Power Your Business with PHP
Zend/PHP Conference & Expo

PHP
Extension Development

Integrating with Existing Systems

Marcus Börger

Zend/PHP Conference

Part I

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 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

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, 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

☑ 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.m4

☑ You can add module dependencies (even to libxml)

- ☑ Optional dependencies are possible (true as 3rd param)

```
dnl $Id: $
PHP_ARG_ENABLE(yourext, enable YourExt suppport,
  [  --enable-yourext            Enable 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

☑ PHP is written in C not C++

- ☑ Do not use // style C++ comments
- ☑ Declarations are only allowed prior to code

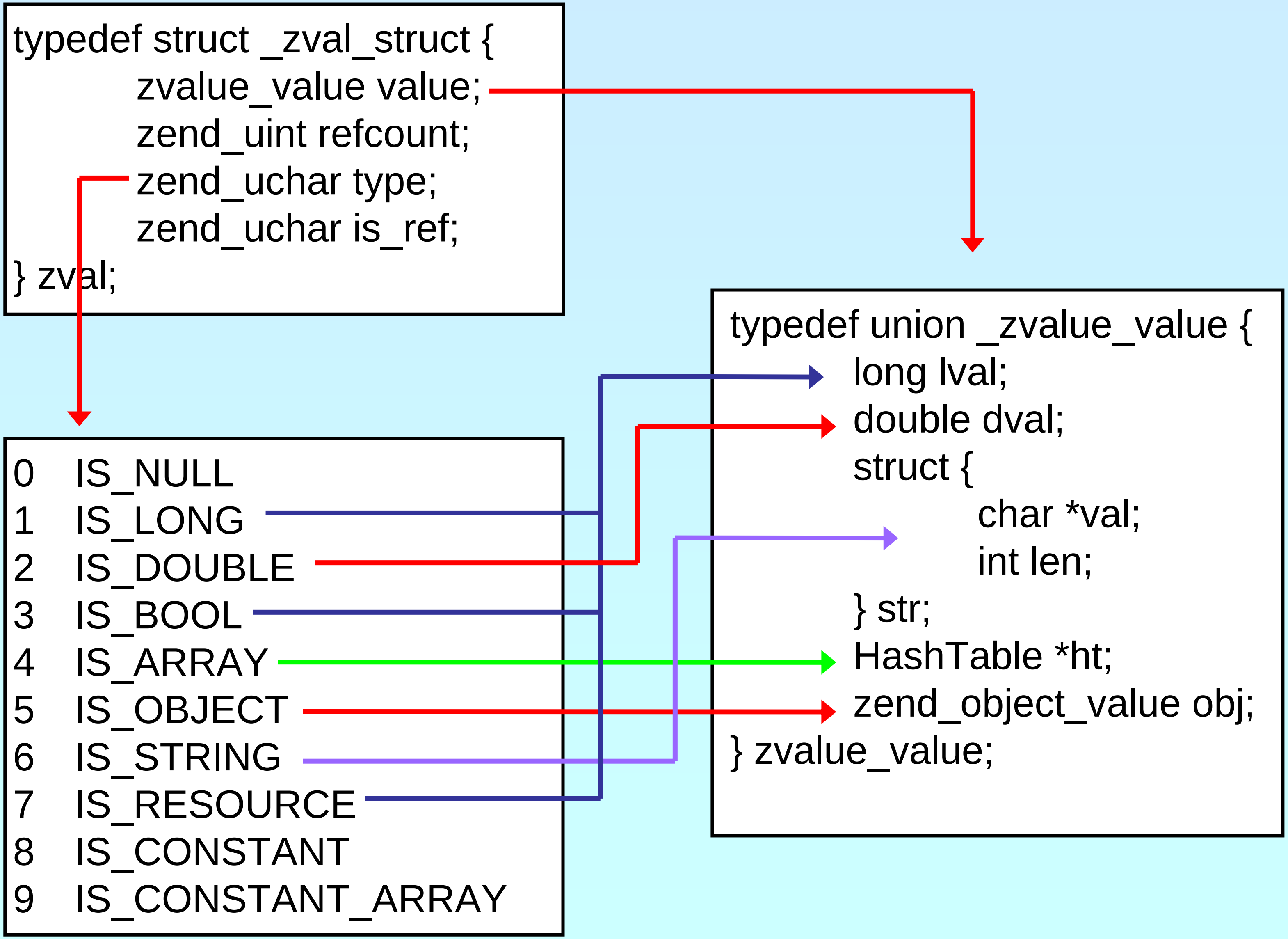
```
/* {{{ proto type yourext_name(params)
 Short description */
PHP_FUNCTION(yourext_name)
{
    /* Local declarations */

    /* Parameter parsing */

    /* Actual code */

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

In PHP all values are zval's



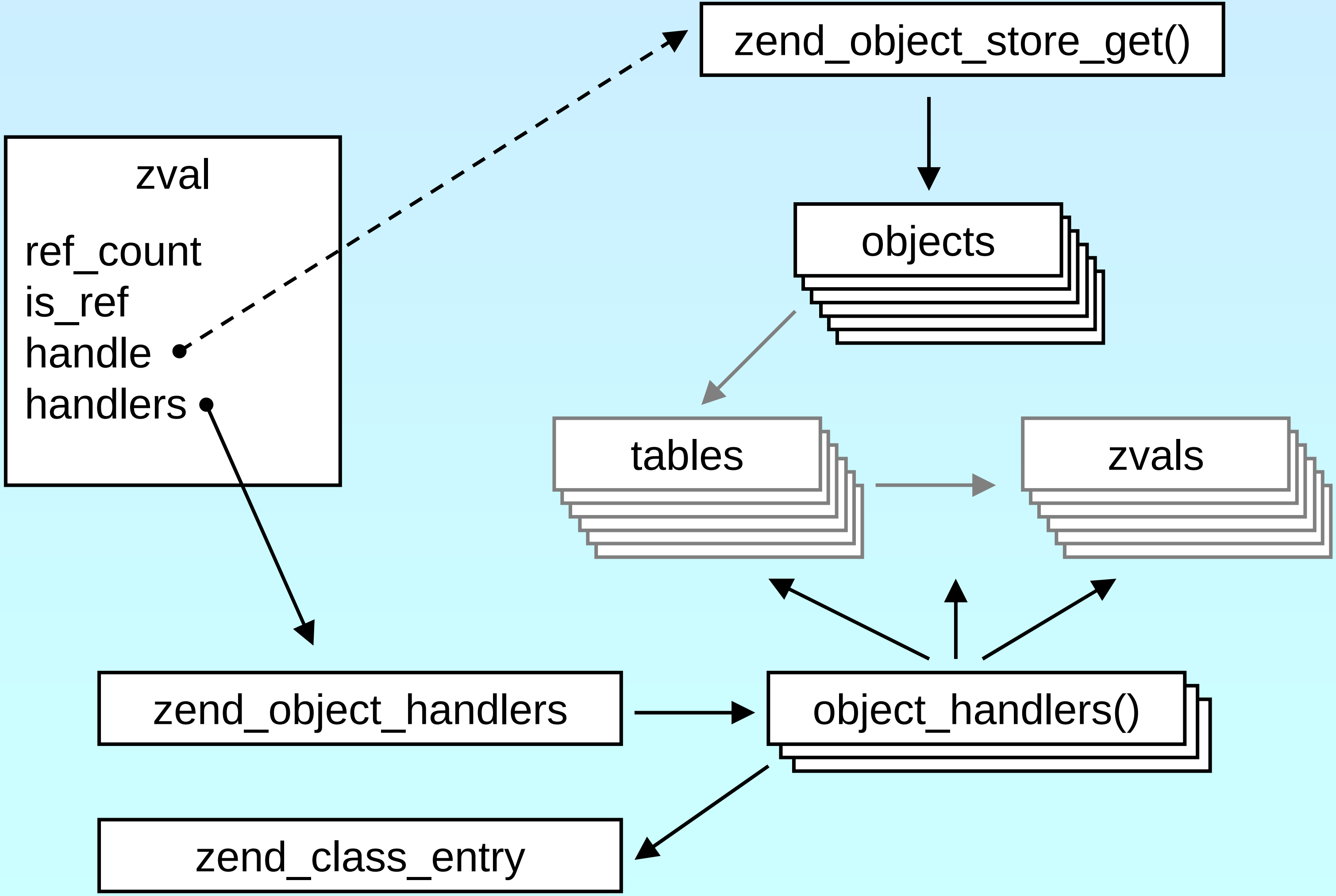
```
typedef struct _zval_struct {
        zvalue_value value;
        zend_uint refcount;
        zend_uchar type;
        zend_uchar is_ref;
} zval;
```

```
0   IS_NULL
1   IS_LONG
2   IS_DOUBLE
3   IS_BOOL
4   IS_ARRAY
5   IS_OBJECT
6   IS_STRING
7   IS_RESOURCE
8   IS_CONSTANT
9   IS_CONSTANT_ARRAY
```

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

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

No need to
copy the string

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_*_P(*zpp)`	

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, *func;
   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, func 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 because it is local

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

Registering consts

☑ MINIT is also the place to register constants
  - ☑ Length here is sizeof()
  - ☑ Thus name must be a real string

```
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);
```

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
- ☑ ZEND_ARG_OBJ_INFO:

  pass_by_ref, name, classname, allow_null

```
static ZEND_BEGIN_ARG_INFO_EX(yourext_args_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_args_name1)
    PHP_FE(yourext_name2,    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

☑ Object support

Part II

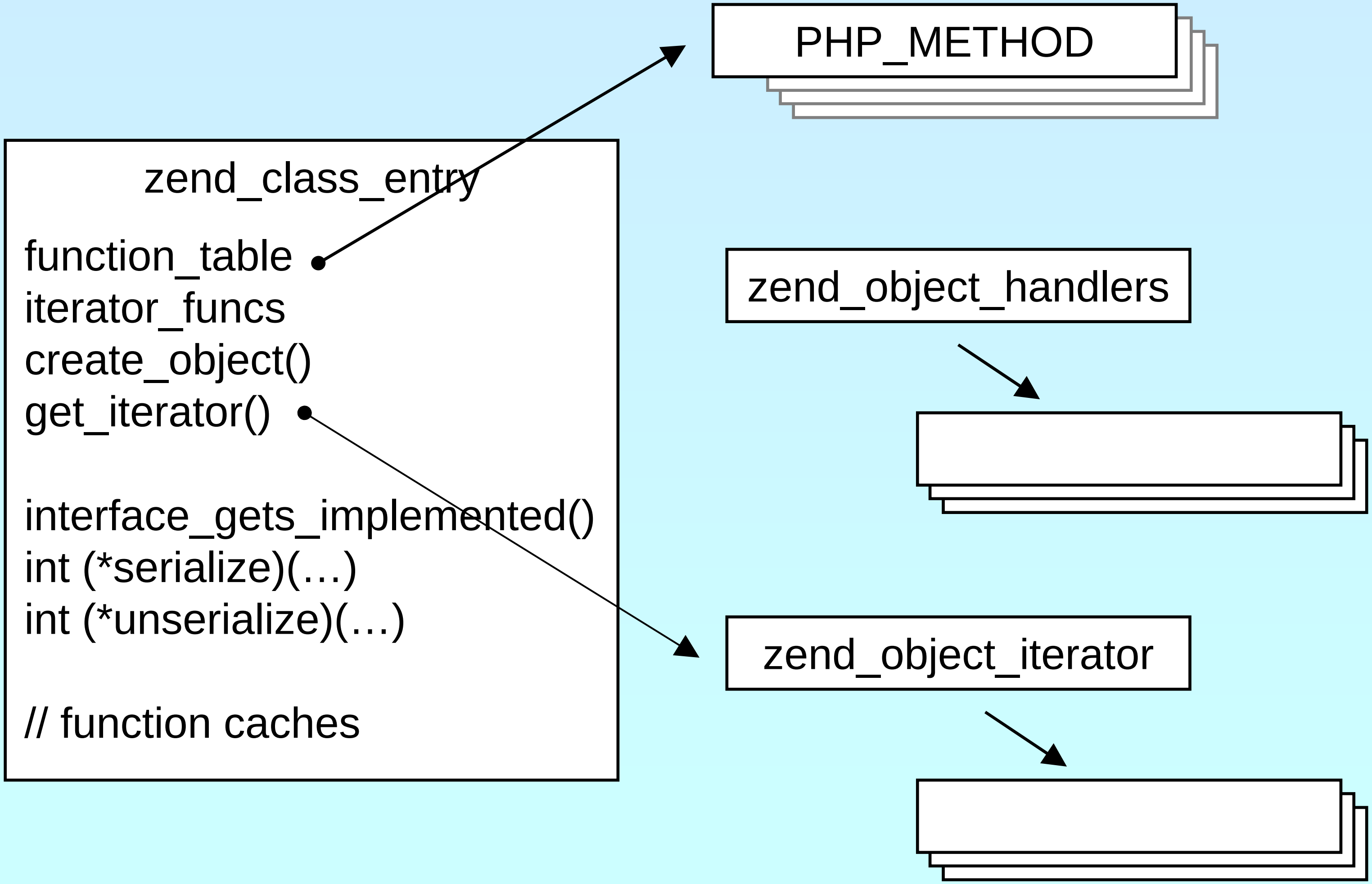
Adding object support

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

What is needed?

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

General class layout



zend_class_entry

function_table
iterator_funcs
create_object()
get_iterator()

interface_gets_implemented()
int (*serialize)(…)
int (*unserialize)(…)

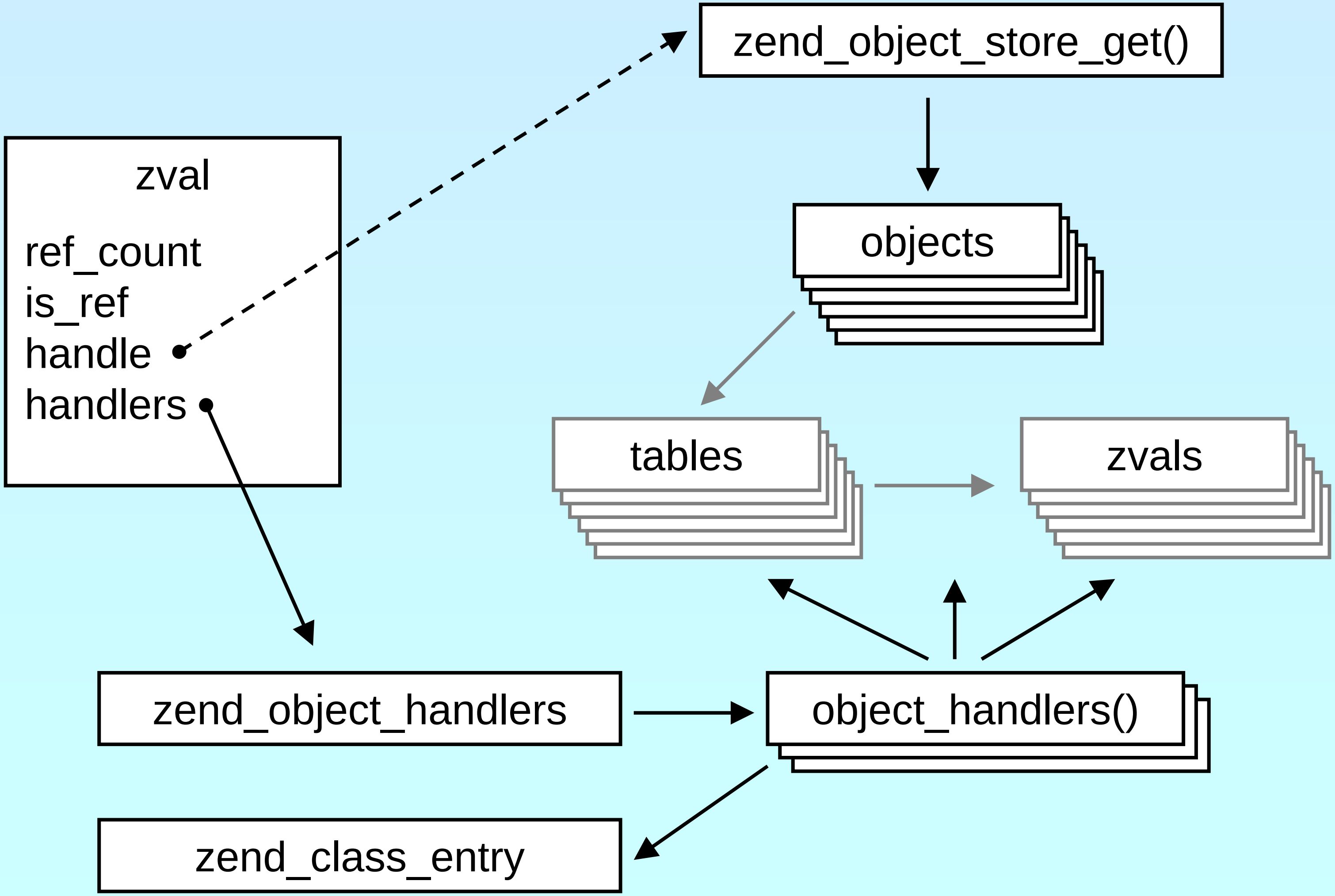
// function caches

PHP_METHOD

zend_object_handlers

zend_object_iterator

General class layout



zend_object_store_get()

zval

ref_count
is_ref
handle
handlers

objects

tables

zvals

zend_object_handlers

object_handlers()

zend_class_entry

Registering

☑ Obviously you have to register your class

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

```
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

☑ You can register class constants

- ☑ Use target zend_class_entry pointer
- ☑ 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

```
/* forward declaration for all methods use (class-name, method-name) */
PHP_METHOD(dir, __construct);
PHP_METHOD(dir, rewind);
PHP_METHOD(dir, hasMore);
PHP_METHOD(dir, key);
PHP_METHOD(dir, current);
PHP_METHOD(dir, next);
PHP_METHOD(dir, getPath);

/* declare method parameters,  */
/* supply a name and default to call by copy */
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_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}
};
```

Declaring methods

☑ Declaring the methods allows

- ☑ To specify parameter names (to support reflection)
- ☑ To specify pass by copy or pass by reference
- ☑ To specify a typehint

See Zend/zend_API.h for ZEND_ARG_*INFO macros

☑ Tip:

If your .c file ends with PHP_MINIT() then you can omit the method forward declarations.

☑ Tip:

There is also zend_parse_method_parameters() but forget about that.

class/object structs

- ☑ It is a good practice to 'inherit' zend_object
  - ☑ That allows your class to support normal properties
  - ☑ 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

☑ Redirect object creation to a general signature

-
  ```
  zend_object_value new(
      zend_class_entry *class_type TSRMLS_DC)
  ```

-
  ```
  zend_object_value new_ex(
      zend_class_entry *class_type,
      util_dir_object **obj TSRMLS_DC)
  ```

```
/* {{{ util_dir_object_new */
/* See util_dir_object_new_ex */
/* creates the object by
   - allocating memory
   - initializing the object members
   - storing the object
   - setting it's handlers
 */
static zend_object_value
util_dir_object_new(zend_class_entry *class_type TSRMLS_DC)
{
     util_dir_object *tmp;
     return util_dir_object_new_ex(class_type, &tmp TSRMLS_CC);
} /* }}} */
```

Object creation/cloning

- ☑ Allcate memory for your struct
- ☑ Initialize the whole struct (Probably by memset(0))
- ☑ Assign the class type
- ☑ Initialize & copy default properties
- ☑ Store the object
- ☑ Assign the handlers

```
intern = emalloc(sizeof(util_dir_object));
memset(intern, 0, sizeof(util_dir_object));
intern->std.ce = class_type;

ALLOC_HASHTABLE(intern->std.properties);
zend_hash_init(intern->std.properties, 0, NULL, ZVAL_PTR_DTOR, 0);
zend_hash_copy(intern->std.properties,
          &class_type->default_properties,
          (copy_ctor_func_t) zval_add_ref,
          (void *) &tmp, sizeof(zval *));

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

Object creation/cloning

```
/* {{{ util_dir_object_new_ex */
static zend_object_value
util_dir_object_new_ex(zend_class_entry *class_type,
          util_dir_object **obj TSRMLS_DC)
{
     zend_object_value retval;
     util_dir_object *intern;
     zval *tmp;

     intern = emalloc(sizeof(util_dir_object));
     memset(intern, 0, sizeof(util_dir_object));
     intern->std.ce = class_type;
     *obj = intern;

     ALLOC_HASHTABLE(intern->std.properties);
     zend_hash_init(intern->std.properties, 0, NULL, ZVAL_PTR_DTOR, 0);
     zend_hash_copy(intern->std.properties,
               &class_type->default_properties,
               (copy_ctor_func_t) zval_add_ref,
               (void *) &tmp, sizeof(zval *));

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

Allocate and init to 0

Standard property support

Register object and make it zval ready

Object cloning

☑ Create a new object (with class entry taken from source)

☑ Clone all struct members

☑ Clone properties and call __clone if defined for that class

```
/* {{{ util_dir_object_clone */
static zend_object_value
util_dir_object_clone(zval *zobject TSRMLS_DC)
{
     zend_object_value new_obj_val, *old_object, *new_object;
     util_dir_object *intern;

     old_object = zend_objects_get_address(zobject TSRMLS_CC);
     new_obj_val = util_dir_object_new_ex(old_object->ce, &intern
                         TSRMLS_CC);
     new_object = &intern->std; /* type conversion */

     util_dir_open(intern, ((util_dir_object*)old_object)->path
                         TSRMLS_CC);

     zend_objects_clone_members(new_object, new_obj_val, old_object,
                         Z_OBJ_HANDLE_P(zobject) TSRMLS_CC);
     return new_obj_val;
} /* }}} */
```

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_hash_destroy(intern->std.properties);
     FREE_HASHTABLE(intern->std.properties);

     if (intern->path) {
          efree(intern->path);
     }
     if (intern->dirp) {
          php_stream_close(intern->dirp);
     }
     efree(object);
} /* }}} */
```

Retrieving the class entry

- ☑ A final class may have its own class entry handler
  - ☑ Little speed-up
  - ☑ Results in problems once you drop 'final'
  - ☑ Standard handler supports inheritance

```
/* {{{ util_dir_get_ce */
static zend_class_entry *util_dir_get_ce(zval *object TSRMLS_DC)
{
        return util_ce_dir;
} /* }}} */
```

A simple method

☑ Macro getThis() gives you access to $this as zval

☑ 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

☑ Remember that your object is already fully initialized

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

☑ Change errors to exceptions to support constructor failure

```
/* {{{ proto void dir::__construct(string path)
 Cronstructs a new dir iterator from a path. */
PHP_METHOD(dir, __construct)
{
     util_dir_object *intern;
     char *path;
     long 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, int should_free 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_STRINGL(retval, "", 0, 1);
        }
        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;
     zend_object_clone_obj_t              clone_obj;
     /* individual object functions */
     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 ?

☑ Iterator support

Part III

Adding Iterator support to your obejcts

☑ Provide an iterator structure

☑ Provide the handlers

☑ Provide an iterator creator 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_has_more(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_has_more,
     util_dir_it_current_data,
     util_dir_it_current_key,
     util_dir_it_move_forward,
     util_dir_it_rewind
}; /* }}} */
```

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 TSRMLS_DC)
{
     util_dir_it *iterator = emalloc(sizeof(util_dir_it));

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

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

Destructing the iterator

☑ Free allocated memory and resources

☑ 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 hasMore()

☑ Check whether more data is available

Note: Return SUCCESS or FAILURE not typical boolean

```
/* {{{ util_dir_it_has_more */
static int
util_dir_it_has_more(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->entry.d_name[0] != '\0' ? SUCCESS : FAILURE;
} /* }}} */
```

Iterator key()

☑ The key may be one of:

- ☑ Integer: HASH_KEY_IS_LONG
  Set ulong * to the integer value
- ☑ 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()

☑ 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()

☑ The data was already fetched on rewind() / next()

☑ Alternatively

- ☑ Reset the cached current/key value in rewind() / next()
- ☑ 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()

☑ Move to next element

☑ 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()

☑ Rewind to first element

☑ 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

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)