

1.0 – Introduction

What is a RESTful Web Service?

A RESTful Web Service is a way of doing communication between client and server.

Client Application	Working With	Protocol
Web Browser	Web pages, HTML, images, text	HTTP
Application Program	Data	HTTP

Conceptually, using RESTful Web Services is very similar to web browsing. But instead of web pages we are working with data. Same HTTP protocol concepts are used. Writing RESTful Web Services are simpler than SOAP and are pretty much universally supported by all operating systems and programming languages. This allows for easy cross-platform integration.

1.1 – RESTful Web Service HTTP Methods

RESTful Web Services use the following HTTP verbs for doing CRUD operations:

- To create a resource on the server, use POST
- To retrieve a resource, use GET
- To change the state of a resource or to update it, use PUT
- To remove or delete a resource, use DELETE

A “resource” is a fancy word for database record, file, or some other type of persistent or dynamic data.

1.2 – RESTful Web Service URL and URI Conventions

To retrieve a collection of records, you would use a URL like this:

<http://www.domain.com/resources>

To retrieve a particular record, you would use a URL like this:

<http://www.domain.com/resources/42>

Both these examples would use the HTTP GET verb. Records will be returned in an XML format.

1.3 – Personal Preference

Following strict RESTful Web Services guidelines is like religion for some people.

The GET verb has issues. You must encode meta-characters in the URL. And with GET, you will show all your parameter data in the URL line which seems messy to me (GUIDs are nasty). In addition, PUT and DELETE are usually blocked by most firewalls.

My preference is to do all communication using HTTP POST. I use the same style of client coding every time. And here is the URL/URI convention I like to follow:

<http://www.domain.com/path/Object/Verb>

Do some Google searches for more information on RESTful Web Services. If you want to follow a strict RESTful methodology you can get more info.

1.4 – A RESTful Service API

You have to be careful when designing your RESTful Web Service API. Whatever you publish, you have to keep in mind that applications will be depending on your specific contracts. If you change your API, it is very easy to break existing applications.

1.5 – WCF-REST Starter Kit

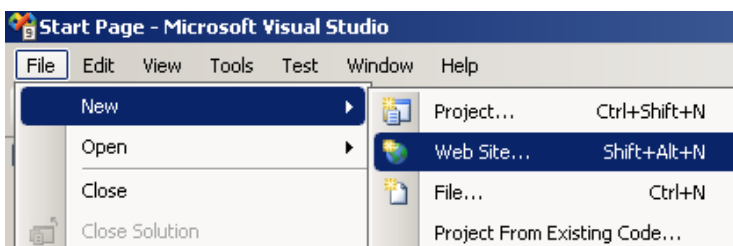
To write RESTful Web Services in .Net you will need the “WCF-REST Starter Kit”.

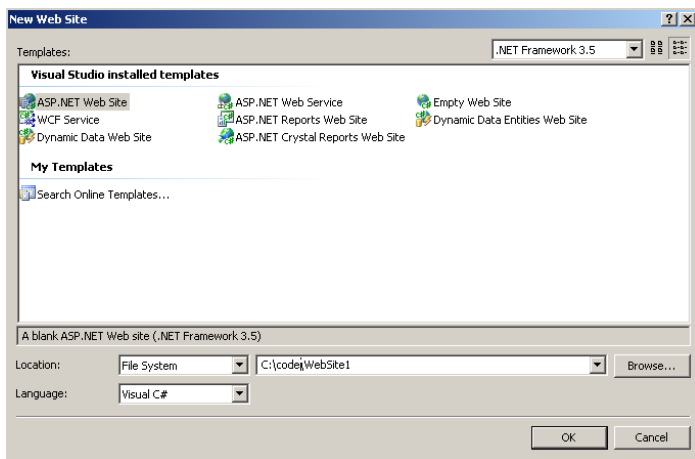
Google “WCF-REST Starter Kit download” to get the necessary installation files. Installation is tricky but I will show you a quick work-around.

2.0 – Writing Your First RESTful Web Service

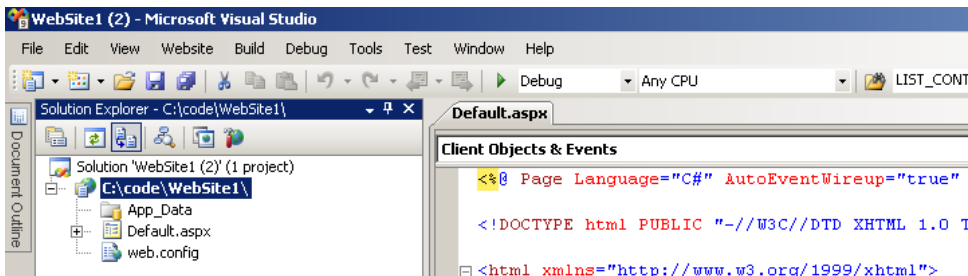
At first this may seem a little tricky and convoluted, but it will become familiar and is the same thing every time.

Launch Visual Studio and create a new web site:





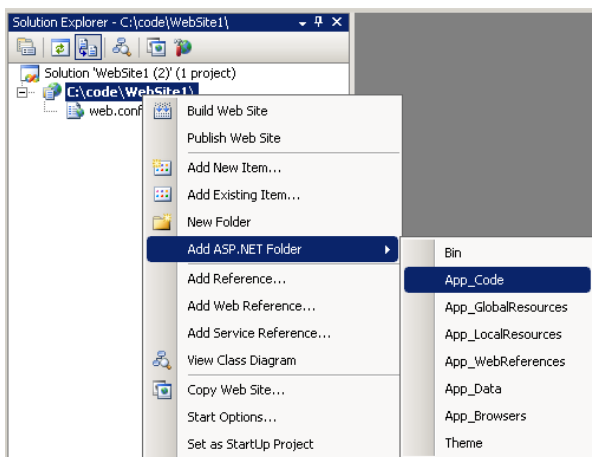
You should see the following:



Delete everything except the web.config:

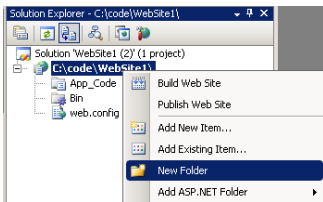


Add an "App_Code" folder by right clicking over the "c:\code\WebSite1" folder as follows:

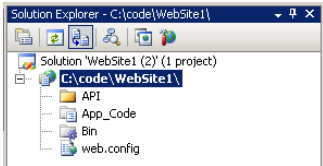


Add a "bin" folder using the same method.

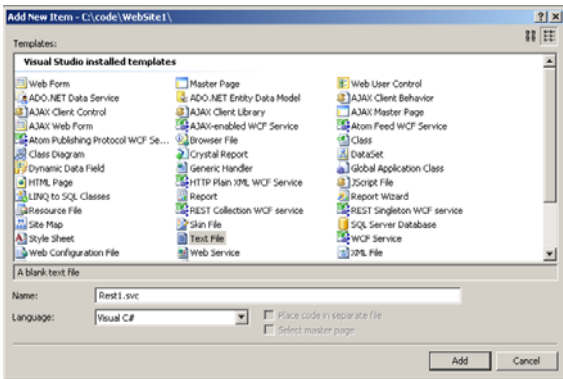
Add an “API” folder by right clicking over the “c:\code\WebSite1” folder as follows:



You should have the following in your Solution Explorer:



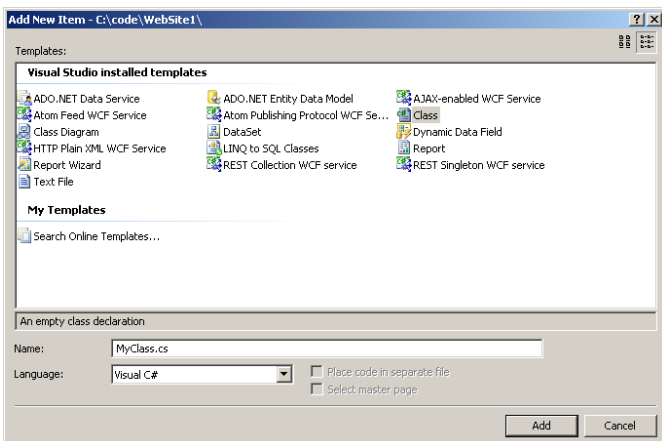
Right click over the “API” folder and select “Add New Item...” from the popup menu. Select “Text File” for the template and “Rest1.svc” for the file name as shown below:



Edit the “Rest1.svc” file to have the following line <<copy in file c:\codeSnippets>>:

```
<%@ ServiceHost Factory="System.ServiceModel.Activation.WebServiceHostFactory" Service="MyNamespace.MyClass" %>
```

Next, we are going to create a new class file. Right click over the “App_Code” folder and select “Add New Item...” from the popup menu. Select “Class” as the template, “MyClass.cs” for the file:



When writing WCF RESTful Web Service, for your code to work correctly, you have to add an interface for your class to implement. Right click over the “App_Code” folder and select “Add New Item...” from the popup menu. Select “Class” as the template, and enter “IMyClass.cs” for the file.

Edit the “IMyClass.cs” to look like the following <<copy in file c:\codeSnippets>>:

```
1  using System;
2  using System.Collections;
3  using System.Collections.Generic;
4  using System.Collections.Specialized;
5  using System.Runtime.Serialization;
6  using System.ServiceModel;
7  using System.ServiceModel.Web;
8  using System.ServiceModel.Activation;
9  using System.Linq;
10 using System.Net;
11 using Microsoft.ServiceModel.Web;
12 using Microsoft.ServiceModel.Web.SpecializedServices;
13
14 namespace MyNameSpace
15 {
16     [ServiceContract]
17     public partial interface IMyClass
18     {
19         [OperationContract]
20         [WebGet(UriTemplate = "/Echo")]
21         string Echo();
22     }
23 }
```

Edit the “MyClass.cs” to look like the following:

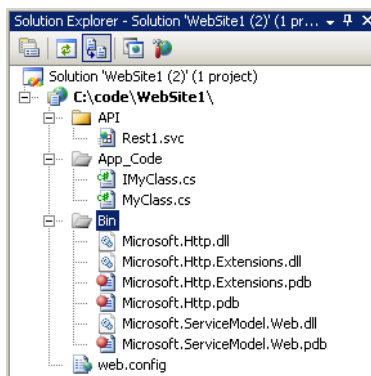
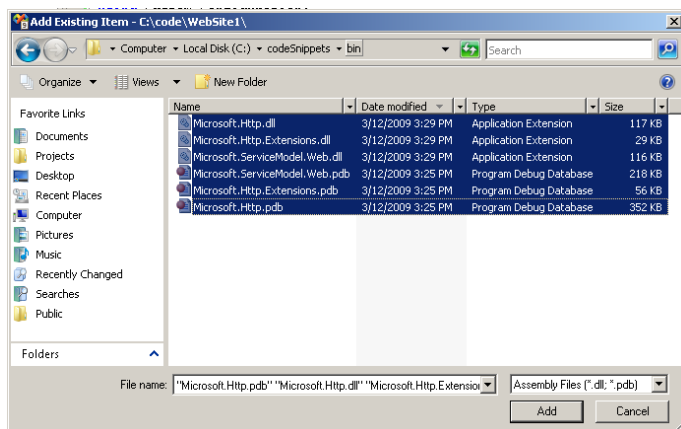
```
1  using System;
2  using System.Data;
3  using System.Configuration;
4  using System.Collections;
5  using System.Collections.Generic;
6  using System.ComponentModel;
7  using System.Configuration;
8  using System.Data;
9  using System.Linq;
10 using System.Runtime.Serialization;
11 using System.Text;
12 using System.Web;
13 using System.Web.Configuration;
14 using System.Web.Security;
15 using System.Web.UI;
16 using System.Web.UI.HtmlControls;
17 using System.Web.UI.WebControls;
18 using System.Web.UI.WebControls.WebParts;
19 using System.Xml.Linq;
20
21 namespace MyNameSpace
22 {
23
24     public class MyClass : IMyClass
25     {
26         public MyClass()
27         {
28         }
29
30         public string Echo()
31         {
32             return "Hello World: " + DateTime.Now.ToString();
33         }
34     }
35 }
```

```

33     }
34 }
35 }

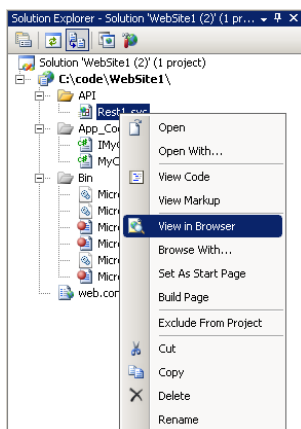
```

Import the WCF RESTful Service DLLs to the website “Bin” folder:

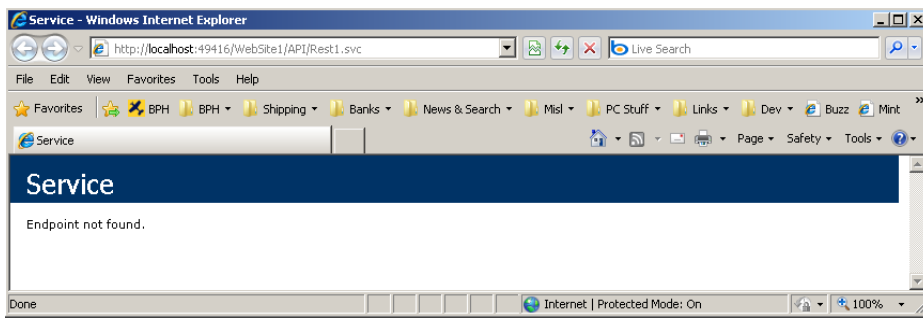


Ideally, these DLLs should be properly installed (see the WCF RESTful Web Service toolkit installation instructions). But this is a quick solution that works great.

To test our WCF RESTful Web Service, we are going to use Internet Explorer as the client. Right click over the file “Rest1.svc” and select “View in Browser...” from the popup menu:

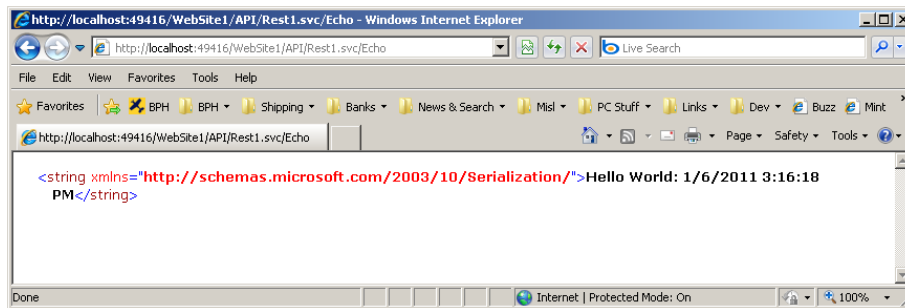


You will see the following:



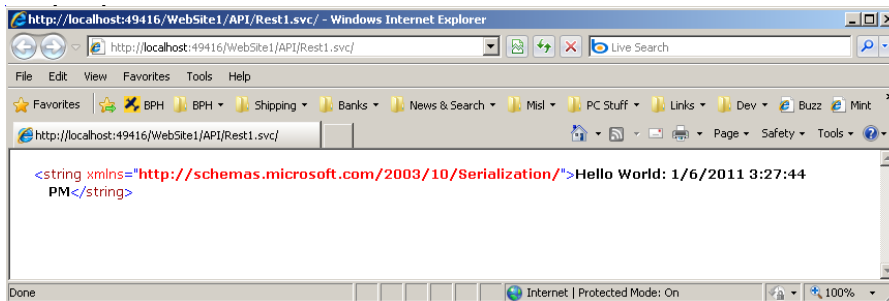
The URL to: <http://.../WebSite1/API/Rest1.svc/Echo>

You should see the following:



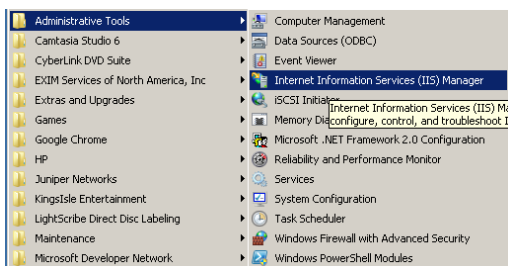
Adding a “/Echo” GET API is very useful for checking that your classes are working correctly.

```
1 [OperationContract]
2 [WebGet(UriTemplate = "/")]
3 string Echo2();
```

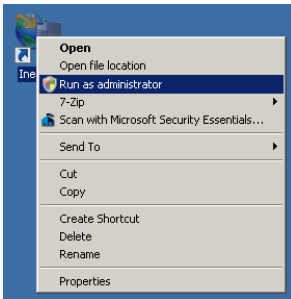


2.1 – Publish to IIS

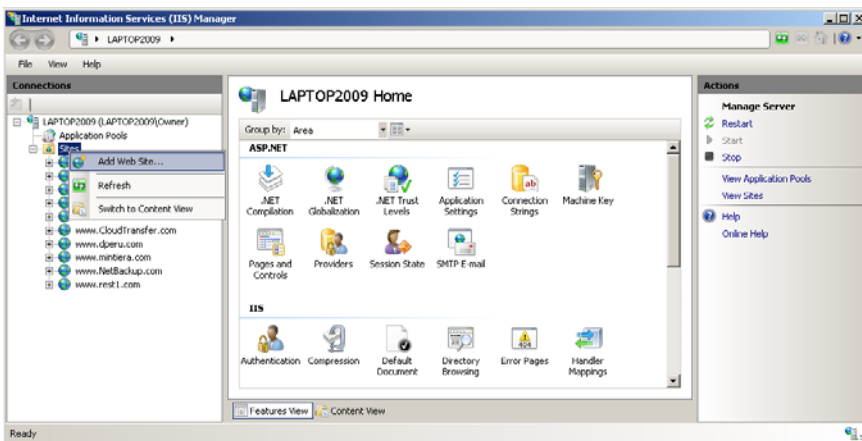
Next, we are going to publish our website to IIS and test our API. To do this, we are going to create a new website. Create a desktop icon for the IIS Manager because we will need to run in admin mode:



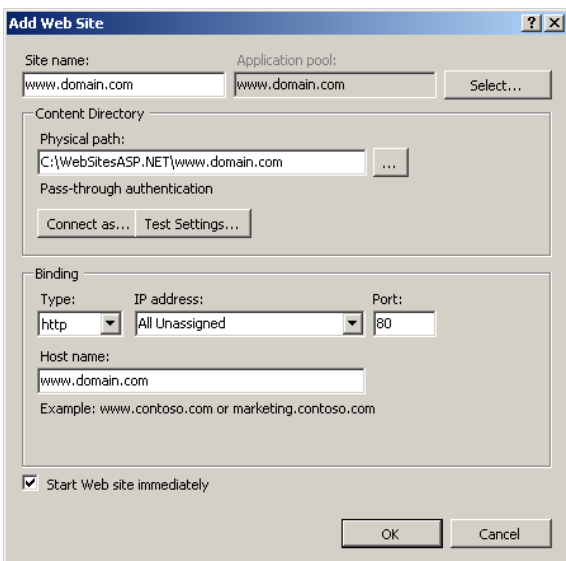
Right click over the desktop icon and select “Run as administrator” from the popup menu:



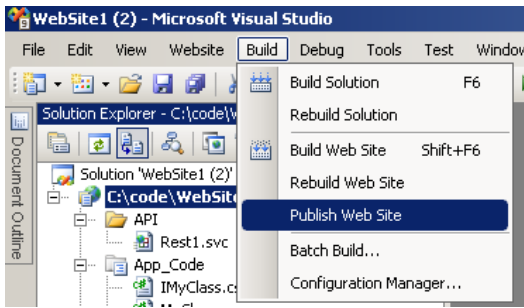
Creating an icon and using the “Run as administrator” menu may only be needed for Vista and Windows 7 (if you do not have IIS running on your Vista or Windows 7 machine, use Google to find out how to do the install). For Windows 2003 and 2008 Server it will probably work fine when logged in as Administrator. Add a new website:



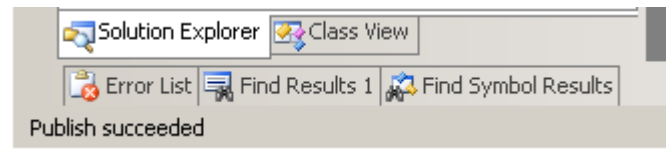
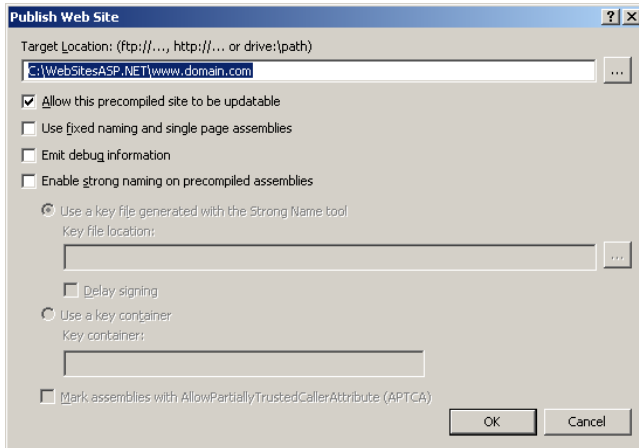
Create a new folder “c:\WebSitesASP.NET\www.domain.com” and use “www.domain.com” for the host header:



Next we return back to Visual Studio and publish our site:



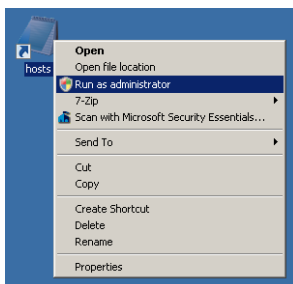
Select the folder “c:\WebSitesASP.NET\www.domain.com” as the publish point:



To test our site using Internet Explorer we have to **fake a DNS entry**. Use the “Start/Run/cmd” and then “ipconfig” to determine our current IP address. Next, create a Shortcut on the desktop for editing the local HOSTS file.

Use target: C:\Windows\System32\notepad.exe C:\Windows\System32\drivers\etc\hosts

Right click over the desktop icon and select “Run as administrator” from the popup:



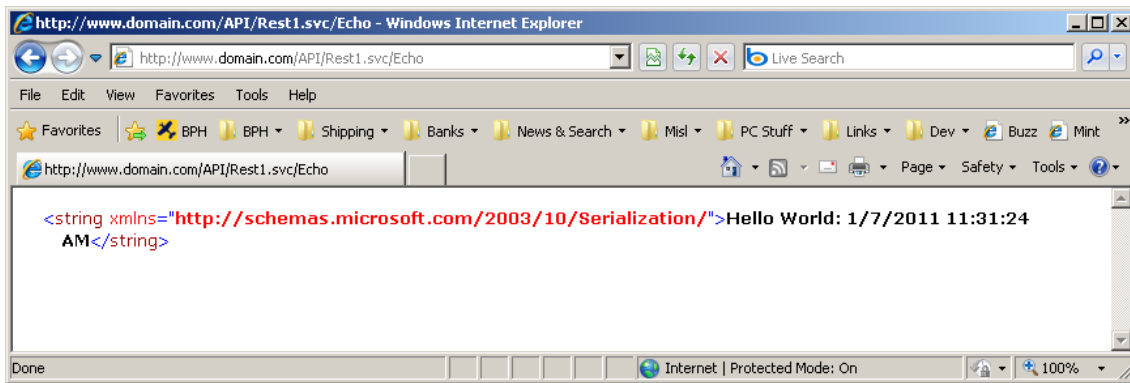
Add the line to the HOSTS file with the correct IP address for our host name:

```
# 102.54.94.97 rhino.acme.com # source server
# 38.25.63.10 x.acme.com # x client host

127.0.0.1 localhost
::1 localhost

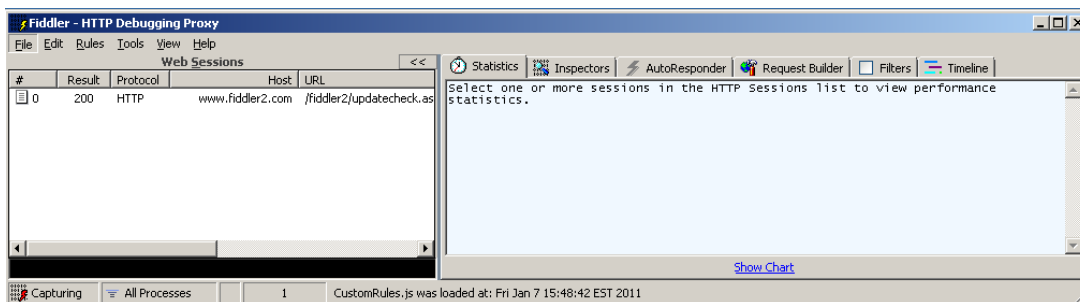
192.168.1.4 www.domain.com
```

Next, in a web browser, type in the URL: <http://www.domain.com/API/Rest1.svc/Echo>

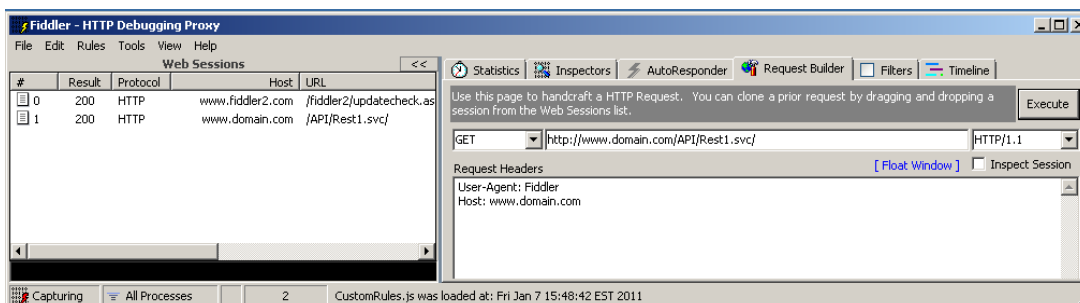


2.2 – Fiddler

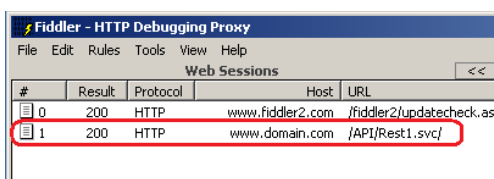
Fiddler is a really nice program you can use for testing your RESTful Web Services:



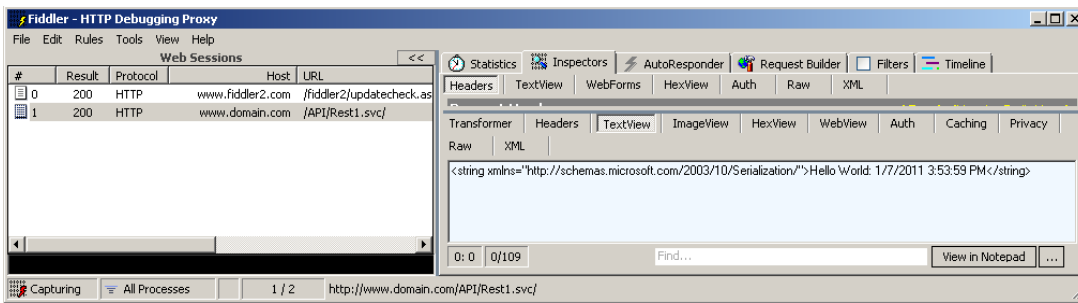
Go to “Request Builder” and enter our URL:



Double click on the return result area:



Click on “TextView” and you will see the following:



<< show IE surfing with Fiddler >>

3.0 – Working With GET Verb Parameters

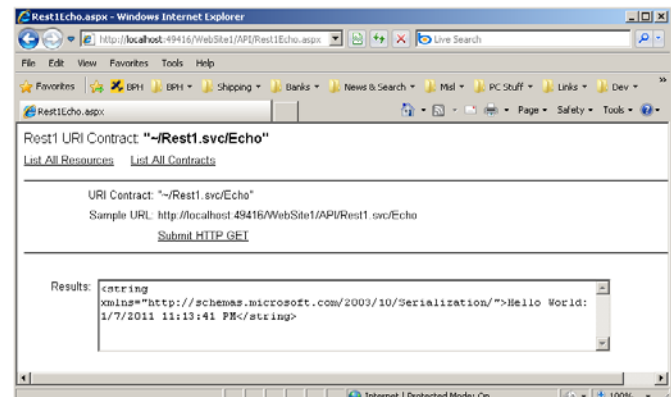
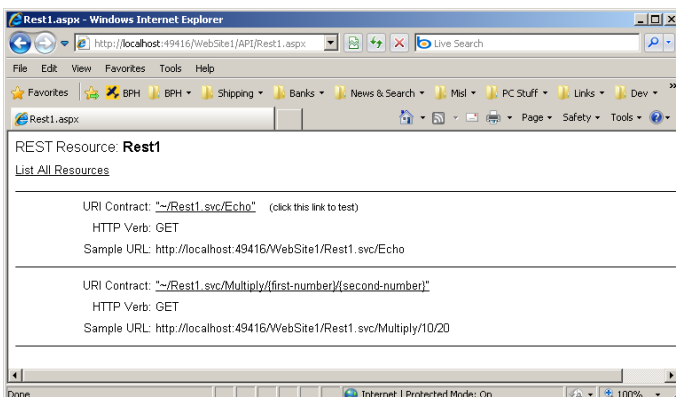
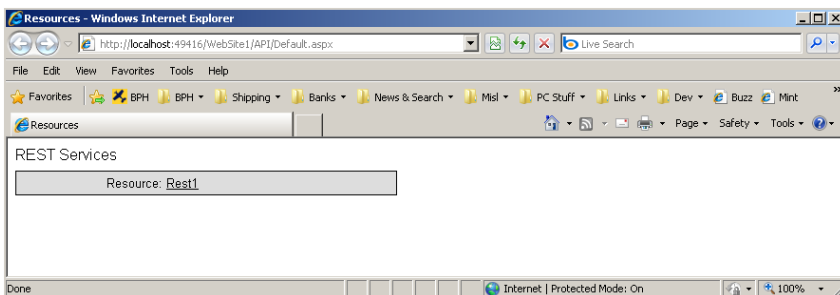
Next, we are going to show how WCF handles passing parameters to a GET contract. Here is the URL we are going to implement:

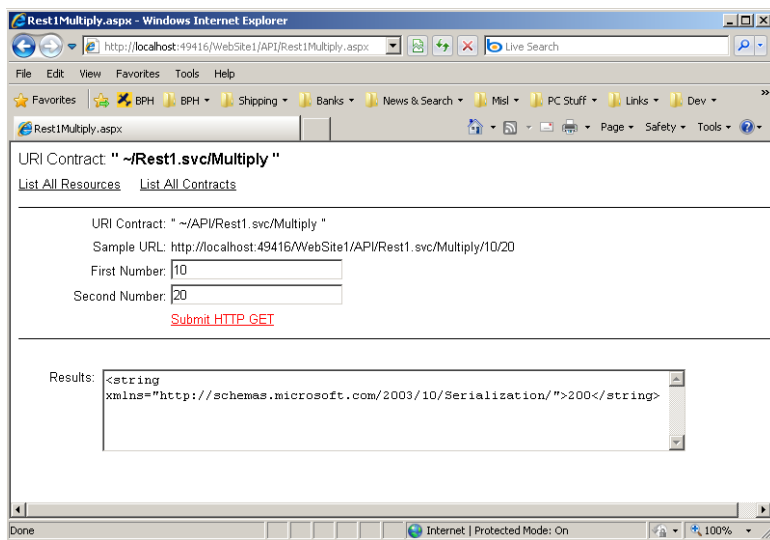
<http://www.domain.com/API/Rest1.svc/Multiply/10/20>

The “10” and the “20” are the two numbers to multiply together. The API will return the multiplied result as a string.

<< copy in c:\codeSnippets\Section3_0 files >>

Here is the new code in action, right click over the Default.aspx and select “View in browser”.





Here is a partial listing for Rest1Multiply.aspx.cs:

```

1  public partial class API_Rest1Multiply : System.Web.UI.Page
2  {
3      protected void Page_Load(object sender, EventArgs e)
4      {
5          string url = GetUrl();
6          lblSampleUrl.Text = url;
7      }
8
9      private string GetUrl()
10     {
11         string s1 = "";
12         string s2 = "";
13         if (Request.UrlReferrer == null)
14         {
15             s1 = Request.Url.AbsolutePath;
16             s2 = Request.Url.AbsoluteUri;
17         }
18         else
19         {
20             s1 = Request.UrlReferrer.AbsolutePath;
21             s2 = Request.UrlReferrer.AbsoluteUri;
22         }
23         int ix = s2.IndexOf(s1);
24         string s3 = s2.Substring(0, ix);
25         StringBuilder sb = new StringBuilder();
26         sb.Append(s3);
27         sb.Append(Request.ApplicationPath);
28         if (sb.ToString().Substring(sb.ToString().Length - 1, 1) != "/")
29         {
30             sb.Append("/");
31         }
32         sb.Append("API/Rest1.svc/Multiply");
33         return sb.ToString();
34     }
35
36     protected void LinkButton1_Click(object sender, EventArgs e)
37     {
38         string url = GetUrl();
39
40         StringBuilder sb = new StringBuilder();
41         sb.Append(url);
42         sb.Append("/");
43         sb.Append(tbxFirstNumber.Text);
44         sb.Append("/");

```

```

45     sb.Append(tbxSecondNumber.Text);
46
47     lblSampleUrl.Text = sb.ToString();
48
49     HttpWebRequest request = (HttpWebRequest)WebRequest.Create(sb.ToString());
50
51     request.Method = "GET";
52     request.Accept = "**/*";
53     request.ContentType = "application/x-www-form-urlencoded";
54     request.ContentLength = 0;
55     request.UserAgent = "Rest1 Service Test Harness";
56
57     try
58     {
59         HttpWebResponse response = (HttpWebResponse)request.GetResponse();
60         Stream responseStream = response.GetResponseStream();
61         StreamReader streamReader = new StreamReader(responseStream);
62         string result = streamReader.ReadToEnd();
63
64         tbxResult.Text = Utils.IndentXMLString(result);
65     }
66     catch (Exception ex)
67     {
68         tbxResult.Text = ex.Message;
69     }
70 }

```

Here is the current listing of IMyClass.cs:

```

1  using ...
14 namespace MyNameSpace
15 {
16     [ServiceContract]
17     public partial interface IMyClass
18     {
19         [OperationContract]
20         [WebGet(UriTemplate = "/Echo")]
21         string Echo();
22
23         [OperationContract]
24         [WebGet(UriTemplate = "/Multiply/{firstNumber}/{secondNumber}")]
25         string Multiply(string firstNumber, string secondNumber);
26     }
27 }

```

Here is the current listing of MyClass.cs:

```

19 namespace MyNameSpace
20 {
21
22     public class MyClass : IMyClass
23     {
24         public MyClass()
25         {
26         }
27
28         public string Echo()
29         {
30             return "Hello World: " + DateTime.Now.ToString();
31         }
32
33         public string Multiply(string firstNumber, string secondNumber)
34         {
35             try
36             {
37                 int n1 = int.Parse(firstNumber);

```

```

38         int n2 = int.Parse(secondNumber);
39         int product = n1 * n2;
40         return product.ToString();
41     }
42     catch (Exception ex)
43     {
44         return ex.Message;
45     }
46 }
47 }
48 }

```

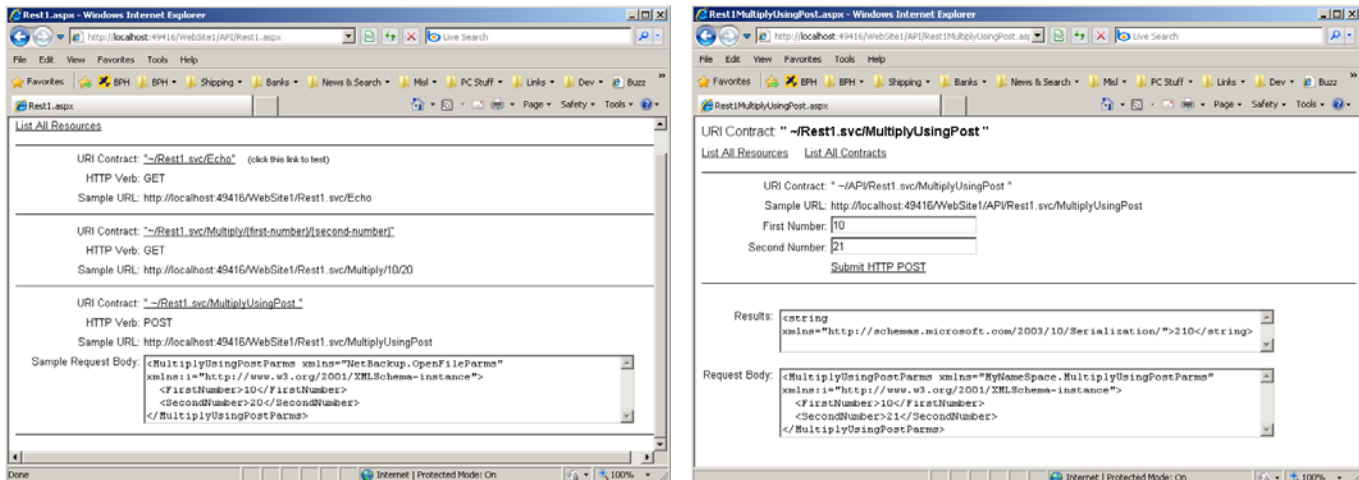
<< publish site and test >>

3.1 – Working With the POST Verb

Next, we are going to show how WCF RESTful support handles passing parameters using a POST contract. Again, the “10” and the “20” are the two numbers to multiply together. And the API will return the multiplied result as a string.

<< copy in c:\codeSnippets\Section3_1 files >>

Here is the new code in action, right click over the Rest1.aspx and select “View in browser”.



Here is a partial listing for Rest1MultiplyUsingPost.aspx.cs (use the same flavor of code below for all POST-type APIs):

```

52     protected void LinkButton1_Click(object sender, EventArgs e)
53     {
54         string url = GetUrl();
55
56         HttpWebRequest request = (HttpWebRequest)WebRequest.Create(url);
57
58         request.Method = "POST";
59         request.Accept = "*//*";
60         request.ContentType = "text/xml";
61         request.UserAgent = "Rest1 Service Test Harness";
62
63         StringBuilder sb = new StringBuilder();
64
65         sb.Append("<MultiplyUsingPostParms xmlns='\"MyNamespace.MultiplyUsingPostParms\"'
xmlns:i='\"http://www.w3.org/2001/XMLSchema-instance\"'>");

```

```

66     sb.Append("<FirstNumber>");
67     sb.Append(tbxFirstNumber.Text.Trim());
68     sb.Append("</FirstNumber>");
69     sb.Append("<SecondNumber>");
70     sb.Append(tbxSecondNumber.Text.Trim());
71     sb.Append("</SecondNumber>");
72     sb.Append("</MultiplyUsingPostParms>");
73
74     reqBody.Text = Utils.IndentXMLString(sb.ToString());
75
76     byte[] encodedRequest = new ASCIIEncoding().GetBytes(sb.ToString());
77     request.ContentLength = encodedRequest.Length;
78
79     Stream reqStream = request.GetRequestStream();
80     reqStream.Write(encodedRequest, 0, encodedRequest.Length);
81     reqStream.Flush();
82     reqStream.Close();
83
84     try
85     {
86         HttpResponseMessage response = (HttpResponseMessage)request.GetResponse();
87         Stream responseStream = response.GetResponseStream();
88         StreamReader streamReader = new StreamReader(responseStream);
89         string result = streamReader.ReadToEnd();
90
91         tbxResult.Text = Utils.IndentXMLString(result);
92     }
93     catch (Exception ex)
94     {
95         tbxResult.Text = ex.Message;
96     }
97 }

```

Here is a partial listing for IMyClass.cs:

```

27     [WebInvoke(UriTemplate = "/MultiplyUsingPost", Method = "POST")]
28     [OperationContract]
29     string MultiplyUsingPost(MyNameSpace.MultiplyUsingPostParms parms);

```

Here is a partial listing for MyClass.cs:

```

19 namespace MyNameSpace
20 {
21     [DataContract(Name = "MultiplyUsingPostParms", Namespace = "MyNameSpace.MultiplyUsingPostParms")]
22     public class MultiplyUsingPostParms
23     {
24         [DataMember(Name = "FirstNumber", Order = 1)]
25         private string _firstNumber;
26         public string FirstNumber
27         {
28             get { return _firstNumber; }
29             set { _firstNumber = value; }
30         }
31         [DataMember(Name = "SecondNumber", Order = 2)]
32         private string _secondNumber;
33         public string SecondNumber
34         {
35             get { return _secondNumber; }
36             set { _secondNumber = value; }
37         }
38     }
39
40     public class MyClass : IMyClass
41     {

```

65

```

66     public string MultiplyUsingPost(MultiplyUsingPostParms parms)
67     {
68         try
69         {
70             int n1 = int.Parse(parms.FirstNumber);
71             int n2 = int.Parse(parms.SecondNumber);
72             int product = n1 * n2;
73             return product.ToString();
74         }
75         catch (Exception ex)
76         {
77             return ex.Message;
78         }
79     }
80 }
81 }

```

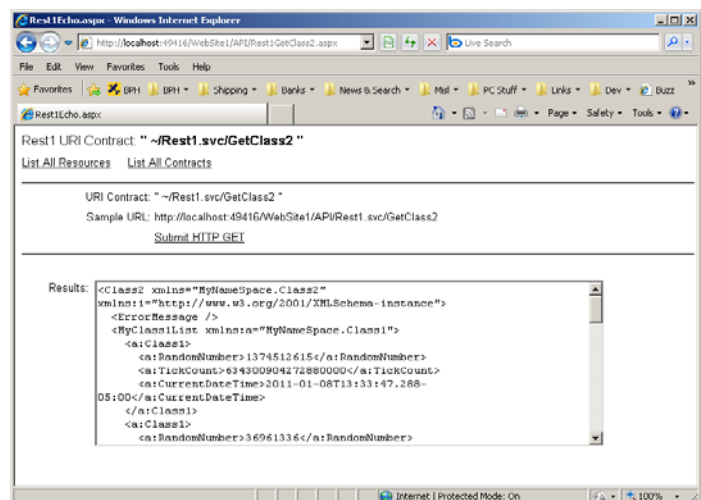
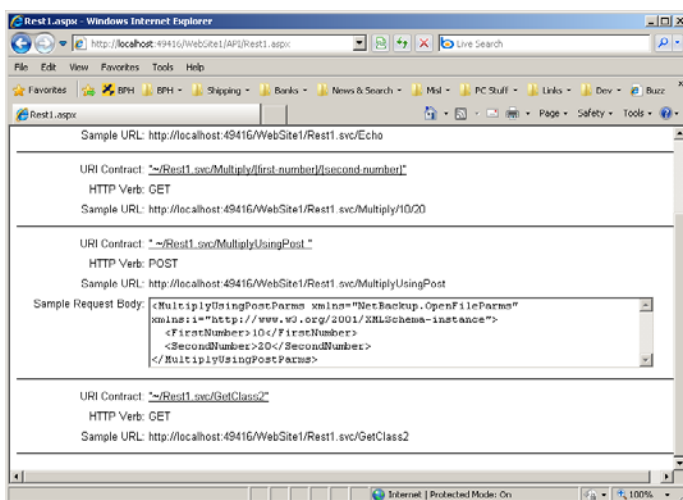
<< publish site >>

3.2 – Parsing XML Return Values

Next, we are going to cover two topics. First, how to return complex classes containing generic collections. And second, how to parse XML containing generic collections.

<< copy in c:\codeSnippets\Section3_2 files >>

Here's our new API:



Here is a partial listing of IMyClass.cs:

```

31     [OperationContract]
32     [WebGet(UriTemplate = "/GetClass2")]
33     Class2 GetClass2();

```

Here is a partial listing of MyClass.cs:

```

40     [DataContract(Name = "Class1", Namespace = "MyNamespace.Class1")]
41     public class Class1
42     {
43         [DataMember(Name = "RandomNumber", Order = 1)]
44         private int _randomNumber;
45         public int RandomNumber

```

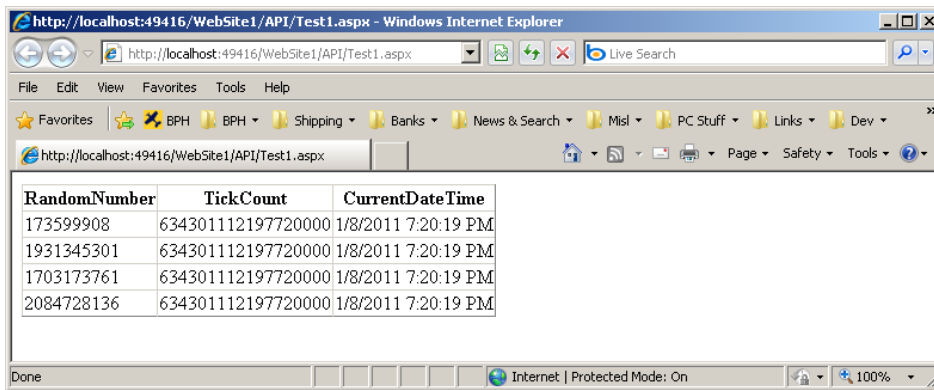


```

46     {
47         get { return _randomNumber; }
48         set { _randomNumber = value; }
49     }
50     [DataMember(Name = "TickCount", Order = 2)]
51     private long _tickCount;
52     public long TickCount
53     {
54         get { return _tickCount; }
55         set { _tickCount = value; }
56     }
57     [DataMember(Name = "CurrentDateTime", Order = 3)]
58     private DateTime _currentDateTime;
59     public DateTime CurrentDateTime
60     {
61         get { return _currentDateTime; }
62         set { _currentDateTime = value; }
63     }
64 }
65
66 [DataContract(Name = "Class2", Namespace = "MyNameSpace.Class2")]
67 public class Class2
68 {
69     [DataMember(Name = "ErrorMessage", Order = 1)]
70     private string _errorMessage;
71     public string ErrorMessage
72     {
73         get { return _errorMessage; }
74         set { _errorMessage = value; }
75     }
76     [DataMember(Name = "MyClass1List", Order = 2)]
77     private List<Class1> _myClass1List;
78     public List<Class1> MyClass1List
79     {
80         get { return _myClass1List; }
81         set { _myClass1List = value; }
82     }
83 }
84
85 public class MyClass : IMyClass
86 {
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126     public Class2 GetClass2()
127     {
128         Class2 result = new Class2();
129         try
130         {
131             result.MyClass1List = new List<Class1>();
132             Random random = new Random();
133             int numberOfRecords = (random.Next()) % 10 + 1;
134             for (int i = 0; i < numberOfRecords; ++i)
135             {
136                 Class1 c1 = new Class1();
137                 c1.CurrentDateTime = DateTime.Now;
138                 c1.TickCount = DateTime.Now.Ticks;
139                 c1.RandomNumber = random.Next();
140                 result.MyClass1List.Add(c1);
141             }
142             result.ErrorMessage = "";
143         }
144         catch (Exception ex)
145         {
146             result.MyClass1List = null;
147             result.ErrorMessage = ex.Message;
148         }
149         return result;
150     }
151 }

```

Now that we have our API, we are going to use another page to demonstrate XML parsing. Here is another page using the GetClass2 API with dynamic binding to a grid control:



RandomNumber	TickCount	CurrentDateTime
173599908	634301112197720000	1/8/2011 7:20:19 PM
1931345301	634301112197720000	1/8/2011 7:20:19 PM
1703173761	634301112197720000	1/8/2011 7:20:19 PM
2084728136	634301112197720000	1/8/2011 7:20:19 PM

Here's the listing for Test1.aspx:

```
1  <%@ Page Language="C#" AutoEventWireup="true" CodeFile="Test1.aspx.cs" Inherits="API_Test1" %>
2
3  <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
4
5  <html xmlns="http://www.w3.org/1999/xhtml">
6  <head runat="server">
7      <title></title>
8  </head>
9  <body>
10     <form id="form1" runat="server">
11         <div style="margin: 10px;">
12             <asp:Label ID="Label1" runat="server" Text=""></asp:Label>
13         </div>
14         <div>
15             <asp:GridView ID="GridView1" runat="server"></asp:GridView>
16         </div>
17     </form>
18 </body>
19 </html>
```

Here's the listing for Test1.aspx.cs:

```
22  using MyNamespace;
23
24  public partial class API_Test1 : System.Web.UI.Page
25  {
26      protected void Page_Load(object sender, EventArgs e)
27      {
28          string url = GetUrl();
29          HttpWebRequest request = (HttpWebRequest)WebRequest.Create(url);
30          request.Method = "GET";
31          request.Accept = "*/*";
32          request.ContentType = "application/x-www-form-urlencoded";
33          request.ContentLength = 0;
34          request.UserAgent = "Rest1 Service Test Harness";
35          try
36          {
37              // Get XML from Rest1:
38              //
39              HttpWebResponse response = (HttpWebResponse)request.GetResponse();
40              Stream responseStream = response.GetResponseStream();
41              StreamReader streamReader = new StreamReader(responseStream);
42              string result = streamReader.ReadToEnd();
43          }
```

```

44 //<Class2 xmlns="MyNameSpace.Class2" xmlns:i="http://www.w3.org/2001/XMLSchema-instance">
45 // <ErrorMessage />
46 // <MyClass1List xmlns:a="MyNameSpace.Class1">
47 // <a:Class1>
48 // <a:RandomNumber>67767601</a:RandomNumber>
49 // <a:TickCount>634301090538540000</a:TickCount>
50 // <a:CurrentDateTime>2011-01-08T18:44:13.854-05:00</a:CurrentDateTime>
51 // </a:Class1>
52 // </MyClass1List>
53 //</Class2>
54
55 // Parse XML into Class2 object:
56 //
57 Class2 c2 = new Class2();
58 byte[] byteArray = Encoding.ASCII.GetBytes(result);
59 MemoryStream stream = new MemoryStream( byteArray );
60 XmlDocument xml = new XmlDocument();
61 xml.Load(stream);
62 XmlNodeList nodeList1 = xml.ChildNodes;
63 foreach (XmlNode node1 in nodeList1)
64 {
65     if (node1.Name == "Class2")
66     {
67         XmlNodeList nodeList2 = node1.ChildNodes;
68         foreach (XmlNode node2 in nodeList2)
69         {
70             if (node2.Name == "ErrorMessage")
71             {
72                 c2.ErrorMessage = node2.InnerText;
73             }
74             if (node2.Name == "MyClass1List")
75             {
76                 c2.MyClass1List = new List<Class1>();
77                 XmlNodeList nodeList3 = node2.ChildNodes;
78                 foreach (XmlNode node3 in nodeList3)
79                 {
80                     if (node3.Name == "a:Class1")
81                     {
82                         Class1 c1 = new Class1();
83                         XmlNodeList nodeList4 = node3.ChildNodes;
84                         foreach (XmlNode node4 in nodeList4)
85                         {
86                             if (node4.Name == "a:RandomNumber")
87                             {
88                                 c1.RandomNumber = int.Parse(node4.InnerText);
89                             }
90                             if (node4.Name == "a:TickCount")
91                             {
92                                 c1.TickCount = long.Parse(node4.InnerText);
93                             }
94                             if (node4.Name == "a:CurrentDateTime")
95                             {
96                                 c1.CurrentDateTime = DateTime.Parse(node4.InnerText);
97                             }
98                         }
99                         c2.MyClass1List.Add(c1);
100                     }
101                 }
102             }
103         }
104     }
105 }
106
107 // Do dynamic data binding with grid view control:
108 //
109 if (c2.MyClass1List != null)
110 {
111     GridView1.DataSource = c2.MyClass1List;
112     GridView1.DataBind();

```

```

113     }
114 }
115 catch (Exception ex)
116 {
117     Label1.Text = ex.Message;
118 }
119 }

```

There may be a better way to rebuild the generic collection from an XML document. But this method has worked okay for me in many applications. Usually, I include the XML parsing code with each class and pass the XmlNodeList as a parameter. This way, each class is responsible for loading its own properties with values.

3.3 – Error Handling and Exceptions

The HTTP protocol forces you to return HTTP status codes for errors. We have all seen and are familiar with these codes: 200: OK, 404: Not Found: 400: Bad Request and 500: Internal Server Error. So there really is not a clean and universally agreed upon way to handle throwing an exception. Here's what I've been doing:

```

70         if (node2.Name == "ErrorMessage")
71         {
72             c2.ErrorMessage = node2.InnerText;
73             if (!string.IsNullOrEmpty(c2.ErrorMessage))
74             {
75                 throw new Exception(c2.ErrorMessage);
76             }
77         }

```

There may be better ways of handling this in WCF but I don't want to get hung up on error handling.

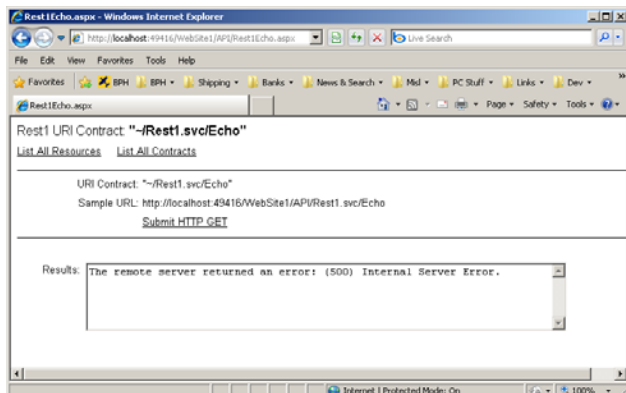
3.4 – Common Mistakes

The ServiceHost Factory is completely unforgiving! It is case sensitive and does not trim input strings. And many of the error messages are completely off topic and misleading. For example:

3.4.1 – Common Mistakes: Namespace Misspelling

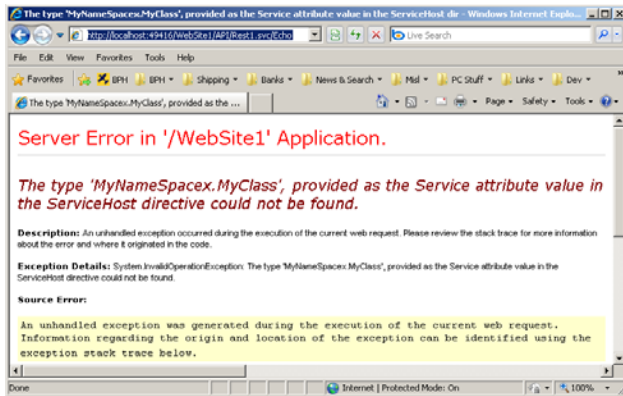
Notice the name space is spelled incorrectly in the Rest1.svc file:

```
<%@ ServiceHost Factory="System.ServiceModel.Activation.WebServiceHostFactory" Service="MyNameSpacex.MyClass" %>
```



The (500) Internal Server Error is tricky.

Sometimes, if you call the API directly you can get more information. Copy and paste the “Sample URL” into Internet Explorer:



“PC Load Letter” type message.

3.4.2 – Common Mistakes: Interface Class Space Error

Sometimes WCF stands for WTF! This one took me forever to figure out:

```
namespace Buzz
{
    [ServiceContract]
    public partial interface IBuzzSecure
    {
        [OperationContract]
        [WebGet(UriTemplate = "/Echo")]
        string CT_handler_Echo();

        [WebInvoke(UriTemplate = "/GroupListGetAll ", Method = "POST")]
        [OperationContract]
        List<Buzz.GroupObject> CT_handler_GroupListGetAll(Buzz.GroupListGetAllParms glgap);
    }
}
```

Be very
Careful!!

Again, I found it with the copying the URL to the browser to see the more detailed error message.

3.4.3 – Common Mistakes: Interface Class Specification Error

Here is another interface class error that can be tricky to find. Here's what the code should be:

```
23     [OperationContract]
24     [WebGet(UriTemplate = "/Multiply/{firstNumber}/{secondNumber}")]
25     string Multiply(string firstNumber, string secondNumber);
```

Here is the buggy version of the code:

```
23     [OperationContract]
24     [WebGet(UriTemplate = "/Multiply")]
25     string Multiply(string firstNumber, string secondNumber);
```

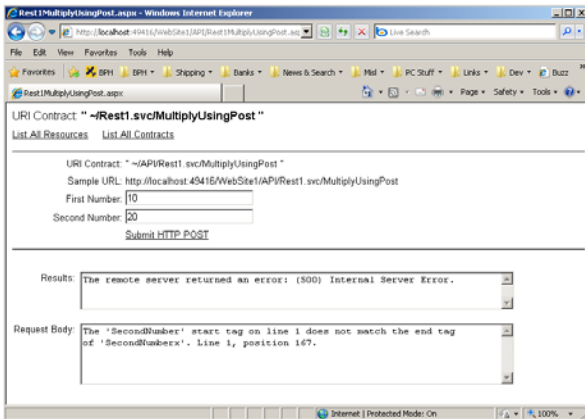
Again, this was caught by running the API directly in Internet Explorer. Make sure the parameter specification on line 24 matches exactly with the parameters on line 25.

3.4.4 – Common Mistakes: Well Formed XML

You can test your XML to make sure it is well formed by using the following:

```
74 reqBody.Text = Utils.IndentXMLString(sb.ToString());
```

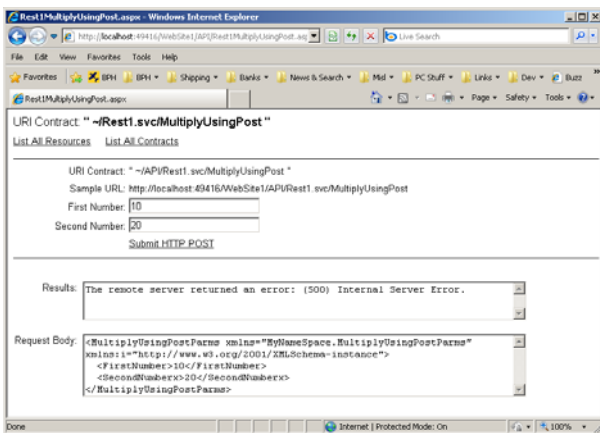
Here's the error you will see:



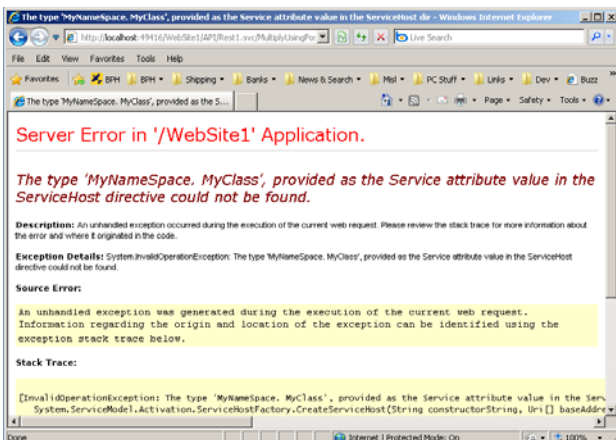
Attribute was named wrong in the request XML.

3.4.5 – Common Mistakes: Bad XML for Class Parameter

Attribute was named wrong in the request XML.

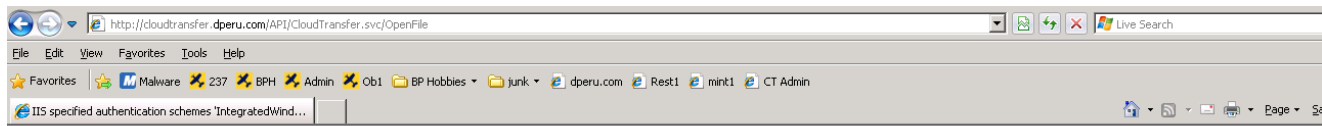


When you put the URL in the browser you get:



3.4.6 – Uncommon Mistakes: Works fine in dev but not at all in production (after publish)

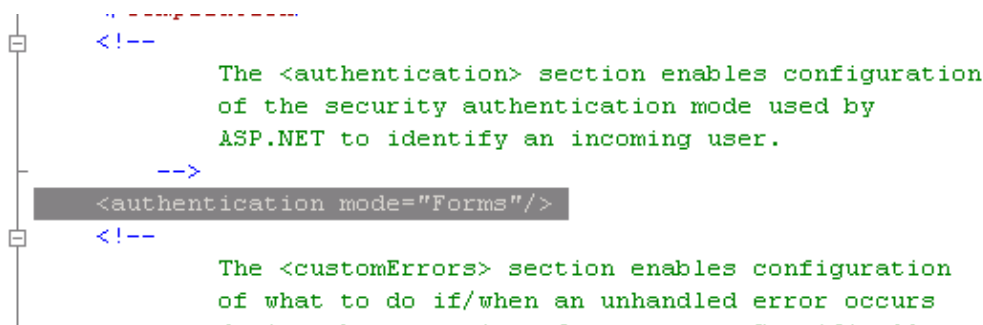
Here is another error that took me forever to figure out. After cutting-n-pasting the URL for the API into a web browser you see the following:



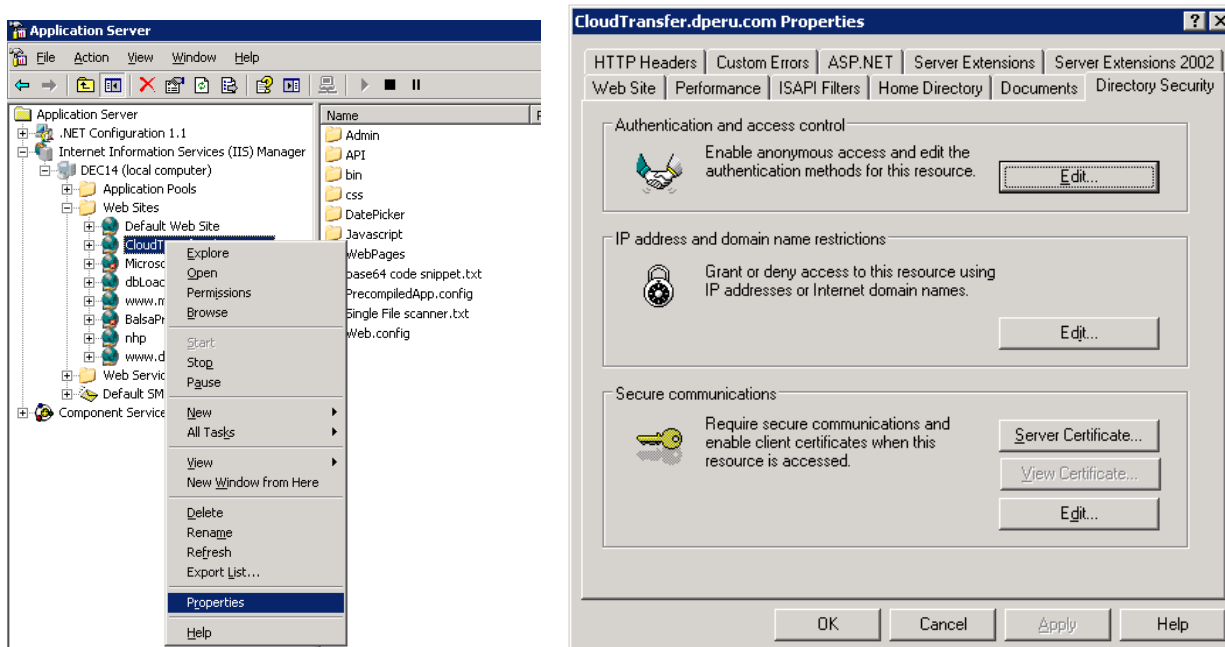
Server Error in '/' Application.

IIS specified authentication schemes 'IntegratedWindowsAuthentication, Anonymous', but the binding only supports specification of exactly one authentication scheme. Valid authentication schemes are Digest, Negotiate, NTLM, Basic, or Anonymous. Change the IIS settings so that only a single authentication scheme is used.

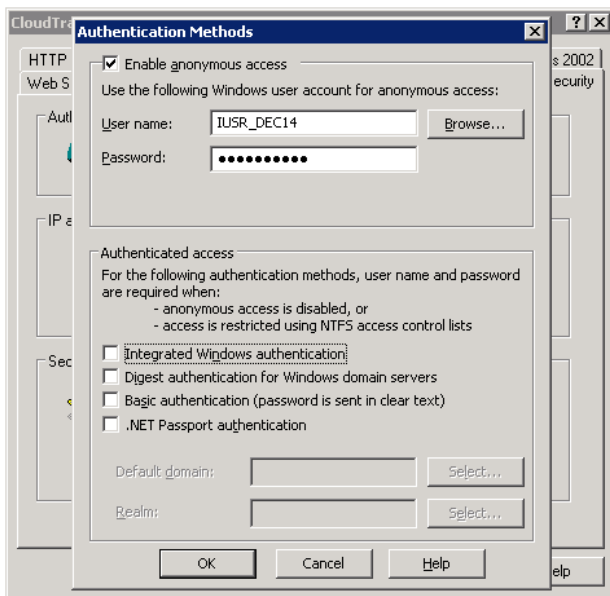
If you see the error above, make sure you are using “Forms” authentication in your web.config file:



In “InetMgr”, right click over the website and select properties from the popup menu. Go to the Security Tab:



Click on the “edit” button.



Uncheck the “Integrated Windows authentication” checkbox.

4.0 – Advanced Topics: Security (Authentication)

The first topic of security is authentication. Here is the most common way to authenticate a user with a RESTful Web Service. Just use HTTPS and pass in credentials with post parameters:

<https://www.domain.com/Secure/API/Rest1Secure.svc/Authenticate>

This API can then return back a class containing an error string and secret key value that can be use during the session. The secret key can then be used to encrypt POST parameters or sign text values in order to authenticate additional API calls without using HTTPS. Or, if you don't trust yourself, you can just use HTTPS and include user credentials with every API call.

4.1 – Advanced Topics: Security (Encryption)

For encryption, we are going to use classes in System.Security.Cryptography namespace to do AES encryption. Here is a quote from Bruce Schneier (last name rhymes with flyer) about AES encryption:

However, at the end of the AES process, Bruce Schneier, a developer of the competing algorithm Twofish, wrote that while he thought successful academic attacks on Rijndael would be developed someday, "I do not believe that anyone will ever discover an attack that will allow someone to read Rijndael traffic."

The pronunciation of Rijndael almost sounds like "Rhine dahl".

Bruce Schneier is a leading expert in cryptography and authored the book titled “Applied Cryptography” which has a great quote on the cover: “The book the National Security Agency wanted never to be published...”

<<Bruce Schneier comments on key management>>

Consider the following Crypto.cs class for doing encryption and decryption:

```
9  public class Crypto
10 {
35  /// <summary>
36  /// Use AES to encrypt data string. The output string is the encrypted bytes as a UTF8 encoded
37  /// base64 string. The same password must be used to decrypt the string.
38  /// </summary>
39  /// <param name="data">Clear string to encrypt.</param>
40  /// <param name="password">Password used to encrypt the string.</param>
41  /// <returns>Encrypted result as Base64 string.</returns>
42  public static string StringEncrypt(string data, string password)
43  {
44      if (data == null)
45          throw new ArgumentNullException("data");
46      if (password == null)
47          throw new ArgumentNullException("password");
48      byte[] encBytes = EncryptData(Encoding.UTF8.GetBytes(data), password, PaddingMode.ISO10126);
49      return Convert.ToBase64String(encBytes);
50  }
51
52  public static byte[] EncryptData(byte[] data, string password, PaddingMode paddingMode)
53  {
54      if (data == null || data.Length == 0)
55          throw new ArgumentNullException("data");
56      if (password == null)
57          throw new ArgumentNullException("password");
58      PasswordDeriveBytes pdb = new PasswordDeriveBytes(password, Encoding.UTF8.GetBytes("Salt"));
59      RijndaelManaged rm = new RijndaelManaged();
60      rm.Padding = paddingMode;
61      ICryptoTransform encryptor = rm.CreateEncryptor(pdb.GetBytes(16), pdb.GetBytes(16));
62      using (MemoryStream msEncrypt = new MemoryStream())
63      using (CryptoStream encStream = new CryptoStream(msEncrypt, encryptor, CryptoStreamMode.Write))
64      {
65          encStream.Write(data, 0, data.Length);
66          encStream.FlushFinalBlock();
67          return msEncrypt.ToArray();
68      }
69  }
70
71  /// <summary>
72  /// Decrypt the data string to the original string. The data must be the base64 string
73  /// returned from the EncryptData method.
74  /// </summary>
75  /// <param name="data">Encrypted data generated from EncryptData method.</param>
76  /// <param name="password">Password used to decrypt the string.</param>
77  /// <returns>Decrypted string.</returns>
78  public static string StringDecrypt(string data, string password)
79  {
80      if (data == null)
81          throw new ArgumentNullException("data");
82      if (password == null)
83          throw new ArgumentNullException("password");
84      byte[] encBytes = Convert.FromBase64String(data);
85      byte[] decBytes = DecryptData(encBytes, password, PaddingMode.ISO10126);
86      return Encoding.UTF8.GetString(decBytes);
87  }
88
89  public static byte[] DecryptData(byte[] data, string password, PaddingMode paddingMode)
90  {
91      if (data == null || data.Length == 0)
92          throw new ArgumentNullException("data");
93      if (password == null)
94          throw new ArgumentNullException("password");
95      PasswordDeriveBytes pdb = new PasswordDeriveBytes(password, Encoding.UTF8.GetBytes("Salt"));
96      RijndaelManaged rm = new RijndaelManaged();
```

```

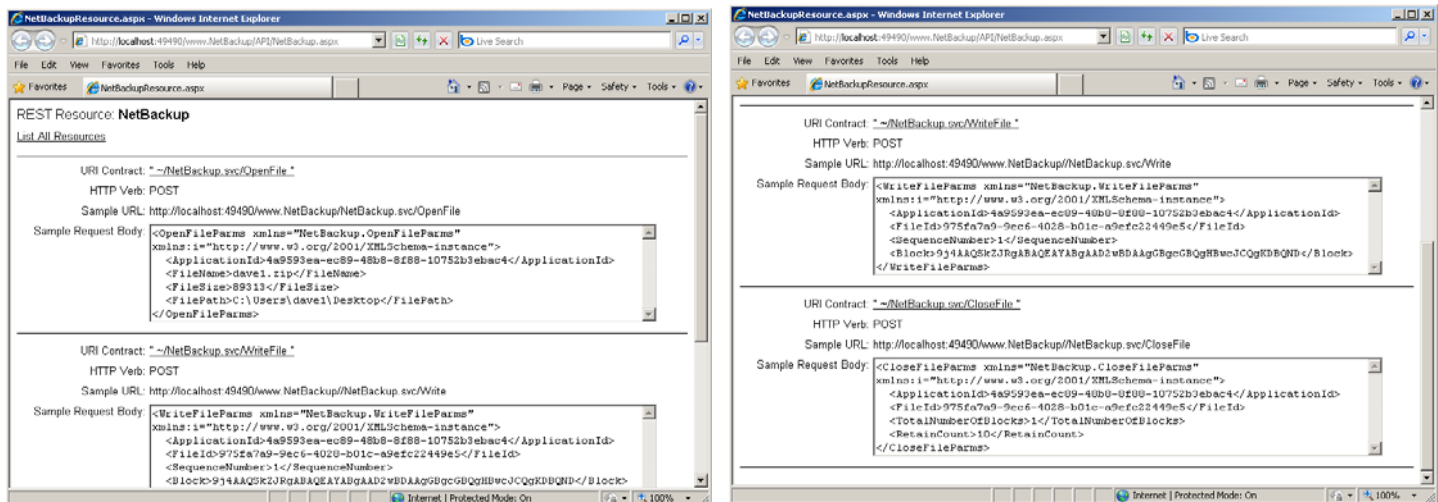
97     rm.Padding = paddingMode;
98     ICryptoTransform decryptor = rm.CreateDecryptor(pdb.GetBytes(16), pdb.GetBytes(16));
99     using (MemoryStream msDecrypt = new MemoryStream(data))
100     using (CryptoStream csDecrypt = new CryptoStream(msDecrypt, decryptor, CryptoStreamMode.Read))
101     {
102         // Decrypted bytes will always be less then encrypted bytes, so len of encrypted data will be big enough for buffer.
103         byte[] fromEncrypt = new byte[data.Length];
104         // Read as many bytes as possible.
105         int read = csDecrypt.Read(fromEncrypt, 0, fromEncrypt.Length);
106         if (read < fromEncrypt.Length)
107         {
108             // Return a byte array of proper size.
109             byte[] clearBytes = new byte[read];
110             Buffer.BlockCopy(fromEncrypt, 0, clearBytes, 0, read);
111             return clearBytes;
112         }
113         return fromEncrypt;
114     }
115 }
116 }

```

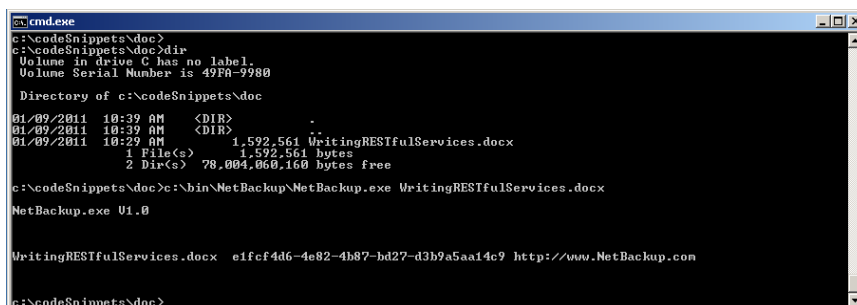
<< Tribeca Story>>

4.3 – Advanced Topics: Working with Binary Data

Consider the following API used to move a binary file to another machine:



The client program is a console application. Here's a screen shot of the console app in action:



Here is a partial listing of the console application:

```

64     {
65         NBU.NetBackupUtility.DoNetBackup(app_id, url, pathFileName, retain_copy_count);
66     }
67     catch (Exception ex)
68     {
69         Console.WriteLine("Error: " + ex.Message + "\n\n");
70     }

137 public static void DoNetBackup(string appld, string serverUrl, string pathFileName, string retainCount)
138 {
139     // Get file length info:
140     //
141     System.IO.FileInfo fileInfo = new System.IO.FileInfo(pathFileName);
142     long totalNumberOfBytes = fileInfo.Length;
143     string currentFilePath = fileInfo.DirectoryName;
144     string currentFileName = fileInfo.Name;
145
146     // Open local file:
147     //
148     FileStream fileStream = new FileStream(pathFileName, FileMode.Open, FileAccess.Read);
149
150     // Initialize read-buffer:
151     //
152     int blockSize = 10000;
153     byte[] buffer = new byte[blockSize];
154     long totalNumberOfBlocks = 0;
155     int currentByteCount = 0;
156
157     // Do REST call to open file:
158     //
159     string fileId = OpenFileAtServer(serverUrl, appld, currentFileName, currentFilePath, totalNumberOfBytes.ToString());
160
161     // loop through reading blocks
162     while (true)
163     {
164         int count = fileStream.Read(buffer, 0, blockSize);
165         if (count <= 0)
166         {
167             break;
168         }
169         currentByteCount += count;
170         ++totalNumberOfBlocks;
171
172         string base64EncodedString = "";
173         if (count < blockSize)
174         {
175             byte[] buffer2 = new byte[count];
176             for (int i = 0; i < count; ++i)
177             {
178                 buffer2[i] = buffer[i];
179             }
180             base64EncodedString = System.Convert.ToBase64String(buffer2);
181         }
182         else
183         {
184             base64EncodedString = System.Convert.ToBase64String(buffer);
185         }
186
187         // Queue up thread pool:
188         //
189         // TODO, replace single thread below
190
191         // Do REST call to write file:
192         //
193         WriteFileAtServer(serverUrl, appld, fileId, totalNumberOfBlocks.ToString(), base64EncodedString);
194     }
195
196     // Close remote file:

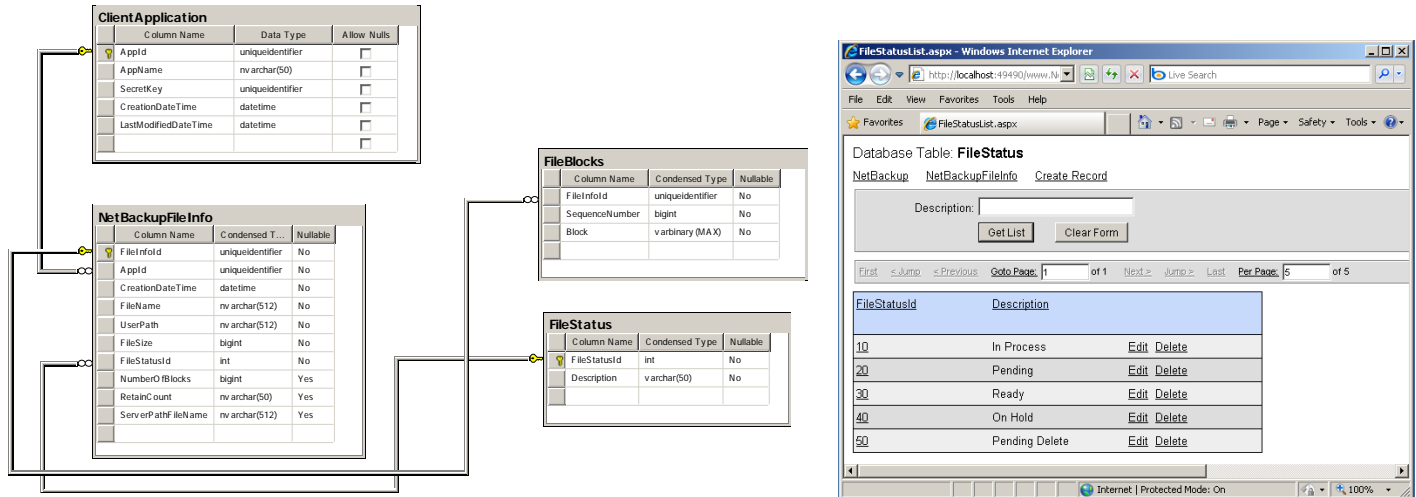
```

```

197      //
198      CloseFileAtServer(serverUrl, applId, fileId, totalNumberOfBlocks.ToString(), retainCount);
199
200  }

```

The file blocks are stored in a database:



Here's the API call that will write the Base64 data block to a varbinary in a database record:

```

84  [DataContract(Name = "WriteFileParms", Namespace = "NetBackup.WriteFileParms")]
85  public class WriteFileParms
86  {

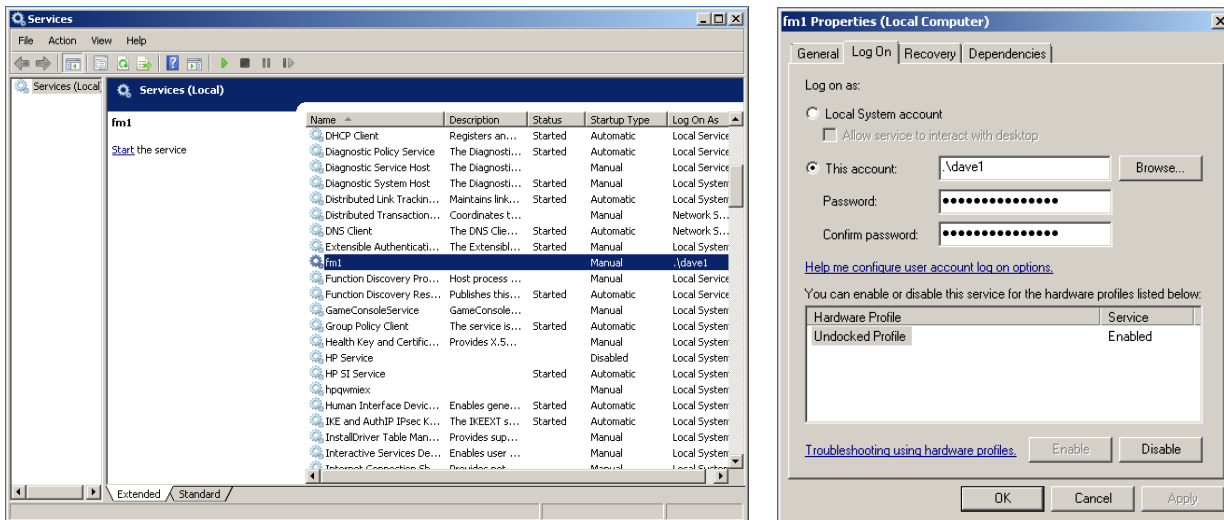
108      [DataMember(Name = "Block", Order = 4)]
109      private byte[] _Block;
110      public byte[] Block           // Base64 encoding automatically converted to Byte[] value by WCF.
111      {
112          get { return _Block; }
113          set { _Block = value; }
114      }
115  }

206      public string NB_Handler_WriteFile(WriteFileParms wfp)
207      {
208          try
209          {

233              // Create new data block record in database:
234              //
235              FileBlocks dataObj = new FileBlocks();
236              dataObj.GenerateAllIds();
237              dataObj.FileInfolId = new Guid(wfp.FileId);
238              dataObj.SequenceNumber = long.Parse(wfp.SequenceNumber);
239              dataObj.Block = wfp.Block;
240              dataObj.BlockByteSize = wfp.Block.Length;
241              FileBlocksBLL fileBlocksBLL = new FileBlocksBLL();
242              fileBlocksBLL.CreateRecord(dataObj);           // Mintiera middle tier data access component.
243          }
244          catch (Exception ex)
245          {
246              return ex.Message;
247          }
248          return "";    // Empty string means no error.
249      }

```

Once files are in the “Pending” state, a Windows Service (agent, daemon, or TSR) will take the file from the database and write it to the file system. File maker one (fm1):



Once the agent creates a new file in the file system, it will transition the file state in the database from “Pending” to “Ready”. We still want to control access to the file. Here is a partial listing of the ASPX code that actually serves the file:

```

24 public partial class Admin_NetBackupFileInfoGetFile : System.Web.UI.Page
25 {
26     protected void Page_Load(object sender, EventArgs e)
27     {
28         Response.ClearHeaders();
29         Response.ClearContent();
30
31         try
32         {
33             // Get Id parameter:
34             //
35             string CurrentTableId = null;
36             if (Request.QueryString["TableId"] == "" || Request.QueryString["TableId"] == null)
37                 CurrentTableId = Request.QueryString["GUID"]; // Not supported.
38             else
39                 CurrentTableId = Request.QueryString["TableId"];
40             if (CurrentTableId == "" || CurrentTableId == null)
41             {
42                 Response.ContentType = "text/plain";
43                 Response.Write("ERROR: Id not set in query string.");
44                 return;
45             }
46
47             // Get NetBackFileInfo record:
48             //
49             NetBackupFileInfoBLL fiBLL = new NetBackupFileInfoBLL(); // Mintiera middle tier data access component.
50             NetBackupFileInfo fi = fiBLL.Get(CurrentTableId);
51
52             if (fi.FileStatusId != (int)NetBackup.DAL.FileStatus.Enum.Ready)
53             {
54                 Response.ContentType = "text/plain";
55                 Response.Write("ERROR: File not ready.");
56                 return;
57             }
58
59             // Write out HTTP headers:
60             //
61             Response.ContentType = "application/octet-stream";

```

```

62      Response.AppendHeader("content-disposition", "attachment; filename=" + fi.FileName);
63
64  #if useBlockRead
65
66      SqlConnection conn = null;
67      try
68      {
69          conn = new SqlConnection(NetBackupFileInfo.ConnectionString);
70          conn.Open();
71
72          SqlCommand cmd = new SqlCommand();
73          cmd.Connection = conn;
74          cmd.CommandType = CommandType.StoredProcedure;
75          cmd.CommandText = "FileBlocks_Get_All_Blocks";
76          cmd.Parameters.AddWithValue("@fileInfold", fi.FileInfold);
77          SqlDataReader reader = cmd.ExecuteReader();
78          while (reader.Read())
79          {
80              if (reader["Block"] != DBNull.Value)
81              {
82                  byte[] ba = (Byte[])reader["Block"];
83                  Response.BinaryWrite(ba);                // This code did NOT work for files greater than 500 megabytes.
84              }
85          }
86      }
87      catch (Exception ex)
88      {
89          throw ex;
90      }
91      finally
92      {
93          if (conn != null)
94              conn.Close();
95      }
96  #else
97      // Write file to HTTP output stream:
98      //
100     Response.TransmitFile(fi.ServerPathFileName);
101
102     //      Response.WriteFile(fi.ServerPathFileName);    // This does not show Save As dialog box on large files.
103 #endif
104
105
106     }
107     catch (Exception ex)
108     {
109         Response.ContentType = "text/plain";
110         Response.Write(ex.Message);
111         return;
112     }
113
114 }
115 }

```

This method of doing binary uploads scales nicely because the RESTful Service APIs are house in IIS. This takes advantage of IIS thread pooling and a Network Load Balancer (NLB) can be used. The files can then be collated using a Windows Service agent non real-time and then served by IIS.

Any Questions?

All the materials from this presentation will be available on www.dperu.com.

If you have any issues or questions about getting the materials you can send me an email.