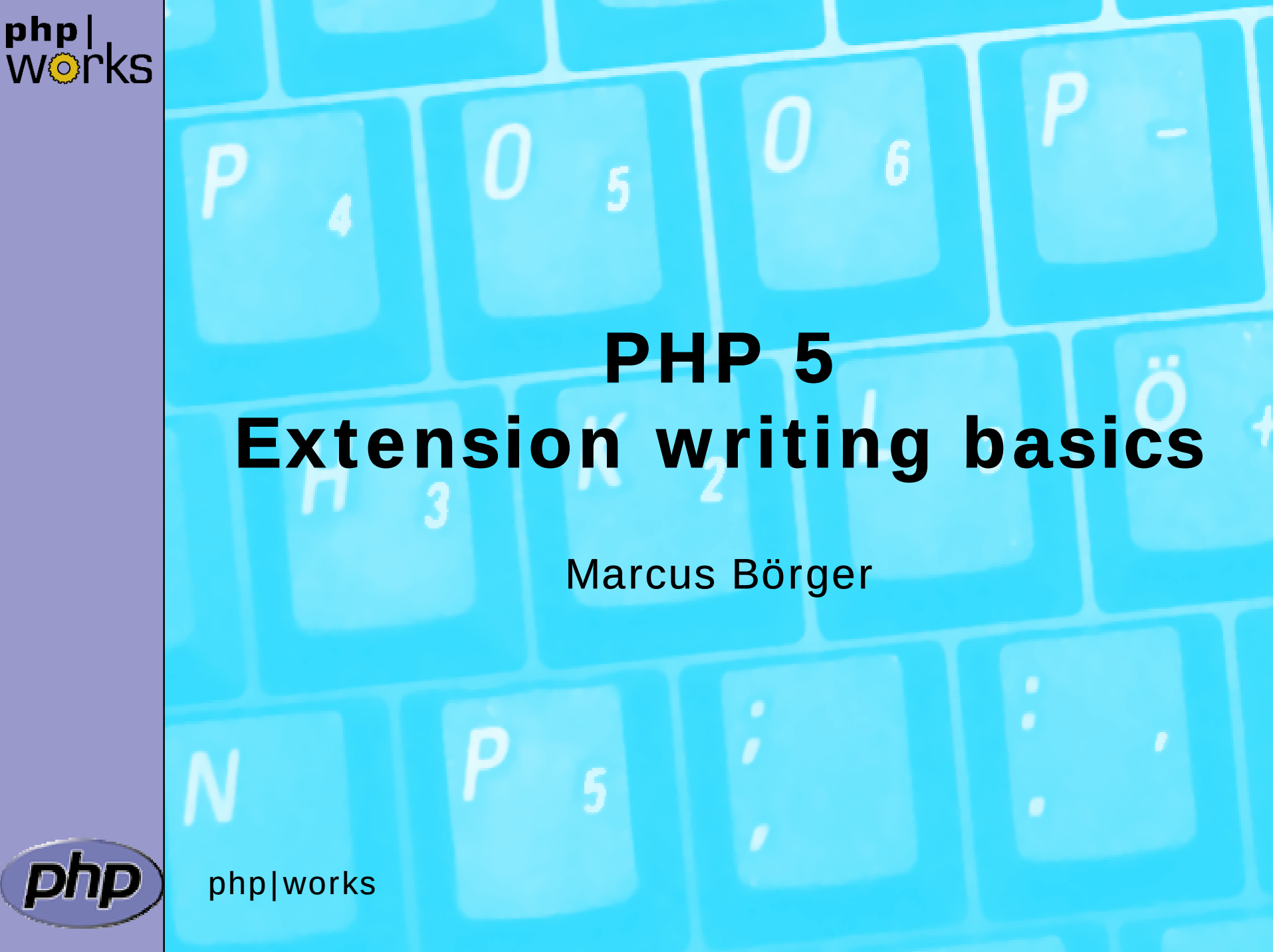


PHP 5
Extension writing basics

Marcus Börger

Creating PHP 5 Extensions

- ☑ How to create your own extension skeleton
- ☑ How PHP handles data
- ☑ How to create your own functions
- ☑ How to work with arrays and hash tables

Creating PHP 5 Extensions

- ☑ PHP 5 extensions are the same as in PHP 4
- ☑ ext_skel generates the basic skeleton

```
marcus@zaphod src/php5/ext $ ./ext_skel --extname=util
Creating directory util
Creating basic files: config.m4 .cvsignore util.c php_util.h CREDITS
EXPERIMENTAL tests/001.phpt util.php [done].

To use your new extension, you will have to execute the following steps:

1.  $ cd ..
2.  $ vi ext/util/config.m4
3.  $ ./buildconf --force
4.  $ ./configure --[with|enable]-util
5.  $ make
6.  $ ./php -f ext/util/util.php
7.  $ vi ext/util/util.c
8.  $ make

Repeat steps 3-6 until you are satisfied with ext/util/config.m4 and
step 6 confirms that your module is compiled into PHP. Then, start
writing code and repeat the last two steps as often as necessary.
```

Necessary for non cvs source (e.g. release packages)

How the slides work

- ☑ Upper part contains some *helpfull* hints
- ☑ Lower part shows c code on blue background

```
Text in yello   Text you should use as presented

Text in green   Text that you have to replace

yourext         Extension name in lowercase
YOUREXT         Extension name in uppercase
YourExt         Extension name in mixed case (camelCaps)
```

Some special explanation
use red text boxes

Files in your extension

- ☑ You need at least two code files
  - ☑ php_*yourext*.h — The header needed by php
  - ☑ php_*yourext*.c — The main extension code ('php_' prefix for .c is not necessary)
- ☑ You need two configuration files
  - ☑ config.m4 — Used under *nix
  - ☑ config.w32 — Used under windows
- ☑ Additional files
  - ☑ .cvsignore — List of files to be ignored by CVS
  - ☑ CREDITS — First line ext name 2nd line all authors
  - ☑ EXPERIMENTAL — If available the API is not yet stable
  - ☑ package.xml — Required for PECL extensions
  - ☑ README — Probably good to provide some lines

config.m4

- ☑ PHP Dev is picky about coding style
  - ☑ Watch your whitespace
  - ☑ Align your PHP_ARG_ENABLE output
- ☑ Make your extension default disabled
  - ☑ 'phpize' or 'pear install' will enable it automatically

```
dnl $Id: $
dnl config.m4 for extension YOUREXT
PHP_ARG_ENABLE(yourext, disable YourExt suppport,
  [  --enable-yourext          Enable YourExt], no)
if test "$PHP_YOUREXT" != "no"; then
  AC_DEFINE(HAVE_YOUREXT, 1,[Whether YourExt is present])
  PHP_NEW_EXTENSION(yourext, php_yourext.c, $ext_shared)
fi
```

config.m4

- ☑ You can prevent the ext from becoming shared

```
dnl $Id: $
dnl config.m4 for extension YOUREXT
PHP_ARG_ENABLE(yourext, disable YourExt suppport,
  [  --enable-yourext          Enable YourExt], no)
if test "$PHP_YOUREXT" != "no";then
  if test "$ext_shared" = "yes"; then
    AC_MSG_ERROR(Cannot build YOUREXT as a shared module)
  fi
  AC_DEFINE(HAVE_YOUREXT,1,[Whether YourExt is present])
  PHP_NEW_EXTENSION(yourext, php_yourext.c, $ext_shared)
fi
```

config.m4

- You can add module dependencies (even to libxml)
  - Optional dependencies are possible (true as 3rd param)

```
dnl $Id: $
PHP_ARG_ENABLE(yourext, disable YourExt suppport,
  [  --enable-yourext          Ensable YourExt], no)
if test "$PHP_YOUREXT"!="no" -a "$PHP_LIBXML"!="no";then
  if test "$ext_shared" = "yes"; then
    AC_MSG_ERROR(Cannot build YOUREXT as a shared module)
  fi
  PHP_SETUP_LIBXML(YOUREXT_SHARED_LIBADD, [
    AC_DEFINE(HAVE_YOUREXT,1,[Whether YourExt is present])
    PHP_NEW_EXTENSION(yourext, php_yourext.c, $ext_shared)
    PHP_SUBST(YOUREXT_SHARED_LIBADD)
  ], [
    AC_MSG_ERROR([xml2-config not found, check libxml2])
  ]
  PHP_ADD_EXTENSION_DEP(yourext, libxml, false)
fi
```

config.w32

- ☑ Windows configuration uses JScript

```
// $Id: $
// vim:ft=javascript
ARG_WITH("yourext", "YourExt support", "yes");
if (PHP_YOUREXT == "yes" && PHP_LIBXML == "yes") {
  if (PHP_YOUREXT_SHARED) {
    ERROR("YOUREXT cannot be compiled as a shared ext");
  }
  AC_DEFINE("HAVE_YOUREXT", 1, "YourExt support");
  EXTENSION("yourext", "php_yourext.c");
  if (!PHP_YOUREXT_SHARED) {
    ADD_FLAG("CFLAGS_YOUREXT", "/D LIBXML_STATIC");
  }
  ADD_EXTENSION_DEP('yourext', 'libxml', false);
}
```

Header of .h and .c

- License, Authors, CVS-Tag
  - PECL accepts PHP License, (LGPL) and compatible
  - PECL does NOT accept GPL

```
/*
   +----------------------------------------------------------------------+
   | PHP Version 5                                                        |
   +----------------------------------------------------------------------+
   | Copyright (c) 1997-2005 The PHP Group                                |
   +----------------------------------------------------------------------+
   | This source file is subject to version 3.0 of the PHP license,       |
   | that is bundled with this package in the file LICENSE, and is        |
   | available through the world-wide-web at the following url:           |
   | http://www.php.net/license/3_0.txt.                                  |
   | If you did not receive a copy of the PHP license and are unable to   |
   | obtain it through the world-wide-web, please send a note to          |
   | license@php.net so we can mail you a copy immediately.               |
   +----------------------------------------------------------------------+
   | Authors: Marcus Boerger <helly@php.net>                              |
   +----------------------------------------------------------------------+
 */

/* $Id: $ */
```

Extension .h file

```
// License Author, CVS-Tag

#ifndef PHP_YOUREXT_H
#define PHP_YOUREXT_H
#include "php.h"

extern zend_module_entry yourext_module_entry;
#define phpext_yourext_ptr &yourext_module_entry

#ifdef PHP_WIN32
#  define YOUREXT_API __declspec(dllexport)
#else
# define YOUREXT_API
#endif

// Place for globals definition

#endif /* PHP_YOUREXT_H */
/* * Local Variables:
 * c-basic-offset: 4
 * tab-width: 4
 * End:
 * vim600: fdm=marker
 * vim: noet sw=4 ts=4
 */
```

Layout of the .c file

- ☑ Header: License, Authors, CVS-Tag, ...
- ☑ Includes
- ☑ Structures and defines not in header
- ☑ Helper Functions
- ☑ PHP Functions
- ☑ Globals Handling
- ☑ MINFO
- ☑ MINIT, MSHUTDOWN
- ☑ RINIT, RSHUTDOWN
- ☑ Function table
- ☑ Module Entry

Includes

- Include path:
  - <PHP Root>/
  - <PHP Root>/Zend
  - <PHP Root>/main
  - <PHP Root>/ext/<Your Extension>

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

#include "php.h"
#include "php_ini.h"
#include "php_yourext.h"
```

Structures and defines not in header

- ☑ What ever you want

?

Helper Functions

- ☑ Use static
  - If you need the funtion only in your .c file
- ☑ Use PHPAPI / *YOREXT*_API
  - If you plan to use the functions in other extensions
- ☑ Use TSRMLS_xx as last function parameter
  - When dealing with PHP Data

Helper Functions

- ☑ Use static
  - If you need the funtion only in your .c file
- ☑ Use PHPAPI
  - If you plan to use the functions in other extensions
- ☑ Use TSRMLS_xx as last function parameter
  - When dealing with PHP Data

TSRMLS_D	in declarations as only param
TSRMLS_C	in implementations as only param

```
static void my_helper(TSRMLS_D);

static void some_function(TSRMLS_D) {
      my_helper(TSRMLS_C);
}
```

Helper Functions

- ☑ Use static
  - If you need the funtion only in your .c file
- ☑ Use PHPAPI
  - If you plan to use the functions in other extensions
- ☑ Use TSRMLS_xx as last function parameter
  - When dealing with PHP Data

TSRMLS_D	in declarations as only param
TSRMLS_DC	in declarations after last param w/o comma
TSRMLS_C	in implementations as only param
TSRMLS_CC	in impl. after last param w/o comma

```
static void my_helper(void * p TSRMLS_DC);

static void some_function(void * p TSRMLS_DC) {
      my_helper(p TSRMLS_CC);
}
```

Helper Functions

- ☑ Use static
  - If you need the funtion only in your .c file
- ☑ Use PHPAPI
  - If you plan to use the functions in other extensions
- ☑ Use TSRMLS_xx as last function parameter
  - When dealing with PHP Data

TSRMLS_D	in declarations as only param
TSRMLS_DC	in declarations after last param w/o comma
TSRMLS_C	in implementations as only param
TSRMLS_CC	in impl. after last param w/o comma
TSRMLS_FETCH	create a TSRM key, must follow last local var

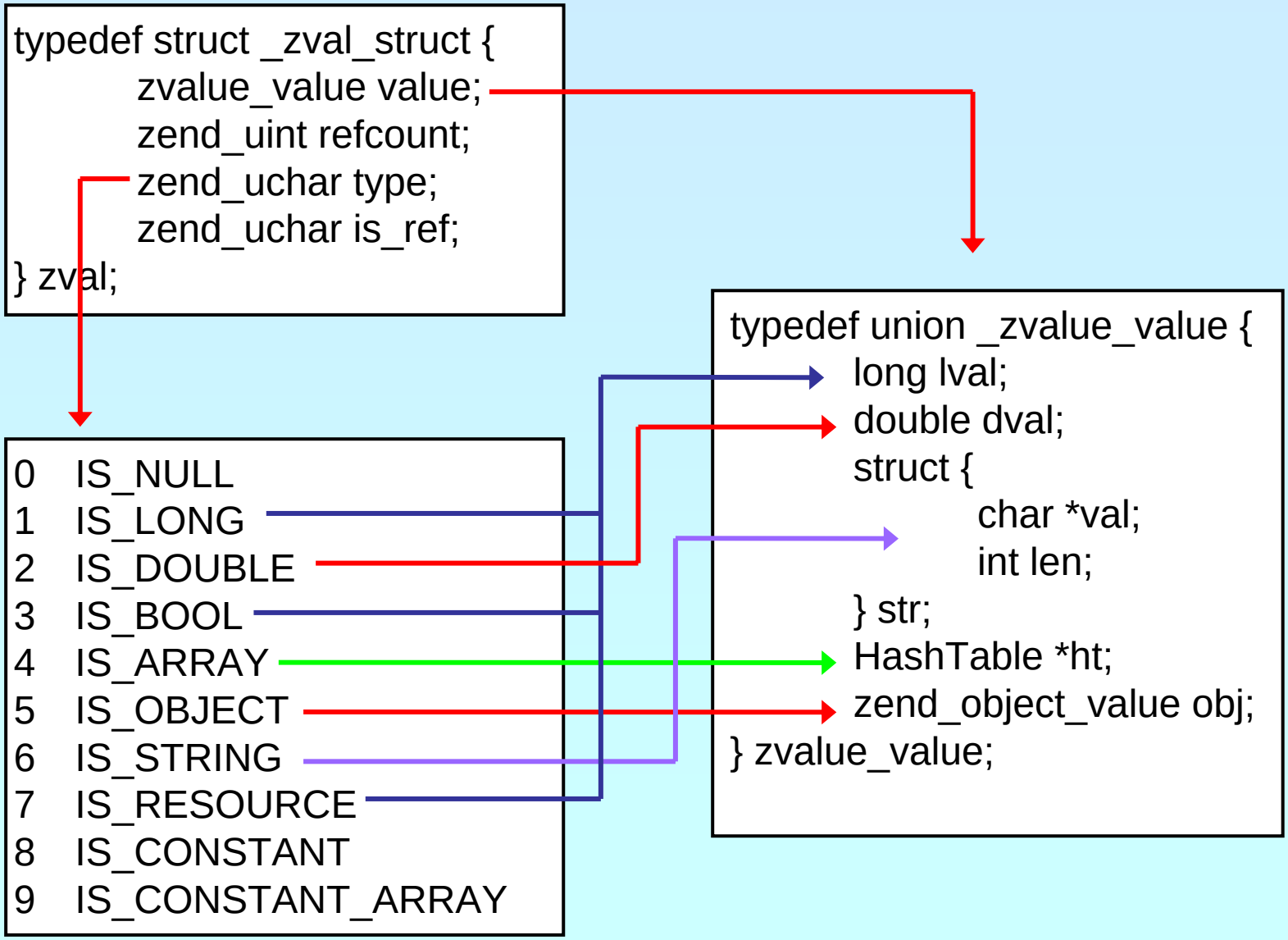
```
static void my_helper(void * p TSRMLS_DC);

static void some_function(void * p) {
      TSRMLS_FETCH();
      my_helper(p TSRMLS_CC);
}
```

PHP Functions

- ☑ Always use the layout below

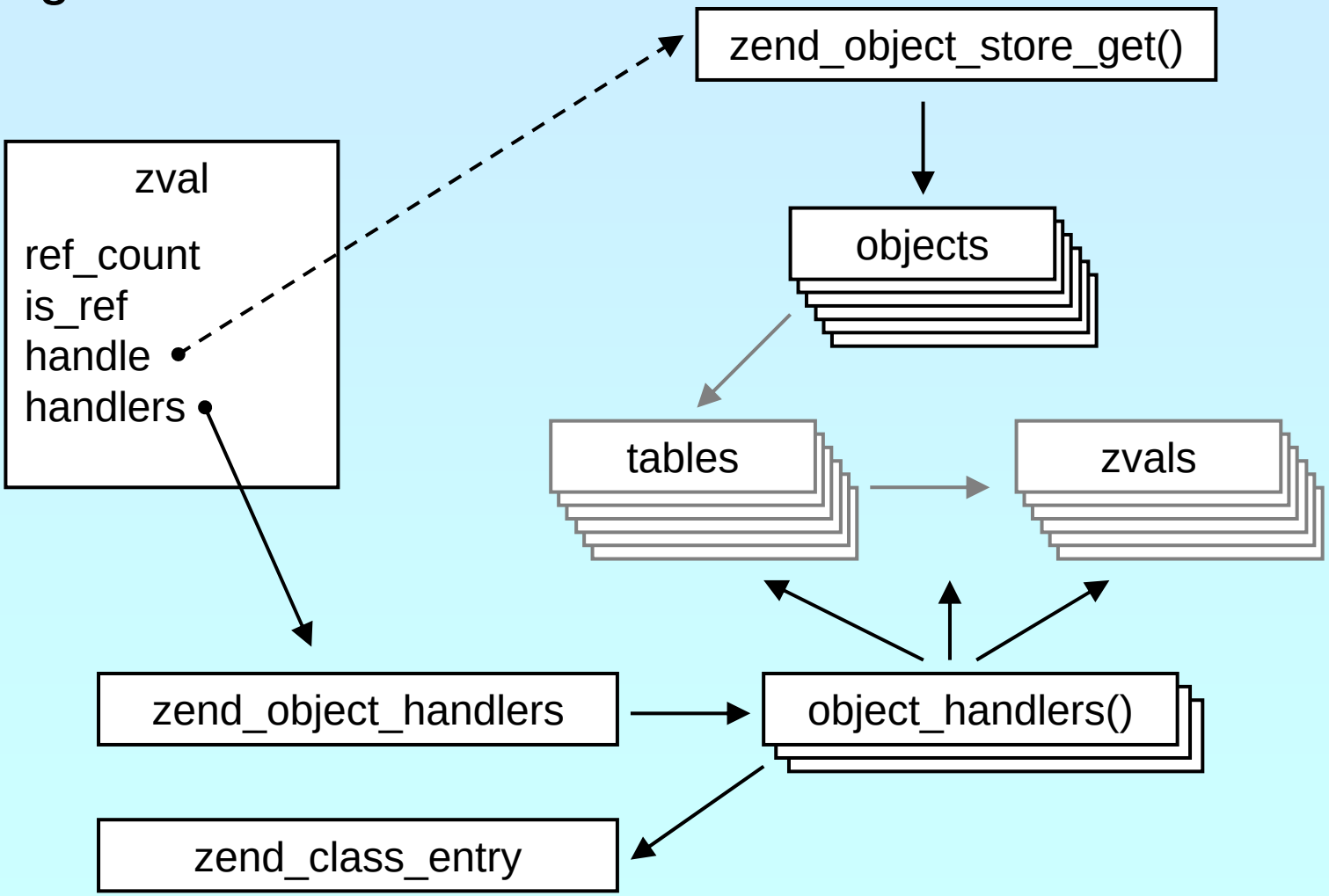
```
/* {{{ proto type yourext_name(params)
  Short description */
PHP_FUNCTION(yourext_name)
{
    // Local declarations

    // Parameter parsing

    // Actual code

    // Return value
}
/* }}} */
```

In PHP all values are zval's



Objects?

- ☑ Forget about this for now



Parsing parameters

☑ zend_parse_parameters is the easy way of parsing

```
int zend_parse_parameters(
    int num_args TSRMLS_DC, char *type_spec, ...);

int zend_parse_parameters_ex(int flags,
    int num_args TSRMLS_DC, char *type_spec, ...);

flags       0 or ZEND_PARSE_PARAMS_QUIET
num_args    use ZEND_NUM_ARGS()
type_spec   sscanf like typelist (though no %)
returns     SUCCESS or FAILURE
            in case of failure an error is already issued
            so no need for ZEND_WRONG_PARAM_COUNT()
            unless using ZEND_PARSE_PARAMS_QUIET
```

Parsing parameters

```
type_spec   sscanf like typelist (though no %)
    l       long      long *
    d       double    double *
    b       boolean   zend_bool *
    a       array     zval **
    o       object    zval **
    O       object    zval **, zend_class_entry *
              Object must be derived from given class
    s       string    char **, int *
              You receive string and length
    r       resource  zval **
    z       zval      zval **
    Z       zval-ref  zval ***

    |       right part is optional
    /       next param gets separated if not reference
    !       Next param returns NULL if param type IS_NULL
```

Setting a zval

- Use ZVAL_<type>() macros
  - Nothing is freed or destructed
  - Type is set to IS_<type>

```
ZVAL_RESOURCE(z, l)          l: long
ZVAL_BOOL(z, b)              b: 0/1 (not 0)
ZVAL_FALSE(z)                ZVAL_BOOL(z, 0)
ZVAL_TRUE(z)                 ZVAL_BOOL(z, 1)
ZVAL_NULL(z)                 just sets the type to IS_NULL
ZVAL_LONG(z, l)              l: long
ZVAL_DOUBLE(z, d)            d: double
ZVAL_STRING(z, s, dup)       s: char *, dup: 0/1 (duplicate)
ZVAL_STRINGL(z, s, l, dup)   s: char *, l: length, dup: 0/1
ZVAL_EMPTY_STRING(z)         set z to an empty string
ZVAL_ZVAL(z,zv,dup,dtor)     zv: other zval*, dup: 0/1,
                             dtor: 0/1 (whether to call dtor)
```

Setting the return value

- The return value is already allocated and IS_NULL
  - These macros do not end the function

```
RETVAL_RESOURCE(l)          ZVAL_RESOURCE(return_value, l)
RETVAL_BOOL(b)              ZVAL_BOOL(return_value, b)
RETVAL_FALSE                ZVAL_BOOL(return_value, 0)
RETVAL_TRUE                 ZVAL_BOOL(return_value, 1)
RETVAL_NULL()               ZVAL_NULL(return_value)
RETVAL_LONG(l)              ZVAL_LONG(return_value, l)
RETVAL_DOUBLE(d)            ZVAL_DOUBLE(return_value, d)
RETVAL_STRING(s, dup)       ZVAL_STRING(return_value, s, dup)
RETVAL_STRINGL(s,l,d)       ZVAL_STRINGL(return_value,s,l,d)
RETVAL_EMPTY_STRING()       ZVAL_EMPTY_STRING(return_value)
RETVAL_ZVAL(zv,dup,dtor)    ZVAL_ZVAL(return_value,zv,dup,dtor)
```

Set return value and return

- Just like RETVAL_<type> but returning directly

```
RETURN_RESOURCE(l) {RETVAL_RESOURCE(return_value,l);return;}
RETURN_BOOL(b)     {RETVAL_BOOL(return_value, b); return;}
RETURN_FALSE       {RETVAL_FALSE; return;}
RETURN_TRUE        {RETVAL_TRUE; return;}
RETURN_NULL()      {RETVAL_NULL(return_value); return;}
RETURN_LONG(l)     {RETVAL_LONG(return_value, l); return;}
RETURN_DOUBLE(d)   {RETVAL_DOUBLE(return_value, d); return;}
RETURN_STRING(s, dup)
     {RETVAL_STRING(return_value, s, dup); return;}

RETURN_STRINGL(s,l,d)
     {RETVAL_STRINGL(return_value,s,l,d); return;}

RETURN_EMPTY_STRING()
     {RETVAL_EMPTY_STRING(return_value); return;}

RETURN_ZVAL(zv,dup,dtor)
     {RETVAL_ZVAL(return_value,zv, dup,dtor); return;}
```

Example 1

- Inverting a single boolean parameter

```
/* {{{ proto bool yourext_invert(bool b)
 Invert a boolean parameter */
PHP_FUNCTION(yourext_invert)
{
    zend_bool b;

    if (zend_parse_parameters(ZEND_NUM_ARGS() TSRMLS_CC,
            "b", &b) == FAILURE) {
        return;
    }

    b = b ? 0 : 1;

    RETURN_BOOL(b);
}
/* }}} */
```

Makes the return value NULL

Example 2

- Incrementing a value with an optional maximum

```
/* {{{ proto bool yourext_increment(int v [, int max])
 Increment a value with optional maximum */
PHP_FUNCTION(yourext_increment)
{
    long l, lmax = LONG_MAX;

    if (zend_parse_parameters(ZEND_NUM_ARGS() TSRMLS_CC,
            "l|l", &l, &lmax) == FAILURE) {
        RETURN_FALSE();
    }

    l = (l+1) % lmax;

    RETURN_LONG(l);
}
/* }}} */
```

Initialize optional values

Use brackets for optional values

A vertical bar separates optional and required parameters

Example 3

- Returning some generated string

```
#define YOUREXT_VERSION_MAJOR    0
#define YOUREXT_VERSION_MINOR    1

/* {{{ proto bool yourext_version()
 Retrieve yourext version */
PHP_FUNCTION(yourext_version)
{
    char * ver;
    int len;

    len = spprintf(&ver, 0, "%d.%d (%s)",
        YOUREXT_VERSION_MAJOR, YOUREXT_VERSION_MINOR,
        "$Id: $");

    RETURN_STRINGL(ver, len, 0);
}
/* }}} */
```

Never use sprintf,
use either snprintf or spprintf

Dealing with arrays

- To initialize a zval as an array: `array_init(zv)`
  - To return an array use: `array_init(return_value)`
- To add elements use the following
  - `add_assoc_<type>(ar, key, ...)`

```
int add_assoc_long(zval *arg, char *key, long n);
int add_assoc_null(zval *arg, char *key);
int add_assoc_bool(zval *arg, char *key, int b);
int add_assoc_resource(zval *arg, char *key, int r);
int add_assoc_double(zval *arg, char *key, double d);
int add_assoc_string(zval *arg, char *key, char *str,
                     int duplicate);
int add_assoc_stringl(zval *arg, char *key, char *str,
                      uint length, int duplicate);
int add_assoc_zval(zval *arg, char *key, zval *value);
```

Dealing with arrays

- To convert a zval into an array: `array_init(zv)`
  - To return an array use: `array_init(return_value)`
- To add elements use the following
  - `add_assoc_<type>(ar, key, ...)`
  - `add_index_<type>(ar, index, ...)`

```
int add_index_long(zval *arg, uint idx, long n);
int add_index_null(zval *arg, uint idx);
int add_index_bool(zval *arg, uint idx, int b);
int add_index_resource(zval *arg, uint idx, int r);
int add_index_double(zval *arg, uint idx, double d);
int add_index_string(zval *arg, uint idx, char *str,
                     int duplicate);
int add_index_stringl(zval *arg, uint idx, char *str,
                     uint length, int duplicate);
int add_index_zval(zval *arg, uint idx, zval *value);
```

Dealing with arrays

- To convert a zval into an array: `array_init(zv)`
  - To return an array use: `array_init(return_value)`
- To add elements use the following
  - `add_assoc_<type>(ar, key, ...)`
  - `add_index_<type>(ar, index, ...)`
  - `add_next_index_<type>(ar, ...)`

```
int add_next_index_long(zval *arg, long n);
int add_next_index_null(zval *arg);
int add_next_index_bool(zval *arg, int b);
int add_next_index_resource(zval *arg, int r);
int add_next_index_double(zval *arg, double d);
int add_next_index_string(zval *arg, char *str,
                          int duplicate);
int add_next_index_stringl(zval *arg, char *str,
                          uint length, int duplicate);
int add_next_index_zval(zval *arg, zval *value);
```

Example 4

- Returning an array

```
/* {{{ proto bool yourext_version_array()
 Retrieve yourext version as array */
PHP_FUNCTION(yourext_version_array)
{
    char *ver;
    int len = spprintf(&ver, 0, "%d.%d",
        YOUREXT_VERSION_MAJOR, YOUREXT_VERSION_MINOR);

    array_init(return_value);
    add_assoc_long(return_value, "major",
        YOUREXT_VERSION_MAJOR);
    add_assoc_long(return_value, "minor",
        YOUREXT_VERISON_MINOR);
    add_assoc_string(return_value, "cvs", "$Id: $", 1);
    add_assoc_stringl(return_value, "ver", ver, len, 0);
}
/* }}} */
```

make return_value an array

Dealing with a HashTable

- ☑ PHP Arrays use SymbolTable, a special HashTable
  - ☑ Numeric keys are treated as hash indices
  - ☑ Non number indices are hashed
  - ☑ SymbolTable keys include terminating \0
    sizeof(key) vs. strlen(key)
- ☑ A HashTable knows about its element count

```
ulong zend_get_hash_value(char *arKey, uint nKeyLength);

int zend_hash_num_elements(HashTable *ht);
```

Dealing with a HashTable

- ☑ You can delete elements (SUCCESS/FAILURE)
  - ☑ by key
  - ☑ by hash index
  - ☑ by symbol

```
int zend_hash_del(HashTable *ht, char *arKey,
    uint nKeyLen);

int zend_hash_index_del(HashTable *ht, ulong h);

int zend_symtable_del(HashTable *ht, char *arKey,
    uint nKeyLength);
```

Dealing with a HashTable

- ☑ You can lookup elements (SUCCESS/FAILURE)
  - ☑ by key
  - ☑ by hash index
  - ☑ by automatic preference of hash index over key (len=0)
  - ☑ by symbol

```
int zend_hash_find(HashTable *ht, char *arKey,
    uint nKeyLength, void **pData);

int zend_hash_quick_find(HashTable *ht, char *arKey,
    uint nKeyLength, ulong h, void **pData);

int zend_hash_index_find(HashTable *ht, ulong h,
    void **pData);

int zend_symtable_find(HashTable *ht, char *arKey,
    uint nKeyLength);
```

Dealing with a HashTable

- ☑ You can check for existance of elements (0/1)
  - ☑ by key
  - ☑ by hash index
  - ☑ by automatic preference of hash index over key (len=0)
  - ☑ by symbol

```
int zend_hash_exists(HashTable *ht, char *arKey,
    uint nKeyLength);

int zend_hash_quick_exists(HashTable *ht, char *arKey,
    uint nKeyLength, ulong h);

int zend_hash_index_exists(HashTable *ht, ulong h);

int zend_symtable_exists(HashTable *ht, char *arKey,
    uint nKeyLength);
```

Dealing with a HashTable

☑ Hash tables support a builtin foreach

```
#define ZEND_HASH_APPLY_KEEP        0
#define ZEND_HASH_APPLY_REMOVE      1<<0
#define ZEND_HASH_APPLY_STOP        1<<1

typedef int (*apply_func_t)(void *pDest TSRMLS_DC);

typedef int (*apply_func_arg_t)(void *pDest, void *argument
            TSRMLS_DC);

typedef int (*apply_func_args_t)(void *pDest, int num_args,
            va_list args, zend_hash_key *hash_key);

void zend_hash_apply(HashTable *ht, apply_func_t apply_func
            TSRMLS_DC);

void zend_hash_apply_with_argument(HashTable *ht,
            apply_func_arg_t apply_func, void * TSRMLS_DC);

void zend_hash_apply_with_arguments(HashTable *ht,
            apply_func_args_t apply_func, int, ...);
```

Example 5 a

- Using zend_hash_apply_with_arguments()

```
/* {{{ proto void yourext_foreach(array ar, mixed func)
  Call a function for all elements: bool apply(mixed param)
PHP_FUNCTION(yourext_foreach)
{
   zval *ar, *zfunc;
   char *fname;

   if ((zend_parse_parameters_ex(ZEND_PARSE_PARAMS_QUIET
        ZEND_NUM_ARGS() TSRMLS_CC, "az", &ar, &func)
        == FAILURE &&
     zend_parse_parameters(ZEND_NUM_ARGS(), "oz", &ar,&func)
         == FAILURE)
     || !zend_is_callable(zfunc, 0, fname)) {
         return;
   }
   zend_hash_apply_with_argument(HASH_OF(ar),
         (apply_func_arg_t)yourext_foreach, zfunc TSRMLS_CC);
} /* }}} */
```

First check array, if that fails try object

Verify function is callable

Example 5 b

- Calling a function for each element

```
/* {{{ */
int yourext_foreach(zval **param, zval *func_name TSRMLS_DC)
{
    zval retval;
    zval *args[1];
    int status;

    args[0] = *param;
    status = call_user_function(EG(function_table), NULL,
        func_name, &retval, 1, args TSRMLS_CC);

    if (!zend_is_true(&retval)) status = FAILURE;
    zval_dtor(&retval);

    return status == SUCCESS
            ? ZEND_HASH_APPLY_KEEP
            : ZEND_HASH_APPLY_STOP;
} /* }}} */
```

retval must be destructed here but not freed

Dealing with a HashTable

- Hash tables need to be initialized
  - Number of initial elements
  - Function used to calculate hash indices from keys
    Though only DJBX33A is ever being used
  - Function used to destruct elements upon deletion
  - Whether elements are persistent (valid outside request)

```
typedef ulong (*hash_func_t)(char *arKey, uint nKeyLen);
typedef void (*dtor_func_t)(void *pDest);

int zend_hash_init(HashTable *ht, uint nSize,
    hash_func_t pHashFunction, dtor_func_t pDestructor,
    zend_bool persistent);

#define ZEND_INIT_SYMTABLE(ht) \
    ZEND_INIT_SYMTABLE_EX(ht, 2, 0)
#define ZEND_INIT_SYMTABLE_EX(ht, n, persist) \
    zend_hash_init(ht, n, NULL, ZVAL_PTR_DTOR, persist)
```

Dealing with a HashTable

- ☑ Hash tables can be cleaned
  - ☑ Fast removal and destruction of all elements
- ☑ Hash tables must be destroyed
  - ☑ Persistent hash tables in MSHUTDOWN()
  - ☑ Non persistent hash tables in RSHUTDOWN()

```
void zend_hash_clean(HashTable *ht);

void zend_hash_destroy(HashTable *ht);
```

Global struct in .h

- ☑ Provide a structure and access macros
- ☑ For hash tables both pointer and member works

```
ZEND_BEGIN_MODULE_GLOBALS(yourext)
    char *        global_string;
    HashTable *   global_hash;
ZEND_END_MODULE_GLOBALS(yourext)

#ifdef ZTS
# define YOUREXT_G(v) \
    TSRMG(yourext_globals_id, zend_yourext_globals*, v)
extern int yourext_globals_id;
#else
# define YOUREXT_G(v) (yourext_globals.v)
extern zend_yourext_globals yourext_globals;
#endif
```

Global Handling in .c

- ☑ Provide the storage/id and an initializer function
  - ☑ Hash tables need to be initialized in RINIT
  - ☑ Strings must be initialized/copied in RINIT
  - ☑ Strings must be either static or malloc'd

```
#ifdef COMPILE_DL_YOUREXT
ZEND_GET_MODULE(yourext)
#endif

ZEND_DECLARE_MODULE_GLOBALS(yourext)

static void yourext_init_globals(
     zend_yourext_globals *globals) /* {{{ */
{
     // Initialize your global struct
     globals->global_string = "somestring";
     globals->global_hash = NULL;
} /* }}} */
```

MINIT/MSHUTDOWN

- ☑ MINIT needs to initialize globals
- ☑ MSHUTDOWN
  - ☑ Needs to free malloc'd globals
  - ☑ Needs to destroy all persistent hash tables

```
PHP_MINIT_FUNCTION(yourext) /* {{{ */
{
    ZEND_INIT_MODULE_GLOBALS(yourext,
        yourext_init_globals, NULL);
    return SUCCESS;
} /* }}} */

PHP_MSHUTDOWN_FUNCTION(yourext) /* {{{ */
{
    // free global malloc'ed memory
    return SUCCESS;
} /* }}} */
```

RINIT/RSHUTDOWN

- ☑ Between RINIT/SHUTDOWN using zval/hash is OK
- ☑ Memory during request time must be ealloc'd
  - ☑ malloc -> emalloc, free -> efree, realloc -> erealloc
  - ☑ strdup -> estrdup, strndup -> estrndup

```
PHP_RINIT_FUNCTION(yourext) /* {{{ */
{
     YOUREXT_G(global_string) =
         estrdup(YOUREXT_G(global_string));
     ALLOC_HASHTABLE(YOUREXT_G(global_hash));
     zend_hash_init(YOUREXT_G(global_hash),
         1, NULL, NULL, 0);
     return SUCCESS;
} /* }}} */
```

RINIT/RSHUTDOWN

- After RSHUTDOWN no emalloc'd data is allowed
  - You need to keep track of manual added data
  - You need to destroy all non persistent hash tables

```
PHP_RSHUTDOWN_FUNCTION(yourext) /* {{{ */
{
    efree(YOUREXT_G(global_string));
    zend_hash_destroy(YOUREXT_G(global_hash));
    FREE_HASHTABLE(YOUREXT_G(global_hash));
    return SUCCESS;
} /* }}} */
```

MINFO

- Provide some information about your extension
  - MINFO has no return value

```
PHP_MINFO_FUNCTION(yourext) /* {{{ */
{
   php_info_print_table_start();
   php_info_print_table_header(2, "YourExt","enabled");

   php_info_print_table_row(2,
      "Version", "$ID: $");

   php_info_print_table_row(2,
      "Somestring", YOUREXT_G(global_string));

   php_info_print_table_end();
}/* }}} */
```

Function Table

- The function table allows to specify the signature
  - ZEND_BEGIN_ARG_INFO_EX:
    name, pass_rest_by_ref, return_ref, required_args
  - ZEND_ARG_INFO:
    pass_by_ref, name

```
static
ZEND_BEGIN_ARG_INFO_EX(yourext_arginfo_name1, 0, 0, 2)
    ZEND_ARG_INFO(0, param_name1)
    ZEND_ARG_INFO(0, param_name2)
ZEND_END_ARG_INFO();

function_entry yourext_functions[] = { /* {{{ */
    PHP_FE(yourext_name1,    yourext_arginfo_name1)
    PHP_FE(yourext_name2,    NULL)
    PHP_FE(yourext_name3,    NULL)
    {NULL, NULL, NULL}
}; /* }}} */
```

Module Entry

- ☑ Keeps everything together
- ☑ Tells PHP how to (de)initialize the extension

```
zend_module_entry yourext_module_entry = { /* {{{ */
    STANDARD_MODULE_HEADER,
    "YourExt",
    yourext_functions,
    PHP_MINIT(yourext),
    PHP_MSHUTDOWN(yourext),
    PHP_RINIT(yourext),
    PHP_RSHUTDOWN(yourext),
    PHP_MINFO(yourext),
    "0.1",
    STANDARD_MODULE_PROPERTIES
}; /* }}} */
```

or NULL

What else ?

- ☑ INI Handling – but avoid it by all means
- ☑ Dealing with resources and streams
- ☑ Providing classes and methods
- ☑ Providing Iterators
- ☑ Overloading objects with several handlers
  - ☑ Array access
  - ☑ Property access
  - ☑ Serializing

References

- ☑ This presentation
  http://talks.somabo.de
- ☑ Documentation and Sources to PHP5
  http://php.net
- ☑ http://www.zend.com/php/internals
- ☑ Advanced PHP Programming
  by George Schlossnagle
- ☑ Extending and Embedding PHP
  by Sara Golemon
  ISBN#0-6723-2704-X
  (Spring 2006)