Tech Note

embarcadero

DataSnap Client Development with RadPHP

José León

November 2010

Americas Headquarters
100 California Street, 12th Floor
San Francisco, California 94111

EMEA Headquarters
York House
18 York Road
Maidenhead, Berkshire
SL6 1SF, United Kingdom

Asia-Pacific Headquarters
L7. 313 La Trobe Street
Melbourne VIC 3000
Australia

INTRODUCTION

DataSnap® technology in Embarcadero's RAD development tools enables developers to easily build multi-tier solutions with native servers built in Delphi® and C++Builder® and connect to them from multiple client types including Windows, .NET and now PHP and JavaScript.

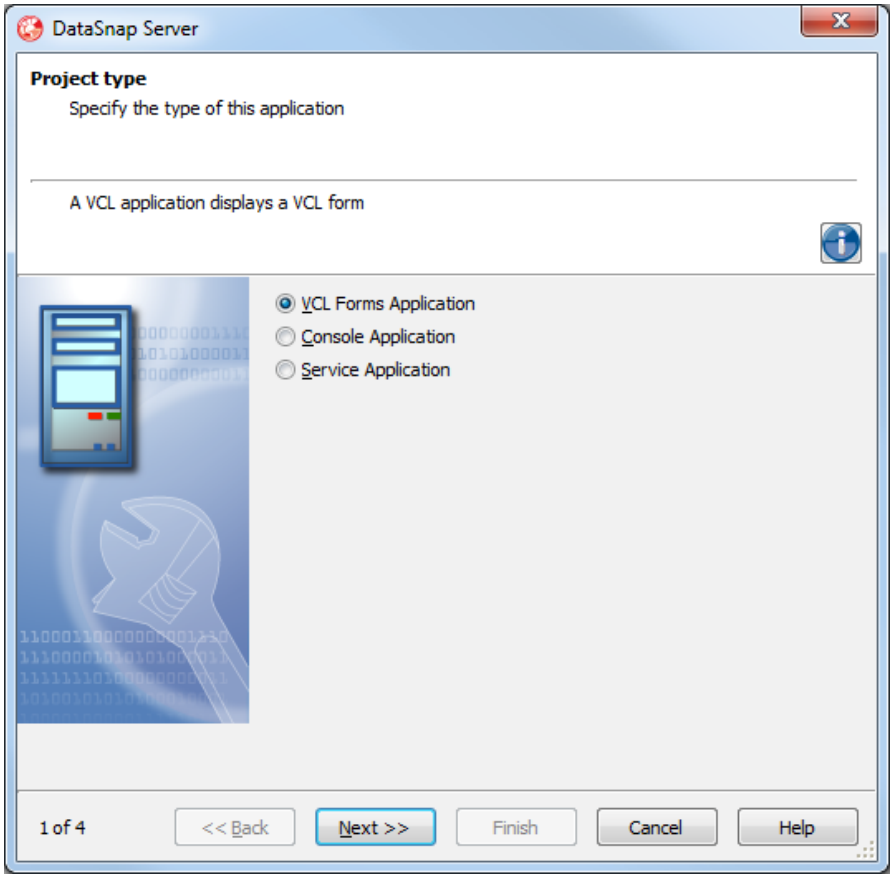
The goal of this document is to show you how to develop a simple DataSnap server with Delphi for Windows, and how to access that server using RadPHP™ XE. You will see how to access DataSnap servers using PHP and JavaScript.

DEVELOPING THE SERVER

Using File | New | Other… menu option, on the DataSnap server category, there is an item called "DataSnap Server", select it and press OK, and the IDE will start a wizard that will request information about the server you want to create.

PROJECT TYPE

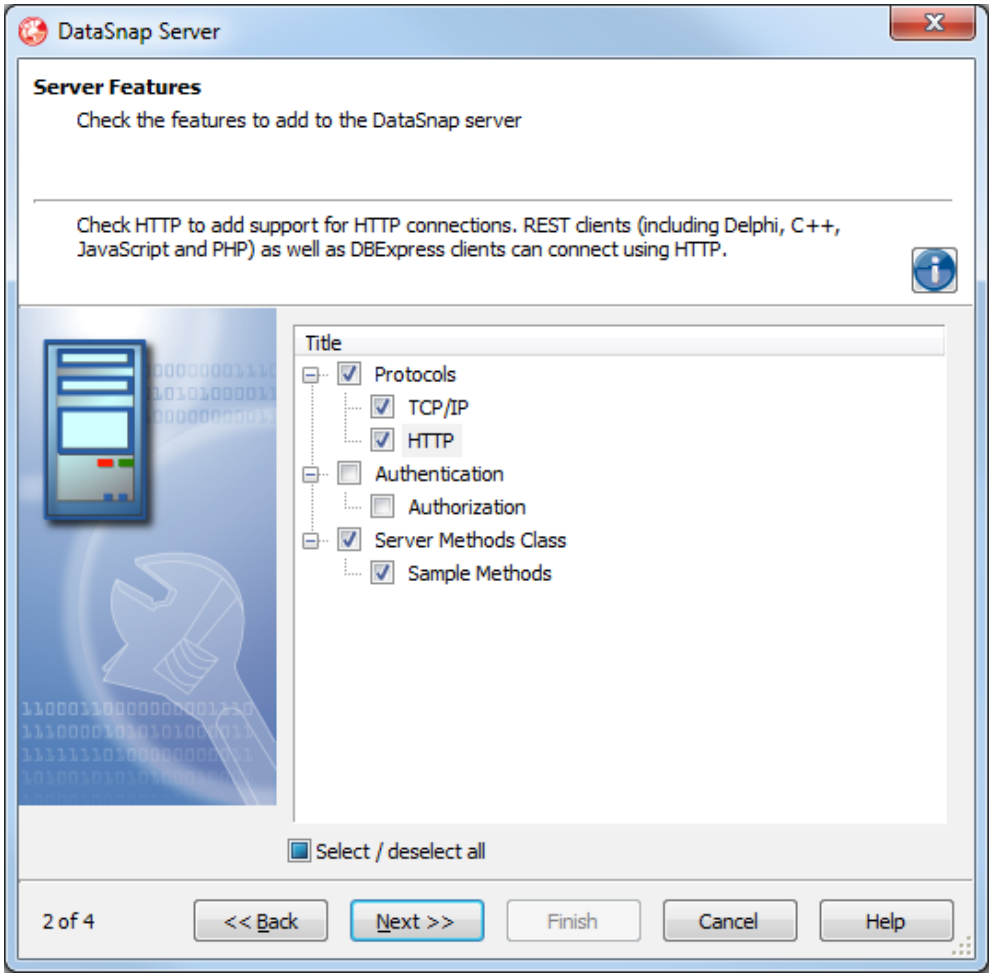
This setting allows you to select which type of project you want to create. Select "VCL Forms Application" if your application is going to be visual. If not, you should choose a Console Application or an application that will become an operating system service. For this sample, we choose VCL Forms Application.



SERVER FEATURES

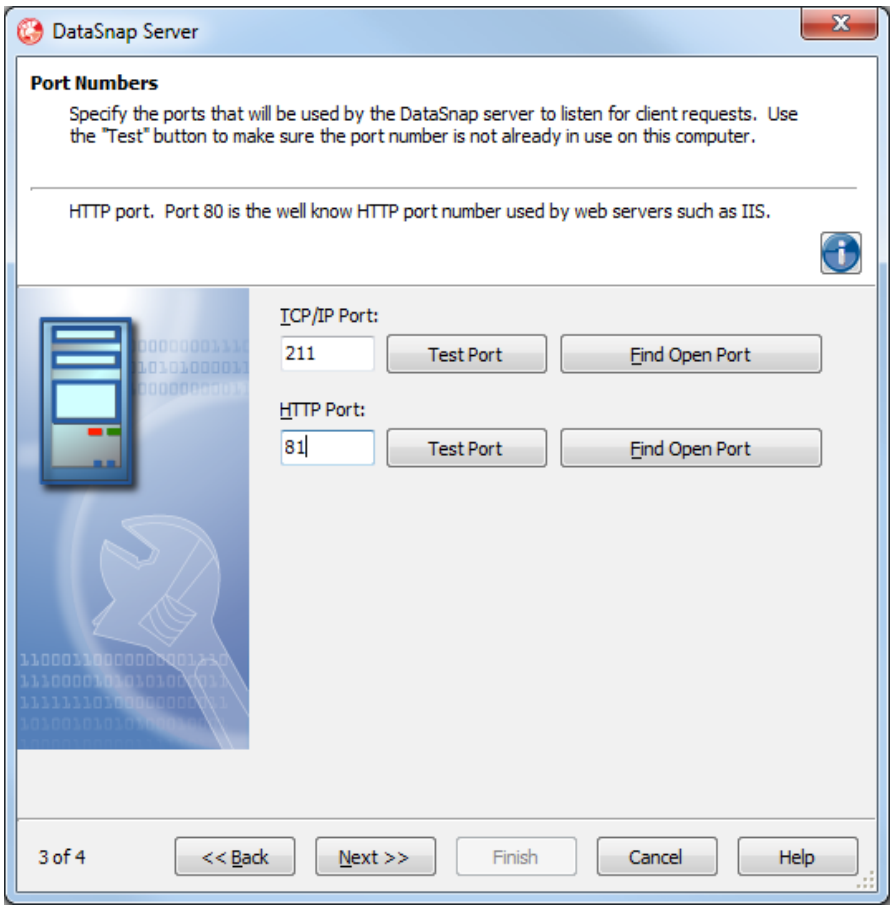
On this wizard step, we are asked about the features we want the wizard to add to the server. If any feature is not selected here at this time, it can still be added later.

For this sample, we select TCP/IP and HTTP protocols.



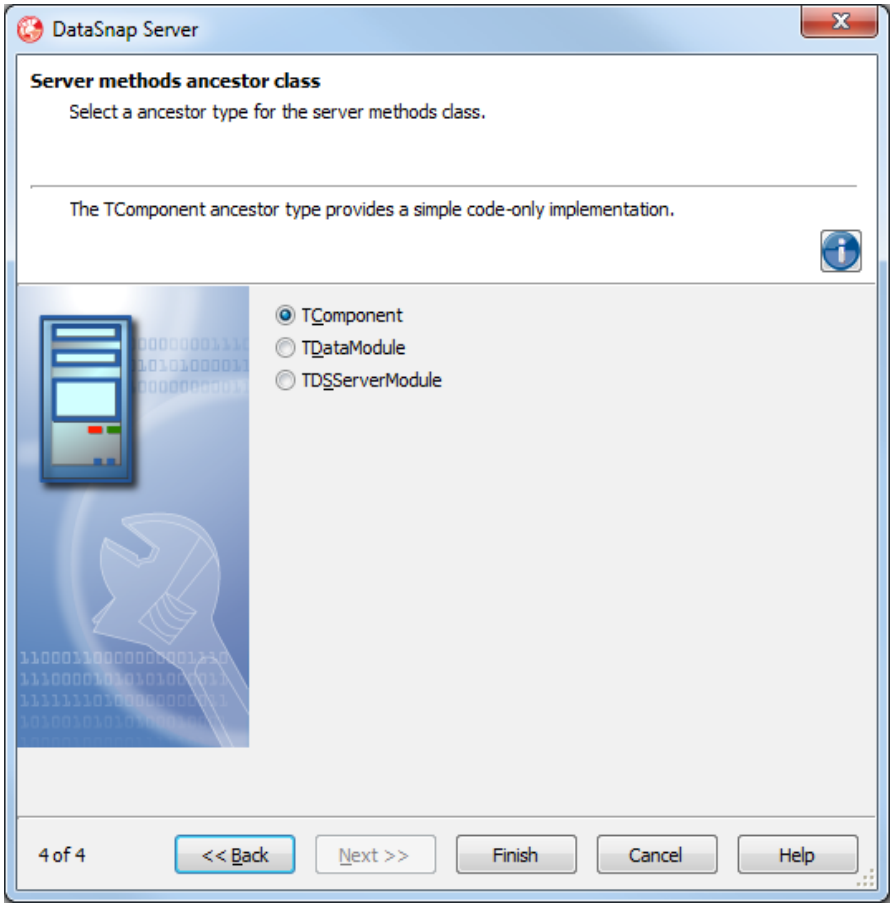
PORTS

Now we have to select on which ports our server is going to use. We need to select two ports – one for TCP/IP and another one for HTTP. In this example, my selections are 211 for TCP/IP and 81 for HTTP.



SERVER METHODS ANCESTOR CLASS

As we used the wizard to create some sample server methods, we need to select the ancestor for the class where these methods are going to be generated. We can use TComponent, TDataModule or TDSServerModule. For this sample, we use TComponent.



SERVER PARTS

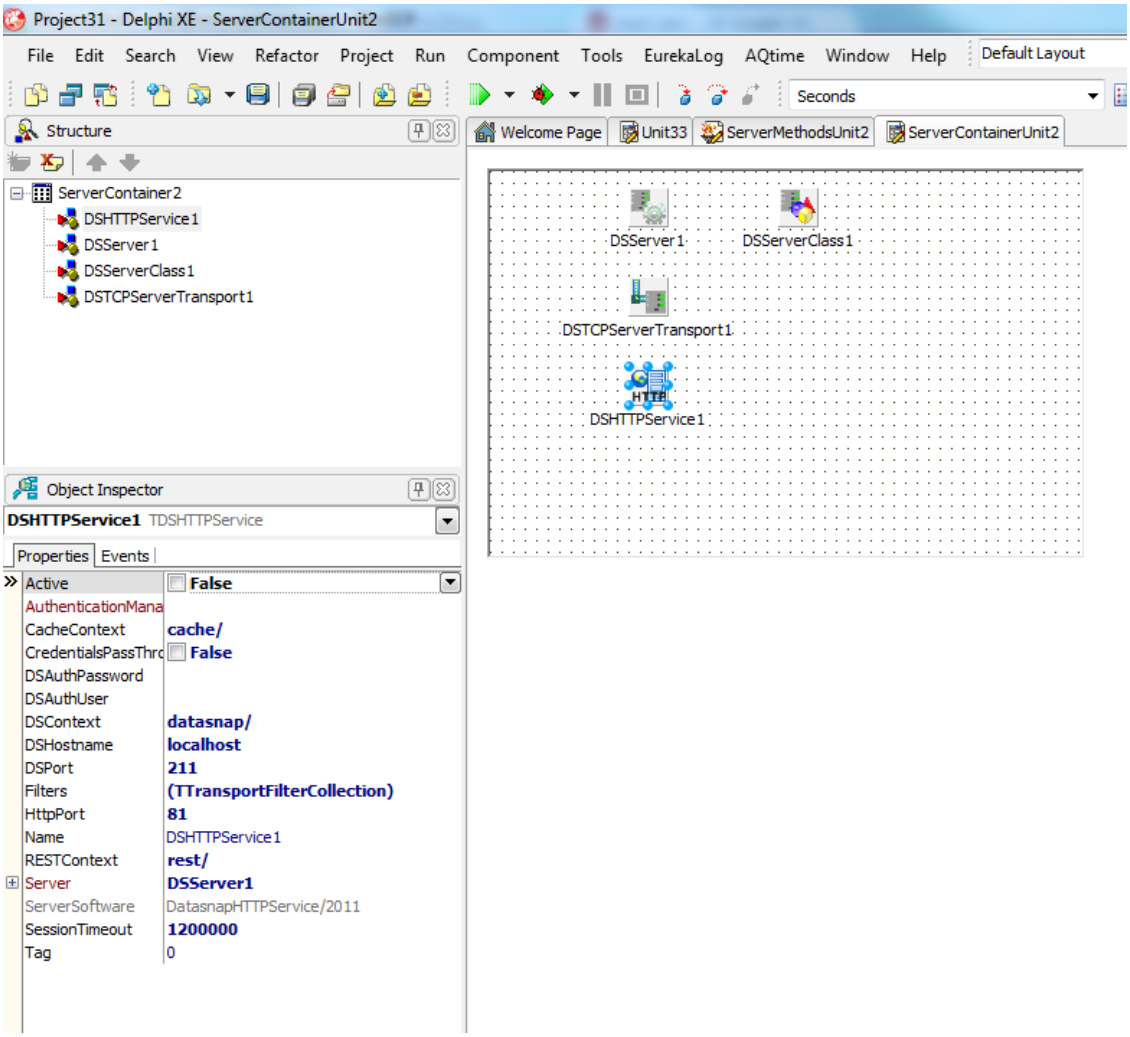
The code generated by the wizard is built in several parts – the main visual form, the Container, and the class holding the methods.

Server Methods Unit

```
1. unit ServerMethodsUnit2;
2.
3. interface
4.
5. uses
6.   SysUtils, Classes, DSServer;
7.
8. type
9. {$METHODINFO ON}
10.     TServerMethods2 = class(TComponent)
11.     private
12.       { Private declarations }
13.     public
14.       { Public declarations }
15.       function EchoString(Value: string): string;
16.       function ReverseString(Value: string): string;
17.     end;
18.   {$METHODINFO OFF}
19.
20.   implementation
21.
22.
23.   uses StrUtils;
24.
25.   function TServerMethods2.EchoString(Value: string): string;
26.   begin
27.     Result := Value;
28.   end;
29.
30.   function TServerMethods2.ReverseString(Value: string): string;
31.   begin
32.     Result := StrUtils.ReverseString(Value);
33.   end;
34.   end.
35.
```

The sample methods added by the wizard, EchoString and ReverseString perform basic operations on the input values and return the result. Here is where you need to start adding your functionality. Notice the METHODINFO directive that makes the compiler to generate extra RTTI information for the methods of that class, which is needed by the DataSnap server.

CONTAINER



The Container is a DataModule holding all non-visual components for the server and the transports. A TDSServer, a TDSTCPServerTransport, a TDSHTTPService and a TDSServerClass.

The DSServerClass1 is the component you need to specify the server which class contains the methods you want to publish. To do that, generate the OnGetClass event and set the PersistentClass variable to the class that holds your methods.

```
1. procedure TServerContainer2.DSServerClass1GetClass(
2.   DSServerClass: TDSServerClass; var PersistentClass: TPersistentClass);
3. begin
4.   PersistentClass := ServerMethodsUnit2.TServerMethods2;
5. end;
```

RUNNING THE SERVER

We decided this application was a VCL Forms based application, so to run it, it's just a matter of clicking Run in the IDE and that's it. The server will open when the application runs. The first time the application runs, you will be asked by the Windows Firewall, because the server is trying to open the ports for communication.

Allow access to the network and that's all you need to do.

DEVELOPING THE CLIENT

To develop the client, we are going to use RadPHP XE. The goal is to create a webpage, which calls the server methods provided by the Delphi DataSnap server and to show the results.

The first step is to create a new RPCL application, so File | New | RPCL Application will create it. After that, just save everything into a new folder.

USING THE CLIENT WIZARD

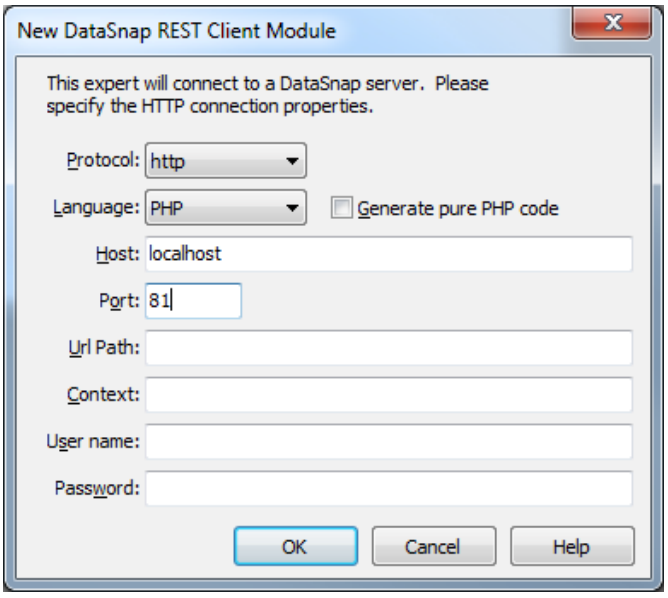
To use the client wizard, be sure the server is running, as it is required to connect to it to get a description of the methods it's serving.

To create the PHP code we need to access the DataSnap server. RadPHP XE includes a wizard to do the job. Use File | New | Other… to get the New Items dialog, and select the PHP Files category. The item we are looking for is the DataSnap REST Client Module. If you execute it, you get a dialog asking you for information.

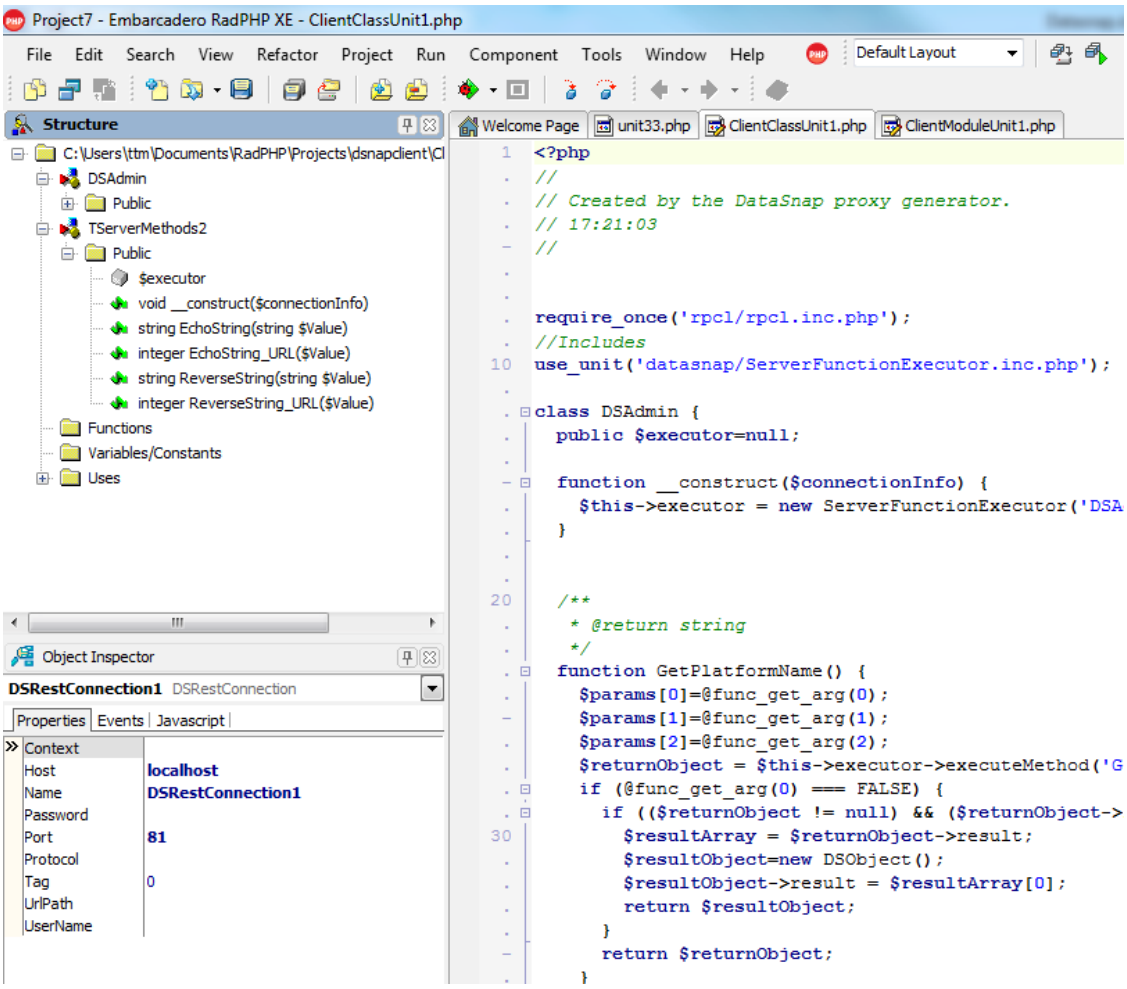
The selections for this dialog are:

- Protocol: http
- Language: PHP
- Host: localhost
- Port: 81

Here's how it looks:



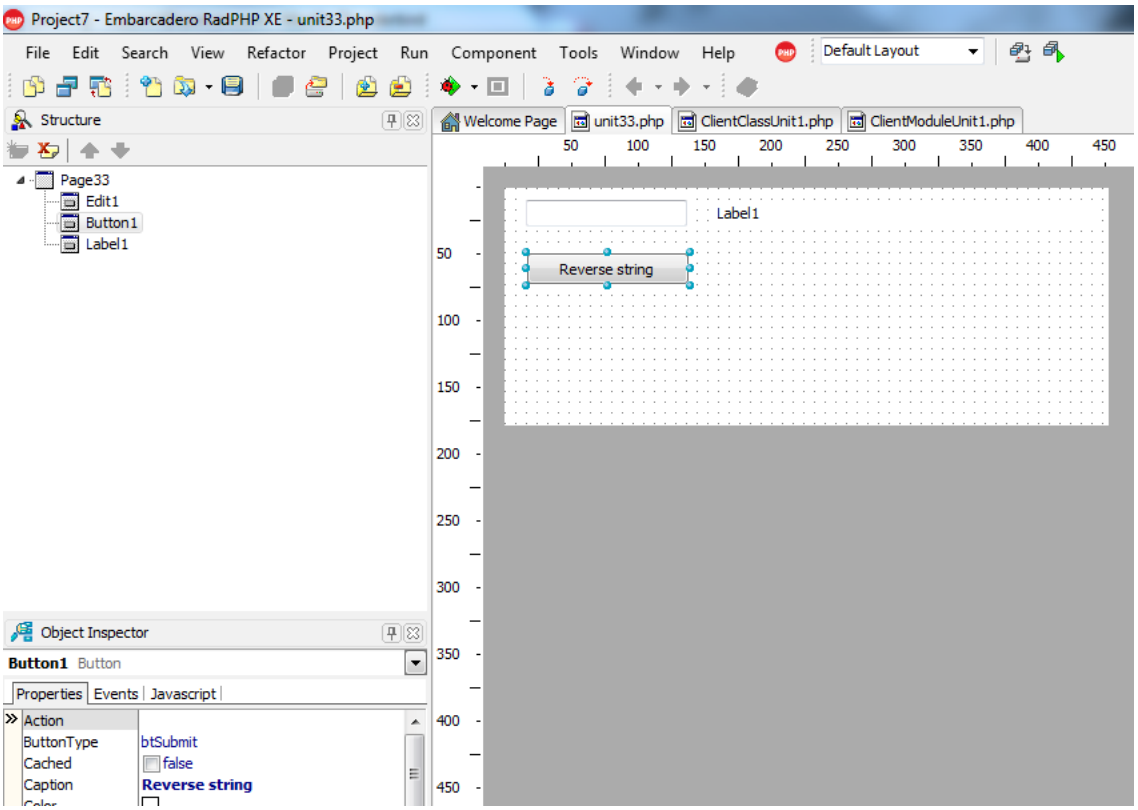
Once you press OK, all the required files will be created, let's take a look at the ClientClassUnit1 to see how that code has been generated and the methods we are publishing on the server, now are accessible from PHP.



DEVELOPING THE CLIENT INTERFACE

We are going to use the ReverseString method on the server, the plan is to allow the user enter a value, and show it reversed.

For that, let's add an Edit box, a Button, and a Label so your interface looks this way.



To access the functionality we have imported, we need to use ClientModuleUnit1.php by File | Use Unit…, which will add to our unit the right "require_once " declaration.

Now, double click your Button, to generate the OnClick event handler, and write this code:

```
1. function Button1Click($sender, $params)
2. {
3.   global $ClientModuleDataModule1;
4.   $this->Label1->Caption=
5.   $ClientModuleDataModule1->ServerMethods2Client->ReverseString($this->Edit1-
   >Text)->result;
6. }
```

First, we need to declare the ClientModuleDataModule as global, so we can access it.

After that, we can access the ServerMethods2Client property of the DataModule and call the ReverseString method with the value entered by the user on the Edit. And then, assign the result of that method call to the Caption of Label1.

If you run the application, enter some text in the Edit component, and click the Button, the reversed string will be placed on the Label.

ACCESSING THE SERVER FROM JAVASCRIPT

What we have done until now is to access the DataSnap server using PHP, now we are going to do the same, but using JavaScript.

To do that, we need to execute again the REST wizard, but we are going to specify JavaScript as the language to generate.

The wizard will generate a .js file, called ClientClassUnit, which contains all the JavaScript code required to access the server.

INCLUDING THE JAVASCRIPT CODE

To include the JavaScript code, we need to insert some html code into the header of the Page. To do that, you can use the OnShowHeader event, this event provides you the opportunity to add any code you want inside the header of the HTML document generated by the Page component.

Here is the code you need to add:

```
1. function Page33ShowHeader($sender, $params)
2. {
3. ?>
4. <script type="text/javascript"
   src="<?php echo RPCL_HTTP_PATH; ?>/js/ServerFunctionExecutor.js"></script>
5. <script type="text/javascript" src="ClientClassUnit2.js"></script>
6. <?php
7. }
```

As we are going to dump some HTML code, let's first scape PHP, and then, include the required JavaScript files. The first one, includes the ServerFunctionExecutor code, and also, uses the constant defined by the RPCL library, called RPCL_HTTP_PATH that is always set to the right value to find the assets required depending on the RPCL library location.

The second line includes the ClientClassUnit, which is relative to the location of the PHP.

CALLING THE METHOD

First, we need to setup the button to don't submit the form. That can be done by changing the ButtonType property of the Button to btNormal.

Then, generate the JavaScript OnClick event handler, where we are going to write the code we need in order to execute the method.

```
1. function Button1JSClick($sender, $params)
2. {
3.     ?>
4.     //begin js
5.     config=new Array();
6.     config['host']='localhost';
7.     config['port']=81;
8.     servermethods=new TServerMethods2(config);
9.
10.    Edit1=findObj('Edit1');
11.
12.    alert(servermethods.ReverseString(Edit1.value).result);
13.    //end
14.    <?php
15.   }
```

The very first thing to do is to create an array to hold the connection options, as that is required in order to create the object that contains the methods we need to call. Once the array is setup, we create the object. After that, let's get the Edit1 JavaScript object, so we can get the value entered by the user, and finally. Call the method, sending the value of Edit1 and showing the result in an alert window.

CONCLUSION

In this white paper, you have seen how easy is to access a Delphi DataSnap server from other languages, like PHP or JavaScript, and how you can reuse your existing code to be used by other software.

DataSnap technology opens up a new world of possibilities for your existing applications and to create new applications which are open and share data with other applications.

Embarcadero Technologies, Inc. is the leading provider of software tools that empower application developers and data management professionals to design, build, and run applications and databases more efficiently in heterogeneous IT environments. Over 90 of the Fortune 100 and an active community of more than three million users worldwide rely on Embarcadero's award-winning products to optimize costs, streamline compliance, and accelerate development and innovation. Founded in 1993, Embarcadero is headquartered in San Francisco with offices located around the world. Embarcadero is online at www.embarcadero.com.