Introduction to WCF Programming in Delphi

By: Pawel Glowacki

Abstract: The "Introduction to WCF Programming in Delphi" session starts from WCF object model basics, including message-based communication stack, service contracts, endpoints, transports, exchange patterns, and interoperability with Web Services. After explaining basics Paweł is going to demonstrate building simple WCF services and clients with Delphi for .NET compiler, and consuming WCF services with Delphi for Win32 Web Services clients.

Introduction to WCF Programming in Delphi

by Pawel Glowacki, CodeGear Technical Evangelist for Benelux/Italy/Spain/Portugal, August 2007

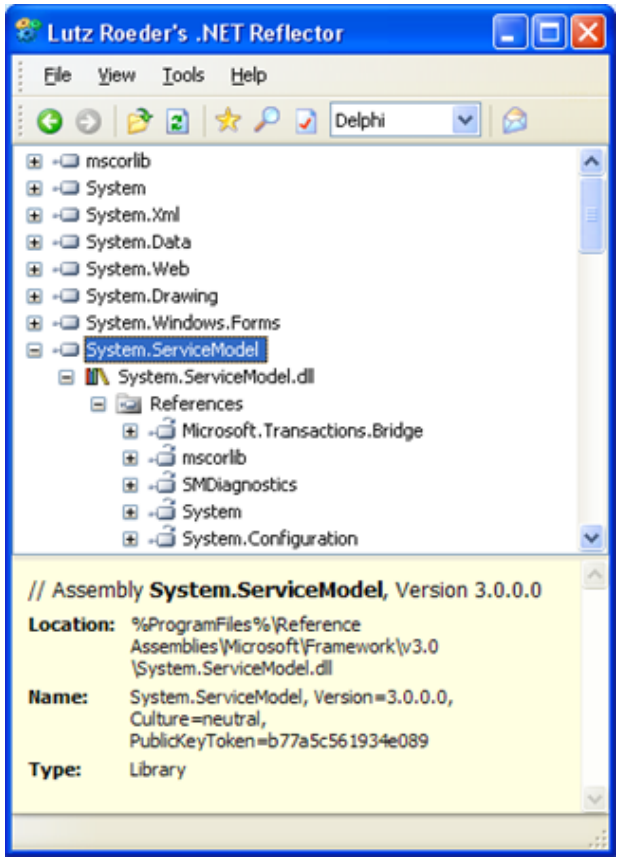
Abstract

The "Introduction to WCF Programming in Delphi" session starts from WCF object model basics, including message-based communication stack, service contracts, endpoints, transports, exchange patterns, and interoperability with Web Services. After explaining basics Pawel is going to demonstrate building simple WCF services and clients with Delphi for .NET compiler, and consuming WCF services with Delphi for Win32 Web Services clients.

Introduction to WCF

Windows Communication Foundation (WCF), formerly code-named "Indigo", is a part of the .NET Framework version 3.0, and provides new architecture for building distributed messaging applications. The other 3.0 technologies are Windows Presentation Foundation (WPF), formerly code-named "Avalon", Windows Workflows (WF) and Windows CardSpace. Technically speaking .NET 3.0 is a repackaged version of .NET 2.0 plus a set of technologies formerly code-named "WinFX". Inspecting System.ServiceModel.dll assembly, which implements the WCF, reveals that even though its version number is 3.0.0.0, it uses .NET system assemblies version 2.0.0.0.

Hide image

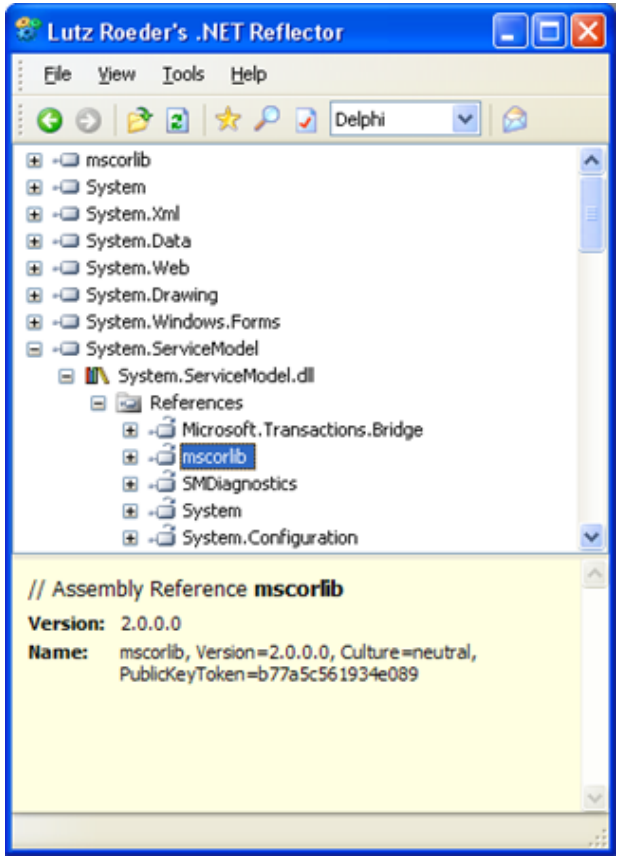


Hide image

ARTICLE TAGS

Delphi WCF

SHARE THIS



This means that all .NET 3.0 assemblies are compatible with .NET 2.0, and 2.0 is the .NET Framework version that is supported by Delphi 2007 for .NET, code-named "Highlander".

Message-based Communication Stack

Windows Communication Foundation aims at providing a uniform object model for interprocess communication on the same machine and over a network through abstracting away the underlying communication protocol. One can think of the WCF as the natural evolution of Web Services Enhancements (WSE). The WSE was an engine for applying advanced Web service protocols to SOAP messages. The WSE added to ASP.NET Web Services the support for different WS-* protocols, including WS-Security, WS-SecureConversation, WS-Trust, and WS-Addressing. That is why SOAP is native protocol to WCF. All messages that are exchanged between clients and services are represented as SOAP envelopes that could possibly be serialized and encoded in different ways. Windows Communication Foundation object model goes further and decouples underlying transport protocol from different SOAP envelopes exchange patterns. WCF supports HTTP(S), TCP, MSMQ and Named Pipes communication protocols, and request/reply, one way and duplex message exchange patterns.

The elegance of the WCF lies in its clean, layered architecture. Due to the fact of coming from Web Services background, it is not surprising that different WCF concepts can be very easily related to the structure of the Web Services Description Language document. The **WSDL Specification** (http://www.w3.org/TR/wsdl.html) describes the following elements in the definition of network services:

- **Types** – a container for data type definitions used to describe the messages exchanged using some type system (such as XML Schema).

Message – an abstract, typed definition of the data being communicated. A message consists of logical parts, each of which is associated with type definitions

- **Operation** – an abstract description of an action supported by the service. Each operation refers to an input message and output messages.
- **Port Type** – an abstract set of operations supported by one or more ports.
- **Binding** – a concrete protocol and data format specification for the operations and messages defined by a particular port type.
- **Port** – a single endpoint defined as a combination of a binding and a network address.
- **Service** – a set of related ports

Going from the bottom to top, the WSDL Specification describes "services" as a set of ports. Port is the combination of a network address and a binding, which is a concrete communication protocol. Although the WSDL specification does not specify any particular communication protocol, in the early days of Web Services, this protocol defaulted to HTTP. The HTTP is the standard protocol for network communication between web servers and browsers. The HTTP is basically a text based protocol for exchanging text messages over the TCP protocol. In case of web browsers and servers these text messages are typically HTTP requests and HTML documents, and in case of web services these are SOAP envelopes containing request, response or fault messages. Both HTML and SOAP messages are ultimately XML documents (or more precisely XML "infosets"). The WCF object model supports HTTP and HTTPS protocols, but also TCP, MSMQ, and Named Pipes for efficient interprocess communication between processes running on the same machine. WCF object model defines a number of standard bindings that are using certain combinations of transport protocols, encodings, and provide different capabilities in terms of security, sessions-support, reliability, transaction support and available message exchange patterns.

WCF provides the following standard bindings:

- **BasicHttpBinding** - A binding that is suitable for communicating with WS-Basic Profile conformant Web Services, for example, ASMX-based services. This binding uses HTTP as the transport and Text/XML as the default message encoding.
- **WSHttpBinding** - A secure and interoperable binding that is suitable for non-duplex service contracts.
- **WSDualHttpBinding** - A secure and interoperable binding that is suitable for duplex service contracts or communication through SOAP intermediaries.
- **WSFederationHttpBinding** - A secure and interoperable binding that supports the WS-Federation protocol, enabling organizations that are in a federation to efficiently authenticate and authorize users.
- **NetTcpBinding** - A secure and optimized binding suitable for cross-machine communication between WCF applications.
- **NetNamedPipeBinding** - A secure, reliable, optimized binding that is suitable for on-machine communication between WCF applications.
- **NetMsmqBinding** - A queued binding that is suitable for cross-machine communication between WCF applications.
- **NetPeerTcpBinding** - A binding that enables secure, multi-machine communication.
- **MsmqIntegrationBinding** - A binding that is suitable for cross-machine communication between a WCF application and existing MSMQ applications.

These predefined WCF bindings provide a variety of choices for communication in distributed applications. The WSDL concept of port corresponds to the WCF endpoint, which puts together the three things: A, B and C. "A" stands for "Address", and answers the question "where?". This is just an URI that describes where service is physically available. "B" is one of the predefined bindings and tells "how?" to communicate with the service, and "C" is the contract, that describes "what?" the service provides. The "contract" corresponds to a "port type" from the WSDL specification, which groups an abstract set of operations provided by a service. The service contract corresponds to an interface definition in a programming language. The individual operation in a contract corresponds to a WSDL operation, or in programming language, to a method or function. Operations refer to input and output messages, that are simply input and output function argument types. The ultimate type system in WSDL and WCF is the XML Schema.

Hosting Options

The WCF object model is hosting agnostic. This means that you can host WCF-based services in any Windows process, including a console application, a VCL.NET Forms, or a Windows Presentation Foundation (WPF) UI application. You can even self-host WCF services in long-running Windows NT services that run in the background working on behalf of a configured identity. WCF services with HTTP-based endpoints can also be hosted inside IIS, much like the traditional Web services as implemented by ASP.NET and ASMX. With IIS 7.0 it is possible to use a new IIS feature called the Windows Process Activation Service (WAS). WAS is a fundamental underlying component of IIS 7.0 that makes it possible to host WCF services beyond HTTP and without having to install the whole IIS package.

- Self-hosting (any Windows process)
- Windows Service
- Internet Information Services (HTTP-only)
- Windows Activation Service (requires IIS 7.0)

Message Exchange Patterns

A service contract specifies what an endpoint communicates to the outside world. At a more concrete level it is a statement about a set of specific messages organized into basic message exchange patterns (MEPs) such as:

- request/reply
- one-way
- duplex

The Anatomy of the WCF Service

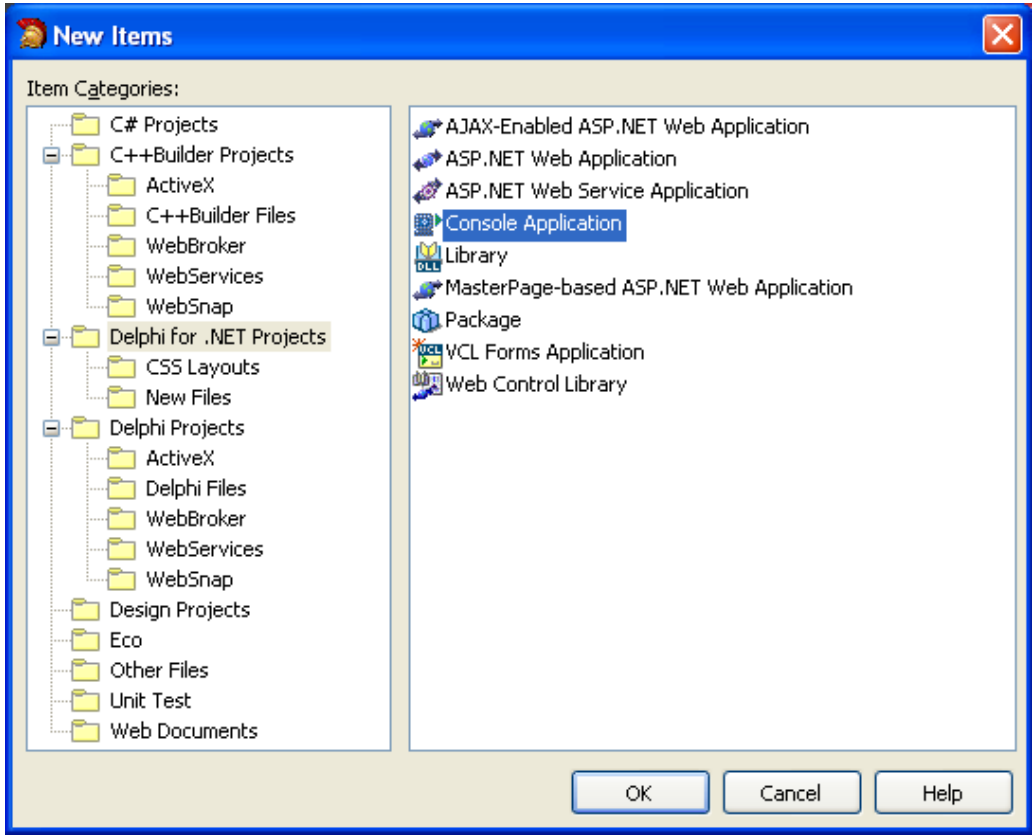
A WCF Service is composed of three parts — a Service class that implements the service to be provided, one or more endpoints to which clients connect, and host environment to host the service. Clients communicate with the WCF service only via the endpoints. The endpoints specify a Contract that defines which methods of the Service class will be accessible via the endpoint; each endpoint may expose a different set of methods. The endpoints also define a binding that specifies how a client will communicate with the service and the address where the endpoint is hosted.

Delphi for .NET Simple Calculator WCF Service Example

Before we can proceed and build simple WCF example application, we need to make sure that .NET 3.0 Framework is installed on our development machine. If we are using Windows Vista, then we do not need to do anything, because .NET 3.0 is preinstalled. If we are on Windows XP, then we need to install .NET 3.0 **redistributable package**. The other platforms that supports WCF are Windows Server 2003 and Windows "Longhorn" Server.

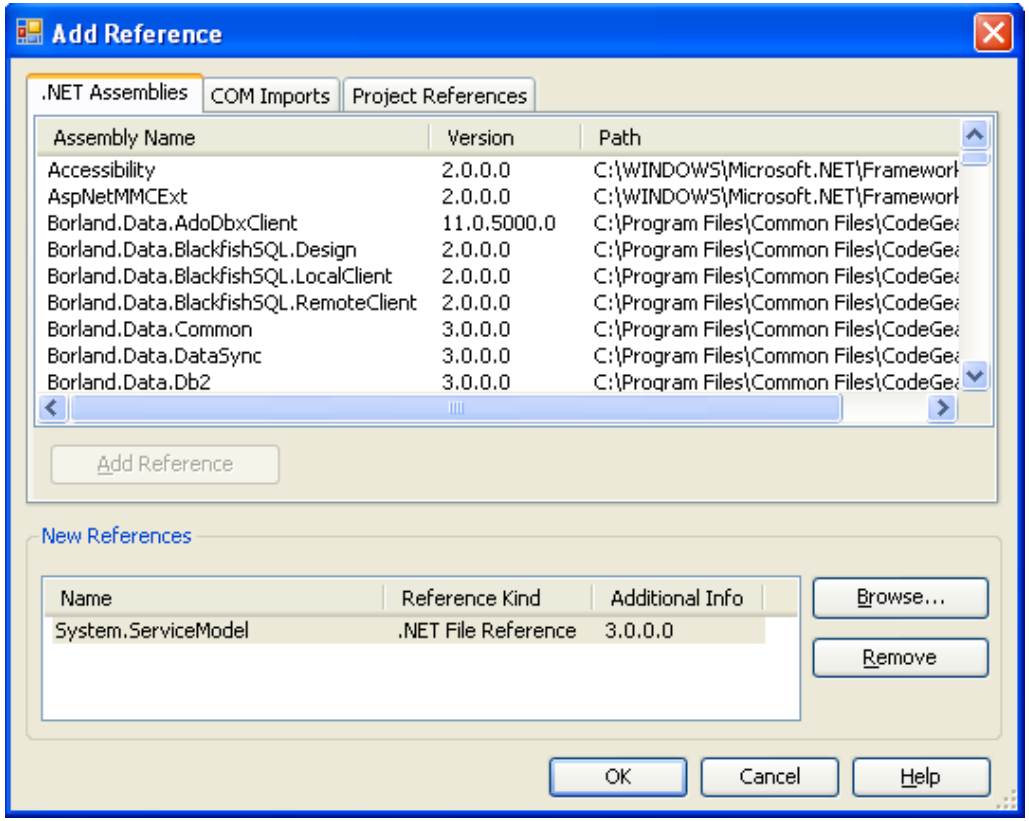
In this example we are going to build a simple calculator service, that will provide two methods, "Add" and "Subtract", both accepting two integers, and returning an integer result. For this we are going to create a new Delphi for .NET Console Application project, and save it as "DelphiWCFSimpleCalcService".

Hide image



Before we start writing code, first we need to add to the project a reference to "System.ServiceModel.dll" .NET assembly, where most of WCF types are defined. For this right-click on "References" node in Project Manager window, or select "Add Reference" from "Project" menu.

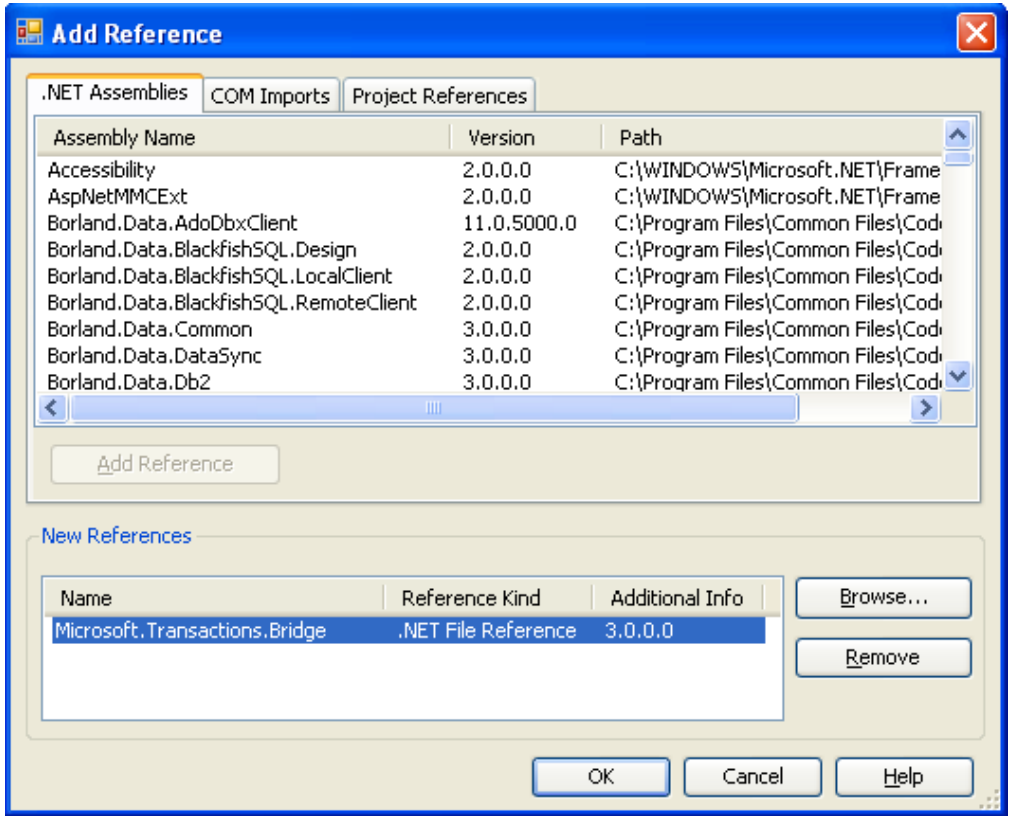
Hide image



In the "Add Reference" dialog click on "Browse" button and add to the project a reference to "System.ServiceModel.dll" located in typical installation in "C:\Program Files\Reference Assemblies\Microsoft\Framework\v3.0" folder. If we try to compile the application at this stage we are going to receive compiler error that "Microsoft.TransactionBridge.dll" cannot be found. This assembly is used internally by "System.ServiceModel.dll" and it is located in a different directory. That is why we need to add to the project a reference to this assembly as well.

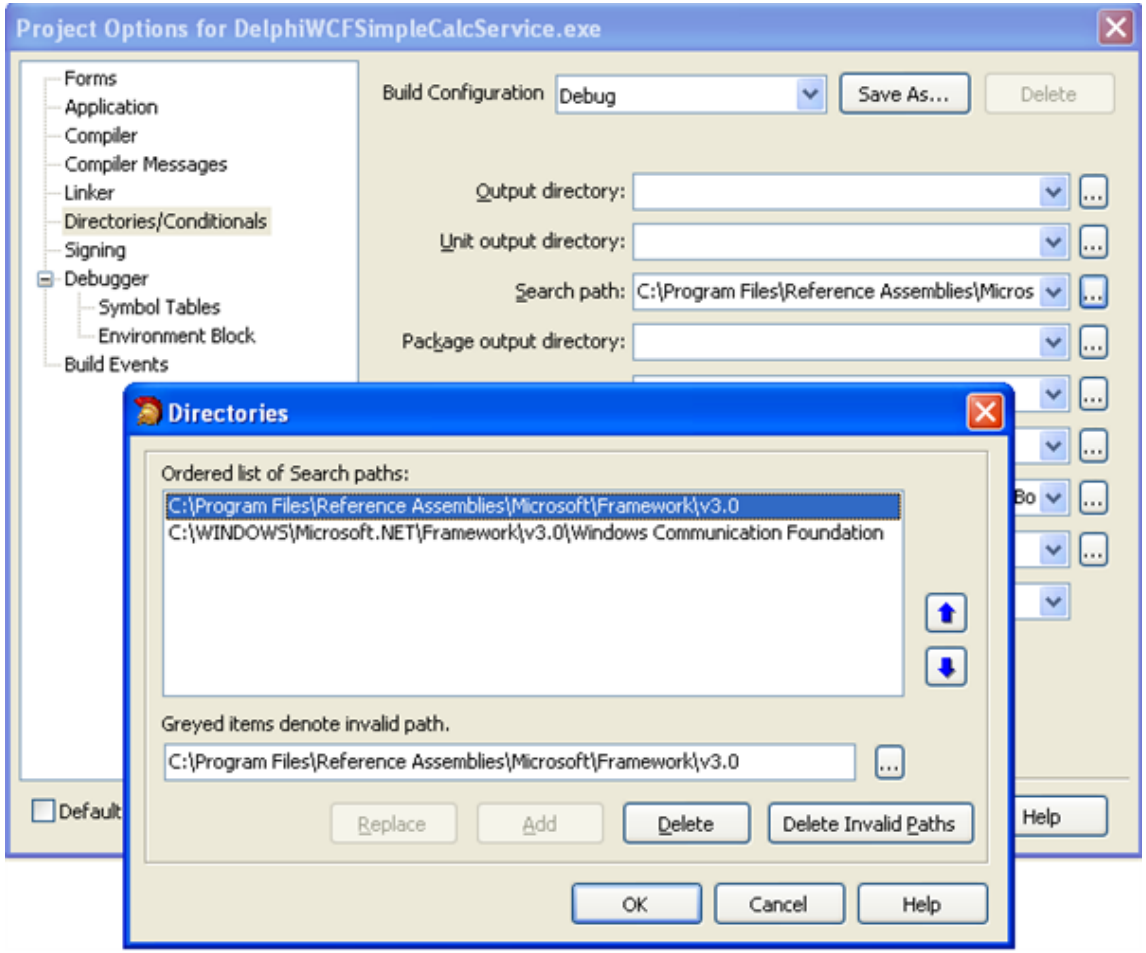
In the "Add Reference" dialog select "Microsoft.TransactionBridge.dll" located in "C:\WINDOWS\Microsoft.NET\Framework\v3.0\Windows Communication Foundation" folder.

Hide image



At this stage our empty console application should compile fine. Just click on the "Run" button, or press "Ctrl-F9", to verify that the project compiles successfully. The side effect of adding these two .NET assemblies to project references was adding two directories where these assemblies are located to project search path so the compiler can also find other dependant .NET assemblies that are not added to the project explicitly. Clicking on "Directories/Conditionals" in "Project Options" dialog shows that two folders were added to the project "Search Path".

Hide image



Now we are ready to start coding. We will start from defining the interface of our service, which defines our Simple Calculator service contract. Let's add an empty unit to the project, and give it a name "IntfSimpleCalc", which will contain the definition of "ISimpleCalc" interface with two functions: "Add" and "Subtract". To tell the WCF runtime that this interface is the WCF service contract, we need to decorate the interface definition with "ServiceContract" custom attribute, and both functions with "OperationContract" attributes. These two attributes are defined in "System.ServiceModel" namespace, so we also need to add this namespace to the "uses" clause of the unit. These attributes serve a similar role as "WebService" and "WebMethod" attributes in traditional ASP.NET Web Services.

```
unit IntfSimpleCalc;

interface

uses
  System.ServiceModel;

type
  [ServiceContract]
  ISimpleCalc = interface

    [OperationContract]
    function Add(a, b: integer): integer;

    [OperationContract]
    function Subtract(a, b: integer): integer;
```

```
  end;

implementation

end.
```

Listing 1: *Simple Calculator WCF Service Contract definition.*

Now we need to define a class that will implement the "ISimpleCalc" interface. For this let's add another empty unit to the project. It will contain "TSimpleCalc" class, with implementation of "Add" and "Subtract" methods.

```
unit ImplSimpleCalc;

interface

uses
  IntfSimpleCalc;

type
  TSimpleCalc = class(TObject, ISimpleCalc)
  public
    function Add(a,b: integer): integer;
    function Subtract(a,b: integer): integer;
  end;

implementation

{ TSimpleCalc }

function TSimpleCalc.Add(a, b: integer): integer;
begin
  Result := a + b;
end;

function TSimpleCalc.Subtract(a, b: integer): integer;
begin
  Result := a - b;
end;

end.
```

Listing 2: *Simple Calculator WCF Service Implementation.*

We have already defined our simple calculator service contract and implementation. Because we are going to host our service inside a .NET console application, we need to add some code to host our implementation object, and define an endpoint for clients to be able to communicate with the service implementation. The "System.ServiceModel" namespace provides "ServiceHost" class that is designed just for this. The "ServiceHost" constructor accepts the type of the implementation object, and a dynamic array of URIs, that contain one or more base addresses to connect to the service. The ServiceHost class will internally instantiate our implementation class. After ServiceHost is created, we need to add to it a service endpoint, and then call "ServiceHost.Open" to start waiting for client calls, and "ServiceHost.Close" just before the application terminates.

```
var
  sh: ServiceHost;
  URIs: array of Uri;

begin
  SetLength(uris, 1);
  URIs[0] := Uri.Create('http://localhost:8000/DelphiSamples/');

  sh := ServiceHost.Create(typeof(TSimpleCalc), URIs);
```

To add a service endpoint to a service host instance we can use its "AddServiceEndpoint" that accepts three parameters. The first parameter is the contract, which is just service interface type. The second is binding. We are going to use "BasicHttpBinding" which provides interoperability with standard web services clients, and the third parameter is a relative address for this particular endpoint within the WCF service.

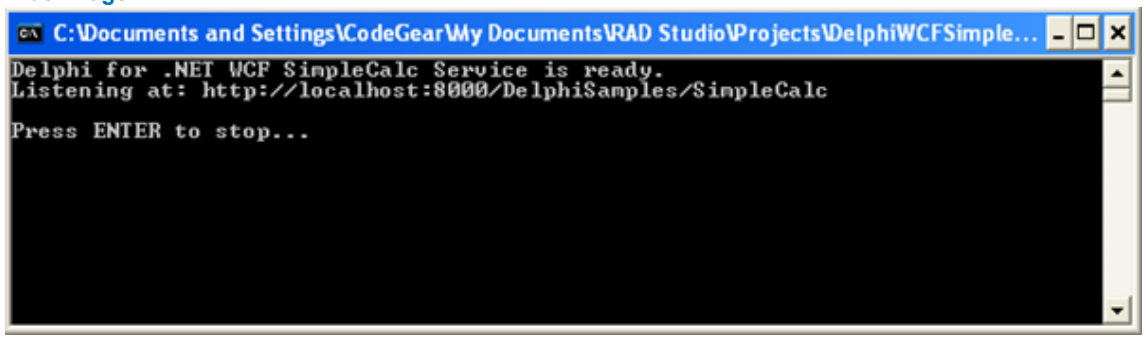
```
  sh.AddServiceEndpoint( typeof(ISimpleCalc), BasicHttpBinding.Create, 'SimpleCalc');
```

This is all it takes to create a simple WCF service application hosted inside a .NET console application. If everything went well you should be able to run the application. In the listing below is the whole application code for creating service host, adding an endpoint, and waiting for incoming client requests. If you press any key, the service application will terminate.

```
program DelphiWCFSimpleCalcService;

{$APPTYPE CONSOLE}

uses
  SysUtils,
  IntfSimpleCalc in 'IntfSimpleCalc.pas',
  ImplSimpleCalc in 'ImplSimpleCalc.pas',
  System.ServiceModel;

var
  sh: ServiceHost;
  URIs: array of Uri;
  aBaseAddress, aRelativeAddress: string;

begin
  try
    aBaseAddress := 'http://localhost:8000/DelphiSamples/';
    SetLength(uris, 1);
    URIs[0] := Uri.Create(aBaseAddress);

    sh := ServiceHost.Create(typeof(TSimpleCalc), URIs);

    aRelativeAddress := 'SimpleCalc';
    sh.AddServiceEndpoint( typeof(ISimpleCalc), BasicHttpBinding.Create,
      aRelativeAddress);

    sh.Open;

    writeln('Delphi for .NET WCF SimpleCalc Service is ready.');
    writeln('Listening at: ' + aBaseAddress + aRelativeAddress);
    writeln;
    writeln('Press ENTER to stop...');
    readln;

    sh.Close;

  except
    on E:Exception do
    begin
      writeln(E.Classname, ': ', E.Message);
      readln;
    end;
  end;
end.
```

Listing 3: *Simple Calculator WCF Service console application.*

Hide image



Our service application is now ready. In this example we have defined an endpoint in code, but it is also very common to define endpoints in a .NET configuration file.

Delphi for .NET Simple Calculator WCF Service Console Client application

And now we are going to create client application for our service. To keep things simple let's add to the service application project group a new Delphi for .NET console application. The new project will be called "DelphiWCFSimpleCalcClient" and the project group "DelphiWCFSimpleCalcGrp". Before we can start coding similarly to the service application we need to add to project references to WCF assemblies as described earlier. To make sure that references were added correctly select "Build All Projects" from "Project" menu.

In the client application code we need to create a channel that is of interface type of the calculator service. One creates a channel using "ChannelFactory" generic class defined in the "System.ServiceModel" namespace and parameterized by the service contract interface type. The ChannelFactory constructor expects a binding and endpoint address as parameters. After ChannelFactory instance is constructed, we can call its "CreateChannel" method that returns a proxy that implements service contract interface. The full source code for the client project is listed below.

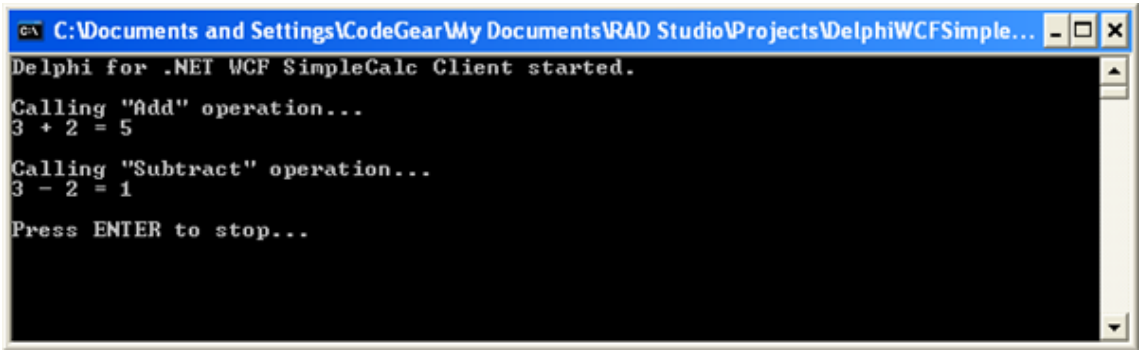
```
program DelphiWCFSimpleCalcClient;

{$APPTYPE CONSOLE}

uses
  SysUtils,
  System.ServiceModel,
  System.ServiceModel.Channels,
  IntfSimpleCalc;

var
  aEndpointAddress: EndpointAddress;
  aBinding: BasicHttpBinding;
  aFactory: ChannelFactory<ISimpleCalc>;
  aChannel: ISimpleCalc;
  i: integer;

begin
  try
    aEndpointAddress := EndpointAddress.Create(
        'http://localhost:8000/DelphiSamples/SimpleCalc');
    aBinding := BasicHttpBinding.Create;
    aFactory := ChannelFactory<ISimpleCalc>.Create(aBinding, aEndpointAddress);
    aChannel := aFactory.CreateChannel;
    try
      Writeln('Delphi for .NET WCF SimpleCalc Client started.');
      Writeln;

      Writeln('Calling "Add" operation...');
      i := aChannel.Add(3,2);
      Writeln('3 + 2 = ' + IntTostr(i));
      Writeln;

      Writeln('Calling "Subtract" operation...');
      i := aChannel.Subtract(3,2);
      Writeln('3 - 2 = ' + IntTostr(i));
      Writeln;

      Writeln('Press ENTER to stop...');
      Readln;

    finally
      IChannel(aChannel).Close;
      IDisposable(aChannel).Dispose;
    end;

  except
    on E:Exception do
    begin
      Writeln(E.Classname, ': ', E.Message);
      Readln;
    end;
  end;
end.
```

Now we are ready to test communication between both: service and client applications. For this we are going to select "Build All Projects" from the "Project" menu.

Our service is self-hosted which means that it has to be manually started before client application to wait for incoming requests.

For this we need to launch service application first, and then client application.

Hide image

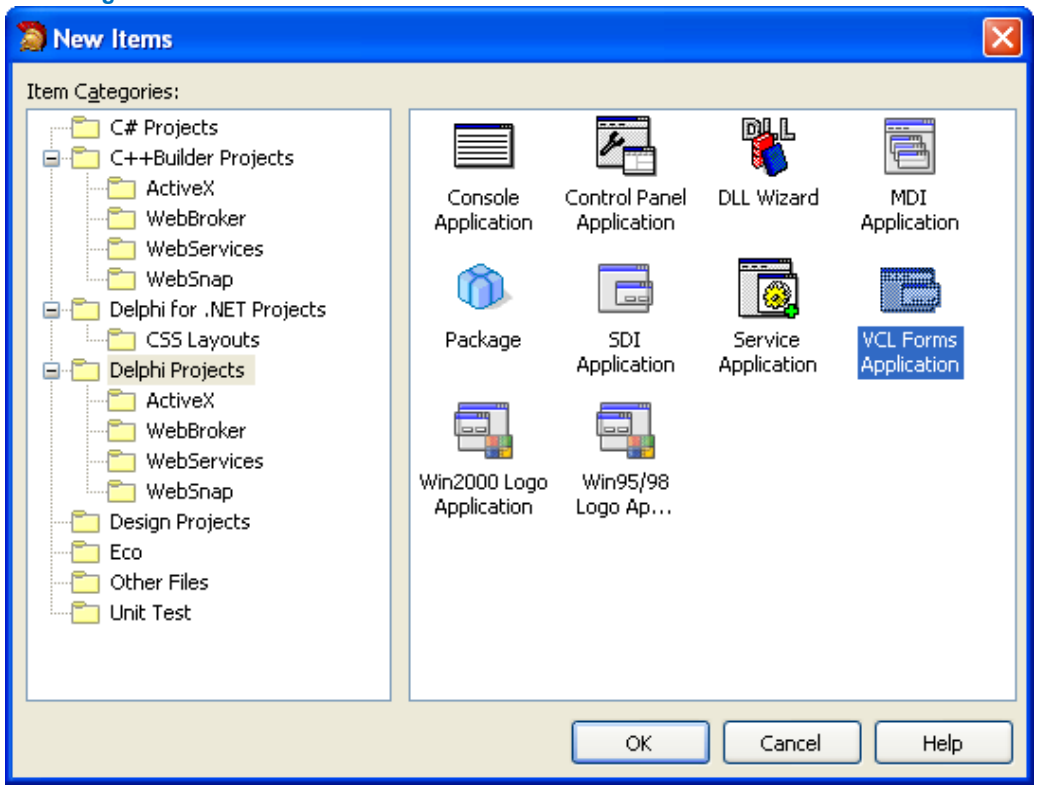


Consuming WCF Service from Delphi for Win32 Web Services client

So far we have managed to create simple, self-hosted WCF service and a .NET console WCF client. Now it is time to demonstrate interoperability between WCF services and arbitrary Web Service client application. Maybe not everybody remembers that Delphi 6 was the first IDE on the market to support Web Services development. We are going to create Delphi 2007 for Win32 VCL Forms Application client for our SimpleCalc service.

Let's add to our project group the third application, and save it as "DelphiWin32SimpleCalcWSClient".

Hide image



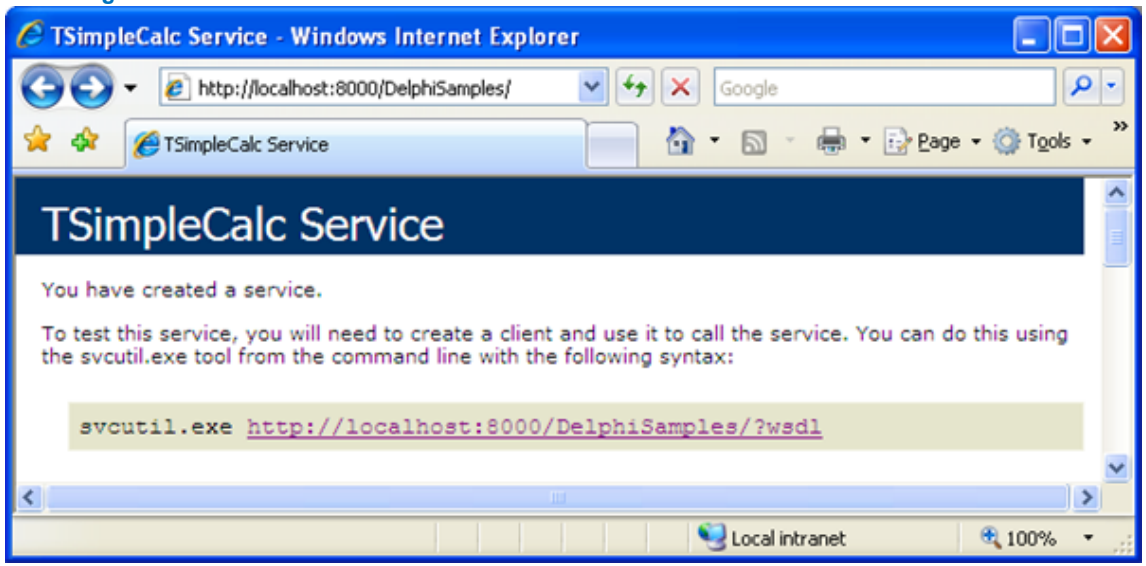
Before client can communicate with a service there must be a piece of shared knowledge between service and client. In case of the previous example this piece was the "IntfSimpleCalc" unit with "ISimpleCalc" interface definition. In case of Web Services this piece of shared knowledge is a WSDL document that is used to generate client proxy code. In case of Delphi for Win32 clients, this WSDL file serves as input for the "WSDL Importer" that generates a Delphi unit with all the code that is needed to communicate with the Web Service application.

WCF Services do not expose a WSDL by default. In order to get WSDL definition we need to add to the service application a MEX endpoint, that implements "IMetadataExchange" interface. In order for the WCF service to expose metadata, the ServiceHost object needs to have a "ServiceMetadataBehaviour" applied to it. Behaviours can be added to a service through code, through custom attributes or via configuration file. Through behaviours we can control different runtime characteristics of the service, like instancing, concurrency, transaction support, and other. In our case we are going to add this behaviour through few lines of code:

```
uses
  System.ServiceModel.Description;

// ...

var
  aBehavior: ServiceMetadataBehavior;

// ...

    aBehavior := ServiceMetadataBehavior.Create;
    aBehavior.HttpGetEnabled := True;
    sh.Description.Behaviors.Add(aBehavior);

    sh.AddServiceEndpoint(
      typeof(IMetadataExchange),
      MetadataExchangeBindings.CreateMexHttpBinding,
      'mex');
```

Now let's run our service application again. If we type in the service address in the web browser, we are going to see the "TSimpleCalc" service test page.

Hide image

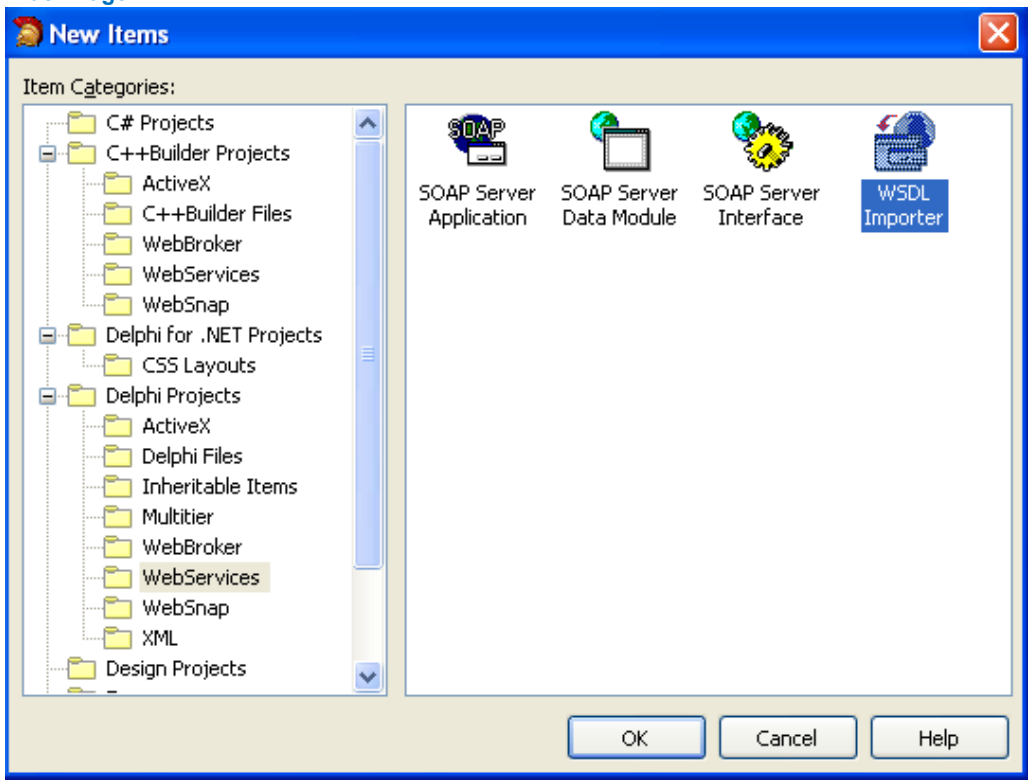


The Windows SDK comes with "svcutil" tool that can generate c# or vb.net source code with definitions of proxies used to talk to the WCF web service. In our case we are going to use Delphi WSDL Importer (wsdlimp.exe) to generate Delphi code. The

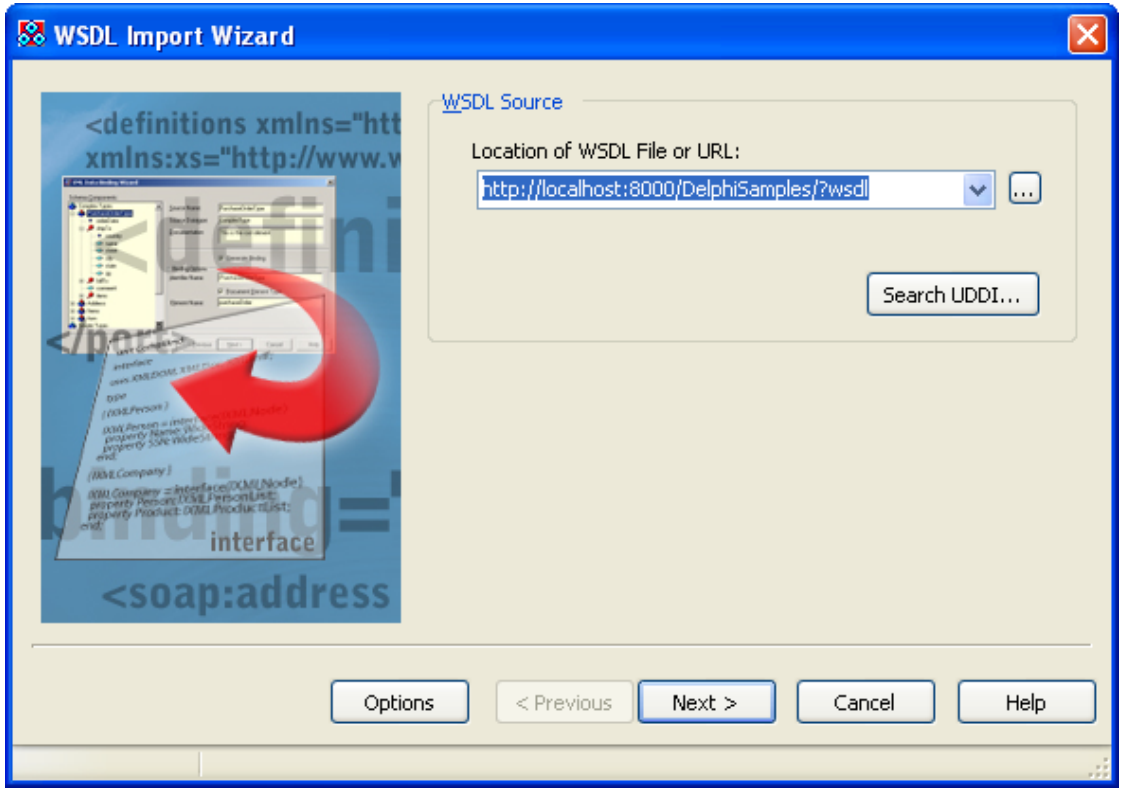
"TSimpleCalc Service" test page contains a link to a WSDL document URL that we are going to use in the WSDL importer wizard (http://localhost:8000/DelphiSamples/?wsdl).

For this we need to make sure that Delphi for Win32 project is active in the project group, and then we need to select "File -> New -> Other" and then "WSDL Importer" icon under "Delphi Projects -> Web Services" node inside "New Items" dialog.

Hide image



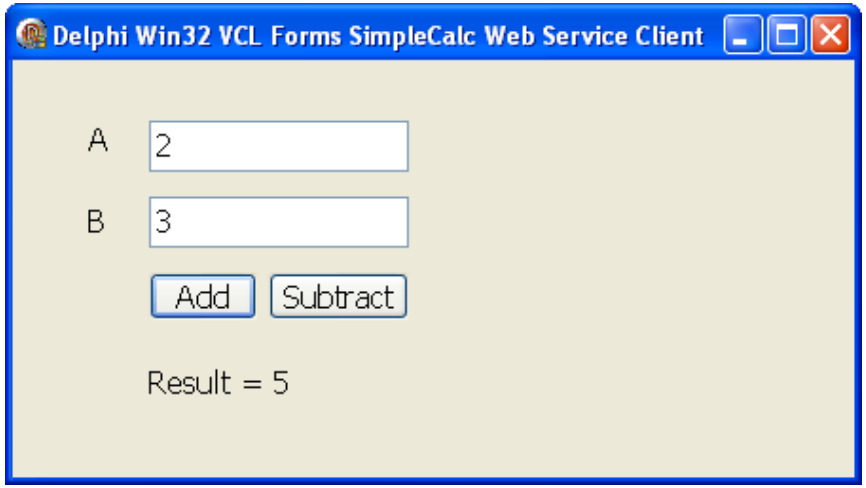
Hide image



The wizard will generate a unit with "ISimpleCalc" interface definition and "GetISimpleCalc" global function for accessing the service. After changing the default unit name generated by the wizard from "_" to more meaningful "WSProxy" we are ready to write some code in the application main form for accessing our TSimpleCalc service. Below is the event handler code the "Add" button in our test VCL Forms application.

```
uses WSProxy;

// ...

procedure TForm1.Button1Click(Sender: TObject);
var a, b, c: integer;
begin
  a := StrToInt(Edit1.Text);
  b := StrToInt(Edit2.Text);
  c := GetISimpleCalc.Add(a, b);
  LabelResult.Caption := 'Result = ' + IntToStr(c);
end;
```

Hide image



Summary

In this article we have looked into Windows Communication Foundation object model, and how simple it is to build WCF services and clients with Delphi 2007 for .NET, and what it takes to expose WCF services to arbitrary Web Service client application on the example of Delphi 2007 for Win32 VCL Forms Application.

The WCF provides very elegant object model for building distributed application that uses multiple communications protocols, and supports many different message exchange patterns.

For some background information on Web Services and the WSDL have a look at **"Introduction to Web Services"** Borland whitepaper by Hartwig Gunzer.

References

- **Source Code for this article**
- **"Windows Communication Foundation" from Wikipedia, the free encyclopedia**
- **"Introducing Indigo: An Early Look" by David Chappell**
- **"Programming Indigo: The Unified Framework for Building Service-Oriented Applications on the Windows Platform Beta Edition (Pro-Developer)" by David Pallmann, MS Press ISBN-10: 0735621519**
- **Book Excerpt, Chapter 1, "Learning WCF" by Michele Leroux Bustamante, O'Reilly Media, 2007 - O'Reilly.com [pdf]**
- **WSDL 2.0 Part 0: Primer**
- **Web Services Enhancements (WSE) Home Page**
- **"Manually Adding MEX Endpoints" from Nicholas Allen's Indigo Blog**
- **"That Indigo Girl" - Michele Leroux Bustamante Indigo blog**
- **"Introduction to Web Services" by Hartwig Gunzer, Sales Engineer, Borland, March 2002**
- **WSDL Specification: Introduction**
- **MSDN: Fundamental Windows Communication Foundation Concepts**
- **MSDN: Windows Vista Technical Articles: Windows Communication Foundation Architecture Overview**
- **MSDN: WCF System-Provided Bindings**
- **MSDN: Windows Communication Foundation Architecture**
- **MSDN Magazine: "Extend Your WCF Services Beyond HTTP With WAS"**

LATEST COMMENTS

Move mouse over comment to see the full text

Server Response from: SC2

 Site Map