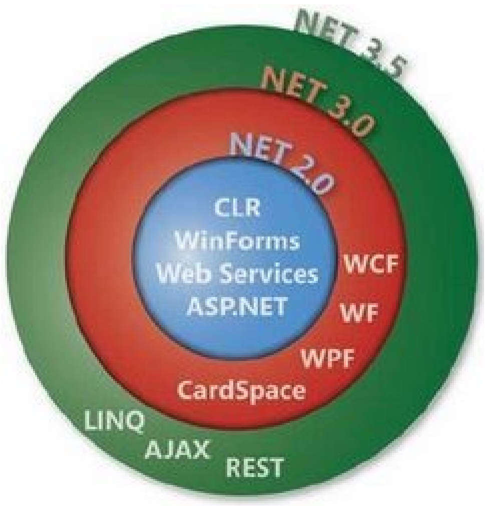
Windows Communication Foundation

Creating a WCF Service Application and Configure this with IIS Server
Comparing Web Services to WCF
WCF Vs Remoting

Regards
Kapil Dhawan
connect2kapil@gmail.com

.Net Version and Components:



Additive versions of the .NET Framework

Above image describes the difference between all versions (2.0 to 3.5).

Net 2.0 = CLR + Win Forms + Web Services + Asp.Net

Net 3.0 = Net2.0 + WCF (Windows Communication Foundation) +
Windows Foundation + WPF (Windows Presentation Foundation) + CardSpace

Net 3.5 = Net 3.0 + LINQ (Language Integrated Query) + AJAX +
REST (Components like List View and Pager)

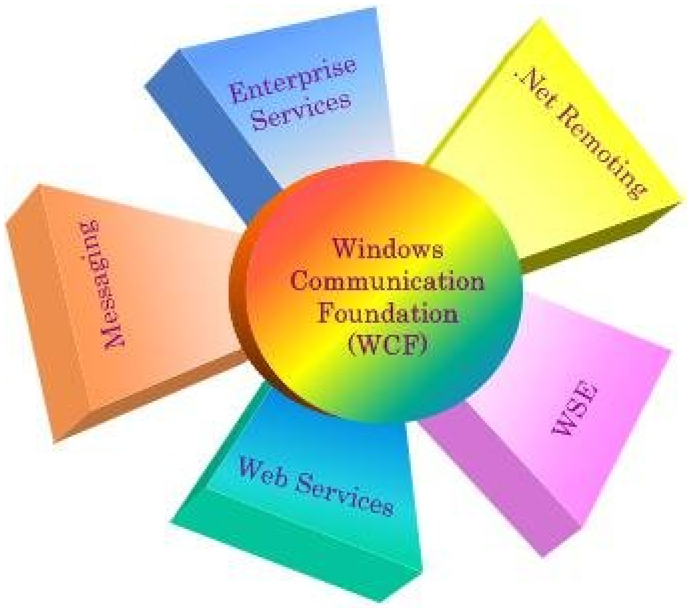
WCF Service:

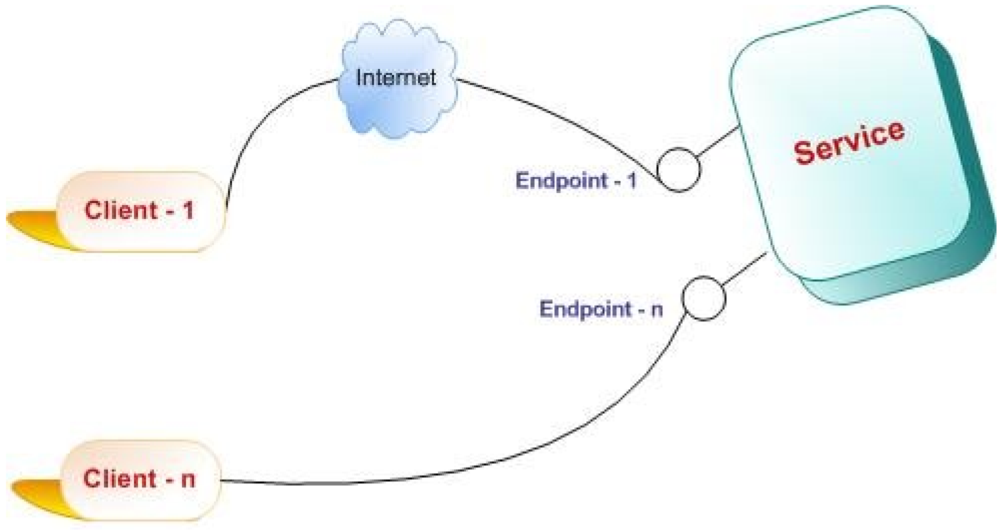
WCF stands for Windows Communication Foundation.

Windows Communication Foundation (WCF) is an SDK for developing and deploying services on Windows. WCF provides a runtime environment for services, enabling you to expose CLR types as services, and to consume other services as CLR types.

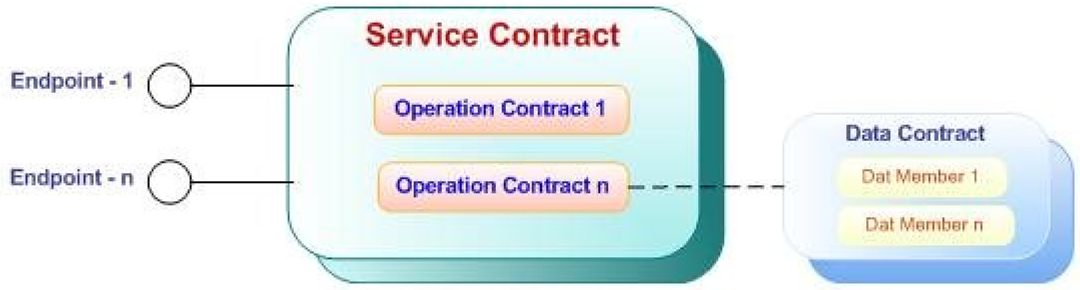
WCF supports for strongly typed as well as untyped messages, this approach allows .net applications to share custom types simply and efficiently. Software created using other platform like java can consume streams of loosely typed XML.

WCF Communication Model:





Windows Communication Foundation (WCF) follows a client–server model to establish communication between applications. Client applications can directly access services through Endpoints exposed by the service. Endpoints are nothing but locations defined, through which messages can be sent or received, and a service can have multiple endpoints.



Basic Feature of a WCF is:

1. Endpoints :
    a. Addresses: - where the service can be found.
    b. Binding : - that contains the information that a client must communicate with the service
    c. Contracts: - that defines the functionality provided by the service to its clients.
2. Data Transfer and Serialization

3. Sessions, Instancing and Concurrency

4. Queues and reliable sessions

5. Transactions

6. WCF Security

7. Peer to peer networking

8. Metadata

9. Clients

10. Hosting

11. Interoperability and Integration

12. Web Programming Model

Every service must have Address that defines where the service resides, Contract that defines what the service does and a Binding that defines how to communicate with the service. In WCF the relationship between Address, Contract and Binding is called Endpoint.

The Endpoint is the fusion of Address, Contract and Binding.

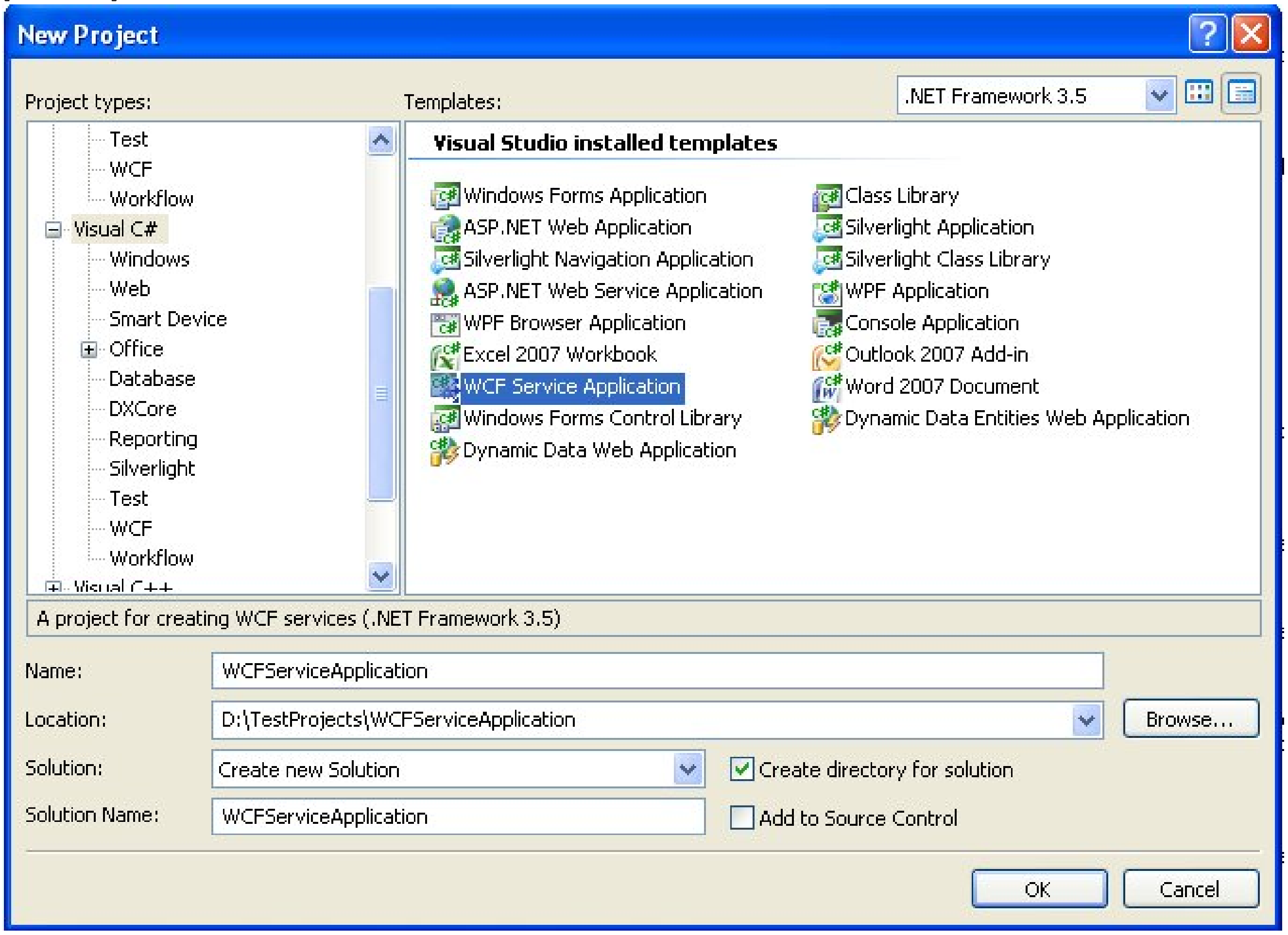
Example 1:

Here we are going to describe an example to create a WCF Service Application and will also configure that to IIS Server.

STEP 1:

Create a new project from VS2008, and create a WCF Service Application template. As shown in screen 1.

[Screen 1]



A project for creating WCF services (.NET Framework 3.5)

Name: WCFServiceApplication

Location: D:\TestProjects\WCFServiceApplication

Solution: Create new Solution

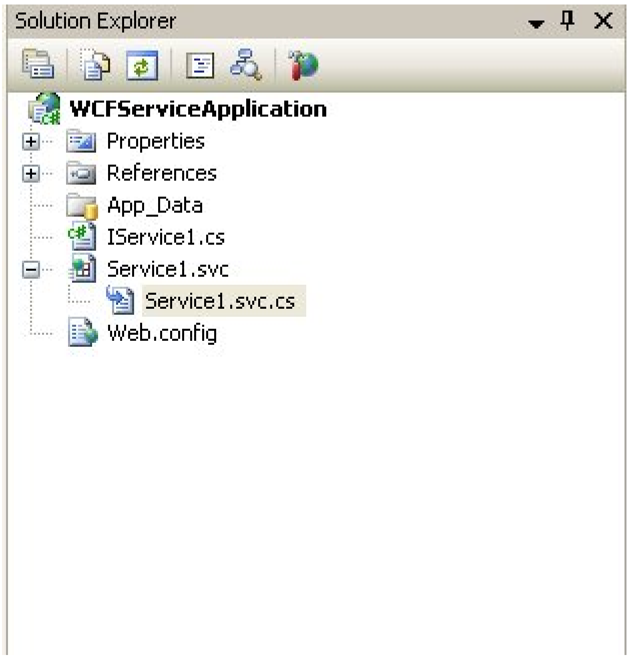
[✓] Create directory for solution

Solution Name: WCFServiceApplication

[ ] Add to Source Control

After creating the new project as WCF Service Application, You will notice that you have some pre-defined items in Solution Explorer, as shown in screen 2.

[Screen 2]



In Solution Explorer, you will find an item named as “Service1.svc”, this file in used to configure the WCF service application to the IIS Server. In case, you rename this file, you need to change its references written in web.config file too.

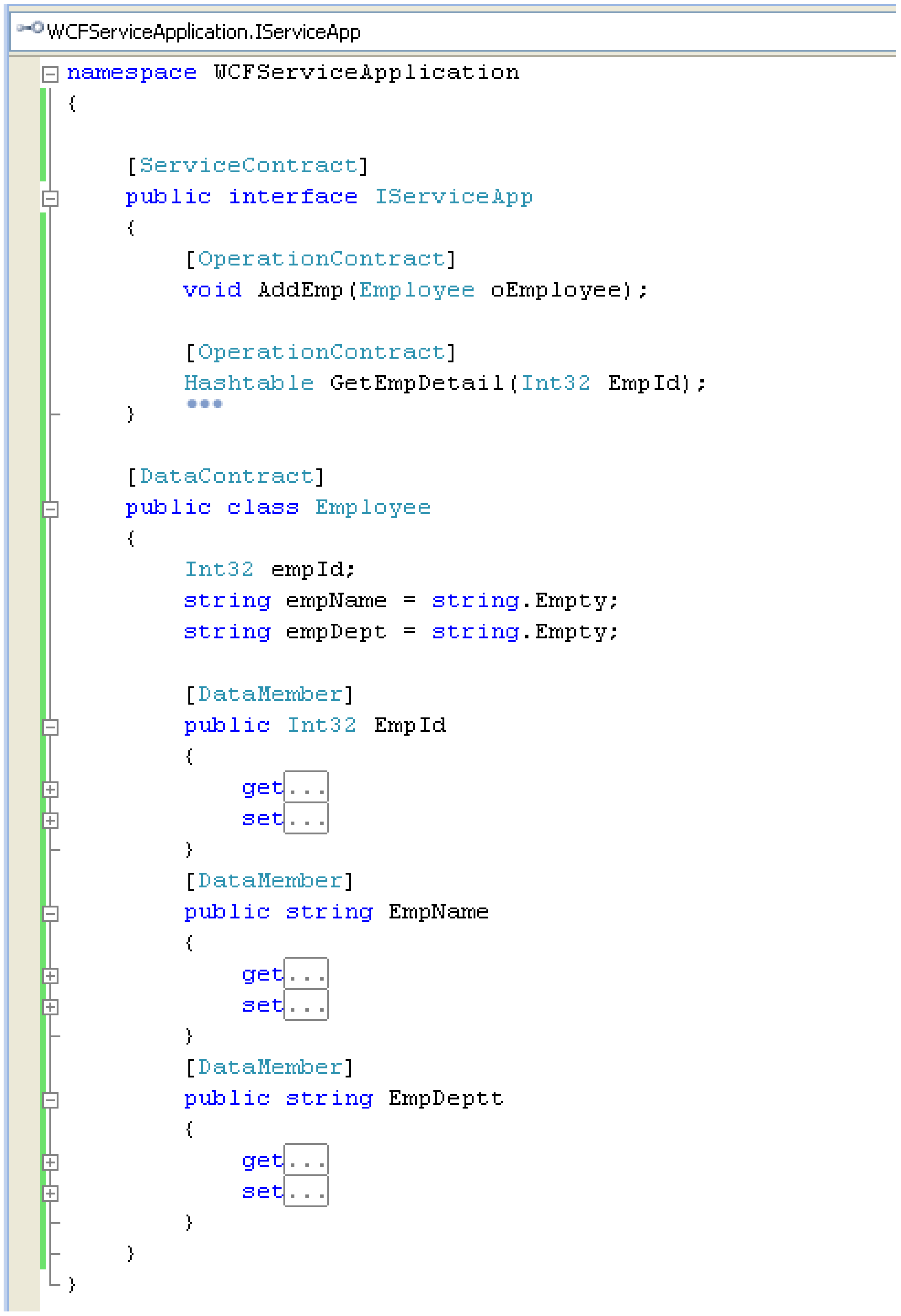
There is also an Interface named as “IService1”, this file also have references in the web.config file, so if you rename this file, you must change its references too.

Because, the item “Service1.svc” inherits the Interface “IService1”, so if you declare any method in the interface, the “Service1.svc” file must implement those methods.

Now we will remove both, the .svc file and the interface, and we will add a new .svc file named as “ServiceApp.svc” and it will automatically add an Interface “IServiceApp.cs”.

We changed the interface “IServiceApp.cs” here, as shown in Screen 3.

[Screen 3]



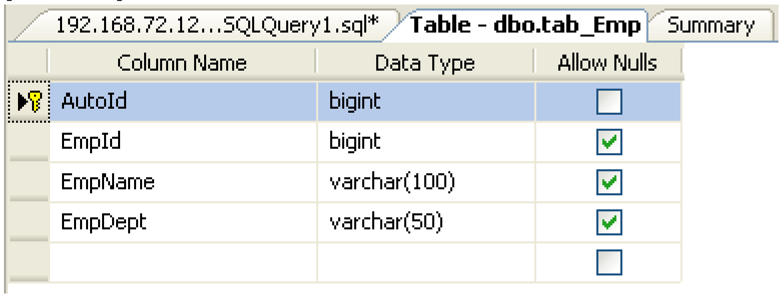
```
WCFServiceApplication.IServiceApp
namespace WCFServiceApplication
{

    [ServiceContract]
    public interface IServiceApp
    {
        [OperationContract]
        void AddEmp(Employee oEmployee);

        [OperationContract]
        Hashtable GetEmpDetail(Int32 EmpId);
    }

    [DataContract]
    public class Employee
    {
        Int32 empId;
        string empName = string.Empty;
        string empDept = string.Empty;

        [DataMember]
        public Int32 EmpId
        {
            get...
            set...
        }
        [DataMember]
        public string EmpName
        {
            get...
            set...
        }
        [DataMember]
        public string EmpDeptt
        {
            get...
            set...
        }
    }
}
```

In the above image [Screen 3], you can see that we have declared two methods to the Interface. One is to add employee detail and the other is to get particular employee detail.

STEP 2:

We are going to use default database “Northwind”. We add a table to the db, named as “tab_Employee”, as shown in Screen 4.

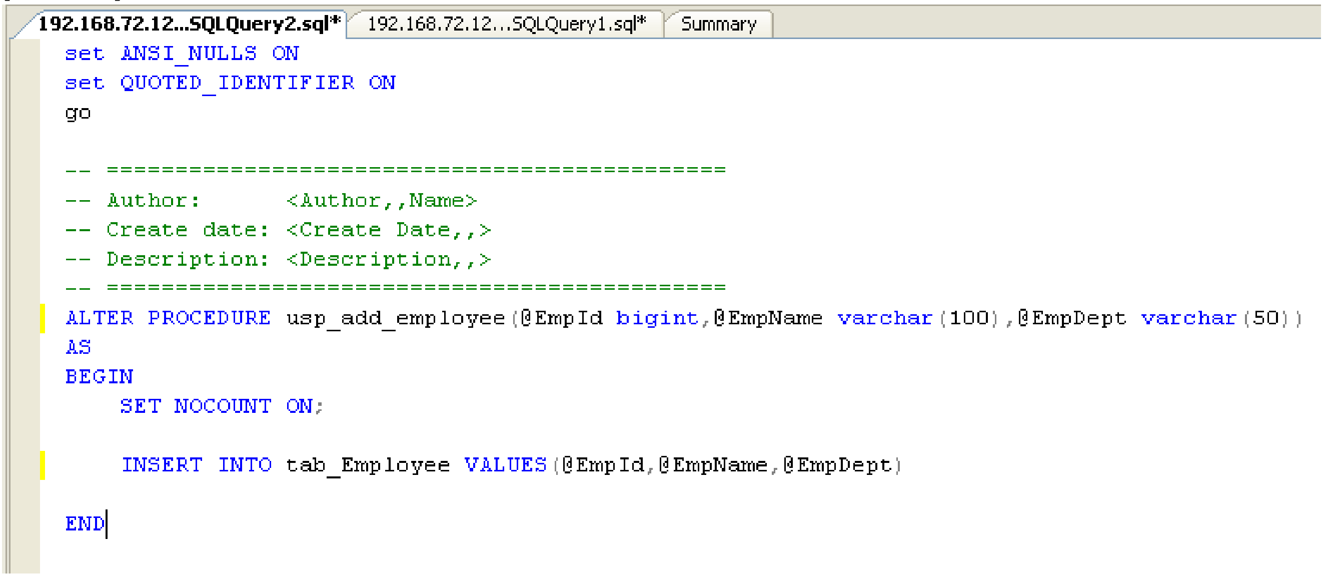
[Screen 4]



Column Name	Data Type	Allow Nulls
AutoId	bigint	☐
EmpId	bigint	✓
EmpName	varchar(100)	✓
EmpDept	varchar(50)	✓
		☐

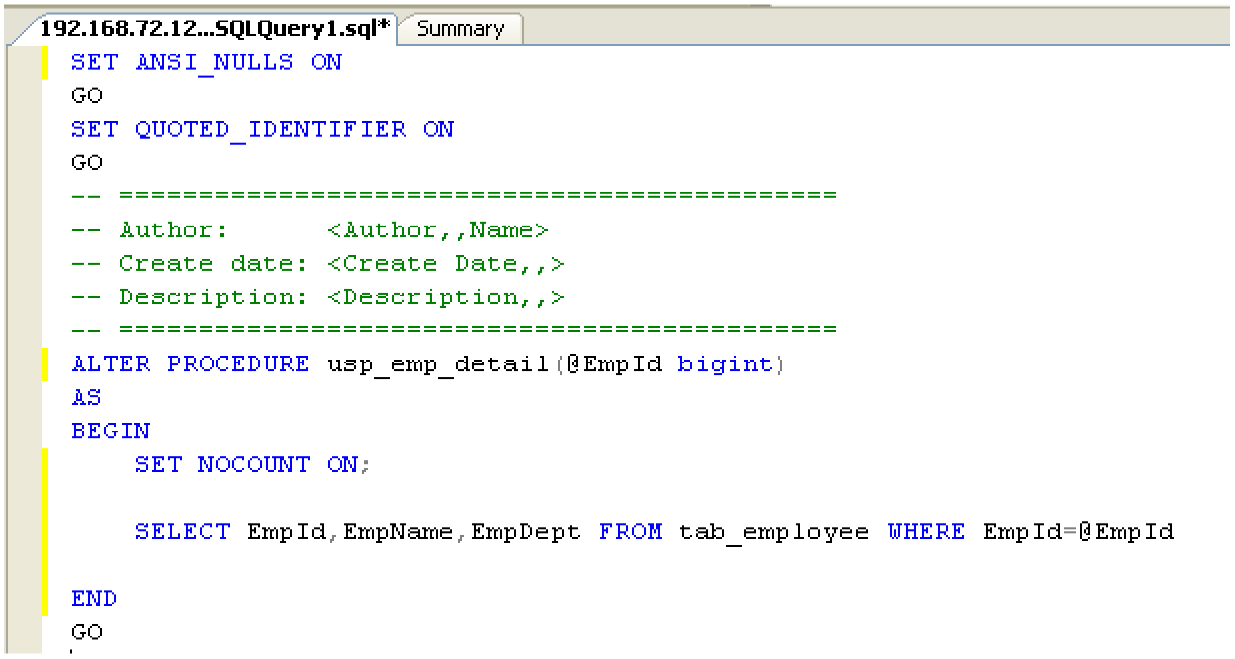
We are going to create stored procedure to the database, one is to save data (Screen 5) to the above table, and second is to retrieve the data based on EmpId (Screen 6).

[Screen 5]


192.168.72.12...SQLQuery2.sql*
192.168.72.12...SQLQuery1.sql*
Summary

```
set ANSI_NULLS ON
set QUOTED_IDENTIFIER ON
go

-- =============================================
-- Author:       <Author,,Name>
-- Create date: <Create Date,,>
-- Description: <Description,,>
-- =============================================
ALTER PROCEDURE usp_add_employee(@EmpId bigint,@EmpName varchar(100),@EmpDept varchar(50))
AS
BEGIN
    SET NOCOUNT ON;

    INSERT INTO tab_Employee VALUES(@EmpId,@EmpName,@EmpDept)

END
```

[Screen 6]



```sql
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
-- =============================================
-- Author:       <Author,,Name>
-- Create date: <Create Date,,>
-- Description: <Description,,>
-- =============================================
ALTER PROCEDURE usp_emp_detail(@EmpId bigint)
AS
BEGIN
    SET NOCOUNT ON;

    SELECT EmpId,EmpName,EmpDept FROM tab_employee WHERE EmpId=@EmpId

END
GO
```

STEP 3:

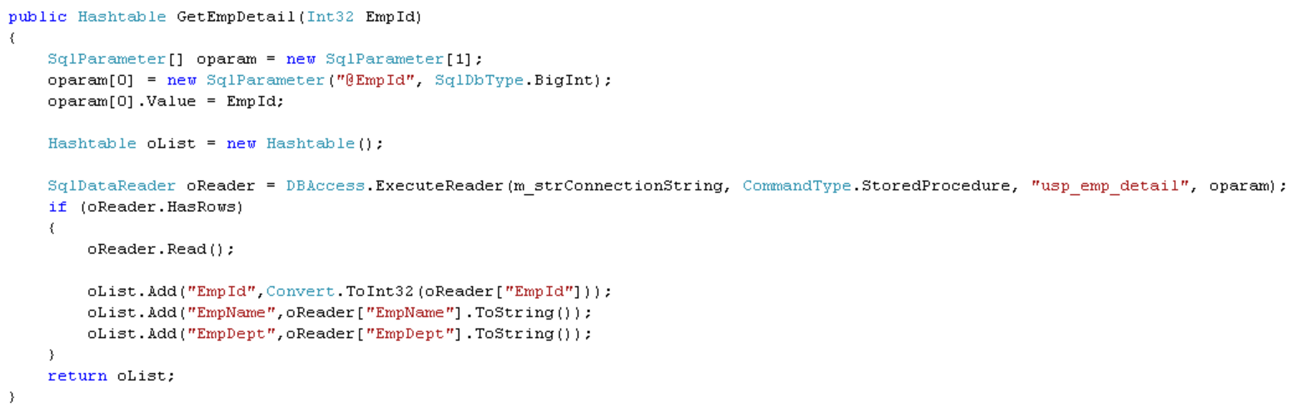
Now we are going to implement the Interface “IServiceApp.cs”, as shown in screen 7 and screen 8. In screen 7, you can see that we have implemented the “AddEmp” method.

[Screen 7]

```csharp
public void AddEmp(Employee oEmployee)
{

    SqlParameter[] oparam = new SqlParameter[3];

    oparam[0] = new SqlParameter("@EmpId", SqlDbType.BigInt);
    oparam[0].Value = oEmployee.EmpId;

    oparam[1] = new SqlParameter("@EmpName", SqlDbType.VarChar, 100);
    oparam[1].Value = oEmployee.EmpName;

    oparam[2] = new SqlParameter("@EmpDept", SqlDbType.VarChar, 50);
    oparam[2].Value = oEmployee.EmpDeptt;

    DBAccess.ExecuteNonQuery(m_strConnectionString, CommandType.StoredProcedure, "usp_add_Employee", oparam);
}
```

And in screen 8, we have implemented the method “GetEmpDetail”.

[Screen 8]



```
public Hashtable GetEmpDetail(Int32 EmpId)
{
    SqlParameter[] oparam = new SqlParameter[1];
    oparam[0] = new SqlParameter("@EmpId", SqlDbType.BigInt);
    oparam[0].Value = EmpId;

    Hashtable oList = new Hashtable();

    SqlDataReader oReader = DBAccess.ExecuteReader(m_strConnectionString, CommandType.StoredProcedure, "usp_emp_detail", oparam);
    if (oReader.HasRows)
    {
        oReader.Read();

        oList.Add("EmpId",Convert.ToInt32(oReader["EmpId"]));
        oList.Add("EmpName",oReader["EmpName"].ToString());
        oList.Add("EmpDept",oReader["EmpDept"].ToString());
    }
    return oList;
}
```

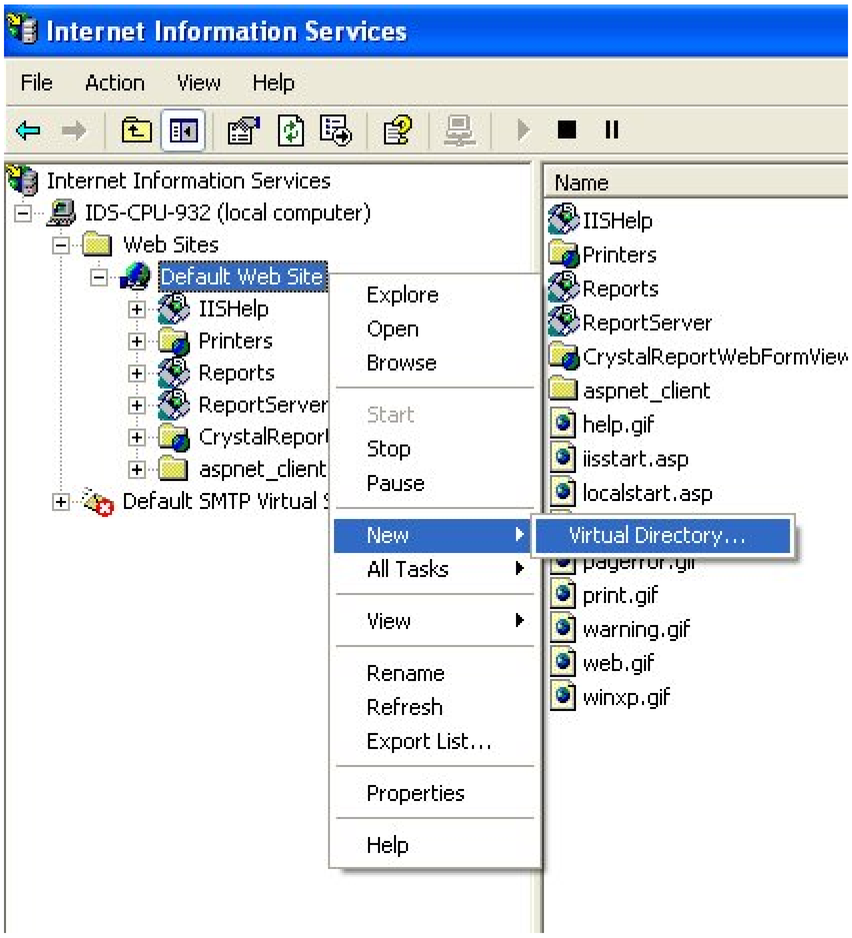
Now the Service has been created and builds successfully.

Configure the WCF Service Application with IIS Server

Because we are done with the WCF Service Application part, now we need to configure the service with IIS.

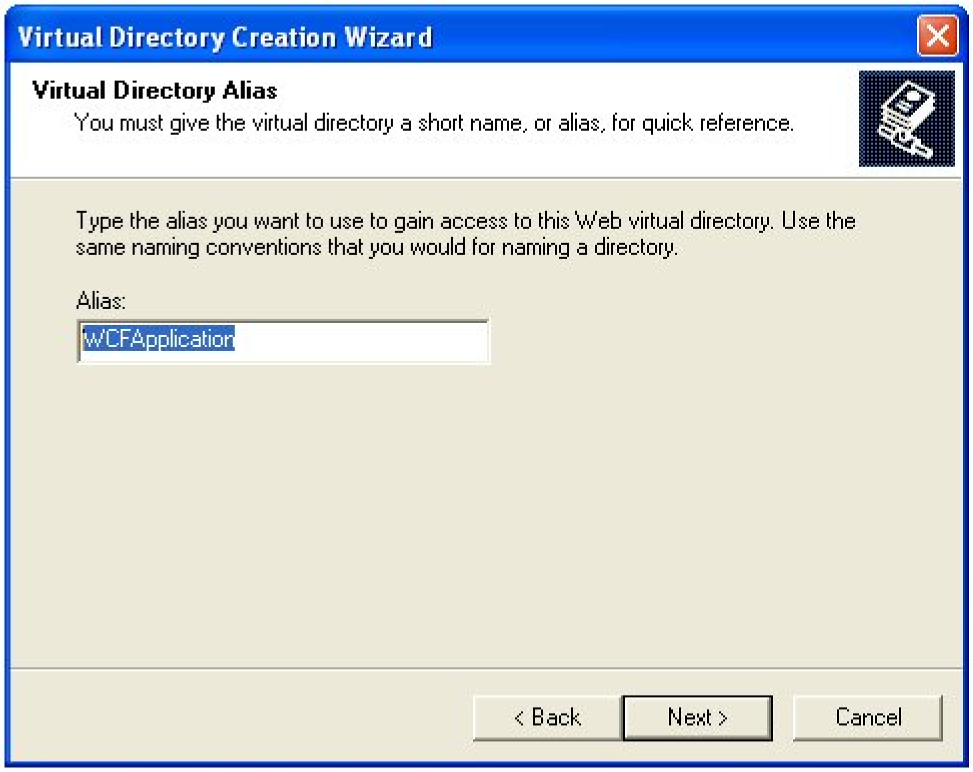
Create a new virtual directory in IIS. As shown in Screen 9.

[Screen 9]



Add an Alias Name for the Service, as shown in Screen 10.

[Screen 10]



Virtual Directory Creation Wizard

Virtual Directory Alias

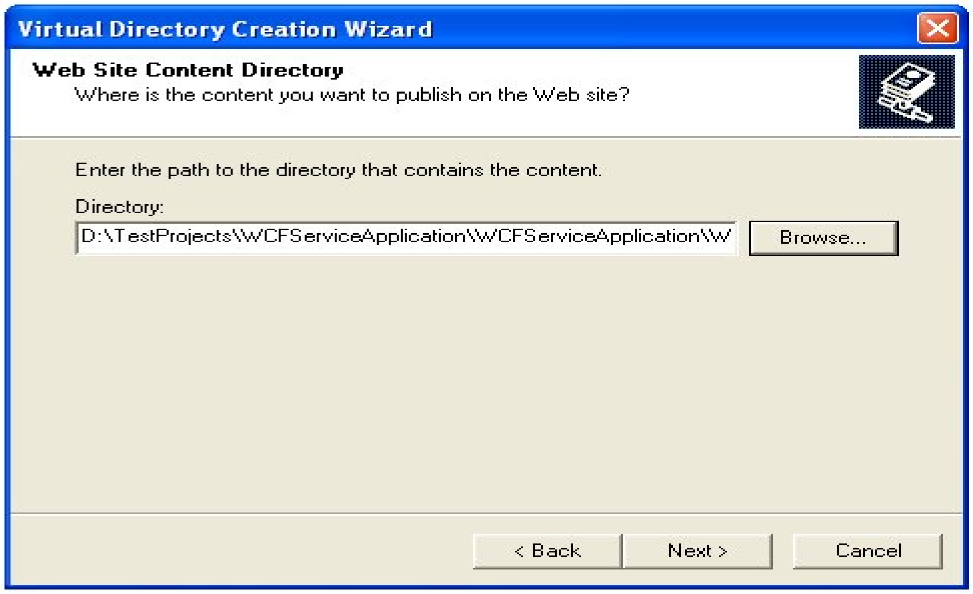
You must give the virtual directory a short name, or alias, for quick reference.

Type the alias you want to use to gain access to this Web virtual directory. Use the same naming conventions that you would for naming a directory.

Alias: WCFApplication

Browse the path of the Application, you just created, as shown in Screen 11.

[Screen 11]



Virtual Directory Creation Wizard

Web Site Content Directory

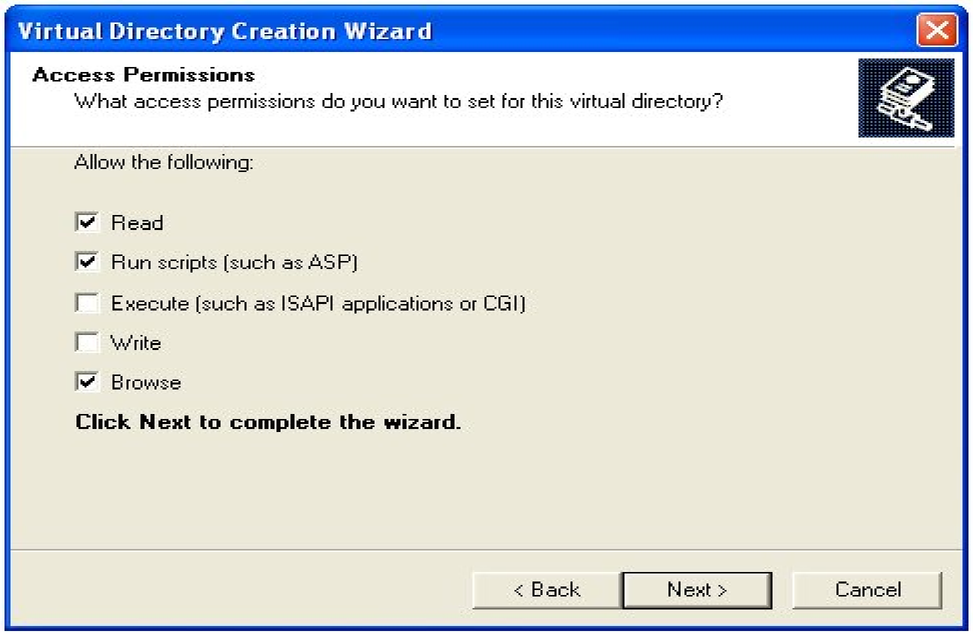
Where is the content you want to publish on the Web site?

Enter the path to the directory that contains the content.

Directory: D:\TestProjects\WCFServiceApplication\WCFServiceApplication\W

Set the permission for the Virtual Directory, as shown in Screen 12.

[Screen 12]


Virtual Directory Creation Wizard

Access Permissions

What access permissions do you want to set for this virtual directory?

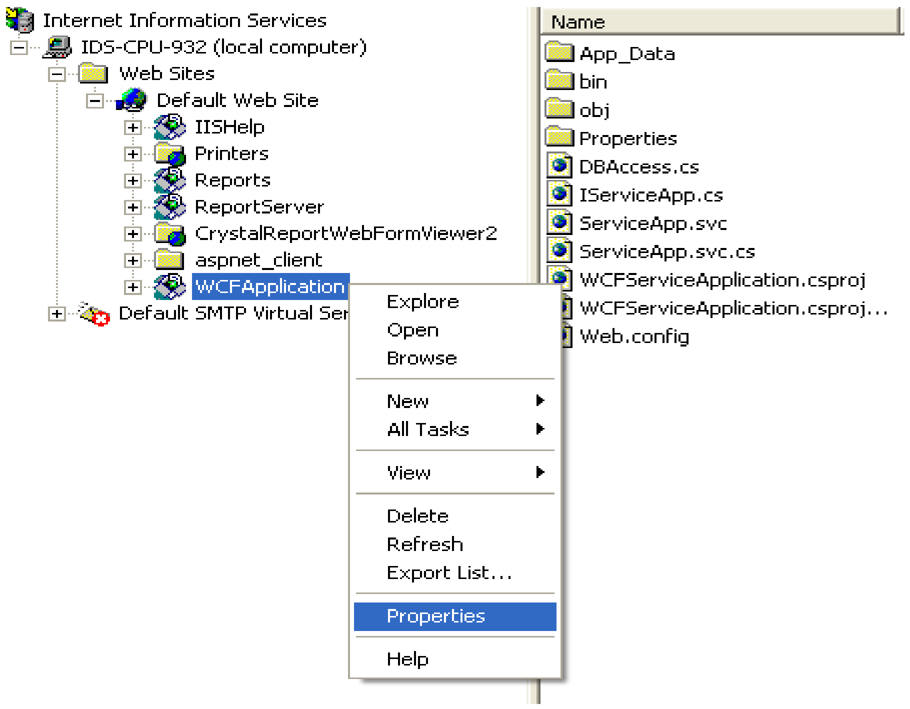
Allow the following:

- [✓] Read
- [✓] Run scripts (such as ASP)
- [ ] Execute (such as ISAPI applications or CGI)
- [ ] Write
- [✓] Browse

Click Next to complete the wizard.

Now Virtual Directory has been created successfully. But to start the service we need to change some properties of this, as shown below in Screen 13.

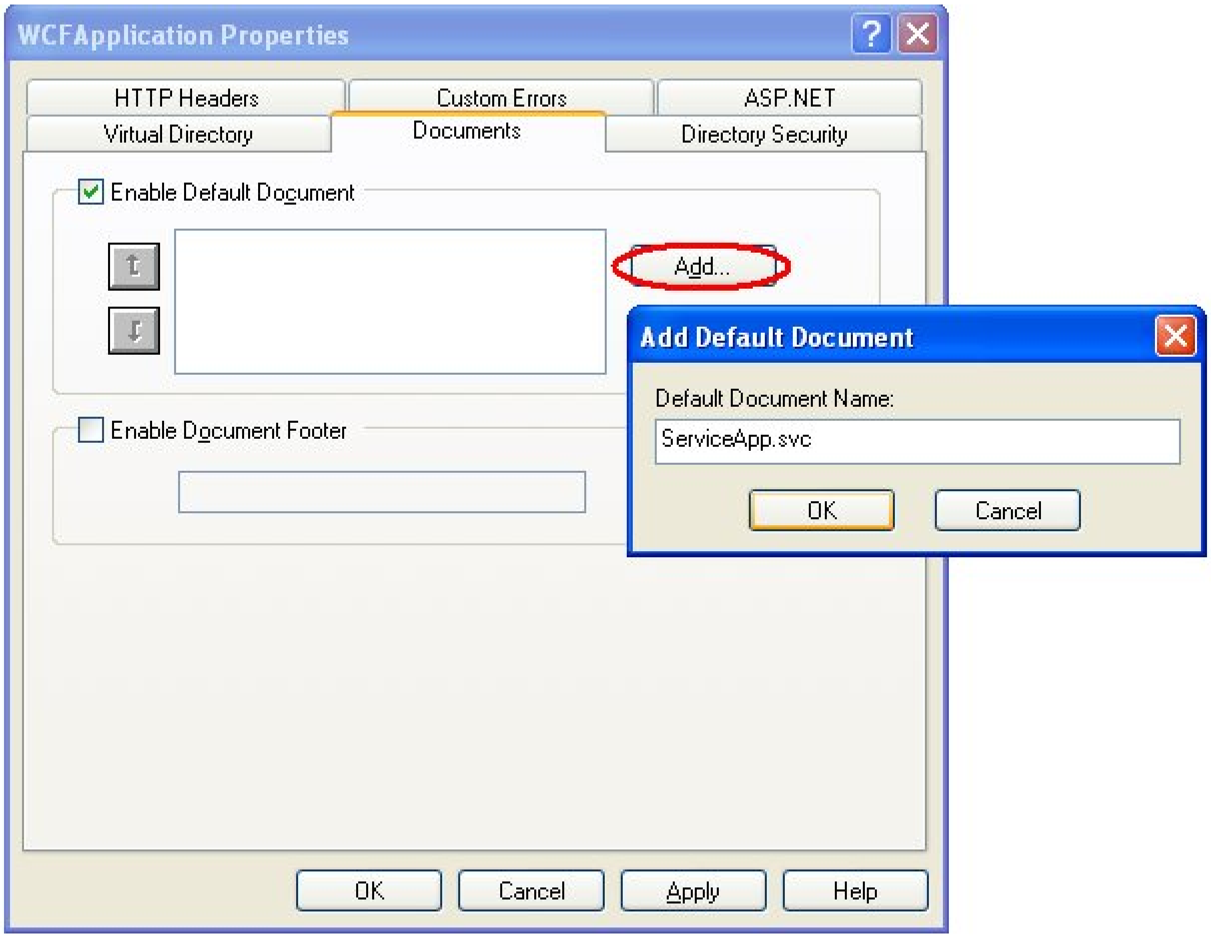
[Screen 13]



Now go to the Document Tab of the Property Window. And Add a default document, as shown in Screen 14.
As you can see, in Default document name, we have input the name of the ***.svc*** file of the WCF Service Application.

"ServiceApp.svc" is the same file we created above, which implements the interface.

[Screen 14]

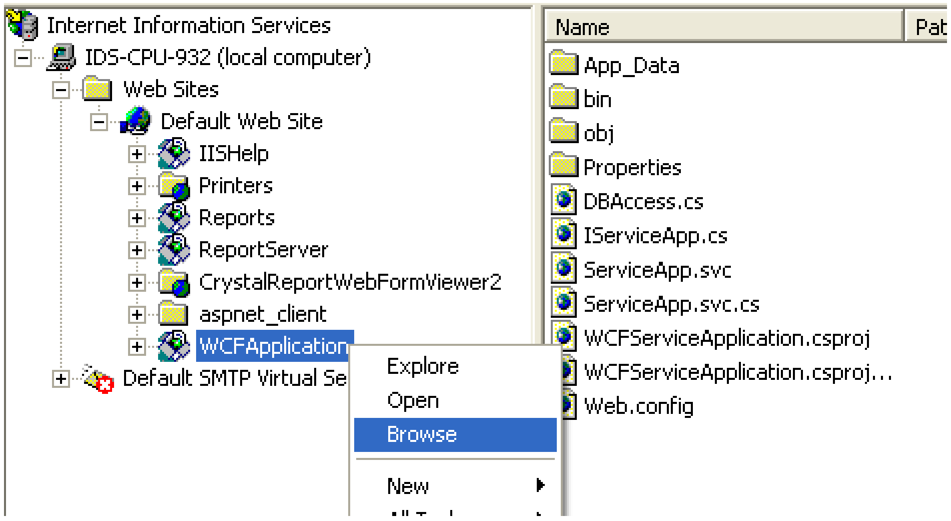


Enable Default Document: ✓

Default Document Name: ServiceApp.svc

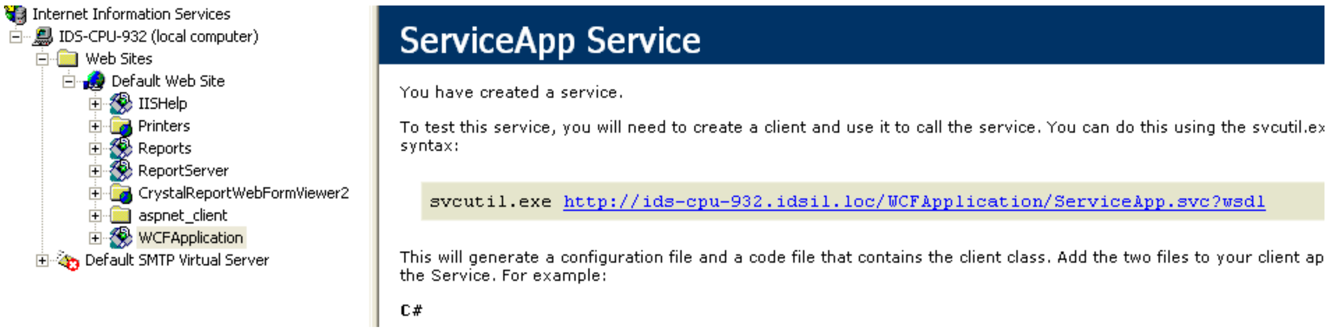
Now to test the WCF Service Application, We need to browse the application from the IIS, as shown below in screen 15.

[Screen 15]



And at last you can see, it has been configured and running successfully, as you can see in screen 16.

[Screen 16]


Internet Information Services
IDS-CPU-932 (local computer)
Web Sites
Default Web Site
IISHelp
Printers
Reports
ReportServer
CrystalReportWebFormViewer2
aspnet_client
WCFApplication
Default SMTP Virtual Server

ServiceApp Service

You have created a service.

To test this service, you will need to create a client and use it to call the service. You can do this using the svcutil.ex
syntax:

```
svcutil.exe http://ids-cpu-932.idsil.loc/WCFApplication/ServiceApp.svc?wsdl
```

This will generate a configuration file and a code file that contains the client class. Add the two files to your client ap
the Service. For example:

C#

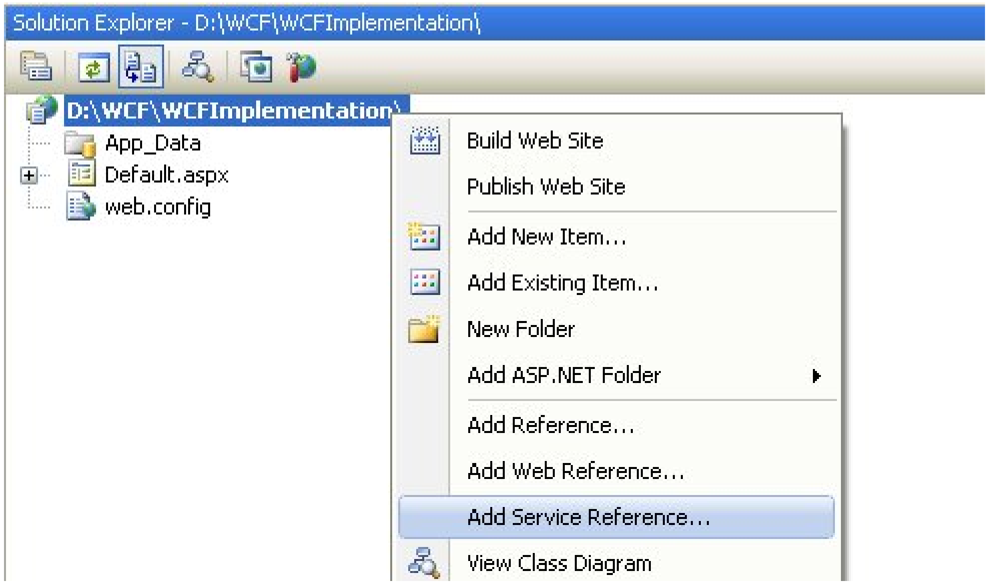
This shows that, we have successfully configured the WCF Service Application with IIS Server.

Please copy the above code from the screen 16, keep this code handy as you may need this in next section. i.e. http://ids-cpu-932........ etc

Implementing WCF App Service

Now we are going to implement the WCF App Service to our project. We created a new website named as “WCFImplementation” and added a *service reference* to the site, as shown in Screen 17.

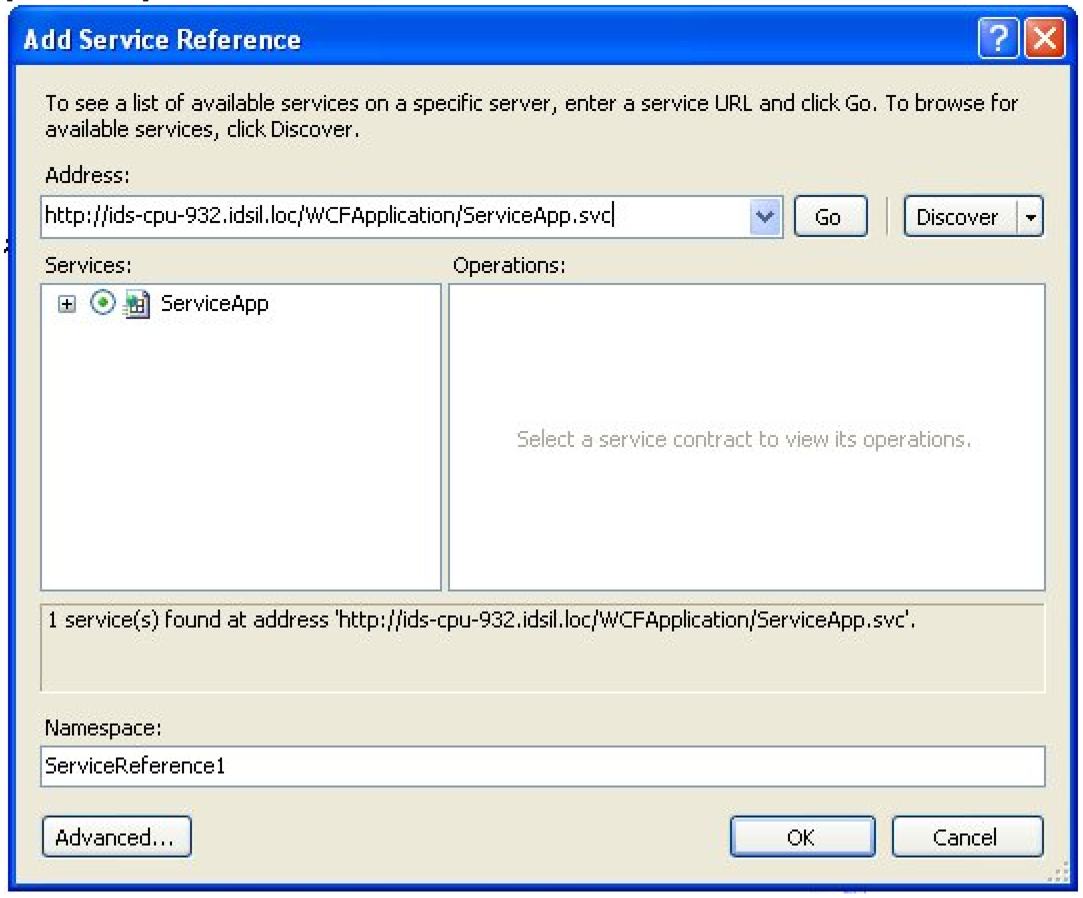
[Screen 17]



Now a new window will be open, where you need to paste the code that you will copy from screen 16, as described above.

Paste the code under Address section and click on ‘Go’ command button, and it will automatically find the service as shown in screen 18.

[Screen 18]



To see a list of available services on a specific server, enter a service URL and click Go. To browse for available services, click Discover.

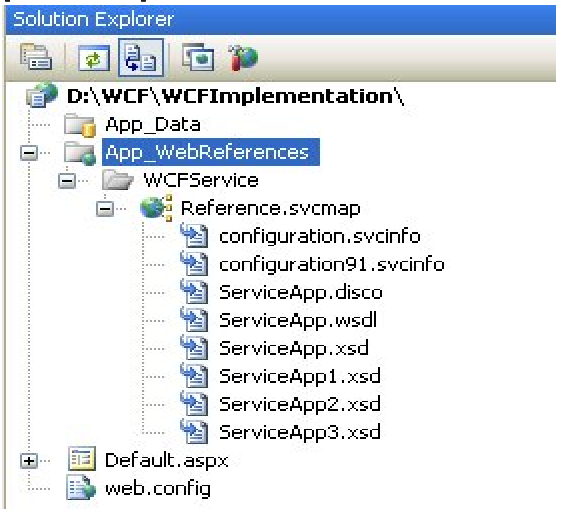
Address:
http://ids-cpu-932.idsil.loc/WCFApplication/ServiceApp.svc

1 service(s) found at address 'http://ids-cpu-932.idsil.loc/WCFApplication/ServiceApp.svc'.

Namespace:
ServiceReference1

You can rename the Namespace according to your preference; we are changing this to ‘WCFService’. Click ‘OK’ and WCFService will be added to our site, as shown in screen 19.

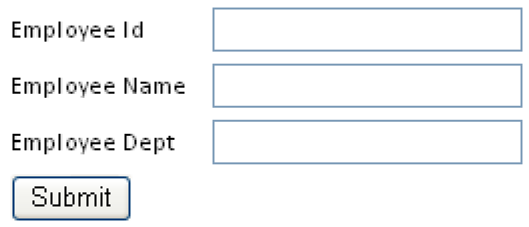
[Screen 19]



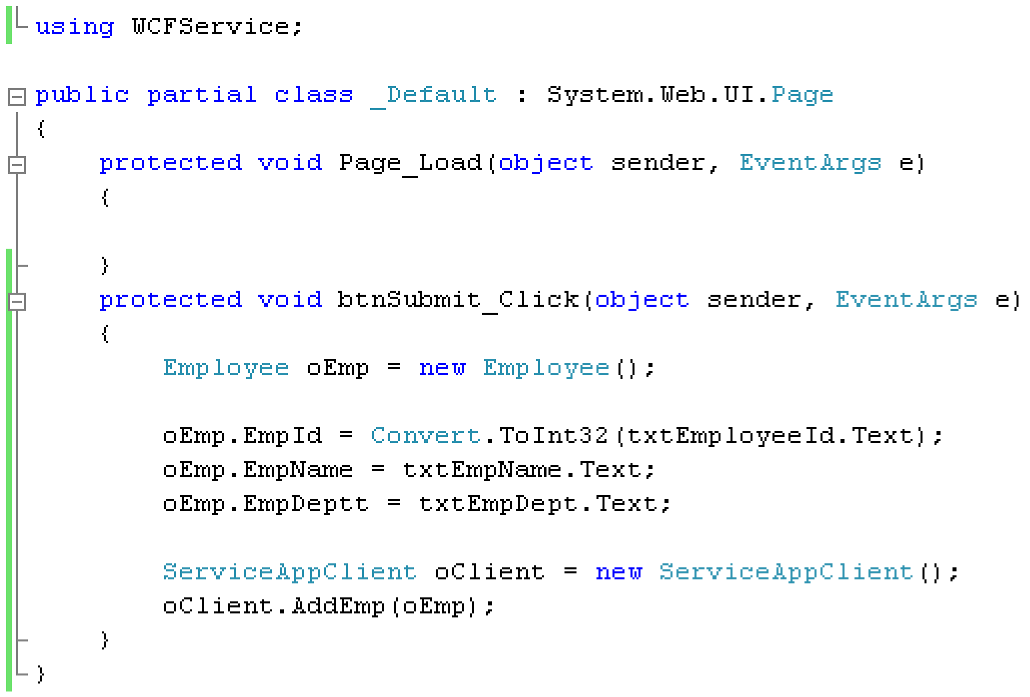
```
Solution Explorer
D:\WCF\WCFImplementation\
  App_Data
  App_WebReferences
    WCFService
      Reference.svcmap
        configuration.svcinfo
        configuration91.svcinfo
        ServiceApp.disco
        ServiceApp.wsdl
        ServiceApp.xsd
        ServiceApp1.xsd
        ServiceApp2.xsd
        ServiceApp3.xsd
  Default.aspx
  web.config
```

Now we are trying to add records into database using WCF Service, and we created a UI for that, as shown in screen 20 and screen 21.

[Screen 20]



[Screen 21]



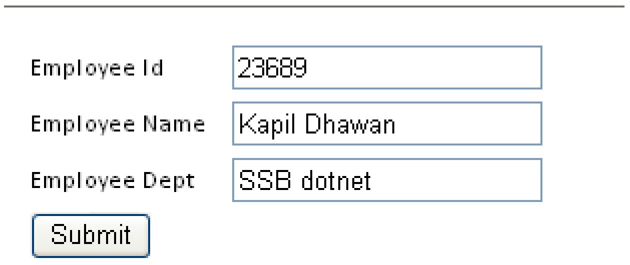
```
using WCFService;

public partial class _Default : System.Web.UI.Page
{
    protected void Page_Load(object sender, EventArgs e)
    {

    }
    protected void btnSubmit_Click(object sender, EventArgs e)
    {
        Employee oEmp = new Employee();

        oEmp.EmpId = Convert.ToInt32(txtEmployeeId.Text);
        oEmp.EmpName = txtEmpName.Text;
        oEmp.EmpDeptt = txtEmpDept.Text;

        ServiceAppClient oClient = new ServiceAppClient();
        oClient.AddEmp(oEmp);
    }
}
```

Here “Employee” is the class that we created in WCF Service Application [Check Screen 3]

And “ServiceAppClient” is a client that is responsible to retrieve the methods from the “ServiceApp.svc”.

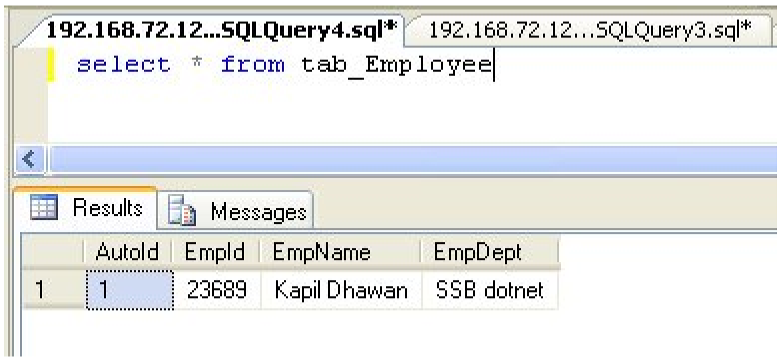
We give some input to UI and submit the information, and we can see that the data was saved in DB, as shown in screen 22 and screen 23.

[Screen 22]



Employee Id	23689
Employee Name	Kapil Dhawan
Employee Dept	SSB dotnet

[Screen 23]



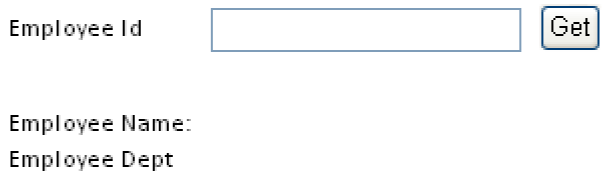
192.168.72.12...SQLQuery4.sql*
192.168.72.12...SQLQuery3.sql*
Results
Messages

```
select * from tab_Employee
```

	AutoId	EmpId	EmpName	EmpDept
1	1	23689	Kapil Dhawan	SSB dotnet

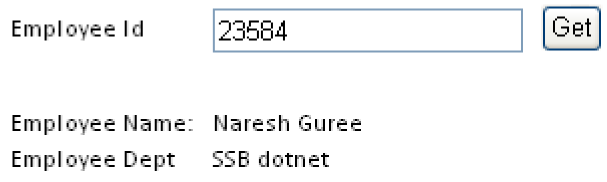
Now we are going to create another UI to fetch the record from the database, as shown in Screen 24.

[Screen 24]



We are giving some input to “EmployeeId” and fetching information based on Employee Id, as shown in Screen 25,

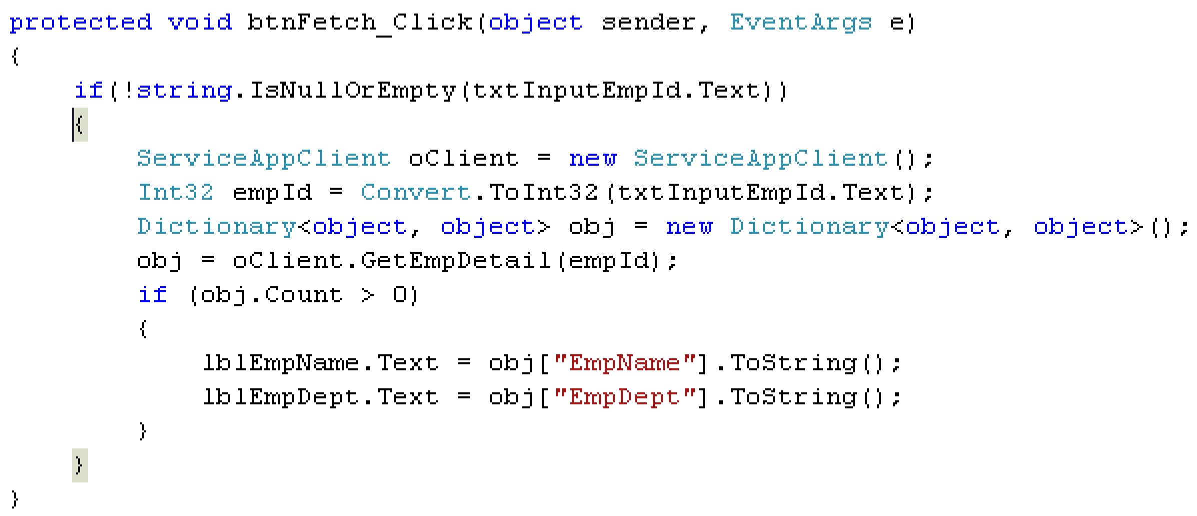
[Screen 25]



Employee Name: Naresh Guree
Employee Dept SSB dotnet

Screen 26 gives you and idea how we implement the code to fetch the record from db using WCF Service.

[Screen 26]



```
protected void btnFetch_Click(object sender, EventArgs e)
{
    if(!string.IsNullOrEmpty(txtInputEmpId.Text))
    {
        ServiceAppClient oClient = new ServiceAppClient();
        Int32 empId = Convert.ToInt32(txtInputEmpId.Text);
        Dictionary<object, object> obj = new Dictionary<object, object>();
        obj = oClient.GetEmpDetail(empId);
        if (obj.Count > 0)
        {
            lblEmpName.Text = obj["EmpName"].ToString();
            lblEmpDept.Text = obj["EmpDept"].ToString();
        }
    }
}
```

Web Services vs WCF

Web Services

ASP.NET Web services were developed for building applications that send and receive messages by using the Simple Object Access Protocol (SOAP) over HTTP. The structure of the messages can be defined using an XML Schema, and a tool is provided to facilitate serializing the messages to and from .NET Framework objects. The technology can automatically generate metadata to describe Web services in the Web Services Description Language (WSDL), and a second tool is provided for generating clients for Web services from the WSDL.

WCF

WCF is for enabling .NET Framework applications to exchange messages with other software entities. SOAP is used by default, but the messages can be in any format, and conveyed by using any transport protocol. The structure of the messages can be defined using an XML Schema, and there are various options for serializing the messages to and from .NET Framework objects. WCF can automatically generate metadata to describe applications built using the technology in WSDL, and it also provides a tool for generating clients for those applications from the WSDL.

The following sections compare Web services and WCF applications using both technologies.

1. Data Representation :

   The development of a Web service with ASP.NET typically begins with defining any complex data types the service is to use. ASP.NET relies on the *XmlSerializer* to translate data represented by .NET Framework types to XML for transmission to or from a service and to translate data received as XML into .NET Framework objects. Defining the complex data types that an ASP.NET service is to use requires the definition of .NET Framework classes that the *XmlSerializer* can serialize to and from XML. Such classes can be written manually, or generated from definitions of the types in XML Schema using the command-line XML Schemas/Data Types Support Utility, xsd.exe.

The following is a list of key issues to know when defining .NET Framework classes that the *XmlSerializer* can serialize to and from XML:

- Only the public fields and properties of .NET Framework objects are translated into XML.
- Instances of collection classes can be serialized into XML only if the classes implement either the *IEnumerable* or *ICollection* interface.
- Classes that implement the *IDictionary* interface, such as *Hashtable*, cannot be serialized into XML.
- The great many attribute types in the *System.Xml.Serialization* namespace can be added to a .NET Framework class and its members to control how instances of the class are represented in XML.

WCF application development usually also begins with the definition of complex types. WCF can be made to use the same .NET Framework types as ASP.NET Web services.

The WCF *DataContractAttribute* and *DataMemberAttribute* can be added to .NET Framework types to indicate that instances of the type are to be serialized into XML, and which particular fields or properties of the type are to be serialized, as shown in Screen 3.

The *DataContractAttribute* signifies that how many type's fields or properties are to be serialized, while the *DataMemberAttribute* indicates that a particular field or property is to be serialized.

The *DataContractAttribute* can be applied to a class or structure. The *DataMemberAttribute* can be applied to a field or a property, and the fields and properties to which the attribute is applied can be either public or private. Instances of types that have the *DataContractAttribute* applied to them are referred to as data contracts in WCF. They are serialized into XML using *DataContractSerializer*.

So here you can notice that all the major difference is *XmlSerializer* and *DataContractSerializer*. So here are the points that show the difference in between both serializations.

a. The *XmlSerializer* and the attributes of the *System.Xml.Serialization* namespace are designed to allow you to map .NET Framework types to any valid type defined in XML Schema, and so they provide for very precise control over how a type is represented in XML.

The *DataContractSerializer*, *DataContractAttribute* and *DataMemberAttribute* provide Very little control over how a type is represented in XML. By not permitting much control over how a type is to be represented in XML, the serialization process becomes highly predictable for the *DataContractSerializer*, and, thereby, easier to optimize. A practical benefit of the design of the *DataContractSerializer* is better performance, approximately ten percent better performance.

b. The attributes for use with the *XmlSerializer* do not indicate which fields or properties of the type are serialized into XML, whereas the *DataMemberAttribute* for use with the *DataContractSerializer* shows explicitly which fields or properties are serialized. Therefore, data contracts are explicit contracts about the structure of the data that an application is to send and receive.
c. The *XmlSerializer* can only translate the public members of a .NET object into XML; the *DataContractSerializer* can translate the members of objects into XML regardless of the access modifiers of those members.
d. As a consequence of being able to serialize the non-public members of types into XML, the *DataContractSerializer* has fewer restrictions on the variety of .NET types that it can serialize into XML. In particular, it can translate into XML types like *Hashtable* that implement the *IDictionary* interface. The *DataContractSerializer* is much more likely to be able to serialize the instances of any pre-existing .NET type into XML without having to either modify the definition of the type or develop a wrapper for it, (You can notice that in screen 8 and screen 26)
e. Another consequence of the *DataContractSerializer* being able to access the non-public members of a type is that it requires full trust, whereas the *XmlSerializer* does not. The Full Trust code access permission give complete access to all resources on a machine that can be access using the credentials under which the code is executing. These options should be used with care as fully trusted code accesses all resources on your machine.

2. Service Development:

To develop a service using ASP.NET, it has been customary to add the *WebService* attribute to a class, and the *WebMethodAttribute* to any of that class' methods that are to be operations of the service. ASP.NET 2.0 introduced the option of adding the attribute *WebService* and *WebMethodAttribute* to an interface rather than to a class, and writing a class to implement the interface. Using this option is to be preferred, because the interface with the *WebService* attribute constitutes a contract for the operations performed by the service that can be reused with various classes that might implement that same contract in different ways.

A WCF service is provided by defining one or more WCF endpoints. An endpoint is defined by an address, a binding and a service contract. The address defines where the service is located. The binding specifies how to communicate with the service. The service contract defines the operations that the service can perform. The service contract is usually defined first, by adding *ServiceContractAttribute* and *OperationContractAttribute* to an interface. The *ServiceContractAttribute* specifies that the interface defines a WCF service contract, and the *OperationContractAttribute* indicates which, if any, of the methods of the interface define operations of the service contract.

Once a service contract has been defined, it is implemented in a class, by having the class implements the interface by which the service contract is defined. A class that implements a service contract is referred to as a service type in WCF.

The next step is to associate an address and a binding with a service type. That is typically done in a configuration file, either by editing the file directly, or by using a configuration editor provided with WCF.

3. Hosting:

ASP.NET Web services are compiled into a class library assembly. A file called the service file is provided that has the extension .asmx and contains a @ WebService directive that identifies the class that contains the code for the service and the assembly in which it is located.

```
<%@ WebService Language="C#" Class="Service, ServiceAssembly" %>
```

The service file is copied into an ASP.NET application root in Internet Information Services (IIS), and the assembly is copied into the \bin subdirectory of that application root. The application is then accessible by using the uniform resource locator (URL) of the service file in the application root.

WCF services can readily be hosted within IIS 5.1 or 6.0, the Windows Process Activation Service (WAS) that is provided as part of IIS 7.0, and within any .NET application. To host a service in IIS 5.1 or 6.0, the service must use HTTP as the communications transport protocol.

The address provided for any endpoint of a WCF service is an address relative to a base address of the endpoint’s host. The host can have one base address for each communication transport protocol. The base address of an endpoint is relative to whichever of the base addresses of the host is the base address for the communication transport protocol of the endpoint. In the sample configuration in the preceding configuration file, the *BasicHttpBinding* selected for the endpoint uses HTTP as the transport protocol, so the address of the endpoint, *EchoService*, is relative to the host's HTTP base address.

4. Client Development:

   Clients for ASP.NET Web services are generated using the command-line tool, WSDL.exe, which provides the URL of the .asmx file as input. The corresponding tool provided by WCF is *Service Model Metadata Tool (Svcutil.exe)*. It generates a code module with the definition of the service contract and the definition of a WCF client class. It also generates a configuration file with the address and binding of the service.

   In programming a client of a remote service it is generally advisable to program according to an asynchronous pattern. The code generated by the WSDL.exe tool always provides for both a synchronous and an asynchronous pattern by default.

   The code generated by the Service *Model Metadata Tool (svcutil.exe)* can provide for either pattern. It provides for the synchronous pattern by default. If the tool is executed with the **/**a sync switch, then the generated code provides for the asynchronous pattern.

   There is no guarantee that names in the WCF client classes generated by ASP. Net's WSDL.exe tool, by default, matches the names in WCF client classes generated by the Svcutil.exe tool. In particular, the names of the properties of classes that have to be serialized using the *XmlSerializer* are, by default, given the suffix Property in the code generated by the Svcutil.exe tool, which is not the case with the WSDL.exe tool.

5. Message Representation:

   The headers of the SOAP messages sent and received by ASP.NET Web services can be customized. A class is derived from *SoapHeader* to define the structure of the header, and then the *SoapHeaderAttribute* is used to indicate the presence of the header. Also, in the ASP.NET syntax, message headers are represented as properties of the service, such as the *ProtocolHeader* property

   The WCF provides the attributes, *MessageContractAttribute*, *MessageHeaderAttribute*, and *MessageBodyMemberAttribute* to describe the structure of the SOAP messages sent and received by a service. In WCF syntax, they are more accurately represented as properties of messages. Also, WCF allows message headers to be added to the configuration of endpoints.

6. Exception Handling:

   In ASP.NET Web services, unhandled exceptions are returned to clients as SOAP faults. You can also explicitly throw instances of the *SoapException* class and have more control over the content of the SOAP fault that gets transmitted to the client.

   In WCF services, unhandled exceptions are not returned to clients as SOAP faults to prevent sensitive information being inadvertently exposed through the exceptions. A configuration setting is provided to have unhandled exceptions returned to clients for the purpose of debugging.

7. Security:

   The options for securing ASP.NET Web services are those for securing any IIS application. Because WCF applications can be hosted not only within IIS but also within any .NET executable, the options for securing WCF applications must be made independent from the facilities of IIS. However, the facilities provided for ASP.NET Web services are also available for WCF services running in ASP.NET compatibility mode.

Remoting vs WCF

WCF:

1. WCF is Microsoft recommended direction
2. WCF was released on January 2006 with Go-Live Licenses, but it will be officially released on November of this year
3. WCF is a 100% Service Oriented Architecture application development platform
4. WCF Security Architecture is interoperable, based on WS-* specifications and it's designed for On-Machine, Cross-Machine, and Cross-Internet scenarios.
5. WCF Transaction Architecture is full flexible, declarative and has method level granularity
6. WCF binary messages are smaller and faster than Remoting's.
7. Services versioning is better dealt with WCF than with Remoting.

.Net Remoting for .Net Framework 2.0:

1. Remoting is not recommended by Microsoft
2. Remoting for .Net Framework 2.0 was released in December 2005
3. Remoting is tightly coupled Object Oriented technology and not loosely coupled Service Oriented.
4. It has security limitations, though now it has the new secure *TcpChannel*
5. With *System.Transactions* we can get similar transaction management capabilities but not in a declarative way.

And of course .Net Remoting doesn't have queued components!

Glossary

<u>Data Member</u>

Define which data types are passed to and from the service. WCF defines implicit contracts for built-in types such as int and string, but we can easily define explicit opt-in data contracts for custom types.

There are two types of Data Contracts.
DataContract - attribute used to define the class
DataMember - attribute used to define the properties.

DataContract is a class comes under the namespace ***System.Runtime.Serialization***.

<u>Contracts</u>

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

WCF defines four types of contracts.

1. Service contracts

Describe which operations the client can perform on the service.
There are two types of Service Contracts.
ServiceContract - This attribute is used to define the Interface.
OperationContract - This attribute is used to define the method inside Interface.

2. Data Contracts
    As described above

3. Fault Contract

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

4. Message Contract

Allow the service to interact directly with messages. Message contracts can be typed or untyped, and are useful in interoperability cases and when there is an existing message format we have to comply with.

ServiceContract class comes from the namespace *System.ServiceModel*
End Points

Every service must have Address that defines where the service resides, Contract that defines what the service does and a Binding that defines how to communicate with the service. In WCF the relationship between Address, Contract and Binding is called Endpoint.

The Endpoint is the fusion of Address, Contract and Binding.

Bindings

A binding defines how an endpoint communicates to the world. A binding defines the transport (such as HTTP or TCP) and the encoding being used (such as text or binary). A binding can contain binding elements that specify details like the security mechanisms used to secure messages, or the message pattern used by an endpoint.

WCF supports nine types of bindings.

Basic binding

Offered by the *BasicHttpBinding* class, this is designed to expose a WCF service as a legacy ASMX web service, so that old clients can work with new services. When used by the client, this binding enables new WCF clients to work with old ASMX services.

TCP binding

Offered by the *NetTcpBinding* class, this uses TCP for cross-machine communication on the intranet. It supports a variety of features, including reliability, transactions, and security, and is optimized for WCF-to-WCF communication. As a result, it requires both the client and the service to use WCF.

Peer network binding

Offered by the *NetPeerTcpBinding* class, this uses peer networking as a transport. The peer network-enabled client and services all subscribe to the same grid and broadcast messages to it.

IPC binding

Offered by the *NetNamedPipeBinding* class, this uses named pipes as a transport for same-machine communication. It is the most secure binding since it cannot accept calls from outside the machine and it supports a variety of features similar to the TCP binding.

<u>Web Service (WS) binding</u>

Offered by the *WSHttpBinding* class, this uses HTTP or HTTPS for transport, and is designed to offer a variety of features such as reliability, transactions, and security over the Internet.

<u>Federated WS binding</u>

Offered by the *WSFederationHttpBinding* class, this is a specialization of the WS binding, offering support for federated security.

<u>Duplex WS binding</u>

Offered by the *WSDualHttpBinding* class, this is similar to the WS binding except it also supports bidirectional communication from the service to the client.

<u>MSMQ binding</u>

Offered by the *NetMsmqBinding* class, this uses MSMQ for transport and is designed to offer support for disconnected queued calls.

<u>MSMQ integration binding</u>

Offered by the *MsmqIntegrationBinding* class, this converts WCF messages to and from MSMQ messages, and is designed to interoperate with legacy MSMQ clients.

Reference:

1. http://www.codeproject.com
2. http://msdn.microsoft.com/

Happy Reading.

******* End of Document *******