these addresses that the client communicates with the endpoint
- E.g. address might look like the following:
  - http://mymachine:8080/myservice

You need to have the address of a service to be able to send it a message. Addresses in WCF are URLs that define the protocol used, the machine where the service is running, and the path to the service. The port number is an optional field in the URL and depends on the protocol used.

The format of a service address is as follows:

scheme://<machinename>[:port]/path1/path2

As you can see, it is similar to a URL of a website. scheme can be any type of supported transport, machinename is the server name, port is the port where the service is listening, and path is essentially to differentiate services running on the same machine.

WCF supports several protocols, and each has its own particular addressing format.

Instructor Notes:

Bindings

- Bindings specify how to communicate with the WCF service
- The binding controls the following:
  - The transport (HTTP, MSMQ, Named Pipes, TCP)
  - The channels (one-way, duplex, request-reply)
  - The encoding (XML, binary, Message Transmission Optimization Mechanism [MTOM])
  - The supported WS-* protocols (WS-Security, WS-Federation, WS-Reliability, WSTransactions)
- At its most basic, a binding must specify the transport to use.
- WCF provides number of predefined bindings

A binding contains the following three categories of information:

Protocol information – Determines security mechanism that is being used, reliable messaging capability and transaction settings

Transport - Information about the underlying transport protocol to use, such as TCP or HTTP

Encoding - Information about the message encoding. E.g. Text or XML, Binary, or MTOM (Message Transmission Optimization Mechanism)

Instructor Notes:

Predefined Bindings in WCF

- 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 transactios.

The great thing about WCF is that it comes with a number of predefined bindings that should meet most of your development criteria.

BasicHttpBinding

This binding is probably the most comprehensive binding when talking in terms of interoperability. If you have written ASPX web services in the past, then you have used BasicHttpBinding in your web service. BasicHttpBinding uses HTTP as the transport protocol and Text/XML as the default message encoding. It represents bindings that a service can use to communicate with ASMX-based clients and web services.

Though simple to use, employment of the BasicHttpBinding presents some issues that developers need to be aware of. These issues are as follows:

❑ Security is disabled by default. To add security to this binding you can use the BasicHttpSecurityMode enumeration and set the value to anything other than None.

❑ This binding does not provide WS-* functionality and is fairly weak on the interoperability. It does not provide SOAP security or any transaction support.

WSHttpBinding

The WSHttpBinding offers a lot more functionality in the area of interoperability. Unlike BasicHttpBinding, WSHttpBinding supports WS-* functionality and distributed transactions with reliable and secure sessions using SOAP security. It uses the HTTP and HTTPS transport for communication as well. This option is the best for those developers looking for this level of WS-* interoperability.

WSDualHttpBinding

The WSDualHttpBinding is almost a mirror image of the WSHttpBinding except for one aspect, which is that WSDualHttpBinding supports duplex services. A duplex service is a service that uses duplex message patterns. These patterns provide the ability for a service to communicate back to the client via a callback.

The WSDualHTTPBinding also supports communication via SOAP intermediaries. Intermediaries are discussed in Chapter 2. They are not a part of WCF but certainly can have a hand in dealing with WCF services and messages in areas such as routing and load balancing. This binding allows the communication through intermediaries.

With WSDualHttpBinding, reliable sessions are enabled by default. Not so with WSHttpBinding. In WSHttpBinding, reliable sessions are disabled by default and must be enabled if you want to use them.

Instructor Notes:

Predefined Bindings (contd..)

- 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

NetMsmqBinding

The NetMsmqBinding provides a secure and reliable queued communication for cross-machine environments. Queuing is provided by using the MSMQ (Microsoft Message Queuing) as a transport, which enables support for disconnected operations, failure isolation, and load leveling. Each of these three is described in detail next.

A disconnected operation means that the client and the service do not have to be online at the same time. The client can initiate the transaction and disconnect. The service can accept the assignment and begin working on it while the client is disconnected.

Load leveling is the process of managing incoming messages so that the receiving service is not completely overwhelmed by the number of incoming messages.

Failure isolation means that messages can fail without affecting the processing of other messages. If a message is received by a service and the receiving service fails, the client can continue to send messages, which will be received by the message queue. When the failed receiving application is up and running again, it will start pulling messages off of the queue. This process guarantees message reliability and system stability.

NetTcpBinding

The NetTcpBinding provides a secure and reliable binding environment for .NET-to-.NET cross-machine communication. It uses the TCP protocol and provides full support for SOAP security, transactions, and reliability.

At runtime the NetTcpBinding creates a communication stack using the WS-ReliableMessaging protocol for reliability, Windows Security for message security and authentication, and TCP for message delivery.

The WS-ReliableMessaging protocol is an interoperable protocol used by both sender and receiver to ensure safe and reliable delivery of messages between the sender and receiver. The guaranteed delivery is specified as a delivery assurance, and it is the responsibility of the sender and receiver to fulfill the delivery assurance, or generate an error.

Instructor Notes:

Contracts

- WCF contracts define the functionality of WCF services
- They are created in code by service developers, and are exposed to clients in the service metadata
- The five types of contracts:
  - Service Contracts
  - Operation Contracts
  - Data Contracts
  - Message Contracts
  - Fault Contracts

WCF contracts are much like a contract that you and I would sign in real life. A contract I may sign could contain information such as the type of work I will perform and what information I might make available to the other party. A WCF contract contains very similar information. It contains information that stipulates what a service does and the type of information it will make available.

Contracts in Windows Communication Foundation provide the interoperability they need to communicate with the client. It is through contracts that clients and services agree as to the types of operations and structures they will use during the period that they are communicating back and forth. Without contracts, no work would be accomplished because no agreement would happen.

Instructor Notes:

Explain the lesson coverage

Service Contracts

- A service contract defines the operations that a service supports which are advertised to the consumers of the services.
- This advertisement generally takes place through a schema and a contract definition that supports a standardized method for publishing the service contract
- This schema is either a Web Services Description Language (WSDL) contract or a WS-MetadataExchange (MEX) contract
- Service contracts are implemented as .NET Framework interfaces that are annotated with the ServiceContract attribute

Service Contract: Defines the methods of a service, that is, what operations are available on the endpoint to the client.

A service contract is what informs the clients and the rest of the outside world what the endpoint has to offer and communicate. Think of it as a single declaration that basically states “here are the data types of my messages, here is where I am located, and here are the protocols that I communicate with.”

It also defines the basic message exchange patterns, such as whether the message behaves in a request/reply, one-way, or duplex behavior.

A service contract exposes specific information to the client, which enables the client to understand what the service has to offer. This information includes the following:

❑ The data types in the message

❑ The locations of the operations

❑ The protocol information and serialization format to ensure the proper and successful communication

❑ Operation grouping

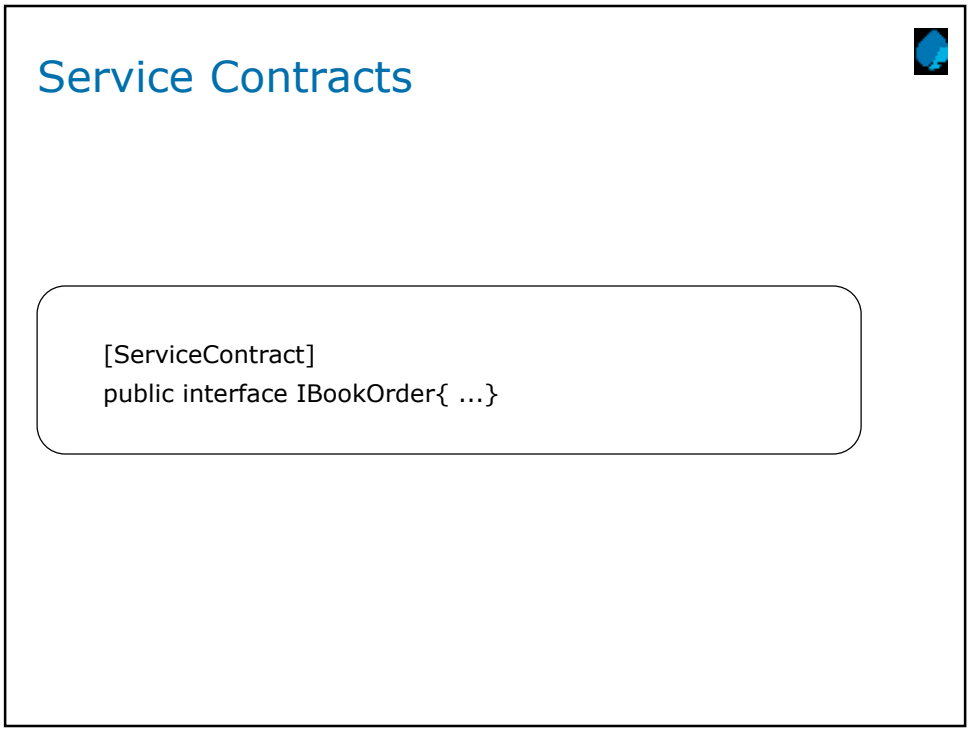
❑ Message exchange pattern (MEPs)

A service contract is defined by simply applying the [ServiceContract] annotation to an interface or class. The following example shows how to define an interface as a service contract:

```
[ServiceContract]
public interface IBookOrder
{
//do some stuff
}
```

Instructor Notes:

Explain the lesson coverage



Service Contract: Defines the methods of a service, that is, what operations are available on the endpoint to the client.

A service contract is what informs the clients and the rest of the outside world what the endpoint has to offer and communicate. Think of it as a single declaration that basically states “here are the data types of my messages, here is where I am located, and here are the protocols that I communicate with.”

It also defines the basic message exchange patterns, such as whether the message behaves in a request/reply, one-way, or duplex behavior.

A service contract exposes specific information to the client, which enables the client to understand what the service has to offer. This information includes the following:

❑ The data types in the message

❑ The locations of the operations

❑ The protocol information and serialization format to ensure the proper and successful communication

❑ Operation grouping

❑ Message exchange pattern (MEPs)

A service contract is defined by simply applying the [ServiceContract] annotation to an interface or class. The following example shows how to define an interface as a service contract:

```
[ServiceContract]
public interface IBookOrder
{
//do some stuff
}
```

Instructor Notes:

Explain the lesson coverage

Operation Contracts

- Operation contracts define the individual operations that a service supports and map to operations in WSDL
- Operations are defined by adding methods to a Service Contract interface that is annotated with the OperationContract attribute

```
[OperationContract]
void CreateOrder(int orderNumber);
```

Service operations are specified by applying the [OperationContract] annotation on the methods of an interface, as illustrated in this example:

```
[OperationContract]
bool PlaceOrder(string orderdate, string bookisbn);

[OperationContract]
bool CheckOrder(int ordernumber);
```

Put these two together to make a complete service contract, as shown here:

```
[ServiceContract]
public interface IBookOrder
{
[OperationContract]
bool PlaceOrder(string orderdate, string bookisbn);
[OperationContract]
bool CheckOrder(int ordernumber);
}
```

Once the interface for the service is defined, the defined interface can then be implemented.

Instructor Notes:

Explain the lesson coverage

Example: Contract

```
[ServiceContract]
public interface IOrderService
{
    [OperationContract]
    void CreateOrder(int orderNumber);

    [OperationContract]
    void AddItemToOrder(int orderNumber, Item itm);

    [OperationContract]
    Order GetOrderDetails(int orderNumber);
}
```

Instructor Notes:

OperationContract

- The OperationContract attribute, along with the ServiceContract attribute, provides greater control over the WSDL generation
- The various properties of OperationContract attribute are:
  - IsInitiating - Indicates that this operation is an initiation of a session
  - IsTerminating - Indicates that this operation terminates the session and the channel should close.
  - IsOneWay - Indicates that the operation returns nothing (void) or can't accept out parameters
  - Name - Overrides the operation name from the method name on the interface

IsInitiating Indicates that this operation is an initiation of a session; the default is true, so if you require session initiation, you need to set all operations to false except the initiation operation.

Instructor Notes:

Explain the lesson coverage

Data Contracts

- Data contracts define how complex types are serialized to and from XML when they are used in WCF service operations
- They are defined by applying the DataContract and DataMember attributes to classes

```
[DataContract]
public class Product
  {
    [DataMember]
     public int ProductID;
}
```

Simply, a data contract describes the data that is to be exchanged.

The data contract specifically defines what each parameter and return type will be serialized/deserialized to and from (Binary and

XML) in order be exchanged from one party to the other.

Data contracts are defined declaratively just like service contracts and operation contracts. To define a data contract, you simply decorate a class or enumeration with the [DataContract] attribute. For example:

```
[DataContract]
public class BuyStock
{
...
}
```

The preceding example defines a data contract. Once you have the data contract defined, you can define the members of a data contract. These members are the fields or properties of your class. Taking the previous example, the following example defines some data members for within the data contract:

```
[DataContract]
public class BuyStock
{
[DataMember] public string symbol;
[DataMember] public int quantity;
[DataMember] public decimal price;
}
```

Once the data contract and its members are defined, you can choose to include that information in a service contract as follows:

```
[ServiceContract]
public interface IBuyStock
{
[OperationContract]
Void BuySomeStock(BuyStock stock);
}
```

Each data contract must have a unique name, but as a default these values are pulled from the class. It is important to also note that each member of the contract must be explicitly defined using the [DataMember] attribute. This helps to ensure that the developer meant to expose the data.

Instructor Notes:

Message Contracts

- Message contract provides additional control over that of a data contract, in that it controls the SOAP messages sent and received by the service
- They can use data contracts and serializable types to emit schema for complex types, and they also make it possible to control the SOAP message headers and body explicitly, by using a single type

Message contracts provide the ultimate control over the formatting of the SOAP messages. In most cases you will not need to employ this level because data contracts provide most of the control you will need. However, should the need arise for this level of customization, message contracts are available.

Probably the biggest reason you would need to obtain this level of control is interoperability. A message contract provides the level of interoperability you would need when you need to communicate with clients or other systems that may use a particular WSDL or schema, or cases when there is some question or concern regarding the SOAP message structure, such as controlling security issues.

What message contracts allow you to do is to have more control over the way contract parameters are formatted in SOAP messages, such as whether or not information goes in the message body or message headers. You should keep in mind that WCF does this automatically, but if you want to override this,you need to use message contracts.

Instructor Notes:

Message Contracts

- Message contracts provide a simple method to add custom SOAP headers to incoming and outgoing messages

```
[MessageContract]
public class MyRequest
{
        [MessageHeader]
        public string field1;

        [MessageBody]
        public string field2;
 }
```

Message contracts are defined by applying the [MessageContract] attribute. You then specify specifics for each message using the [MessageBodyMember] and [MessageHeader] attributes. The following example illustrates these three attributes by defining a simple message contract:

```
[MessageContract]
public class BuyStock
{
[MessageHeader]
Public string Title;
[MessageBodyMember]
Public decimal Cost;
}
```

The [MessageContract] attribute defines a strongly typed class that is associated with a SOAP message.

This in turn creates a typed message, which then enables the passing of messages between services (service operations). It also provides the ability to use custom messages as parameters within these same services (and service operations).

Instructor Notes:

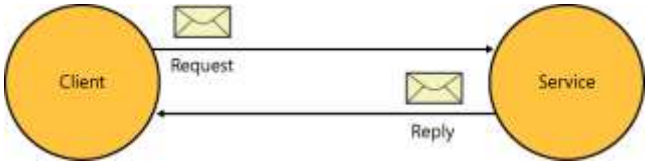
Message Exchange patterns

- WCF service contracts can express three message exchange patterns that can be used in your services.
- Note that the binding that you choose will limit the message exchange patterns that are available

Instructor Notes:

Messaging Patterns

- Request - Reply
  - Two way messaging
  - Every request is paired with a response



Two-way communication in which a request is paired with a reply is the request-reply messaging pattern. A client makes a request and waits for a reply. Figure shows a client and a service communicating in request-reply fashion.

Instructor Notes:

Simplex / One Way

- Permits a "fire and forget" style

Simplex communication permits a "fire-and-forget" style of messaging, where the message is transmitted without waiting for a

reply from the service. This is similar to calling an asynchronous method (a delegate) with a void return type.

Instructor Notes:

Duplex

- Duplex
  - Two way messaging
  - Either side can communicate freely

The duplex messaging pattern is two-way communication in which the two sides can be communicating freely in either direction without synchronization of any kind.

Instructor Notes:

Channels

- The channels are responsible for transferring a message from the client side to the service side, and vice versa, depending on the messaging exchange pattern used
- On the client side, channels are created by factories so that they can talk to the service side across a specific channel or set of channels.
- On the service side, listeners accept messages from channels
- Bindings make it much easier for the client and service sides to work together

The address and binding together manifest themselves in the messaging layer of WCF. The address expresses where the message should go, and the binding is the model you use to manipulate the message.

The binding is the mechanism that controls the channel stack. When you pick one of the predefined bindings mentioned earlier, you are picking, in essence, specific channels.

Channels have input and output handlers that are responsible for consuming messages. Consuming messages can mean forwarding the messages across a certain transport, or receiving messages across a certain transport through a specific messaging exchange pattern. The channel also applies security and performs validations.

Instructor Notes:

Explain the lesson coverage

Example: Service

```
public class OrderService : IOrderService
{
    void CreateOrder(int orderNumber)
    {
        // implementation details
    }
    void AddItemToOrder(int orderNumber, Item itm)
    {
        // implementation details
    }
    Order GetOrderDetails(int orderNumber)
    {
        // implementation details
    }
}
```

Instructor Notes:

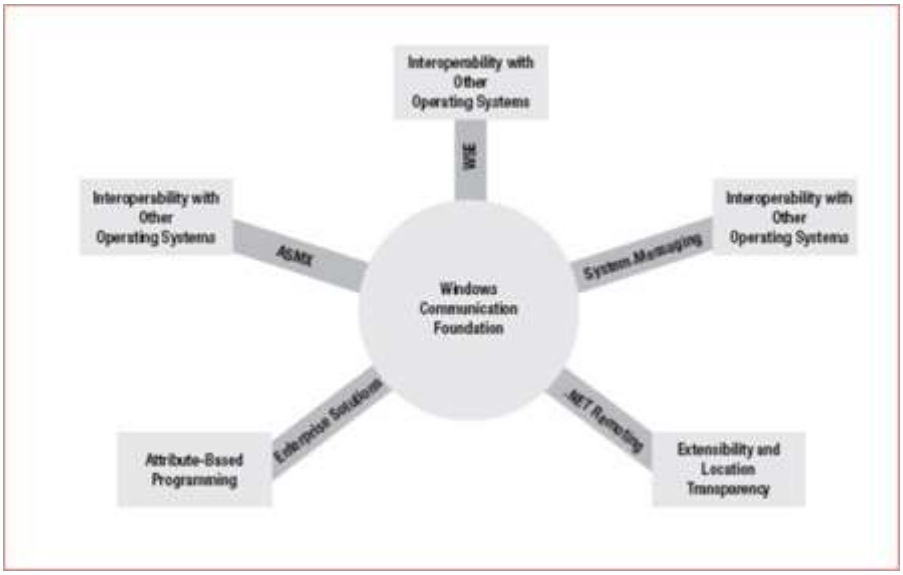
None

Demonstration

- Creating a WCF Service by defining Service Contract and Implementing the contract

Instructor Notes:

WCF 3.0

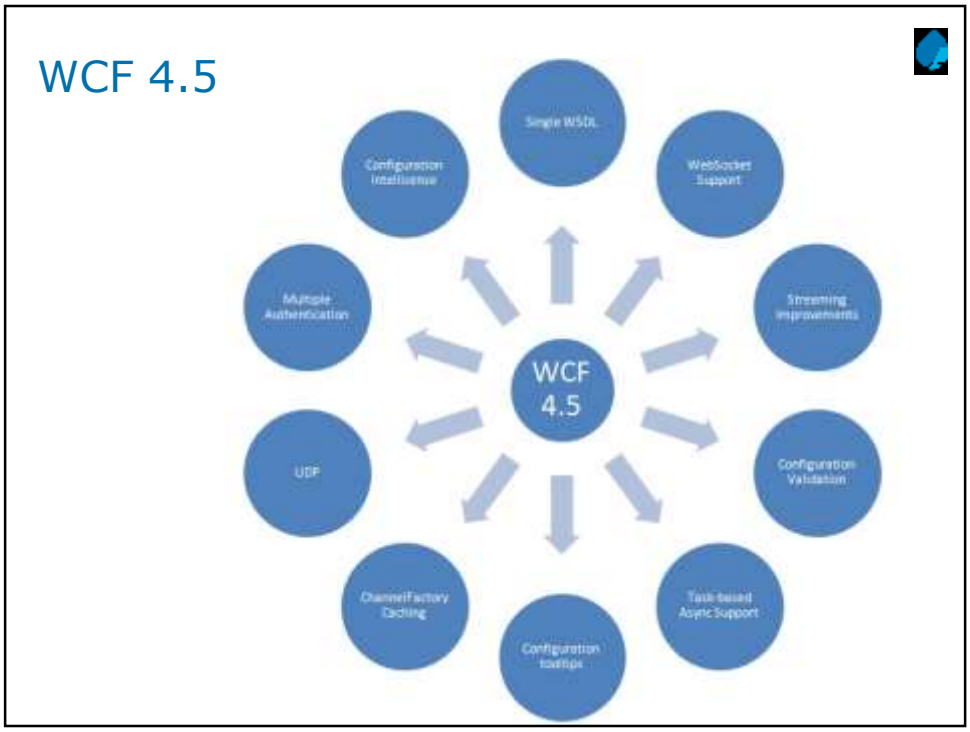


Instructor Notes:

WCF 4.0

Feature Area	Description
Simplified Configuration	Simplification of the WCF configuration section through support for default endpoints, binding and behavior configurations. These changes make it possible to host configuration-free services, greatly simplifying the developer experience for the most common WCF scenarios.
Discovery	New framework support for both ad hoc and managed service discovery behaviors, which conform to the standard WS-Discovery protocol.
Routing Service	New framework support for a configurable routing service that you can use in your WCF solutions. Provides features for content-based routing, protocol bridging, and error handling.
REST Improvements	Enhancements to WCF WebHttp Services with some additional features and tooling that simplify REST service development.
Workflow Services	Rich framework support for integrating WCF with WF to implement declarative long-running workflow services. This new programming model gives you the best both frameworks have to offer (WCF & WF).

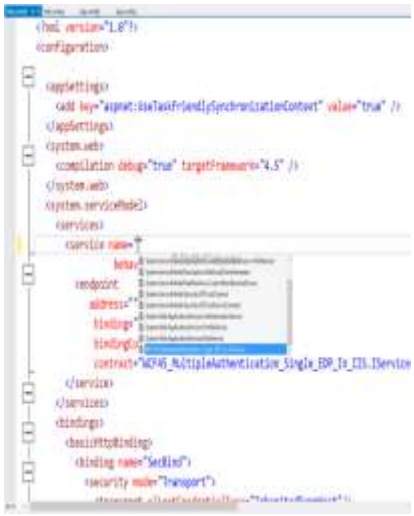
Instructor Notes:



Instructor Notes:

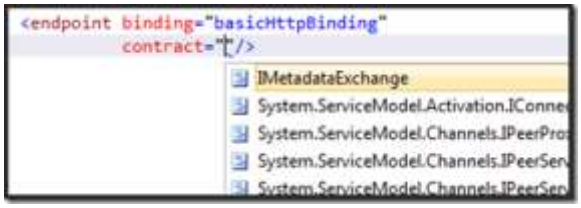
Improved Intellisense

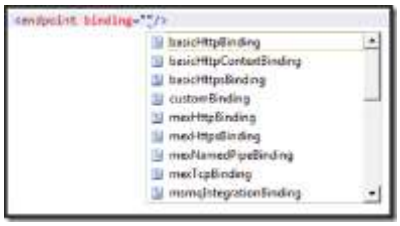
- With WCF 4.5, Visual Studio 2012 provides significantly improved Intellisense support in configuration files too.
- For example as we can see below we have auto completion of Service Names



Instructor Notes:

Improved Intellisense





Instructor Notes:

Simplified Generated Configuration Files

- When you add a service reference in Visual Studio or use the SvcUtil.exe tool, a client configuration file is generated. In previous versions of WCF these configuration files contained the value of every binding property even if its value is the default value.
- In WCF 4.5 the generated configuration files contain only those binding properties that are set to a non-default value.

```
<?xml version="1.0" encoding="utf-8"?>
<configuration>
  <system.serviceModel>
    <bindings>
      <basicHttpBinding>
        <binding name="BasicHttpBinding_IService1" />
      </basicHttpBinding>
    </bindings>
    <client>
      <endpoint address="http://localhost:36906/Service1.svc" binding="basicHttpBinding"
bindingConfiguration="BasicHttpBinding_IService1" contract="IService1"
name="BasicHttpBinding_IService1" />
    </client>
  </system.serviceModel>
</configuration>
```

Instructor Notes:

WCF Configuration Validation

- As part of the build process within Visual Studio, WCF configuration files are now validated. A list of validation errors or warnings are displayed in Visual Studio if the validation fails.

Instructor Notes:

WCF Configuration

- Also the configuration errors come up in the Tasks window as warnings:

Instructor Notes:

Single WSDL

- For WCF 4 and previous versions, the schema part in the WSDL is a set of import directives for additional files:

```
<xsd:schema targetNamespace="http://tempuri.org/Imports">
  <xsd:import schemaLocation="http://localhost:8732/Design Time Addresses/TestMEX/Service1/?xsd=xsd0"
              namespace="http://tempuri.org/"/>
  <xsd:import schemaLocation="http://localhost:8732/Design Time Addresses/TestMEX/Service1/?xsd=xsd1"
              namespace="http://schemas.microsoft.com/2003/10/Serialization/"/>
  <xsd:import schemaLocation="http://localhost:8732/Design Time Addresses/TestMEX/Service1/?xsd=xsd2"
              namespace="http://schemas.datacontract.org/2004/07/TestMEX"/>
</xsd:schema>
```

Instructor Notes:

Single WSDL

Instructor Notes:

Single WSDL

- In addition to the ?wsdl option, we now get another option - ?singleWsdl. The singleWsdl link will return a single WSDL file, containing all the schemas in it:

Instructor Notes:

Additional notes for instructor

Summary

- WCF 4.5 becomes easier to use through the simplified configuration model and better support for common defaults.

Add the notes here.

Instructor Notes:

Answers for the Review Questions:Service Contract Data Contract Message Contract
True

Review Questions

- Question 1: What are the three types of Contract?
  - Service Contract
  - Data Contract
  - Message Contract
  - WCF Contract
- Question 2: Simplex communication permits a "fire-and-forget" style of messaging.
  - True/False

Add the notes here.