Web Content

Web posting date	7/15/02
Magazine home page or URL	http://www.fawcette.com/vsm/2002_07/online/csharp_jlowy_07_15_02/
DART tag	
Main file name	LowyRemoting_t5.doc
Locator+ code(s)	vsep020715cs_t
Accompanying ZIP file name	vsep020715cs.zip
Listing file name	
Sidebar file name	
Table file name	
Screen capture file names	Figure 1.bmp through Figure 4.bmp
Infographic/illustration file names	
Photos or book scans	
Special instructions for Art dept.	
Editor	NG
Status	first pass (_t2); juval's additions (_t3); NG cleanup (_t5)
Spellchecked (set Language to English U.S.)	
300- to 315-character blurb (for home page lead, can also add 50-character bullet items)	
180-character blurb (for newsletters and left column of home pages)	With three activation modes and multiple ways of creating remote objects, it's no wonder .NET remoting baffles many developers. Download Juval Löwy's sample app to experiment with the different scenarios.
90-character blurb (for right column of home pages)	

TITLE TAG & METATAGS

<title>**Fawcette.com – Get a Handle on .NET Remoting**</title>
<!-- Start META Tags -->
<meta name="Category" content="**.NET**">
<meta name="Subcategory" content="**C#**">
<meta name="Keywords" content=".NET, C#, remoting, host, client, singleton, single call, TCP, HTTP, channels">
<meta name="DESCRIPTION" content="With three activation modes and three ways to create remote objects, it's no wonder .NET remoting baffles many developers. Download Juval Löwy's sample app to experiment with the different scenarios.">
<meta name="Author" content="Juval Löwy">
<meta name="Issue" content="**July 15, 2002**">
<!-- End META Tags -->

Head (38-character maximum):

Get a Handle on .NET Remoting

Deck:

Download a sample app to experiment with .NET remoting permutations.

Byline:

by Juval Lowy

Dateline:

Posted July 15, 2002

It's no wonder that .NET remoting baffles many developers. A remote application, which can accept calls in binary or SOAP format on either TCP or HTTP channels, must register the objects it exports and the channels it's willing to accept calls on. Plus, a client application must register the objects it wishes to access remotely and indicate the activation channel. In addition, you have three activation modes and multiple ways of creating remote objects.

Remote objects can be either client-activated or server-activated, and the server-activated objects can be either singleton objects or single-call objects. Client-activated objects are activated similarly to the way objects are activated in the classic client/server activation model: When a client creates an object, it gets a dedicated object. A singleton server-activated object is shared by all clients, whereas a single-call object is newly activated with each client and disposed of at the end of the call. The client decides whether it needs a client- or a server-activated object, and the server decide whether it will activate a singleton object or single-call object. The client and server must agree on the transport protocol (TCP or HTTP), the port to communicate on, and the communication format (SOAP or binary).

You can either configure these details programmatically (with **the ability to perform** runtime decisions **for** maximum flexibility[[JL: not sure what you mean by the text in these parentheses—NG]]), **you can** use a configuration file at load time, or do a combination of the two. Download the sample app[[link to code zip on download page]] for a comprehensive demo of a remote application and a client. The sample demonstrates

the various programmatic permutations and scenarios using a simple user interface, as well as how the host application and the client can easily share the server metadata. In a future download, I will provide the same demo application using configuration files instead of programmatic calls.

Open the RemotingDemo.sln solution. The solution contains a host server EXE assembly, a DLL class library, and a client EXE assembly (see Figure 1). The client requires the class library metadata to compile, so it has a reference to the Server assembly. Similarly, the host application requires the server metadata to register the remote object, so it too has a reference to the Server assembly. The server object displays its hosting app domain name in a message box, as well as the value of a counter, to indicate singleton/single-call state:

```
public void Count()
{
   Counter++;
   string appName =
AppDomain.CurrentDomain.FriendlyName;
   MessageBox.Show("Counter value is " +
Counter.ToString(),appName);
}
```

Set the client as the startup project, and run it (see Figure 2). Without registering, click on the "new" button. The client creates the object and calls the Count() method twice:

```
private void OnNew(object sender,EventArgs e)
{
   MyServer obj;
   obj = new MyServer();
   obj.Count();
   obj.Count();
}
```

The server object is activated locally, in the client's app domain (see Figure 3). The client's Main() method registers these channels:

```
static void Main()
{
   //No port number as parameter.
   //Input port is in activation URL
   IChannel tcpChannel = new TcpChannel();
   //Done only once per app domain:
   ChannelServices.RegisterChannel(tcpChannel);

   IChannel httpChannel = new HttpChannel();
   //Done only once per app domain:
   ChannelServices.RegisterChannel(httpChannel);

   Application.Run(new ClientForm());
}
```

When you click on Register, the client registers the server object as a remote object:

```
private void OnRegister(object sender,EventArgs
e)
{
   Type serverType =
```

```
typeof(RemoteServer.MyServer);
   string url = GetActivationURL();

   if(radioServer.Checked)
   {
      RemotingConfiguration.
RegisterWellKnownClientType(serverType,url);

      //Enable GetObject() if in server mode
      GetObjectButton.Enabled = true;
   }
   else //Client activation mode
   {
      //Register just once:
      RemotingConfiguration.
RegisterActivatedClientType(serverType,url);
   }
}
```

The client registers the server activation mode based on the Activation Mode radio button. Look at the code for GetActivationURL(): It generates the object location (URL) based on the channel selection radio button and the object activation mode. Run the client again, and register the object as remote. Try creating the object using new, CreateInstance(), and GetObject(). This time, all activation methods should fail. You need to have a remote host running, and have the host register the objects it wants to export. Launch the RemoteServerHost application. Select server activation mode, and register the object (see Figure 4).

The RemoteServerHost Main() method registers the channels (both TCP and HTTP) on which the host application will accept calls to the object:

```
static void Main()
{
   //Must register at least one channel
   //Registering TCP channel
   IChannel tcpChannel = new TcpChannel(8005);
   ChannelServices.RegisterChannel(tcpChannel);

   //Registering http channel
   IChannel httpChannel = new HttpChannel(8006);
   ChannelServices.RegisterChannel(httpChannel);

Application.Run(new ServerHostDialog());
}
```

The code for the host OnRegister() method decides on the server activation mode (singleton or single call) based on the Activation Mode radio button:

```
private void OnRegister(object sender,EventArgs
e)
{
   Type serverType;
   serverType = typeof(RemoteServer.MyServer);

   //Allow client activation:
   RemotingConfiguration.
RegisterActivatedServiceType(serverType);
```

```
   //Server activation: choose mode
   if(radioSingleCall.Checked)
   {
      //Allow Server activation, single mode:
      RemotingConfiguration.
RegisterWellKnownServiceType(serverType,
"CounterServer",WellKnownObjectMode.SingleCall);
   }
   else
   {
      //Allow Server activation, singleton mode:
      RemotingConfiguration.
RegisterWellKnownServiceType(serverType,
"CounterServer",WellKnownObjectMode.Singleton);
   }
}
```

Run the host application, and click on Register Objects. Right-click on the client project, select Debug | Start new instance. Once the client is running, click on Register. According to the activation mode, the counter value will increment or indicate a new object each time. Experiment with the different permutations: new, CreateIntance(), and GetObject() as ways of creating client-activated, singleton, and single-call objects. Perform the experiments both with TCP and HTTP channels, and with or without client registration.

<!--#include virtual="/includes/authorbios/jlowy.inc"-->

Captions & tags:

Figure 1: **Study the Solution**
The RemotingDemo solution contains a host server EXE assembly, a DLL class library, and a client EXE assembly.

<title>**Fawcette.com – Get a Handle on .NET Remoting – Figure 1**</title>
<!-- Start META Tags -->
<meta name="Category" content="**.NET**">
<meta name="Subcategory" content="**C#**">
<meta name="Keywords" content=".NET, C#, remoting, host, client, singleton, single call, TCP, HTTP, channels">
<meta name="DESCRIPTION" content="The RemotingDemo solution contains a host server EXE assembly, a DLL class library, and a client EXE assembly.">
<meta name="Author" content="Juval Löwy">
<meta name="Issue" content="**July 15, 2002**">
<!-- End META Tags -->

Figure 2: **Run the Client Project**
You can invoke the call locally or remotely by registering the server object. You need to select a channel and choose a way of creating the object.

<title>**Fawcette.com – Get a Handle on .NET Remoting – Figure 2**</title>
<!-- Start META Tags -->

<meta name="Category" content="**.NET**">
<meta name="Subcategory" content="**C#**">
<meta name="Keywords" content=".NET, C#, remoting, host, client, singleton, single call, TCP, HTTP, channels">
<meta name="DESCRIPTION" content="You can invoke the call locally or remotely by registering the server object. You need to select a channel and choose a way of creating the object.">
<meta name="Author" content="Juval Löwy">
<meta name="Issue" content="**July 15, 2002**">
<!-- End META Tags -->

Figure 3: **Activate the Server Object Locally**
When used locally, the server is in the client's app domain (Client.exe). The object maintains state, and the constructor is called when the client creates the object.

<title>**Fawcette.com – Get a Handle on .NET Remoting – Figure 3**</title>
<!-- Start META Tags -->
<meta name="Category" content="**.NET**">
<meta name="Subcategory" content="**C#**">
<meta name="Keywords" content=".NET, C#, remoting, host, client, singleton, single call, TCP, HTTP, channels">
<meta name="DESCRIPTION" content="When used locally, the server is in the client's app domain (Client.exe). The object maintains state, and the constructor is called when the client creates the object.">
<meta name="Author" content="Juval Löwy">
<meta name="Issue" content="**July 15, 2002**">
<!-- End META Tags -->

Figure 4: **The Server Host Application**
You need to select server activation mode, then register the object.

<title>**Fawcette.com – Get a Handle on .NET Remoting – Figure 4**</title>
<!-- Start META Tags -->
<meta name="Category" content="**.NET**">
<meta name="Subcategory" content="**C#**">
<meta name="Keywords" content=".NET, C#, remoting, host, client, singleton, single call, TCP, HTTP, channels">
<meta name="DESCRIPTION" content="Here's the server host application. You need to select server activation mode, then register the object.">
<meta name="Author" content="Juval Löwy">
<meta name="Issue" content="**July 15, 2002**">
<!-- End META Tags -->