Wrapping C++ Classes in a PHP Extension

April 22, 2009

Tutorials

One of the most common reasons people write extensions for PHP is to link against an external library and make available its API to PHP userspace scripts. Often the external library is written in C or C++. If you are tasked with writing such an extension that links against an object oriented C++ library, you will probably want to wrap the C++ classes and make them available to PHP userspace scripts. This is pretty easy to do once you get over a few initial hurdles.

In this tutorial I am going to walk you through creating a PHP extension called “vehicles” which will expose a single class called “Car” (obviously in the real-world, your extensions will expose many classes, but I’m trying to keep things simple). The extension will be built for PHP 5. I am only going to give instructions for building the extension in a UNIX-like environment, although most of what I cover should apply to Windows extension development as well. Most of what I detail here will be an expanded or simplified version of material covered in the somewhat abandoned Using C++ With PHP. I am not going to cover the basics of PHP extension writing, for that I recommend Sara Goleman’s Extension Writing series or her book Extending and Embedding PHP.

File Layout

The extension you will be creating consists of several files which you should place in a directory called vehicles:

- car.h – Contains the definition of the C++ Car class.
- car.cc – Contains the implementation of the C++ Car class.
- php_vehicles.h – Contains PHP specific includes, external variables, etc.
- vehicles.cc – The main source file of the extension. This will contain the PHP Car class.
- config.m4 – PHP Build system / autoconf configuration file.

Configuring the PHP Build System

In order to use C++ in your PHP extension, you must instruct the PHP build system to use a C++ compiler. This is done by calling the PHP_REQUIRE_CXX() macro in your config.m4 file. PHP_REQUIRE_CXX() is defined in aclocal.m4 in the root directory of the PHP source distribution and wraps the autoconf macro AC_PROG_CXX(), which is documented here. You must also link the C++ standard library (libstdc++ on most systems), which is done with the PHP_ADD_LIBRARY() macro. Put the following in your config.m4 file:

```
PHP_ARG_ENABLE(vehicles,
    [Whether to enable the "vehicles" extension],
    [  --enable-vehicles      Enable "vehicles" extension support])

if test $PHP_VEHICLES != "no"; then
    PHP_REQUIRE_CXX()
    PHP_SUBST(VEHICLES_SHARED_LIBADD)
    PHP_ADD_LIBRARY(stdc++, 1, VEHICLES_SHARED_LIBADD)
    PHP_NEW_EXTENSION(vehicles, vehicles.cc car.cc, $ext_shared)
fi
```

Depending on your compiler and how you compiled PHP, you may have to wrap your includes and certain macros in a linkage specification. If you’re not sure what that means, this article explains it nicely. So, together with all the usual skeleton code for a PHP extension, your php_vehicles.h header file should look something like this:

```
#ifndef PHP_VEHICLES_H
#define PHP_VEHICLES_H

#define PHP_VEHICLES_EXTNAME  "vehicles"
#define PHP_VEHICLES_EXTVER   "0.1"

#ifdef HAVE_CONFIG_H
#include "config.h"
#endif

extern "C" {
#include "php.h"
}

extern zend_module_entry vehicles_module_entry;
#define phpext_vehicles_ptr &vehicles_module_entry;

#endif /* PHP_VEHICLES_H */
```

Additionally, you’ll probably need to wrap the call to the ZEND_GET_MODULE() macro in vehicles.cc in a similar linkage specification. With that in mind, your vehicles.cc file should look like the following:

```
#include "php_vehicles.h"

PHP_MINIT_FUNCTION(vehicles)
{
    return SUCCESS;
}
```

```
zend_module_entry vehicles_module_entry = {
#if ZEND_MODULE_API_NO >= 20010901
    STANDARD_MODULE_HEADER,
#endif
    PHP_VEHICLES_EXTNAME,
    NULL,                  /* Functions */
    PHP_MINIT(vehicles),
    NULL,                  /* MSHUTDOWN */
    NULL,                  /* RINIT */
    NULL,                  /* RSHUTDOWN */
    NULL,                  /* MINFO */
#if ZEND_MODULE_API_NO >= 20010901
    PHP_VEHICLES_EXTVER,
#endif
    STANDARD_MODULE_PROPERTIES
};

#ifdef COMPILE_DL_VEHICLES
extern "C" {
ZEND_GET_MODULE(vehicles)
}
#endif
```

Side Note: There is a set of macros provided with the PHP / Zend API called BEGIN_EXTERN_C / END_EXTERN_C but I don't use them. They're defined in Zend/zend.h which is included from php.h, so they're not available when you need them in php_vehicles.h. In my opinion, the inconsistency of using macros in some places and not in others just isn't worth the debatable aesthetic benefit you might obtain from using them.

At this point you can do a quick sanity check and make sure that your extension is compiling properly. Possible Gotchya: Even though we haven't done anything with it yet, make sure that car.cc exists since we referred to it in config.m4 earlier. If the file does not exist, you will run into problems with this step.

```
[paul@helm vehicles]$ phpize
PHP Api Version:         20041225
Zend Module Api No:      20060613
Zend Extension Api No:   220060519
[paul@helm vehicles]$ ./configure --enable-vehicles
checking for grep that handles long lines and -e... /bin/grep
checking for egrep... /bin/grep -E
checking for a sed that does not truncate output... /bin/sed
checking for gcc... gcc
checking for C compiler default output file name... a.out
...
[paul@helm vehicles]$ make
...
[paul@helm vehicles]$ sudo make install
...
[paul@helm vehicles]$ php -d"extension=vehicles.so" -m
[PHP Modules]
bcmath
bz2
calendar
ctype
tokenizer
vehicles
wddx

[Zend Modules]

[paul@helm vehicles]$
```

I trimmed the output, but you get the idea. You should see the extension listed when you run php -m. Also replace sudo with whatever is appropriate for your system / setup. Maybe you're doing this as a user who has permission to write to the php extension_dir or maybe you use "su -c" or something similar.

The C++ Class

Now that you have the basic skeleton of your extension and the PHP build system knows to use a C++ compiler to compile it, you can work on the class that you're going to expose to PHP. Notice how I've kept the C++ class definition and implementation separate from any extension specific code? This isn't strictly necessary, but it decreases coupling and is generally a good idea (if you want to reuse your C++ class, you can do so without pulling over any PHP extension cruft).

car.h:

```
#ifndef VEHICLES_CAR_H
#define VEHICLES_CAR_H

// A very simple car class
class Car {
public:
    Car(int maxGear);
    void shift(int gear);
    void accelerate();
    void brake();
    int getCurrentSpeed();
    int getCurrentGear();
private:
    int maxGear;
    int currentGear;
    int speed;
};

#endif /* VEHICLES_CAR_H */
```

car.cc:

```
#include "car.h"

Car::Car(int maxGear) {
    this->maxGear = maxGear;
    this->currentGear = 1;
    this->speed = 0;
}

void Car::shift(int gear) {
    if (gear < 1 || gear > maxGear) {
        return;
    }
    currentGear = gear;
}

void Car::accelerate() {
    speed += (5 * this->getCurrentGear());
}

void Car::brake() {
    speed -= (5 * this->getCurrentGear());
}

int Car::getCurrentSpeed() {
    return speed;
}

int Car::getCurrentGear() {
    return currentGear;
}
```

In order to expose this class to PHP userspace scripts, you'll first need to define a PHP class called Car with a function_entry table that contains a method for each of the C++ class methods you wish to expose. I should point out that the PHP class does not need to have the same name as the C++ class, and it is completely feasible to include methods in your PHP class that do not exist in your C++ class and vice versa. Again, I'm striving for simplicity here so I've kept the class names the same and I've maintained a 1-to-1 mapping of C++ and PHP methods. Update vehicles.cc to contain the following:

```
#include "php_vehicles.h"

zend_class_entry *car_ce;

PHP_METHOD(Car, __construct)
{
}
PHP_METHOD(Car, shift)
{
}
PHP_METHOD(Car, accelerate)
{
}
PHP_METHOD(Car, brake)
{
}
PHP_METHOD(Car, getCurrentSpeed)
{
}
PHP_METHOD(Car, getCurrentGear)
{
}

function_entry car_methods[] = {
    PHP_ME(Car,  __construct,     NULL, ZEND_ACC_PUBLIC | ZEND_ACC_CTOR)
    PHP_ME(Car,  shift,           NULL, ZEND_ACC_PUBLIC)
    PHP_ME(Car,  accelerate,      NULL, ZEND_ACC_PUBLIC)
    PHP_ME(Car,  brake,           NULL, ZEND_ACC_PUBLIC)
    PHP_ME(Car,  getCurrentSpeed, NULL, ZEND_ACC_PUBLIC)
    PHP_ME(Car,  getCurrentGear,  NULL, ZEND_ACC_PUBLIC)
    {NULL, NULL, NULL}
};

PHP_MINIT_FUNCTION(vehicles)
{
    zend_class_entry ce;
    INIT_CLASS_ENTRY(ce, "Car", car_methods);
    car_ce = zend_register_internal_class(&ce TSRMLS_CC);
    return SUCCESS;
}

zend_module_entry vehicles_module_entry = {
#if ZEND_MODULE_API_NO >= 20010901
    STANDARD_MODULE_HEADER,
#endif
    PHP_VEHICLES_EXTNAME,
    NULL,        /* Functions */
    PHP_MINIT(vehicles),        /* MINIT */
    NULL,        /* MSHUTDOWN */
    NULL,        /* RINIT */
    NULL,        /* RSHUTDOWN */
    NULL,        /* MINFO */
#if ZEND_MODULE_API_NO >= 20010901
    PHP_VEHICLES_EXTVER,
#endif
    STANDARD_MODULE_PROPERTIES
};

#ifdef COMPILE_DL_VEHICLES
extern "C" {
ZEND_GET_MODULE(vehicles)
}
#endif
```

If this is looking a little foreign to you, I recommend reading Chapters 10 and 11 of "Extending PHP".

Connecting the Dots

At this point you have a C++ class and a PHP class with identical (or similar) definitions. Now you need to connect the two. Each instance of the PHP class must have access to exactly one instance of the C++ class. One way to do this is to use a struct to keep track of the C++ and the PHP instances. Doing so requires that you implement your own object handlers. Remember that in PHP5, an object is just a handle (a unique id that allows the Zend Engine to locate your class), a method table, and a set of handlers. You can override handlers to perform the various tasks that need to be performed at various stages of the object's lifecycle. Add the following to the top your vehicles.cc file:

```
#include "car.h"

zend_object_handlers car_object_handlers;

struct car_object {
    zend_object std;
    Car *car;
};
```

The car_object structure will be used to keep track of our C++ instances, associating them to a zend_object. Add the following functions to your vehicles.cc file, above the PHP_METHOD declarations:

```
void car_free_storage(void *object TSRMLS_DC)
{
    car_object *obj = (car_object *)object;
    delete obj->car;

    zend_hash_destroy(obj->std.properties);
    FREE_HASHTABLE(obj->std.properties);

    efree(obj);
}

zend_object_value car_create_handler(zend_class_entry *type TSRMLS_DC)
{
    zval *tmp;
    zend_object_value retval;

    car_object *obj = (car_object *)emalloc(sizeof(car_object));
    memset(obj, 0, sizeof(car_object));
    obj->std.ce = type;

    ALLOC_HASHTABLE(obj->std.properties);
    zend_hash_init(obj->std.properties, 0, NULL, ZVAL_PTR_DTOR, 0);
    zend_hash_copy(obj->std.properties, &type->default_properties,
        (copy_ctor_func_t)zval_add_ref, (void *)&tmp, sizeof(zval *));
    retval.handle = zend_objects_store_put(obj, NULL,
        car_free_storage, NULL TSRMLS_CC);
    retval.handlers = &car_object_handlers;

    return retval;
}
```

And update your PHP_MINIT_FUNCTION in vehicles.cc:

```
PHP_MINIT_FUNCTION(vehicles)
{
    zend_class_entry ce;
    INIT_CLASS_ENTRY(ce, "Car", car_methods);
    car_ce = zend_register_internal_class(&ce TSRMLS_CC);
    car_ce->create_object = car_create_handler;
    memcpy(&car_object_handlers,
        zend_get_std_object_handlers(), sizeof(zend_object_handlers));
    car_object_handlers.clone_obj = NULL;
    return SUCCESS;
}
```

There's quite a bit going on here. First, once we have a dedicated zend_class_entry pointer in the MINIT_FUNCTION, we can assign a create_object handler to it. The create_object handler allocates memory for the object, sets up the zend_object and generates a handle for the object by storing the car_object pointer. You'll also notice that the zend_objects_store_put function accepts a function pointer to what is basically a c-level destructor for the object being handled.

Finally, in your PHP class constructor, you'll parse the user parameters like usual and pass them on to the C++ constructor. Once the C++ class instance is created, you can fetch the car_object pointer from the zend object store and set the car field in the struct. This makes the C++ class instance available to other instance methods of your PHP class.

```
PHP_METHOD(Car, __construct)
{
    long maxGear;
    Car *car = NULL;
    zval *object = getThis();

    if (zend_parse_parameters(ZEND_NUM_ARGS() TSRMLS_CC, "l", &maxGear) == FAILURE) {
        RETURN_NULL();
    }

    car = new Car(maxGear);
    car_object *obj = (car_object *)zend_object_store_get_object(object TSRMLS_CC);
    obj->car = car;
}
```

Now, in order to obtain an instance of the C++ class, instance methods of your PHP class just need to call zend_object_store_get_object:

```
PHP_METHOD(accelerate)
{
    Car *car;
    car_object *obj = (car_object *)zend_object_store_get_object(
        getThis() TSRMLS_CC);
    car = obj->car;
    if (car != NULL) {
        car->accelerate();
    }
}
```

```
PHP_METHOD(getCurrentSpeed)
{
    Car *car;
    car_object *obj = (car_object *)zend_object_store_get_object(
        getThis() TSRMLS_CC);
    car = obj->car;
    if (car != NULL) {
        RETURN_LONG(car->getCurrentSpeed());
    }
    RETURN_NULL();
}
```

The remaining methods all work in basically the same way, I'll leave implementing them as an exercise.

That's it, we're done! Now let's take our new extension for a test drive. Don't forget to re-compile, install and of course enable it in php.ini (or continue to use the -dextension=vehicles.so command line argument).

```
// create a 5 gear car
$car = new Car(5);
print $car->getCurrentSpeed();  // prints '0'
$car->accelerate();
print $car->getCurrentSpeed(); // prints '5'
```

If you can run this script, congratulations, you've just created a PHP extension that wraps a C++ class.

About Zend Community

View all posts by Zend Community →

Subscribe

Subscribe to our e-mail newsletter to receive updates.

Related Posts:

- Announcement : Google's Peter S Magnusson to Keynote ZendCon 2013
- Guest Post : PHP's Remarkable Hexadecimals
- XML- RPC Client
- Announcement : "David I" will Keynote ZendCon 2013
- Introducing Chris Tankersley – ZendCon 2013 Speaker

← MySQL announcements: Oracle & MySQL 5.4
The ZendCon Sessions Episode 17: SQL Query Tuning: The Legend of Drunken Query Master →

33 Responses to "Wrapping C++ Classes in a PHP Extension"

1. *akki_nitsch* Says:
   October 17, 2011 at 12:28 pm

   Hi everybody,

   first of all i’ve got to thank paulosman for this tutorial – great job 🙂

   I was able to reproduce the tutorial on a virtual machine under Linux on a i386 processor – everything worked perfectly fine.
   After that i tried to cross-compile php and the PHP extension for a board with an ARM processor (in this case an ARM926EJ-S).
   Cross-compiling of PHP and the extension worked but sadly the installation on the board failed.

   When i call "php -m" on the console, the extension is not listed (i already enabled the extension in php.ini).
   The attempt to load the extension dynamically via console with "php -d"extensions=vehicles.so" -m" does not produce any
   warning or error – but the extention is not loaded.

   Loading the module with "dl(vehicles.so)" in a php script produces the following warning:
   "Warning: dl(): Cannot dynamically load vehicles.so – dynamic modules are not supported in /usr/local/lib/php/extensions/no-debug-non-zts-20090626/test.php on line 12"

   Has anyone experiences with cross-compiling of self-written PHP-extensions. I would be very glad for any hint.

   Best regards

   Andreas

2. *fan_me* Says:
   September 12, 2011 at 8:35 am

   Hi,

   I have the following .h:

```
public:
class First
{
First();
public;
class Second
{
Second();
class Data
{
//methods, variables
};
bool methodd(First::Second &obj);
};
};
```

how to rewrite your vehicles.cc in this case? how to create a method that waits for a reference? how to link to classes and nested classes?

YHX . APPRECIATE

3. *nina_din* Says:
[May 25, 2011 at 6:13 pm]

Hi,
Thank you for your helpful sample.
I have to develop an extension with 2 class. the first class , its name is Point- gets int x , int y as constructor parameter.also it has 3 methods, getx(), getY() and getXY() functions. this works well.
The second extension is another class that constructor's parameters are Point class type. for example SecClass (new Point (2,3));
the implementation in c++ is correct and it works. but when i test it as extension, it does not return correct value.can u help me what should I do?
I can send my classes to you.

I really appreciate.

4. *miracletan* Says:
[December 16, 2010 at 8:51 am]

Hi:
I follow these steps, but i got a problem below:

```
-bash-3.2# php -m
PHP Warning: PHP Startup: Unable to load dynamic library '/usr/lib64/php/modules/vehicles.so' –
/usr/lib64/php/modules/vehicles.so: undefined symbol: _ZN3CarC1Ei in Unknown on line 0
```

I tried a lot,but it still doesn't work…

PS:I changed "CC=gcc" to "CC=g++" in Makefile.

5. *miracletan* Says:
[December 16, 2010 at 8:50 am]

Hi:
I follow these steps, but i got a problem below:

```
-bash-3.2# php -m
PHP Warning: PHP Startup: Unable to load dynamic library '/usr/lib64/php/modules/vehicles.so' –
/usr/lib64/php/modules/vehicles.so: undefined symbol: _ZN3CarC1Ei in Unknown on line 0
```

I tried a lot,but it still doesn't work…

PS:I changed "CC=gcc" to "CC=g++" in Makefile.

6\.
contentwithstyle Says:
November 6, 2010 at 9:24 am

One thing I didn't see mentioned here is any potential speed benefit. I don't know if there is one for PHP, but in the world of Perl there is the concept of XSification (http://perldoc.perl.org/perlxs.html), effectively writing Perl classes in C, which has in many cases a massive speed improvement.

7\.
a_white Says:
September 29, 2010 at 12:03 pm

```
struct car_object {
zend_object std;
Car *car;
};
```

Don't understand why we must add a zend_object member in car_object. Check out zend_objects_store_put function and zend_object_store_get_object function in zend source code(zend_objects_API.c),they don't use the "zend_object std" in car_object,so i don't what it is for?

Could you explain why we must put one zend_object into first place in object struct ?
Thanks.

8\.
thebigonion Says:
May 18, 2010 at 12:43 pm

Grrrrr.

If I do this:
—————-

```
string myString1;
int myInt;
string myString2;

if(zend_parse_parameters(ZEND_NUM_ARGS() TSRMLS_CC, "sls", &myString1, &myInt,
&myString2) == FAILURE)
{
RETURN_NULL();
}
```

——————-

And then run it, I get a "Segmentation Fault (core dumped)" error.

But if I add in the extra ints for my strings like:
——————-

```
string myString1;
```

```
int myInt;
string myString2;
int myStringInt1, myStringInt2;

if(zend_parse_parameters(ZEND_NUM_ARGS() TSRMLS_CC, "sls", &myString1, &myStringInt1,
&myInt, &myString2, &myStringInt2) == FAILURE)
{
RETURN_NULL();
}
```
————

when I run it I get a bunch of text. Here's part:

————

```
Start: Overflown (magic=0x345ACE99F instead of 0x0A9CEA43)
1 byte(s) overflown
End: OK
…
=== Total 2 memory leaks detected ===
```
————

Can someone please help me figure out what I am doing wrong?

9. *scaupp* Says:
[May 17, 2010 at 6:31 pm]

I've never tried using std::string, so I don't know if that will parse correctly, you may have to use a pointer to a char array.

If you're asking if you have to initialize the string length vars, then no, you don't, the zend_parse_parameters call fills that in with the length of the string passed in.

I'm 99% certain that you can't just pass the values by reference and have PHP see the results, but I'm no expert on this, I just happen to have gotten our extension to finally work… 🙂

10. *thebigonion* Says:
[May 17, 2010 at 5:47 pm]

And do I need to set any values to the string lengths?

11. *thebigonion* Says:
[May 17, 2010 at 3:55 pm]

Thank you so much. You have been a great help.
One more question:

Does this look right?

```
std::string ab_IDCode1;
int iabCode1Len;
int is_by;
std::string ab_IDCode3;
std::string ab_IDCode2;
int iabCode2Len, iabCode3Len;

if (zend_parse_parameters(ZEND_NUM_ARGS() TSRMLS_CC,
"slss",
&ab_IDCode1, &iabCode1Len, &is_by,
&ab_IDCode2, &iabCode2Len,
&ab_IDCode3, &iabCode3Len,) == FAILURE) {
RETURN_NULL();
}
```

Also, my &iabCode3 is a parameter that is sent via reference and so my call in PHP should be able to see the value in that variable after the call to the method.

Thanks again.

12. *scaupp* Says:
[May 13, 2010 at 4:58 pm]

The ‘ssll’ indicates two strings and two integers. The six variables hold the first string, the length of the first string, the second string, the length of the second string, the first integer, and the second integer. I.e. for every string param you pass in you have to parse it out to two variables, one for the string and one for the length.

Extending and Embedding PHP (the author is a woman, BTW) has a few holes that make it a little tricky if you’re not already familiar with how the PHP stuff is coded in C.

Code like the above should work, I checked it against our working extension DLL.

13. *thebigonion* Says:
[May 13, 2010 at 3:13 pm]

It’s weird,
if you look in "Extending and Embedding PHP" on page 82, the example is like yours, but has 2 strings and 2 integers. But in his zend_parse_parameters he only declares "ss". What about the integers?

14. *scaupp* Says:
[May 13, 2010 at 2:45 pm]

It’s been a while, but something like this, I believe:

```
unsigned char *ab_IDCode1, *ab_IDCode2;
int iabCode1Len, iabCode2Len;
int is_by, iDays;

if (zend_parse_parameters(ZEND_NUM_ARGS() TSRMLS_CC,
"ssll",
&ab_IDCode1, &iabCode1Len,
&ab_IDCode2, &iabCode2Len,
&is_by, &iDays) == FAILURE) {
RETURN_NULL();
}
```

HTH

15.
thebigonion Says:
[May 13, 2010 at 2:32 pm]

ok, so I see how to do it with:

```
if (zend_parse_parameters(ZEND_NUM_ARGS() TSRMLS_CC, "l", &maxGear) == FAILURE)
{
RETURN_NULL();
}
```

But, how do I pass in a string and an integer? The examples in "Extending and Embedding PHP" don't show two different argument types.

Thanks

16.
thebigonion Says:
[May 12, 2010 at 8:49 pm]

What should this look like?:

```
PHP_METHOD(Car, shift)
{
}
```

In my real code, I have a method that takes two arguments:
function this(const std:string myString, const int myInt)
but i do not know how to change my PHP_METHOD to handle it.

Can you please show me using the vehicle example? (shift takes an argument)

Thanks.

17.
davidweaverking Says:
[April 21, 2010 at 8:52 pm]

No, but your comment did help. Apparently when you specify libraries for ld to link it (like what eventually happens in this process) ld looks for a library called "lib<libraryName>.so". In my previous tries that meant that ld was looking for a library called liblibvirt.so and (obviously) not finding it. So… what finally worked was

```
PHP_ADD_LIBRARY(virt, 1, PHPVIRT_SHARED_LIBADD)
```

I hope this helps some else as well. It took me a day and a half to solve it

18. *dabaipang* Says:
April 21, 2010 at 8:38 pm

did you try:
make LDFLAGS="-lvirt"

or did you put your .so file under liken/usr/local/php/lib
which is a default path for your php to find it.

19. *davidweaverking* Says:
April 21, 2010 at 8:32 pm

Is there documentation anywhere (besides the source code) of the functions available in the config.m4 file (like the PHP_ADD_LIBRARY function)? I'm trying to link against the shared library /usr/lib/libvirt.so so I added the line

```
PHP_ADD_LIBRARY(libvirt, 1, PHPVIRT_SHARED_LIBADD)
```

to my config.m4 file but when I run make it gives the error

```
/usr/bin/ld: cannot find -llibvirt
```

I've tried using libvirt.so and PHP_ADD_LIBRARY_WITH_PATH but it always gives the same error. Any ideas what I'm doing wrong?

20. *dabaipang* Says:
April 19, 2010 at 8:34 pm

1.
```
dnl PHP_NEW_EXTENSION(vehicles, vehicles.c, $ext_shared)
PHP_SUBST(VEHICLES_SHARED_LIBADD)
PHP_ADD_LIBRARY_WITH_PATH(stdc++, "", VEHICLES_SHARED_LIBADD)
PHP_REQUIRE_CXX()
PHP_NEW_EXTENSION(vehicles, vehicles.cpp, $ext_shared)

PHP_NEW_EXTENSTION(vehicles, vehicles.c, $ext_shared)
|
```

^
vehicles.cc

2. in vehicles.cc:
Must:
#ifdef COMPILE_DL_VEHICLES
extern "C" {
ZEND_GET_MODULE(vehicles)
}
#endif

This is crucial, as it's already pointed out by author.

3. in php_vehicles.h
Must:
extern "C" {
#ifdef ZTS
#include "TSRM.h"
#endif
}

It's not mentioned in the post.

If you ever have ***.so maybe not a PHP library error, it means you have forgotten to extern "C" somewhere in either .h or .cpp/.cc files;

Everytime, make sure:
phpize
./configure –enable-vehicles
make
make install
php vehicles.php

Thank you.

21. *dabaipang* Says:
April 18, 2010 at 11:18 pm

I have followed exactly what you have written on the post, but at very first, vehicle.cc is something must be manually created.
Did you deleted vehicle.c?

I tried deleting vehicle.c at first, but there is this error when I 'make'
make: *** No rule to make target `/home/qsl/download/php-5.3.1/ext/vehicles/vehicles.c', needed by `vehicles.lo'. Stop.

Then I created a blank vehicles.c, somehow, it works, but when
I use: $php -d"extension=vehicles.so" -m

There is this error:

PHP Warning: PHP Startup: Invalid library (maybe not a PHP library) 'vehicles.so' in Unknown on line 0

I can’t understand why there are so many people running through this without this problem.

22. *scaupp* Says:
 November 16, 2009 at 8:56 pm

 With others, I’d like to thank you for posting this, it saved me quite some time.

 And as others, I’ve got a question for you 🙂

 I adopted your approach versus the "direct" one in Sara Golemon’s "Extending and Embedding PHP" since we had an existing C++ object whose functionality we needed to integrate into our PHP scripts and didn’t want to muddy the original object with the PHP code as the C++ version is used separately in other applications.

 One thing I have not been able to grok on my own is how to expose the properties in the underlying C++ object to PHP. Do you have any hints on that that you could share? I think the trick might be in implementing __set and __get functions and reading and setting the C++ object properties there, but I’m pretty much at sea at this point.

 Thank you.

23. *nastikkl* Says:
 October 21, 2009 at 5:38 am

 Thank you for good article. Anyway I cant decide what to do in my situation.
 I have .dll file and .h file describing structure with pointers to library functions.
 I defined and registered that structure as a resource type in PHP, defined function which create and return the resource type. Still cant figure out how to work with function pointers in this case.

24. *nicolae-serban* Says:
 September 18, 2009 at 8:03 am

 Very good article, thank you

25. *niniwzw* Says:
 August 12, 2009 at 12:46 pm

 i got the error:

 Error 2 error LNK2019: unresolved external symbol __zval_ptr_dtor_wrapper, the symbol in function "struct _zend_object_value __cdecl car_create_handler (struct _zend_class_entry *, void * * *)" (? Car_create_handler @ @ YA? AU_zend_object_value @ @ PAU_zend_class_entry @ @ PAPAPAX @ Z) was quoted vehicles.obj please help me , thanks.

26. _____*anonymous*_____ Says:
    June 10, 2009 at 10:42 pm

    I’ve got a statically linked library I’d like to make into a php module, it has a couple classes in it (but only one needed for interface). Anyway, I’d like the resulting php shared object module to have the entire library linked into it. I have the sources for the library .c files compiled into a .o files, then combined into a library archive .a file. Is there a way to add .o files to a config.m4 file? or even better to just add a .a file to it directly?

    thanks

    P.S. the documentation for php module config macros on php.net is next to non-existant for php 5 =)

27. *EABonney* Says:
    June 9, 2009 at 10:59 pm

    Thanks for a great article! It has really gotten me started on the path I need for a project that I am working on. I was wondering though, is there a way to create an extension wrapper like this so that it is only exposed to the php project that needs it and not the entire server? I thought I read somewhere that it was possible to create such an extension but now I can’t figure out how.

    I would like to create an extension and want it only available for my project and I don’t want my end users to have to install the .so file and then restart Apache etc..

    Is this possible?

    Thanks again!
    -Eric

28. *paulosman* Says:
    June 9, 2009 at 3:17 am

    In order to be able to use a shared library from your extension code, you’ll need to modify your config.m4 file to link to it. To do this you can use the PHP_ADD_LIBRARY macro like we did in the tutorial in order to link to the libstdc++ library. If your shared library is installed somewhere non-standard (i.e. not in /lib, /usr/lib, etc) you can use the PHP_ADD_LIBRARY_WITH_PATH macro which accepts a path as an argument, i.e.: PHP_ADD_LIBRARY_WITH_PATH(external_lib, $EXTERNAL_LIB_DIR, VEHICLE_SHARED_LIBADD). Hope that helps!

29. *pdah87* Says:
    May 3, 2009 at 4:28 am

This is really a great tutorial since you mentioned about linking the extension against an C/C++ library (which has more reality than the other tutorials). In your post, the library (vehicle) is built together with the extension. It’s a little different from my case :
I had an existing shared library compiled and installed in my system, and I tried to build my extension to use the functions from that library (without implementing it again in the extension source code).
But I can’t get my extension link to the library even the extension was compiled and installed correctly.
I think the problem is somewhere around ".configure" or "make" command, but I didn’t find any solution yet. Can you suggest me a way to make it work?

30. *sherman81* Says:
[May 2, 2009 at 7:46 pm]

    i’ve compiled source code from your link and its working fine.

31. *paulosman* Says:
[April 26, 2009 at 3:05 am]

    <p>To free the memory allocated for the C++ Car instance, you could add "delete obj->car;" to the car_free_storage() function anywhere before the "efree(obj);" line. I’ll edit the article and include that in the example, thanks! </p>

    <p>As for compiling the example code, sorry to hear you are having trouble. Can you download <a href="http://www.eval.ca/files/vehicles.tgz">this version</a> and compile it? If so, you can use that to compare against your own copy. If you have trouble with that, post back here and I’ll do more troubleshooting. </p>

32. *sherman81* Says:
[April 24, 2009 at 11:53 pm]

    I built ext. and installed it, but when i try to load and use it from userspace i have following error:

    Warning: dl(): Invalid library (maybe not a PHP library)

33. *sherman81* Says:
[April 24, 2009 at 11:15 pm]

    And what about dynamic memory alloc/dealloc in c++ classes?

php·2013
zendcon
October 7-10, 2013 • Santa Clara, CA
November 18-20, 2013 • Paris, France
2 locations
Learn from the PHP experts
RETURN HOME UP TO DATE ON BEST PRACTICES
Register Now!

Categories

- Articles
- Events
- News
- PHP
- PHP Cloud
- Tutorials
- Videos
- Zend Framework
- Zend Server
- Zend Studio

Free PHP Tools

Zend Server Free
Zend Framework
Eclipse PDT

Popular Content

PHP 101
PHP SOAP Extension
Using MySQL Full-text Searching
Reading and Writing Spreadsheets with PHP
Decorators with Zend_Form
Debugging PHP applications with xdebug
Getting Started with Zend Server CE

Tags

Ajax Andi Gutmans API AWS Call for Papers Cloud Computing Community DB2 Documentation eBay Hosting IBM i Interviews Java Mobile oAuth Old PHP4 PHP5 PHP6 PHP101 phpcloud.com Podcasts

Press Release Project Management REST RSS Security SOAP Twitter Unit Testing User Groups XML XML-RPC Zend ZendCon ZendCon2013 ZendCon Sessions Zend Framework Zend Server Zend Studio Zend_Config Zend_Form ZF1 ZF2

Login / Register

Login

Username:

Password:

Send

✓ Remember me

Recover password | Create an Account

Quick Links

- Zend Home Page
- Zend Server
- Zend Studio
- Zend Guard
- Zend PHP Training

Recent Posts

- Announcing the ZendCon 2013 App Contest Winner
- Set up a Complete PHP Environment on Linux with Vagrant
- User Group Members Discount for ZendCon 2013
- Q&A : Yard Internet – The 10,000th Zend Certified Engineer (ZCE)
- The ElePHPant's are HOT for ZendCon 2013

Archives

Select Month

phpdeveloper.org

- NetTuts.com: Travis-CI: What, Why, How
- Anthony Ferrara: Beyond Design Patterns
- Stoyan Stefanov: Server-side React with PHP - part 2

phparch.com

- php[architect] Web Summit Series: WordPress
- Introduction to PHP
- php[tek] returns in 2014

ZFDaily

- DluTwBootstrap updated to ZF2 RC6
- DluTwBootstrap updated to ZF2 RC5 and Twitter Bootstrap 2.1.0
- DluTwBootstrap updated to ZF2 RC3
- Getting dependencies into ZF2 Controllers
- DluTwBootstrap updated to ZF2 Beta4
- Configuring PHP ini Settings in ZF2
- ZF2 Gets the baseUrl Wrong
- Twitter Bootstrap Forms with ZF2. Easily.
- Fighting circular dependencies in Bootstrap