PHP4Delphi 5.0

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 5.0 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.

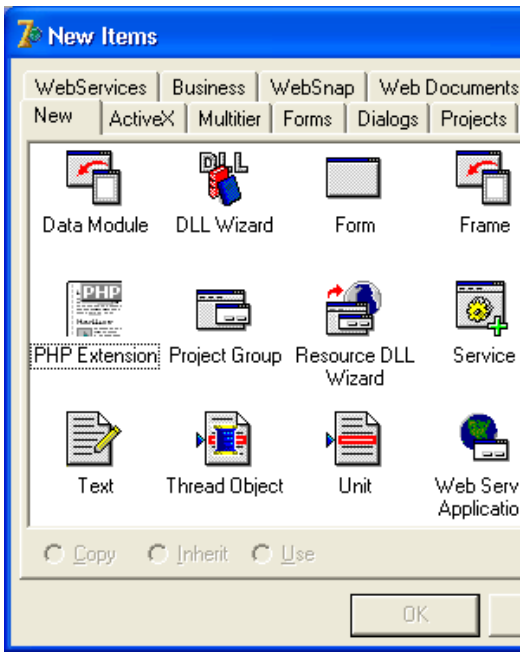
PHP is freely available from http://www.php.net/
For more information on the PHP Group and the PHP project, please see http://www.php.net.

What's new in version 5.0:

- PHP API and ZEND API converted from C to Delphi
- psvPHP component written completely in Delphi without additional C wrapper DLL
- phpLibrary component which allows to add new build-in PHP functions to psvPHP component
- New visual PHP extension development framework to create PHP extensions using Delphi.

2. Creation PHP Extension

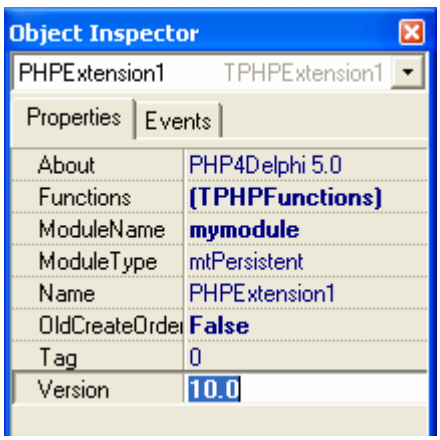
The creation PHP Extension DLL is similar to development anyone standard DLL. For this purpose it is necessary to load Delphi, in the menu **File** choose item **New**, then in a New Items dialog box choose an icon *PHP Extension* and to press the **OK** button.



You have got on interactive environment of development PHP Extensions. Your project contains the special Data Module PHPExtension; it allows to place in him various components and provides 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 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 first activated when application starts and it is activated for every PHP request.

- **OnCreate:** Occurs when an application instantiates a data module. Write an OnCreate event handler to take specific actions when an application instantiates a data module. For example, if a 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 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 string lists, 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.).

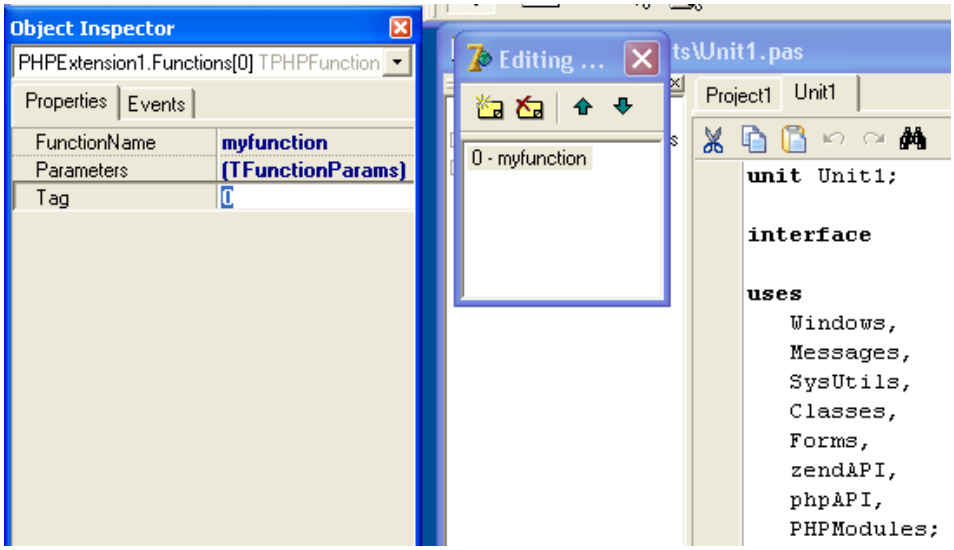
- **OnModuleShutdown:** This event occurs once upon module shutdown and can be used to do one-time deinitialization steps (such as memory deallocation). This is the counterpart to **OnModuleInit** event.

- **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

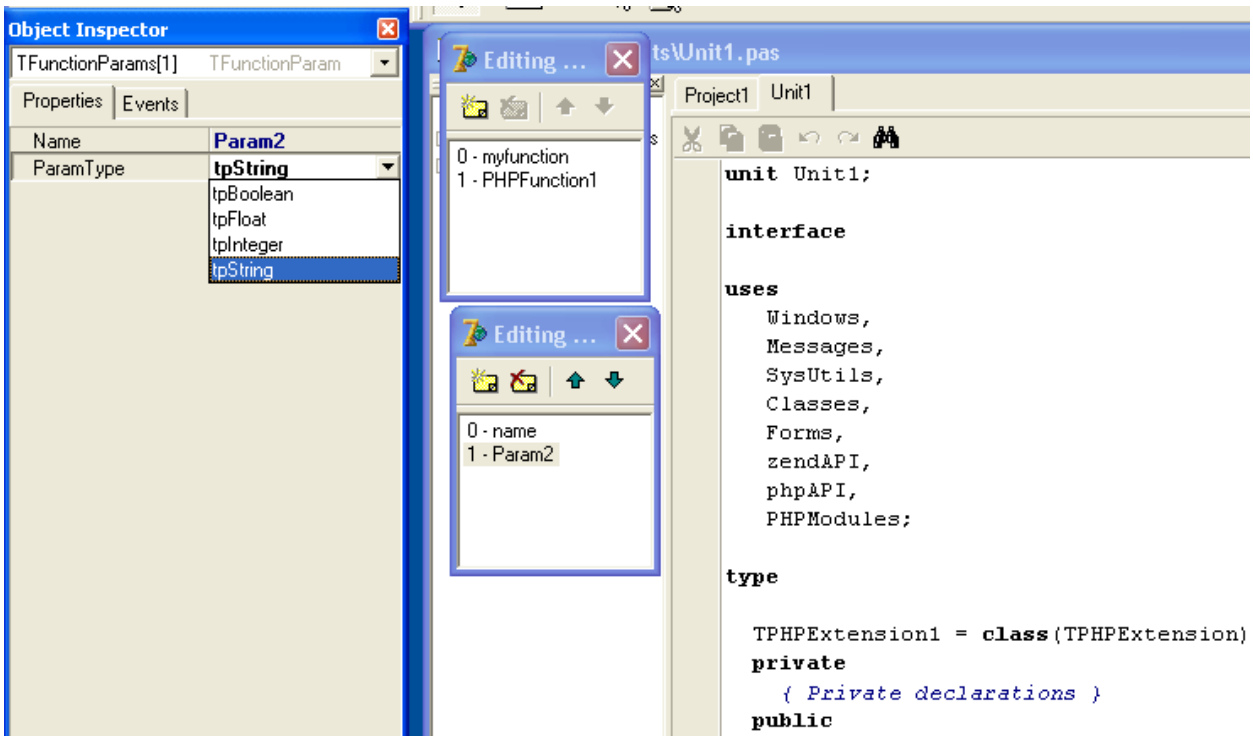
Now is time to add new functions to your PHP Extension.



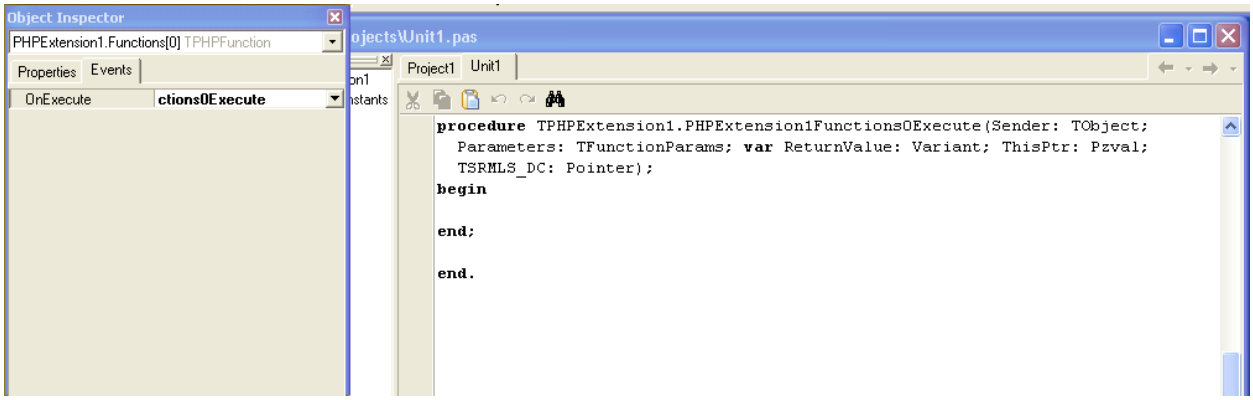
TPHPFunction properties:

- **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 *Name* property to identify a particular parameter within a TFunctionParams object.
ParamType property indicates the type of value the parameter represents.



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.
```

PHP extension is ready. Just compile it 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 build PHP extensions also in classical way – using ZEND API.
ZEND API documentation you can find here: http://www.zend.com/apidoc/

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

```
library skeleton;

uses
 Windows, SysUtils, ZENDAPI, 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 : array of pzval;
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);
end;


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


function get_module : Pzend_module_entry; cdecl;
begin
    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 for Delphi 5, 6 and 7.
PHP4Delphi allows to execute the PHP scripts within the Delphi program directly without a WebServer.
PHP4Delphi also contains PHP API and ZEND API and PHP extension visual development framework.
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 have no 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 can either copy them to the system directory (e.g. winnt/system32 or windows/system).

1. Delphi 5.x:

Uninstall previous installed version of php4Delphi Library from Delphi 5 IDE.
Remove previously compiled php4Delphi packages from your hard disk.

Use "File\Open..." menu item of Delphi IDE to open php4Delphi runtime package php4DelphiR5.dpk. In "Package..." window click "Compile" button to compile packages php4DelphiR5.dpk.
Put compiled BPL file into directory that is accessible through the search PATH (i.e. DOS "PATH" environment variable;
for example, in the Windows\System directory).

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

Use "File\Open..." menu item to open design-time package php4DelphiD5.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.

2. Delphi 6.x:

Uninstall previous installed version of php4Delphi Library from Delphi 6 IDE.
Remove previously compiled php4Delphi packages from your hard disk.

Use "File\Open..." menu item of Delphi IDE to open php4Delphi runtime

package php4DelphiR6.dpk. In "Package..." window click "Compile" button to compile packages php4DelphiR6.
Put compiled BPL file into directory that is accessible through the search PATH (i.e. DOS "PATH" environment variable; for example, in the Windows\System directory).

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

Use "File\Open..." menu item to open 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 previous installed version of php4Delphi Library from Delphi 7 IDE.
Remove previously compiled php4Delphi packages from your hard disk.

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

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

Use "File\Open..." menu item to open 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.

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.

Author: Serhiy Perevoznyk,
Belgium

Send any remarks to : serge_perevoznyk@hotmail.com
Please clearly state which Delphi version you are using and which version of the component you are using. In case of doubt, download the latest version first at http://users.chello.be/ws36637