تکنولوژي کاملا کارآمد است.وقتي شما مي توانيد راه حل هايي را براي رهايي از مشکلات گوناگون با استفاده WCF صدق WCF از اصول مشابه فراهم کنيد مي دانيد که شما يک کارهوشمندانه اي به کار گرفته ايد.اين موضوع در مورد و سرويس هاي named pipe، TCP، Java، PHP، ASMX مي کند با راه اندازي تنها يک سرويس ،شما مي توانيد به براي هر نوع ارتباطي که نياز به پشتيباني XML دسترسي داشته باشيد تنها با اضافه کردن يک عنصر JSON بر پايه شما مي توانيد به هر يک از اين نوع سرويس ها متصل شويد و WCF دارد.در طرف ديگر ،تنها با يک سرويس گيرنده امکان پذير است.در اين مقاله شما چگونگي دسترسي به XML دوباره به طور مشابه با اضافه کردن تنها يک عنصر خواهيد ديد.در اين مقاله هيچ پروکسي ،کد توليد شده اي ،ابزار کمکي و SilverLight 2 با استفاده از WCF سرويس .به کاربرده نخواهد شد Add Service Reference استفاده از

راه اندازي Service

وقتي با WCF سر کار داريد شما با يک سيستم کاملا کارآمد در حال کار هستيد.در اين سيستم اصطلاح ABC مفهوم پايه اي می باشد که مخفف آدرس(address)،بايند(binding) و قرارداد(contract) است.اين سيستم براي هر ارتباطي در همه جا استفاده مي شود حتي وقتی که در حالي صحبت با يک شخص ديگر هستيد در اين زمان شما بايد بدانيد چه کسي هست،در چه مورد هست،و چگونه است.اگر شما اين سه تا را نداريد پس هيچ ارتباطي نمي توانيد داشته باشيد.

توسط اين سه تکه از اطلاعات ،شما يا يک endpoint طرف سرور(service-side) ايجاد مي کنيد که سرويس گيرنده به دسترسي خواهد داشت يا يک کانال(channel) طرف سرويس گيرنده(client-side) که سرويس گيرنده براي ارتباط با سرويس استفاده خواهد کرد.
سرويس هاي WCF با يک متد سه مرحله اي نصب مي شوند:
۱-ايجاد يکservice contract با يکي يا چندين operation service
۲-ايجاد يک service implementation براي اين قراردادها(contract)
۳-تنظيم کردن يک ميزبان سرويس(service host) که فراهم مي کند پياده سازي را با يک endpoint براي قرارداد(contract)مشخص.

تعريف service contract

يک اينترفيس ساده در.net که صفتSystem.ServiceModel.ServiceContractAttribute براي آن پذيرش مي شود.اين اينترفيس حاوي قراردادهاي عملياتي(operation contract) گوناگوني خواهد بود که متدهاي امضا شده اي سادهاي هستند با System.ServiceModel.OperationContractAttribute براي هر کدام پذيرفته شده است.هر دوي اين صفات در اسمبلي System.ServiceModel است.
هيچ شرطي مشخصي مستقيما براي پياده سازي صفت ServiceContract پذيرش نمي شود(مانند کلاس).قابليت براي انجام اين کار احتمالا بدترين ويژگي در WCF است.اين تمام اهداف کاربرد WCF را الغا مي کند :آدرس شما ، بايند شما ،قرارداد شما و پياده سازي شما کاملا مجزا مي باشد.اگر پياده سازي قراردادهايتان اصولي نباشد همه

فايل هاي پيكر بندي تان براي افراد آشنا با WCF گيج كننده مي تواند باشد.وقتي ما يک قرار داد (contract) را جستجو مي كنيم دنبال چيزي با حرف آغازين "I" مي گرديم.
كد زير طريقه تعريف قراراداد(contract) كه در اين مقاله استفاه مي شود را نشان مي دهد.

```
Using System;
Using System.SrviceModel;
Namespace Contract.Service
{
[ServiceContract(NameSpace=Information.Namespace.Contact)]
public interface IPersonService
    {
        //- GetPersonData -//
        [OperationContract]
        Person GetPersonData(String personGuid);
    }
}
```

به خاطر داشته باشيد هر وقت شما براي WCF طراحي مي كنيد تا آنجا كه ممكن است اينترفيس تان را ساده نگه داريد.قانون كلي مورد اشاره اينكه شما در بعضي مواقع بين ٣ تا ٧ عملگر(operation) براي هر قرار داد سرويس(service contract) بايد داشته باشيد وقتي شما ١٢تا ١٤ مورد مشخص مي كنيد اين زماني كه بايد در مورد عملگرهايتان بررسي دوباره انجام دهيد. اين طراحي خيلي مهمي است وما جلوتر دوباره اشاره خواهيم كرد كه ما بالاتر از ده ها قرار داد سرويس(service contract) برا ي هر سرويس خواهيم داشت.شما بايد دائم در ذهن داشته باشيد كه هدفتان براي ايجاد اين سرويس ، منطبق بر ديد SOA باشد.سرويس هاي WCF را نبايد مانند framework طراحي كرد كه نقاط دسترسي زيادي داشته باشد.

خصوصيتNamespace روي صفتي قرار مي گيرد كه فضاي نام مورد استفاده توسط سرويس هاي سازمان دهي منطقي مشخص مي سازد.شبيه .NET كه براي جدا سازي كلاس ها ،ساختارها(structs)واينترفيس هاي متفاوت،سرويس هاي SOAP براي جدا سازي فعاليت هاي گوناگون از فضاي نام ها(namespaces) استفاده مي كنند.فضاي نام شايد به طور دلخواه انتخاب شود،اما سرويس گيرنده (client) و سرويس بايد روي اين فضاي نام توافق داشته باشند.در اين مورد ،فضاي نام URI است.اين فضاي نام همچنين برا ي سرويس گيرنده(client)خواهد بود.اين يک URL(universal resource locator) فيزيكي نيست،بلكه يک URI(universal resource identifier) منطقي مي باشد.هر دو اصطلاح تفاوت زيادي نسبت به هم ندارند .همه urlها، uriها هستند،اما عكس آن درست نيست.

باتوجه به اين اينترفيس،يک متد واسط وجود دارد كه Person را برمي گرداند.اين يک قرارداد داده(data contract) مي باشد قرارداد داده(data contract) ها كلاس هايي هستند كه صفتSystem.Runtime.Serialization.DataContractAttribute كه براي آنها پذيرفته مي شوند.آنها يک يا چندين اعضا داده(data members)دارند،كه ويژگي (property) يا فيلد خصوصي((private يا عمومي (public) هستند كه

صفت System.Runtime.Serialization.DataMemberAttribute براي آنها پذيرفته مي شود.هر دوي اين صفات در اسمبلي System.Runtime.Serialization وجود دارند.نكته مهم اين است كه شايد شما فرض كنداين صفات در اسمبلي System.ServiceModel تعرف شده اند در اين صورت قرارداد شما كامپايل نمي شود.

آنچنان كه گفتيم اعضا داده ها(data members)، ويژگي (property) يا فيلدي كه خصوصي(private) يا عمومي(public) هستند و برخلاف سريالايز كننده ها (serializer) كه تنها براي عمومي (public) بودن كاربرد داشتن.اين امكان مخفي سازي از گسترش دهنده به شما را مي دهد اما به سرويس ها اجازه استفاده را مي دهد.چگونگي تمايز بين صفت DataContract با صفت Serializable اين گونه است كه وقتي شما از صفت Serializable استفاده مي كنيد شما مدل opt-out به كار مي برید يعني وقتي صفتي برا ي كلاسي پذيرفته مي شود هرعضو آن قابل سريالايز است.شماسپس فيلدهاي مشخص opt-out (نه ويژگي ها((properties)با استفاده از صفت System.NonSerializedAttribute

مشخص كنيد.در طرف ديگر وقتي شما صفت DataContract را مي پذيري،شما يک مدل opt-in را استفاده مي كنيد.بنابراين زماني اين صفت پذيرفته مي شود كه شما بايد در هر ويژگي (property) يا فيلد opt-in شما مي خواهيد سريالايز شود توسط صفت DataMember پذيرفته شود.قرارداد داده Person در زير آمده است:

```
[DataContract(Namespace = Information.Namespace.Contact)]
public class Person
{
    //- @Guid -//
    [DataMember]
    public String Guid { get; set; }
//- @FirstName -//
    [DataMember]
    public String FirstName { get; set; }
//- @LastName -//
    [DataMember]
    public String LastName { get; set; }
//- @City -//
    [DataMember]
    public String City { get; set; }
//- @State -//
    [DataMember]
    public String State { get; set; }
//- @PostalCode -//
    [DataMember]
    public String PostalCode { get; set; }
}
```

مهم که شما بدانید چه چيزي شما مي خواهي به سمت سرويس گيرنده((client بفرستيد. فرض کنيد شما شي تجاري داخلي با ۱۰،۰۰۰ ويژگيproperty)) داريد به اين معني نيست که سرويس گيرنده شما همواره خواهد توانست که اين را کنترل کند.خواست هاي تجاري شما هرگز مکان جامعه را تغيير نخواهد داد.شما بايد با اين سناريو مشخص سرويس گرا را در ذهنتان طراحي کنيد.در مورد سيلورلايت ،اين موضوع از اهميت بيشتري برخوردار است به اين خاطر که شما با اطلاعاتي سرو کار داريد که قبل از اين که در plug-in سيلورلايت قادر به ديدن آنها باشد توسط م مرورگر وب دريافت مي شود.هر بار که شما ويژگي اضافي را روي سيم بفرستيد،شماموجب کاهش پاسخگويي برنامه سيلورلايت شما مي شويد.

اگر شما يک سيستم راطراحي کنيد که روي سيم استفاده مي شود،بايد بعضي احتمالات را نيزمورد بررسي قرار گيرد. امنيت،کارايي و طراحي API مناسب امکاناتي از برنامه نيستند بلکه آنها قسمت هاي هسته سيستم مي باشند.مثلا طراحي ۱۰ کلاس متفاوت ،که نمايش مي دهند يک ويژگي(property) که در کلاس ديگر استفاده خواهد شد به نوبت سريالايز و روي سيم منتقل مي شوند.اين به قدري سنگيني که هيچ کسي نمي تواند آن را کنترل کند.اگر شما بيش از ۱۵ ويژگي پيرامون گراف شي داريد.مطمئنا زماني که دوباره فکر کنيد که چه چيزي را مي خواهيد ارسال کنيد و ابدا يک System.Data.DataSet روي سيم نفرستيد.اين بسيار حجيم وباعث انتقال ۱۰،۰۰۰ ويژگي داده مي شود که درظاهر شي سبکي مي رسيد.در حقيقت چيزي که قابل سريالايز است به اين معني نيست که خود چيز سريالايز مي شود.
اين علت اصلي که نبايد براي همه کلاس ها صفت Serializable را پذيرفت.يادآوري،اين صفت يک مدل opt-out (يک ضعف در آن)دنبال مي کند.اگر شما مي خواهيد که اشيا تجارتي روي فريمورک شما و همچنين روي سيم کار کند بايد صفت Serializable حذف وصفت DataContract را بپذيريد.اين به شما اجازه مي دهد که از طريق صفت DataMember مشخص کني که ويژگي ها(properties)نيز روي سيم قابل استفاده باشند،در عين حال فريمورک موجود شما را کاملا دست نخورده باقي مي گذارد و اين علت وجود صفتDataContract مي باشد.مايکروسافت معين کرد که صفت Serializable براي اهداف SOA به اندازه کافي خوب به نظر نمي رسد.آنها همچنين مشخص کردند که دليلي براي اجبار هرکس در جهان به منظور نوشتن اشياي داده انتقالي براي هر عملي((operation نيست.حتي پس از آن،همانطور که شما بايد تا آنجا که ممکن خصوصي(private) و يا داخلي ((internal نگه داريد شما بايد تا آنجا که ممکنه غير قابل سريالايز قرار دهيد.

مرحله دوم ،ما با استفاده از اين قراردادها(contracts) يک پياده سازي ايجاد مي کنيم.پياده سازي سرويس فقط يک کلاسي که يک قرارداد سرويس((service contract را پياده سازي مي کند.پياده سازي سرويس براي ما واقعا خيلي ساده است:

```
using System;
//+
namespace Contact.Service
{
    public class PersonService : Contact.Service.IPersonService
    {
```

```
        //- @GetPersonData -//
        public Person GetPersonData(String personGuid)
        {
            return new Person
            {
                FirstName = "John",
                LastName = "Doe",
                City = "Unknown",
                Guid = personGuid,
                PostalCode = "66062",
                State = "KS"
            };
        }
    }
}
```

مرحله سوم یک میزبان سرویس را توسط endpoint مناسب پیکربندي کنیم.در این مقاله ما از یک سرویس بر پایه HTTP استفاده مي کنیم.بنابراین بعد از آنکه ما یک وب سایت جدید راه اندازي کردیم،ما یک فایل Person.svc در ریشه پروژه ایجاد وبه علاوه یک هدایتگر سرویس (service directive) براي پیاده سازي سرویسمان مشخص مي کنیم.این فایل Person.svc است:

```
<%@ ServiceHost Service="Contact.Service.PersonService" %>
```

اگر شما پیاده سازتان در این کلاس باشد آنگاه استفاده درستي از WCF نخواهید داشت.در WCF،شما آدرستان ،بایندهایتان،قراردادهایتان،و پیاده سازي هایتان را کاملا جدا مي سازید.با قرار دادن پیاده سازیتان در این فایل،شما آدرس را به پیاده سازي متصل کرده اید.این کار تمام اهداف WCF را از بین مي برد.بنابراي کد بالا تمام آن چیزي است که باید در یک فایل svc قرار گیرد.

این یک میزبان سرویس پیکربندي نشده است بنابراین شما با قرار دادن یک endpoint در فایل web.config سرویس وب سایت ،آدرس وبایندینگ و قرارداد را مشخص مي کنید.مانند زیر:

```
<?xml version="1.0" encoding="UTF-8"?>
<configuration>
  <system.serviceModel>
    <services>
      <service name="Contact.Service.PersonService">
        <endpoint address="" binding="basicHttpBinding" contract="Contact.Service.IPersonService"
/>
      </service>
    </services>
```

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

اين مي تواندbasicHttpBinding باشد كه ازطريقContact.Service.IPersonService درآدرس Person.svcبه Contact.Service.PersonService مرتبط است.

آدرس مشخص شده يک آدرس نسبي است و مقدار اين صفت به آدرس پايه اضافه مي کند وآدرس پايه توسط وب سرور ما مشخص مي شود و در مورد يک سرويس خارج از يک وب سرور،شما مي توانيد اينجا يک آدرس مطلق مشخص کنيد.اما وقتي وب سرور به کار مي برید،وب سرور بايند هاي پورت و آدرسIP را کنترل مي کند.سرويس ما در Person.svc هست،بنابراين قبلا براي ما URL پايه فراهم کرده است.در اين مورد آدرس خالي است،از اين صفت وقتي استفاده مي شود وقتي چندين endpointبيشتري داريد آنچنان که شما بعدا خواهيد ديد.

بايند مشخص مي کند که اطلاعات چگونه براي انتقال فرمت شوند.اين در واقع تنها يک کالکشن از عناصر بايند(binding element) و پيش فرض از قبل تنظيم شده است که به آساني تنظيمات تغيير داده مي شوند.هر عنصر بايند حداقل دوتا عنصر بايند دارد.

يکي از اينها عنصر بايند رمزگذاري پيغام (message encode binding element) که چگونه پيغام فرمت خواهد شد.براي مثال ، پيغام مي تواند متن (از طريق کلاس TextMessageEncodingBindingElement ،عناصر بايند را در فضاي نامSystem.ServiceModel.Channels قرار دارند)،باينري(از طريق کلاس BinaryMessageEncodingBindingElement)،يا بعضي رمزگذاريهاي ديگر باشد.

عنصر بايند مورد نياز ديگر عنصر بايند انتقال(transport binding element) که چگونه انتقال پيغام روي سيم مشخص مي کند .براي مثال پيغام مي تواند روي HTTP(از طريق HttpTransportBindingElement)،HTTPS(از طريق (HttpsTransportBindingElement)،TCP(ازطريق TcpTransportBindingElement) ، يا حتي يک گروه ديگر.يک بايند همچنين عناصر بايند ديگري براي افزودن ويژگي هاي بيشتر دارد.ما دوباره اشاره مي کنيم وقتي يک بايند استفاده مي کنيم.

قسمت آخر در رابطه با endpoint که قبلا بحث شده است. موردي که نياز به ياد آوردي داريد اگرچه که شما از طريق يک قرارداد((contractبه سرويس ميزبان ارتباط داريد.

اگر يک کلاس يک اينترفيس را پياده سازي کند،شما مي توانيد به شي نمونه از طريق اينترفيس دسترسي داشته باشيد.براي مثال در کد زير شما قادر به دسترسي شي Dude از طريق اينترفيس Ispeak مي باشيد:

```
interface ISpeak
{
    void Speak(String text);
}
class Dude : ISpeak
{
    public void Speak(String text)
    {
        //+ speak text
```

```
    }
}
public class Program
{
    public void Run()
    {
        ISpeak dude = new Dude();
        dude.Speak("Hello");
    }
}
```

شما به یک سرویس WCF شبیه این مي توانید دسترسي داشته باشید.شما بیشتر حتي در مقایسه مي توانید انجام دهید.گفتیم کلاس Dude اینترفیس IEat را همچنین پیاده سازي مي کند.سپس ما به یک شي Dude نمونه سازي شده از طریق اینترفیس IEatمي توانیم دسترسي داشته باشیم.این کد مورد تظر مي باشد:

```
interface ISpeak
{
    void Speak(String text);
}
interface IEat
{
    void Eat(String nameOfFood);
}
class Dude : ISpeak, IEat
{
    public void Speak(String text)
    {
        //+ speak text
    }
}
public class Program
{
    public void Run()
    {
        IEat dude = new Dude();
        dude.Eat("Pizza");
```

```
    }
}
```

وقتي يک سرويس WCF تنظيم مي کنيد شما يک endpoint براي قرارداد(contract)خواهيد اضافه کرد به طوري که شما دوست داريد که سرويستان در دسترس باشد.

اگر چه خارج از اين مقاله است،WCF به شما اجازه نسخه بندي قراردادي ها(contracts) را مي دهد.شايد شما يک پارامتر به/از قرار داد(contract) اضافه يا حذف کني و اگر شما دسترسي همه سرويس گيرنده ها را به سرويس نمي خواهيد قطع کنيد شما بايد قرارداد(contract) قديمي را براي سرويس خود به کار ببريد(اينترفيس قبلي کلاس سرويس را نگه مي دارد) و endpoint قديمي را توسط تنظيم يک endpoint به صورت موازي در حال اجرا نگه داريد . شما يک سرويس endpoint جديد خواهيد اضافه خواهيد کرد هر بار که شما نسخه (version) ،قرارداد (contract)،يا بايند(binding) را تغييرمي دهيد.