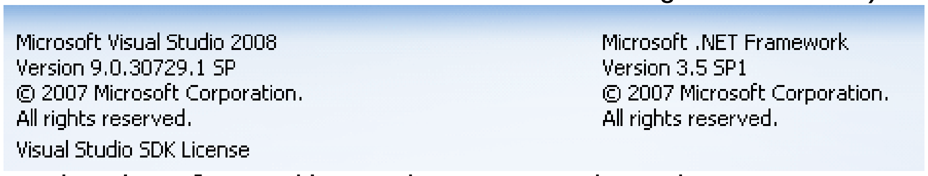
Introduction to WCF 3.5 and Rest

Presented at WKB, january 2010
By Kurt Evenepoel

What do you need

- Visual Studio 2008 SP1
- .NET 3.5 SP1
- Windows SDK (was included in my installation: check if you have C:\Program Files\Microsoft SDKs\Windows\v6.0\Bin\SvcConfigEditor.exe)



Microsoft Visual Studio 2008
Version 9.0.30729.1 SP
© 2007 Microsoft Corporation.
All rights reserved.
Visual Studio SDK License

Microsoft .NET Framework
Version 3.5 SP1
© 2007 Microsoft Corporation.
All rights reserved.

Both Visual Studio and .NET need to have SP1, not only .NET.

WCF Fundamentals

Since .NET 3.0 (Codename .NET FX) Microsoft has added lots of new libraries to the .NET framework. One of these is WCF. WCF is expanded with Restful features in .NET 3.5 (C# 3.0), and integrated with Workflow Foundation and other technologies. Many new .NET 3.5 technologies use WCF as their underlying basis too. In .NET 4.0, WCF gets more important, and in the future years to come WCF will replace all microsoft distributed development technologies.

Windows Communication Foundation (WCF) is all about *communicating*. In a stand-alone application like say XML Spy, or Notepad you would not generally use services.
In a *business context*, many dollars go into creating systems that support the business. It’s generally accepted that in such scenario’s, you do not want to tie a function to a specific application. All applications, should they need that specific function, should be able to re-use it. You can do this by *distributing a dll* with the application, but then you also need update scripts when a new version is released, and manage the users access rights within Windows and the database, and people could try to run your application with old DLLs. There are several disadvantages to this scenario.

Or you can expose the functionality as a *service*, where one service is a building block of functionality offered by a central server or cluster of servers. This building block or service can use other services, or dlls. What is important is that the functionality *is centrally managed and exposed to possible clients*.

Some of the advantages are :
- by separating groups of functions (like objects) into services, each group of functions can be hosted on any server on the network, including the same one, or on multiple servers under load-balancing
- easy to loose-couple services because generally proxy classes are used, so you can get intellisense, but the implementation itself runs on the server

So what is WCF all about then? It's about writing *distributed systems and services*, and *exposing the functionality without being dependant on the method of communication*, so that the method of communication can be freely chosen and changed.

A *WCF Service* is a program that exposes a collection of *Endpoints*. Each Endpoint is a portal for *communicating with the world*. Communication is handled by WCF, the logic how to respond to the messages by the developer. This enables the developer to develop logic that is not dependant on the way it is communicated.
Eg: TCP, Peer-to-Peer, SOAP, Rest, wether they are authenticated or secured with certificates or not.

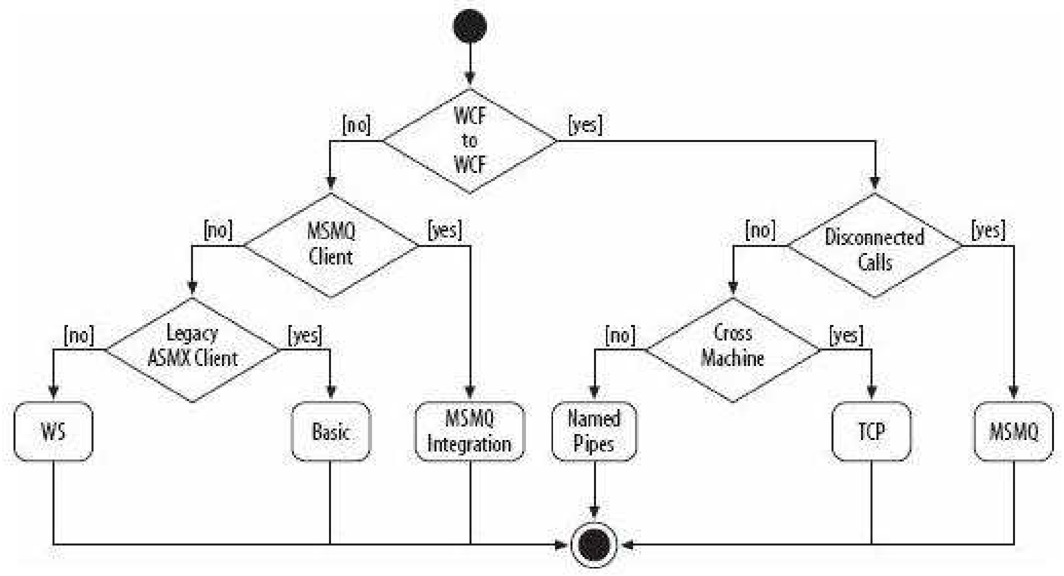
A *Client* is a program that exchanges messages with one or more Endpoints. A Client may also expose an Endpoint to receive Messages from a Service in a duplex message exchange pattern.

A *message exchange pattern* is a pattern how the client and the service exchange messages. There are a number of message patterns. Rest uses normal HTTP, so that's a *request-reply* pattern: the browser or client requests the page, and the HTTP server returns a reply.
WCF supports other patterns like Datagram (fire-and-forget) and Duplex (two-way). *Sessions* are also supported.

WCF is *extensible*: if a protocol is not supported yet, you can inject your own logic almost everywhere in the framework to ensure minimal development costs for changes in protocol, transport method and the likes. This is called a Behaviour, more on that later.

Traditional WCF in .NET 3.0 did not know Restful features, these were added in .NET 3.5. These features include ajax support, *Atom and RSS feeds* and more.

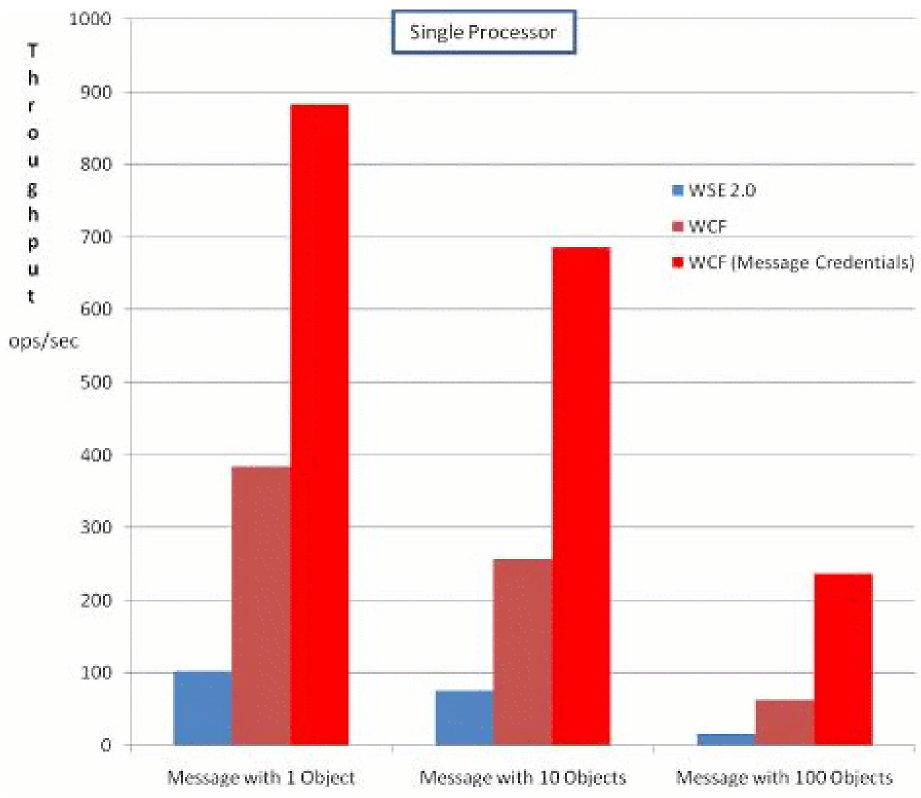
This decision tree helps decide what (binding) technology to use for your endpoints in a non-restful scenario (a Restful scenario would always use the same type of setup for endpoints, as you will see later).



(source: http://weblogs.asp.net/spano/archive/2007/10/02/choosing-the-right-wcf-binding.aspx)

Don’t be overwhelmed by this diagram. You will find it useful someday. When choosing a SOAP binding, be careful what you choose, your choice has serious performance implications. Always pick the fastest technology that supports your goals. Prefer two fast, specific endpoints to one do-everything endpoint over WS- so make that TCP endpoint for .NET clients by all means.

That said, WCF is one-on-one faster than any technology it replaces, including remoting, ASMX and most of all, WSE 2.0, where WCF is many times faster and results in the same endresult.



For the full performance comparison :
`http://msdn.microsoft.com/en-us/library/bb310550.aspx`

REST Fundamentals

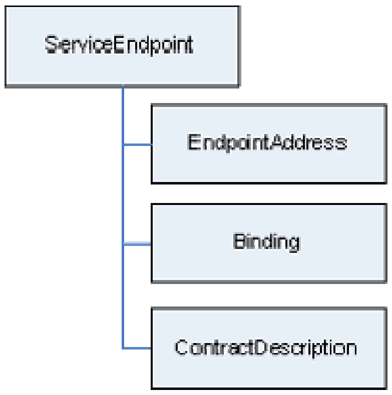
REST is a *resource-oriented* architecture. The focus lies on the data, not on actions. Content as we know it in WKB is a prime example of a resource.
REST is opposed to SOAP. *SOAP* is a complex protocol for interoperating, offering many layers of *security, versioning and more*, and therefore is bloating the messages by a large amount. Usually SOAP runs over HTTP, but in WCF it can run over TCP or other connections too. SOAP bypasses a lot of HTTP semantics and implements them again as its own. *REST* on the other hand is *lightweight, simpler protocol* that follows the HTTP way of working, and all of it semantics: a header value of GET to retrieve something, PUT or POST to save something, and DELETE to delete a resource. SOAP is more versatile, but very complex and slow. REST is a lot faster and offers a lot less, and is limited to HTTP.

Endpoints ‘R Us

A Service Endpoint has

- an *Address*,
- a *Binding*,
- and a *Contract*.

= The ABC of Communication Foundation



The Endpoint's *Address* is a network addres or URI =***WHERE***.

The Endpoint's *Contract* specifies *what the Endpoint communicates* and is essentially a collection of messages organized in operations that have basic Message Exchange Patterns (MEPs) such as one-way, duplex, and request/reply. Generally we use an interface with attributes to define a contract =***WHAT***.

There are several types of contracts:
Service contracts
Describes which operations the client can perform on the service.
- ServiceContractAttribute
  applied to the interface.
- OperationContractAttribute
  applied to the interface methods you want to expose

Data contracts
Defines which custom data types are passed to and from the service (high-level).
- DataContract
  applied to the class that holds the data
- DataMember
  applied the the class' properties you want to exchange

- OR –

Message contracts
Interact directly with SOAP messages (low-level). Useful when there is an existing message format we have to comply with.

Fault contracts

Defines how the service handles and propagates errors to its clients.

The Endpoint's *Binding* specifies *how the Endpoint communicates with the world* including things like transport protocol (e.g., TCP, HTTP), encoding (e.g., text, binary), and security requirements (e.g., SSL, SOAP message security) =***HOW***.

A *behavior* is a class that implements a special interface for *"plugging into" the execution process*. This is your primary WCF *extensibility and customization point* if something is not supported out of the box. =***TWEAK WHERE/WHAT/HOW***
There are a few types of behaviours:

Service Behaviour
- Applies to the entire service runtime in an application
- Implements IServiceBehavior.

Contract Behaviour
- Applies to the entire contract
- Implements IContractBehavior.

Operation Behaviour
- Applies to the service operation (think: method call)
- Implements IOperationBehavior.

Endpoint Behaviour
- Applies to the endpoints
- Implements IEndpointBehavior

Creating behaviours is something you can try yourself after this session, or can request another session on. There are 3 ways:

Writing code
- Implement the behaviour interface
- Add your behaviour to the Behaviors collection of the related Start the service host.

Decorating with an attribute
(not for Endpoint behaviours)
- Implement the behaviour interface
- Inherit from Attribute
- Decorate the wanted class with your class.

Configuration

(not for Contract or Operation behaviours)
- Derive from BehaviorExtensionElement.
(see http://msdn.microsoft.com/en-gb/library/ms730137.aspx)

It’s important to note that behaviours are a *server-side only* thing: the WCF client classes make no use of them.

In many projects behaviours won’t be needed, and all you’ll need is the ABC. It’s common however to need to tweak things a little. This is where we add another B – ABBC. Almost ABBA but not quite.

All of this may seem a bit daunting at first. Don’t worry about the specifics too much yet. You need will need to know more about them when you use SOAP. There is much to know because WCF is so powerful. Simple scenario’s don’t require you to know everything in-depth as you will see, and if you know some basics you will get your service running in no time. For Rest scenario’s, knowing all the details is not needed to get you started, so we won’t elaborate on them in this session.

Now lets get started in practise.

Exploring Your Hosting Options

On the Microsoft .NET platform, you have several types of managed Windows applications that you can create with Microsoft Visual Studio.NET:

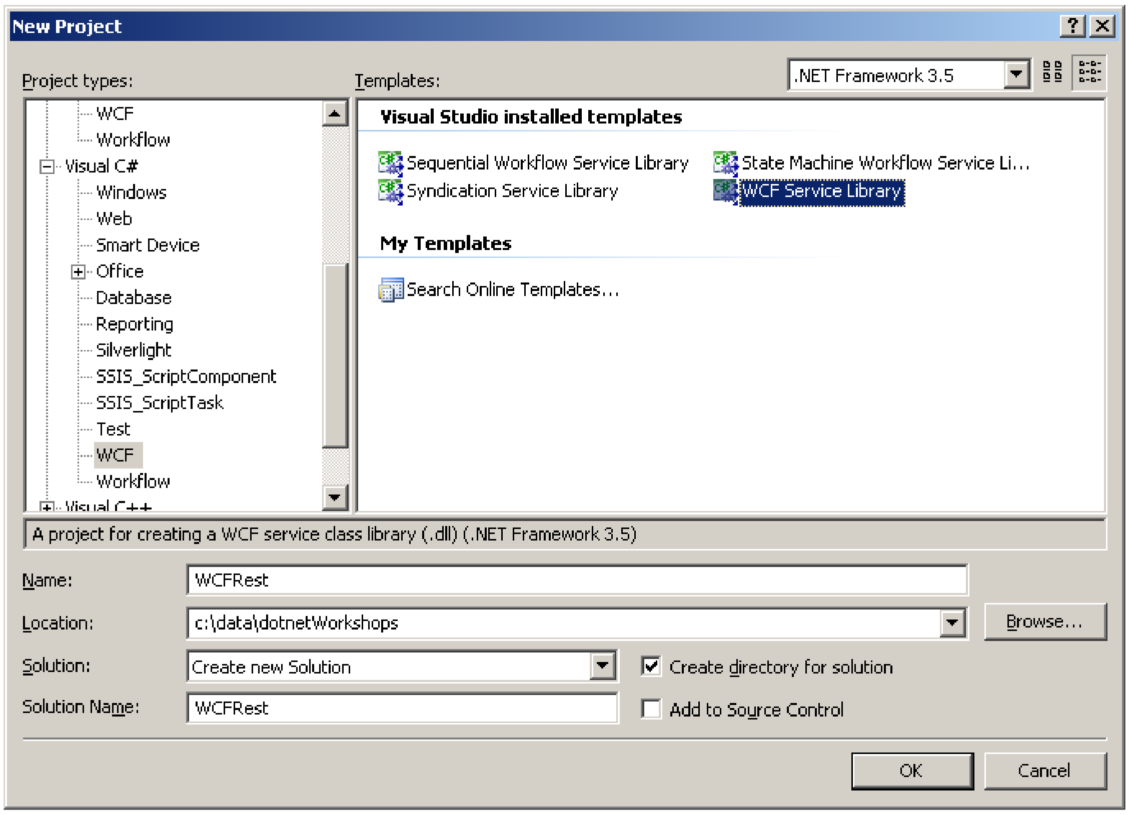
- WinForms applications
- Console applications
- Windows services
- Web applications (ASP.NET) hosted on Internet Information Services (IIS)
- WCF services inside IIS 7.0 and WAS on Windows Vista or Windows Server 2008

WCF doesn't block you from running your service in any other type of application as long as it provides you with a .NET application domain. It all comes down to the requirements you have for your host. To summarize the options, think about the following three generic categories of hosts for your WCF services:

- *Self-hosting* in any managed .NET application
- Hosting in a *Windows service*
- Hosting in different versions of *IIS/ASP.NET*

We will be self-hosting the service eventually in a console window, creating your own webserver (sortof)!

Creating a new Service Library


New Project

Project types:

- WCF
- Workflow
- Visual C#
  - Windows
  - Web
  - Smart Device
  - Office
  - Database
  - Reporting
  - Silverlight
  - SSIS_ScriptComponent
  - SSIS_ScriptTask
  - Test
  - WCF
  - Workflow
- Visual C++

Templates: .NET Framework 3.5

Visual Studio installed templates

- Sequential Workflow Service Library
- State Machine Workflow Service Li...
- Syndication Service Library
- WCF Service Library

My Templates

- Search Online Templates...

A project for creating a WCF service class library (.dll) (.NET Framework 3.5)

Name: WCFRest

Location: c:\data\dotnetWorkshops

Solution: Create new Solution

[✓] Create directory for solution

Solution Name: WCFRest

[ ] Add to Source Control

- Create a new *WCF Service Library*.
- Delete both Service.cs and IService.cs

Why did we start a Service Library only to toss everything out? Some project types add entries to menus. Like in this case, it will add the ability to visually edit the app.config.

Defining and Implementing a Contract

The easiest way to define a contract is creating an interface or a class and annotating it with *ServiceContractAttribute*.
For example:

```
using System.ServiceModel;

//a WCF contract defined using an interface
[ServiceContract]
public interface IMath
{
    [OperationContract]
    int Add(int x, int y);
}
```

Then create a class that implements **IMath**. That class becomes the WCF Service class.For example:

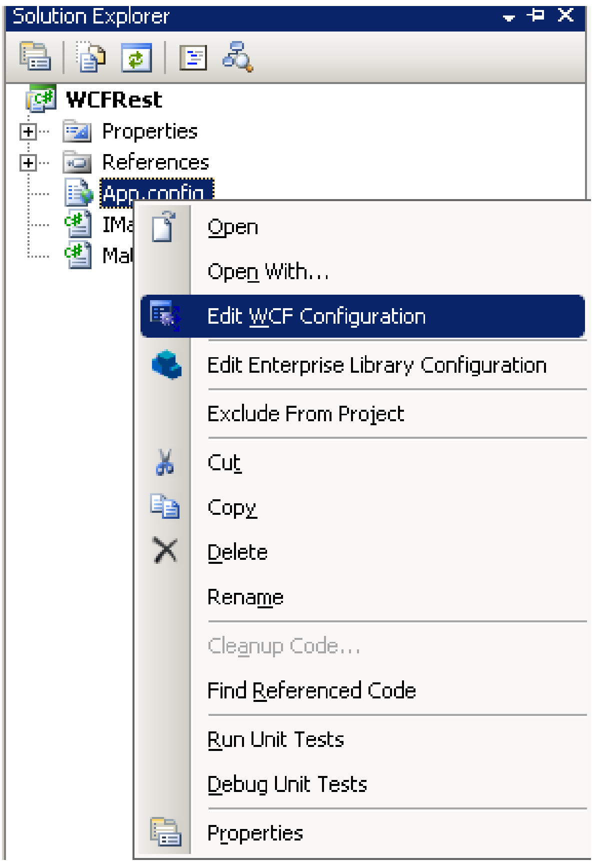
```
//the service class implements the interface
public class MathService : IMath
{
    public int Add(int x, int y)
    { return x + y; }
}
```

Defining Endpoints

Endpoints can be defined in *code* or in *config*. We'll be using config.

```
<configuration>
        <system.serviceModel>
                <behaviors>
                        <serviceBehaviors>
                                <behavior name="mexSvcBehaviour">
                                        <serviceMetadata
httpGetEnabled="false" httpsGetEnabled="false" />
                                </behavior>
                        </serviceBehaviors>
                </behaviors>
                <bindings />
                <services>
                        <service behaviorConfiguration="mexSvcBehaviour"
name="WCFRest.MathService">
                                <endpoint
address="http://localhost:41000/MathService/Ep1"
binding="wsHttpBinding"
                                 contract="WCFRest.IMath" />
                                <endpoint
address="http://localhost:41000/MathService/mex"
binding="mexHttpBinding"
                                 bindingConfiguration=""
contract="IMetadataExchange" />
                        </service>
                </services>
        </system.serviceModel>
</configuration>
```

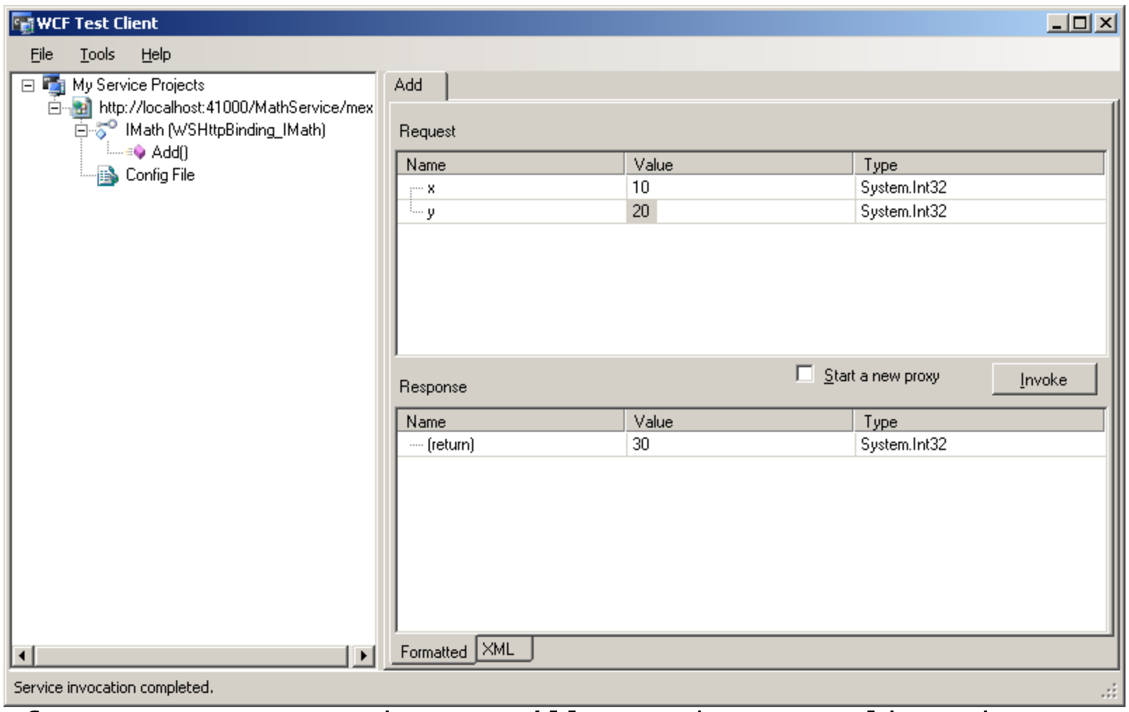
Generally config is preferred. In some project types you will find the *Edit WCF Configuration* menu option by right-clicking on the App.config.



Using this editor you can easily set the options needed for your service. If not present run C:\Program Files\Microsoft SDKs\Windows\v6.0A\bin\SvcConfigEditor.exe on *app.config* for the same editor.

For now, just copy-paste the App.config from above into yours.

The Test Client



Name	Value	Type
x	10	System.Int32
y	20	System.Int32

Name	Value	Type
(return)	30	System.Int32

Service invocation completed.

If you run your service, you'll see the test client is *automatically configured to run* when you run the dll.

Note: the test client is not useful for Rest-tests. If you started a SyndicationLibrary project, IE will be used instead of the WCF Test Client.

Consuming the service

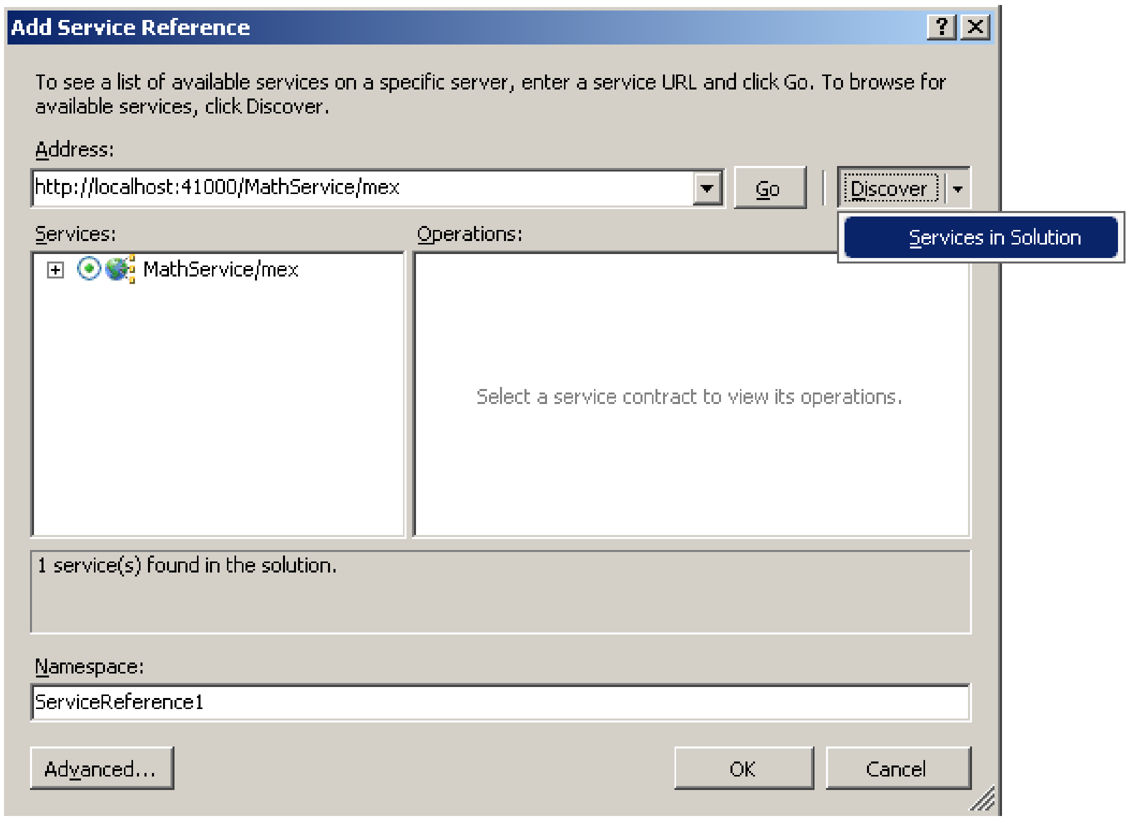
There are 2 ways to generate classes

SvcUtil.exe

C:\Program Files\Microsoft SDKs\Windows\v6.0A\bin\svcutil.exe
Useful tool for generating classes based on WDSL.

Add Service Reference

Like *Add Reference* except the menu option just below


Add Service Reference

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://localhost:41000/MathService/mex

Services: MathService/mex

Operations: Select a service contract to view its operations.

1 service(s) found in the solution.

Namespace: ServiceReference1

- Add a new windows console project called *WCFClient* to the solution.
- Add a *service reference* to the service in the solution as shown above, use a namespace you'll remember.
- Add a normal reference to System.ServiceModel
- In static void main, do the following:

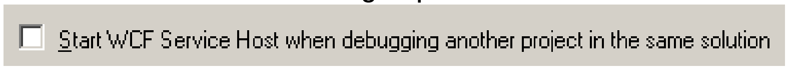
```
    internal class Program
    {
        /// <summary>
        /// Starts the application
        /// </summary>
        /// <param name="args"></param>
        internal static void Main(string[] args)
        {
            MathClient client = new MathClient();
            Console.WriteLine(client.Add(10, 20));
            Console.ReadKey();
        }
    }
```

Tip: Should your application stop working at a later point because the client cannot figure out what binding to connect to, change the following line:

```
MathClient client = new MathClient("WSHttpBinding_IMath");
```

Hosting the service from console

- Add a new *windows console project* to your solution called *WCFHost*.
- Add a reference in the project to your service project.
- Add a reference to *System.ServiceModel*.
- Edit the *properties of the WCF library* project you referenced, and go to the tab *WCF Options*. Then untick the following option:



☐ Start WCF Service Host when debugging another project in the same solution

- Copy the App.config from the *WCF library* to your *WCFHost* project.
- In *static void main* method of your *WCFHost* project (type svm tab tab if you don't have one) do the following:

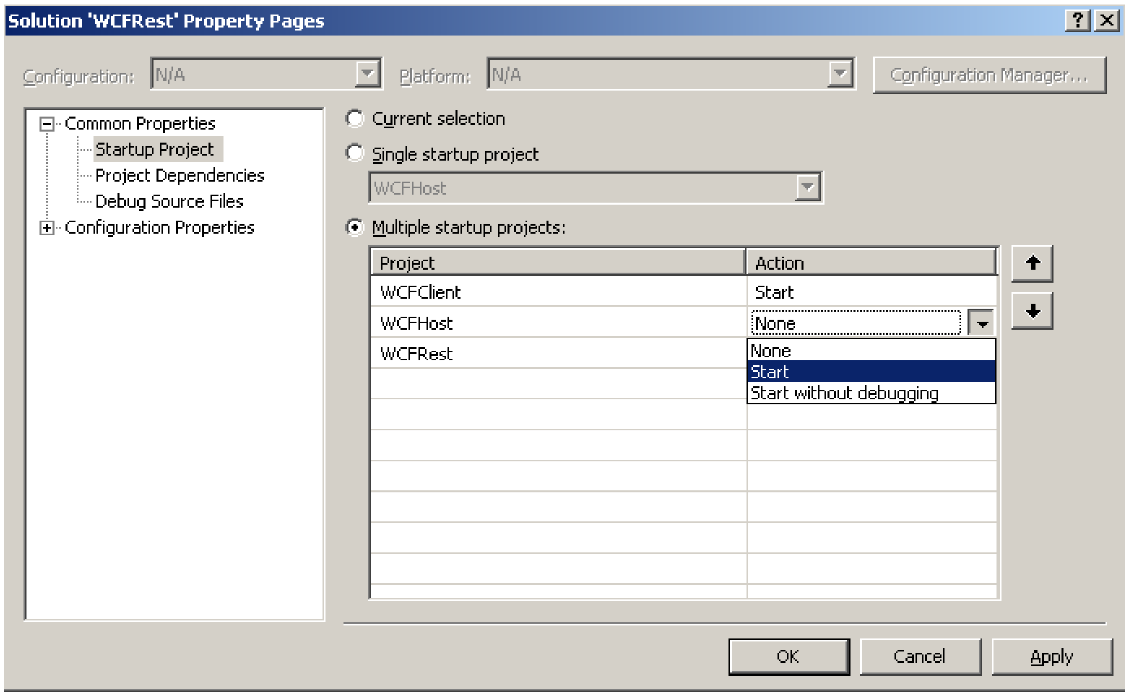
```
    internal static class Program
    {
        /// <summary>
        /// The main entry point for the application.
        /// </summary>
        internal static void Main()
        {

            using (ServiceHost serviceHost = new
ServiceHost(typeof(WCFRest.MathService)))
            {
                try
                {
                    // Open the ServiceHost to start listening for
messages.
                    serviceHost.Open();

                    // The service can now be accessed.
                    Console.WriteLine("The service is ready.");
                    Console.WriteLine("Press <ENTER> to terminate
service.");
                    Console.ReadLine();

                    // Close the ServiceHost.
                    serviceHost.Close();
                }
                catch (TimeoutException timeProblem)
                {
                    Console.WriteLine(timeProblem.Message);
                    Console.ReadLine();
                }
                catch (CommunicationException commProblem)
                {
                    Console.WriteLine(commProblem.Message);
                    Console.ReadLine();
                }
            }
        }
    }
```

Now, make sure that both projects run when you start:
Right-click the *solution* and select Properties
In the *Startup Project* list, change the action of the
WCFClient and the WCFHost to Start.


Solution 'WCFRest' Property Pages

Configuration: N/A   Platform: N/A   Configuration Manager...

- Common Properties
  - Startup Project
  - Project Dependencies
  - Debug Source Files
- Configuration Properties

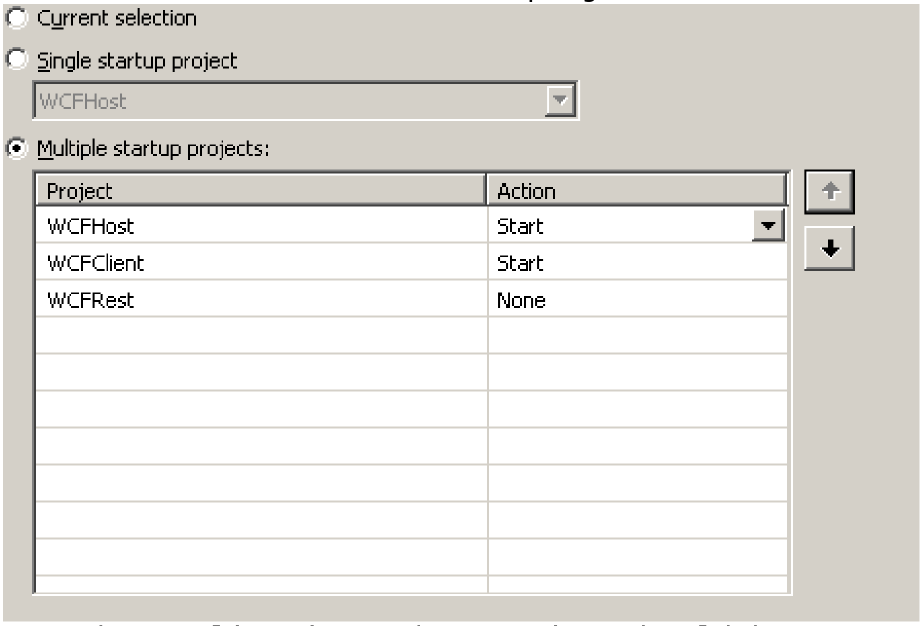
Current selection
Single startup project: WCFHost
[✓ Multiple startup projects:]

Project	Action
WCFClient	Start
WCFHost	None
WCFRest	

None
Start
Start without debugging

OK   Cancel   Apply

Then press the arrow-up button to make sure the WCFHost starts before the WCFClient project.



Project	Action
WCFHost	Start
WCFClient	Start
WCFRest	None

Run the application. The service should be started, and you should have ‘30’ as a reply from your service.

What’s happening?

- WCFHost is hosting the service in a windows application, any application can now host services.
- WCFClient is connecting to that service via a generated proxy class so you can compile and get intellisense.
- WCFClient calls methods via the proxy, and the proxy transparently calls the service.

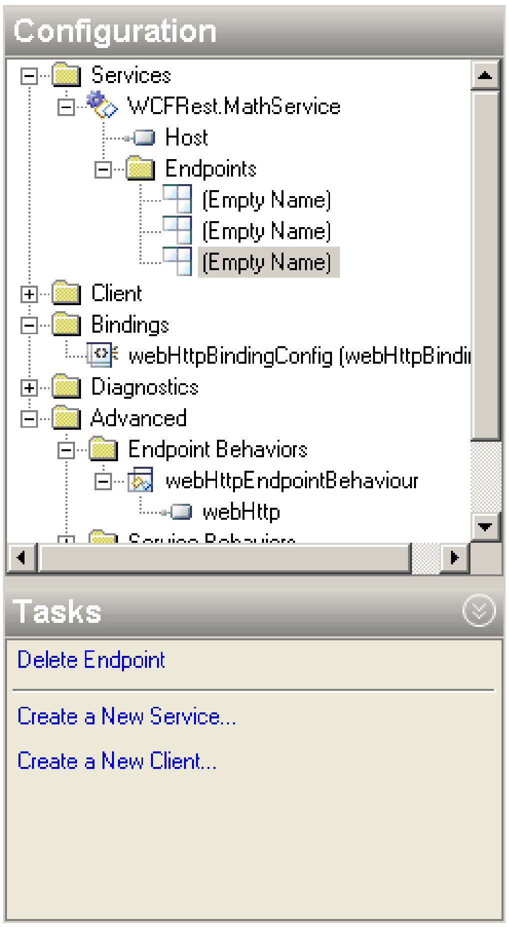
Transforming the service into a REST web service

Note: transforming the math service, with is an *action-oriented* service (does something), to a Rest-service, which is *resource-oriented* (CRUD on data), may not be a good practise. What is shown here can hardly be seen as good design, but here goes.

Back to your service project, and to the *IMath* interface.

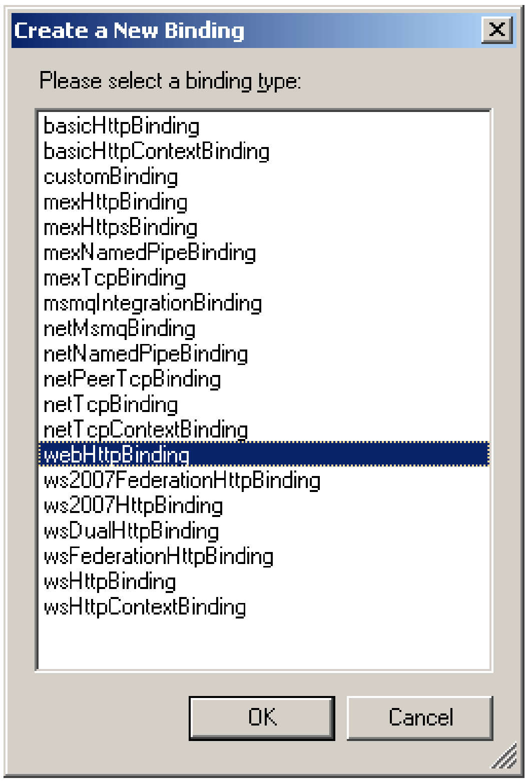
- To see your options, you’ll have to first add a reference to *System.ServiceModel.Web*.
- Then change the interface as follows:

```
        [WebInvoke(Method = "GET",
            ResponseFormat = WebMessageFormat.Json,
            BodyStyle = WebMessageBodyStyle.Bare,
            UriTemplate = "?x={x}&y={y}")]
        int Add(int x, int y);
```

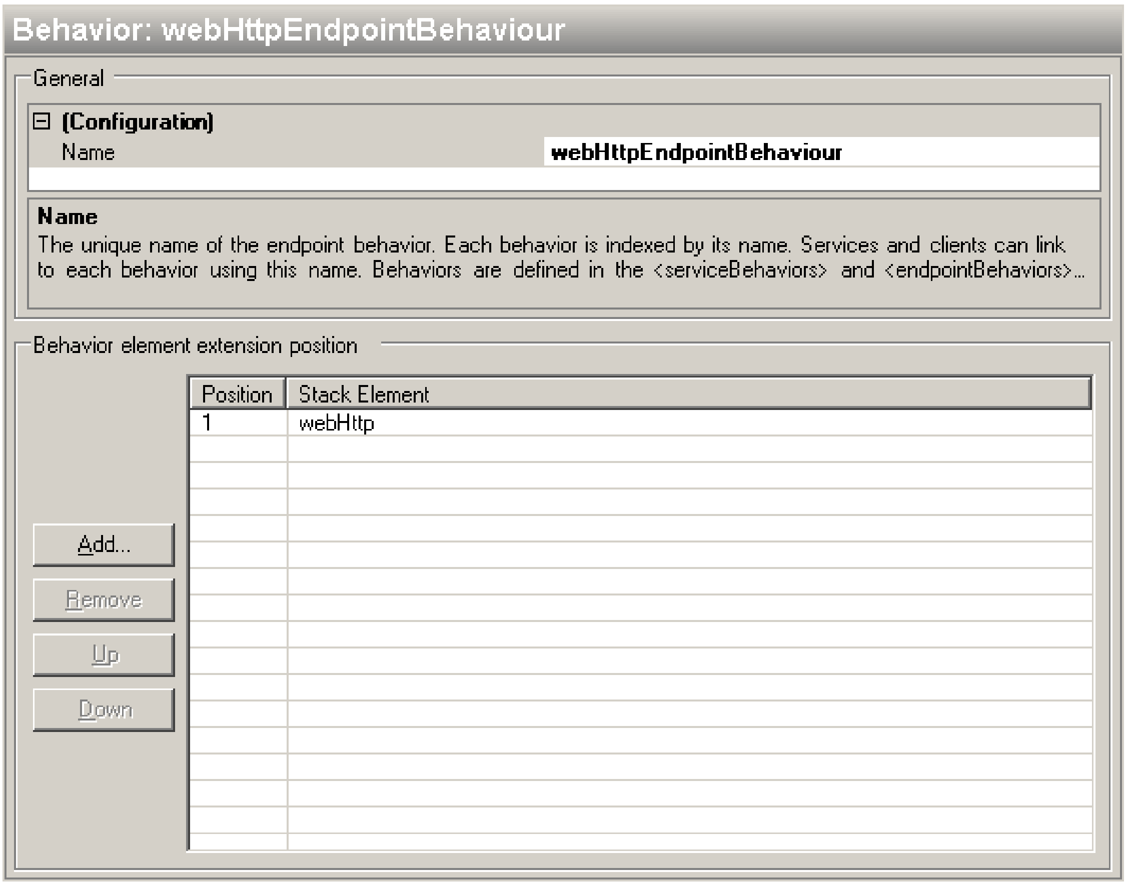


Note that you can use *numbers* and *strings* in your interface as parameters for the *uri template*, but only if they're part of the querystring. Inside the url itself, only strings are supported. You can return other classes or accept classes from the HTTP body instead (see later).

- Add configuration for a *binding* as shown and name it *webHttpBindingConfig.*



- Under Advanced, define a new service *endpoint behaviour*, and add the *webHttp* element. Call it *webHttpEndpointBehaviour.*



Behavior: webHttpEndpointBehaviour

General

(Configuration)

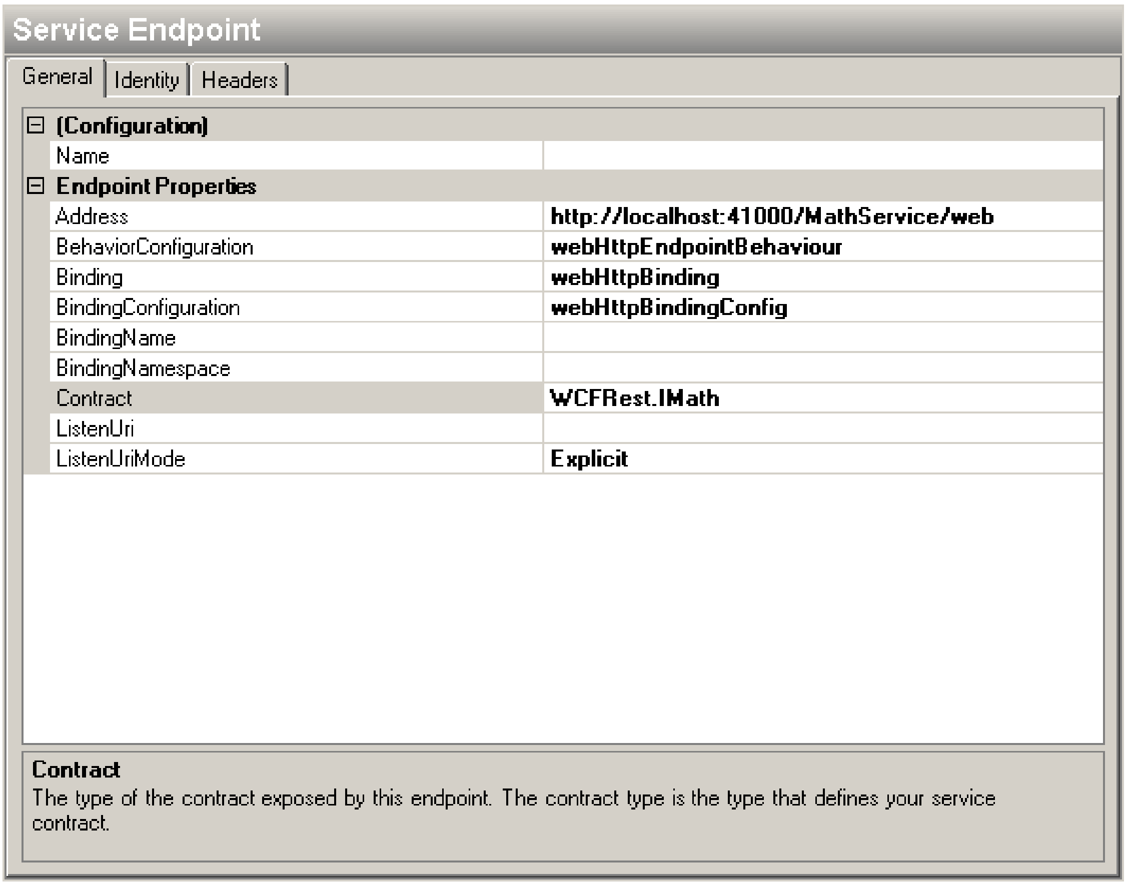
Name webHttpEndpointBehaviour

Name
The unique name of the endpoint behavior. Each behavior is indexed by its name. Services and clients can link to each behavior using this name. Behaviors are defined in the <serviceBehaviors> and <endpointBehaviors>...

Behavior element extension position

Position	Stack Element
1	webHttp

- Then finally, add a new service endpoint and fill in the fields as shown below.



Service Endpoint

(Configuration)	
Name	
Endpoint Properties	
Address	**http://localhost:41000/MathService/web**
BehaviorConfiguration	**webHttpEndpointBehaviour**
Binding	**webHttpBinding**
BindingConfiguration	**webHttpBindingConfig**
BindingName	
BindingNamespace	
Contract	**WCFRest.IMath**
ListenUri	
ListenUriMode	**Explicit**

Contract
The type of the contract exposed by this endpoint. The contract type is the type that defines your service contract.

- Save

Run the application as before, the two applications should still work. While they are running, direct your favourite browser to:

```
http://localhost:41000/MathService/web/?x=10&y=20
```

Notice how this is the url for the endpoint combined with the url template for the method call.
This should give you a json reply. If you download it, it should contain ‘30’.

Congratulations on your first WCF Rest service. It can add two numbers and tell you how much the result is.

Returning files or large amounts of data

You can also return *streams*. Streams are recommended for large amounts of data, or files you have on disk and want to return. For example, your webservice might return links to images, and you need to return the images too.

Add a method to the interface like this:

```
        [OperationContract]
        [WebInvoke(Method = "GET",
            ResponseFormat = WebMessageFormat.Json,
            BodyStyle = WebMessageBodyStyle.Bare,
            UriTemplate = "notepad")]
        Stream GetStream();
```

Note: you can also use the *WebGet* attribute wich is specifically meant for GET requests. Try it out with a WebGet attribute instead.

Then implement the new method in your concrete MathService class:

```
        public Stream GetStream()
        {
            return File.OpenRead(@"c:\windows\notepad.exe");
        }
```

Run the application again and download your own notepad by directing your browser to the following url:

```
http://localhost:41000/MathService/web/notepad
```

Note that there are a few service settings that *limit the size of the messages that can be sent or received*. On top of that, if you’re hosting inside of IIS, the IIS server’s limits need to be adjusted too in web.config.

More specifically, common settings that you may need to adjust to receive and send large files are the *message sizes for the endpoint, the service AND the client* itself (added through built-in behaviours), and possibly for ASP.NET, the *HTTP request size*.

```
<httpRuntime maxRequestLength="xxx"/>
```

Returning a feed

With WCF there are two standard feed formats supported:

- Atom 1.0
- RSS 2.0

Generating a feed is done with the *SyndicationFeed* class.

- Create the SyndicationFeed instance
- Assign properties and content
- Return and format it using a formatter:
    - Atom10FeedFormatter
    - Rss20FeedFormatter

So let’s get started.

Add the following code to your *IMath* interface:

```
        [WebGet(UriTemplate = "feed?rss={rss}")]
        [OperationContract]
        SyndicationFeedFormatter GetFeed(bool rss);
```

Then implement the new method as following:

```
        public SyndicationFeedFormatter GetFeed(bool rss)
        {
            List<SyndicationItem> items = new List<SyndicationItem>();

            // make link to notepad
            SyndicationItem item = new SyndicationItem();
            item.Title = new TextSyndicationContent
                                  ("Notepad on server.");
            item.Links.Add(new SyndicationLink(new
Uri("http://localhost:41000/MathService/web/notepad")));
            item.PublishDate = DateTime.Now;
            items.Add(item);

            SyndicationFeed feed =
                new SyndicationFeed
                    ("Notepad Feed",
                    "Feed with links to notepad.exe",
                    new Uri("http://www.google.be"), // alternate link
is google
                    items);
            feed.LastUpdatedTime = DateTime.Now;
            if (rss)
            {
                return new Rss20FeedFormatter(feed);
            }
            return new Atom10FeedFormatter(feed);
        }
```

Run the application. The application works. If you direct your browser to get the feed we just made:

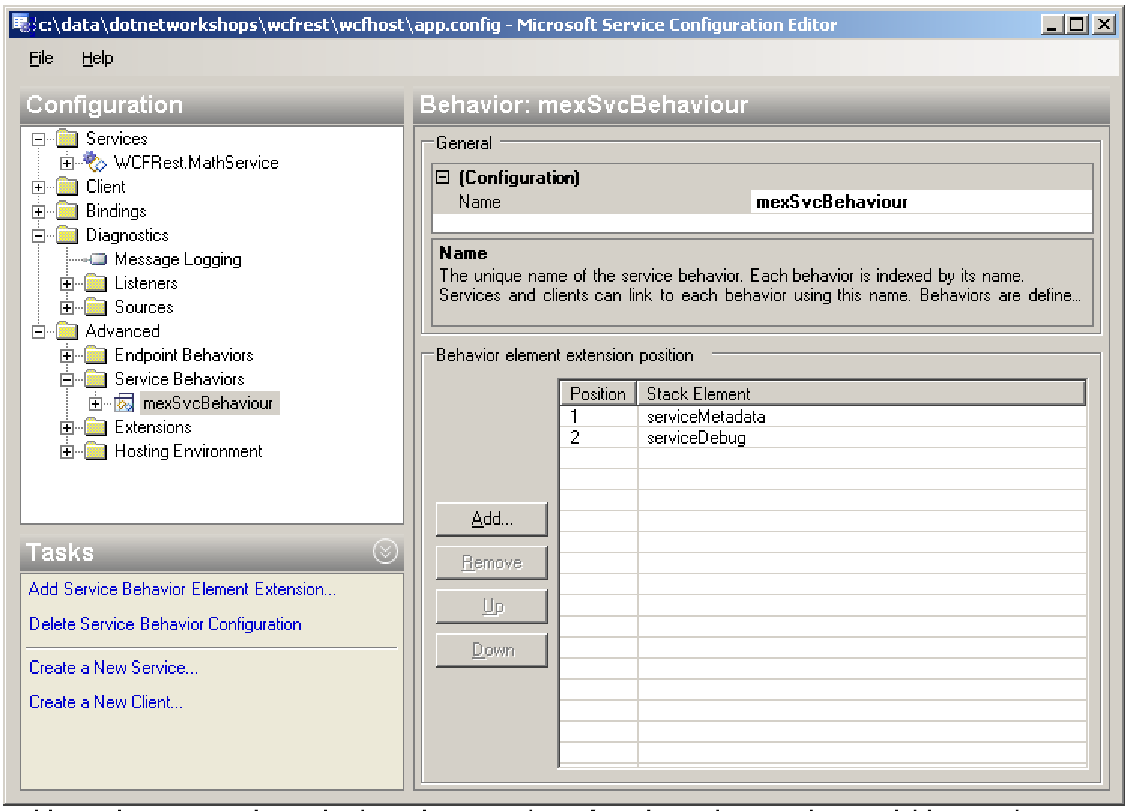
```
http://localhost:41000/MathService/web/feed
```

Doh! An error occurred, and if you'd step through the method call while debugging, no exception happens at all. The webpage doesn't tell you anything useful. What gives?

Debugging WCF

To find out what's wrong, lets enable a few things we *do not want to be active in a production scenario*. We'll only edit the app.config of the WCFHost project, not the one that produces our dll.

First, enable exception information. This is mostly for SOAP.



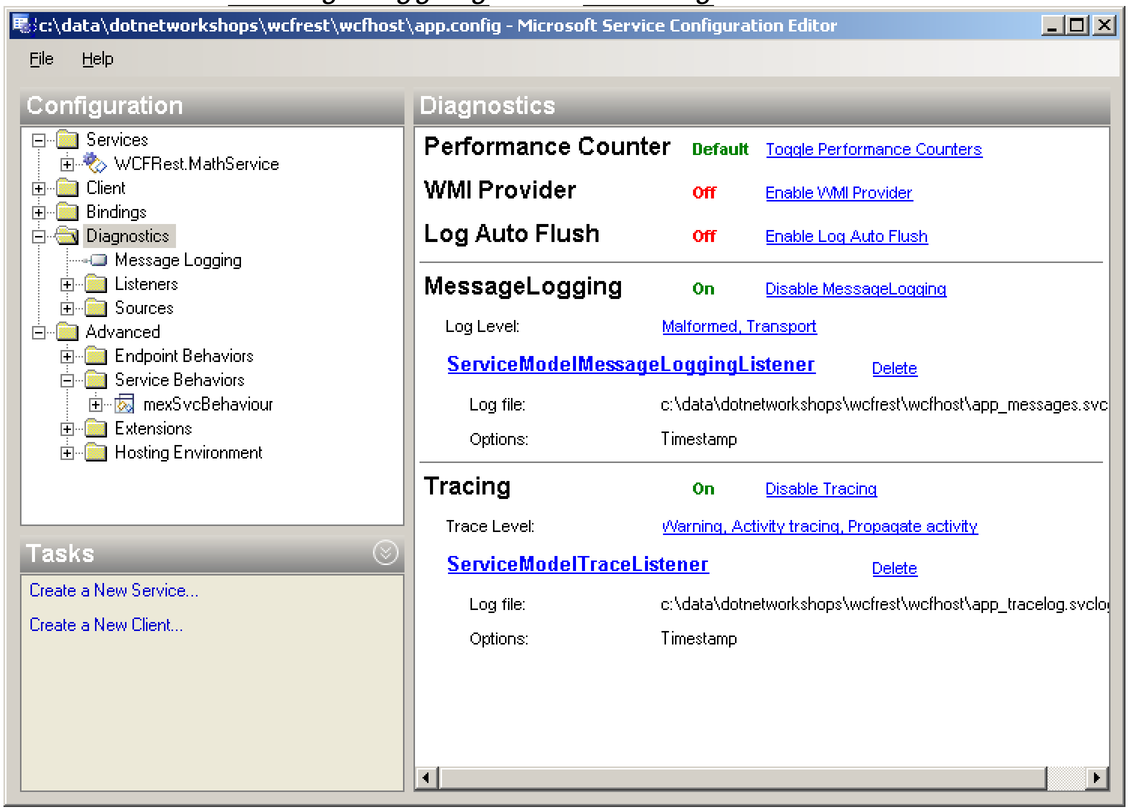
(Configuration)
Name mexSvcBehaviour

Name
The unique name of the service behavior. Each behavior is indexed by its name. Services and clients can link to each behavior using this name. Behaviors are define...

Position	Stack Element
1	serviceMetadata
2	serviceDebug

Edit the service behaviour that's in there by adding the *serviceDebug* element (the other one is to enable *metadata* for the test client and Add Service Reference to generate classes from).

Then enable *messagelogging* and *tracing*.

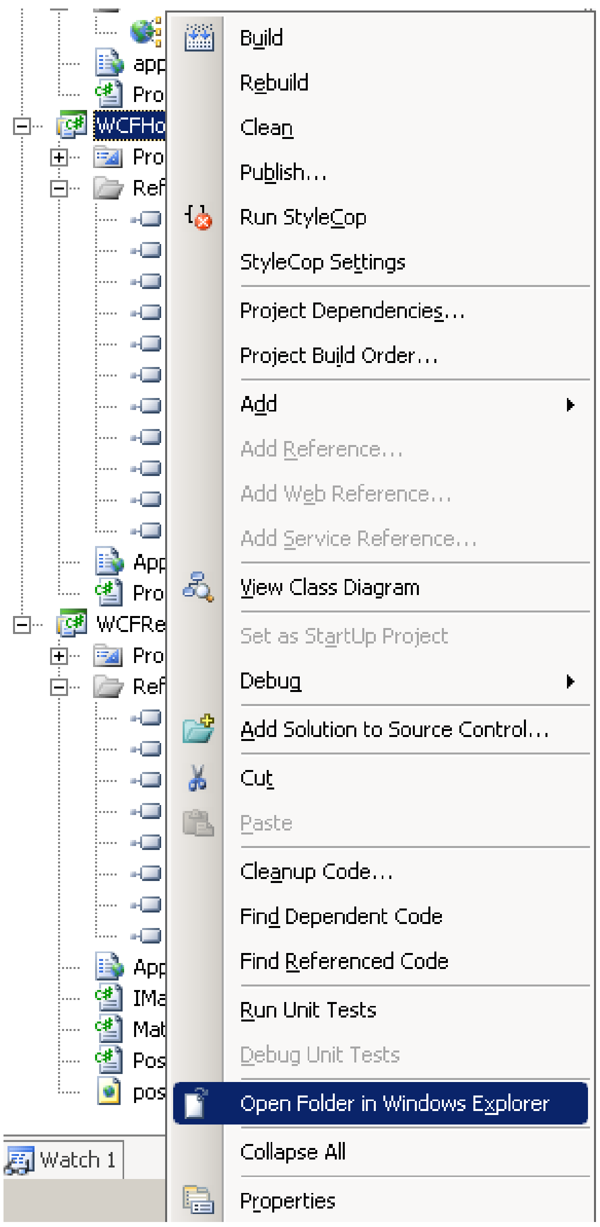


Performance Counter	Default	Toggle Performance Counters
WMI Provider	Off	Enable WMI Provider
Log Auto Flush	Off	Enable Log Auto Flush
MessageLogging	On	Disable MessageLogging
Log Level:	Malformed, Transport	
ServiceModelMessageLoggingListener		Delete
Log file:	c:\data\dotnetworkshops\wcfrest\wcfhost\app_messages.svc	
Options:	Timestamp	
Tracing	On	Disable Tracing
Trace Level:	Warning, Activity tracing, Propagate activity	
ServiceModelTraceListener		Delete
Log file:	c:\data\dotnetworkshops\wcfrest\wcfhost\app_tracelog.svclo	
Options:	Timestamp	

Finally, *Save* your app.config and run the application again.
Use your browser to visit the feed page again:

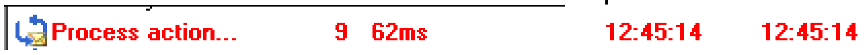
```
http://localhost:41000/MathService/web/feed
```

Then open the folder of the WCFHost project.



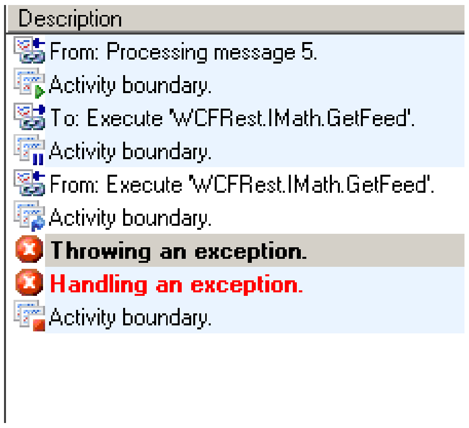
Notice the new files app_tracelog.svclog and app_messages.svclog in the folder.

Open *app_tracelog* by doubleclicking on it.
Microsoft Service Trace Viewer opens. Notice the *red lines*:



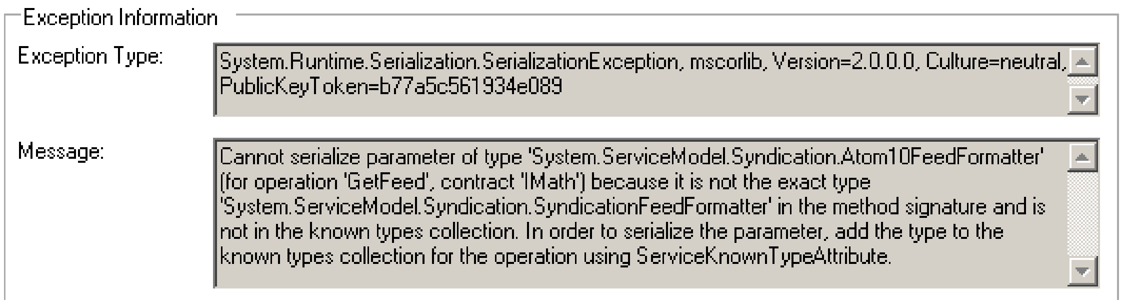
```
Process action...    9  62ms     12:45:14    12:45:14
```

These indicate errors. Let's see what went wrong.



```
Description
From: Processing message 5.
Activity boundary.
To: Execute 'WCFRest.IMath.GetFeed'.
Activity boundary.
From: Execute 'WCFRest.IMath.GetFeed'.
Activity boundary.
Throwing an exception.
Handling an exception.
Activity boundary.
```

Select the part where the service throws an exception after GetFeed. The exception is displayed right there:



Exception Information

Exception Type: System.Runtime.Serialization.SerializationException, mscorlib, Version=2.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089

Message: Cannot serialize parameter of type 'System.ServiceModel.Syndication.Atom10FeedFormatter' (for operation 'GetFeed', contract 'IMath') because it is not the exact type 'System.ServiceModel.Syndication.SyndicationFeedFormatter' in the method signature and is not in the known types collection. In order to serialize the parameter, add the type to the known types collection for the operation using ServiceKnownTypeAttribute.

Apparently we forgot to add the *[ServiceKnownType]* attribute, because we said we were going to return a *SyndicationFeedFormatter*, wich is the base class of Atom10FeedFormatter and Rss20FeedFormatter. We returned an *Atom10FeedFormatter*, but the service expected a SyndicationFeedFormatter. The ServiceKnownType attribute allows us to specify derived classes off base classes or interfaces used in our service.

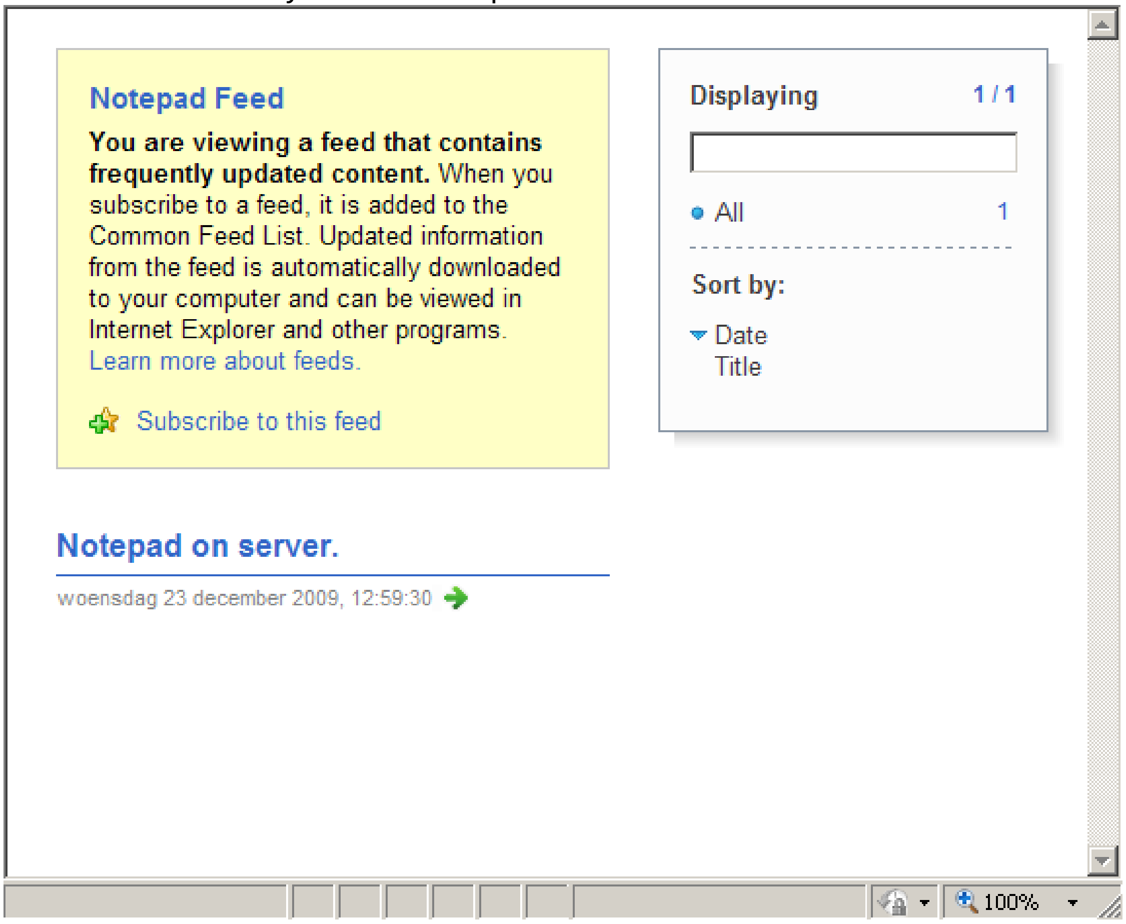
Let's fix that right now.

Looking up the ServiceKnownTypeAttribute class, the example indicates that you place one on top of an interface. Your IMath interface.

```
http://msdn.microsoft.com/en-us/library/system.servicemodel.serviceknowntypeattribute.aspx
```

Add the attributes to the interface like this:
```
    [ServiceContract]
    [ServiceKnownType(typeof(Atom10FeedFormatter))]
    [ServiceKnownType(typeof(Rss20FeedFormatter))]
    public interface IMath
```

Close your browser window if it's IE and start a new one. Revisit the url to get your feed:
```
http://localhost:41000/MathService/web/feed
```
You should see your feed operational.


Notepad Feed

You are viewing a feed that contains frequently updated content. When you subscribe to a feed, it is added to the Common Feed List. Updated information from the feed is automatically downloaded to your computer and can be viewed in Internet Explorer and other programs. Learn more about feeds.

Subscribe to this feed

Displaying 1 / 1

All 1

Sort by:

Date

Title

Notepad on server.

woensdag 23 december 2009, 12:59:30

Don't forget to turn message logging and tracing off again, and delete the service log files or you'll spend minutes parsing the files when you actually need them!

Accepting data

WCF can accept input from javascript/xml (in the body) or from the uri. The system looks for all the *method parameters* that are not used in the uri template in the body. Members from ajax (out of the box) or webforms (with

WCF Rest starterkit on codeplex: request a session if interested) can be used for communication. For ajax, XML or JSON can be used to fill in members of classes in the method call.

Ajax
Asynchrnonous Javascript and XML, used to update webpages without doing a full page postback and refreshing everything, saving processing time and bandwith when used well. Theoretically you’d have to use XML, but JSON is more common. *Ajaj* doesn’t sound very reliable though.
JSON
JavaScript Object Notation. In short, the way JavaScript writes down object variables.
XML
Extensible Markup Language. Another way of writing objects, in this context.

Note: WCF can cooperate with the *ASP.NET scriptmanager* and ASP.NET ajax(*If this is of interest to you, request a session)*. WCF can also cooperate with *jQuery*, the leading JavaScript library for dynamic page manipulation. Both of these topics are beyond the scope of this session. If it talks XML, SOAP or JSON, they can interoperate.

First, download and install *Fiddler* from:

```
http://www.fiddler2.com/Fiddler2/version.asp
```

Fiddler is a *HTTP debugger* that allws you to craft custom requests- many webmasters think *Security Through Obscurity* is a good thing, but we’re not one of them.

Next we’ll prepare our service.
Create a new class named PostedData like this:

```
    [DataContract]
    public class PostedData
    {
        [DataMember]
        public string FirstName { get; set; }
        [DataMember]
        public string LastName { get; set; }
    }
```

Add the following to your service interface:

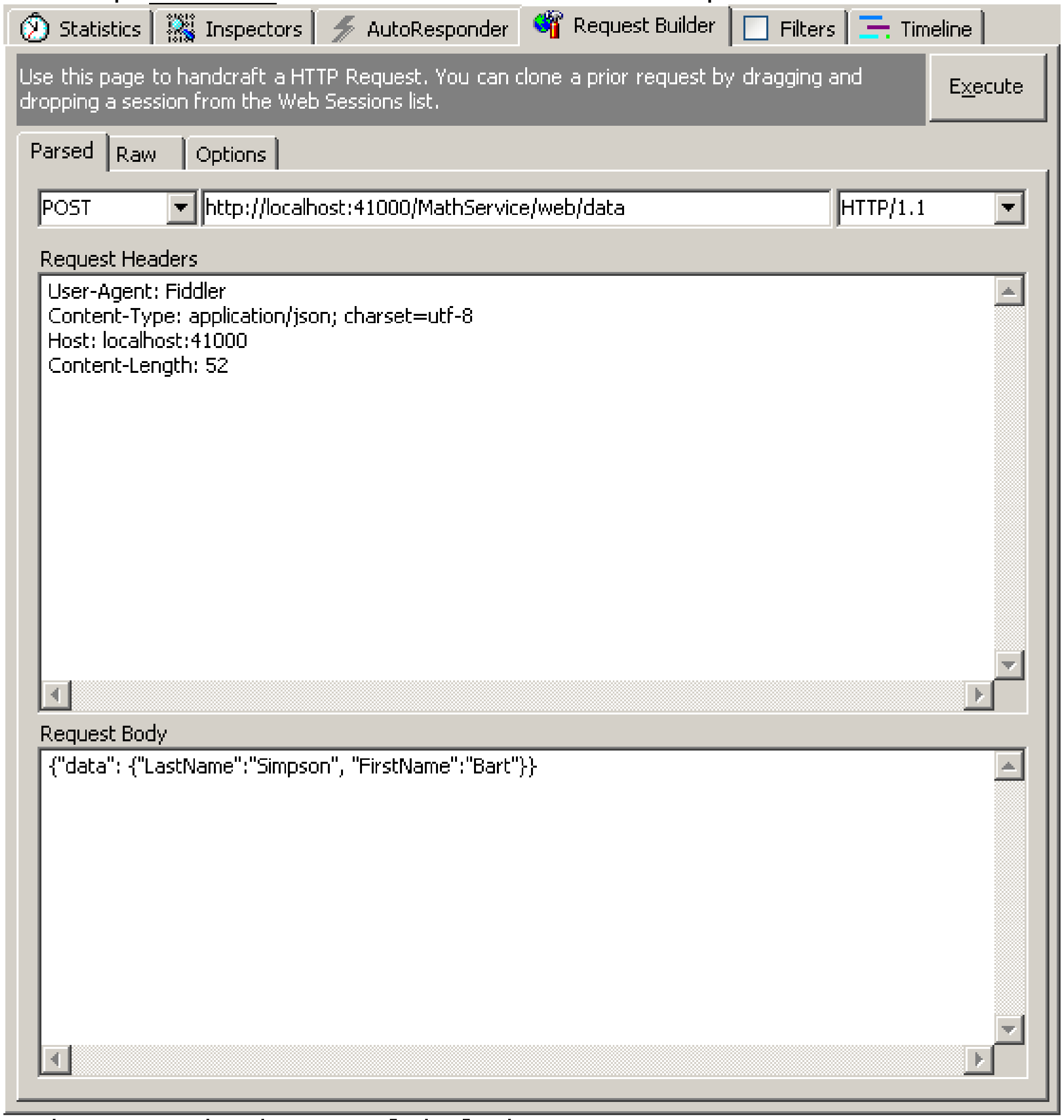
```
        [OperationContract]
        [WebInvoke(Method = "POST",
            ResponseFormat = WebMessageFormat.Json,
            BodyStyle = WebMessageBodyStyle.WrappedRequest,
            UriTemplate = "data")]
        void PostData(PostedData data);
```

Then implement the method in your service with the following code:

```
        public void PostData(PostedData data)
        {
            if (data == null)
            {
                Console.WriteLine("null");
                return;
            }

            Console.WriteLine("Received: " + data.FirstName + " " +
data.LastName);
        }
```

Run your application.
Fire up *Fiddler* and create a custom request as shown:



Use this page to handcraft a HTTP Request. You can clone a prior request by dragging and dropping a session from the Web Sessions list.

POST http://localhost:41000/MathService/web/data HTTP/1.1

Request Headers

```
User-Agent: Fiddler
Content-Type: application/json; charset=utf-8
Host: localhost:41000
Content-Length: 52
```

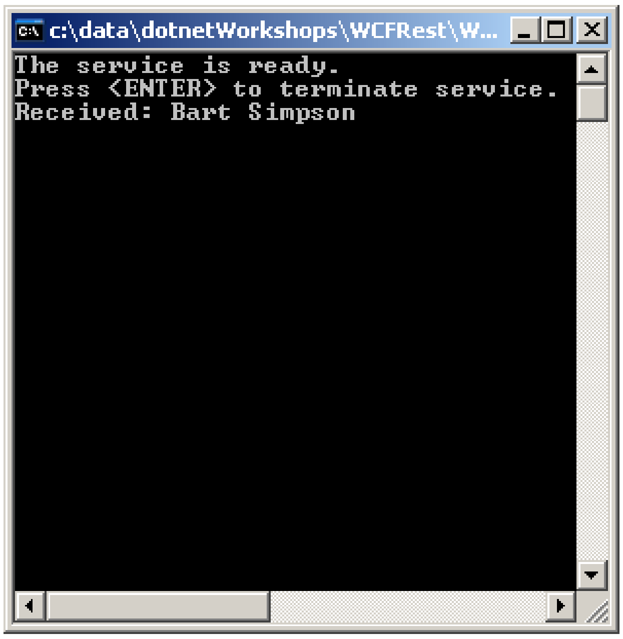
Request Body

```
{"data": {"LastName":"Simpson", "FirstName":"Bart"}}
```

And press the button labeled *Execute*.

You may be wondering why we've wrapped the sent data class with FirstName and LastName properties in a parent object with one property called 'data'.

```
{"data": {"LastName":"Simpson", "FirstName", "Bart"}}
wrapped parent / object sent
```



```
The service is ready.
Press <ENTER> to terminate service.
Received: Bart Simpson
```

This is because we've set the *bodystyle* to *WrappedRequest* in the interface attribute for the method call, wich can be used to send in multiple parameters.

If we set the bodystyle to *Bare*, we would just send the object and not a wrapper object:

```
{"LastName":"Simpson",
"FirstName", "Bart"}
```

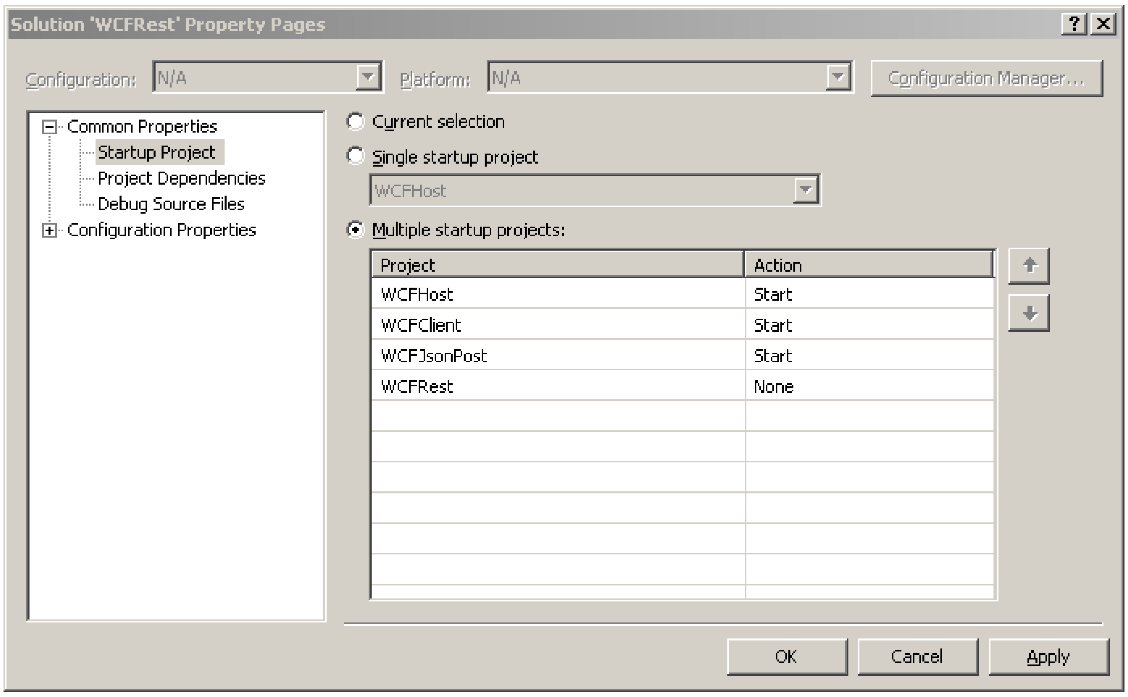
Try it out now!

Sending JSON with C#

Working with REST services cannot be done with Add Service Reference like you could with SOAP. However, there are classes to help you get where you need to:

- *WebHttpRequest*
  Makes HTTP requests where you place the body contents in a Stream and read the reply from another Stream.
- *DataContractJsonSerializer*
  Serializes a type to JSON

Add a new *windows console project* called *WCFJsonPost* to the solution. Set it up to start when you run the application:


Solution 'WCFRest' Property Pages

Configuration: N/A Platform: N/A

- Common Properties
  - Startup Project
  - Project Dependencies
  - Debug Source Files
- Configuration Properties

( ) Current selection
( ) Single startup project: WCFHost
(●) Multiple startup projects:

Project	Action
WCFHost	Start
WCFClient	Start
WCFJsonPost	Start
WCFRest	None

Add a reference in your project to *System.ServiceModel.Web* and *System.Runtime.Serialization*.

Then add the following code to your Program class:

```
    internal class Program
    {
        internal static void Main(string[] args)
        {
            Thread.Sleep(10000);
            TestWcfRestPost();
        }

        private static void TestWcfRestPost()
        {
            // create the serializer that saves classes as JSON
            DataContractJsonSerializer serializer = new
DataContractJsonSerializer(typeof(PostedData));
            PostedData dataToPost = new PostedData
                                        {
                                            FirstName = "Bart",
                                            LastName = "Simpson"
                                        };
            // set up our request
            HttpWebRequest request =
(HttpWebRequest)HttpWebRequest.Create(@"http://localhost:41000/MathServ
ice/web/data");
            request.Accept = "*/*";
            // we're going to post JSON
            request.ContentType = "application/json";
            request.Method = "POST";
            using (Stream stream = request.GetRequestStream())
            {
                // send data
                serializer.WriteObject(stream, dataToPost);
                stream.Flush();
            }

            // get the response
            HttpWebResponse response =
(HttpWebResponse)request.GetResponse();
            using(Stream stream = response.GetResponseStream())
            {
                using(TextReader reader = new StreamReader(stream))
                {
                    Console.WriteLine(reader.ReadToEnd());
                }
            }
        }
    }
```

Run the application to see the same results as the ones you had with Fiddler, that is, if you have put the *bodystyle* to *Bare* at the end of previous topic.

Further reading

Basic WCF Programming
http://msdn.microsoft.com/en-us/library/ms731067.aspx
WCF Syndication
http://msdn.microsoft.com/en-us/library/bb412202.aspx
WCF Rest
http://msdn.microsoft.com/en-us/library/bb412169.aspx
Rest in general (O'Reilly Restful Web Services)
http://oreilly.com/catalog/9780596529260

or find any good book to help you get started. WCF is not going away, and more and more Microsoft technologies use it. In a few years ASP.NET ASMX web services will have disappeared in favour of WCF services. Try to look at the book contents before hand though, most books on WCF hardly focus on restful web service development.

Glossary (from MSDN)

message
A message is a self-contained unit of data that may consist of several parts, including a body and headers.

service
A service is a construct that exposes one or more endpoints, with each endpoint exposing one or more service operations.

endpoint
An endpoint is a construct at which messages are sent or received (or both). It comprises a location (an address) that defines where messages can be sent, a specification of the communication mechanism (a binding) that described how messages should be sent, and a definition for a set of messages that can be sent or received (or both) at that location (a service contract) that describes what message can be sent.

An WCF service is exposed to the world as a collection of endpoints.

application endpoint
An endpoint exposed by the application and that corresponds to a service contract implemented by the application.

infrastructure endpoint
An endpoint that is exposed by the infrastructure to facilitate functionality that is needed or provided by the service that does not relate to a service contract. For example, a service might have an infrastructure endpoint that provides metadata information.

address
An address specifies the location where messages are received. It is specified as a Uniform Resource Identifier (URI). The URI schema part names the transport mechanism to use to reach the address, such as HTTP and TCP. The hierarchical part of the URI contains a unique location whose format is dependent on the transport mechanism.

The endpoint address enables you to create unique endpoint addresses for each endpoint in a service, or under certain conditions share an address across endpoints.

binding
A binding defines how an endpoint communicates to the world. It is constructed of a set of components called binding elements that "stack" one on top of the other to create the communication infrastructure. At the very least, 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. For more information, see Configuring Windows Communication Foundation Services.

binding element
A binding element represents a particular piece of the binding, such as a transport, an encoding, an implementation of an infrastructure-level protocol (such as WS-ReliableMessaging), or any other component of the communication stack.

behaviors
A behavior is a component that controls various run-time aspects of a service, an endpoint, a particular operation, or a client. Behaviors are grouped according to scope: common behaviors affect all endpoints globally, service behaviors affect only service-related aspects, endpoint behaviors affect only endpoint-related properties, and operation-level behaviors affect particular operations. For example, one service behavior is throttling, which specifies how a service reacts when an excess of messages threaten to overwhelm its handling capabilities. An endpoint behavior, on the other hand, controls only aspects relevant to endpoints, such as how and where to find a security credential.

system-provided bindings
WCF includes a number of system-provided bindings. These are collections of binding elements that are optimized for specific scenarios. For example, the WSHttpBinding is designed for interoperability with services that implement various WS-* specifications. These predefined bindings save time by presenting only those options that can be correctly applied to the specific scenario. If a predefined binding

does not meet your requirements, you can create your own custom binding.

configuration versus coding
Control of an application can be done either through coding, through configuration, or through a combination of both. Configuration has the advantage of allowing someone other than the developer (for example, a network administrator) to set client and service parameters after the code is written and without having to recompile. Configuration not only enables you to set values like endpoint addresses, but also allows further control by enabling you to add endpoints, bindings, and behaviors. Coding allows the developer to retain strict control over all components of the service or client, and any settings done through the configuration can be inspected and if needed overridden by the code.

service operation
A service operation is a procedure defined in a service's code that implements the functionality for an operation. This operation is exposed to clients as methods on a WCF client. The method may return a value, and may take an optional number of arguments, or take no arguments, and return no response. For example, an operation that functions as a simple "Hello" can be used as a notification of a client's presence and to begin a series of operations.

service contract
The service contract ties together multiple related operations into a single functional unit. The contract can define service-level settings, such as the namespace of the service, a corresponding callback contract, and other such settings. In most cases, the contract is defined by creating an interface in the programming language of your choice and applying the ServiceContractAttribute attribute to the interface. The actual service code results by implementing the interface.

operation contract
An operation contract defines the parameters and return type of an operation. When creating an interface that defines the service contract, you signify an operation contract by applying the OperationContractAttribute attribute to each method definition that is part of the contract. The operations can be modeled as taking a single message and returning a single message, or as taking a set

of types and returning a type. In the latter case, the system will determine the format for the messages that need to be exchanged for that operation.

message contract
A message contract describes the format of a message. For example, it declares whether message elements should go in headers versus the body, what level of security should be applied to what elements of the message, and so on.

fault contract
A fault contract can be associated with a service operation to denote errors that can be returned to the caller. An operation can have zero or more faults associated with it. These errors are SOAP faults that are modeled as exceptions in the programming model.

data contract
The data types a service uses must be described in metadata to enable others to interoperate with the service. The descriptions of the data types are known as the data contract, and the types can be used in any part of a message, for example, as parameters or return types. If the service is using only simple types, there is no need to explicitly use data contracts.

hosting
A service must be hosted in some process. A host is an application that controls the lifetime of the service. Services can be self-hosted or managed by an existing hosting process.

self-hosted service
A self-hosted service is one that runs within a process application that the developer created. The developer controls its lifetime, sets the properties of the service, opens the service (which sets it into a listening mode), and closes the service.

hosting process
A hosting process is an application that is designed to host services. These include Internet Information Services (IIS), Windows Activation Services (WAS), and Windows Services. In these hosted scenarios, the host controls the lifetime of the service. For example, using IIS you can set up a virtual directory that contains the service assembly

and configuration file. When a message is received, IIS starts the service and controls its lifetime.

instancing
A service has an instancing model. There are three instancing models: "single," in which a single CLR object services all the clients; "per call," in which a new CLR object is created to handle each client call; and "per session," in which a set of CLR objects are created, one for each separate session. The choice of an instancing model depends on the application requirements and the expected usage pattern of the service.

client application
A client application is a program that exchanges messages with one or more endpoints. The client application begins by creating an instance of a WCF client and calling methods of the WCF client. It is important to note that a single application can be both a client and a service.

channel
A channel is a concrete implementation of a binding element. The binding represents the configuration, and the channel is the implementation associated with that configuration. Therefore, there is a channel associated with each binding element. Channels stack on top of each other to create the concrete implementation of the binding: the channel stack.

WCF client
A WCF client is a client-application construct that exposes the service operations as methods (in the .NET Framework programming language of your choice, such as Visual Basic or Visual C#). Any application can host a WCF client, including an application that hosts a service. Therefore, it is possible to create a service that includes WCF clients of other services.

A WCF client can be automatically generated by using the ServiceModel Metadata Utility Tool (Svcutil.exe) and pointing it at a running service that publishes metadata.

metadata
The metadata of a service describes the characteristics of the service that an external entity needs to understand to communicate with the service. Metadata can be consumed by the ServiceModel Metadata Utility Tool (Svcutil.exe) to

generate a WCF client and accompanying configuration that a client application can use to interact with the service.

The metadata exposed by the service includes XML schema documents, which define the data contract of the service, and WSDL documents, which describe the methods of the service.

When enabled, metadata for the service is automatically generated by WCF by inspecting the service and its endpoints. To publish metadata from a service, you must explicitly enable the metadata behavior.

security
Security in WCF includes confidentiality (encryption of messages to prevent eavesdropping), integrity (the means for detection of tampering with the message), authentication (the means for validation of servers and clients), and authorization (the control of access to resources). These functions are provided by either leveraging existing security mechanisms, such as TLS over HTTP (also known as HTTPS), or by implementing one or more of the various WS-* security specifications.

transport security mode
Security can be provided by one of three modes: transport mode, message security mode, and transport with message credential mode. The transport security mode specifies that confidentiality, integrity, and authentication are provided by the transport layer mechanisms (such as HTTPS). When using a transport like HTTPS, this mode has the advantage of being efficient in its performance, and well understood because of its prevalence on the Internet. The disadvantage is that this kind of security is applied separately on each hop in the communication path, making the communication susceptible to a "man in the middle" attack.

message security mode
Message security mode specifies that security is provided by implementing one or more of the security specifications, such as the specification named "Web Services Security: SOAP Message Security" (available at http://go.microsoft.com/fwlink/?LinkId=94684). Each message contains the necessary mechanisms to provide security during its transit, and to enable the receivers to detect tampering and to decrypt the messages. In this sense, the security is encapsulated within every message, providing

end-to-end security across multiple hops. Because security information becomes part of the message, it is also possible to include multiple kinds of credentials with the message (these are referred to as claims). This approach also has the advantage of enabling the message to travel securely over any transport, including multiple transports between its origin and destination. The disadvantage of this approach is the complexity of the cryptographic mechanisms employed, resulting in performance implications.

transport with message credential security mode
This mode uses the transport layer to provide confidentiality, authentication, and integrity of the messages, while each of the messages can contain multiple credentials (claims) required by the receivers of the message.

WS-*
Shorthand for the growing set of Web Service (WS) specifications, such as WS-Security, WS-ReliableMessaging, and so on, that are implemented in WCF.