C++ Programming

Developing Custom PHP Extensions: Part 1
Contributed by Igal Raizman
2002−07−18
[*Send Me Similar Content When Posted*]
[*Add Developer Shed Headlines To Your Site*]

DISCUSS NEWS SEND PRINT PDF

Article Index:

Ever wanted to add your own custom functionality to PHP? In this series of articles Igal will teach us how to do just that using the Microsoft Visual C++ compiler in Windows.Hello and welcome to part one of the "Developing Custom PHP Extensions" article series. After reading this article's name, one of the first things you might be asking yourself is: what exactly is a PHP extension? After knowing the answer to that question, the next one – presumably – will be: what are PHP extensions needed for? And finally (assuming you have not lost your interest), the last question will be: How do I develop my own extensions?

In this series of articles I will answer those questions in my favorite way: developing our own "useful" PHP extension from scratch. However, before we reach that goal, we must answer some of the questions mentioned above.

In this and subsequent article(s), I will assume basic knowledge of the Visual C++ compiler as well as a fairly decent knowledge of C. Basic knowledge of PHP is also required. A PHP extension, in the most basic of terms, is a set of instructions (i.e. code) that is designed to add functionality to PHP. For example, the widely used GD library (used for the creation of dynamic images) is an extension. This library added new functionality by allowing PHP to generate images on the fly. Another example is the MySQL extension, which allows us to connect to and work with MySQL databases.

What are PHP extensions needed for?
There are several reasons why extensions are needed. One of them, as stated above, is to add new functionality to PHP. For instance, where would we be today if someone did not add the functionality to work with MySQL? Where will we be tomorrow if someone does not add the functionality to work with tomorrow's databases or tomorrow's technologies? As PHP continues to grow, it is likely that new "features" will be required by the ever−growing number of web developers. Some of the new features will be popular enough to be added to the official distribution, while others will not. Either way, those extensions will serve their creators well.

On the other hand, we might use PHP extensions to improve the efficiency and speed of our programs. Some processor intensive functions might be better coded as an extension rather than straight PHP code. Since extensions are written in C (more on the actual coding later), they will work much faster than straight PHP code too.

Another possible reason to employ extensions is to reuse frequently utilized code. Instead of moving the same old functions from project to project, you could place them all in one extension and allow all your projects to utilize that extension.

How do I develop my own extensions?
Before this question can be answered, we must look at the different "types" of extensions available. Extensions come in three different flavors: Zend engine extensions, built−in extensions and external extensions.

Zend Engine extensions are extensions that are implemented right into the engine itself. For those of you who do not know, the Zend engine is what PHP is built on. It is the engine that parses, interprets and executes your PHP scripts. Changing the engine itself will change the way PHP works. Anything that will affect the language itself or its features is added to the Zend engine; this includes if statement evaluation, object orientation, mathematical expressions evaluation, etc.

Although extending the engine is possible, it's not recommenced for reasons such as incompatibility with servers that run the officially distributed engine. In other words, not too many server administrators will agree to use an unofficial version of the Zend engine.

Built−in extensions are extensions that are compiled right into PHP and are loaded with the PHP processes. The advantages of this method are: programmers aren't required to load extensions manually, and no extension files are required (since it is compiled right into the PHP binary itself). The disadvantages, on the other hand, are: any changes to the extension will require a complete re−compilation of the PHP binary itself, and the size of the binary will grow with each new extension (as will the amount of memory it will consume).

External extensions are extensions that are manually added during run time. All the functionality of the extension will be available to the script that loaded it. When the script ends, the extension is released and the memory is freed. As you might guess, the advantages are: only the extension itself needs to be re−compiled after any changes and a small PHP binary. Also, you don’t provide the functionality of your extension to scripts that do not require it. And, as always, where advantages go, disadvantages follow: extensions are loaded during run time, a process that takes time, and the programmer must remember to load the extension since it is not automatically available.

Although loading external extensions each time the script is executed takes time, it is fairly quick. I personally do not feel any speed differences when I load my external extensions. Of course, if the site receives heavy traffic, a speed difference might be apparent and built−in extensions might be the most appropriate solution. Nevertheless, in these articles we will develop an external extension. Note that the difference between built−in extensions and external extensions – code wise – is virtually nonexistent. In this tutorial I will use Microsoft's Visual C++ 6.0 compiler. Compiling extensions on Unix / Linux systems is fairly well documented. On windows, however, this isn't the case.

The first thing you must do is download the PHP source code distribution, php−4.2.1.tar.gz, from the PHP web site.

At the time of writing the latest version of PHP was 4.2.1

Once the source code is downloaded, create a directory where you will keep the source code and where you will hold the sources of your extensions. In my case, this will be D:\devarticles\php−dev\php−4.2.1\. Extract the contents of php−4.2.1.tar.gz into that directory (I used WinRAR to extract the file).

Next, move into the \ext directory, which is D:\devarticles\php−dev\php−4.2.1\ext\ in my case. There you will see all of the different extensions that are currently being distributed with PHP. Any time during your coding process when something does not work, it is a good idea to just open an extension and read through the source code.

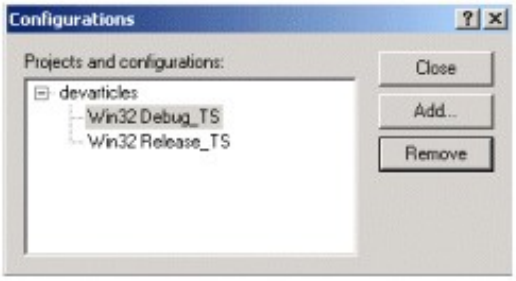
Moving right along, open your MS Visual Studio and make sure no other projects or files are open. Under the File menu, choose New and from the list of wizards choose the "Win32 Dynamic−Link Library". All PHP extensions under windows must be in form of a DLL, thus it is imperative to choose the right type of project. Choosing anything different will result in ugly and indecipherable linker errors (If you've ever worked with Visual C++, no doubt you are familiar with the cryptic messages the compiler throws at you).

Now, for the location. Enter – or rather browse to – the extension subdirectory in your PHP development directory. For the project name enter "devarticles" and this should be appended automatically to your location.

Click the OK button. On the next menu choose "An empty DLL project" and click the finish button. A new window should open up and show you all the information relating to the project; click the OK button again.

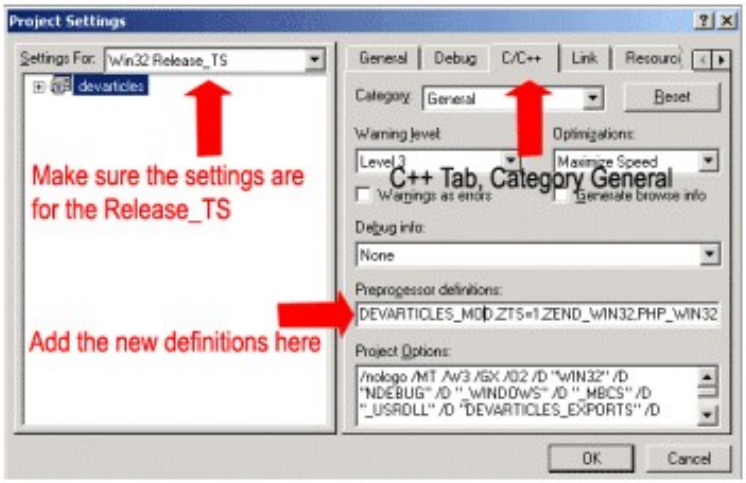
Now, the following few steps must, unfortunately, be completed for each extension project you create:

1. Under the Build menu select Configuration (Build−>Configuration). In that dialog, add two configurations: Release_TS and Debug_TS. Make sure Release_TS copies the settings from the Release configuration and Debug_TS copies the setting from the Debug configuration. After this is done, remove the Release and Debug configurations. Make sure the dialog looks as follows and close it:



2. Now we will set the options in the Release_TS configuration. Open up Project > Settings. On the C/C++ tab, General Category, add the following preprocessor definitions:

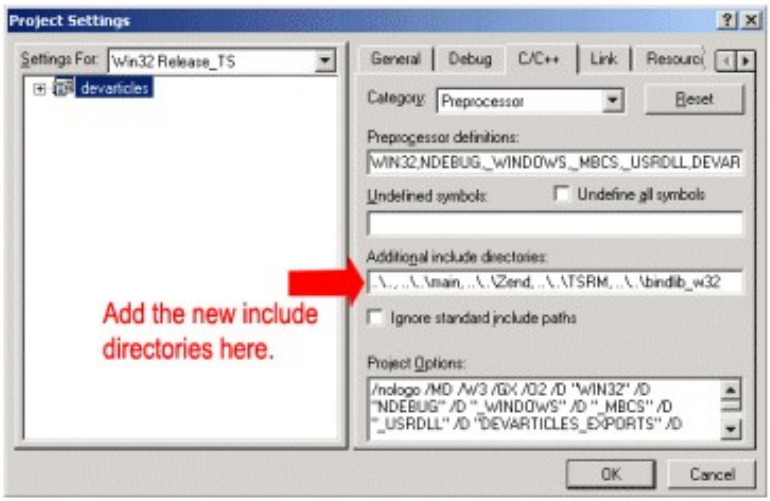
ZEND_DEBUG=0, COMPILE_DL_YOUR_EXTENSION_NAME, ZTS=1, ZEND_WIN32, PHP_WIN32. YOUR_EXTENSION_NAME is the name of your extension. For example: COMPILE_DL_DEVARTICLES_MOD.



3. In the Code Generation Category, change Use run−time Library to "Multithreaded DLL".

4. In the Preprocessor Category, add the following "Additional include directories": ..\.., ..\..\main, ..\..\Zend, ..\..\TSRM, ..\..\bindlib_w32.

The directories are relative to where your project is found. Since our project lives in the \ext directory, the paths are right. However, if you plan to hold your extensions elsewhere you will need to change the directories appropriately.



5. On the Link tab in the General Category, change the Output file name to ..\..\Release_TS/php_devarticlesmod.dll . In future projects you will be able to change the name to anything (*.dll). However, most PHP extensions come with the prefix php_ to indicate they are in−fact PHP extensions. (php_*.dll)

6. On the same tab (Link), same category (General), add php4ts.lib to Object/library modules.

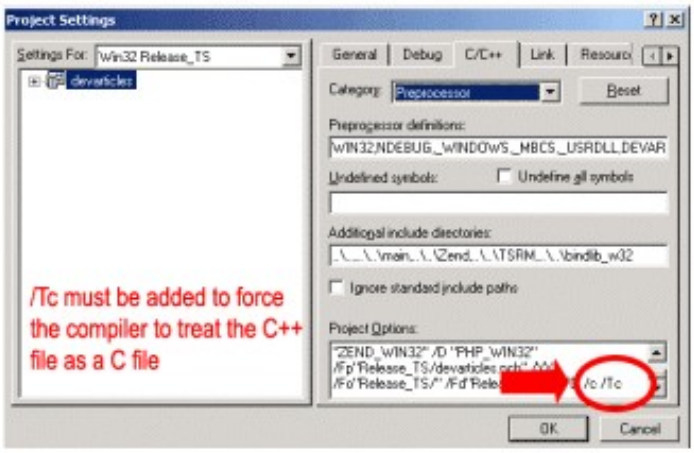
7. In the Input Category, add the following to Additional library path: ..\..\Release_TS.

Close the dialog and set the active configuration to Release_TS. This is done by the Build−>Set Active Configuration... dialog.

Our development environment is almost set – or it is completely set, depending on what you do next. You see, PHP was written in C and not C++, thus all extensions must be written in such a way that they use the C naming convention. Using C++ and not C would cause a linker error (LNK2001 to be exact).

When you add a new source code file to your project, the default extension (not to be mixed with our topic) is *.CPP. This extension causes the compiler to use the C++ naming convention and ultimately leads to the linker error. You can avoid this by changing the extension on the source code file to .C, which will force to compiler to use the C naming convention. On the other hand, if you prefer to continue using .CPP files you will need to add the following option in your project options: /Tc

This will force to compiler to use C naming convention even though it is a C++ source code file.



If you followed all of the instructions carefully then you should be able to commence writing your own extensions. I will leave all the serious coding to the next article and just show you a basic extension that prints "Hello World" five times. Copy and paste the following code into your project; compile and build it:

```
/* include standard header */
/* you will need to include this in all of your php extension projects*/
#include "php.h"

/* All the functions that will be exported (available) must be declared */
ZEND_FUNCTION(hello_world);
PHP_MINFO_FUNCTION(devarticlesmod);

/* Just a basic int to be used as a counter*/
int i;

/* function list so that the Zend engine will know what's here */
```

```
zend_function_entry devarticlesmod_functions[] =
{
ZEND_FE(hello_world, NULL)
{NULL, NULL, NULL}
};

/* module information */
zend_module_entry devarticlesmod_module_entry =
{ STANDARD_MODULE_HEADER,
"DevArticles",
devarticlesmod_functions,
NULL,
NULL,
NULL,
NULL,
PHP_MINFO(devarticlesmod),
NO_VERSION_YET,
STANDARD_MODULE_PROPERTIES };

#if COMPILE_DL_DEVARTICLES_MOD
ZEND_GET_MODULE(devarticlesmod)
#endif

PHP_MINFO_FUNCTION(devarticlesmod)
{
php_info_print_table_start();
php_info_print_table_row(2, "DevArticles Extension", "All Systems Go");
php_info_print_table_end();
}

ZEND_FUNCTION(hello_world)
{

for(i=0;i<5;i++)
{
zend_printf("Hello World<br>");
}
}
```

Don't worry if you do not understand a lot of what's in the C code above. This article is only meant to show you how to setup the environment and explain how to use the extensions in your scripts. All of the code−related material will be explained in the subsequent article(s).

After you've compiled and built the extension, you should take the newly created .dll file (php_devarticlesmod.dll in our case) and place it in your web−server directory. In that same directory, create a new PHP file and type the following:

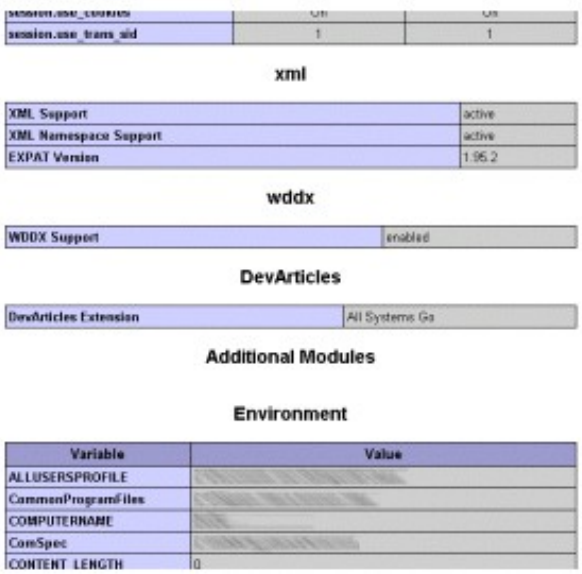
```
<?php

dl("php_devarticlesmod.dll");
```

```
hello_world();

phpinfo();

?>
```

The first line tells php to load the module/extension. The function takes the name of the extension file as the parameter.

The second line, hello_world(), is the function we created in our extension. Upon calling this function you should see "Hello World" printed five times on the screen.

And, lastly, phpinfo() is called. If you scroll down through the output you should see a confirmation that your extension is indeed loaded:



session.use_trans_sid	1	1

xml

XML Support	active
XML Namespace Support	active
EXPAT Version	1.95.2

wddx

WDDX Support	enabled

DevArticles

DevArticles Extension	All Systems Go

Additional Modules

Environment

Variable	Value
ALLUSERSPROFILE	
CommonProgramFiles	
COMPUTERNAME	
ComSpec	
CONTENT_LENGTH	0

Although we haven't created an extension just yet, we've laid the necessary foundation to do so in the next article. You now have an understanding about extensions in PHP, the different kinds of extensions that we can create, and most importantly, you have the development environment set.

In the next article I will assume the development environment is set, and we will begin working on our extension. Some of the topics that will be covered are:

- Passing variables to functions
- Returning values from functions
- Memory allocation and management
- Zend engine macros