

# PHP Extension Writing

Marcus Börger  
Johannes Schlüter

Creating PHP 5 Extension  
PHP Lifecycle  
Adding objects  
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

How PHP handles data

How to create your own extension skeleton

How to create your own functions

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;`

`$a`

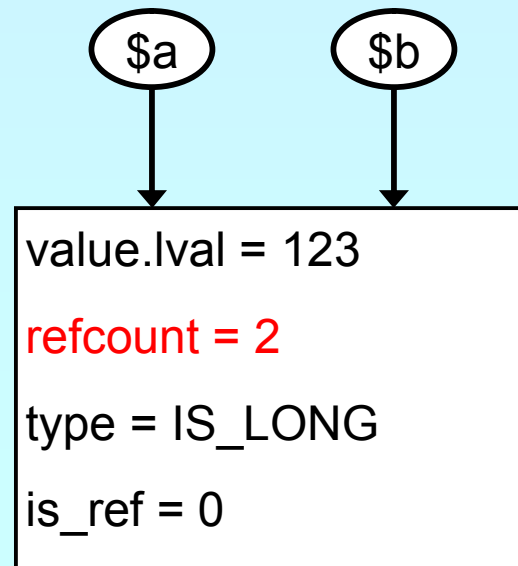
value.lval = 123  
refcount = 1  
type = IS\_LONG  
is\_ref = 0

# 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;
```





# 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

```
value.lval = 123  
refcount = 1  
type = IS_LONG  
is_ref = 0
```

\$b

```
value.lval = 456  
refcount = 1  
type = IS_LONG  
is_ref = 0
```

# 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;`

\$a

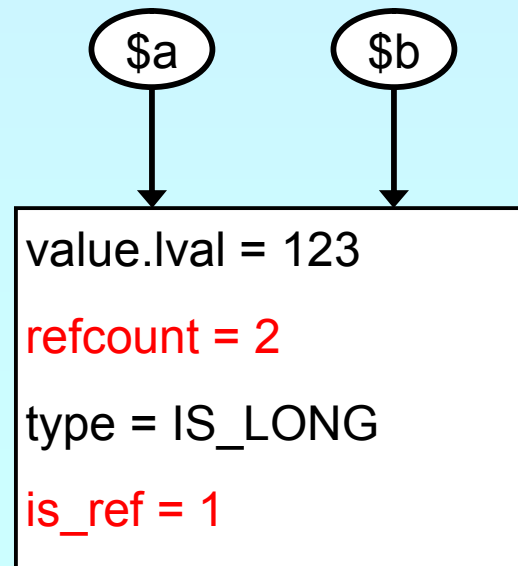
value.lval = 123  
refcount = 1  
type = IS\_LONG  
is\_ref = 0

# 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;
```

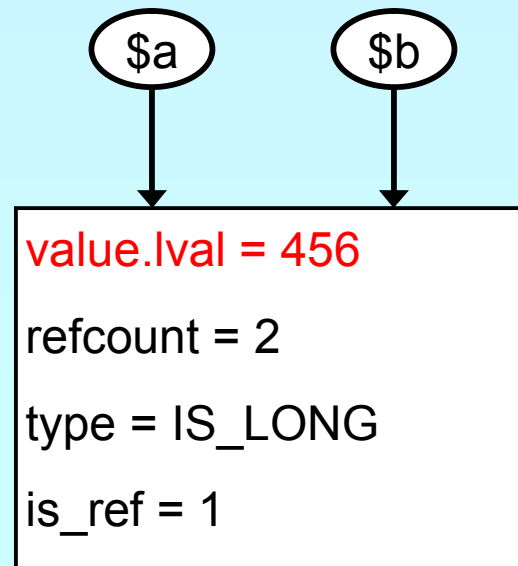


# Full Reference

```
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;
```



# 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

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

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.



# 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

Optional 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

package2.xml         Required for PECL extensions

README              Probably good to provide some lines

# config.m4

PHP Dev is picky about coding style

Read CODING\_STANDARDS in php-src

Watch your whitespace

Align your PHP\_ARG\_ENABLE output

Make your extension default disabled

'phpize' or 'pear install' will enable it automatically

```
dn1 $Id: $
dn1 config.m4 for extension YOUREXT
PHP_ARG_ENABLE(yourext, enable YourExt support,
[ --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

```

dn1 $Id: $
dn1 config.m4 for extension YOUREXT
PHP_ARG_ENABLE(yourext, enable YourExt support,
  [ --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

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

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

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



# Structures and defines not in header

What ever you want

Local storage structures?

Constants?

Macros?

```
typedef struct _php_younext_data {
    int type;

    char *name;
    int name_len;

    php_stream *stream;
} php_younext_data;

#define PHP_YOUREXT_MEANING        42
#define PHP_YOUREXT_COLOR         "purple"

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



# Helper Functions

Use **TSRMLS\_xx** as last function parameter

When dealing with PHP Data

Use **--enable-maintainer-zts** when building PHP

Use **static** or **inline**

If you need the function only in your .c file

Use **PHPAPI** / **YOREXT\_API**

If you plan to use the functions in other extensions

# Helper Functions

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

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

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
}; /* }}} */

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

Diagram: A red box highlights the function list in the code, with a line pointing to a separate box containing the text "or NULL".

# Function List

Exports your functions to userspace

Must be terminated by NULL triplet

```
zend_function_entry yourex_functions[] = { /* {{{ */  
    PHP_FE(yourex_func1,      yourex_args_func1)  
    PHP_FE(yourex_func2,      NULL)  
    PHP_FALIAS(yourex_func3, yourex_func2, NULL)  
    PHP_NAMED_FE(yourex_func4, _yourex_func4_impl,  
                NULL)  
  
    {NULL, NULL, NULL}  
};
```

# ArgInfo / Signatures

The function table allows specifying 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

Namespace your functions with your ext's name

Documentation is your friend

Avoid // style C++ comments

Avoid declarations inline with code

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

    /* Parameter parsing */

    /* Actual code */

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



# Outputting Content

Do not send content to stdout

use PHP's output buffering mechanisms

php\_printf() works just like printf()

PHPWRITE() respects binary safety

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

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

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



# 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	scanf 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

<code>type_spec</code>	scanf like typelist (though no %)	
<code>l</code>	<code>long</code>	<code>long *</code>
<code>d</code>	<code>double</code>	<code>double *</code>
<code>b</code>	<code>boolean</code>	<code>zend_bool *</code>
<code>a</code>	<code>array</code>	<code>zval **</code>
<code>o</code>	<code>object</code>	<code>zval **</code>
<code>O</code>	<code>object</code>	<code>zval **, zend_class_entry *</code>
	Object must be derived from given class	
<code>s</code>	<code>string</code>	<code>char **, int *</code>
	You receive string and length	
<code>r</code>	<code>resource</code>	<code>zval **</code>
<code>z</code>	<code>zval</code>	<code>zval **</code>
<code>Z</code>	<code>zval-ref</code>	<code>zval ***</code>
<code> </code>	right part is optional	
<code>/</code>	next param gets separated if not reference	
<code>!</code>	Next param returns NULL if param type IS_NULL	



# Parsing Parameters

```
/* {{{ proto null youext_hello(string name)
   Greet by name */
PHP_FUNCTION(youext_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

## Marking success

```

/* {{{ proto bool youext_hello(string name)
   Greet by name */
PHP_FUNCTION(youext_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

Simple scalars use intuitive RETURN\_\*() macros

```
RETURN_NULL();  
RETURN_BOOL(b);  
RETURN_TRUE;  
RETURN_FALSE;  
RETURN_LONG(l);  
RETURN_DOUBLE(d);
```

b: 0 => FALSE, non-0 => TRUE  
RETURN\_BOOL(1)  
RETURN\_BOOL(0)  
l: Integer value  
d: Floating point value



# Returning Values

Strings are slightly more complex

The string value must "belong" to the engine

- Will not survive the destruction of the zval

- Will be freed using `efree()`

Pass 0 (zero) for *dup* to give it the string

Pass 1 (one) for *dup* to make a copy (*duplicate*)

```
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

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

Inverting a single boolean parameter

```
/* {{{ proto bool youext_invert(bool b)
   Invert a boolean parameter */
PHP_FUNCTION(youext_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

Incrementing a value with an optional maximum

```

/* {{{ proto int youext_increment(int v [, int max])
   Increment a value with optional maximum */
PHP_FUNCTION(youext_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

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

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

Never use sprintf,  
use either snprintf or sprintf

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

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, ...)`

```
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

## Returning an array

```

/* {{{ proto array youext_version_array()
Retrieve youext version as array */
PHP_FUNCTION(youext_version_array)
{
    char *ver;
    int len = sprintf(&ver, 0, "%d.%d",
        YOUREXT_VERSION_MAJOR, YOUREXT_VERSION_MINOR);

    array_init(return_value); ← make return_value an array
    add_assoc_long(return_value, "major",
        YOUREXT_VERSION_MAJOR);
    add_assoc_long(return_value, "minor",
        YOUREXT_VERSION_MINOR);
    add_assoc_string(return_value, "cvs", "$Id: $", 1);
    add_assoc_stringl(return_value, "ver", ver, len, 0);
}
/* }}} */

```

# Dealing with a HashTable

Multiple values stored in key/value pairs

Arrays are special HashTables (Symbol tables)

Numeric keys get converted to strings

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 existence** 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

<code>Z_REFCOUNT(zval)</code>	Retrieve reference count
<code>Z_SET_REFCOUNT(zval,rc)</code>	Set reference count to <rc>
<code>Z_ADDREF(zval)</code>	Increment reference count
<code>Z_DELREF(zval)</code>	Decrement reference count
<code>Z_ISREF(zval)</code>	whether <code>zval</code> is a reference
<code>Z_SET_ISREF(zval)</code>	Makes <code>zval</code> a reference variable
<code>Z_UNSET_ISREF(zval)</code>	Resets the is-reference flag
<code>Z_SET_ISREF_TO(zval,is)</code>	Make <code>zval</code> a reference is <is> != 0
<code>Z*_P(zp)</code>	<code>Z_*( *zp)</code>
<code>Z*_PP(zpp)</code>	<code>Z_*( **zpp)</code>



# 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

<code>ALLOC_ZVAL(zp)</code>	Allocate a zval using <code>emalloc()</code>
<code>INIT_PZVAL(zp)</code> <code>INIT_ZVAL(zval)</code>	Set reference count and <code>isref</code> 0 Initialize and set NULL, no pointer
<code>ALLOC_INIT_ZVAL(zp)</code> <code>MAKE_STD_ZVAL(zp)</code>	Allocate and initialize a zval Allocate, initialize and set NULL

Example:

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



# Dealing with a HashTable

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

Hash tables have builtin "foreach" functions

Each function requires a different type of callback

Callbacks return one of three status values

Prior to 5.2.1 all non zero return values result in deletion

```
/* Continue iterating 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

Using zend\_hash\_apply\_with\_arguments()

```

/* {{{ proto void youext_foreach( array names,
                                string greeting)
   Say hello to each person */
PHP_FUNCTION(youext_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)youext_foreach, greet TSRMLS_CC);
} /* }}} */

```



# Example 5 b

Calling a function for each element

```

/* {{{ youext_foreach
   Callback for outputting a greeting
   for each name in a user-provided array */
int youext_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

The PHP Lifecycle

Memory Allocation and Garbage Collection

Globals

Constants



# STARTUP

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



# ACTIVATION

Triggered upon receiving a new request (page hit)

Initialize environment and variables  
(symbol\_table, EGPCS)

Activate (RINIT) static built modules

Activate (RINIT) shared modules



# RUNTIME

Actual execution of scripts happens here.

Compile and execute `auto_prepend_file`.

Compile and execute `main_file`.

Compile and execute `auto_append_file`.

# DEACTIVATION

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

Call user-defined shutdown functions.

Destroy object instances.

Flush output.

Deactivate (RSHUTDOWN) modules

(in reverse of activation order)

Clean up environment

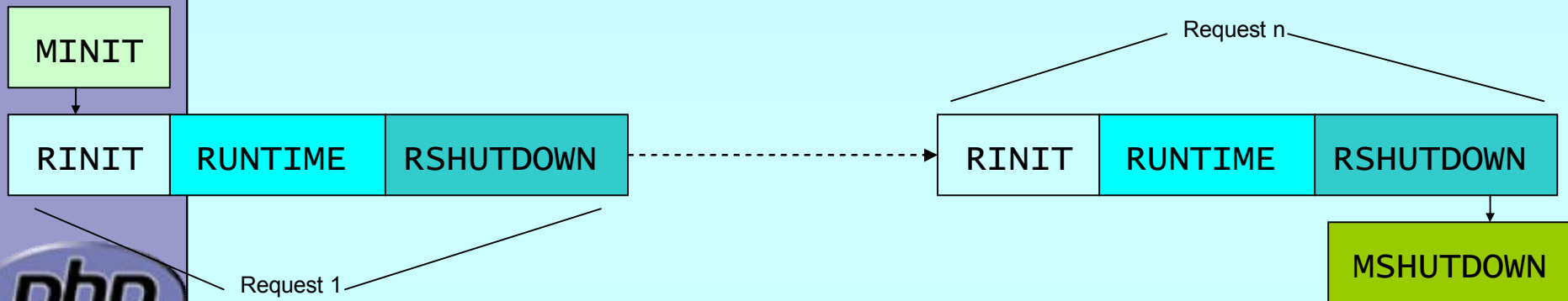
Implicitly free remaining non-persistent memory.

# 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

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

Traditional malloc() family may be used

Non-persistent allocators prefixed with e

Additional helpers provided by engine

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 adt1);  
void *STR_EMPTY_ALLOC(void);
```



# Memory Allocation

Traditional malloc() family may be used

Non-persistent allocators prefixed with *e*

Selective allocators prefixed with *pe*

pestrndup() not available

safe\_pemalloc() requires PHP  $\geq$  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 *errmsg = NULL;

PHP_FUNCTION(youext_unthreadsafe) {
    long ret;

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



# Global struct in .h

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) \
    TSRMLSMG(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(youext) {  
    ZEND_INIT_MODULE_GLOBALS(youext,  
        youext_globals_ctor, youext_globals_dtor);  
    return SUCCESS;  
}  
  
PHP_MSHUTDOWN_FUNCTION(youext) {  
#ifndef ZTS  
    youext_globals_dtor(&youext_globals TSRMLS_CC);  
#endif  
    return SUCCESS;  
}
```



# RINIT/RSHUTDOWN

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

```
PHP_RINIT_FUNCTION(youext) {  
    /* Track number of times this thread/process  
     * has serviced requests */  
    YOUREXT_G(counter)++;  
    return SUCCESS;  
}  
  
PHP_RSHUTDOWN_FUNCTION(youext) {  
    if (YOUREXT_G(str)) {  
        efree(YOUREXT_G(str));  
        YOUREXT_G(str) = NULL;  
    }  
    return SUCCESS;  
}
```



# Globals Access

Access global values using `YOUEXT_G(v)` macro

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



# Globals Access

Access global values using `YOUEXT_G(v)` macro

```
PHP_FUNCTION(youext_get_string) {  
    if (YOUEXT_G(str)) {  
        RETURN_STRINGL(YOUEXT_G(str), YOUEXT_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

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(str));

    php_info_print_table_end();
}
```





# What else ?

INI Handling

Dealing with resources and streams

Object support

# Part III

## Adding objects

How to create your own classes

How to create interfaces

How to create methods

What can be overloaded

# What is needed?

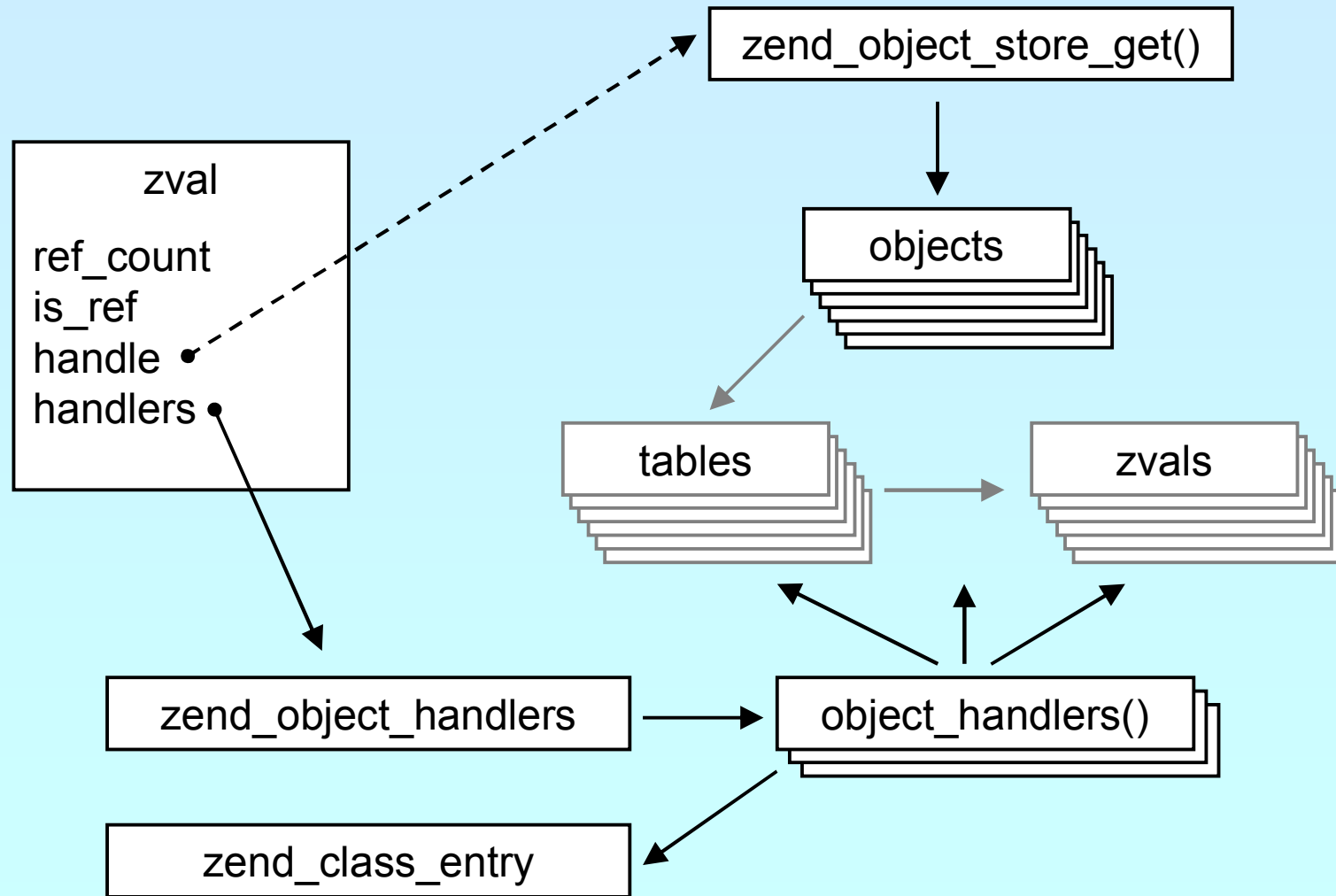
Providing methods

Providing a `zend_class_entry` pointer

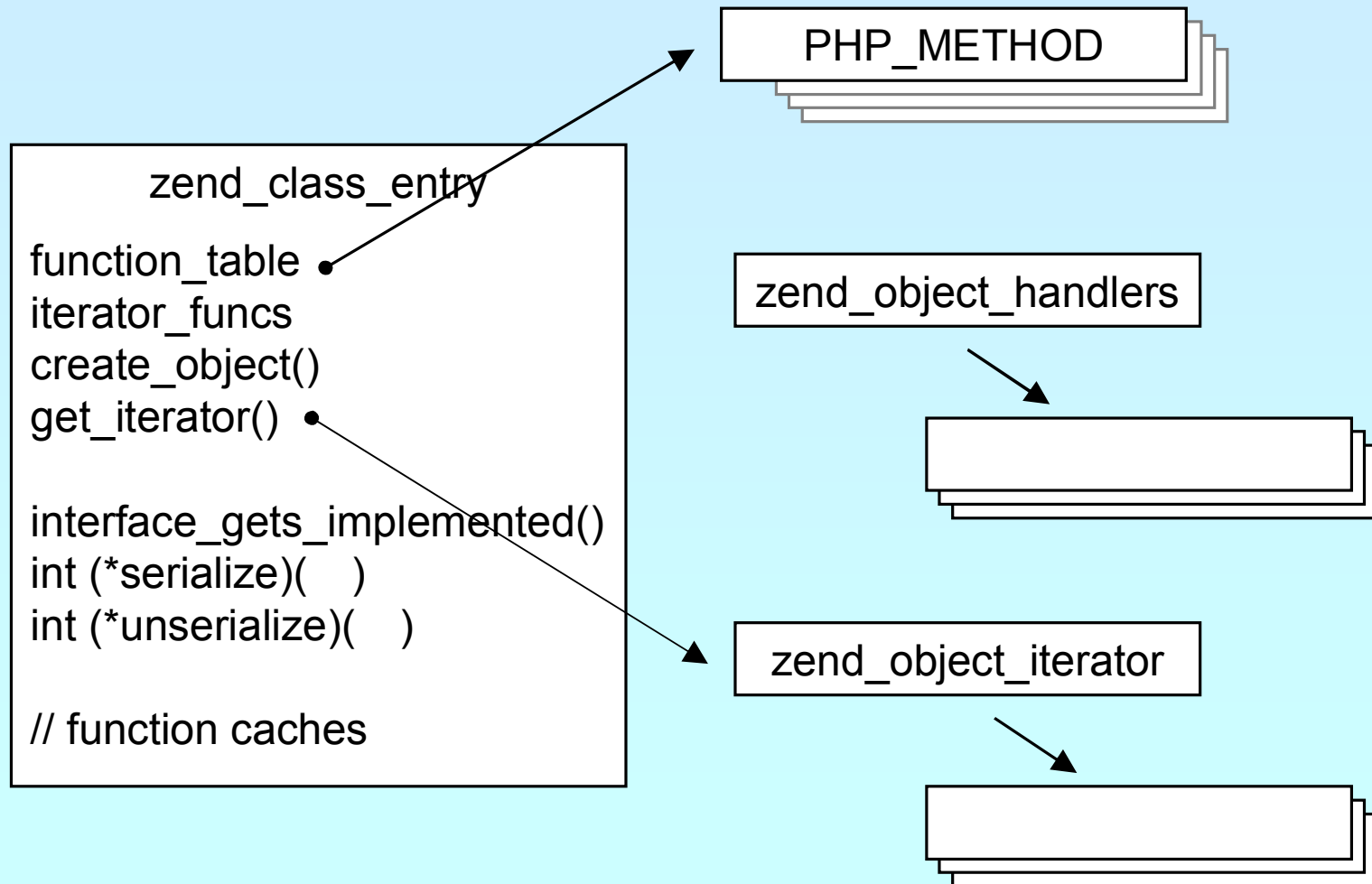
Providing object handlers

Registering the class

# General class layout



# General class layout



# 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

```
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

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

```

/* 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

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;

/* declare 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

Allcate memory for your struct

Initialize the whole struct (probably by using `ecalloc()`)

Initialize the base Zend object

Copy default properties

Store the object

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

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.

```
/* {{{ 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

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)
   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;          Don't touch these
    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;   Keep or inherit
    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;

```

# What else ?

Iterator support

# Part IV

## Adding Iterators to objects

Provide an iterator structure

Provide the handlers

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

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 valid()

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()

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

- Have the iterator struct part of your object struct

  - Use `offset_of()` for pointer conversion

# 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

