PHP4Delphi 6.0

PHP Extensions Development Framework

1. Introduction

PHP, which stands for "PHP: Hypertext Preprocessor" is a widely-used Open Source general-purpose scripting language that is especially suited for Web development and can be embedded into HTML. Its syntax draws upon C, Java, and Perl, and is easy to learn. The main goal of the language is to allow web developers to write dynamically generated WebPages quickly, but you can do much more with PHP.

PHP4Delphi 6.0 is a Visual Development Framework for creating custom PHP Extensions using Delphi. PHP extension, in the most basic of terms, is a set of instructions that is designed to add functionality to PHP.

PHP4Delphi also allows executing the PHP scripts within the Delphi program directly from file or memory. You can read and write global PHP variables and set the result value.

PHP4Delphi allows you to embed the PHP interpreter into your Delphi application so you can extend and customize the application without having to recompile it.

PHP is freely available from http://www.php.net/
For more information on the PHP Group and the PHP project, please see http://www.php.net.

PHP4Delphi is organized into the following subprojects:

- *PHP scripting* (using php in Delphi applications)

PHP4Delphi allows to execute the PHP scripts within the Delphi program directly without a WebServer.

- *PHP extensions development framework* (using Delphi to extend PHP functionality)

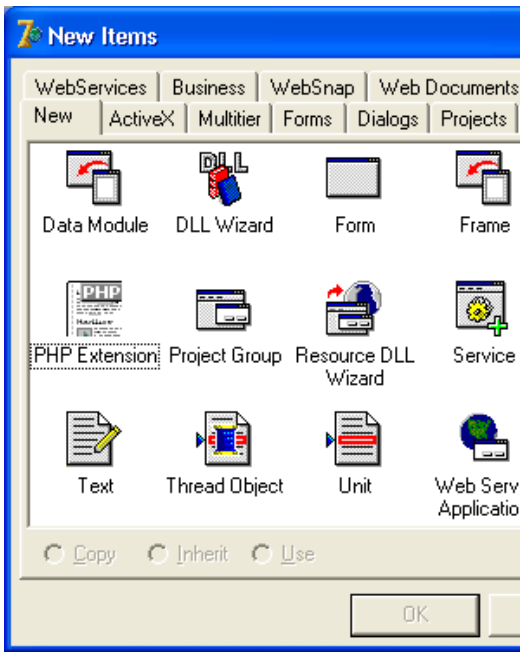
Visual Development Framework gives possibility to create custom PHP Extensions using Delphi.

- *PHP4Applications* (integrate PHP in any application)

Supports C, C++, Visual Basic, VBA, etc…

2. Creation PHP Extension

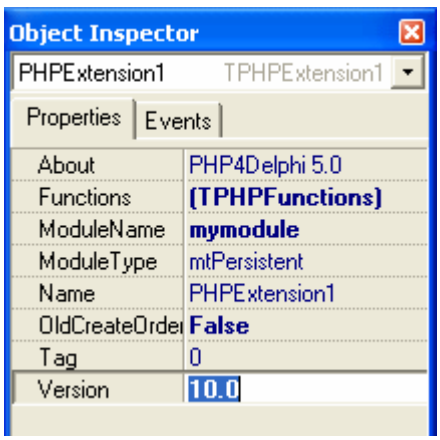
The creation of a PHP Extension DLL is similar to the development of any standard DLL. For this purpose it is necessary to load Delphi, in the menu **File** choose item **New**, then in a New Items dialog box choose an icon *PHP Extension* and to press the **OK** button.



PHP4Delphi provides a full design time environment for the development of PHP Extensions. A special Data Module PHPExtension is added to your new project; You can place any non-visual controls in it and work with them.

2.1. TPHPExtension properties

The main properties of PHPExtension module are:



- **Name:** Contains the module name (for example, "File functions", "Socket functions", "Crypt", etc.). This name will show up in *phpinfo()*, in the section "Additional Modules."

- **Version:** The version of the module. You can use NO_VERSION_YET if you don't want to give the module a version number yet, but we really recommend that you add a version string here. Such a version string can look like this (in chronological order): "2.5-dev", "2.5RC1", "2.5" or "2.5pl3".

- **Functions:** Contains all the functions that are to be made available externally, with the function's name as it should appear in PHP

2.2. TPHPExtension Events

- **OnActivation:** Occurs when the Extension module is activated. Write an OnActivation event handler to perform any initializations when the Extension module is first activated. The Extension module is first activated when application starts and is also activated on every PHP request.

- **OnCreate:** Occurs when an application instantiates the data module. Write an OnCreate event handler to take specific actions when an application instantiates the data module. For example, if the data module contains database and dataset components, an application may establish a database connection immediately.

- **OnDeactivation:** Occurs when the Extension module is deactivated. Write an OnDeactivation event handler to perform any final cleanup when the Extension module is deactivated. PHPExtension modules are deactivated after the PHP request has been processed and a custom PHP function returns a result.

- **OnDestroy:** Occurs when the data module is about to be destroyed. Write an OnDestroy event handler to take specific actions when an application frees a data module. For example, if the unit code for the data module instantiates any objects of its own, such as a TStringlist, the OnDestroy event handler can be used to free those objects.

- **OnModuleInfo:** When *phpinfo()* is called in a script, Zend cycles through all loaded modules and calls this function. Every module then has the chance to print its own "footprint" into the output page. Generally this is used to dump environmental or statistical information.

- **OnModuleInit:** This event occurs once upon module initialization and can be used to do one-time initialization steps (such as initial memory allocation, etc.).

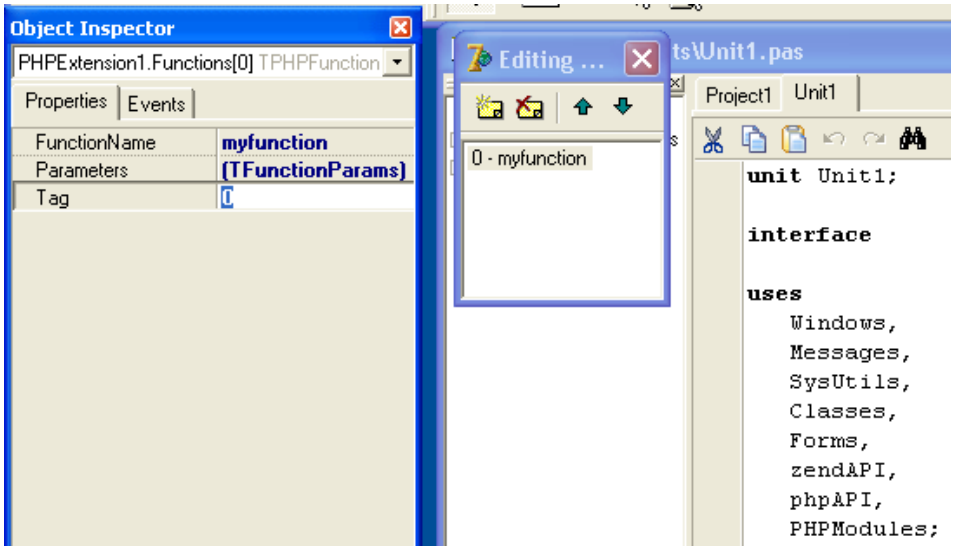
- **OnModuleShutdown:** This event occurs once upon module shutdown and can be used to do one-time deinitialization steps (such as memory deallocation). This is the counterpart to **OnModuleInit** event.

- **OnRequestInit:** This event occurs once upon every page request and can be used to do one-time initialization steps that are required to process a request. *Note:* As dynamic loadable modules are loaded only on page requests, the request startup function is called right after the module startup function (both initialization events happen at the same time).

- **OnRequestShutdown:** This event occurs once after every page request and works as counterpart to **OnRequestInit** event. *Note:* As dynamic loadable modules are loaded only on page requests, the request shutdown function is immediately followed by a call to the module shutdown handler (both deinitialization events happen at the same time).

2.3. Add functions

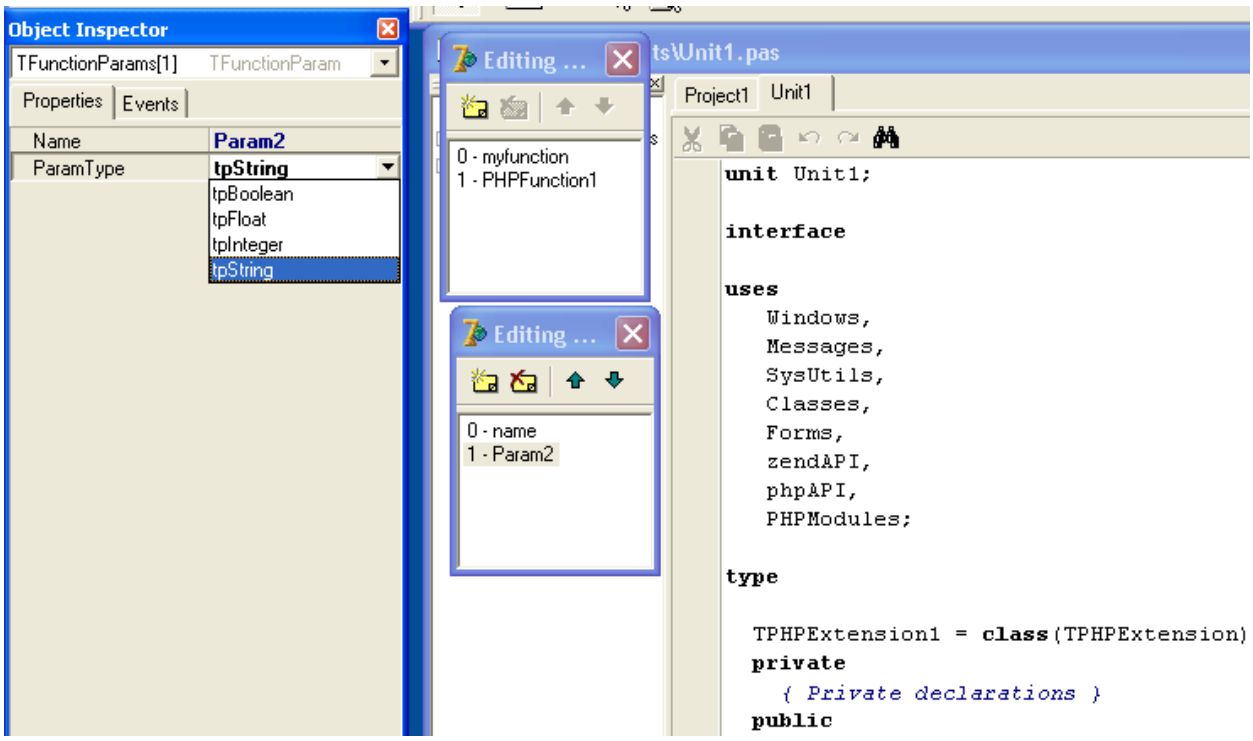
Now is time to add some new functions to your PHP Extension.



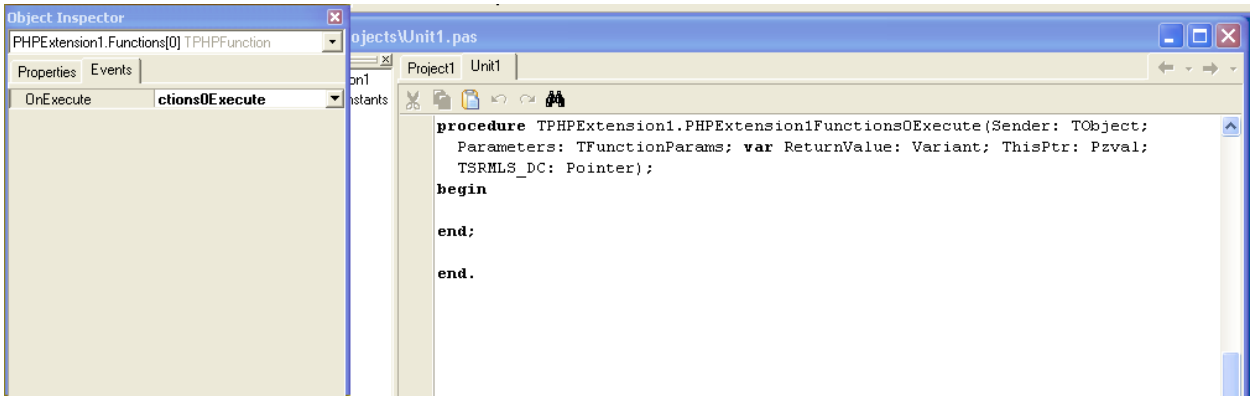
TPHPFunction properties:

- **Name:** Denotes the function name as seen in PHP (for example, fopen, mysql_connect, or, in our example, *myfunction*).

- **Parameters:** Contains the collection of function parameters. Each parameter object in the collection represents an individual parameter. Use Items to access a particular parameter. Index indicates the specific parameter to access. Index identifies the parameter's position in the collection of parameters, in the range 0 to Count - 1.



TFunctionParam represents a parameter. Use the *Name* property to identify a particular parameter within the TFunctionParams object.
The *ParamType* property indicates the datatype of the value the parameter represents.



Next write an **OnExecute** event handler to implement a response to function call sent by the PHP script.

Example:

```
procedure TPHPExtension1.PHPExtension1FunctionsOExec
  Parameters: TFunctionParams; var ReturnValue: Vari
  TSRMLS_DC: Pointer);
begin
  ReturnValue := 'Hello, Delphi';
end;

end.
```

Your PHP extension is now ready. Just compile and use.

To test your project you can use PHP script like this:

```
<?
if(!extension_loaded('extname')) {
        dl('skeleton.dll');
}


$str = confirm_extname_compiled("skeleton");
echo "$str\n";

$module = 'extname';

if (extension_loaded($module)) {
  $str = "module loaded";
} else {
        $str = "Module $module is not compiled into PHP";
}
echo "$str\n";
```

```
$functions = get_extension_funcs($module);
echo "Functions available in the $module extension:<br>\n";
foreach($functions as $func) {
   echo $func."<br>\n";
}

?>
```

3. PHP Extensions – classical way

You can also build PHP extensions in the classical way – using the ZEND API.
You can find the ZEND API documentation here: http://www.zend.com/apidoc/

We'll start with the creation of a very simple extension at first. This basically does nothing more than implement a function that returns a string.

```
library skeleton;

uses
 Windows, SysUtils, zendTypes, ZENDAPI, phpTypes, PHPAPI;


function rinit (_type : integer; module_number : integer; TSRMLS_DC :
pointer) : integer; cdecl;
begin
  Result := SUCCESS;
end;

function rshutdown (_type : integer; module_number : integer; TSRMLS_DC :
pointer) : integer; cdecl;
begin
  Result := SUCCESS;
end;

procedure php_info_module(zend_module : Pzend_module_entry; TSRMLS_DC :
pointer); cdecl;
begin
  php_info_print_table_start();
  php_info_print_table_row(2, PChar('extname support'), PChar('enabled'));
  php_info_print_table_end();
end;

function minit (_type : integer; module_number : integer; TSRMLS_DC :
pointer) : integer; cdecl;
begin
  RESULT := SUCCESS;
end;

function mshutdown (_type : integer; module_number : integer; TSRMLS_DC :
pointer) : integer; cdecl;
begin
  RESULT := SUCCESS;
end;

procedure confirm_extname_compiled (ht : integer; return_value : pzval;
this_ptr : pzval;
   return_value_used : integer; TSRMLS_DC : pointer); cdecl;
var
```

```
arg : PChar;
str : string;
param : pzval_array;
begin
  if ( not (zend_get_parameters_ex(ht, Param) = SUCCESS )) then
    begin
      zend_wrong_param_count(TSRMLS_DC);
      Exit;
    end;
   arg := param[0]^.value.str.val;
   str := Format('Congratulations! You have successfully modified
ext/%.78s/config.m4. Module %.78s is now compiled into PHP.', ['extname',
arg]);
   ZVAL_STRING(return_value, PChar(str), true);
   dispose_pzval_array(param);
end;


var
  moduleEntry : Tzend_module_entry;
  module_entry_table : array[0..1]  of zend_function_entry;


function get_module : Pzend_module_entry; cdecl;
begin
  if not PHPLoaded then
    LoadPHP;
  ModuleEntry.size := sizeof(Tzend_module_entry);
  ModuleEntry.zend_api := ZEND_MODULE_API_NO;
  ModuleEntry.zts := USING_ZTS;
  ModuleEntry.Name := 'extname';
  ModuleEntry.version := '0.0';
  ModuleEntry.module_startup_func := @minit;
  ModuleEntry.module_shutdown_func := @mshutdown;
  ModuleEntry.request_startup_func := @rinit;
  ModuleEntry.request_shutdown_func := @rshutdown;
  ModuleEntry.info_func := @php_info_module;
  Module_entry_table[0].fname := 'confirm_extname_compiled';
  Module_entry_table[0].handler := @confirm_extname_compiled;
  Module_entry_table[0].func_arg_types := nil;
  ModuleEntry.functions :=  @module_entry_table[0];
  ModuleEntry._type := MODULE_PERSISTENT;
  Result := @ModuleEntry;
end;



exports
  get_module;

end.
```

This code contains a complete PHP module.

All PHP modules follow a common structure:

- Declaration of exported functions (required to declare the Zend function block)
- Declaration of the Zend function block
- Declaration of the Zend module block

- Implementation of get_module()
- Implementation of all exported functions

To declare functions that are to be exported (i.e., made available to PHP as new native functions), you have to add procedures with the following declaration:

```
procedure <procedure name> (ht : integer; return_value : pzval; this_ptr :
pzval; return_value_used : integer; TSRMLS_DC : pointer); cdecl;
```

Parameter	Description
Ht	The number of arguments passed to the Zend function.
Return_value	This variable is used to pass any return values of your function back to PHP.
This_ptr	Using this variable, you can gain access to the object in which your function is contained, if it's used within an object.
Return_value_used	This flag indicates whether an eventual return value from this function will actually be used by the calling script. 0 indicates that the return value is not used; 1 indicates that the caller expects a return value. Evaluation of this flag can be done to verify correct usage of the function as well as speed optimizations in case returning a value requires expensive operations
TSRMLS_DC	This variable points to global settings of the Zend engine. You'll find this useful when creating new variables, for example

Now that you have declared the functions to be exported, you also have to introduce them to Zend. Introducing the list of functions is done by using an array of zend_function_entry. This array consecutively contains all functions that are to be made available externally, with the function's name as it should appear in PHP and its name as defined in the Delphi source.

```
zend_function_entry = record
    fname : Pchar;
    handler : pointer;
    func_arg_types : Pbyte;
 end;
```

Entry	Description
Fname	Denotes the function name as seen in PHP (for example, fopen, mysql_connect, or, in our example, first_module).
Handler	Pointer to the Delphi function responsible for handling calls to this function
Func_arg_types	Allows you to mark certain parameters so that they're forced to be passed by reference. You usually should set this to Nil.

You can see that the last entry in the list always has to be {Nil, Nil, Nil}. This marker has to be set for Zend to know when the end of the list of exported functions is reached.

4. Setup

PHP4Delphi is a Delphi interface to PHP. It works with Delphi 5, 6 and 7.
PHP4Delphi allows you to execute PHP scripts from within your Delphi program directly, without needing a WebServer. PHP4Delphi also contains the PHP API and ZEND API and a visual development framework for PHP extensions.

This is a source-only release of php4Delphi. It includes designtime and runtime packages for Delphi 5 through 7.

Before using php4Delphi library:

If you do not have PHP installed, you have to download and install PHP separately. It is not included in the package. You can download the latest version of PHP from http://www.php.net/downloads.php

ZEND API documentation available at http://www.zend.com
PHP API documentation available at http://www.php.net

You need to ensure that the dlls which php uses can be found. php4ts.dll is always used. If you are using any php extension dlls then you will need those as well. To make sure that the dlls can be found, you should copy them to your system directory (e.g. winnt/system32 or windows/system).

1. Delphi 5.x:

Uninstall any previously installed versions of the php4Delphi Library from your Delphi 5 IDE. You should also remove any previously compiled php4Delphi packages from your hard disk.

Select PHP version you are going to use. php4Delphi supports PHP 4.x and PHP 5.x, but not at the same time. You have to compile php4Delphi for selected target version of PHP.

Open PHP.INC file.
If you are using PHP5:
a) Comment (remove) PHP4 directive {$DEFINE PHP4}
b) Uncomment (remove dot) directive {$DEFINE PHP5}
c) Save PHP.INC file

If you are using PHP4:
a) Comment (remove) PHP5 directive {$DEFINE PHP5}
b) Uncomment (remove dot) directive {$DEFINE PHP4}
c) If you are using PHP version 4.2.x...4.3.0 add {$DEFINE PHP430} and remove {$DEFINE PHP433}
If you are using PHP version 4.3.3...4.3.x add {$DEFINE PHP433} and remove {$DEFINE PHP430}
d) Save PHP.INC file

Use the "File\Open..." menu item in the Delphi IDE to open php4Delphi runtime package php4DelphiR5.dpk. In "Package..." window click "Compile" button to compile the package php4DelphiR5.dpk. Put the compiled BPL file into a directory that is accessible through the

search PATH (i.e. DOS "PATH" environment variable; for example, in the Windows\System directory).

After you have compiled the php4Delphi run-time package you must install the design-time package into the IDE.

Use "File\Open..." menu item to open the design-time package php4DelphiD5.dpk.
In "Package..." window click "Compile" button to compile the package and then click "Install" button to register the php4Delphi Library components on the component palette.

2. Delphi 6.x:

Uninstall any previously installed versions of the php4Delphi Library from your Delphi 6 IDE. You should also remove any previously compiled php4Delphi packages from your hard disk.

Select PHP version you are going to use. php4Delphi supports PHP 4.x and PHP 5.x,
but not at the same time. You have to compile php4Delphi for selected target version of PHP.

Open PHP.INC file.
If you are using PHP5:
a) Comment (remove) PHP4 directive {$DEFINE PHP4}
b) Uncomment (remove dot) directive {$DEFINE PHP5}
c) Save PHP.INC file

If you are using PHP4:
a) Comment (remove) PHP5 directive {$DEFINE PHP5}
b) Uncomment (remove dot) directive {$DEFINE PHP4}
c) If you are using PHP version 4.2.x...4.3.0 add {$DEFINE PHP430} and remove {$DEFINE PHP433}
If you are using PHP version 4.3.3...4.3.x add {$DEFINE PHP433} and remove {$DEFINE PHP430}
d) Save PHP.INC file

Use the "File\Open..." menu item in the Delphi IDE to open the php4Delphi runtime package php4DelphiR6.dpk. In "Package..." window click "Compile" button to compile the package php4DelphiR6. Put the compiled BPL file into a directory that is accessible through the search PATH (i.e. DOS "PATH" environment variable; for example, in the Windows\System directory).

After compiling the php4Delphi run-time package you must install the design-time package into the IDE.

Use "File\Open..." menu item to open the design-time package php4DelphiD6.dpk.
In "Package..." window click "Compile" button to compile the package and then click "Install" button to register php4Delphi Library components on the component palette.

3. Delphi 7.x:

Uninstall any previously installed versions of the php4Delphi Library from your Delphi 7 IDE. You should also remove any previously compiled php4Delphi packages from your hard disk.

Select PHP version you are going to use. php4Delphi supports PHP 4.x and PHP 5.x,
but not at the same time. You have to compile php4Delphi for selected target version of PHP.

Open PHP.INC file.
If you are using PHP5:
a) Comment (remove) PHP4 directive {$DEFINE PHP4}
b) Uncomment (remove dot) directive {$DEFINE PHP5}
c) Save PHP.INC file

If you are using PHP4:
a) Comment (remove) PHP5 directive {$DEFINE PHP5}
b) Uncomment (remove dot) directive {$DEFINE PHP4}
c) If you are using PHP version 4.2.x...4.3.0 add {$DEFINE PHP430} and remove {$DEFINE PHP433}
If you are using PHP version 4.3.3...4.3.x add {$DEFINE PHP433} and remove {$DEFINE PHP430}
d) Save PHP.INC file

Use the "File\Open..." menu item of your Delphi IDE to open the php4Delphi runtime package php4DelphiR7.dpk. In "Package..." window click "Compile" button to compile the package php4DelphiR7.dpk. Put the compiled BPL file into a directory that is accessible through the search PATH (i.e. DOS "PATH" environment variable; for example, in the Windows\System directory).

After compiling the php4Delphi run-time package you must install the design-time package into the IDE.

Use "File\Open..." menu item to open the design-time package php4DelphiD7.dpk
In "Package..." window click "Compile" button to compile the package and then click "Install" button to register php4Delphi Library components on the component palette.

5. Special Thanks

Daaron Dwyer
Sebastien Hordeaux
Blake Schwendiman
Colin Nelson
Vasja Klanjsek
Toby Allen
Mauro Diomelli
Kim Bracknell
Bart Libert
Peter Enz
Joao Inacio
Eddie Shipman
Thomas Weinert

and all developers who sent me comments, remarks and bug reports.

6. Support

Since this is a freeware you are strongly encouraged to look at the source code and improve on the components if you want to.
Of course I would appreciate it if you would send me back the changes and bug fixes you have made.

In case if you have questions, remarks, suggestions, visit PHP4Delphi Forum
https://sourceforge.net/forum/forum.php?forum_id=324242

When you post a question about php4Delphi please:

1. Clearly state which Delphi version you are using
2. Specify php4Delphi version
3. Specify PHP version (4 or 5)
4. Please post your question in PHP4Delphi forum instead to send direct e-mail (I have not enough time to answer all mail)
5. In case of problems, always try to use the latest version available first.

In case of doubt, download the latest version first at http://users.chello.be/ws36637

7. Distribution tips

Written by Joao Inacio

Notice: i believe this can be used to deploy apps in any machine ,with or without php already installed, without disrupting or altering in any way an already installed php.

create a php folder within you app folder:
"my app"
|- "php"

copy "php4ts.dll", "php.ini" and "Licence" files into it.
(Note: php4ts.dll should be renamed to php.dll, and changes made in "ZendAPI.pas" and PHPApi.pas" files acordingly :
change "const DllFileName: string = 'php4ts.dll' to 'php.dll';

create an "extensions" dir inside the "php" folder
"my app"
|- "php"
|- "extensions"

edit php.ini:
extension_dir = "php\extensions"

if extensions require adicional dlls, they can either be thrown in %windir% or they're directory should be added to "path" environment var.

8. FAQ

- **What means "PHP engine is not active" error ?**

This error means, that an error occurs during the activation of PHP engine by psvPHP component. For example, the component can not find php4ts.dll (php5ts.dll). You need to ensure that the dlls which php uses can be found. php4ts.dll (php5ts.dll) is always used. If you are using any php extension dlls then you will need those as well.

To make sure that the dlls can be found, you can either copy them to the system directory (e.g. winnt/system32 or windows/system).

Copy the file, php.ini-dist to your %WINDOWS% directory on Windows 95/98 or to your %SYSTEMROOT% directory under Windows NT, Windows 2000 or Windows XP and rename it to php.ini. Your %WINDOWS% or %SYSTEMROOT% directory is typically: c:\windows for Windows 95/98 c:\winnt or c:\winnt40 for NT/2000/XP servers

- **“Unable to initialize module” error**

PHP Startup: p,2|p40|: Unable to initialize module
Module compiled with module API=20020429, debug=0,thread-safety=1
PHP compiled with module API=20040412, debug=0,thread-safety=1

Module API=20020429 means that you compiled extension for PHP4, not PHP5.
PHP4 has API=20020429 and PHP5 20040412.

Open PHP.INC file, delete or comment line
{$DEFINE PHP433} and uncomment line
{$DEFINE PHP5}

After you have to rebuild php4DelphiRx and php4DelphiDx packages and build the project again. Because PHP4 and PHP5 are not compatible you can install PHP4Delphi only for one specific version of PHP.

- **How to run PHP script from a file ?**

I need the ability to run a script in an external php file from my delphi program passign it a string and returning another string. Is this possible with PHP4Delphi?
This is NOT a PHP Extension, just a regular Delphi program.

Example:

```
procedure TForm1.Button1Click(Sender: TObject);
var
  phpVariable : TPHPVariable;
```

```
begin
   phpVariable := psvPHP1.Variables.Add;
   phpVariable.Value := 'initial value';
   psvPHP1.Execute('myphpscript.php');
  ShowMessage('My variable after execution ' + phpVariable.Value);
end;
```

- **Return an array of strings from a PHP Library funciton**

I need to return an array of strings from a PHP Library funciton. I know that I have to use the _array_init ZEND API but again I'm lost as to how to do this.

```
procedure array_init(avalue : pzval);
begin
  _array_init(avalue, nil, 0);
end;

procedure add_assoc_string(avalue : pzval; akey : pchar; astring : pchar);
begin
  add_assoc_string_ex(avalue, akey, strlen(akey)+1, astring, 1);
end;

procedure TForm1.PHPLibrary1Functions0Execute(Sender: TObject;
Parameters: TFunctionParams; var ReturnValue: Variant; ThisPtr: Pzval;
TSRMLS_DC: Pointer);
 var
   res : pzval;
   key : PChar;
   value : PChar;
begin
  res := PHPLibrary1.Functions[0].ZendVar.AsZendVariable;
  array_init(res);

  key := 'name';
  value := 'Valerie';
  add_assoc_string(res, key, value);

  key := 'city';
  value := 'Leuven';
  add_assoc_string(res, key, value);

end;
```

- **How to access an components of the current form from PHP script?**

```
unit Unit1;

interface
```

```
uses
Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
Dialogs, StdCtrls, PHPCustomLibrary, phpLibrary, php4delphi, phpFunctions,
zendAPI, ZendTypes;

type
TForm1 = class(TForm)
psvPHP1: TpsvPHP;
PHPLibrary1: TPHPLibrary;
btnTarget: TButton;
btnExcute: TButton;
procedure FindAButtonExecute(Sender: TObject;
Parameters: TFunctionParams; var ReturnValue: Variant;
ThisPtr: Pzval; TSRMLS_DC: Pointer);
procedure ClickAButtonExecute(Sender: TObject;
Parameters: TFunctionParams; var ReturnValue: Variant;
ThisPtr: Pzval; TSRMLS_DC: Pointer);
procedure btnTargetClick(Sender: TObject);
procedure btnExcuteClick(Sender: TObject);
private
{ Private declarations }
public
{ Public declarations }
end;

var
Form1: TForm1;

implementation

{$R *.dfm}

procedure TForm1.FindAButtonExecute(Sender: TObject;
Parameters: TFunctionParams; var ReturnValue: Variant; ThisPtr: Pzval;
TSRMLS_DC: Pointer);
begin
ReturnValue := Integer(btnTarget);
end;

procedure TForm1.ClickAButtonExecute(Sender: TObject;
Parameters: TFunctionParams; var ReturnValue: Variant; ThisPtr: Pzval;
TSRMLS_DC: Pointer);
var
btn : integer;
begin
btn := Parameters.ParamByName('abutton').Value;
TButton(btn).Click;
end;

procedure TForm1.btnTargetClick(Sender: TObject);
begin
ShowMessage('PHP script clicked me!');
```

```
end;

procedure TForm1.btnExcuteClick(Sender: TObject);
begin
psvPHP1.RunCode('$btn = find_a_button(); click_a_button($btn);');
end;

end.


unit Unit1;

interface

uses
Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
Dialogs, StdCtrls, PHPCustomLibrary, phpLibrary, php4delphi, phpFunctions,
zendAPI, ZendTypes, DelphiFunctions;

type
TForm1 = class(TForm)
psvPHP1: TpsvPHP;
PHPLibrary1: TPHPLibrary;
btnTarget: TButton;
btnExcute: TButton;
procedure btnTargetClick(Sender: TObject);
procedure btnExcuteClick(Sender: TObject);
procedure FindAndRegExecute(Sender: TObject;
Parameters: TFunctionParams; var ReturnValue: Variant;
ThisPtr: Pzval; TSRMLS_DC: Pointer);
procedure ClickMeExecute(Sender: TObject; Parameters: TFunctionParams;
var ReturnValue: Variant; ThisPtr: Pzval; TSRMLS_DC: Pointer);
private
{ Private declarations }
public
{ Public declarations }
end;

var
Form1: TForm1;

implementation

{$R *.dfm}


procedure TForm1.btnTargetClick(Sender: TObject);
begin
ShowMessage('PHP script clicked me!');
end;

procedure TForm1.btnExcuteClick(Sender: TObject);
```

```
begin
psvPHP1.RunCode('$btn = find_and_register("btnTarget"); $btn->Caption = "Found me";
click_me($btn->tag);');
end;

procedure TForm1.FindAndRegExecute(Sender: TObject;
Parameters: TFunctionParams; var ReturnValue: Variant; ThisPtr: Pzval;
TSRMLS_DC: Pointer);
var
cnt_name : string;
cnt : TComponent;
properties : array[0..0] of pchar;
begin
cnt_name := Parameters[0].Value;
cnt := FindComponent(cnt_name);
if Assigned(cnt) then
begin
cnt.Tag := integer(cnt);
properties[0] := 'instance';
_object_init_ex(PHPLibrary1.Functions.FunctionByName('find_and_register').ZendVar.AsZend
Variable , DelphiObject, nil, 0, TSRMLS_DC );
add_property_long_ex(PHPLibrary1.Functions[0].ZendVar.AsZendVariable, properties[0],
strlen(properties[0]) + 1, Integer(cnt));
end;
end;

procedure TForm1.ClickMeExecute(Sender: TObject;
Parameters: TFunctionParams; var ReturnValue: Variant; ThisPtr: Pzval;
TSRMLS_DC: Pointer);
var
btn : TButton;
n : integer;
begin
n := Parameters[0].Value;
btn := TButton(n);
btn.Click;
end;

end.
```

- **How to dump variables after execute**

Small example how to dump variables after script was executed

```
procedure TForm1.DumpArray(ht : PHashtable);
var
buck : PBucket;
val : ppzval;
p : pointer;
S : string;
```

```
V : string;
begin
buck := ht^.pListHead;
if buck = nil then
Exit;
while buck^.pListNext <> nil do
begin
val := ppzval(buck^.pData);
buck := buck^.pListNext;
p := @Buck^.arKey;
Setlength(S, Buck^.nKeyLength-1);
Move(P^, S[1], Buck^.nKeyLength-1);
if val^^._type <> IS_ARRAY then
begin
convert_to_string(val^);
V := val^^.value.str.val;
end
else
begin
V := 'ARRAY';
end;
with ListView1.Items.Add do
begin
Caption := S;
SubItems.Add(V);
end
end;
end;

procedure TForm1.psvPHP1AfterExecute(Sender: TObject);
var
ht : PHashTable;
buck : PBucket;
begin
ht := GetSymbolsTable(psvPHP1.ThreadSafeResourceManager);
buck := ht^.pListHead;
if buck = nil then
Exit;
DumpArray(ht);
end;
```

Author: Serhiy Perevoznyk,
Belgium