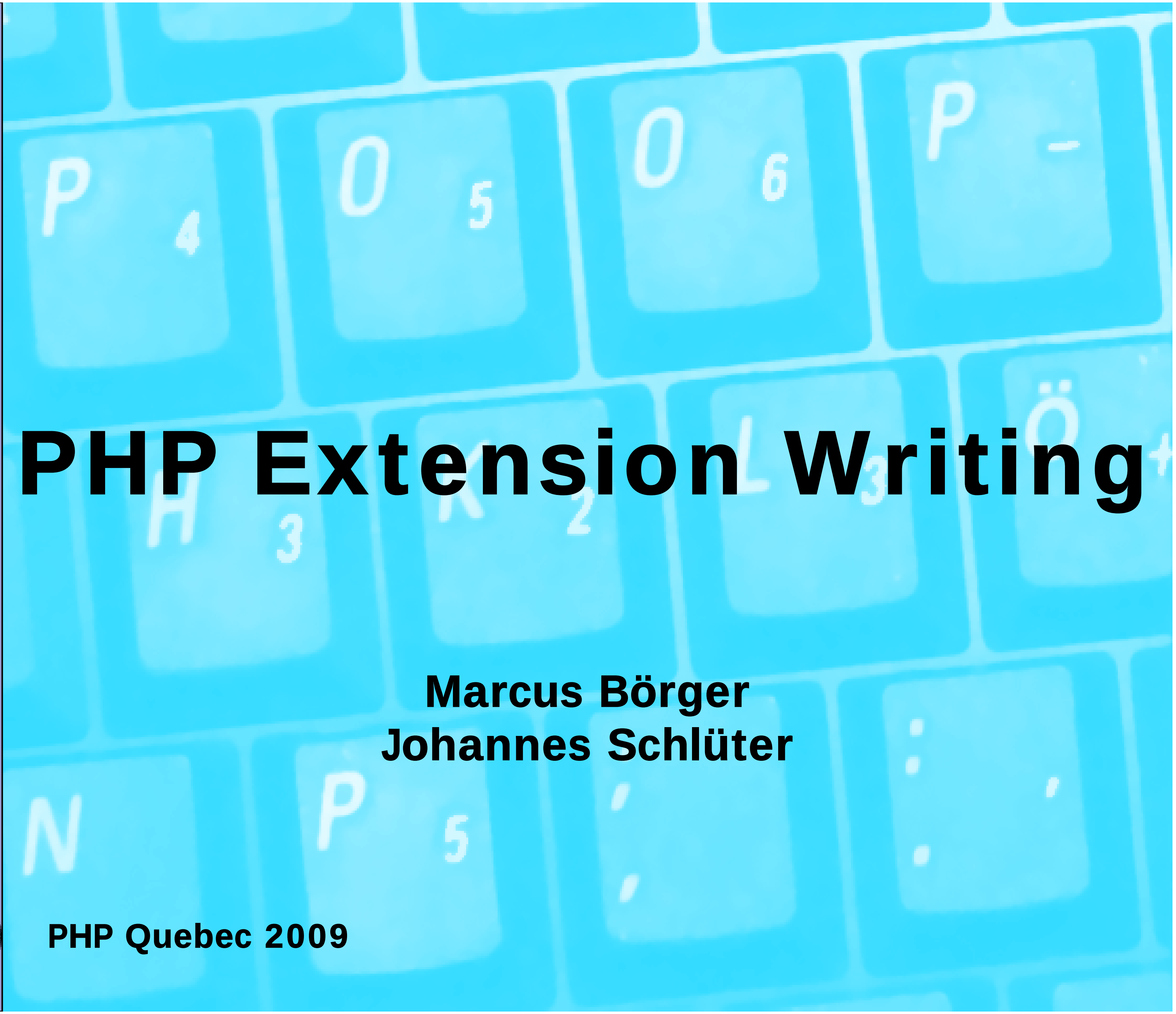


PHP Extension Writing

Marcus Börger
Johannes Schlüter

PHP Quebec 2009

- þ Creating PHP 5 Extension
- þ PHP Lifecycle
- þ Adding objects
- þ Adding iterators to objects

How the slides work

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

```
Text in yellow  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)
```

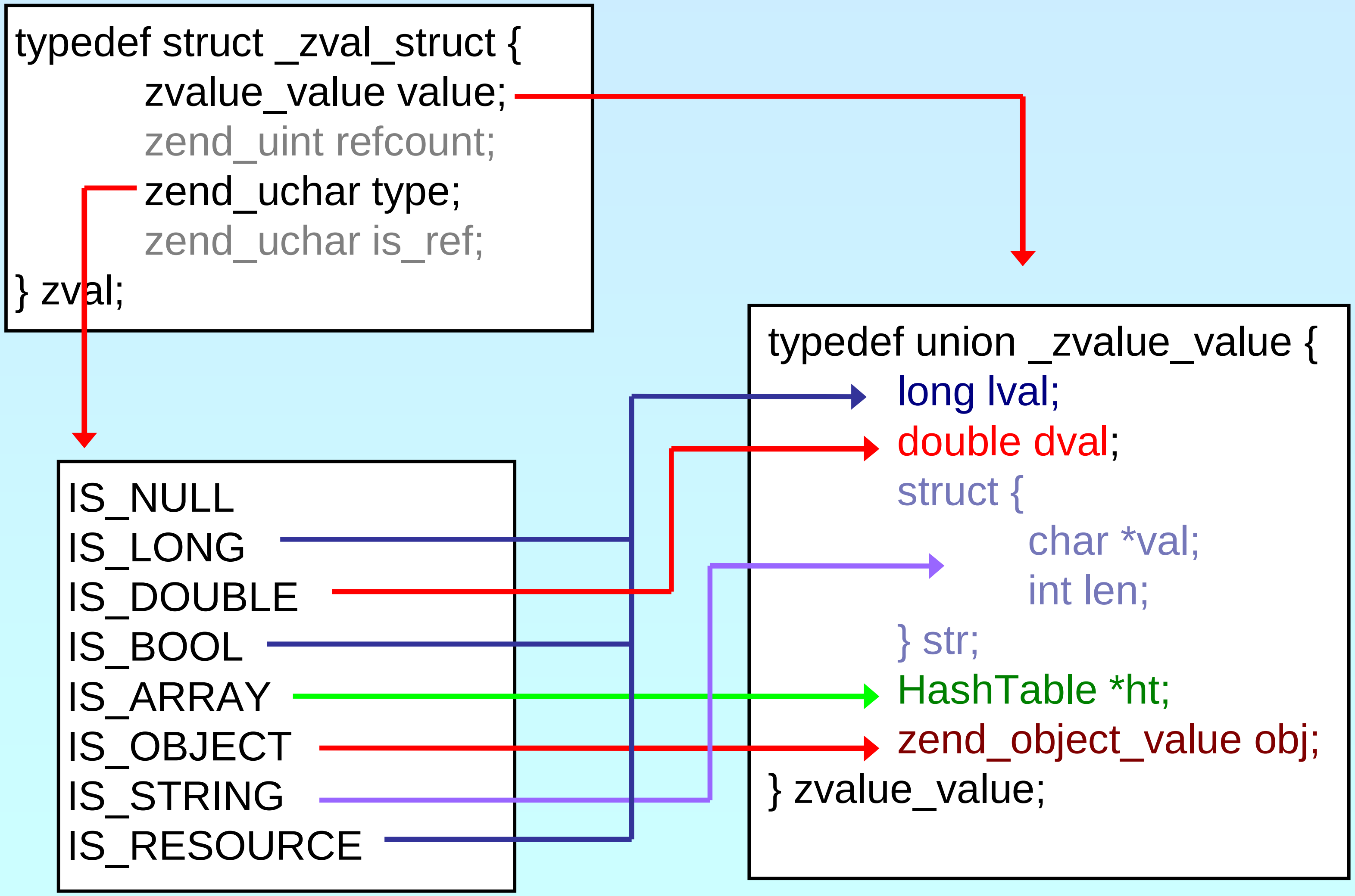
Some special explanation
use red text boxes

Part I

Creating PHP 5 Extensions

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

In PHP all values are zval's

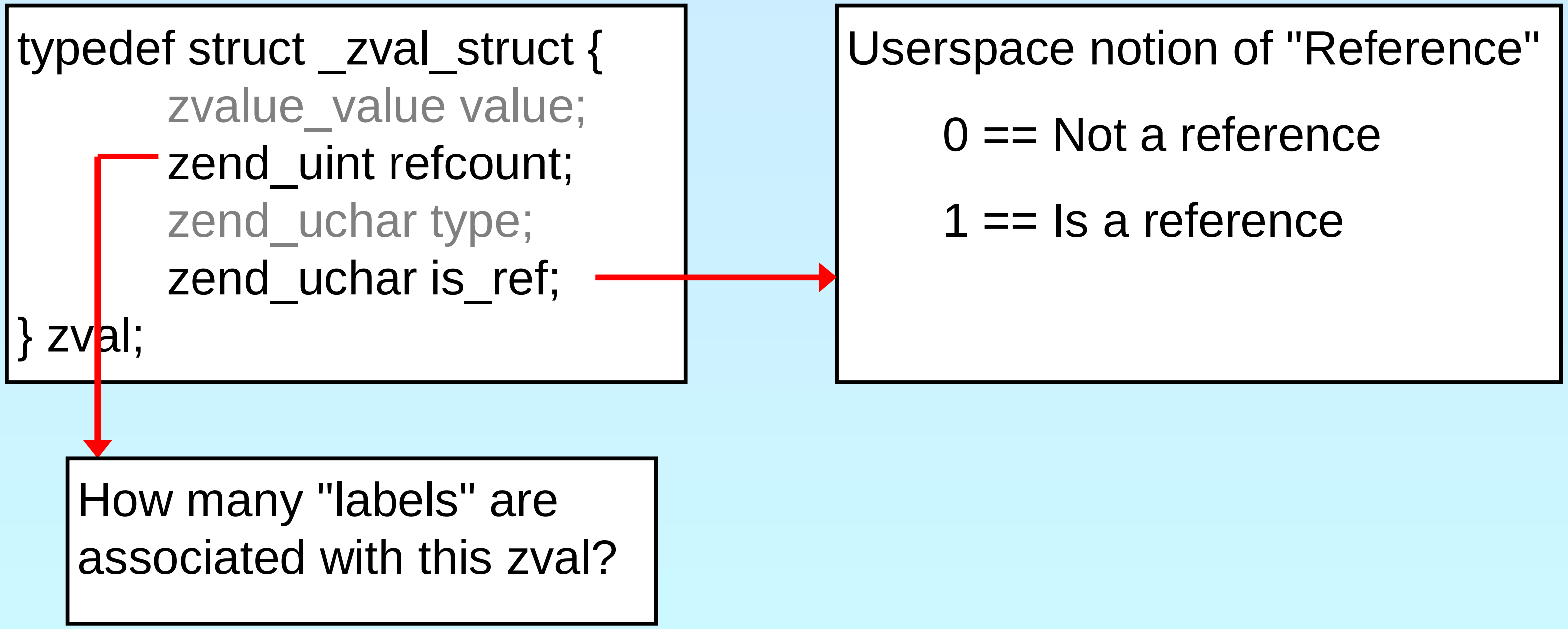


```
typedef struct _zval_struct {
        zvalue_value value;
        zend_uint refcount;
        zend_uchar type;
        zend_uchar is_ref;
} zval;
```

```
IS_NULL
IS_LONG
IS_DOUBLE
IS_BOOL
IS_ARRAY
IS_OBJECT
IS_STRING
IS_RESOURCE
```

```
typedef union _zvalue_value {
        long lval;
        double dval;
        struct {
                char *val;
                int len;
        } str;
        HashTable *ht;
        zend_object_value obj;
} zvalue_value;
```

In PHP all values are zval's



```
typedef struct _zval_struct {
        zvalue_value value;
        zend_uint refcount;
        zend_uchar type;
        zend_uchar is_ref;
} zval;
```

Userspace notion of "Reference"

0 == Not a reference

1 == Is a reference

How many "labels" are associated with this zval?

Copy On Write

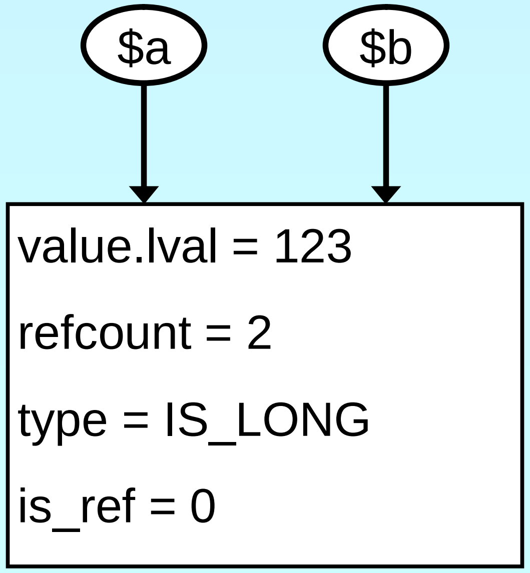
```
typedef struct _zval_struct {
        zvalue_value value;
        zend_uint refcount;
        zend_uchar type;
        zend_uchar is_ref;
} zval;
```

• Has a value of 0 (zero)

• zval shared by 1 or more labels

• If one label wants to make a change, it must leave other labels with the original value.

```
$a = 123;

$b = $a;


$b = 456;
```

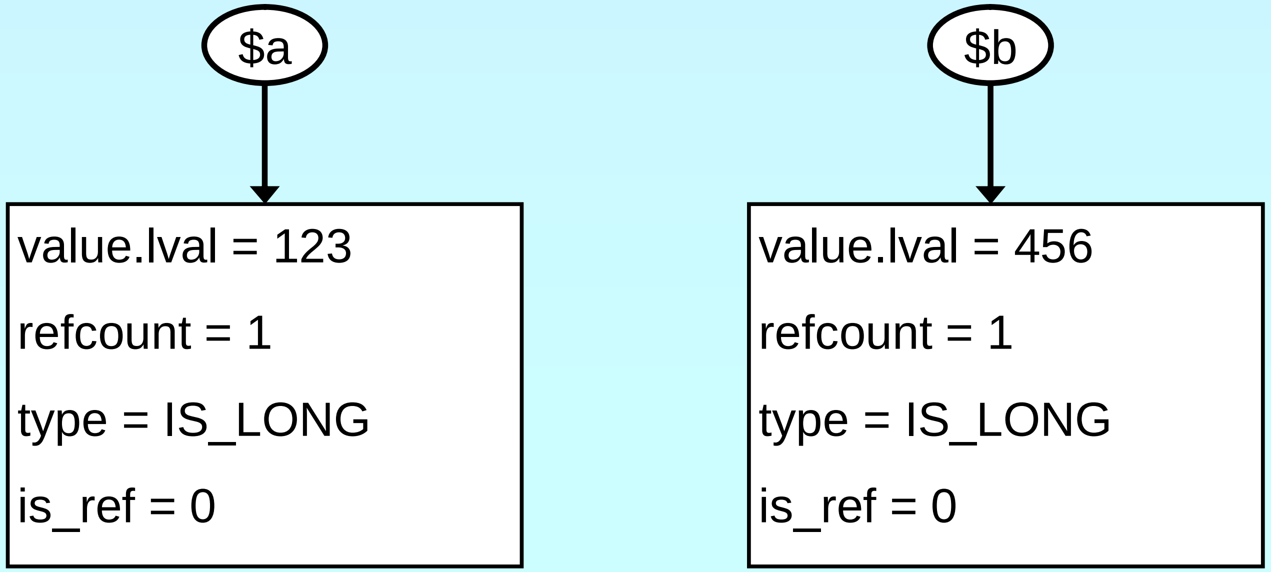


Copy On Write

```
typedef struct _zval_struct {
        zvalue_value value;
        zend_uint refcount;
        zend_uchar type;
        zend_uchar is_ref;
} zval;
```

- Has a value of 0 (zero)
- zval shared by 1 or more labels
- If one label wants to make a change, it must leave other labels with the original value.

```
$a = 123;

$b = $a;


$b = 456;
```

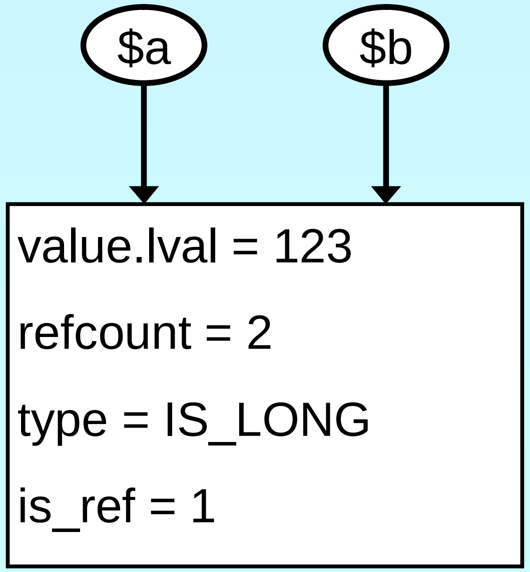


```
$a = 123;

$b = &$a;


$b = 456;
```

Full Reference

```
typedef struct _zval_struct {
        zvalue_value value;
        zend_uint refcount;
        zend_uchar type;
        zend_uchar is_ref;
} zval;
```

- Has a value of 1 (one)
- zval shared by 1 or more labels
- If one label wants to make a change, it does so, causing other labels to see the new value.



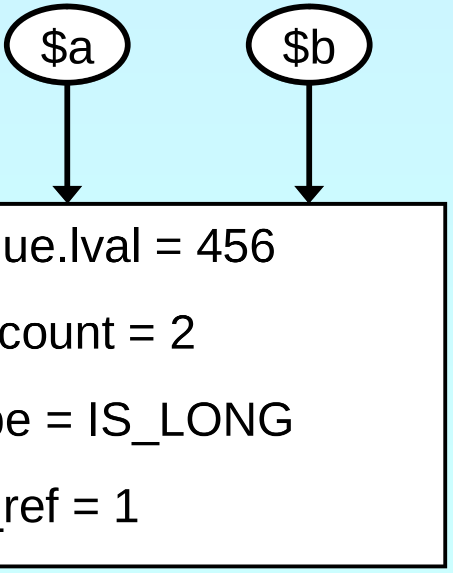
```
value.lval = 123
refcount = 2
type = IS_LONG
is_ref = 1
```

Full Reference

```
typedef struct _zval_struct {
        zvalue_value value;
        zend_uint refcount;
        zend_uchar type;
        zend_uchar is_ref;
} zval;
```

- Has a value of 1 (one)
- zval shared by 1 or more labels
- If one label wants to make a change, it does so, causing other labels to see the new value.

```
$a = 123;

$b = &$a;


$b = 456;
```



value.lval = 456

refcount = 2

type = IS_LONG

is_ref = 1

Creating PHP 5 Extensions

- Most PHP 4 exts will build in PHP5 w/o Changes
- ext_skel can be used to generate a 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.  $ ./sapi/cli/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)

Files in your extension

- You need at least two code files

þ php_*yourext*.h	The header needed by php
þ php_*yourext*.c	The main extension code ('php_' prefix for .c is not necessary)

- You need two configuration files

þ config.m4	Used under *nix
þ config.w32	Used under windows

- Optional files

þ .cvsignore	List of files to be ignored by CVS
þ CREDITS	First line ext name 2nd line all authors
þ EXPERIMENTAL	If available the API is not yet stable
þ package2.xml	Required for PECL extensions
þ README	Probably good to provide some lines

config.m4

- þ PHP Dev is picky about coding style
  - þ Read CODING_STANDARDS in php-src
  - þ Watch your whitespace
  - þ Align your PHP_ARG_ENABLE output
- þ Make your extension default disabled
  - þ 'phpize' or 'pear install' will enable it automatically

```
dnl $Id: $
dnl config.m4 for extension YOUREXT
PHP_ARG_ENABLE(yourext, enable 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

þ **You can prevent the ext from becoming shared**

```
dnl $Id: $
dnl config.m4 for extension YOUREXT
PHP_ARG_ENABLE(yourext, enable 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.w32

þ **Windows configuration uses JScript**

```
// $Id: $
// vim:ft=javascript
ARG_ENABLE("yourext", "YourExt support", "yes");

if (PHP_YOUREXT == "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");
}
```

Extension .h file

þ **Declares data for static linking and symbol exports**

```
/* License, Author, CVS-Tag, Etc... */

#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

/* Only needed if you'll be exporting symbols */
#ifdef PHP_WIN32
# define YOUREXT_API __declspec(dllexport)
#else
# define YOUREXT_API
#endif

/* Place for globals definition */
#endif /* PHP_YOUREXT_H */
```

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

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

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

#include "php.h"
#include "php_ini.h"
#include "ext/standard/info.h"
#include "ext/standard/php_string.h"
#include "php_yourext.h"
```

Structures and defines not in header

þ What ever you want
  - þ Local storage structures?
  - þ Constants?
  - þ Macros?

```
typedef struct _php_yourext_data {
    int type;

    char *name;
    int name_len;

    php_stream *stream;
} php_yourext_data;

#define PHP_YOUREXT_MEANING         42
#define PHP_YOUREXT_COLOR           "purple"

#define PHP_YOUREXT_STRLEN(v)       (v ? strlen(v) : 0)
```

Helper Functions

þ Use TSRMLS_xx as last function parameter
  - When dealing with PHP Data
  - Use --enable-maintainer-zts when building PHP

þ Use static or inline
  - If you need the funtion only in your .c file

þ Use PHPAPI / *YOREXT*_API
  - If you plan to use the functions in other extensions

Helper Functions

þ **Use TSRMLS_xx as last function parameter**

When dealing with PHP Data

TSRMLS_D	**in declarations as only param**
TSRMLS_C	**in uses (calls) as only param**

```
static void my_helper(TSRMLS_D);

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

Helper Functions

þ **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 uses (calls) as only param
TSRMLS_CC	in uses 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

þ **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(char *p, int p_len TSRMLS_DC);

static void some_function(char *p) {
      int p_len;
      TSRMLS_FETCH();

      p_len = strlen(p);
      my_helper(p, p_len TSRMLS_CC);
}
```

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
}; /* }}} */

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

or NULL

Function List

- Exports your functions to userspace
  - Must be terminated by NULL tripplet

```
zend_function_entry yourext_functions[] = { /* {{{ */
    PHP_FE(yourext_func1,        yourext_args_func1)
    PHP_FE(yourext_func2,                          NULL)
    PHP_FALIAS(yourext_func3,    yourext_func2, NULL)
    PHP_NAMED_FE(yourext_func4, _yourext_func4_impl,
                                                   NULL)
    {NULL, NULL, NULL}
};
```

ArgInfo / Signatures

- The function table allows specifing the signature
  - ZEND_BEGIN_ARG_INFO_EX:
    name, pass_rest_by_ref, return_ref, required_args
  - ZEND_ARG_INFO:
    pass_by_ref, name
  - ZEND_ARG_PASS_INFO:
    pass_by_ref
  - ZEND_ARG_ARRAY_INFO:
    pass_by_ref, name
  - ZEND_ARG_OBJ_INFO:
    pass_by_ref, name, classname, allow_null

```
static ZEND_BEGIN_ARG_INFO_EX(yourext_args_func1,0,0,2)
    ZEND_ARG_INFO(0, param_name1)
    ZEND_ARG_ARRAY_INFO(1, param_name2)
ZEND_END_ARG_INFO();
```

PHP Functions

- þ Namespace your functions with your ext's name
- þ Documentation is your friend
  - þ Avoid // style C++ comments
  - þ Avoid declarations inline with code

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

    /* Parameter parsing */

    /* Actual code */

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

Outputting Content

- þ Do not send content to stdout
- þ use PHP's output buffering mechanisms
  - þ php_printf() works just like printf()
  - þ PHPWRITE() respects binary safety

```
/* {{{ proto null yourext_hello_world()
  Say Hello */
PHP_FUNCTION(yourext_hello_world)
{
    char *greeting = "Hello World";

    php_printf("%s!\n", greeting);

    PHPWRITE(greeting, strlen(greeting));
    php_printf("!\n");
}
/* }}} */
```

Parsing parameters

þ **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 %)
...         References to the types given in type_spec
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
```

Parsing Parameters

```
/* {{{ proto null yourext_hello(string name)
  Greet by name */
PHP_FUNCTION(yourext_hello)
{
    char *name;
    int name_len;

    if (zend_parse_parameters(ZEND_NUM_ARGS() TSRMLS_CC,
            "s", &name, &name_len) == FAILURE) {
        return;
    }

    php_printf("Hello %s!\n", name);
}
/* }}} */
```

Returning Values

þ **Marking success**

```
/* {{{ proto bool yourext_hello(string name)
  Greet by name */
PHP_FUNCTION(yourext_hello)
{
    char *name;
    int name_len;

    if (zend_parse_parameters(ZEND_NUM_ARGS() TSRMLS_CC,
            "s", &name, &name_len) == FAILURE) {
        return;
    }

    php_printf("Hello %s!\n", name);

    RETURN_TRUE;
}
/* }}} */
```

Makes the return value NULL

Returning Values

þ **Simple scalars use intuitive RETURN_*() macros**

```
RETURN_NULL();
RETURN_BOOL(b);               b: 0 => FALSE, non-0 => TRUE
RETURN_TRUE;                  RETURN_BOOL(1)
RETURN_FALSE;                 RETURN_BOOL(0)
RETURN_LONG(l);               l: Integer value
RETURN_DOUBLE(d);             d: Floating point value
```

Returning Values

- þ Strings are slightly more complex
- þ The string value must "belong" to the engine
  - þ Will not survive the destruction of the zval
  - þ Will be freed using efree()
- þ Pass 0 (zero) for *dup* to give it the string
- þ Pass 1 (one) for *dup* to make a copy (*dup*licate)

```
RETURN_STRING(str, dup)     str: char* string value
                            dup: 0/1 flag, duplicate string?
RETURN_STRINGL(str, len, dup)
                            len: Predetermined string length

RETURN_STRING("Hello World", 1);
RETURN_STRING(estrdup("Hello World"), 0);
RETURN_EMPTY_STRING();
```

Setting Returning Values

þ **RETURN_*() macros automatically exit function**

```
#define RETURN_NULL()       { RETVAL_NULL();     return; }
#define RETURN_TRUE         { RETVAL_TRUE;       return; }
#define RETURN_FALSE        { RETVAL_FALSE;      return; }
#define RETURN_BOOL(b)      { RETVAL_BOOL(b);    return; }
#define RETURN_LONG(l)      { RETVAL_LONG(l);    return; }
#define RETURN_DOUBLE(d)    { RETVAL_DOUBLE(d);  return; }

#define RETURN_STRING(str, dup)            \
        { RETVAL_STRING(str, dup);           return; }
#define RETURN_STRINGL(str, len, dup)      \
        { RETVAL_STRINGL(str, len, dup);     return; }
#define RETURN_EMPTY_STRING()              \
        { RETVAL_EMPTY_STRING();             return; }
```

Setting Returning Values

- RETURN_*() macros automatically exit function
- RETVAL_*() family work the same without exiting

```
#define RETVAL_NULL()         ZVAL_NULL(return_value)
#define RETVAL_TRUE           ZVAL_TRUE(return_value)
#define RETVAL_FALSE          ZVAL_FALSE(return_value)
#define RETVAL_BOOL(b)        ZVAL_BOOL(return_value, b)
#define RETVAL_LONG(l)        ZVAL_LONG(return_value, l)
#define RETVAL_DOUBLE(d)      ZVAL_DOUBLE(return_value, d)

#define RETVAL_STRING(str, dup)            \
        ZVAL_STRING(return_value, str, dup)
#define RETVAL_STRINGL(str, len, dup)      \
        ZVAL_STRINGL(return_value, str, len, dup)
#define RETVAL_EMPTY_STRING()              \
        ZVAL_EMPTY_STRING(return_value)
```

Setting Returning Values

- RETURN_*() macros automatically exit function
- RETVAL_*() family work the same without exiting
- ZVAL_*() family work on specific zval (later)

```
#define RETVAL_NULL()        ZVAL_NULL(return_value)
#define RETVAL_TRUE          ZVAL_TRUE(return_value)
#define RETVAL_FALSE         ZVAL_FALSE(return_value)
#define RETVAL_BOOL(b)       ZVAL_BOOL(return_value, b)
#define RETVAL_LONG(l)       ZVAL_LONG(return_value, l)
#define RETVAL_DOUBLE(d)     ZVAL_DOUBLE(return_value, d)

#define RETVAL_STRING(str, dup)            \
        ZVAL_STRING(return_value, str, dup)
#define RETVAL_STRINGL(str, len, dup)      \
        ZVAL_STRINGL(return_value, str, len, dup)
#define RETVAL_EMPTY_STRING()              \
        ZVAL_EMPTY_STRING(return_value)
```

Example 1

þ **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);
}
/* }}} */
```

Example 2

þ **Incrementing a value with an optional maximum**

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

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

    n = (n+1) % nmax;

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

Initialize optional values

Use brackets for optional values

A vertical bar separates optional and required parameters

Example 3

þ **Returning some generated string**

```
#define YOUREXT_VERSION_MAJOR    0
#define YOUREXT_VERSION_MINOR    1

/* {{{ proto string 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

No need to
copy the string

Dealing with arrays

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

```
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 dup);
int add_assoc_stringl(zval *arg, char *key, char *str,
                          uint len, int dup);
int add_assoc_zval(zval *arg, char *key, zval *value);
```

Dealing with arrays

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

þ **Returning an array**

```
/* {{{ proto array 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

þ **Multiple values stored in key/value pairs**

þ **Arrays are special HashTables (Symbol tables)**

- þ **Numeric keys get converted to strings**
- þ **All values are zval* pointers.**

```
/* arKey hashed using DJBX33A */
ulong zend_get_hash_value(char *arKey, uint nKeyLength);

/* count($ht) */
int zend_hash_num_elements(HashTable *ht);

/* Removes all elements from the HashTable */
int zend_hash_clean(HashTable *ht);
```

Adding to HashTables

- add_assoc/index_*() functions wrap zend_symtable_update()
- Symbol table keys include terminating NULL byte
  sizeof(key) vs. strlen(key)

```
add_assoc_zval(arr, "foo", val);
add_assoc_zval_ex(arr, "foo", sizeof("foo"), val);

zend_symtable_update(Z_ARRVAL_P(arr),
                    "foo", sizeof("foo"),
                    &val, sizeof(zval*), NULL);
```

Deleting from HashTables

- 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);
```

Searching HashTables

- **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);
```

Searching HashTables

- 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, void **pData);
```

Searching HashTables

- **Symbol Tables store zval* pointers**
- **When fetching, a reference to a zval** is passed**

```
zval **tmp;

if (zend_symtable_find(ht, "key", sizeof("key"),
                                  (void**)&tmp) == SUCCESS) {

    /* Do something with tmp */
    if (Z_TYPE_PP(tmp) == IS_STRING) {
        PHPWRITE(Z_STRVAL_PP(tmp), Z_STRLEN_PP(tmp));
    }
}
```

Accessing a zval

Z_LVAL(zval)	long	value
Z_BVAL(zval)	zend_bool	value
Z_DVAL(zval)	double	value
Z_STRVAL(zval)	char*	value
Z_STRLEN(zval)	int	length
Z_ARRVAL(zval)	HashTable*	only array
Z_OBJ_HANDLE(zval)	int	obj id
Z_OBJ_HT(zval)	zend_object_handlers*	obj handlers
Z_OBJCE(zval)	zend_class_entry*	obj class
Z_OBJPROP(zval)	HashTable*	properties
Z_OBJ_HANDLER(zval,hf)	Z_OBJ_HT((zval))->hf	obj handler
Z_RESVAL(zval)	int	resource id
Z_TYPE(zval)	int	IS_*
HASH_OF(zval)	HashTable*	array+props
Z_*_P(zp)	Z_*(*zp)	
Z_*_PP(zpp)	Z_*(**zpp)	

Reference count and is-ref

```
Z_REFCOUNT(zval)         Retrieve reference count
Z_SET_REFCOUNT(zval,rc)  Set reference count to <rc>
Z_ADDREF(zval)           Increment reference count
Z_DELREF(zval)           Decrement reference count

Z_ISREF(zval)            Whether zval is a reference
Z_SET_ISREF(zval)        Makes zval a reference variable
Z_UNSET_ISREF(zval)      Resets the is-reference flag
Z_SET_ISREF_TO(zval,is)  Make zval a reference is <is> != 0

Z_*_P(zp)                Z_*(*zp)
Z_*_PP(zpp)              Z_*(**zpp)
```

Setting types and values

```
ZVAL_NULL(zp)              IS_NULL       Just set the type
ZVAL_RESOURCE(zp,l)        IS_RESOURCE   Set to resource <l>
ZVAL_BOOL(zp,b)            IS_BOOL       Set to boolean <b>
ZVAL_FALSE(zp)             IS_BOOL       Set to false
ZVAL_TRUE(zp)              IS_BOOL       Set to true
ZVAL_LONG(zp,l)            IS_LONG       Set to long <l>
ZVAL_DOUBLE(zp,d)          IS_DOUBLE     Set to double <d>
ZVAL_STRING(zp,s,dup)      IS_STRING     Set string
ZVAL_STRINGL(zp,s,l,dup)   IS_STRING     Set string and length
ZVAL_EMPTY_STRING(zp)      IS_STRING     Set as empty string

ZVAL_ZVAL(zp,zv,copy,dtor)
     Copy the zval and its type.
     Allows to call copying, necessary for strings etc.
     Allows to destruct (delref) the original zval.
```

Allocate and Initialize a zval

```
ALLOC_ZVAL(zp)              Allocate a zval using emalloc()

INIT_PZVAL(zp)              Set reference count and isref 0
INIT_ZVAL(zval)             Initialize and set NULL, no pointer

ALLOC_INIT_ZVAL(zp)         Allocate and initialize a zval
MAKE_STD_ZVAL(zp)           Allocate, initialize and set NULL
```

```
Example:

zval *val;
ALLOC_INIT_ZVAL(val);
ZVAL_STRINGL(val, “Myval”, sizeof(“myval”)-1, 1)
```

Dealing with a HashTable

þ **Hash tables have builtin "foreach" functions**

```
/* array_walk($ht, $apply_func) */
void zend_hash_apply(HashTable *ht,
        apply_func_t apply_func TSRMLS_DC);

/* array_walk($ht, $apply_func, $data) */
void zend_hash_apply_with_argument(HashTable *ht,
        apply_func_arg_t apply_func, void * TSRMLS_DC);

/* Multiple argument version,
 * This is also the only variant which provides
 * the key to the callback */
void zend_hash_apply_with_arguments(HashTable *ht,
        apply_func_args_t apply_func, int, ...);
```

Dealing with a HashTable

- Hash tables have builtin "foreach" functions
- Each function requires a different type of callback

```
/* pDest contains a pointer to
 * what's stored in the HashTable
 * Since there is a zval* in SymbolTables
 * we wind up with a zval** being passed as pDest*
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);
```

Dealing with a HashTable

- þ **Hash tables have builtin "foreach" functions**
- þ **Each function requires a different type of callback**
- þ **Callbacks return one of three status values**
  - þ **Prior to 5.2.1 all non zero return values result in deletion**

```
/* Continue itterating the HashTable */
#define ZEND_HASH_APPLY_KEEP                0

/* Remove this element, but continue processing */
#define ZEND_HASH_APPLY_REMOVE              1<<0

/* Terminate the loop (break;) */
#define ZEND_HASH_APPLY_STOP                1<<1
```

Example 5 a

þ **Using zend_hash_apply_with_arguments()**

```
/* {{{ proto void yourext_foreach( array names,
                                   string greeting)
Say hello to each person */
PHP_FUNCTION(yourext_foreach)
{
   zval *names;
   char *greet;
   int greet_len;

   if (zend_parse_parameters(ZEND_NUM_ARGS() TSRMLS_CC,
         "as", &names, &greet, &greet_len) == FAILURE) {
         return;
   }

   zend_hash_apply_with_argument(Z_ARRVAL_P(names),
         (apply_func_arg_t)yourext_foreach, greet TSRMLS_CC);
} /* }}} */
```

Example 5 b

þ **Calling a function for each element**

```
/* {{{ yourext_foreach
  Callback for outputting a greeting
  for each name in a user-provided array */
int yourext_foreach(zval **param, char *greeting TSRMLS_DC)
{
    if (Z_TYPE_PP(param) == IS_STRING) {
        php_printf("%s %s\n", greeting, Z_STRVAL_PP(param));

        return ZEND_HASH_APPLY_KEEP;
    } else {
        php_error_docref(NULL TSRMLS_CC, E_WARNING,
            "Non-string value passed in $names array");

        return ZEND_HASH_APPLY_STOP;
    }
} /* }}} */
```

Part II

PHP Lifecycle

STARTUP

- Initial startup of a PHP process space
- Initialize engine and core components
- Parse php.ini
- Initialize (MINIT) staticly built modules
- Initialize (MINIT) shared modules
  (loaded by php.ini)
- Finalize Initialization

ACTIVATION

- þ Triggered upon receiving a new request (page hit)
- þ Initialize environment and variables (symbol_table, EGPCS)
- þ Activate (RINIT) static built modules
- þ Activate (RINIT) shared modules

RUNTIME

- þ Actual execution of scripts happens here.
- þ Compile and execute auto_prepend_file.
- þ Compile and execute main_file.
- þ Compile and execute auto_append_file.

DEACTIVATION

þ Upon exit(), die(), E_ERROR,
or end of last script execution.

þ Call user-defined shutdown functions.

þ Destroy object instances.

þ Flush output.

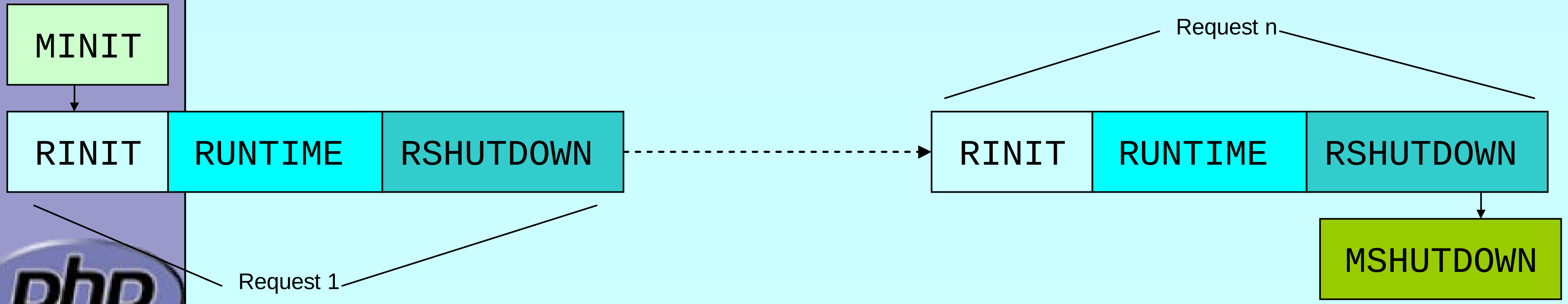
þ Deactivate (RSHUTDOWN) modules
(in reverse of activation order)

þ Clean up environment

þ Implicitly free remaining non-persistent memory.

SHUTDOWN

- Final good-night. Called as process space is terminating (apache child termination).
- Shutdown (MSHUTDOWN) all modules
  (rev. startup order)
- Shutdown the engine



Memory Allocation

þ **Traditionall malloc() family may be used**

```
void *  malloc(size_t size);
void *  calloc(size_t nmemb, size_t size);
void *  realloc(void *ptr, size_t size);
void *  strdup(char *str);
void *  strndup(char *str, size_t len);
void   free(void *ptr);
```

Memory Allocation

- þ Traditionall malloc() family may be used
- þ Non-persistent allocators prefixed with *e*
  - þ Additional helpers provided by engine
  - þ Automatically freed by engine during DEACTIVATION

```
void * emalloc(size_t size);
void * ecalloc(size_t nmemb, size_t size);
void * erealloc(void *ptr, size_t size);
void * estrdup(char *str);
void * estrndup(char *str, size_t len);
void  efree(void *ptr);

void *safe_emalloc(size_t nmemb, size_t size,
                     size_t adtl);
void *STR_EMPTY_ALLOC(void);
```

Memory Allocation

- þ Traditionall malloc() family may be used
- þ Non-persistent allocators prefixed with *e*
- þ Selective allocators prefixed with *pe*
  - þ pestrndup() not available
  - þ safe_pemalloc() requires PHP >= 5.1

```
void *pemalloc(size_t size, int persist);
void *pecalloc(size_t nmemb, size_t size, int persist);
void *perealloc(void *ptr, size_t size, int persist);
void *pestrdup(char *str, int persist);

void pefree(void *ptr, int persist);

void *safe_pemalloc(size_t nmemb, size_t size,
                     size_t addtl, int persist);
```

Storing Global Values

- Do **NOT** store transient data in the global scope!
  - Threaded SAPIs **will** break

```
static char *errormsg = NULL;

PHP_FUNCTION(yourext_unthreadsafe) {
    long ret;

    ret = do_something("value", &errormsg);
    if (errormsg) {
        php_error_docref(NULL TSRMLS_CC, E_WARNING,
            "do_something() failed with: %s", errormsg);
        free(errormsg);
        errormsg = NULL;
    }
}
```

Global struct in .h

þ **Provide a structure and access macros**

```
ZEND_BEGIN_MODULE_GLOBALS(yourext)
    char        *str;
    int          strlen;
    long         counter;
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 ctor/dtor functions
  - Initializer called once at (thread) startup
  - Destructor called once at (thread) shutdown
  - Allocations made here must be persistent (malloc’d)

```
ZEND_DECLARE_MODULE_GLOBALS(yourext)

static void yourext_globals_ctor(
     zend_yourext_globals *globals) {
     /* Initialize your global struct */
     globals->str     = NULL;
     globals->strlen  = 0;
     globals->counter = 0;
}

static void yourext_globals_dtor(
     zend_yourext_globals *globals) {
     /* Clean up any allocated globals */
}
```

MINIT/MSHUTDOWN

- **Allocate local storage for globals in ZTS mode**
- **Call globals initialization and destruction as needed**

```
PHP_MINIT_FUNCTION(yourext) {
     ZEND_INIT_MODULE_GLOBALS(yourext,
        yourext_globals_ctor, yourext_globals_dtor);
     return SUCCESS;
}


PHP_MSHUTDOWN_FUNCTION(yourext) {
#ifndef ZTS
     yourext_globals_dtor(&yourext_globals TSRMLS_CC);
#endif
     return SUCCESS;
}
```

RINIT/RSHUTDOWN

- **Initialize request specific settings at RINIT**
- **Clean up their values at RSHUTDOWN**

```
PHP_RINIT_FUNCTION(yourext) {
     /* Track number of times this thread/process
      * has serviced requests */
     YOUREXT_G(counter)++;
     return SUCCESS;
}

PHP_RSHUTDOWN_FUNCTION(yourext) {
     if (YOUREXT_G(str)) {
         efree(YOUREXT_G(str));
         YOUREXT_G(str) = NULL;
     }
     return SUCCESS;
}
```

Globals Access

- Access global values using *YOUREXT*_G(v) macro

```
PHP_FUNCTION(yourext_set_string) {
     char *str;
     int str_len;
     if (zend_parse_parameters(ZEND_NUM_ARGS(), "s",
                             &str, &str_len) == FAILURE) {
        return;
     }
     if (YOUREXT_G(str)) {
        efree(YOUREXT_G(str));
     }
     YOUREXT_G(str) = estrndup(str, str_len);
     YOUREXT_G(strlen) = str_len;
     RETURN_TRUE;
}
```

Globals Access

þ **Access global values using *YOUREXT*_G(v) macro**

```
PHP_FUNCTION(yourext_get_string) {
   if (YOUREXT_G(str)) {
      RETURN_STRINGL(YOUREXT_G(str), YOUREXT_G(strlen), 1);
   } else {
      RETURN_EMPTY_STRING();
   }
}
```

Registering consts

- Register constants during MINIT (usually)
  - name_len here is sizeof()
  - Thus name must be a real string
    Do not use string variables!

```
int zend_get_constant(char *name, uint name_len,
     zval *result TSRMLS_DC);

REGISTER_LONG_CONSTANT(name, lval, flags)
REGISTER_DOUBLE_CONSTANT(name, dval, flags)
REGISTER_STRING_CONSTANT(name, str, flags)
REGISTER_STRINGL_CONSTANT(name, str, len, flags)

int zend_register_constant(zend_constant *c TSRMLS_DC);

/* Case-sensitive */
#define CONST_CS                (1<<0)
/* Persistent */
#define CONST_PERSISTENT        (1<<1)
```

Registering consts

- Persistent constants require CONST_PERSISTENT
- Non-persistent string constants must be estrdup'd

```
PHP_MINIT_FUNCTION(yourext) {
	REGISTER_LONG_CONSTANT("YOUREXT_CONSTNAME", 42,
	                       CONST_CS | CONST_PERSISTENT);
	REGISTER_STRING_CONSTANT("YOUREXT_VERSION", "$ID: $",
	                       CONST_CS | CONST_PERSISTENT);
	return SUCCESS;
}


PHP_RINIT_FUNCTION(yourext) {
	REGISTER_LONG_CONSTANT("YOUREXT_COUNTER",
	                       YOUREXT_G(counter), CONST_CS);
	return SUCCESS;
}
```

MINFO

þ **Provide some information about your extension**
- þ **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(str));

   php_info_print_table_end();
}
```

What else ?

- þ INI Handling
- þ Dealing with resources and streams
- þ Object support

Part III

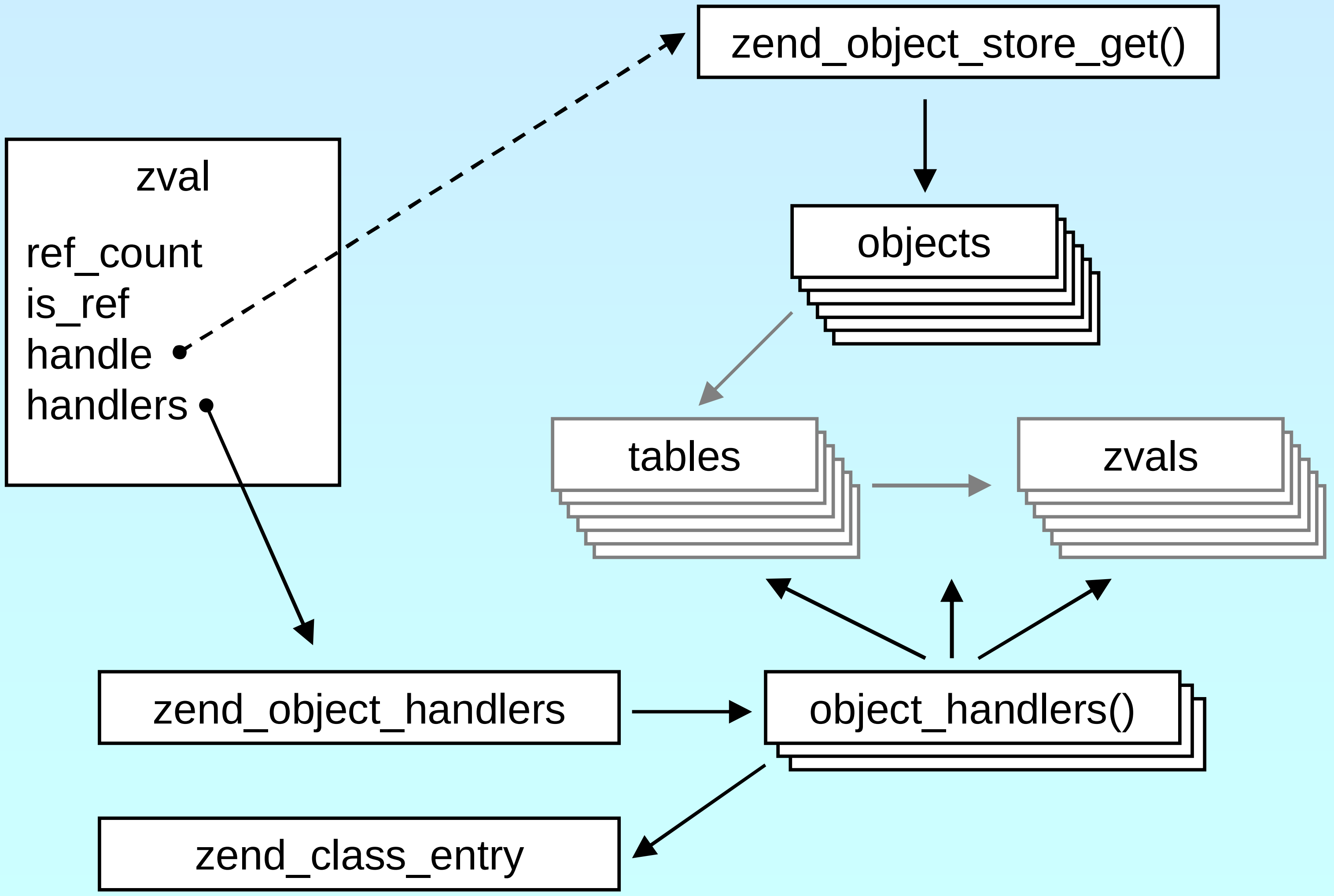
Adding objects

- þ How to create your own classes
- þ How to create interfaces
- þ How to create methods
- þ What can be overloaded

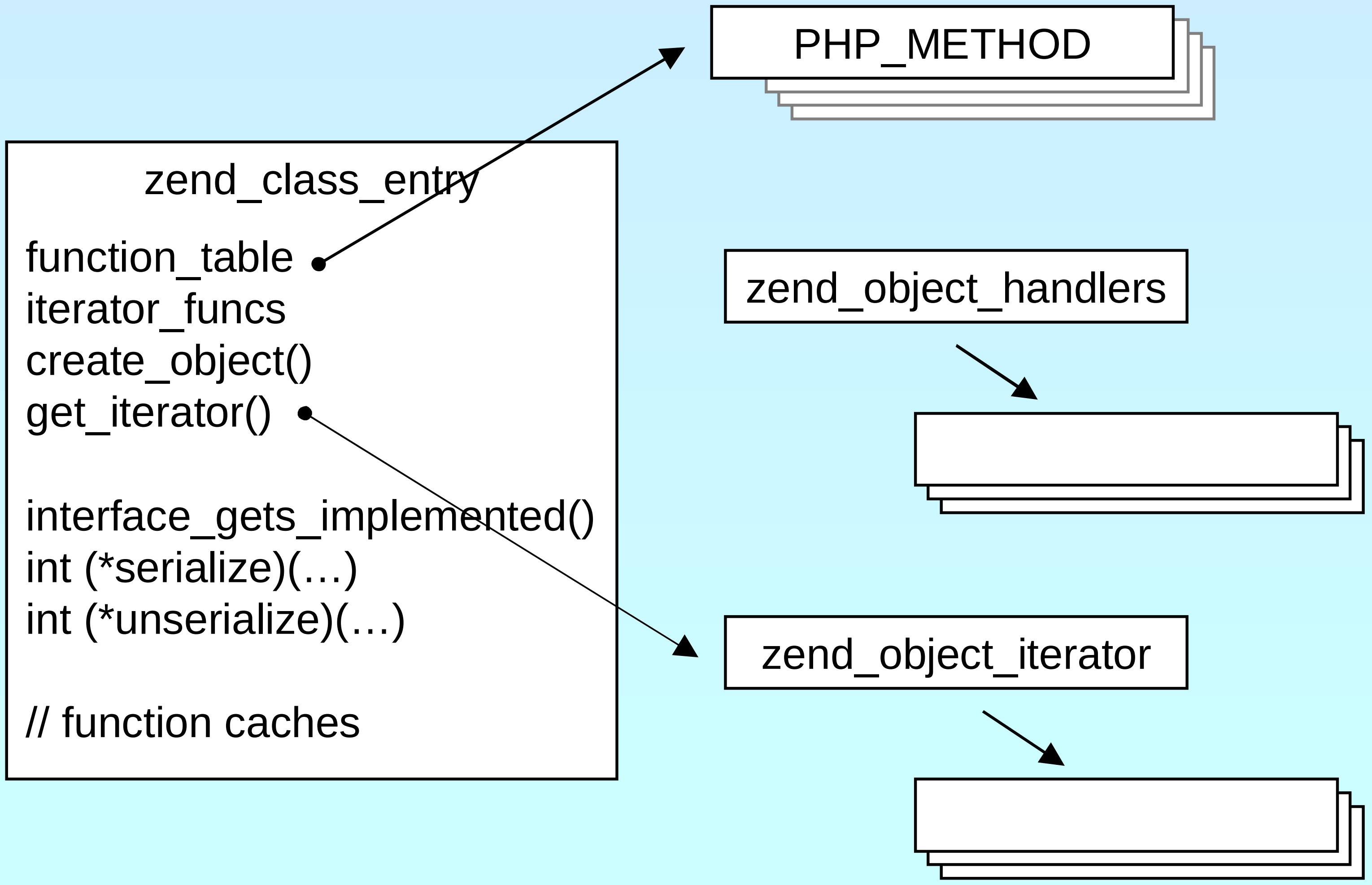
What is needed?

- þ Providing methods
- þ Providing a zend_class_entry pointer
- þ Providing object handlers
- þ Registering the class

General class layout



General class layout



Registering

- þ Obviously you have to register your class
  - þ A temporary zend_class_entry is necessary first
  - þ After basic registering you have a dedicated pointer
  - þ Now you have to specify the c-level constructor function
  - þ Provide your own handler funcs or copy and modify defaults
  - þ Finally implement interfaces, set class flags, specify iterator

```
zend_class_entry *util_ce_dir;
PHP_MINIT_FUNCTION(util) /* {{{ */
{
     zend_class_entry ce;
     INIT_CLASS_ENTRY(ce, "dirs", util_dir_class_functions);
     util_ce_dir = zend_register_internal_class(&ce TSRMLS_CC);
     util_ce_dir->create_object = util_dir_object_new;
     memcpy(&util_dir_handlers, zend_get_std_object_handlers(),
          sizeof(zend_object_handlers));
     util_dir_handlers.clone_obj = util_dir_object_clone;
     zend_class_implements(util_ce_dir TSRMLS_CC, 1, zend_ce_iterator);
     util_ce_dir->ce_flags |= ZEND_ACC_FINAL_CLASS;
     util_ce_dir->get_iterator = util_dir_get_iterator;
     return SUCCESS;
} /* }}} */
```

Declaring class constants

- þ You can register class constants
  - þ Use target zend_class_entry pointer
  - þ Use sizeof() not strlen() for const name

```
int zend_declare_class_constant(zend_class_entry *ce,
     char *name, size_t name_len, zval *value TSRMLS_DC);

int zend_declare_class_constant_long(zend_class_entry *ce,
     char *name, size_t name_len, long value TSRMLS_DC);

int zend_declare_class_constant_bool(zend_class_entry *ce,
     char *name, size_t name_len, zend_bool value TSRMLS_DC);

int zend_declare_class_constant_double(zend_class_entry *ce,
     char *name, size_t name_len, double value TSRMLS_DC);

int zend_declare_class_constant_stringl(zend_class_entry *ce,
     char *name, size_t name_len, char *val, size_t val_len TSRMLS_DC);

int zend_declare_class_constant_string(zend_class_entry *ce,
     char *name, size_t name_len, char *value TSRMLS_DC);
```

Declaring methods

```
/* declare method parameters,  */
static ZEND_BEGIN_ARG_INFO(arginfo_dir___construct, 0)
     ZEND_ARG_INFO(0, path)  /* parameter name */
ZEND_END_ARG_INFO();

/* each method can have its own parameters and visibility */
static zend_function_entry util_dir_class_functions[] = {
     PHP_ME(dir, __construct, arginfo_dir___construct,
                                    ZEND_ACC_CTOR | ZEND_ACC_PUBLIC)
     PHP_ME(dir, rewind,      NULL, ZEND_ACC_PUBLIC)
     PHP_ME(dir, hasMore,     NULL, ZEND_ACC_PUBLIC)
     PHP_ME(dir, key,         NULL, ZEND_ACC_PUBLIC)
     PHP_ME(dir, current,     NULL, ZEND_ACC_PUBLIC)
     PHP_ME(dir, next,        NULL, ZEND_ACC_PUBLIC)
     PHP_ME(dir, getPath,     NULL, ZEND_ACC_PUBLIC)
     {NULL, NULL, NULL}
};
```

class/object structs

þ It is a good practice to 'inherit' zend_object
  - þ That allows your class to support normal properties
  - þ Thus you do not need to overwrite all handlers

```
/* declare the class handlers */
static zend_object_handlers util_dir_handlers;

/* decalre the class entry */
static zend_class_entry *util_ce_dir;

/* the overloaded class structure */

/* overloading the structure results in the need of having
   dedicated creatin/cloning/destruction functions */
typedef struct _util_dir_object {
        zend_object       std;
        php_stream        *dirp;
        php_stream_dirent entry;
        char              *path;
        int               index;
} util_dir_object;
```

Inherit zend_object by placing it as first member of your object struct

Object creation/cloning

- þ Allcate memory for your struct
  Initialize the whole struct (probably by using ecalloc())
- þ Initialize the base Zend object
- þ Copy default properties
- þ Store the object
- þ Assign the handlers

```
zend_object_value util_dir_object_new(zend_class_entry *ce TSRMLS_DC) {
     zend_object_value retval;
     util_dir_object *intern;

     intern = ecalloc(1, sizeof(util_dir_object));
     zend_object_std_init(&(intern->std), ce TSRMLS_CC);
     zend_hash_copy(intern->std.properties,
          &ce->default_properties, (copy_ctor_func_t) zval_add_ref,
          NULL, sizeof(zval *));

     retval.handle = zend_objects_store_put(intern,
          util_dir_object_dtor, NULL TSRMLS_CC);
     retval.handlers = &util_dir_handlers;
     return retval;
}
```

Object destruction

- Free properties
- Free all resources and free all allocated memory
- Free memory for object itself

```
/* {{{ util_dir_object_dtor */
/* close all resources and the memory allocated for the object */
static void
util_dir_object_dtor(void *object, zend_object_handle handle TSRMLS_DC)
{
	util_dir_object *intern = (util_dir_object *)object;

	zend_object_std_dtor(&(intern->std) TSRMLS_CC);

	if (intern->path) {
		efree(intern->path);
	}
	if (intern->dirp) {
		php_stream_close(intern->dirp);
	}

	efree(object);
} /* }}} */
```

A simple method

- þ Macro getThis() gives you access to $this as zval
- þ The returned zval is used to get your struct

```
/* {{{ proto string dir::key()
   Return current dir entry */
PHP_METHOD(dir, key)
{
     zval *object = getThis();
     util_dir_object *intern = (util_dir_object*)
          zend_object_store_get_object(object TSRMLS_CC);

     if (intern->dirp) {
          RETURN_LONG(intern->index);
     } else {
          RETURN_FALSE;
     }
} /* }}} */
```

The constructor

þ **Remember that your object is already fully initialized**

In this case we chose to either finish initialization in the constructor or throw an exception.

```
/* {{{ proto void dir::__construct(string path)
 Constructs a new dir iterator from a path. */
PHP_METHOD(dir, __construct)
{
     util_dir_object *intern;
     char *path;
     int len;




     if (zend_parse_parameters(ZEND_NUM_ARGS() TSRMLS_CC, "s", &path,
               &len) == SUCCESS) {
          intern = (util_dir_object*)
               zend_object_store_get_object(getThis() TSRMLS_CC);
          util_dir_open(intern, path TSRMLS_CC);
     }

} /* }}} */
```

The constructor

þ Remember that your object is already fully initialized

In this case we chose to either finish initialization in the constructor or throw an exception.

þ Change errors to exceptions to support constructor failure

```
/* {{{ proto void dir::__construct(string path)
 Constructs a new dir iterator from a path. */
PHP_METHOD(dir, __construct)
{
     util_dir_object *intern;
     char *path;
     int len;

     php_set_error_handling(EH_THROW, zend_exception_get_default()
          TSRMLS_CC);

     if (zend_parse_parameters(ZEND_NUM_ARGS() TSRMLS_CC, "s", &path,
               &len) == SUCCESS) {
          intern = (util_dir_object*)
               zend_object_store_get_object(getThis() TSRMLS_CC);
          util_dir_open(intern, path TSRMLS_CC);
     }
     php_set_error_handling(EH_NORMAL, NULL TSRMLS_CC);
} /* }}} */
```

Object casting

```
/* {{{ */
static int zend_std_cast_object_tostring(zval *readobj, zval *writeobj,
     int type TSRMLS_DC)
{
     zval *retval == NULL;
     if (type == IS_STRING) {
        zend_call_method_with_0_params(&readobj, NULL, NULL,
           "__tostring", &retval);
        if (retval) {
           if (Z_TYPE_P(retval) != IS_STRING) {
              zend_error(E_ERROR, "Method %s::__toString() must"
                 " return a string value", Z_OBJCE_P(readobj)->name);
           }
        } else {
           MAKE_STD_ZVAL(retval);
           ZVAL_EMPTY_STRING(retval);
        }
        ZVAL_ZVAL(writeobj, retval, 1, 1);
        INIT_PZVAL(writeobj);
     }
     return retval ? SUCCESS : FAILURE;
} /* }}} */
```

Other handlers to overload

- Objects can overload several handlers
  - Array access
  - Property access
  - Serializing

zend_object_handlers

```
typedef struct _zend_object_handlers {
	/* general object functions */
	zend_object_add_ref_t                add_ref;
	zend_object_del_ref_t                del_ref;
	zend_object_delete_obj_t             delete_obj;
	/* individual object functions */
	zend_object_clone_obj_t              clone_obj;
	zend_object_read_property_t          read_property;
	zend_object_write_property_t         write_property;
	zend_object_read_dimension_t         read_dimension;
	zend_object_write_dimension_t        write_dimension;
	zend_object_get_property_ptr_ptr_t   get_property_ptr_ptr;
	zend_object_get_t                    get;
	zend_object_set_t                    set;
	zend_object_has_property_t           has_property;
	zend_object_unset_property_t         unset_property;
	zend_object_unset_dimension_t        unset_dimension;
	zend_object_get_properties_t         get_properties;
	zend_object_get_method_t             get_method;
	zend_object_call_method_t            call_method;
	zend_object_get_constructor_t        get_constructor;
	zend_object_get_class_entry_t        get_class_entry;
	zend_object_get_class_name_t         get_class_name;
	zend_object_compare_t                compare_objects;
	zend_object_cast_t                   cast_object;
	zend_object_count_elements_t         count_elements;
} zend_object_handlers;
```

Don't touch these

Keep or inherit

What else ?

þ Iterator support

Part IV

Adding Iterators to objects

þ **Provide an iterator structure**

þ **Provide the handlers**

þ **Provide an iterator creation function**

Iterators

```
/* define an overloaded iterator structure */
typedef struct {
     zend_object_iterator  intern;
     zval                  *current;
} util_dir_it;

static void util_dir_it_dtor(zend_object_iterator *iter TSRMLS_DC);
static int util_dir_it_valid(zend_object_iterator *iter TSRMLS_DC);
static void util_dir_it_current_data(zend_object_iterator *iter,
          zval ***data TSRMLS_DC);
static int util_dir_it_current_key(zend_object_iterator *iter,
          char **str_key, uint *str_key_len, ulong *int_key TSRMLS_DC);
static void util_dir_it_move_forward(zend_object_iterator *iter
          TSRMLS_DC);
static void util_dir_it_rewind(zend_object_iterator *iter TSRMLS_DC);

/* iterator handler table */
zend_object_iterator_funcs util_dir_it_funcs = {
     util_dir_it_dtor,
     util_dir_it_valid,
     util_dir_it_current_data,
     util_dir_it_current_key,
     util_dir_it_move_forward,
     util_dir_it_rewind,
     NULL /* invalidate current */
}; /* }}} */
```

Creating the iterator

- Allocate and initialize the iterator structure
- It is a good idea to increase the original zvals refcount

```
/* {{{ util_dir_get_iterator */
zend_object_iterator *util_dir_get_iterator(zend_class_entry *ce,
                                     zval *object, int by_ref TSRMLS_DC)
{
     util_dir_it *iterator = emalloc(sizeof(util_dir_it));

     if (by_ref) {
          zend_error(E_ERROR, “Iterator invalid in foreach by ref");
     }

     Z_ADDREF_P(object);
     iterator->intern.data = (void*)object;
     iterator->intern.funcs = &util_dir_it_funcs;
     iterator->current = NULL;

     return (zend_object_iterator*)iterator;
} /* }}} */
```

Destructing the iterator

þ Free allocated memory and resources

þ Don't forget to reduce refcount of referenced object

```
/* {{{ util_dir_it_dtor */
static void util_dir_it_dtor(zend_object_iterator *iter TSRMLS_DC)
{
	util_dir_it *iterator = (util_dir_it *)iter;
	zval        *intern = (zval*)iterator->intern.data;

	if (iterator->current) {
		zval_ptr_dtor(&iterator->current);
	}
	zval_ptr_dtor(&intern);

	efree(iterator);
} /* }}} */
```

Getting the data

- Data is read on rewind() and next() calls
- A zval* is stored inside the iterator
- Release current zval
- Create a new zval and assign the value

```
/* {{{ util_dir_it_current */
static void
util_dir_it_current(util_dir_it *iterator, util_dir_object *object
          TSRMLS_DC)
{
      if (iterator->current) {
          zval_ptr_dtor(&iterator->current);
      }
      MAKE_STD_ZVAL(iterator->current);
      if (object->dirp) {
          ZVAL_STRING(iterator->current, object->entry.d_name, 1);
      } else {
          ZVAL_FALSE(iterator->current);
      }
} /* }}} */
```

Iterator valid()

þ **Check whether data is available**

Note: Return `SUCCESS` **or** `FAILURE` **not typical boolean**

```
/* {{{ util_dir_it_valid */
static int
util_dir_it_valid(zend_object_iterator *iter TSRMLS_DC)
{
     util_dir_it     *iterator = (util_dir_it *)iter;
     util_dir_object *object = (util_dir_object*)
                         zend_object_store_get_object(
                              (zval*)iterator->intern.data TSRMLS_CC);

     return object->dirp
          && object->entry.d_name[0] != '\0' ? SUCCESS : FAILURE;
} /* }}} */
```

Iterator key()

þ **The key may be one of:**

- þ **Integer:** `HASH_KEY_IS_LONG`
  Set `ulong *` **to the integer value**
- þ **String:** `HASH_KEY_IS_STRING`
  Set `uint *` **to string length + 1**
  Set `char **` **to copy of string (**`estr[n]dup`**)**

```
/* {{{ util_dir_it_current_key */
static int util_dir_it_current_key(zend_object_iterator *iter, char
**str_key, uint *str_key_len, ulong *int_key TSRMLS_DC)
{
     util_dir_it *iterator = (util_dir_it *)iter;
     zval              *intern = (zval*)iterator->intern.data;
     util_dir_object   *object = (util_dir_object*)
                         zend_object_store_get_object(intern TSRMLS_CC);

     *int_key = object->index;
     return HASH_KEY_IS_LONG;
} /* }}} */
```

Iterator current()

þ **The data was already fetched on rewind() / next()**

```
/* {{{ util_dir_it_current_data */
static void util_dir_it_current_data(zend_object_iterator *iter, zval
     ***data TSRMLS_DC)
{
     util_dir_it *iterator = (util_dir_it *)iter;

     *data = &iterator->current;
} /* }}} */
```

Iterator current()

- þ The data was already fetched on rewind() / next()
- þ Alternatively
  - þ Reset the cached current/key value in rewind() / next()
  - þ Check the cache on access and read if not yet done

```
/* {{{ util_dir_it_current_data */
static void util_dir_it_current_data(zend_object_iterator *iter, zval
     ***data TSRMLS_DC)
{
     util_dir_it *iterator = (util_dir_it *)iter;
     util_dir_object *object;

     if (!iterator->current) {
          object = (util_dir_object*)zend_object_store_get_object(
                                   (zval*)iterator->intern.data TSRMLS_CC);
          util_dir_it_current(iterator, object TSRMLS_CC);
     }
     *data = &iterator->current;
} /* }}} */
```

Iterator next()

þ Move to next element

þ Fetch new current data

```
/* {{{ util_dir_it_move_forward */
static void
util_dir_it_move_forward(zend_object_iterator *iter TSRMLS_DC)
{
	util_dir_it       *iterator = (util_dir_it *)iter;
	zval              *intern = (zval*)iterator->intern.data;
	util_dir_object   *object = (util_dir_object*)
	                zend_object_store_get_object(intern TSRMLS_CC);

	object->index++;
	if (!object->dirp
	|| !php_stream_readdir(object->dirp, &object->entry))
	{
		object->entry.d_name[0] = '\0';
	}

	util_dir_it_current(iterator, object TSRMLS_CC);
} /* }}} */
```

Iterator rewind()

þ Rewind to first element

þ Fetch first current data

```
/* {{{ util_dir_it_rewind */
static void
util_dir_it_rewind(zend_object_iterator *iter TSRMLS_DC)
{
     util_dir_it     *iterator = (util_dir_it *)iter;
     zval            *intern = (zval*)iterator->intern.data;
     util_dir_object *object = (util_dir_object*)
                         zend_object_store_get_object(intern TSRMLS_CC);

     object->index = 0;
     if (object->dirp) {
          php_stream_rewinddir(object->dirp);
     }
     if (!object->dirp
     || !php_stream_readdir(object->dirp, &object->entry))
     {
          object->entry.d_name[0] = '\0';
     }
     util_dir_it_current(iterator, object TSRMLS_CC);
} /* }}} */
```

Iterator drawbacks

- Either implement native iterators at c-level
- Or provide iterator methods and inherit Iterator
- If you want both
  - Your PHP methods call a specialized C-Level handler
  - Provide a cache for your method pointers
  - C-Level iterator functions check this cache
    - On a match call C-Level handler
    - Else call the method
  - Have the iterator struct part of your object struct
    - Use offset_of() for pointer conversion

References

þ This presentation
  http://talks.somabo.de

þ Documentation and Sources to PHP5
  http://php.net

þ http://www.zend.com/php/internals

þ Advanced PHP Programming
  by George Schlossnagle

þ Extending and Embedding PHP
  by Sara Golemon
  ISBN#0-6723-2704-X