1. Add a new WCF service to the application: right-click the WebSites and choose Add | New Item. Select the **WCF Service template**. Name the service **ShipperService.svc**.

|  |  |  |
| --- | --- | --- |
|  | Notice that three files are created:   * an interface file named **IShipperService**, * a class file named **ShipperService**, * and a **ShipperService.svc** file. |  |

2. Open  **IShipperService.cs t**he file will be

using System;

using System.Collections.Generic;

using **System.Runtime.Serialization;**

using **System.ServiceModel;**

using System.Text;

[**ServiceContract**]

public interface **IShipperService**

{

[OperationContract]

Shipper **GetShipper**(int shipperId);

[OperationContract]

void **SaveShipper**(Shipper shipper);

}

**// Define data to interact: DataContract**

[**DataContract**]

public class **Shipper**

{

[DataMember]

public int **ShipperId** { get; set; }

[DataMember]

public string **CompanyName** { get; set; }

[DataMember]

public string **Phone** { get; set; }

}

1. **WORK WITH ShipperService**

3. Implement the methods in the **ShipperService.cs**:

* **SaveShipper** method by updating a shipping record with data inside a Shipper instance.
* **GetShipper** method by calling the database to retrieve a record from the Shipper table. Copy this data into a **Shipper instance** and return it as a result of the function. (Shipper instance is a class with the fields to return)

The file **ShipperService.cs** code will be

using System;

using System.Collections.Generic;

using System.Runtime.Serialization;

using System.ServiceModel;

using System.Text;

using System.Data;

using System.Data.SqlClient;

using System.Configuration;

public class **ShipperService : IShipperService**

{

private string \_cnnString =ConfigurationManager.ConnectionStrings["NwConnectionString"].ToString();

**// In IShipperService is the public class Shipper define with 3 fields to return**

public **Shipper** **GetShipper**(int shipperId)

{

StringBuilder sql = new StringBuilder();

sql.Append("SELECT shipperId, companyName, phone ");

sql.Append("FROM shippers WHERE (shipperId = @ShipperId) ");

SqlConnection cnn = new SqlConnection(\_cnnString);

SqlCommand cmd = new SqlCommand(sql.ToString(), cnn);

cmd.Parameters.Add("ShipperId", SqlDbType.Int, 0).Value = shipperId;

SqlDataAdapter adp = new SqlDataAdapter(cmd);

DataSet ds = new DataSet();

adp.Fill(ds);

Shipper s = new Shipper();

s.ShipperId = shipperId;

s.CompanyName = ds.Tables[0].Rows[0]["companyName"].ToString();

s.Phone = ds.Tables[0].Rows[0]["phone"].ToString();

return s;

}

public void **SaveShipper**(Shipper shipper)

{

StringBuilder sql = new StringBuilder();

sql.Append("UPDATE Shippers set phone=@Phone, ");

sql.Append("companyName=@CompanyName WHERE shipperId = @ShipperId ");

SqlConnection cnn = new SqlConnection(\_cnnString);

SqlCommand cmd = new SqlCommand(sql.ToString(), cnn);

cmd.Parameters.Add("Phone", SqlDbType.NVarChar, 24).Value = shipper.Phone;

cmd.Parameters.Add("CompanyName", SqlDbType.NVarChar, 40).Value = shipper.CompanyName;

cmd.Parameters.Add("ShipperId", SqlDbType.Int, 0).Value = shipper.ShipperId;

cnn.Open();

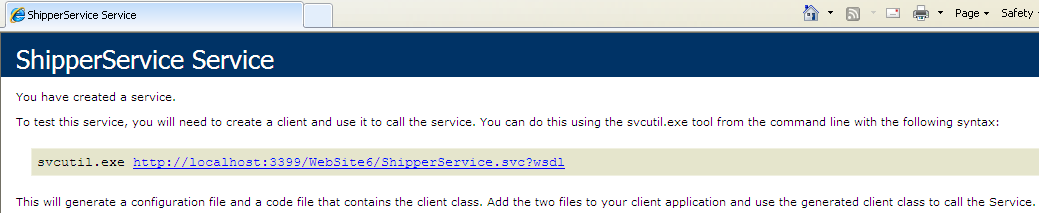
cmd.ExecuteNonQuery();

cnn.Close();

}

}

4. Right-click your **ShipperService.svc** file and choose **Set As Start Page**. Compile and run your service application. Here you will see details on how your WCF service should be called.



1. **CONSUME THE WCF SERVICE**

5. To access the WCF first add a new website to the solution: right-click the solution and choose Add | New Web Site. Select ASP.NET Web Site. Name the website **ShipperClient**. Right-click the website and choose Set As StartUp Project.

|  |  |
| --- | --- |
| Right-click the web site and choose Add Service Reference | Click **Discover** and select the **ShipperService**.  Set the namespace to **NwServices**. Click OK to close the dialog box. |
|  |  |

**NwServices** is the proxy name to be used in the code

6. Open the **Default.aspx** page of the **ShipperClient** site. Add controls to the web page.

|  |  |
| --- | --- |
|  | <form id="form1" runat="server">  <div>  <h2>Edit Shipper Information</h2>  Enter Shipper Code:  <br />  <asp:TextBox ID="TxShipperCode" runat="server"></asp:TextBox>  <asp:Button ID="BtGet" runat="server" Text="Get Shipper"  onclick="**BtGet\_Click**" />  <hr />  <h3>Edit Shipper Data</h3>  Shiper Id:<br />  <asp:TextBox ID="TxShipperID" runat="server"></asp:TextBox>  <br />  Company Name:<br />  <asp:TextBox ID="TxCompany" runat="server"></asp:TextBox>  <br />  Phone:<br />  <asp:TextBox ID="TxPhone" runat="server"></asp:TextBox>  <br />  <asp:Button ID="BtSave" runat="server" Text="Save" onclick="**BtSave\_Click**" />  </div>  </form> |

7. Add code to the Get Shipper button’s click event to call the WCF service and put the results into the Shipper Data form.

protected void **BtGet\_Click**(object sender, EventArgs e)

{

**NwServices**.ShipperServiceClient nwShipper = new **NwServices.**ShipperServiceClient();

**NwServices**.Shipper shipper = new **NwServices**.Shipper();

int **shipperId** = int.Parse(TxShipperCode.Text);

shipper = nwShipper.GetShipper(**shipperId**);

TxShipperID.Text = shipper.ShipperId.ToString();

TxCompany.Text = shipper.CompanyName;

TxPhone.Text = shipper.Phone;

}

16. Add code to the Save button’s click event to call the WCF service with the values from the various TextBox controls.

protected void **BtSave\_Click**(object sender, EventArgs e)

{

**NwServices**.Shipper shipper = new **NwServices**.Shipper();

shipper.ShipperId = int.Parse(TxShipperID.Text);

shipper.CompanyName = TxCompany.Text;

shipper.Phone = TxPhone.Text;

NwServices.ShipperServiceClient nwShipper = new NwServices.**ShipperServiceClient**();

nwShipper.SaveShipper(shipper);

}

17. Run the application. Enter a Shipper ID (**1**, **2**, or **3**). Edit the data and save it back to the database.

= = = = = =

**Calling a WCF Service from Client Script by Using AJAX (REST and JSON)**

One such service type is based on representational state transfer (REST) and JavaScript Object Notation (JSON). . Programming with AJAX becomes easier with a simple service (REST) based on a simple message format (JSON).

A REST service is a web service you create that responds to HTTP GET requests. Clients can therefore call a REST service the same way they would access a page: by using a URL and a query string. The server then responds with text, as it would for any HTTP GET request. This way, a REST service does not require knowledge of the XML schema used to call the service. Instead, it simply sends the request and processes the text-based response (usually JSON formatted data).

*Note* **Securing a REST-based ser vice** REST-based services do not use SOAP. Many service-based security models are based on SOAP. Therefore, if the security of the data being passed is a concern, you should use HTTPS between the client and server for all RESTful services.

The response of a REST service is typically in the form of JSON data. JSON is a message data format that evolved from the heavy use of AJAX. The message format is not based on XML. Instead, it is simple, lightweight, and text based. A JSON message can be processed easily by the JavaScript engine. In fact, a JSON message can be parsed by using the JavaScript eval function, because basically it is syntactically formatted JavaScript.

The following is an example of a JSON-formatted message.

{

"productName": "Computer Monitor",

"price": "229.00",

"specifications": {

"size": 22,

"type": "LCD",

"colors": ["black", "red", "white"]

}

}

**Writing a WCF Service Based on REST and JSON**

The **AJAX-Enabled WCF Service** template is a WCF template use to create a WCF service that takes advantage of the REST calling mechanism and the JSON data format, to use AJAX.

namespace **PricingServices**

{

[ServiceContract(Namespace = "**PricingServices**")]

[AspNetCompatibilityRequirements(RequirementsMode = **AspNetCompatibilityRequirementsMode.Allowed**)]

public class **PricingService**

{

[**OperationContract**]

[**WebInvoke**]

public double **CalculatePrice**(string itemId)

{

double price = 45 \* 1.06 ; //simulate product price lookup based on item ID

return price;

}

}

}

**Method options**

1. **WebInvoke** if you are sending data to the server or if you don’t want your request to be cached by a browser or a server.

This option is called by using HTTP POST, this service can be called by an HTTP request. This can be important if you are sending data to the server to be written or if you do not want your request to be cached by a browser or the server.

1. **WebGet** If your service returns data static. This indicates an HTTP GET request is supported and the results can be cached.

The ASP.NET AJAX ScriptManager control can work with both HTTP GET and POST services.

Visual Studio also updates the site’s Web.config file when the AJAX-enabled WCF service is added to the project.

Notice the <**enableWeb-Script** /> element. This indicates that the endpoint is a RESTful service that uses the JSON data format and can therefore be consumed by AJAX. Also notice that the binding is set to **webHttpBinding**, indicating again that this service is called via HTTP (and not SOAP).

<system.serviceModel>

<services>

<service name="PricingService">

<endpoint address=""

behaviorConfiguration="PricingServiceAspNetAjaxBehavior"

binding="webHttpBinding"

contract="PricingService" />

</service>

</services>

<behaviors>

<endpointBehaviors>

<behavior name="PricingServiceAspNetAjaxBehavior">

<**enableWebScript** />

</behavior>

</endpointBehaviors>

<serviceBehaviors>

<behavior name="">

<serviceMetadata httpGetEnabled="true" />

<serviceDebug includeExceptionDetailInFaults="false" />

</behavior>

</serviceBehaviors>

</behaviors>

<serviceHostingEnvironment

aspNetCompatibilityEnabled="true"

multipleSiteBindingsEnabled="true" />

</system.serviceModel>

**Calling a JSON-Based WCF Service from AJAX**

To call the service add a ScriptManager control to your page. You then define a **ServiceReference** to the actual service.

<asp:ScriptManager ID="ScriptManager1" runat="server">

<Services>

<asp:ServiceReference Path="PricingService.svc" />

</Services>

</asp:ScriptManager>

You can then define a script block on your page to call the proxy class that is generated based on this service reference.

<script language="javascript" type="text/javascript">

function **ButtonCalculate\_onclick**() {

var service = new **PricingServices.PricingService**();

service.**CalculatePrice**(document.forms[0].MainContent\_TextBoxProduct.value, onSuccess, onFail, null);

}

function onSuccess(result){

**LabelPrice**.innerText = result;

}

function onFail(result){

alert(result);

}

</script>

The following code shows the markup for the page’s controls, to complete the example.

<div>

Product:<br />

<asp:TextBox ID="TextBoxProduct" runat="server"></asp:TextBox>

<br />

<input name="ButtonCalculate" type="button" value="Get Price" onclick="**ButtonCalculate\_onclick**()" />

<br />

<asp:Label ID="**LabelPrice**" runat="server"></asp:Label>

</div>

Note: The ScriptManager control converts your complex type into a JSON message structure. After the call completes, you can access the individual values of the complex type by using the ***result.member*** syntax.

**Creating WCF Data Services**

WCF Data Services allow you to expose and work with data by using the Open Data Protocol (OData). Data is exposed over REST-based services to allow for standard HTTP GET and POST requests. The data can be accessed based on a URL and standard query string formats for accessing tables and filtering data. These formats allow access from any client that can receive XML or JSON message notations.

There is a standard set of steps to follow when creating OData services with ASP.NET. These steps include the following:

1. Define a data model that uses an Entity Data Model (**EDM**) by using the ADO.NET Entity Data Model template. For more information on EDM, see Chapter 12, “**Working with Data Source Controls and Data-Bound Controls**.” Assume a model from Northwind is called **NwdModel.edmx**.

2. Create a data service by using the WCF Data Service template.

Create the WCF Data Services-based service ( **NwdDataService.svc )** This service includes an SVC file that is used as a URI for the services. The file contains the **@ ServiceHost** directive, which points to the actual service class and indicates that this service is a data service. The following shows sample markup.

<%@ ServiceHost Language="C#" Factory="System.Data.Services.DataServiceHostFactory" Service="NwdDataService" %>

3. Add code to the data service to enable access to the EDM.

The actual service class is inside **App\_Code** which inherits from **System.Data.Services.DataService,** a generic class. Here you indicate the Entity Data Model you intend your service to expose—in this case, **northwndModel.NwdEntities**.

public class NwdDataService : **DataService<northwndModel.NwdEntities**>

{

4. Create a client application to access the exposed data service.

Add rules to the data service inside the **InitializeService** method to indicate which entities in your model are visible and can be updated. The following code indicates that the customer table is exposed as a data service. This also provides for the ability to write and to read the entire table. This code is called once to initialize the service.

public static void InitializeService(DataServiceConfiguration config)

{

config.DataServiceBehavior.MaxProtocolVersion = DataServiceProtocolVersion.V2;

config.SetEntitySetAccessRule("Customers", EntitySetRights.AllRead | EntitySetRights.AllWrite);

}

The service itself is now complete. In fact, it can be accessed by using the OData querystring syntax. For example, the following request returns a single customer based on ID.

http://localhost:53908/WcfDataSrv-cs/NwdDataService.svc/Customers('ALFKI')

*Note* **OData URI conventions**

The OData specification provides several ways in which you can access and filter data by using the query string. For more details, review the specification at [*http://www.odata.org*](http://www.odata.org) */developers/protocols/uri-conventions*.

The results of this REST call are shown in Figure 10-12. Note that you must turn off the feed reader view in Windows Internet Explorer to see the actual XML (Tools | Internet Options | Content | Feeds And Web Slices Settings).



**Figure 10-12** Calling a WCF data service by using basic REST protocol and OData syntax.

5. Add a service reference to the client application to call the data service. This will generate a WCF proxy for calling the service.

After the client has been created, you right-click the project and choose Add Service Reference as you would for any WCF service. You give the service a namespace to be used for the proxy, which in this case will be **NwdEntitiesSrv.**

Your client application could be a Windows application, a console application, Microsoft Silverlight, ASP.NET, or any other client capable of calling a service. In this example, we will create an ASP.NET website client.

6. Write code to work with the exposed data. You can query the data by using LINQ, bind the data to controls, and use the proxy class to save any changes.

You can then use this reference, along with the WCF Data Services client libraries, to work with the WCF Data Services. As an example, the following code instantiates a reference to the service, gets all customers back from the service, and binds the results to a GridView control.

Uri svcUri = new Uri("http://localhost:53908/WcfDataSrv/NwdDataService.svc");

NwdEntitiesSrv.NwdEntities nwd = new NwdEntitiesSrv.NwdEntities(svcUri);

DataServiceQuery<NwdEntitiesSrv.Customer> q = nwd.Customers;

this.GridViewCustomers.DataSource = q;

this.GridViewCustomers.DataBind();

For more information, see “Using a Data Service in a Client Application” ([*http://msdn.microsoft.com/en-us/library/dd728282.aspx*](http://msdn.microsoft.com/en-us/library/dd728282.aspx)).

**Quick Check**

1. How do you mark a class or interface as a WCF service?

2. How do you mark methods in an interface or class so that they are exposed as part of the class’s service contract?

3. Which attribute do you add to a WCF service to indicate that it can be called by an HTTP request that uses REST?

4. From what class should your WCF Data Services derive?

**Quick Check Answers**

1. You use the ServiceContract Attribute class to tag an interface or class as a WCF service.

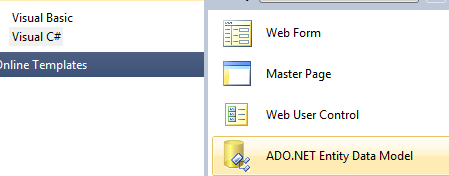
2. You use the OperationContract Attribute class to tag a method as a service method.

3. You mark the service with the WebInvoke Attribute class.

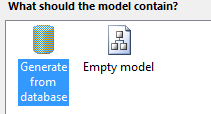
4. You should derive from the DataService class found in the System.Data.Services namespace.

**WCF with Entity Framework**

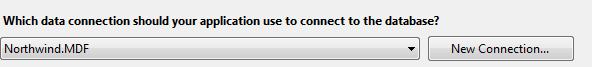
1. Create a new WebSite
2. Copy Northwind.mdb in the folder App\_Data
3. Add in the project Northwind.mdb as existing item
4. Create the Entity Framework, when right-click the project and Add new item, choose and called it Model.edmx



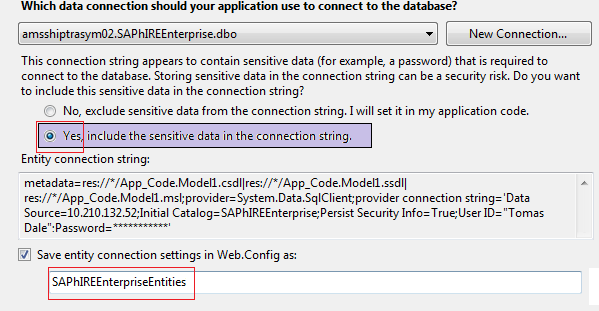
1. Generate from Database



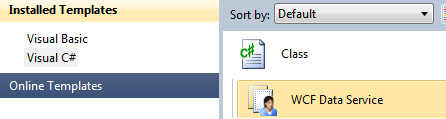
1. Select Northwind



1. Or select New Connection for a new connection



1. Select tables and click finish
2. Click the project and add a WCF DataService



1. Inside this file add the model



public class WcfDataService : DataService< SAPhIREEnterpriseModel.SAPhIREEnterpriseEntities>

{

// This method is called only once to initialize service-wide policies.

public static void InitializeService(DataServiceConfiguration config)

{

config.DataServiceBehavior.MaxProtocolVersion = DataServiceProtocolVersion.V2;

config.SetEntitySetAccessRule("Customers", EntitySetRights.AllRead | EntitySetRights.AllWrite);

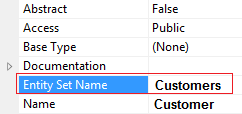
config.SetEntitySetAccessRule("tblSiteInfoes", EntitySetRights.AllRead | EntitySetRights.AllWrite);

config.SetEntitySetAccessRule("tblCurrencies", EntitySetRights.AllRead | EntitySetRights.AllWrite);

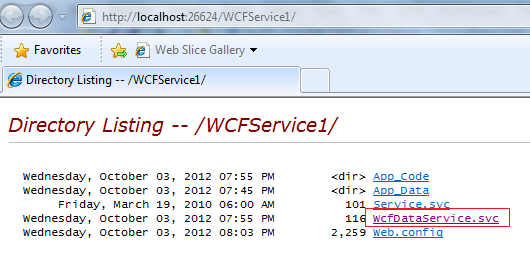
}

}

1. Check the name of the table in properties Entity Set Name, which is most of the time the plural.



1. If you run the application it will be seen



1. Click WcfDataService.svc and you will se

<?xml version="1.0" encoding="utf-8" standalone="yes" ?>

[**-**](http://localhost:26624/WCFService1/WcfDataService.svc/) <service xml:base="**http://localhost:26624/WCFService1/WcfDataService.svc/**" xmlns:atom="**http://www.w3.org/2005/Atom**" xmlns:app="**http://www.w3.org/2007/app**" xmlns="**http://www.w3.org/2007/app**">

[**-**](http://localhost:26624/WCFService1/WcfDataService.svc/) <workspace>

<atom:title>**Default**</atom:title>

[**-**](http://localhost:26624/WCFService1/WcfDataService.svc/) <collection href="**Customers**">

<atom:title>**Customers**</atom:title>

</collection>

</workspace>

</service>



1. Then in the internet explorer type

http://localhost:26624/WCFService1/WcfDataService.svc/Customers (‘ALFKI’)

http://localhost:26624/WCFService1/WcfDataService.svc/tblSiteInfoes(78696577113894021)

If you don’t set the ID it retrieves all data from the table

The results of this REST call are shown in Figure 10-12. Note that you **must turn off the feed reader view in Windows Internet Explorer** to see the actual XML (Tools | Internet Options | Content | Feeds And Web Slices Settings).

If you received

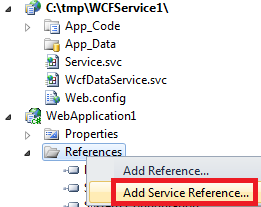
*The service cannot be activated because it does not support ASP.NET compatibility. ASP.NET compatibility is enabled for this application. Turn off ASP.NET compatibility mode in the web.config or add the AspNetCompatibilityRequirements attribute to the service type with RequirementsMode setting as 'Allowed' or 'Required'.*

Change web.config

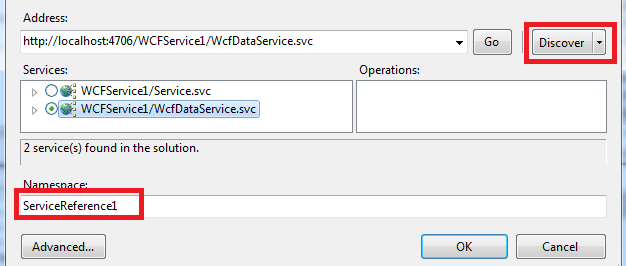
<serviceHostingEnvironment multipleSiteBindingsEnabled="true" aspNetCompatibilityEnabled="false"/>

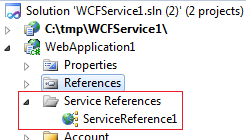
**Now you call the service**

1. Create a new web site project
2. Add the reference to the service



1. Click discover to retrieve possible service





1. Code in the client

protected void Page\_Load(object sender, EventArgs e)

{

Uri svcUri = new Uri("http://localhost:26624/WCFService1/WcfDataService.svc");

ServiceReference1.SAPhIREEnterpriseEntities nwd = new ServiceReference1.SAPhIREEnterpriseEntities(svcUri);

DataServiceQuery<ServiceReference1.tblCurrency> q = nwd.tblCurrencies;

DataServiceQuery<ServiceReference1.tblSiteInfo> q2 = nwd.tblSiteInfoes;

this.GridView1.DataSource = q;

this.GridView1.DataBind();

var query = from qq in q

where qq.CurrencySymbol !="¢"

select qq;

this.GridView2.DataSource = query.ToList();

this.GridView2.DataBind();

}

1. Run the application