Extending PHP 4

By Sterling Hughes

Agenda

How PHP works: An Overview

The PHP API

Compiling a PHP extension

The PHP Build System

HOW?

When adding a module to the PHP4 build system there a couple of steps you must follow. First you have to create a folder in the php4/ext/ directory with the same name as your module. That directory should contain your sources, header files, and your Makefile (**`Makefile.in`**) and your configuration file (**`config.m4`**).

`Makefile.in`

WHAT?

`Makefile.in`, as its name suggests, is the Makefile for your PHP module.

Example, the CURL makefile

```

LTLIBRARY_NAME     = libcurl.la
LTLIBRARY_SOURCES = curl.c
LTLIBRARY_SHARED_NAME    = curl.la
LTLIBRARY_SHARED_LIBADD = $(CURL_SHARED_LIBADD)

include $(top_srcdir)/build/dynlib.mk
```

---

config.m4

WHAT?

The **config.m4** contains configuration information for your PHP module.

Example, The CURL config.m4 file

```
dnl config.m4 for extension CURL

PHP_ARG_WITH(curl, for CURL support,
[  --with-curl[=DIR]        Include CURL support])

if test "$PHP_CURL" != "no"; then
  if test -r $PHP_CURL/include/curl/easy.h; then
    CURL_DIR=$PHP_CURL
  else
    AC_MSG_CHECKING(for CURL in default path)
    for i in /usr/local /usr; do
      if test -r $i/include/curl/easy.h; then
        CURL_DIR=$i
        AC_MSG_RESULT(found in $i)
      fi
    done
  fi

  if test -z "$CURL_DIR"; then
    AC_MSG_RESULT(not found)
    AC_MSG_ERROR(Please reinstall the libcurl distribution -
    easy.h should be in <curl-dir>/include/curl/)
  fi

  AC_ADD_INCLUDE($CURL_DIR/include)

  PHP_SUBST(CURL_SHARED_LIBADD)
  AC_ADD_LIBRARY_WITH_PATH(curl, $CURL_DIR/lib, CURL_SHARED_LIBADD)

  AC_DEFINE(HAVE_CURL,1,[ ])

  PHP_EXTENSION(curl, $ext_shared)
fi
```

Accessing Function Parameters

Overview

To access function parameters use the **zend_get_parameters_ex** function. Then convert them to the correct type using the **convert_to_*_ex** functions. You can then use the **Z_*_PP** macro's to access their values.

Example

```
PHP_FUNCTION (somefunc)
{
    zval **uArg1;
    char *arg1value;
    int arg1len;

    if (ZEND_NUM_ARGS() != 1 ||
        zend_get_parameters_ex (1, &uArg1) == FAILURE) {
        WRONG_PARAM_COUNT;
     }

     convert_to_string_ex (uArg1);

     arg1value = Z_STRVAL_PP (uArg1);
     arg1len   = Z_STRLEN_PP (uArg1);

     RETURN_STRINGL (arg1value, arg1len, 1);
 }
```

Allocating memory

HOW?

When allocating memory in PHP you should use Zend's built-in memory allocation functions instead of the standard memory allocation libraries (you use PHP's functions exactly the same way).

PHP API memory allocation functions & there C equivalents

PHP API Function	C Equivalent
`emalloc()`	`malloc()`
`efree()`	`free()`
`ecalloc()`	`calloc()`
`erealloc()`	`realloc()`
`estrdup()`	`strdup()`
`estrndup()`	`strndup()`

Accessing arrays

Numerically indexed arrays

To access a numerically indexed array convert the array to a '`HashTable *`', then loop through the array with the `zend_hash_num_elements()` & `zend_hash_index_find()` functions.

Example: Accessing an array

```
PHP_FUNCTION(print_array)
{
  zval **uArray, **uSeperator, **element;
  HashTable *ar;
  char *seperator = " ";
  int argcount = ZEND_NUM_ARGS(), numelems, i;

  if (argcount < 1 || argcount > 2 ||
    zend_get_parameters_ex (argcount, &uArray, &uSeperator) == FAILURE) {
     WRONG_PARAM_COUNT;
   }

   if (argcount > 1) {
     convert_to_string_ex (uSeperator);
     seperator = estrndup (Z_STRVAL_PP (uSeperator), Z_STRLEN_PP (uSeperator));
   }

   ar = HASH_OF (*uArray);
   if (!ar) {
     php_error (E_WARNING, "Wrong datatype to print_array(),
                the first argument must be an array");
     RETURN_NULL();
   }

   numelems = zend_hash_num_elements (ar);

   for (i=0; i<numelems; i++) {
     if (zend_hash_index_find(ar, i, (void **)&element) == SUCCESS) {
       SEPERATE_ZVAL(element);
       convert_to_string_ex (element);
       PUTS(Z_STRVAL_PP(element));

       if (i != (numelems-1))
         PUTS(seperator);
       }
   }
 }
```

Accessing arrays

Associative arrays

To loop through associative arrays use a combination of a for loop, `zend_hash_internal_pointer_reset()`, `zend_hash_get_current_data()`, `zend_hash_move_foward()`, and `zend_hash_get_current_key()`.

Example: Accessing an associative array

```
PHP_FUNCTION(print_assoc_array)
{
    zval **uAssocArray, **ent;
    HashTable *ar;
    char *key = NULL, *prnBuf = NULL;
    ulong idx;
    int type;

    if (ZEND_NUM_ARGS() != 1 ||
        zend_get_parameters_ex(1, &uAssocArray) == FAILURE) {
        WRONG_PARAM_COUNT;
     }

     ar = HASH_OF (*uAssocArray);

     for (zend_hash_internal_pointer_reset(ar);
          zend_hash_get_current_data(ar, (void **)&ent) == SUCCESS;
          zend_hash_move_foward (ar))
     {
        SEPERATE_ZVAL(ent);
        convert_to_string_ex(ent);
        type = zend_hash_get_current_key (ar, &key, &idx);

        if (type != HASH_KEY_IS_STRING) {
            sprintf(key, "%d", idx);
        }

        sprintf(prnBuf, "%s: %s\n<br>\n", key, Z_STRVAL_PP (ent));
        PUTS(prnBuf);
     }
 }
```

Calling user functions

HOW?

To call user functions with PHP use the `call_user_function_ex()` function on the global function table.

Example, Calling a user function with PHP

```
PHP_FUNCTION(call_func)
{
    zval **params[2], *func_name, *retval_ptr;
    CLS_FETCH();

    ZVAL_STRING(func_name, "throw_cat", 1);

    ZVAL_LONG(*params[0], 32);
    ZVAL_STRING(*params[1], "meters", 1);

     if (call_user_function_ex(CG(function_table), NULL, func_name, &retval_ptr, 2,
              params, 0, NULL) == SUCCESS) {
          zval_ptr_dtor(&retval_ptr);
     }
 }

```

Creating Global PHP variables

HOW?

To create global PHP variables from your PHP function use the `ZEND_SET_SYMBOL()` macro on the `EG(symbol_table)`.

Example, Adding a global variable to the current PHP script

```
PHP_FUNCTION(add_error_message)
{
    zval *message;
    MAKE_STD_ZVAL(message);

    ZEND_STRING(message, "Error, all h*ll is breaking loose", 1);

    ZEND_SET_SYMBOL(&EG(symbol_table), "error_message", message);
}
```

Declaring Functions

When you declare a function that is to be added to the PHP namespace (via the Function entry), you must declare it using the `PHP_FUNCTION()` macro:

```

PHP_FUNCTION(some_func)
{
     /* Contents of the function go here */
}

```

NOTE: You don't need to specify any return type, the return type is automatically specified by the `PHP_FUNCTION()` macro (always `void`, you set the return value from your function by setting the `return_value` variable)

Deleting Resources

HOW?

To delete a resource first fetch the resource with the `ZEND_FETCH_RESOURCE()` macro, the delete it use the the `zend_list_delete()` function.

Example, Deleting a resource

```
PHP_FUNCTION(close_file)
{
     zval **uFp;
     FILE *fp;

     if (ZEND_NUM_ARGS() != 1 ||
         zend_get_parameters_ex (1, &uFp) == FAILURE) {
         WRONG_PARAM_COUNT;
     }

      ZEND_FETCH_RESOURCE(fp, FILE *, uFp, -1, "File-
Pointer", php_file_le_fopen ());
      zend_list_delete (Z_LVAL_PP(uFp));
 }
```

Fetching a resource

HOW?

To fetch a resource use the `ZEND_FETCH_RESOURCE()` macro.

Example, fetching a resource

```
PHP_FUNCTION(print_to_file)
{
    zval **uFp, **uTxt;
    FILE *fp;

    if (ZEND_NUM_ARGS() != 2 ||
        zend_get_parameters_ex (2, &uFp, &uTxt) == FAILURE) {
        WRONG_PARAM_COUNT;
    }
     convert_to_string_ex (uTxt);

     /*
      * The php_file_le_fopen() function is a PHP API function containing the
      * resource type of a 'FILE *'
      */
     ZEND_FETCH_RESOURCE(fp, FILE *, uFp, -1, "File-
Pointer", php_file_le_fopen());

     RETURN_LONG(fputs(Z_STRVAL_PP (uTxt), fp));
 }
```

Managing Global Variables with PHP

HOW?

Place all your global variables into a singular global structure, add a few magic macros and voila.

Example, A global structure with macros and all

php_extname.h, the header file

```
typedef struct {
    struct *some_other_struct;
    zval **some_var;
    int le_pointer;
} php_extname_globals;

/* The Magic Macro's,
    helping to enable thread safety in the world */
 #ifdef ZTS
 #define EXTNAMEG(v) (extname_globals->v)
 #define EXTNAMELS_FETCH() php_ext_globals *extname_globals =
   ts_resource(gd_extname_id)
 #else
 #define EXTNAMEG(v) (extname_globals.v)
 #define EXTNAMELS_FETCH()
 #endif

```

extname.c, the source file

```
#ifdef ZTS
int extname_globals_id;
#else
php_extname_globals extname_globals;
#endif

```

The API itself

WHAT?

The Function & Module entries interface your PHP extension with the rest of PHP. However, there is much more to the PHP API, including functions and constructs that must be used to declare functions, access function parameters, traverse arrays, check types, return values and more.

List Destructors

WHAT?

List destructors are registered when your module is first initialize, when you delete a resource the list destructor for that resource is called.

HOW?

List destructors are registered when your module is initialized (in the **`PHP_MINIT()`** function), use the **`register_list_destructors()`** function to register your list destructor.

Example, Registering a list destructor

```
static void php_myResource_close (MyResourceHandle *rh);
static int le_myResource;

static void php_myResource_close (MyResourceHandle *rh)
{
    close (rh);
}

PHP_MINIT_FUNCTION (myModule)
 {
     le_myResource = register_list_destructors(php_myResource_close, NULL);
 }
```

Manipulating Zvals

WHAT?

zvals are the basic variable type of the PHP API, they are used for managing resources, fetching function parameters, calling user functions, looping through arrays and many other things. The PHP API supports a few important macros for manipulating **zval**s that we will now cover.

The Macros

`SEPERATE_ZVAL(zval **)`

Performs a zval seperation on the given container. The new zval is datached from internal data and has a local scope, therefore it can be modified without any changes to the data attached to the old variable.

`MAKE_STD_ZVAL(zval *)`

Allocates and initializes a new zval. Using the **`MAKE_STD_ZVAL()`** macro is the same as using the **`ALLOC_ZVAL()`** and **`INIT_ZVAL()`** macros in conjunction. Memory allocated by this macro will be freed when the script finishes execution, however, you should manually free the memory with the **`efree()`** function.

`ZVAL_*(zval **, ..)`

Set the value of a zval, the * standands for the type, ie, to set the value of a long the appropriate macro would be the **`ZVAL_LONG()`** macro. These macros are a substitute for having to set the type and value of a zval seperately.

```
/*  Without the ZVAL_*() macros */
Z_TYPE_PP(somezval) = IS_DOUBLE;
Z_DVAL_PP(somezval) = 32.11;

/*  With the ZVAL_*() macros */
ZVAL_DOUBLE(somezval, 32.11);
```

Manipulating zvals

zval_ctor & zval_dtor

The **zval_ctor** function allows you to copy the contents of one zval into another zval. The **zval_dtor** function frees the memory allocated when you copy one zval onto another.

Example, A useless example showing the use of zval_copy_ctor and zval_dtor

```
zval **current;
HashTable *ar = HASH_OF(*uAr);

zend_hash_internal_pointer_reset(ar);
while (zend_get_current_data(ar, (void **)&current) == SUCCESS)
{
    char *str;
    int len;
    zval tmp;

     if (Z_TYPE_PP(current) != IS_STRING) {
         tmp = **current;

         zval_copy_ctor(&tmp);
         convert_to_string(&tmp);

         str = Z_STRVAL(tmp);
         len = Z_STRLEN(tmp);
     } else {
         str = Z_STRVAL_PP(current);
         len = Z_STRLEN_PP(current);
     }

     if (Z_TYPE_PP(current) != IS_STRING)
         zval_dtor(&tmp);

     zend_hash_move_forward(ar);
 }

```

The PHP_MINFO() function

WHAT?

The **PHP_MINFO()** function contains information about the current module that should be printed out when you call the **phpinfo()** function.

Example, The `PHP_MINFO()` function

```
PHP_MINFO_FUNCTION(moduleName)
{
    php_print_table_start();
    php_print_table_row(2, "Module Name Support", "enabled");
    php_print_table_row(2, "Module Name Version", modName_version());
    php_print_table_end();
}
```

Accessing Objects

Accessing Object Properties

To access object properties use the `zend_hash_find()` function on the `value.obj.properties` of an object.

Example: Accessing an object property

```
PHP_FUNCTION(print_name)
{
    zval **uObject, **name;
    HashTable *obj;

    if (ZEND_NUM_ARGS() != 1 ||
        zend_get_parameters_ex (1, &uObject) == FAILURE) {
        WRONG_PARAM_COUNT;
    }
     convert_to_object_ex(uObject);

     obj = HASH_OF(*uObject);

     if (zend_hash_find(obj, "name", sizeof("name"), (void **)&name) == SUCCESS) {
         SEPERATE_ZVAL(name);
         convert_to_string_ex(name);

         PUTS(Z_STRVAL_PP (name));
     } else {
         php_error(E_WARNING, "Cannot find name property");
         RETURN_NULL();
     }
 }
```

Registering a resource

HOW?

To register a new resource create a resource type and register a list destructor for the resource (like we did in the previous slides), then simply use the `ZEND_REGISTER_RESOURCE()` macro to register your resource.

Example, Registering a resource and returning the identifier to the user

```
PHP_FUNCTION(myModule_open)
{
    zval **filename;
    MyResourcePointer *rh;

    if (ZEND_NUM_ARGS() != 1 ||
        zend_get_parameters_ex (1, &filename) == FAILURE) {
        WRONG_PARAM_COUNT;
    }
     convert_to_string_ex (filename);

     rh = myModule_open (Z_STRVAL_PP(filename));

     ZEND_REGISTER_RESOURCE (return_value, rh, le_myModule);
 }
```

Resource Identifiers

WHAT?

PHP keeps track of different types of resources, such as file handles or database handles through the use of what are known as Resource Identifiers. When you register a resource Zend stores your resource in its resource table and returns a resource identifier.

le_whatever

The term "Resources" is very broad, both a MySQL pointer and a GD Image pointer fit under the class of a resource yet they are completely different. Therefore, to better clarify what resource is what type, you always have a resource type identifier usually named '`le_extname`' which keeps track of the current resource type.

Resource Management with PHP

An Outline

- Resource Indentifiers
- List destructors
- Registering Resources
- Fetching Resources
- Deleting Resources

Returning Arrays

HOW?

To return an array from a function first initialize the **`return_value`** with the **`array_init()`** function then add values to it using the **`add_*()`** functions.

Example, Returning an array from a function

```
PHP_FUNCTION(ret_array)
{
    if (array_init (return_value) == FAILURE) {
        php_error(E_WARNING, "Cannot initialize return value");
        RETURN_NULL();
    }

     add_index_long (return_value, 0, 32);
     add_next_index_double (return_value, 34.32);
     add_assoc_long (return_value, "long_val", 32);
     add_next_index_stringl (return_value, "some string",
          strlen("some string"), 1);
     add_assoc_string (return_value, "string_val", "Hello World",  1);
 }
```

Returning Multi-dimensional Arrays

HOW?

To return multi-dimensional arrays from your function, build a normal array and then use the **`zend_hash_next_index_insert()`** function to insert your array into the parent array.

Example, Returning a multi-dimensional array from a function

```
PHP_FUNCTION(return_multi)
{
    zval *childAr;

    if (array_init (childAr) == FAILURE) {
        php_error (E_ERROR, "Cannot initialize childAr from
              return_multi");
        RETURN_NULL();
    }

     add_next_index_long(childAr, 32);
     add_next_index_string(childAr, "Hello World", 1);

     if (array_init (return_value) == FAILURE) {
         php_error (E_ERROR, "Cannot initialize return value
             from return_multi");
         RETURN_NULL();
     }

     if (zend_hash_next_index_insert (return_value, &childAr, sizeof
         (zval *), NULL) == FAILURE) {
         php_error(E_WARNING, "Cannot insert childAr into return_value");
         RETURN_NULL();
     }

     add_next_index_long (return_value, 32);
 }
```

Returning Objects

HOW?

Returning an object is similair to returning an array, initialize the `return_value` using the `object_init()` function. Then you can add properties to the object using the `add_property_*()` functions.

Example, Returning an object from a function

```
PHP_FUNCTION(ret_object)
{
    if (object_init (return_value) == FAILURE) {
        php_error (E_ERROR, "Cannot initialize return value from
             ret_object");
        RETURN_NULL();
    }

     add_property_long(return_value, "some_long", 32);
     add_property_double(return_value, "some_double", 47.32);
     add_property_string(return_value, "some_string", "Hello World", 1);
 }
```

Returning Values

HOW?

To return basic values from PHP you can use the **`RETURN_*()`** macros.

Example, Returning a long from a function

```
PHP_FUNCTION(somefunction)
{
    int ret;
    ret = some_other_func ();

    RETURN_LONG(ret);
}
```

Example, Returning a double from a function

```
PHP_FUNCTION(somefunction)
{
    double ret;
    ret = some_other_other_func ();

    RETURN_DOUBLE(ret);
}
```

Example, Returning a string from a function

```
PHP_FUNCTION(somefunction)
{
    char *ret;
    ret = some_other_other_other_func ();

    RETURN_STRING(ret, 1);
}
```

PHP & C types

WHAT?

Basic PHP variables represented in C can be one of three types, they can be **`long`**'s, **`double`**'s, or '**`char *`**' strings. To access the specific values of PHP variables you must know there types and then use the appropriate macros.

Example: Accessing PHP variables of different types

```
PHP_FUNCTION (somefunc)
{
    zval **uStringVar, **uLongVar, **uDoubleVar;
    char *string_val, *ret;
    int string_len, long_val;
    double double_val;

    if (ZEND_NUM_ARGS() != 3 ||
        zend_get_parameters_ex (3, &uStringVar, &uLongVar, &uDoubleVar)
          == FAILURE) {
         WRONG_PARAM_COUNT;
     }

     /* Make sure the value of uStringVar is a 'char *' string */
     convert_to_string_ex (uStringVar);

     /* Make sure the value of uLongVar is a 'long' */
     convert_to_long_ex (uLongVar);

     /* Make sure the value of uDoubleVar is a 'double' */
     convert_to_double_ex (uDoubleVar);

     string_val = Z_STRVAL_PP (uStringVar);
     string_len = Z_STRLEN_PP (uStringVar);

     long_val   = Z_LVAL_PP (uLongVar);

     double_val = Z_DVAL_PP (uDoubleVar);
     if (array_init (return_value) == FAILURE) {
         php_error (E_ERROR, "Cannot initialize return value from somefunc");
         RETURN_FALSE;
     }

     add_assoc_stringl (return_value, "string_value", string_val,
             string_len, 1);
     add_assoc_double (return_value, "double_value", double_val);
     add_assoc_long (return_value, "long_value", long_val);
 }
```

./ext_skel

WHAT?

The **`ext_skel`** program will create a extension skeleton given the name of your module. Run the ext_skel script from the php4/ext directory.

Example, Using `ext_skel` to create a CURL extension skeleton

```
% cd php4
% cd ext
% ./ext_skel -
-
extname=curl -
-no-help
```

Function Entry

WHAT?

The function entry contains a list of all the different PHP "user" available functions in the current module.

A SAMPLE FUNCTION ENTRY

```
function_entry sample_functions[] = {
    PHP_FE (sample_function1,        NULL)
    PHP_FE (sample_function2,        first_arg_force_ref)
    PHP_FE (sample_function3,        NULL)
};
```

Module Entry

WHAT?

The module entry contains information about the current module, such as the module's functions, the module's name and other information.

A SAMPLE MODULE ENTRY

```
zend_module_entry samp_module_entry = {
    "sample",
    sample_functions,
    PHP_MINIT(sample),
    PHP_MSHUTDOWN(sample),
    PHP_RINIT(sample),
    PHP_RSHUTDOWN(sample),
    PHP_MINFO(sample),
    STANDARD_MODULE_PROPERTIES
 };
```

The PHP4 Extension API

Outline

- Module Entry
- Function Entry

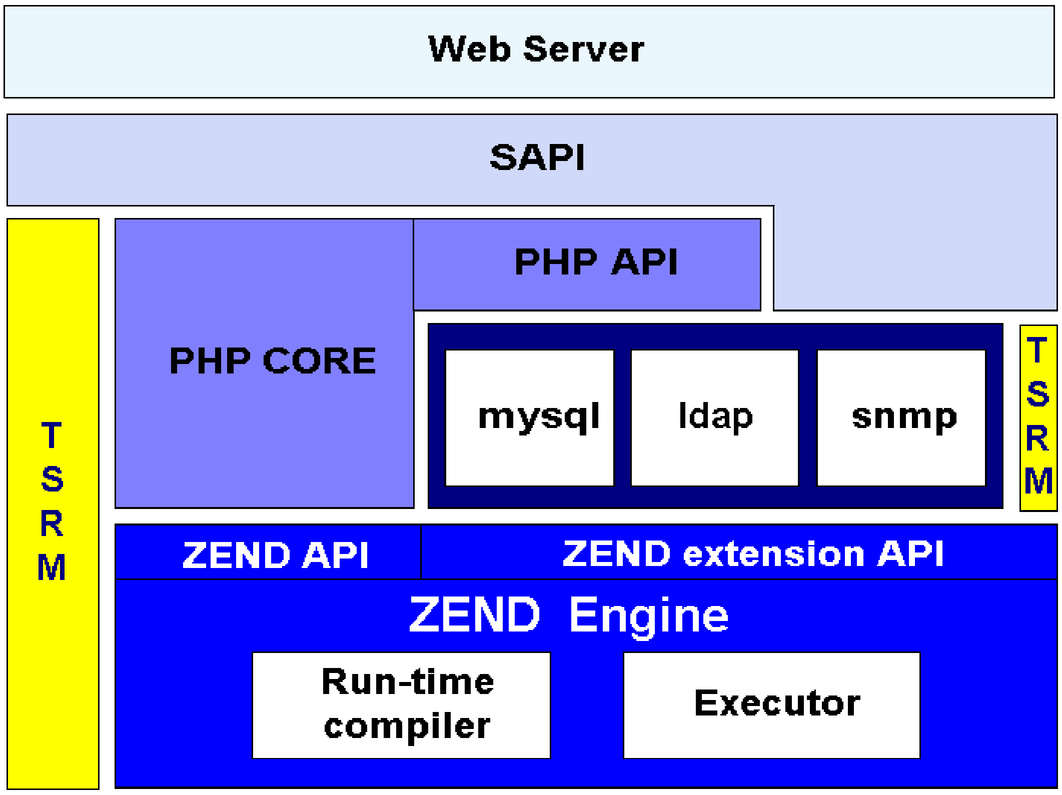
- An API to Manipulate PHP

Required Elements of a PHP4 Extension

- A Source file with the neccessary elements.
- A configuration file (named `config.m4`).
- A Makefile (named `Makefile.in`).

The Extension API

- It is a high level interface that allows you to extend PHP using C.
- All of the PHP extensions, such as the *MySQL* extension or the *SWF* extension, use the Extension API.
- The Extension API consists of three parts
    - Module Entry
    - Function Entry
    - An API to manipulate PHP



Part 1:
The Extension API

How PHP Works

An overview

Creating a self contained extension

WHAT?

Self contained extensions are extensions that can be distributed seperately from PHP itself. To create a self contained extension simply create a normal PHP extension and then run the **`phpize`** program in your source directory.

Example, making CURL a self-contained extension

```
% cd curl
% phpize
```

Where to get more information

- LXR, http://lxr.php.net/
- README.SELF-CONTAINED-EXTENSIONS
- README.EXT_SKEL
- Zend, http://www.zend.com/
- This presentation, http://conf.php.net/pres/
- Web Application Development with PHP by Tobias Ratschiller and Till Gerken.
- The PHP Developer's Cookbook by Sterling Hughes with Andrei Zmievski.

Modifying function parameters

HOW?

To modify function parameters with PHP, simply provide the PHP_FE() macro with a second argument that describes function parameters.

Example, Modifying the second argument

```

    char second_arg_force_ref[] = { BYREF_NONE, BYREF_FORCE };

    function_entry your_functions[] = {
        PHP_FE(one_func,      NULL)
        PHP_FE(modify_func,   second_arg_force_ref)
    };

```