



Webbtjänster med API er

Mål med lektionen!

- Titta på hur service:ar fungerar och hur vi programmerar dem.

Vad lektionen omfattar

- WCF Service

WCF Services

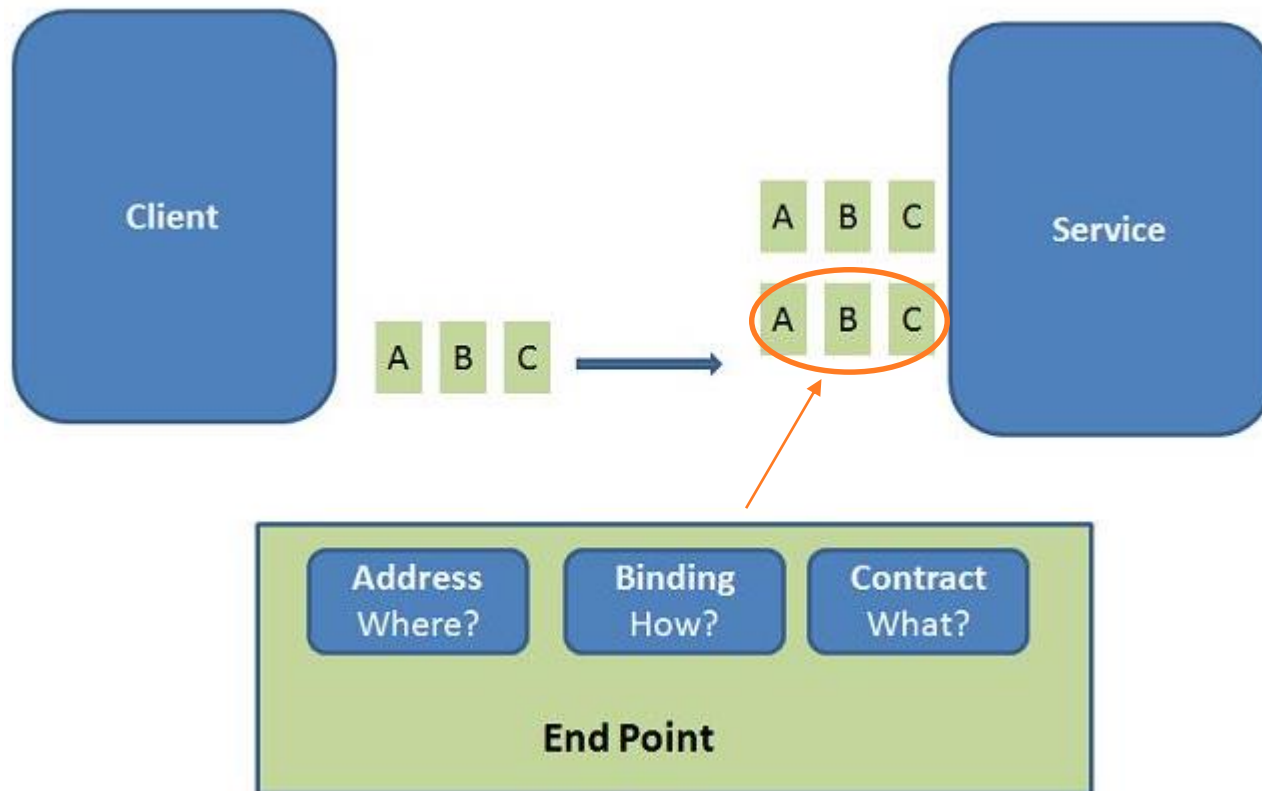
Vad är en WCF service?

En WCF Service är ett program som visar/ställer ut en samling av endpoints. Varje endpoint är en portal för att kommunicera med världen (ifrån servicens ögon sett).

All kommunikation sker genom en/flera endpoints, en endpoint består av tre komponenter:

- **Address** – vart man skall skicka meddelanden.
- **Binding** – hur man skall skicka meddelanden.
- **Contract** – vad meddelanden måste innehålla.

Enkel bild



Behaviors

Med endpoints kontrollerar vi hur klienter kan kommunicera med tjänsten medan med behaviors kan vi påverka hur tjänsten körs lokalt på/inom .Net app domänen.

```
<system.serviceModel>
  <behaviors>
    <serviceBehaviors>
      <behavior name="Metadata">
        <serviceMetadata httpGetEnabled="true" httpGetUrl="http://localhost:8080/evals/meta" />
        <serviceDebug includeExceptionDetailInFaults="true"/>
      </behavior>
    </serviceBehaviors>
  </behaviors>
  <bindings>
    <wsHttpBinding>
      <binding name="NoSecurityPlusRM">
        <reliableSession enabled="true" />
        <security mode="None" />
      </binding>
    </wsHttpBinding>
  </bindings>
  <services>
    <service behaviorConfiguration="Metadata" name="EvalServiceLibrary.EvalService">
      <endpoint address="http://localhost:8080/evals/basic" binding="basicHttpBinding"
        contract="EvalServiceLibrary.IEvalService" />
      <endpoint address="http://localhost:8080/evals/ws" binding="wsHttpBinding"
        bindingConfiguration="NoSecurityPlusRM" contract="EvalServiceLibrary.IEvalService" />
      <endpoint address="net.tcp://localhost:8081/evals" binding="netTcpBinding"
        contract="EvalServiceLibrary.IEvalService" />
      <endpoint address="net.pipe://localhost/evals" binding="netNamedPipeBinding"
        contract="EvalServiceLibrary.IEvalService" />
      <endpoint address="http://localhost:8080/evals/onemore" binding="basicHttpBinding"
        bindingConfiguration="" contract="EvalServiceLibrary.IEvalService" />
      <endpoint address="http://localhost:8080/evals/mex" binding="mexHttpBinding"
        contract="IMetadataExchange" />
    </service>
  </services>
</system.serviceModel>
```

Contracts

I WCF så är alla tjänster exponerade som kontrakt.

Kontrakt är ett plattformsnutralt sätt för att beskriva vad tjänsten gör. I huvudsak är det fyra olika typer av kontrakt inom WCF som är tillgängliga/används.

- Service Contract
- Data Contract
- Message Contract
- Fault Contract

Service Contract

Service kontraktet beskriver operationen/metoden som tjänsten erbjuder. En tjänst kan ha mer än ett kontrakt men minst ett kontrakt. Detta kontrakt definieras som en slags opt-in modell med attribut.

Vi använder [ServiceContract] och [OperationContract] attributen och vi använder en interface definition för att modellera dessa saker eftersom det passar bra in i det vi vill uträtta.

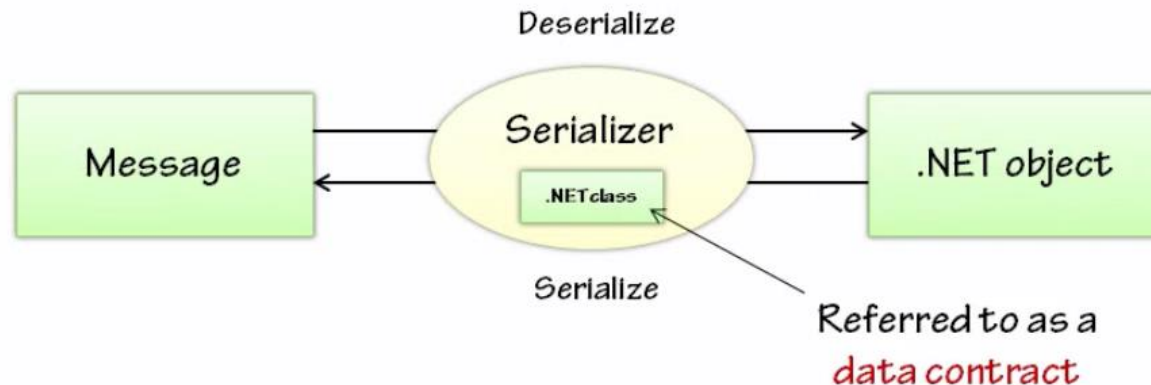
```
[ServiceContract()]
public interface ISimpleCalculator
{
    [OperationContract()]
    int Add(int num1, int num2);
}
```

[En länk till properties för den intresserade.](#)

WCF Messaging

Lite hur WCF Messaging fungerar "under huven".

Alla meddelanden som skickas /tas emot är modellerade efter en (intern .Net) klass som heter Message, som vi kan se som en logisk representation av ett meddelande. Detta (meddelande) innehåller kropp och associerande headers och har olika metoder för att interagera med meddelandet.



Data Contract

För WCF så finns det ett flertal serializers men vi ska titta/använda default varianten som heter DataContractSerializer.

Den kräver att vi explicit kommenterar/noterar vilka typer exakt som vi vill inkludera i meddelandet.

Det finns två attribut som ni kommer att använda en hel del:

- DataContract
- DataMember

```
[DataContract]
public class Eval
{
    [DataMember]
    public string Submitter;
    [DataMember]
    public DateTime TimeSent;
    [DataMember]
    public string Comments;
}
```

Message Contract

WCF använder SOAP meddelanden för att kommunicera, mestadels så kan vi koncentrera oss på DataContract och serialisering av datan osv eftersom WCF normalt tar hand om meddelandet. Men i vissa kritiska lägen så kan det vara bra att kunna ha kontroll över meddelande formatet och i dessa fall har vi MessageContract.

```
[MessageContract]
public class Eval
{
    [MessageHeader]
    public string submitter;
    [MessageBodyMember]
    public DateTime TimeSent;
    [MessageBodyMember]
    public string comments;
}

[ServiceContract]
public interface IEvalService
{
    [OperationContract]
    void SubmitEval(Eval eval);
    [OperationContract]
    List<Eval> GetEval();
}
```

Fault Contract

I vanliga applikationer hanterar vi fel genom try-catch block, i WCF så kan vi skicka denna informationen till klienten om vi använder oss av FaultContract.

```
[DataContract()]
public class CustomException
{
    [DataMember()]
    public string Title;
    [DataMember()]
    public string ExceptionMessage;
    [DataMember()]
    public string InnerException;
    [DataMember()]
    public string StackTrace;
}

[ServiceContract()]
public interface ISimpleCalculator
{
    [OperationContract()]
    [FaultContract(typeof(CustomException))]
    int Add(int num1, int num2);
}

public int Add(int num1, int num2)
{
    //Do something
    CustomException ex = new CustomException();
    ex.Title = "Error Funtion:Add()";
    ex.ExceptionMessage = "Error occur while doing add function.";
    ex.InnerException = "Inner exception message from serice";
    ex.StackTrace = "Stack Trace message from service.";
    throw new FaultException(ex,"Reason: Testing the Fault contract") ;
}

Console.WriteLine("Sum of two numbers... 5+5 =" + proxy.Add(5, 5)
Console.ReadLine();
}
catch (FaultException<MyCalculatorService.CustomException> ex)
{
    //Process the Exception
}
```

Implementera tjänsten

Nu har vi våra kontrakt på plats, och vi kan då börja bygga vår tjänst.

Det vi skall tänka på är att vi bör ärva ifrån i alla fall minst ett ServiceContract (för ärver vi inget så vad är det för tjänst vi erbjuder då?). Implementera alla de metoder som du behöver för att bygga Business logiken.

```
[ServiceBehavior(InstanceContextMode=InstanceContextMode.Single)]
public class EvalService : IEvalService
{
    List<Eval> evals = new List<Eval>();
    public void SubmitEval(Eval eval)
    {
        evals.Add(eval);
    }

    public List<Eval> GetEval()
    {
        return evals;
    }
}
```

Hosting WCF Services

self-hosting
IIS hosting
WAS hosting

self hosting Övning

Skapa en tom solution och lägg till ett projekt av typen WCF Service Library, ta bort de två filerna som VS skapade och skapa en ny klass i stället.

Skapa en publik (DataContract) klass som heter Eval och har följande publika properties (DataMember):

- string Submitter
- DateTime Timesent
- string Comments

Skapa en ServiceContract klass som heter IEvalService med följande OperationContract's

- void SubmitEval(Eval eval);
- List<Eval> GetEvals();

self hosting Övning

Ärv sedan ServiceContract i din service klass och ovanför där du deklarerat service klassen skall vi lägga till följande:

```
[ServiceBehavior(InstanceContextMode=InstanceContextMode.Single)]  
public class EvalService : IEvalService
```

Skapa en lista av typen Eval och i metoden SubmitEval så skall vi adda den eval parametern till vår lista.

I metoden GetEvals så skall vi enkelt returnera listan med Evals.

self hosting Övning (hosten)

I vår solution så skall vi lägga till en ny konsoll applikation som heter ConsoleHost.

Referera in EvalServiceLibrary, System.ServiceModel.

Och gör using på dem i Program klassen som skapades.

Därefter skapa en instans av ServiceHost klassen som heter host och är av typen EvalService.

self hosting Övning (hosten)

- Skapa upp en try-catch
- I try delen så skall du öppna hosten.
 - skapa en instans av EvalService
 - en instans av Eval,
- Ge eval-instansen lite värden och anropa SubmitEval och skicka in eval-instansen.
- Därefter anropa GetEval och tilldela retur värdet till en Eval lista som ni skapat.
- Sedan för varje Eval i listan så skall ni skriva ut värdet på dess properties som finns där.
- Stäng hosten när ni är klara.
- I catchen anropa Abort på hosten.

self-hosting

```
class Program {  
    static void Main(string[] args) {  
        ServiceHost host =  
            new ServiceHost(typeof(InvoiceService));  
        ... // configure the host before opening  
  
        try {  
            host.Open();  
            Console.ReadLine();  
            host.Close();  
        }  
        catch (Exception e) {  
            Console.WriteLine(e);  
            host.Abort();  
        }  
    }  
}
```

IIS Host Övning

- Skapa en folder på er dator, samt en subfolder som heter App_Code.
- Öppna IIS
- Öppna mappen sites
- Skapa en site, ge den ett sitenamn och den fysiska sökvägen skall vara till mappen ni skapade precis.
- Bindningen skall vara http, hostnamnet:www.(ert sitenamn).com.
- Högerklicka på siten ni skapade och välj "Add Application", skriv in "IISHostedCalc" som Alias och peka på nytt ut samma fysiska sökväg.

IIS Host Övning

I den första foldern ni skapade, där skall ni skapa följande filer:

- Service.svc
- Web.config

I App_Code mappen skall ni skapa följande fil:

- Service.cs

.svc filen skall innehålla följande:

```
<%@ServiceHost language=c# Debug="true" Service="Microsoft.ServiceModel.Samples.CalculatorService"%>
```

Vi måste också lägga in följande rad i host filen:

127.0.0.1 *www.sitenamn.com*

IIS Host Övning

.cs filen skall se ut enligt följande:

```
using System;
using System.ServiceModel;

namespace Microsoft.ServiceModel.Samples
{
    [ServiceContract(Namespace="http://Microsoft.ServiceModel.Samples")]
    public interface ICalculator
    {
        [OperationContract]
        double Add(double n1, double n2);
        [OperationContract]
        double Subtract(double n1, double n2);
        [OperationContract]
        double Multiply(double n1, double n2);
        [OperationContract]
        double Divide(double n1, double n2);
    }

    public class CalculatorService : ICalculator
    {
        public double Add(double n1, double n2)
        {
            return n1 + n2;
        }
        public double Subtract(double n1, double n2)
        {
            return n1 - n2;
        }
        public double Multiply(double n1, double n2)
        {
            return n1 * n2;
        }
        public double Divide(double n1, double n2)
        {
            return n1 / n2;
        }
    }
}
```

IIS Host Övning

Web.config skall se ut enligt följande:

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <system.serviceModel>
    <behaviors>
      <serviceBehaviors>
        <behavior name="Metadata">
          <serviceMetadata httpGetEnabled="true" httpGetUrl="http://www.hostedcalc.com/IISHostedCalc/mex" />
        </behavior>
      </serviceBehaviors>
    </behaviors>
    <services>
      <!-- This section is optional with the default configuration
           model introduced in .NET Framework 4 -->
      <service name="Microsoft.ServiceModel.Samples.CalculatorService" behaviorConfiguration="Metadata">

        <!-- This endpoint is exposed at the base address provided by host:http://www.hostedcalc.com/IISHostedCalc/service.svc -->
        <endpoint address=""
                  binding="wsHttpBinding"
                  contract="Microsoft.ServiceModel.Samples.ICalculator" />

        <!-- The mex endpoint is exposed at http://www.hostedcalc.com/IISHostedCalc/service.svc/mex -->
        <endpoint address="mex"
                  binding="mexHttpBinding"
                  contract="IMetadataExchange" />

      </service>
    </services>
  </system.serviceModel>
</configuration>
```

Configuring Services

Det finns två sätt för oss att konfigurera vår tjänst när/om vi behöver det.

Antingen explicit i program koden eller deklarativt i din konfigurations fil. Saker som vi kan konfigurera är endpoints och behaviors, sedan är det så att kan du göra det i kod så kan du göra det i konfigurations filen i alla fall när det gäller konfigurera tjänsten.

Endpoint: address

Endpoints är vanligtvis konfigurerad med en unik adress.

Endpoints kan "dela" en adress om de "delar" samma binding instans, men de måste då vara konfigurerade med olika kontrakt.

Adresser måste alltid vara i forma av en URI vilken är beroende på vilken typ av transport som vi använder för vår endpoint, och transporten är bestämd av vår bindning.

Endpoint: address

Transport	Scheme	Example
HTTP	http://	http://pluralsight.com/myservice
TCP	net.tcp://	net.tcp://pluralsight.com:8081/myservice
Pipes	net.pipe://	net.pipe://pluralsight.com/myservice
MSMQ	net.msmq://	net.msmq://pluralsight.com/private/myservice

De vanligaste transport typerna och deras protokoll schema tillsammans med exempel på hur uri'n skulle se ut.

Endpoint: binding

En binding är som ett recept som specificerar tre olika kommunikations detaljer.

Binding

Transport Protocol: HTTP, TCP, MSMQ, *etc*

Message format: XML, MTOM, JSON, binary, *etc*

Message protocols: SOAP, WS-*, none, *etc*

Konfigurera bindings

Precis som jag sagt innan så kan vi göra detta både med kod och i config-filen.

```
BasicHttpBinding b = new BasicHttpBinding();  
b.Security.Mode = BasicHttpSecurityMode.Transport;  
b.Security.Transport.ClientCredentialType =  
    HttpClientCredentialType.Basic;  
  
host.AddServiceEndpoint(  
    typeof(IInvoiceService), b,  
    "http://server/invoiceservice");
```

kod

config

```
<configuration>  
  <system.serviceModel>  
    <services>  
      <service name="InvoiceService">  
        <endpoint  
          address="https://server/invoiceservice"  
          binding="basicHttpBinding"  
          bindingConfiguration="MyBindingConfiguration"  
          contract="IInvoiceService"/>  
        ...  
      </service>  
    </services>  
    <bindings>  
      <basicHttpBinding>  
        <binding name="MyBindingConfiguration">  
          <security mode="Transport">  
            <transport clientCredentialType="Basic" />  
          </security>  
        </binding>  
      </basicHttpBinding>  
    </bindings>
```

Lägga till behaviors till tjänsten

Precis som vi kan konfigurera bindings och endpoints med kod så kan vi konfigurera vår tjänst med både kod och i configfilen.

```
ServiceHost host =  
    new ServiceHost(typeof(InvoiceService));
```

```
ServiceMetadataBehavior smb =  
    new ServiceMetadataBehavior();  
smb.HttpGetEnabled = true;  
  
host.Description.Behaviors.Add(smb);
```

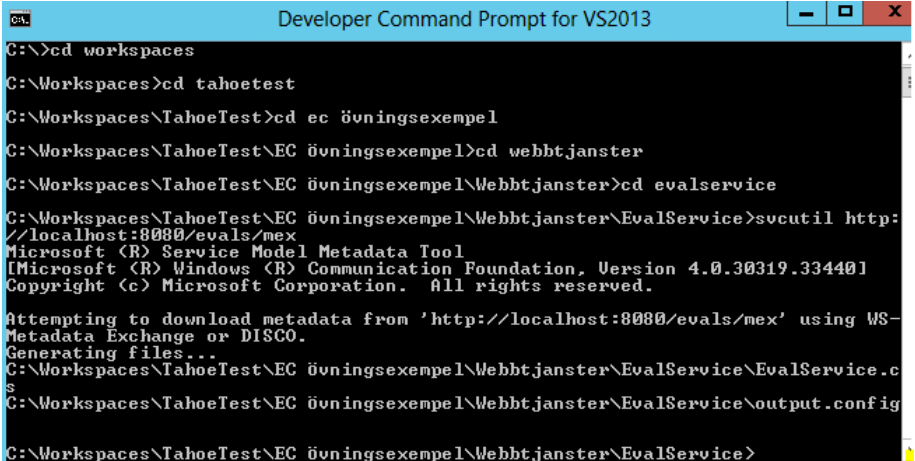
```
<configuration>  
  <system.serviceModel>  
    <services>  
      <service name="InvoiceService"  
        behaviorConfiguration="MetadataBehavior">  
        ...  
      </service>  
    </services>  
    <behaviors>  
      <serviceBehaviors>  
        <behavior name="MetadataBehavior">  
          <serviceMetadata httpGetEnabled="true" />  
        </behavior>  
      </serviceBehaviors>  
    </behaviors>
```

Publisha metadata

Om vi gör som jag visade i förra sliden så kommer WCF att automatiskt

vilket gör att servicen automatiskt publishar upp metadata om tjänsten i form av en wsdl fil som alla tilltänkta klienter kan ladda ner och ev bygga en klient av/med hjälp av för att kunna kommunicera med vår tjänst.

```
<endpoint address="http://localhost:8080/evals/mex" binding="mexHttpBinding"
          contract="IMetadataExchange" />
```



```
Developer Command Prompt for VS2013
C:\>cd workspaces
C:\Workspaces>cd tahoeTest
C:\Workspaces\TahoeTest>cd ec övningsexempel
C:\Workspaces\TahoeTest\EC övningsexempel>cd webbtjanster
C:\Workspaces\TahoeTest\EC övningsexempel\Webbtjanster>cd evalservice
C:\Workspaces\TahoeTest\EC övningsexempel\Webbtjanster\EvalService>svcutil http:
//localhost:8080/evals/mex
Microsoft (R) Service Model Metadata Tool
[Microsoft (R) Windows (R) Communication Foundation, Version 4.0.30319.33440]
Copyright (c) Microsoft Corporation. All rights reserved.

Attempting to download metadata from 'http://localhost:8080/evals/mex' using WS-
Metadata Exchange or DISCO.
Generating files...
C:\Workspaces\TahoeTest\EC övningsexempel\Webbtjanster\EvalService\EvalService.c
s
C:\Workspaces\TahoeTest\EC övningsexempel\Webbtjanster\EvalService\output.config
C:\Workspaces\TahoeTest\EC övningsexempel\Webbtjanster\EvalService>
```

Exceptions

Hur får vi exceptions att sprida sig från vår tjänst "ner" till klienten?

Exceptions är egentligen väldigt teknik specifika och de kan inte korsa den gräns som utgörs av vår tjänst -> klienten.

WCF kommer automatiskt att översätta alla ohanterade exceptions till SOAP fel som skickas vidare och dessa är väldigt generiska som inte säger särskilt mycket.

```
public void DoSomething(string input) {  
    ...  
    throw new FaultException("Something bad happened");  
}
```

WCF vs Webservice

För att kunna veta skillnaden på dessa två behöver vi veta vad de är.

Webservice:

"... En webbtjänst är en metod för kommunikation mellan två elektroniska enheter via World Wide Web. Det är en programvarufunktion via en nätverksadress tillgänglig över webben som alltid är igång som i begreppet utility computing ".

Webbtjänst är en mekanism som tillhandahåller data som en tjänst/svar över http-protokollet på begäran av andra program.

WCF vs Webservices

WCF:

Windows Communication Foundation (WCF) är ofta använd för att utveckla och distribuera nätverks distribuerade tjänster baserade på principerna för Service Oriented Architecture (SOA). De grundläggande egenskaperna för WCF är samverkan med andra system. Detta ger dig en mer hanterbar strategi för att skapa och konsumera webbtjänster.

WCF vs Webservices

Egenskaper:

WCF tjänster är definierade av ServiceContract och OperationContract egenskaperna medans en Web service är definierad av WebService och WebMethod egenskaperna.

Protokoll:

WCF stödjer en rad protokoll: HTTP, Named Pipes, TCP och MSMQ, medans webservice endast stödjer HTTP.

WCF vs Webservices

Hosting tekniker:

Olika aktiverings tekniker är tillgängliga för WCF Hosting (IIS, WAS, Self-Hosting och Windows Service), medans en webservice endast hostas genom IIS.

Serializer:

WCF stödjer DataContractSerializer genom att använda System.Runtime.Serialization, när webservicarstödjer XML Serialization genom att använda System.XML.Serialization.

WCF vs Webservices

HashTable:

Det är möjligt att serialisera en hashtabell i WCF men inte för en webservice.

Bindings:

WCF stödjer flertalet olika bindningar medans en webservice endast stödjer SOAP eller XML.