Tech Note

embarcadero

Delphi for PHP "In Action"

By José León

qadram software S.L.

May 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

SUMMARY

PHP is an interpreted programming language which was initially developed to enable the creation of dynamic web pages. It is mainly used in the development of web applications and is executed on the server, although it has now evolved into a general purpose scripting-language and is even being used to develop desktop applications.

One of its most noteworthy features is that it is a multiplatform language – PHP scripts can be interpreted in a wide range of operating systems and environments, which makes it highly versatile. Another of its main features is the use of extensions, which provide the language with functions and enrich applications in a very simple way.

Its syntax is similar to that of C, with some touches of Perl, so it is very easy to learn. It does not require definition of variable data types, although it is possible to find out the data type of a variable at any time.

Delphi for PHP represents a revolution in the world of PHP application development as it provides a fully-integrated visual development environment and, most important of all, a class library for component building which makes interface development much easier.

Delphi for PHP is the simplest option for Delphi Windows programmers who want to learn to create web applications, as it uses the same IDE and a visual class library which has been modelled based on VCL for Windows.

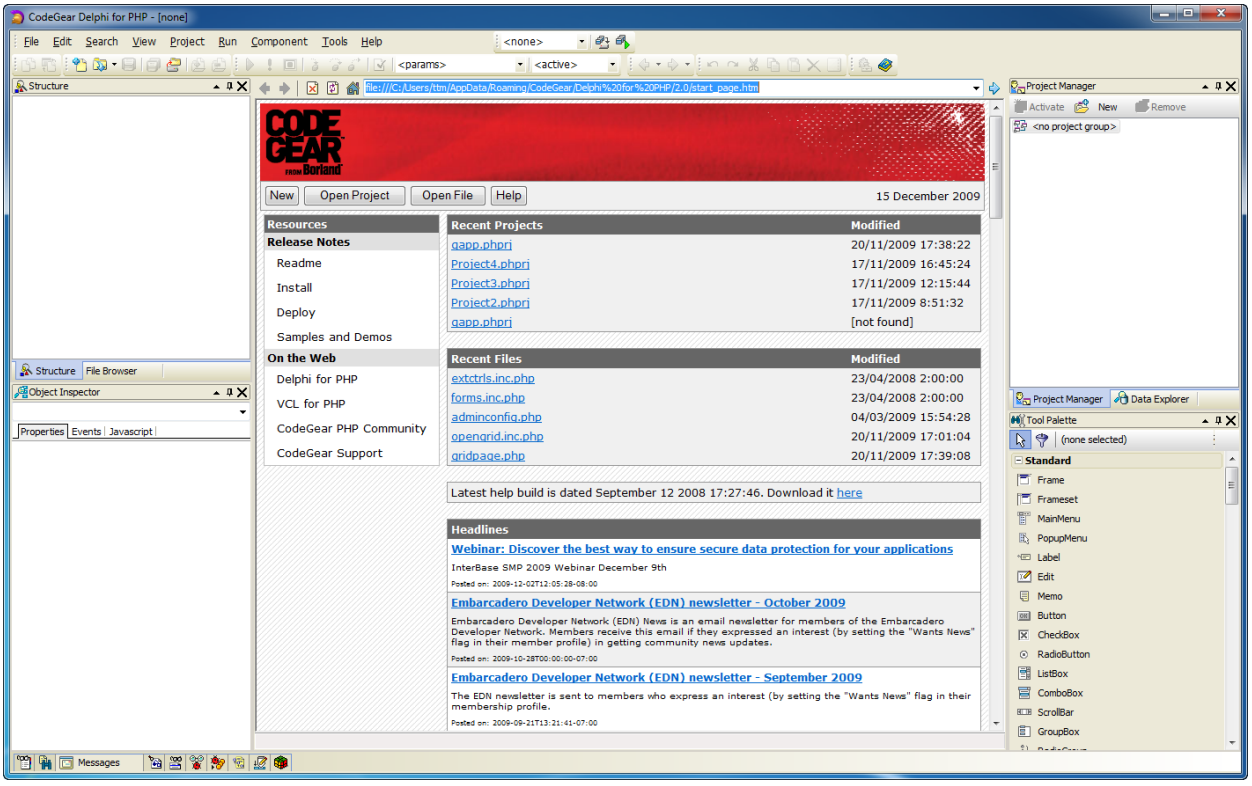
If you are considering developing web applications in PHP, this document will provide you with a guide to the most important features of the product. If you are already using PHP, it will show you how to increase your productivity as a Delphi for PHP user.

A TOUR OF THE IDE

MAIN WINDOW

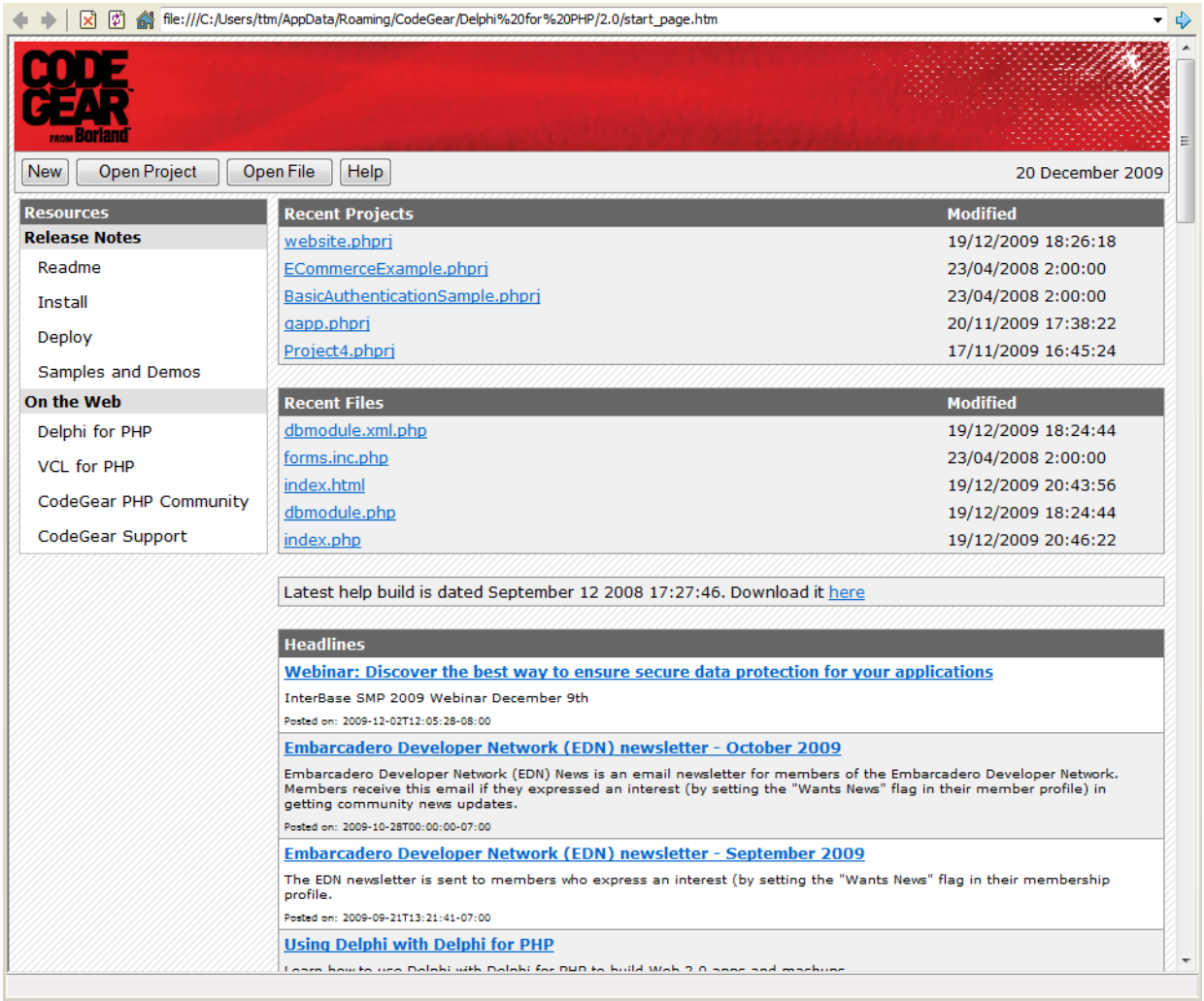
The main window groups all the elements that Delphi for PHP provides for application development. When the IDE starts up for the first time, the centre of the Welcome Page shows recent projects, news and links to other resources etc. On the left is the Structure panel and the Object Inspector and on the right the Project Manager, Data Explorer and Tool Palette.

The interface is made up of dockable windows which enable you to configure the development environment to suit the way you work. To move a tool across the screen, simply click on it and drag it to the required location. The tool windows can also be grouped in tabs so that they share the same workspace. These configurations can be saved using the tool bar at the top and the drop-down menus also enable you to change the current configuration.



WELCOME PAGE

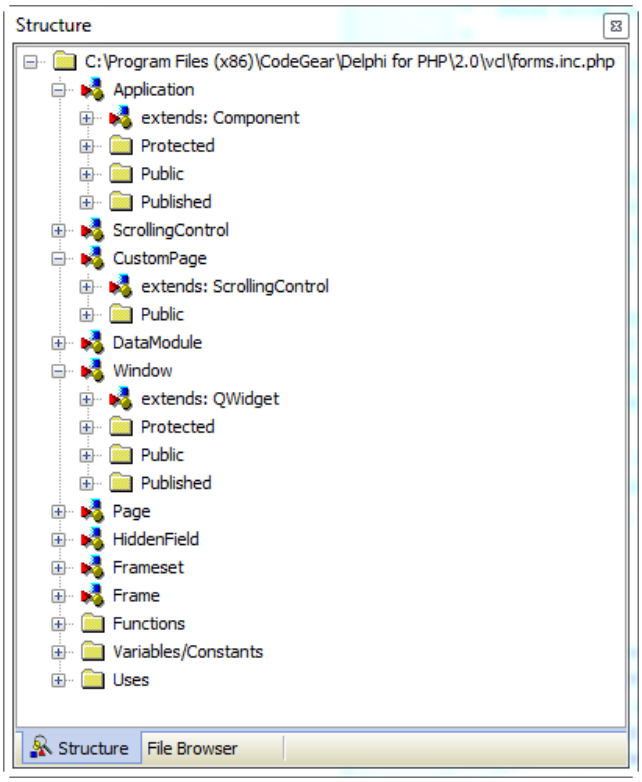
When it starts up the IDE shows the Welcome Page, which can be used to open a recent project or file, read the latest news about Delphi for PHP, access the release notes and installation readme file. It also provides web links for more information. Towards the top there are four buttons that enable you to create new elements, such as projects or components, open projects and files and get help. In fact this page is a browser and can therefore be used to browse the internet.

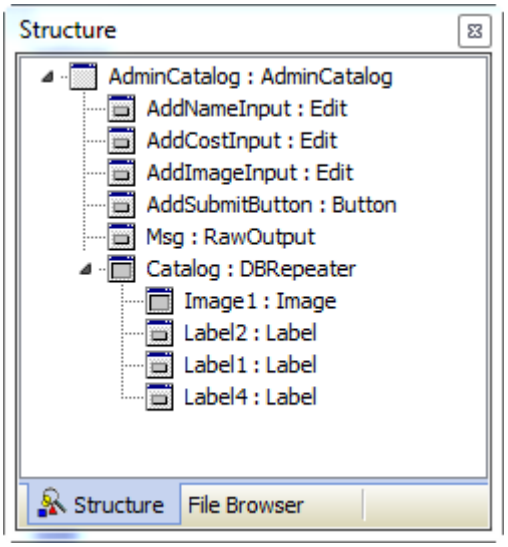


STRUCTURE

This panel shows the structure of the element being edited currently in the content area. For instance, if you are editing PHP code, the panel will show a tree structure of the source code content. However, if you are editing a form, the panel will show a tree structure of the components on the form.

Different actions are possible depending on the current view. If the panel is showing the source code structure, double-click on any element to jump to its declaration in the Code Editor. If the panel shows the structure of a form, double-clicking has no effect, but you can drag and drop elements onto each another, to insert one control inside another.





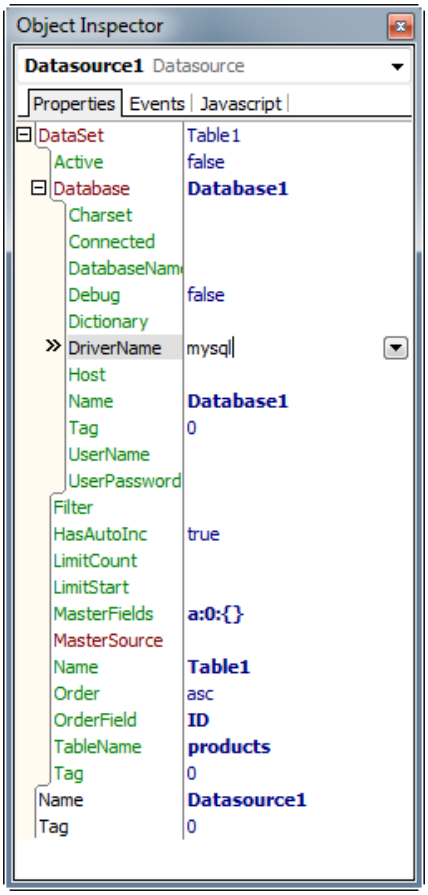
OBJECT INSPECTOR

The Object Inspector displays the properties and events of the object selected in the designer, both their name and value. It also lets you change the value or assign an event. Depending on the type of element selected, the tabs available change – for example, if it is a VCL for PHP component, the tabs displayed are Properties, Events and Javascript. If it is an HTML component, only the Properties tab is shown, the others not being relevant to HTML elements.

It is important to underline that the properties whose value differs from the default value are shown in bold. Also, properties that make reference to another component are shown in red and if they are assigned a value, they can be expanded so that the values of the referenced component can be changed directly.

The Object Inspector can be used to edit complex properties, such as Font, which enables nested properties to be set independently, such as Align or Size.

When editing a property, the type of editor that is shown depends on the property type – it can be a normal text editor, a value list or a custom editor.

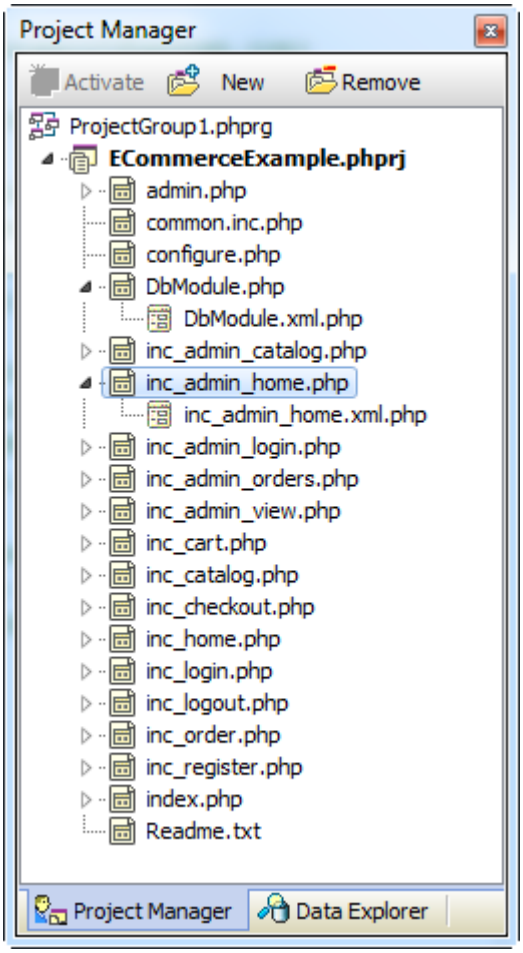


PROJECT MANAGER

This tool is used to group all the files in an application in projects and these in turn are grouped in project groups. Delphi for PHP does not require the creation of a project as it can be used perfectly well for working on independent files. However, there are tools, such as the internationalization wizard or the deployment wizard, which are more useful when working on a project since they need to know all the files that make up an application.

Any file can be part of a project, including HTML templates or CSS style sheets.

To add a new project to a project group, use the "New" button on the toolbar. An existing project can also be added to the current project group using the command Project | Add existing project.



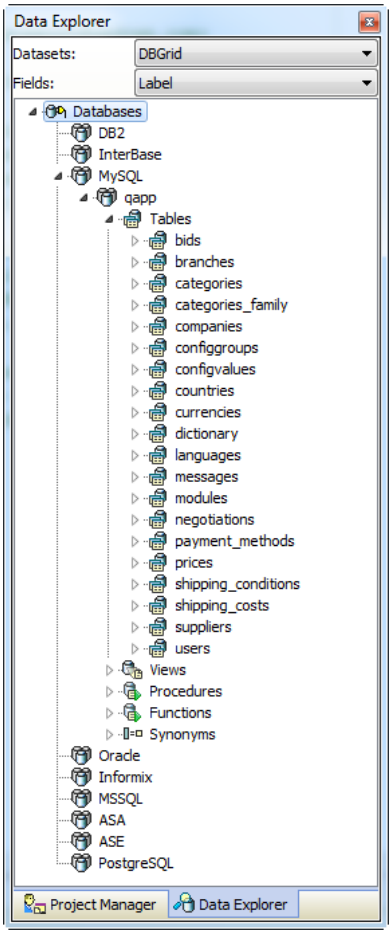
Finally, you can right-click on a project and choose "Add folder to project". This is a very useful way of including individual files in the project. This tool allows you to filter by file extension so that only files that meet the filter condition are added.

DATA EXPLORER

This tool has a double function – first, it allows you to explore databases, adding connection information and then expanding each level. You can explore tables, views, stored procedures, functions and synonyms along with their relevant fields and parameters.

The second function is the interaction with the Form Designer - you can drag and drop information from the Data Explorer on the active form. If you drag a database, a Database component will be created, with properties set to the appropriate values. If you drag a table, the necessary components will be created (Database, Table, Datasource), with required connections set, and also a DBGrid connected to the Datasource. If you drag a field from a table, an Edit field will be created and all the data access components that are required.

You can set the control type to be created with the two drop-down lists at the top – "Datasets" and "Fields". This is very useful when you are creating forms for data access.



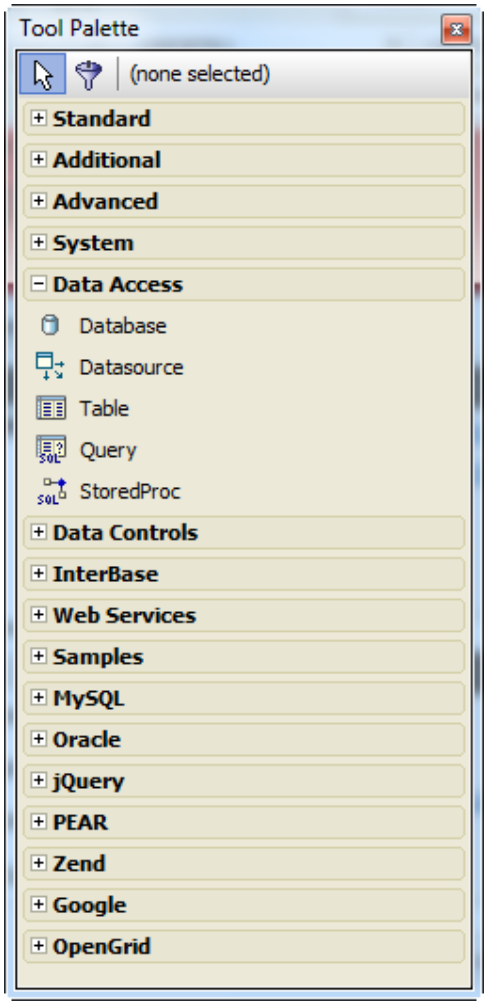
TOOL PALETTE

The Tool Palette contains all the components that you need to develop your application. These components are installed in the IDE through "packages" in the option Component | Packages. These components are grouped in categories such as "Standard", "Additional",

etc. And you can use the (-) symbol to collapse the categories and leave visible only those that interest you.

To use a component from the palette, click on its icon or its name and then click on a form. The component will be created at the default size. You can also drag and drop to create the component with dimensions that you set, and you can double click directly on the Tool Palette to create the component within the active control – centred and at the default size.

To search for a component, click on the filter icon and then, using the keyboard, write part of its name. As you type, the list of components will be filtered and only those whose name contains the string you are typing will be displayed.



VCL FOR PHP

STRUCTURE

VCL for PHP is the class library used by Delphi for PHP to create applications. It has been written using the same structure as VCL for Windows, so all base classes are the same and have the same uses.

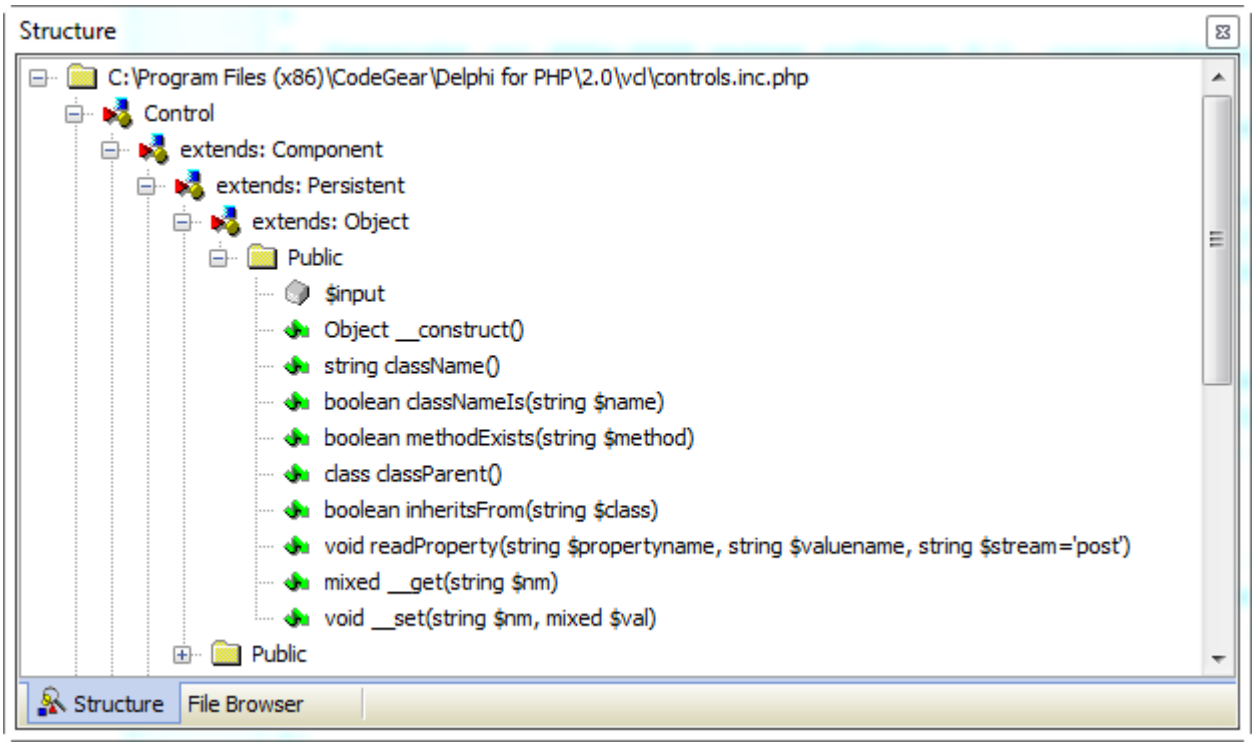
For example, the base class for all classes in the library is Object, the same as in VCL for Windows, which is TObject. This class provides basic methods such as "classname", which are available to all the classes derived from it.

Directly derived from Object is Persistent, which exposes methods and properties which enable you to store and access a class in the session.

Next, Component is derived from Persistent. Component is the minimal class to be used in writing a component which can be placed on a form.

Component exposes important properties such as Name and the concept of "property" between components.

Control is derived from component and is the base control for creating visual controls, that is, controls that produce code that is then interpreted by the browser.



PROPERTIES AND EVENTS

In Delphi for Windows, we can create a property for a particular component – i.e. an attribute that determines a feature of an object and which can be accessed in two ways. One is used to set the value and the other to access it. In this way the property can be formatted and validated or you can be notified when a change takes place.

In PHP syntax there is no keyword to define a "property", understood in this sense. However, there is no need for alarm as VCL for PHP provides a similar mechanism.

In VCL for PHP a property is made up of two methods – the "setter" and the "getter", and once the object has defined these two methods code can be written as follows:

```
01 <?php
02 require_once("vcl/vcl.inc.php");
03 //Includes
04 use_unit("system.inc.php");
05
06 class MyClass extends Object
07 {
08 protected $_myproperty="";
09
10 function getMyProperty() { return $this->_myproperty; }
11 function setMyProperty($value) { $this->_myproperty=$value; }
12
13 }
14
15 $object=new MyClass();
16
17 $object->MyProperty="¡hello, world!";
18
19 echo $object->MyProperty;
20
21 ?>
```

There are two types of property, Public and Published. The difference is in the name of the methods, read/write or set/get. In this way the IDE can know which properties to present or not present in the Object Inspector.

```
01 <?php
02 class MyClass extends Object
03 {
04 protected $_publicproperty="";
05
06 function readPublicProperty() { return $this->_publicproperty; }
07 function writePublicProperty($value) { $this->_publicproperty=$value; }
08
09
10 protected $_publishedproperty="";
11
12 function getPublishedProperty() { return $this->_publishedproperty; }
13 function setPublishedProperty($value) { $this->_publishedproperty=$value; }
14
15 }
16 ?>
```

PERSISTENCE

One of the differences between web and desktop applications is the data persistence. In a desktop application, the program does not finish until the main window closes and the values

assigned to component properties are therefore not lost until the application closes. In a web application, the scripts it is composed of are executed from start to finish and therefore lose their property values.

In VCL for PHP this is resolved automatically since the components store their values and recover them the next time the script is run.

This has the desired effect – i.e. whenever we set a property value, the property keeps it for the lifetime of the application.

DESIGN-TIME FEATURES

For an IDE to be able to manage components visually it is necessary for the components to be able to distinguish when they are running in the IDE and when they are running in the user application.

For example, at design-time, the component may want to show indicators, guides or additional information, exclusively for the developer. For this reason we need a way of knowing if we are in design-time or run-time. To achieve this we can find out if the bit csDesigning is set in the ControlState property – if it is activated, it tells us that the component is being used in the IDE.

```
01 <?php
02 if (($this->ControlState & csDesigning) == csDesigning)
03 {
04   echo "I am in the IDE";
05 }
06 else
07 {
08   echo "This is run-time";
09 }
10 ?>
```

Furthermore, for the developer to use the components requires making additional information available to the IDE – for instance indicating valid values for a particular property, or the type of editor to be used to edit a property.

We might also want to indicate to the IDE, through the component register, in which category on the component palette the component should appear.

CONTROL TYPES

There are currently several types of controls and components which range from less to more visual:

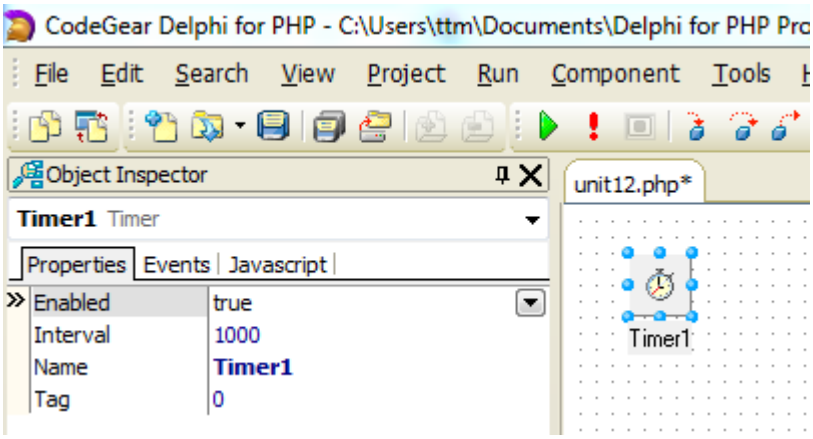
- Non-visual components
- Controls based on standard HTML tags
- Compound HTML controls
- Javascript controls

- Image controls
- Flash controls

Let's look in more detail at the main features of each one.

NON-VISUAL COMPONENTS

These components are derived directly from Component and they do not produce any output in the browser, but provide functionality such as the Timer component which is used to get notification through an event, depending on a time interval. It produces no visible code for the user, but it does use Javascript for all its programming. Data access components are another non-visual component type – they enable controls to connect with database tables.

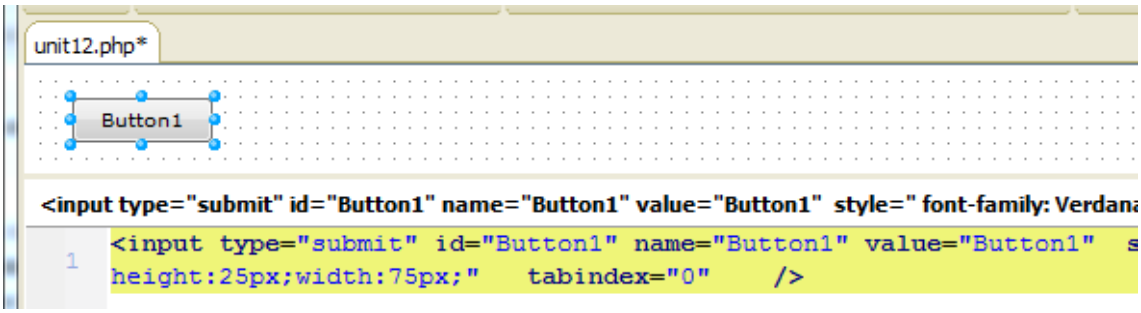


COMPONENTS BASED ON STANDARD HTML TAGS

These components generate code based on standard HTML tags such as <input> or <select> and use their properties to generate all the attributes that make up the tag's appearance.

Controls such as Button, Edit or ComboBox are based on these tags. The advantage of using these controls is that they are supported by practically all browsers and they adapt visually to the user interface within the operating system.

These controls are useful when advanced features are not required, as they are much more lightweight and run faster than more complex controls.

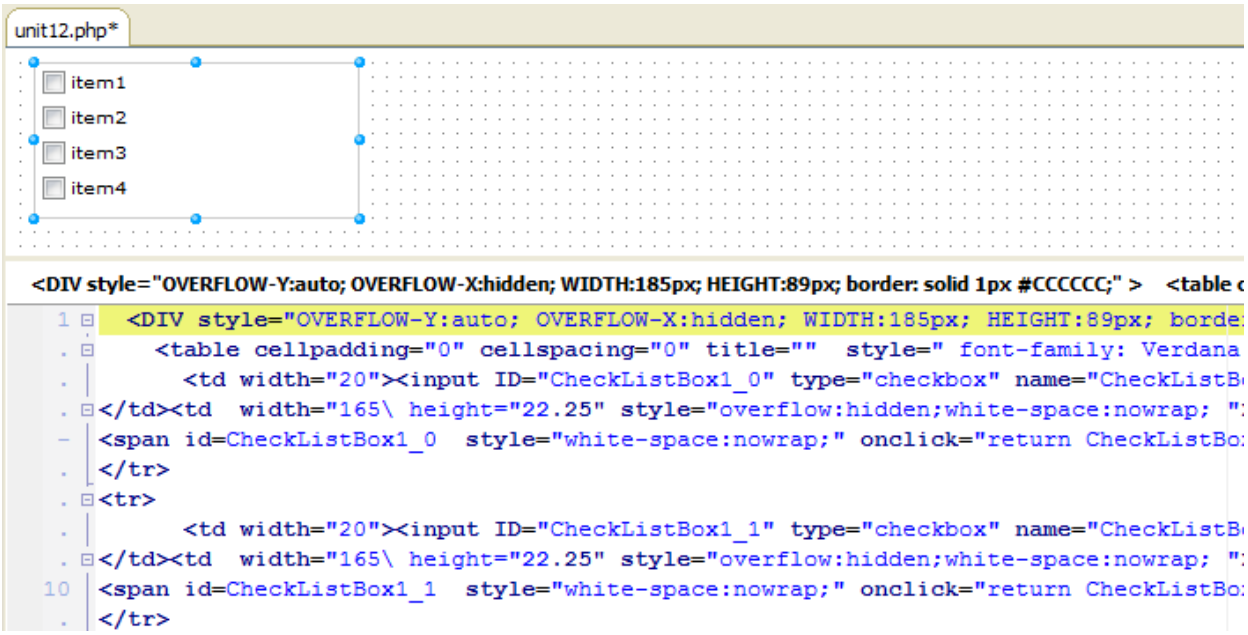


COMPOUND HTML COMPONENTS

Controls such as CheckListBox function using the available HTML – in this case tables, checkboxes etc. These components are also very lightweight, since they hardly use any Javascript and they work very well on different systems.

Actually there are many things that can be done with HTML in a browser to emulate desktop applications. Making use of the advantages of VCL for PHP enables us to create very sophisticated components that use HTML, images and CSS.

These components can also use Ajax to interact with the server and update their state without needing to do a complete send.



JAVASCRIPT CONTROLS

These controls are usually based on Javascript libraries, such as jquery, qooxdoo, xinha etc. They use the power of Javascript to create advanced controls, such as RichEdit or DBGrid, which emulate the complex controls that can be found in desktop applications.

Currently, browsers are improving their Javascript execution speed to an amazing extent, so that the use of these components are making web applications more and more like desktop applications.

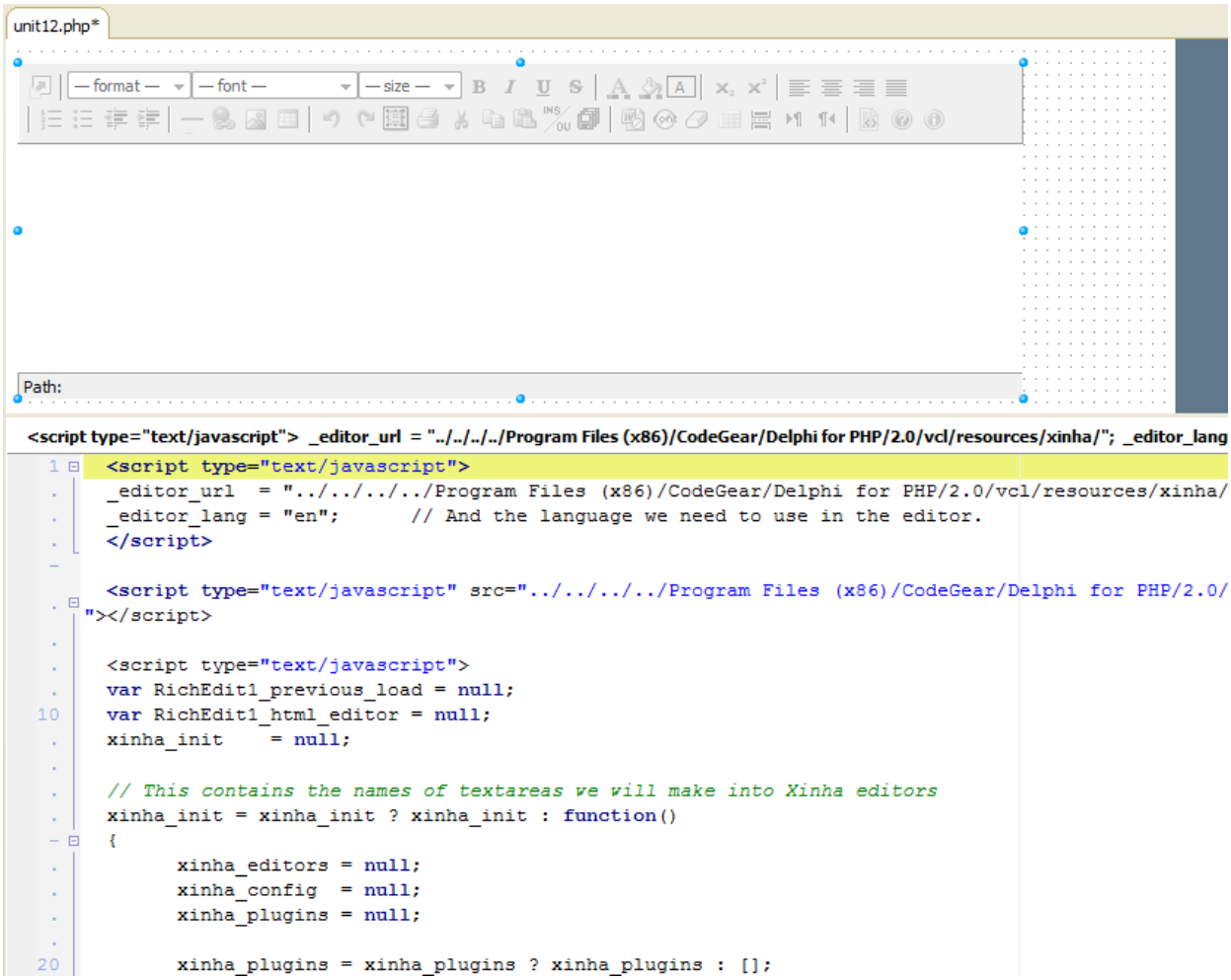


IMAGE CONTROLS

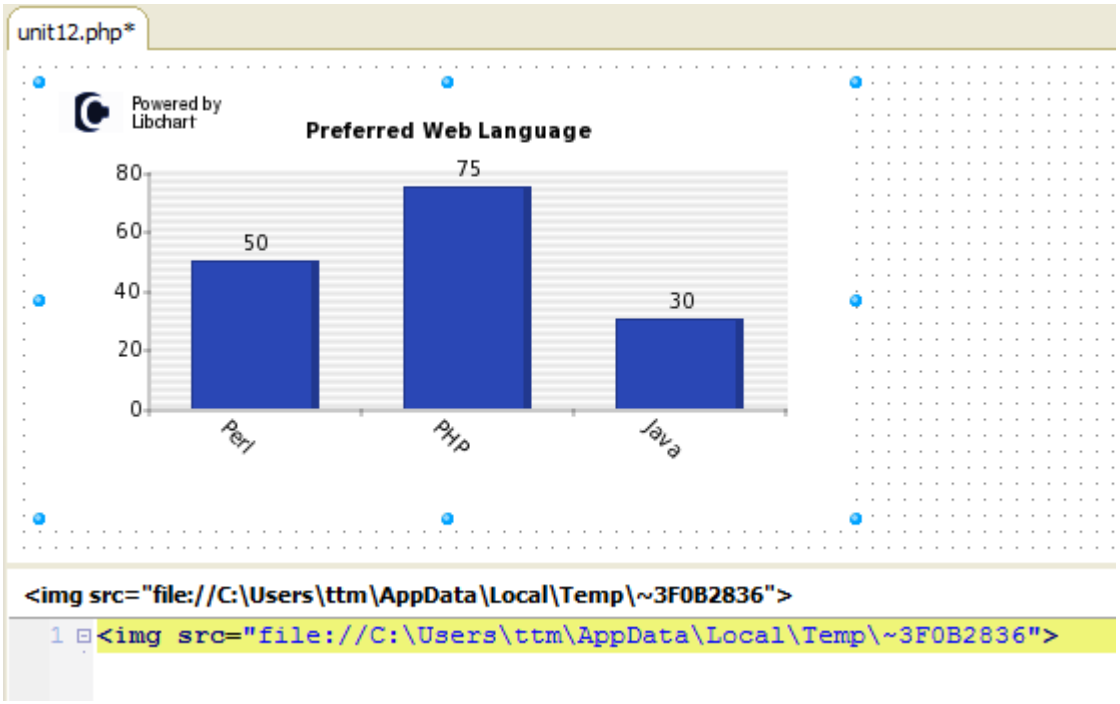
Sometimes a component, instead of generating HTML or Javascript code, may generate an image file, such as a PNG. This type of control is called an image control – an example is the SimpleChart component. These components use property values to generate an image dynamically and send it to output.

If you want to create this type of control, you need to set the csImageContent style to 1 for the control, which tells the IDE that it needs to manage it differently.

You can do this with this line of code in the builder:

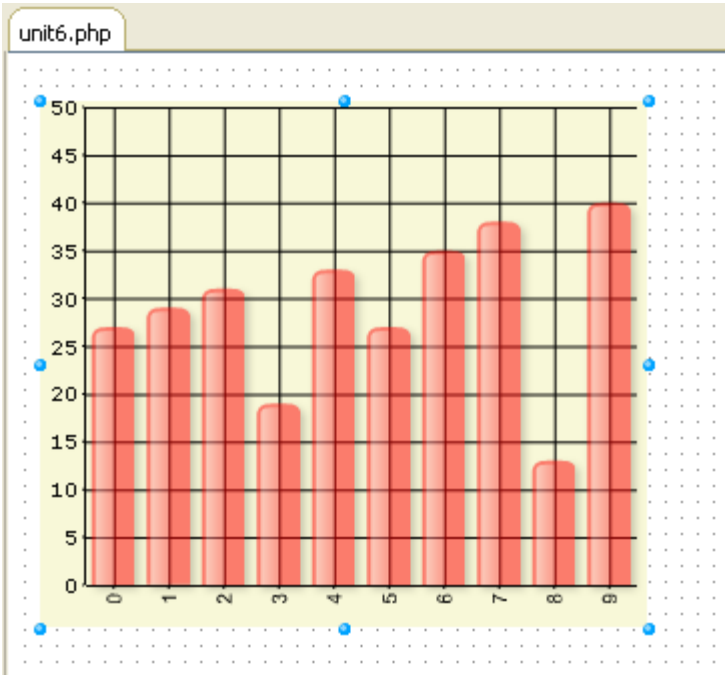
```
1 $this->ControlStyle="csImageContent=1";
```

As we can see in the picture below, the IDE first generates the image in a temporary file and then displays it using the <img> tag.



FLASH CONTROLS

We can say that this is the most advanced type of control. To function, the user's browser must have the Flash plug-in available. The property values of these components are sent to the Flash movie which uses them to display the control to the user. An example of this control type is OpenChart, based on a Flash library.



PACKAGES USING EXISTING CODE

The easiest way to create a VCL for PHP component is to package existing code. There is a host of libraries – Javascript, PHP, based on jquery, Flash etc. – which are ideal candidates for integration into Delphi for PHP. The class library was designed with this concept in mind and it is therefore a very simple process. All that is needed is the creation in the library of a component that will act as a wrapper, offering a common interface for properties, methods and events that will be used to call the library we want. In this way, we can bring the functionality of the library into the IDE and design interfaces visually and with minimum effort.

COMPONENT CREATION WIZARD

The simplest way to create a new component is to run the wizard provided by the IDE, using Component | New component. We need to indicate the base class for the development of our new component – this is normally Control. Next we must give the new component class a name and, finally, if we want to create the package for installation in the IDE, we must select the palette category we want.

Once we click "Accept" the unit will be created with the base code for our component:

```
01 <?php
```

```
02 require_once("vcl/vcl.inc.php");
03 //Includes
04
05 //Class definition
06 class MyNewComponent extends Control
07 {
08   function __construct($aowner = null)
09   {
10     parent::__construct($aowner);
11   }
12
13   function dumpContents()
14   {
15     parent::dumpContents();
16   }
17 }
18 ?>
```

And also, if we have selected it, the package code:

```
01 <?php
02 require_once("vcl/vcl.inc.php");
03 use_unit("designide.inc.php");
04
05 setPackageTitle("Put the title of your package here");
06 //Change this setting to the path where the icons for the components reside
07 setIconPath("./icons");
08
09 //Change yourunit.inc.php to the php file which contains the component code
10 registerComponents("MyComponents",array("MyNewComponent"),"unit1.inc.php");
11 ?>
```

COMPONENT PACKAGES

A component package is simply a PHP file which contains calls to PHP functions which interact with the IDE to indicate how a particular component or group of components is to be installed.

Apart from indicating where to install the components, they also contain instructions for the IDE about the properties of the components and how to edit them in the Object Inspector.

CODE GENERATION

Unlike components in desktop applications, components in VCL for PHP do not render their appearance (except in the case of image controls). Instead, they generate code for the browser to do the rendering.

The basic task, therefore, for a VCL for PHP component developer is to analyze the code to be generated, code that will be understood by the browser (HTML, Javascript, CSS) so that the component can change based on its property values.

This is perhaps the feature that most differentiates the creation of VCL for PHP components from the creation of VCL for Windows components. The rest - properties, methods and events - is exactly the same.

The main code generation takes place by overwriting the dumpContents() method, which is called by the VCL for PHP to get the component code. There are other methods, such as dumpHeaderCode(), which are more specific.

```
01 <?php
02 function dumpContents()
03 {
04   // set type depending on $_ispassword
05   $type = ($this->_ispassword) ? 'password' : 'text';
06
07   if ($style != "") $style = "style=\"$style\"";
08
09   // call the OnShow event if assigned so the Text property can be changed
10   if ($this->_onshow != null)
11   {
12     $this->callEvent('onshow', array());
13   }
14
15   $avalue=$this->_text;
16   $avalue=str_replace('"','&quot;',$avalue);
17   echo "<input type=\"$type\" id=\"$this->_name\" name=\"$this->_name\"
value=\"$avalue\" $style />";
18 }
19 ?>
```

APPLICATION LOCALIZATION

Localization of applications in Delphi for PHP consists of two very different processes: translation of the strings or literals used in the code and translation of component properties of string type.

CODE STRINGS

To translate code text strings, the world standard method for application translation - *nix – is used. This is the *gettext* which is fully supported by the PHP engine.

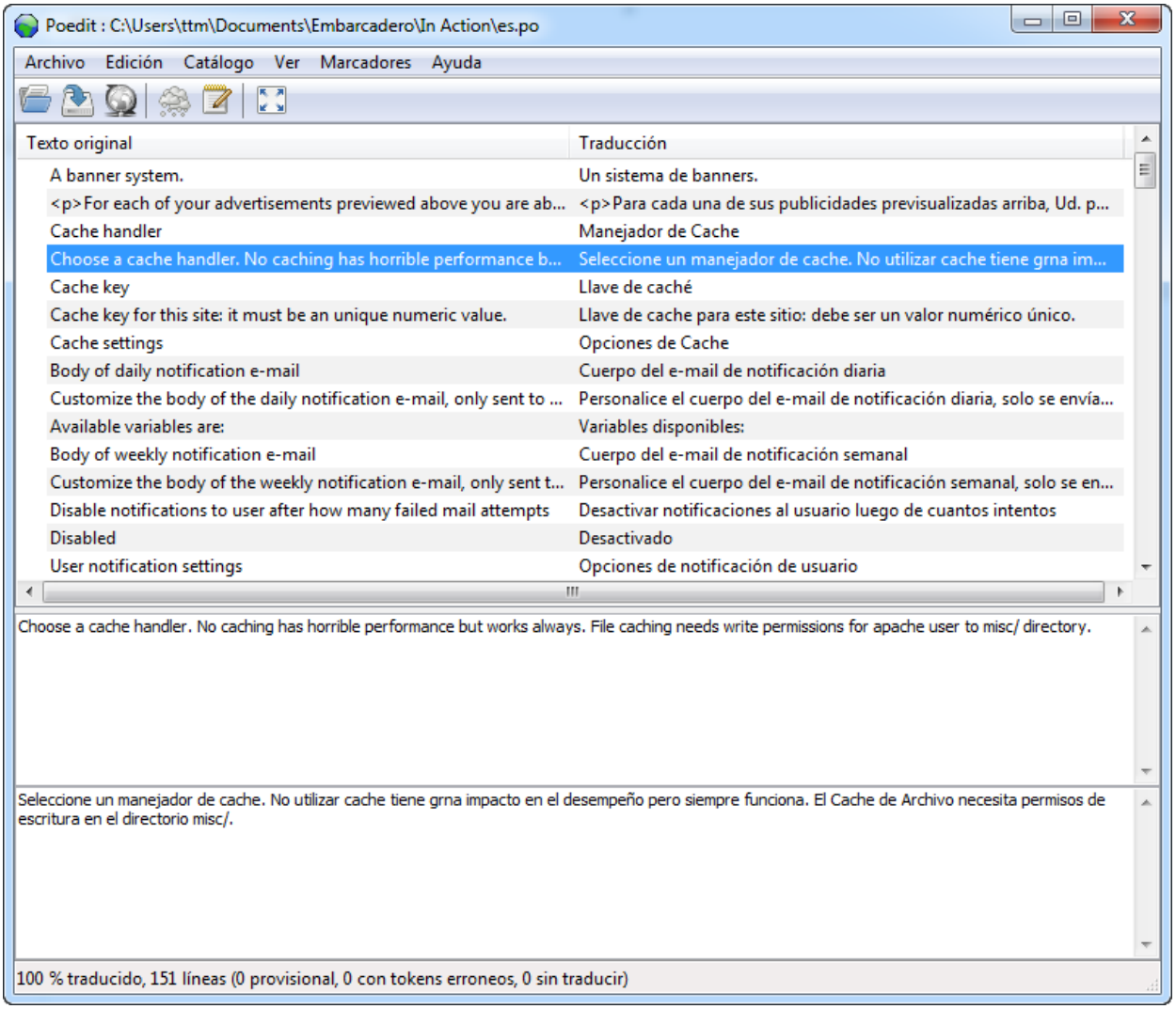
To indicate that a text string is to be translated, we must put it inside the function _("") so that the parser which extracts these strings can identify them. Also, calling this function ensures that the returned string will be the right translation for the current language.

```
1 <?php
2 function myfunction($parameter)
3 {
4   echo _("String to be translated");
5 }
6 ?>
```

To translate the strings, we use the internationalization wizard which groups the files that make up the project and requests information about the languages into which they will be translated. It then runs the xgettext program on the strings which extracts them and sends them to a file with a .po extension.

Once we have this file with the .po extension, we can use an editing tool called poEdit (or any other editing program that supports .po) which enables us to translate the stored strings and generate a binary file with a .mo extension. This .mo file is used by the PHP engine to translate the strings requested through the _("") function.

We can repeat this process as many times as we like as we develop our project, since any new strings will be added to the .po file and those that have already been translated are not lost.



VISUAL PROPERTIES

To translate visual properties, we have to change the Language property of the form that we are working on. If we change to another language, for example French, all the property changes that we make in that language will be stored in a separate .xml.php file – not only

properties of string type, but all types. These properties are used only when the current language changes.

Depending on the language into which we wish to translate our application, there are times when the controls are not big enough to hold the new text strings. We can therefore re-size our form to that the controls are correctly formatted.

CHANGING THE CURRENT LANGUAGE

To change the language of our application at run-time we need to use the Language property of global object application:

```
01 <?php
02 function Button1Click($sender, $params)
03 {
04 global $application;
05
06 if ($application->Language == "(default)")
07 {
08 $application->Language = "Spanish (Traditional Sort)";
09 }
10 else
11 {
12 $application->Language = "(default)";
13 }
14 }
15 ?>
```

In this way the PHP engine will use the .mo file appropriate to the new language. Depending on the operating system on which the application runs, we may need to make some adjustments for it to function correctly. In some versions of Linux we have to install the right "locales" for the languages in which our application functions, but normally it will run by default on any server.

USING CASCADING STYLE SHEETS (CSS)

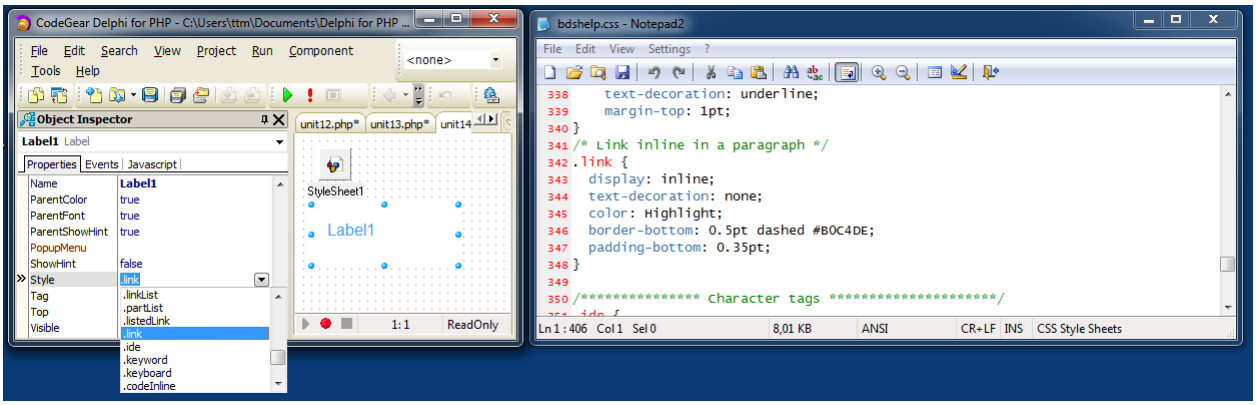
Apart from the properties used by the components to set their appearance, we can use CSS style sheets and assign them to the components.

Components can have properties that have the style to be used for a particular aspect. For example a DBGrid can have one property to set the style of the headers and another to set the style of the data cells.

STYLESHEET COMPONENT

The easiest way to attach a style sheet to a form is through the StyleSheet component. This component is a .css archive linked through the property FileName.

Once this link is made, any component with a style sheet property has a drop-down list in the Object Inspector whose values are the styles contained in the style sheet. You can try this out with the Label component and its Style property.



JAVASCRIPT

JAVASCRIPT EVENTS

So far, when we have referred to events, discussion has been restricted to PHP events - i.e. those events that take place on the server and whose code is written in PHP.

Apart from these events, browsers also support the execution of Javascript code which runs on the client computer, or more specifically in the browser, without sending any request to the server. This type of event is therefore very useful when we want to carry out operations that need a rapid response (e.g. responses to mouse movements) or when we want to update the information seen by the user without needing to go to the server for it.

VCL for PHP also supports Javascript events. These events can be accessed on the Javascript tab in the Object Inspector. Double-clicking on any of these events will generate the event handler where we can write our code, and the language we must use to do so is Javascript.

```
1 <?php
2 function Button1Click($sender, $params)
3 {
4 ?>
5 //Add your javascript code here
6 window.alert("This event takes place when the button is clicked.");
7 <?php
8 }
9 ?>
```

FINDOBJ

Something that surprises developers when they start to use Javascript is that it is not compatible with all browsers. Although there is a standard for the language that all browsers

should comply with, the reality is rather different, although fortunately, little by little, these differences are reducing.

Among the most common Javascript programming tasks is searching for objects. For example to search for an edit box or a button on a form we need to get an object reference to be able to use it. VCL for PHP includes a standard routine for getting an object reference, which abstracts away the differences between browsers and even functions when the objects are in different frames.

This routine is called findObj and the name of the object we want to get is passed to it.

```
1 //Find the object
2 hidden=findObj('edit_hidden');
3
4 //Use it
5 hidden.value=edit.value;
```

DATA ACCESS

PHP is a data access language par excellence. Because it was originally a programming language for web page creation, it can make available a range of ways of accessing data and - more importantly - very quickly.

DATA ACCESS MODEL

VCL for PHP implements the same data access model as VCL for Windows: Database -> Dataset -> Datasource -> Control

ADODB

Standard data access components are based on the ADOdb library which, in spite of its name, is not to be confused with the Microsoft ADO technology. This is a library written 100% in PHP, which provides an abstraction layer over the database engine used in our application, and which facilitates migration to other database engines with minimal changes, if any.

So, when we use the standard components, we don't need to worry about what database we are accessing, since these components do most of the work for us.

NATIVE COMPONENTS

Obviously, using an abstraction layer over the data has a negative impact on performance. Though this is not a problem for most applications, for some where data access speed is crucial, it may be important.

To address this, VCL for PHP includes native data access components for InterBase, Oracle and MySQL. Rather than ADOdb, these components use native PHP functions for data access.

AJAX

Ajax is a technology which enables calls to be made transparently from a browser to a server, without the user perceiving any refresh in the browser content.

If we want to use Ajax on a form, we need to set the UseAjax property to TRUE, so that requests are processed correctly. To make an Ajax call, we must use a Javascript event, for instance, by pressing a button. In the code for the event, we need to use the AjaxCall method, which generates all the Javascript code required to make the call:

```
01 //Class definition
02 class Index extends Page
03 {
04   public $Label1 = null;
05   public $Button1 = null;
06   function Button1JSClick($sender, $params)
07   {
08     //Dump the call using Ajax to the Button1Click event
09     echo $this->Button1->ajaxCall("Button1Click");
10   ?>
11     //Return false to prevent the button submit the form
12     return(false);
13   <?php
14   }
15
16   function Button1Click($sender, $params)
17   {
18     $this->Button1->Caption = "clicked Ajax " . date("Y-m-d g:i:s a");
19     $this->Label1->Caption = "Hello from Ajax!! " . date("Y-m-d g:i:s a");
20     $this->Label1->Color = "#" . dechex(rand(0, 0xFFFFFF));
21   }
22 }
```

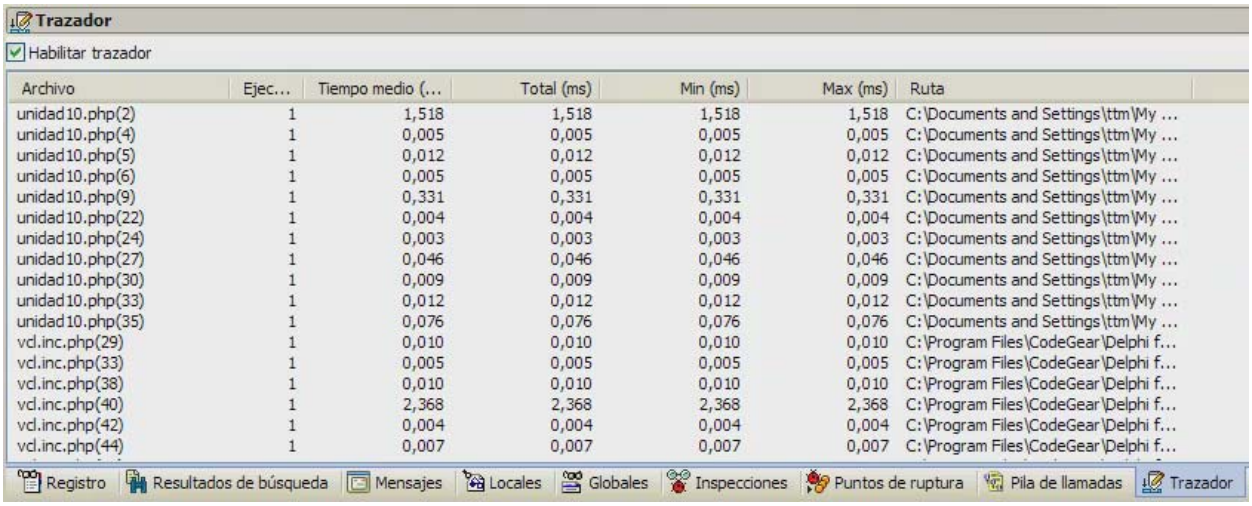
APPLICATION DEBUGGING

Delphi for PHP includes an integrated and fully configured debugger which enables you to debug PHP applications from the beginning. You just need to insert a breakpoint and press F9.

TRACER

This tool enables you to find the bottlenecks in your application, since it produces a report with all the lines in the application that have been executed, how many times and the time each line took to run.

To use the tracer we need to enable it by ticking the "Enable tracer" check-box and then using it in debug mode.



REAL WORLD APPLICATIONS

In this second part of the document, let's put theory to one side and focus on practice. We'll see how quickly we can develop a corporate web page, and then see how to interact with Delphi for Windows.

CORPORATE WEB PAGE

Let's create a simple example of a corporate web page, which we can use to see all the programming features of Delphi for PHP. In this example we will see how to use templates, how to connect to a database and how to deploy our application.

ESTABLISHING REQUIREMENTS

First let's specify the pages which will make up our website:

- Start
- Services
- Customers
- Contact
- Company

The content of these pages will be stored in a database, so that in the future we can create an interface where our imaginary customers can update their content. Also when we create a new page, reusing the visual appearance will be easier. We also want the graphical features to be independent of the content, so we are going to use a template system.

CREATION OF THE DATABASE

We will create a corporate website, where the text of the pages will be stored in a database table. In this way, we can centralize the script that will display the pages and we can also create new pages more easily.

For our application, we will use MySQL, which is probably the database engine most used by PHP programmers. It is very fast and usually comes preinstalled on most hosting packages available for PHP.

The database will be very small, with a single table to hold the pages, with a field for the identifier and another for the text to be displayed.

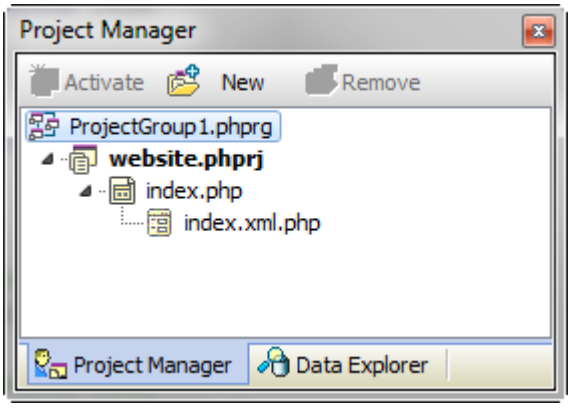
```
1 CREATE TABLE IF NOT EXISTS `pages` (
2 `id` int(11) NOT NULL AUTO_INCREMENT,
3 `page` varchar(40) NOT NULL,
4 `text` text NOT NULL,
5 PRIMARY KEY (`id`)
6 ) ENGINE=InnoDB DEFAULT CHARSET=latin1 AUTO_INCREMENT=1 ;
```

This document does not explain the installation and running of a MySQL server, but once installed, we recommend installing phpMyAdmin (http://www.phpmyadmin.net) which enables all the necessary operations to be performed. With this tool, we can create the database, and within it run the database table creation script above.

You can also enter data with PhpMyAdmin, which makes it very useful for entering the default page information.

CREATION OF THE PROJECT

To begin programming, we open Delphi for PHP and we create a new application. When a new application is created, it always contains a new form. Using the Object Inspector, we can give this form a name - in this case "Index". Now we save everything to a directory of our choosing and name the unit that has been created "index.php", since this will be our main script. Once everything is saved it should look like this:

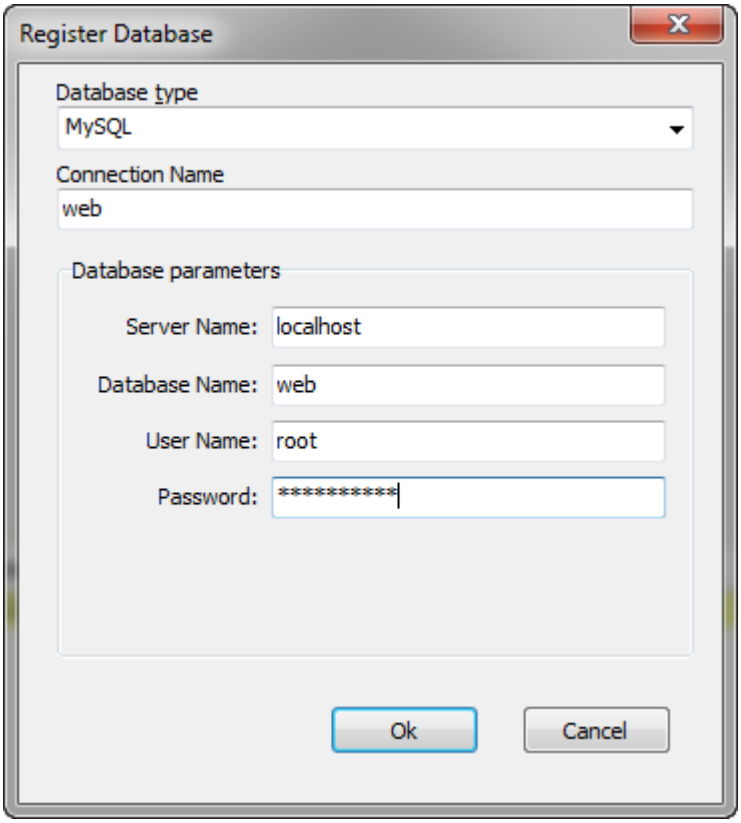


CONNECTION TO THE DATABASE

To access the database, we will use a DataModule. For such a small project, it is not strictly necessary, but it is a good idea to get used to storing database connections in a separate module.

To add a new data module, we click File | New | Data module. As we can see this is a container - but it only supports non-visual components. Using the Object Inspector, we change the Name property to "DBModule" and save this data module in the same folder as the rest of the project, with the name, for example, "dbmodule.php".

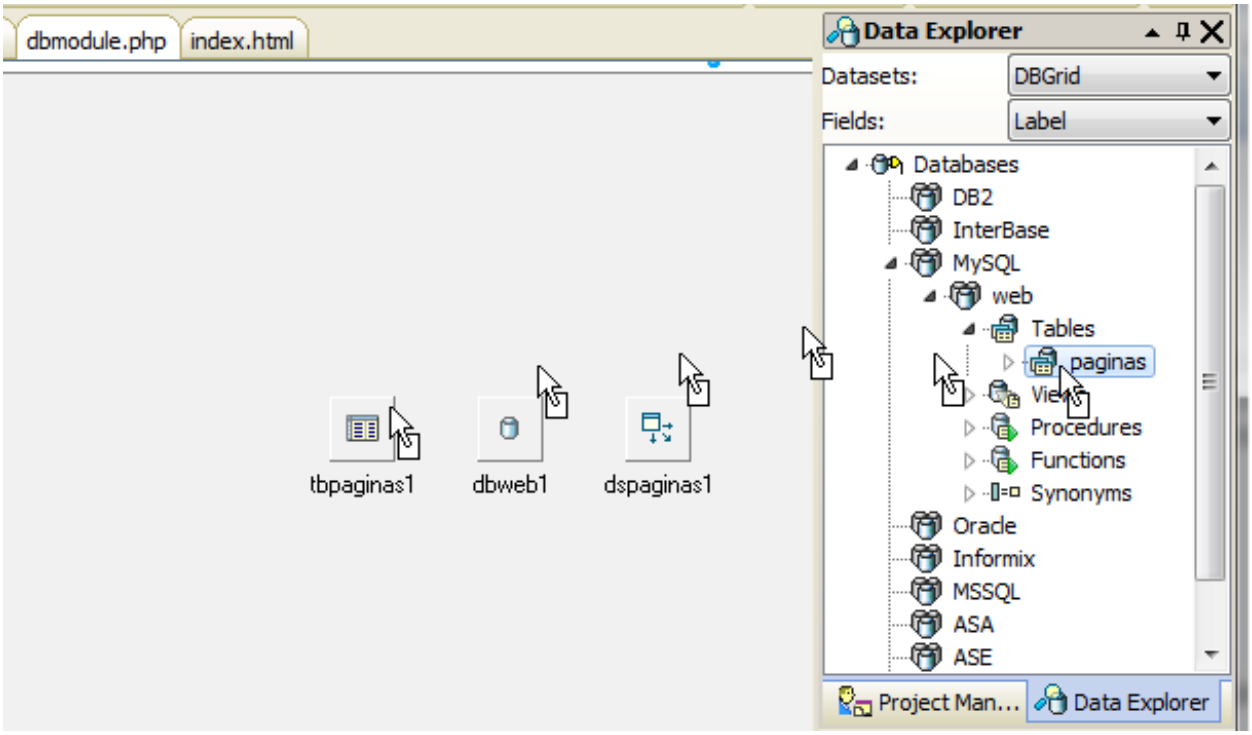
Now we will use the Data Explorer. We right-click on the MySQL node and select "Register Database". We now see a dialogue box, where we need to enter the connection information for our data base.



Once we have done this, we click OK and our database is now registered. We can now deploy it to see its contents - more concretely, the tables it contains.

This operation can be carried out with any of the supported database types.

And now, we need to click and drag the "pages" table from the Data Explorer to the data module.



This operation has created three new components for us:

- Database
- Table
- Datasource

The three components already have all the necessary properties set to the correct values and they are interconnected and ready to work.

If we had to do all this by hand, we would have to do the following:

- Put the Database component on the data module and set the correct values in DatabaseName, DriverName, Host, Username and UserPassword.
- Put the Table component on the data module, connect its Database property and set the value of the TableName property.
- Put a Datasource component on the data module and connect its Dataset property to the Table component.

As we can see, it is much simpler to drag and drop than to do all this.

TEMPLATE SYSTEM

We are going to use templates, because they offer lots of personalization and flexibility, normally templates are what we receive from designers. For this project, we will download a free corporate template, which conforms to the XHTML standard.

We need to remember that we can change the template later without this affecting at all the programming of our website.

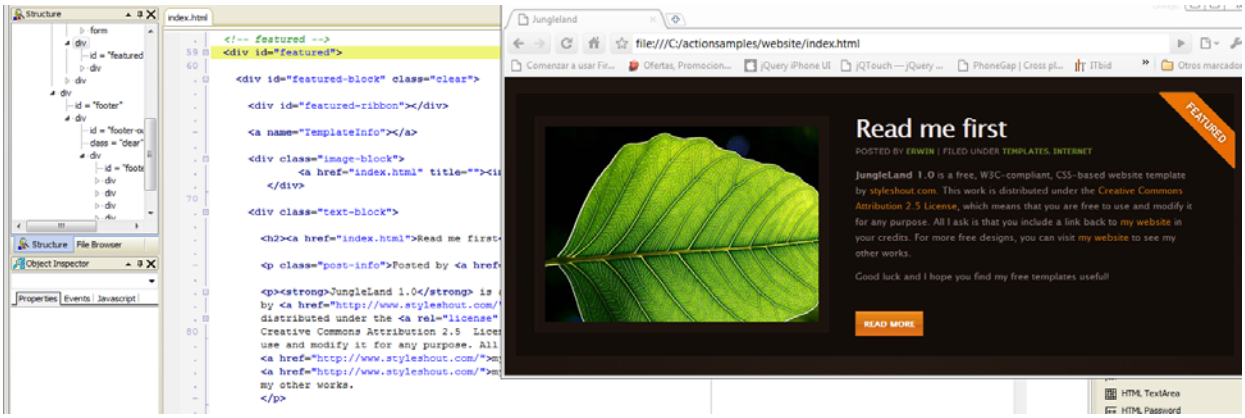
For this example, we are using this template:

http://www.styleshout.com/templates/preview/Jungleland1-0/index.html

We download it from the internet, we decompress it and save its contents in the project folder itself. Alternatively it's no problem to put it in a sub-folder, but it is better in the project folder because otherwise designers might not notice where it is.

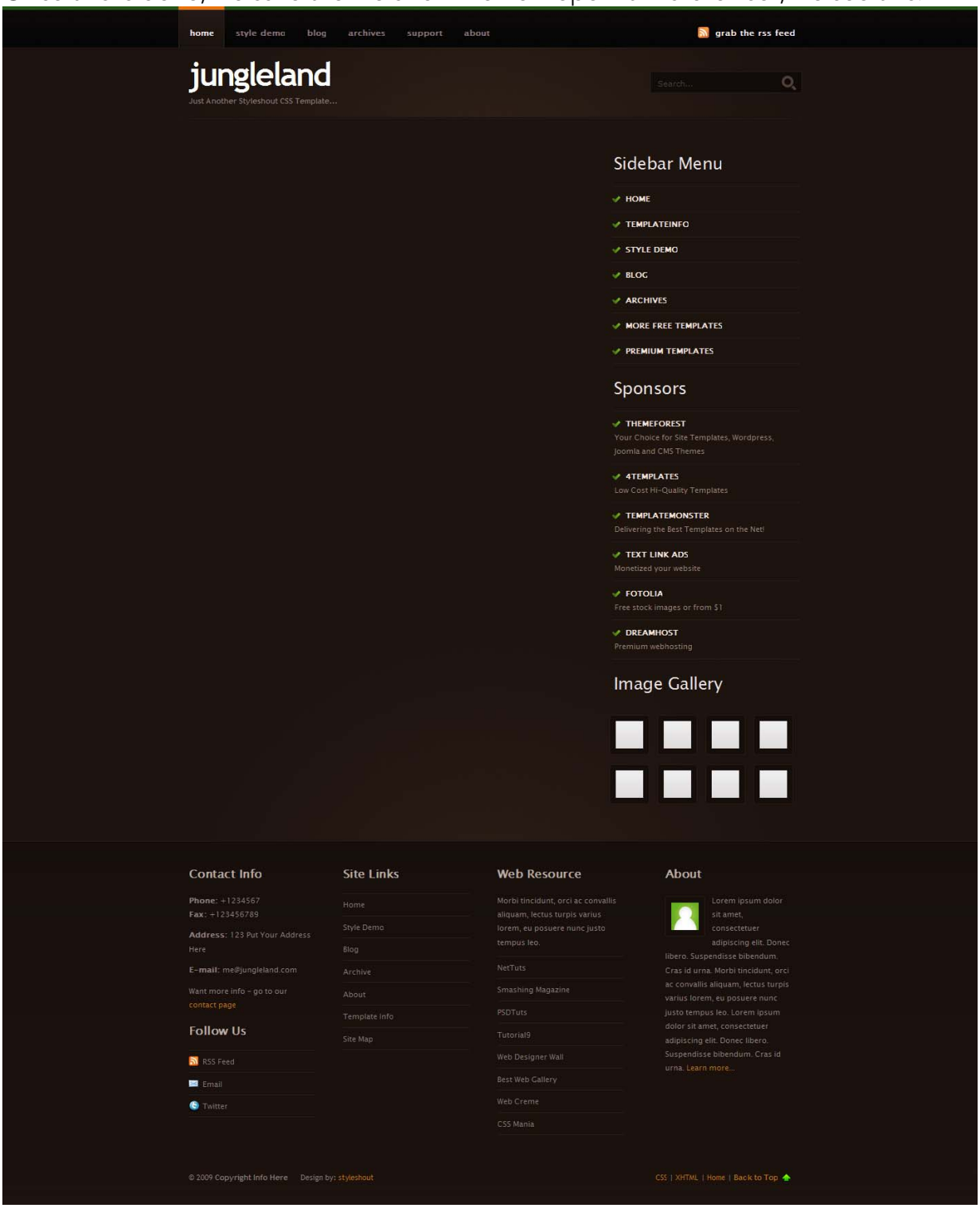
The first thing we need to do is empty the central area, which is where we will enter the content from the database.

To do this, we edit the index.html file and remove the tag with id "featured" - this section is relevant only to the main page. It's sensible to make a copy of the template, since several of these removed sections will be used again later.



When we remove this we need to find the tag and empty the whole tag, since this is the content that will vary on each page.

Once this is done, we save the file and if we now open it in a browser, we see this:



Now we will configure the page so that when it is displayed it uses this template. To do this we need to use two properties:

- *TemplateEngine*, which specifies the template engine to be used - we select "SmartyTemplate"
- *TemplateFilename*, which specifies the template to be used - in our case, "index.html"

With these simple steps all we need to do now is run index.php using F9 and we will get the template exactly as it is in the index.html file.

SmartyTemplate is based on the template engine that is most used by PHP programmers - it is named Smarty and there is more information about it at: http://www.smarty.net/

To be able to place dynamic content inside the template, we need to create a placeholder - this is achieved with the special tags {%$CONTENT%}.

```
<!-- main -->
<div id="main">
{%$CONTENT%}
<!-- /main -->
</div>
```

In this way we can insert the content we need into the template. We can try this out by inserting some content, to see that it all works. To do this we generate the page's OnTemplate event and we enter this code:

```
function IndexTemplate($sender, $params)
{
//Get the object template
$template=$params['template'];

//Assign content to the placeholder
$template->_smarty->assign('CONTENT','Hello World!');
}
```

Here we are getting a direct reference to the smarty object and assigning a value to it. If we run this, we will see how the content area now contains the string "Hello World!".

PAGE PROCESSING SCRIPT

We've already seen how to insert content in the placeholder we created, so now we are going to process the page requests and insert the content that is in the database.

Links to the pages will have the form index.php?page=name, where name should match the "page" field in the database table.

We need to search for the record and insert the content of the "text" field in the content area on the template.

To do this we need to include in our script the data module that holds the connection with the database. We use File | Use unit and we select "dbmodule.php"

This adds, in the include section, the command require_once with the source code that we have indicated. Now we can access the object itself.

First we'll see if there is any page request, and we do this using the $input object to create a filter and prevent any type of attack:

```
1 global $input;
2
3 //We look for the "page" parameter in the input
4 $page=$input->page;
5
6 //If the parameter is there, we filter with a string
7 //If there is no parameter we will get the default page
8 if (is_object($page)) $page=$page->asString();
9 else $page='start';
```

Now we need to use this page name to search for the content in the database table. For this, we simply filter the table and take the first record. Here is the full routine:

```
01 function IndexTemplate($sender, $params)
02 {
03 //Get the object template
04 $template=$params['template'];
05
06 global $input;
07
08 //We look for the "page" parameter in the input
09 $page=$input->page;
10
11 //If the parameter is there, we filter with a string
12 //If there is no parameter we will get the default page
13 if (is_object($page)) $page=$page->asString();
14 else $page='start';
15
16 global $DBModule;
17
18 //We filter the page table for the page that we want to display
19 $DBModule->tbpages1->close();
20 $DBModule->tbpages1->Filter='page="'.$page.'"';
21 $DBModule->tbpages1->open();
22
23 //Fetch the content
24 $content=$DBModule->tbpages1->text;
25
26 //Assign content to the placeholder
27 $template->_smarty->assign('CONTENT',$content);
28 }
```

Now we just need to enter the texts for our pages in the database. These will be: start, services, customers, contact, company.

For the start page we will use the section that we removed from index.html, the <div "featured"> tag. We can insert this with phpMyAdmin in the same database. Here is the section:

```
<!-- featured -->
<div id="featured">
<div id="featured-block" class="clear">
<div id="featured-ribbon"></div>
<a name="TemplateInfo"></a>
<div class="image-block">
<a href="index.html" title=""><img src="images/img-featured.jpg" alt="featured"
width="350px" height="250px"/></a>
</div>

<div class="text-block">
<h2><a href="index.html">Read me first</a></h2>
<p class="post-info">
Posted by <a href="index.html">erwin</a> | Filed under
<a href="index.html">templates</a>,
<a href="index.html">internet</a>
</p>
<p>
<strong>JungleLand 1.0</strong> is a free, W3C-compliant, CSS-based website template
by <a href="http://www.styleshout.com/">styleshout.com</a>. This work is
distributed under the <a rel="license"
href="http://creativecommons.org/licenses/by/2.5/">
Creative Commons Attribution 2.5 License</a>, which means that you are free to
use and modify it for any purpose. All I ask is that you include a link back to
<a href="http://www.styleshout.com/">my website</a> in your credits. For more free
designs, you can visit
<a href="http://www.styleshout.com/">my website</a> to see
my other works.
</p>

<p>Good luck and I hope you find my free templates useful!</p>
<p><a href="index.html" class="more-link">Read More</a></p>
</div>
</div>
</div>
```

If we now run the application, we will see how in the centre the "featured" HTML section appears.

For the services page, we will put a section of style.html, which is a blog post:

```
<div class="post">
<h2><a href="index.html">A Blog Post</a></h2>
<p class="post-info">Posted by <a href="index.html">erwin</a> | Filed under <a
href="index.html">templates</a>, <a
href="index.html">internet</a></p>
<p>Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Donec libero. Suspendisse
bibendum.
Cras id urna. Morbi tincidunt, orci ac <a href="index.html">convallis aliquam</a>,
lectus turpis varius lorem, eu
```

```
06 posuere nunc justo tempus leo.</p>
07 <p>
08 Donec mattis, purus nec placerat bibendum, <a href="index.html">dui pede
condimentum</a>
09 odio, ac blandit ante orci ut diam. Cras fringilla magna. Phasellus suscipit, leo a
pharetra
10 condimentum, lorem tellus eleifend magna, <a href="index.html">eget fringilla velit</a>
magna id neque. Curabitur vel urna
11 In tristique orci porttitor ipsum. Lorem ipsum dolor sit amet, consectetuer adipiscing
elit. Donec libero. Suspendisse bib
12 Cras id urna. Morbi tincidunt, orci ac convallis aliquam, lectus turpis varius lorem,
eu
13 posuere nunc justo tempus leo.</p>
14 <p class="postmeta">
15 <a href="index.html" class="readmore">Read more</a> |
16 <a href="index.html" class="comments">Comments (3)</a> |
17 <span class="date">August 20, 2009</span>
18 </p>
19 </div>
```

If we now open the browser and go to the following address:

http://localhost:3570/index.php?page=services

We will see how the same template is displayed, but the content of the page changes.

Now we only need to repeat the process for the other pages and enter them into the database.

TIDYING IT ALL UP

We are going to make a few small changes to complete the website and make it more user-friendly. In this case, we will change the template heading to do two things:

- Show the links to the pages in our corporate website.
- Highlight the correct link for the active page.

For this, we need to change the <ul> list inside in index.html, so that it looks like this:

```
1 <div id="nav">
2 <ul>
3 <li id="{%$START%}"><a href="index.php">Start</a></li>
4 <li id="{%$SERVICES%}"><a href="index.php?page=services">Services</a></
5 <li id="{%$CUSTOMERS%}"><a href="index.php?page=customers">Customers</a></li>
6 <li id="{%$CONTACT%}"><a href="index.php?page=contact">Contact</a></li>
7 <li id="{%$COMPANY%}"><a href="index.php?page=company">Company</a></li>
8 </ul>
9 </div>
```

This will enable us to see the links to all the pages we want at the top and we will also be able to specify which of them is to be highlighted by setting the page name space to "current". To do this we must add the following line to the end of the OnTemplate event:

```
1 $template->_smarty->assign(strtoupper($page),'current');
```

Doing it like this makes it very simple for us to add pages in future by following the same procedure.

DEPLOYMENT

For our final step we are going to look at the tasks necessary to deploy a Delphi for PHP application. We can use the deployment wizard or create the installation ourselves.

In either case, installation is very simple, really only involving copying files. The only issue we need to pay attention to is the location of the folder that contains VCL for PHP.

By default, applications are prepared to work with the class library as a subdirectory of the application itself. That is, is our application was in /var/www/mywebsite.com, the VCL must be in /var/www/mywebsite.com/vcl. In this way, we do not need to alter the search path at all.

Whatever the case, VCL for PHP must be in a directory whose content is accessible to the web server, since several components use resources (images, style-sheets etc.) which are stored in the VCL folder.

For this example, we will put VCL for PHP in a sub-folder of our application.

First we need to create the database in the target directory. For this, we can make a dump of the database that we have locally and then recover it using the dump on the server.

Then we can use the deployment wizard, which will guide us through the whole process of copying the necessary files from a folder on our hard disk. It is important to point out that the wizard does not copy files referenced by component properties, but only those files that make up the project and the required files from VCL for PHP.

Therefore, if we want the deployment wizard to include our template, we need to add to the project all the files that comprise it.

First the deployment wizard collects all the files that make up our application. Second, it identifies the components used on our forms and the source code of the corresponding VCL for PHP. The third step lists the files and folders to be copied and finally it asks for a folder where they will be placed.

These are all the files necessary for our application to function on a computer other than the one used to develop it. I advise you to copy the complete VCL for PHP since this will not change and we can always include new components. Then it is only necessary to copy or update the application files to test it out.

Once these files have uploaded to the server, we can change the database access configuration so that it matches with that of the server. For this, we can edit the file dbmodule.xml.php, which is a text file with a simple structure. We need to find the connection properties of the Database component and change them with the right values.

```
<object class="Database" name="dbweb1" >
<property name="Left">285</property>
<property name="Top">163</property>
<property name="Connected">1</property>
<property name="DatabaseName">web</property>
<property name="Host">localhost</property>
<property name="Name">dbweb1</property>
<property name="UserName">root</property>
<property name="UserPassword">test</property>
</object>
```

That's it! Our application should now work perfectly on the server. I advise you to use Linux servers, since their performance when running PHP is much better than Windows. In any case, you should always use Apache as a web server with PHP loaded as a module rather than a CGI.

INTERACTING WITH DELPHI FOR WINDOWS

WEB SERVICE

One of the best areas for which we can use Delphi for PHP is creating web service applications, which provide information and carry out operations on the server and communicate with a Windows client developed in Delphi for Windows.

For this example, we will create a web service which searches a database for a customer and returns the status of a fictional credit account.
If there is more than one customer that matches the search, it will return a list of the matching customers so that the user can choose.

CREATION OF THE DATABASE

As in the previous example, we will create a database and a table where the customers and their credit account status will be stored.

```
CREATE TABLE `webservice`.`customers` (
`id` INT( 11 ) NOT NULL AUTO_INCREMENT PRIMARY KEY ,
`firstname` VARCHAR( 40 ) NOT NULL ,
`lastname` VARCHAR( 40 ) NOT NULL ,
`credit` FLOAT( 10, 2 ) NOT NULL DEFAULT '0'
) ENGINE = INNODB;
```

Now we enter several dummy records so that we have data we can use to perform specific tests.

```
1 INSERT INTO `webservice`.`customers`
2 (`id`, `firstname`, `lastname`, `credit`)
3 VALUES
4 (NULL, 'Jane', 'Smith', '200'), (NULL, 'Joe', 'Black', '400');
```

DEVELOPMENT OF THE WEB SERVICE

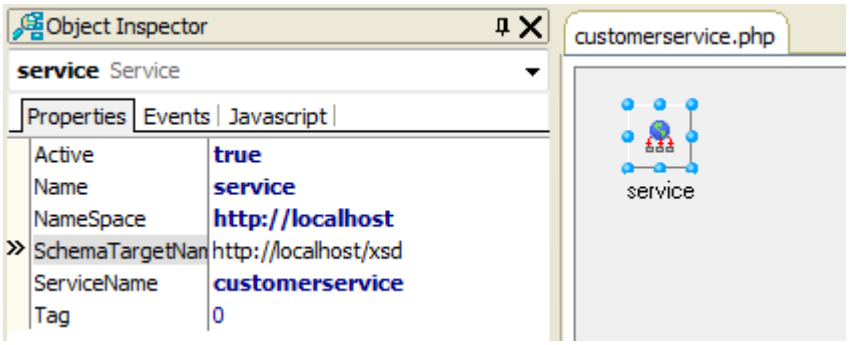
We create a new application using File | New | Application and since we will not need a form, we close the new page that has been created, answering "No" to the question about saving our changes or not.

Our web service, like the data access components that we are going to use, are non-visual components, so a data module is more than sufficient for this project. So, using File | New | Data Module, we create a new one.

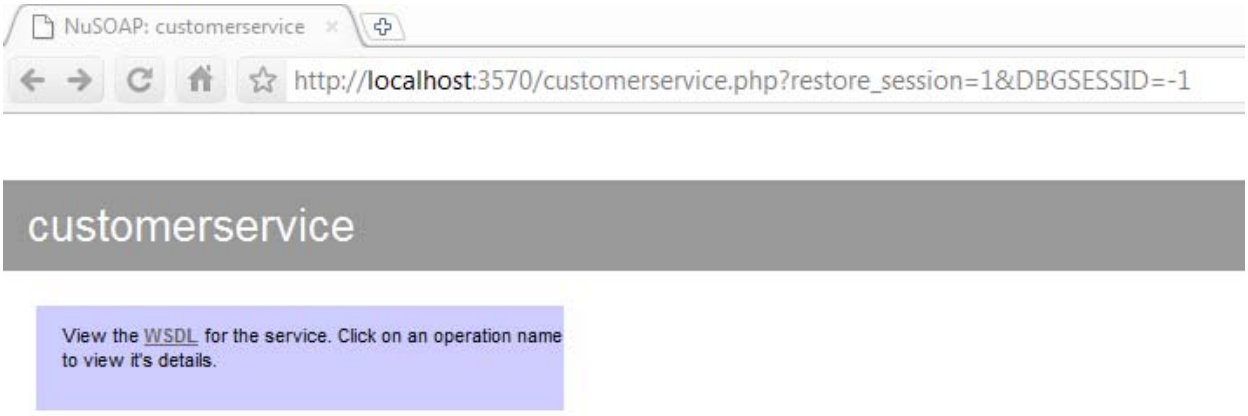
Now we can save the whole application in the directory we want and give names to the files. For example, we could call the datamodule "customerservice" since it's highly possible that we will add functionality to it in the future.

To create a web service we only need one component, called Service, which we find on the component palette and then place in the data module.
We change its ServiceName to "customerservice" and set its Active property to "true".

This is what we should now have in the IDE:



If we now run the application, we will see how this component already provides some basic functionality by providing a simple front end which will interact with our future web service.



Now we need to provide the web service with some functionality, for which we must register the routines and procedures that we are going to make available to the web service customers.

Initially there will only be one routine, named getCredit, which will receive an input string with the name or part of the name of a customer and will return an array of strings. If there is only one string in the array, it will be the credit status of the customer, but if there are more than one, they will be the names of the customers that match the search.

To return an array of strings, we need add a complex data type. For this we use the event OnAddComplexTypes of the Service component and call the addComplexType method.

```
01 function serviceAddComplexTypes($sender, $params)
02 {
03 $this->service->addComplexType
04 (
05 'ArrayOfstring',
06 'complexType',
07 'array',
08 '',
09 'SOAP-ENC:Array',
10 array(),
11 array(array('ref'=>'SOAP-ENC:arrayType','wsdl:arrayType'=>'string[]')),
12 'xsd:string'
13 );
14 }
```

Now that we have defined the complex data type, we need to register the function that will be published by the web service. This is done in the event OnRegisterServices.

```
1 function serviceRegisterServices($sender, $params)
2 {
3 $this->service->register('getCredit',
4 array('input'=>'xsd:string'),
5 array('return'=>'tns:ArrayOfstring'),
6 'http://localhost/');
```

```
7 }
```

Once we have registered our function we need to implement it. Let's remind ourselves what we need to do:

- Search the customers table for a customer whose name matches the input parameter.
- Return all the matching records as an array of strings.

Here you can see the implementation of the function:

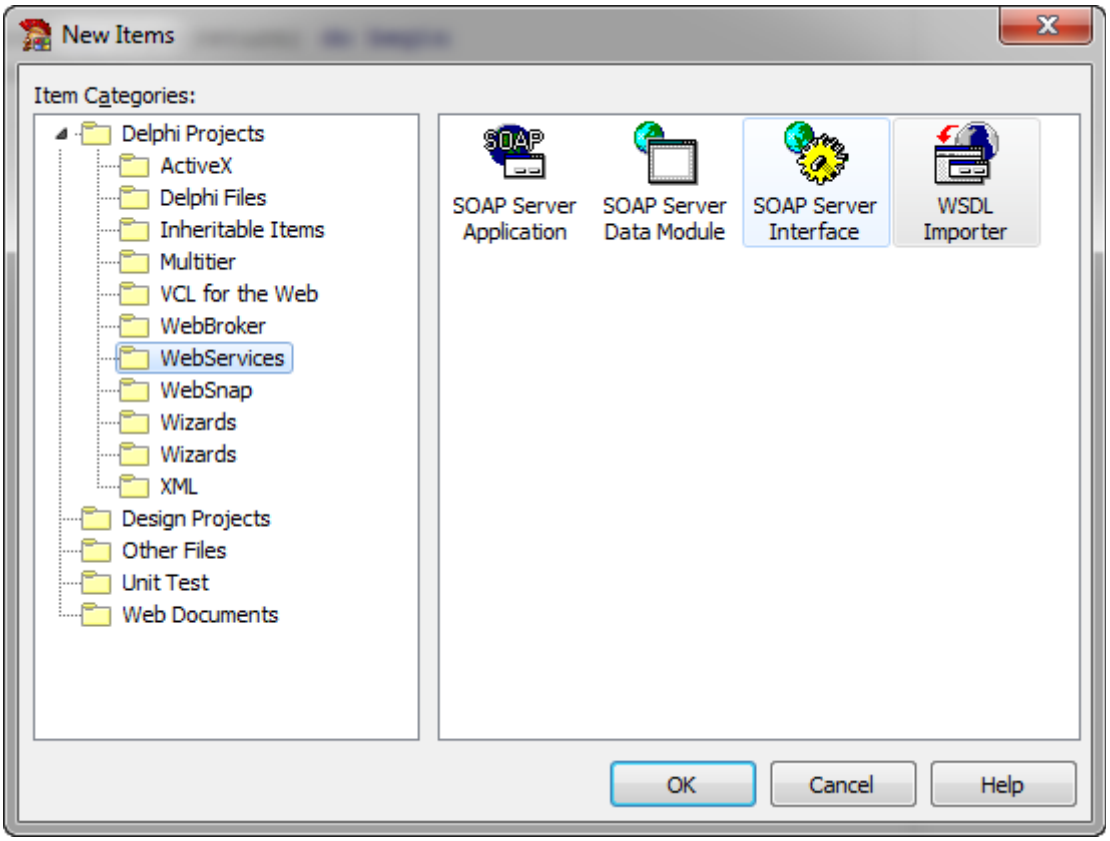
```
01 function getCredit($input)
02 {
03 global $Webservice;
04
05 $Webservice->tbcustomers1->close();
06 $Webservice->tbcustomers1->Filter="concat(firstname,' ',lastname) like '%$input%'"
07 $Webservice->tbcustomers1->open();
08 $result=array();
09
10 while (!$Webservice->tbcustomers1->EOF)
11 {
12 $result[]=$Webservice->tbcustomers1->firstname.' '.
13 $Webservice->tbcustomers1->lastname.'|'.
14 $Webservice->tbcustomers1->credit;
15 $Webservice->tbcustomers1->next();
16 }
17
18 return($result);
19 }
```

If we now run our project the WSDL description already provides all the information necessary to connect and call this service.

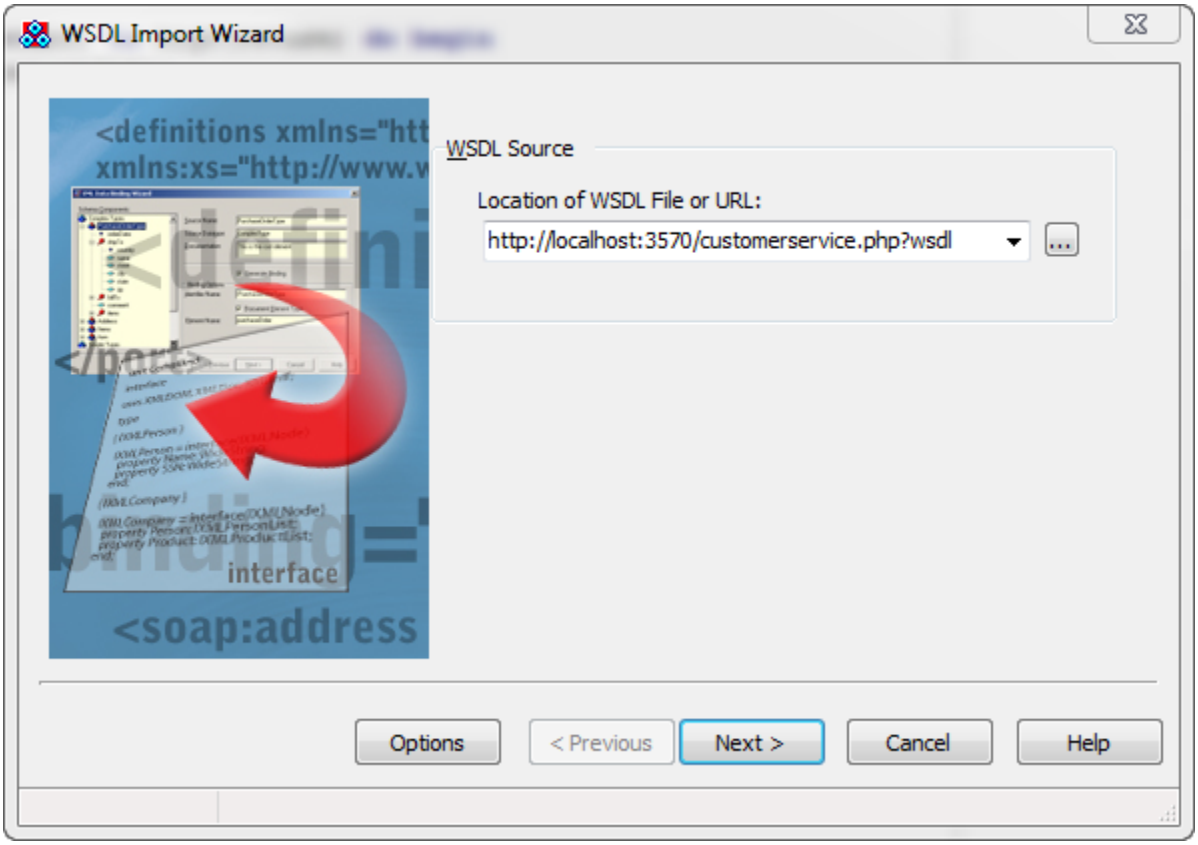
DEVELOPMENT OF THE CLIENT

To develop the client we will use Delphi for Windows but it should be noted that any development tool with SOAP support can be used to consume our service.

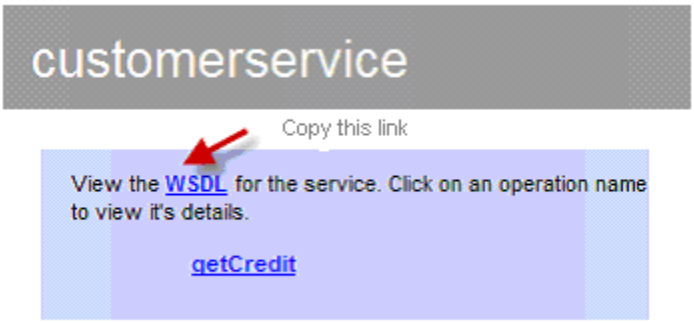
First we need to create a new application with Delphi for Windows. Once it is created we will use the WSDL Import Wizard which is in the New Items dialog box in the WebServices category.



When it is run, the wizard asks for the URL of the WSDL that we want to import. This WSDL is accessible through a link published by the service itself.

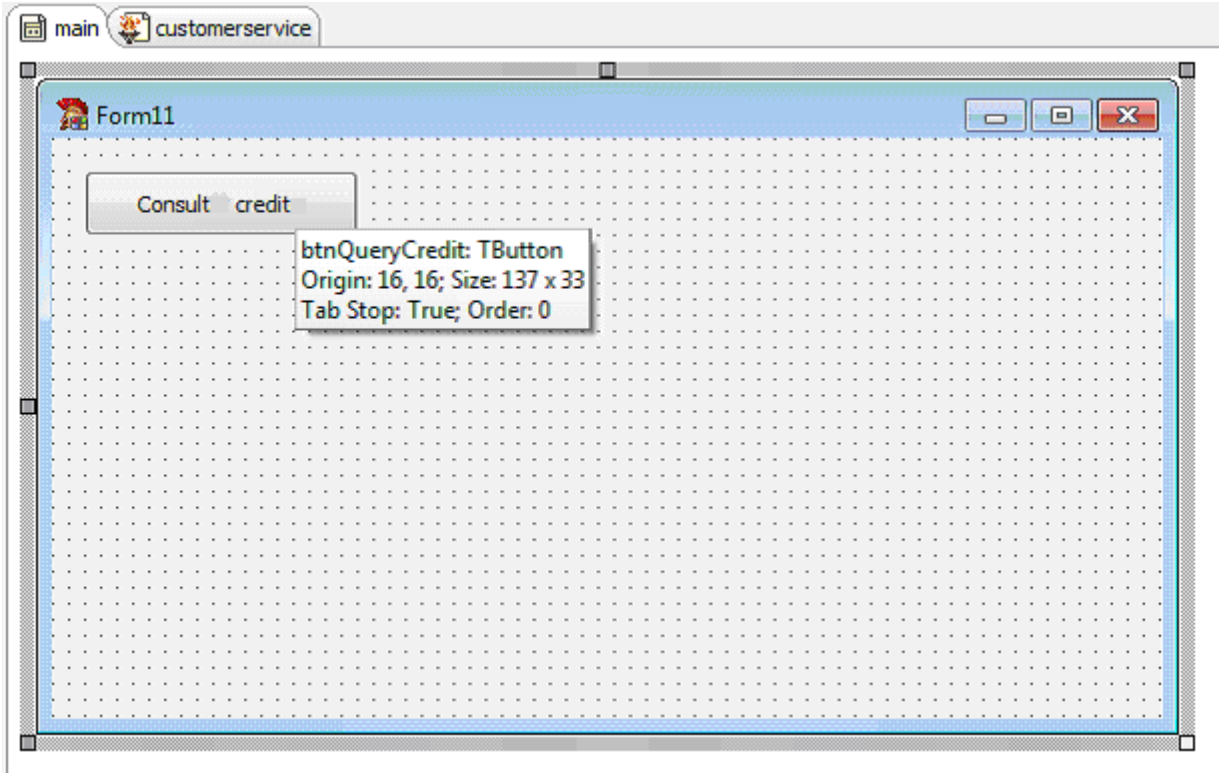


The URL of the WSDL is held in the description of our web service:



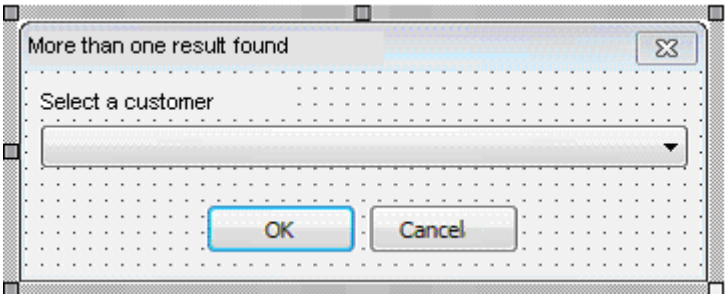
Once we have the link on the clipboard paste it in the input field on the wizard and click Next. We will see a form with the detected data types and the code unit to be added to the project to access the web service.

If we click the Finish button the unit will be added to our project. Now we are going to see how to use it. For this, we put a TButton on the form and we call it btnQueryCredit.



As we saw in the requirements, it is possible that more than one result will be returned, if the search string matches more than one customer. For this reason we must add another form that we will only use to ask for a customer to be selected.

The form consists of a drop-down list and two buttons. It is a normal modal form so I won't go into any detail. Here is a screen capture:



Now on the form that calls the web service, we need to include in the "uses" clause the unit "customerservice" which is the one that contains the code we need to call and also the unit that contains our selection form. Next, we generate the OnClick event for the button and we write this code:

```
01 procedure TMainForm.btnQueryCreditClick(Sender: TObject);
02 var
03   value: string;
04   return: ArrayOfstring;
05   i: integer;
06   customername: string;
07   k: integer;
08 begin
09   //First we request the name of a customer to search
10   if (InputQuery('Consult credit','Customer:',value)) then begin
11
12     //The call is made to the web service
13     return:=GetcustomerservicePortType.getCredit(value);
14
15     //If only one result is returned, we display the credit
16     if (length(return)=1) then begin
17       showmessage('Customer credit:'+return[0]);
18     end
19     else if (length(return)>1) then begin
20
21       //If there is more than one then ask which one the user wants to search
22       with TfrmSelectCustomer.Create(nil) do begin
23         try
24           cbCustomers.Clear;
25
26           //We fill the drop-down list
27           for i := low(return) to high(return) do begin
28             customername:=return[i];
29             k:=pos('|',customername);
30             if (k<>0) then customername:=copy(customername,1,k-1);
31             cbCustomers.Items.Add(customername);
32           end;
33
34           cbCustomers.ItemIndex:=0;
35           if (showmodal=mrOK) then begin
36             //If the user accepts the operation we call the web service again
37             //it is redundant, but it's only an example ;-)
38             return:=GetcustomerservicePortType.getCredit(cbCustomers.Text);
39             showmessage('Customer credit:'+return[0]);
40           end;
41         finally
42           free;
43         end;
44       end;
45     end
46     else begin
47       showmessage('No customers were found');
48     end;
49   end;
50 end;
```

If we now run our application, we will see that it works as expected.

- If we ask for the credit status of a customer who is not in our database, we will get a message telling us this.
- If we ask for the credit status of a customer who is in the database, we will see his credit status.
- If the search string matches several customers, we will see a screen telling us to select one of the customers and if we do this we will see the credit status of the customer we select.

The possibilities for this type of system are limitless. We can create web services that interact with the server, carry out complex requests and return the right data to lightweight client applications. Changes or updates can be made to the web services without having to update the clients. Data can be sent to different clients on different platforms with the same backend etc.

CONCLUSION

It is very difficult to describe in a single document all the potential of Delphi for PHP and the benefits to be derived from its use. I have tried to give a global vision of what the product is and, above all, from a practical perspective, I hope it will be of some help to you in your decision-making. If you are already a Delphi for PHP user, maybe you have found information of use for your projects.

Delphi for PHP is the first and only purely visual development tool for PHP. It incorporates the RAD technologies which have been so beneficial to Delphi for Windows programmers.

ABOUT THE AUTHOR

José León is the main architect of Delphi for PHP. He started to write a first version around 2001 and in 2007 formed an agreement with Embarcadero (or CodeGear as it was then called) to develop Delphi for PHP as one of the premier products in the Embarcadero catalogue.

Additionally, he is the CEO of Qadram Software S.L., a software development company, based in Spain, that uses Delphi for PHP in the solutions it develops for its customers.

You can visit his company's website - http://www.qadram.com , you can follow the evolution of Qadram's software development at http://twitter.com/qadram and contact José himself at support@qadram.com

José wishes to thank Paweł Głowacki for his interest in Delphi for PHP and his unlimited capacity for communication.

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.