php|Tropics

Implementing PHP 5 OOP extensions

Marcus Börger

After creating a basic PHP5 Extensions

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

PHP 5 Extensions

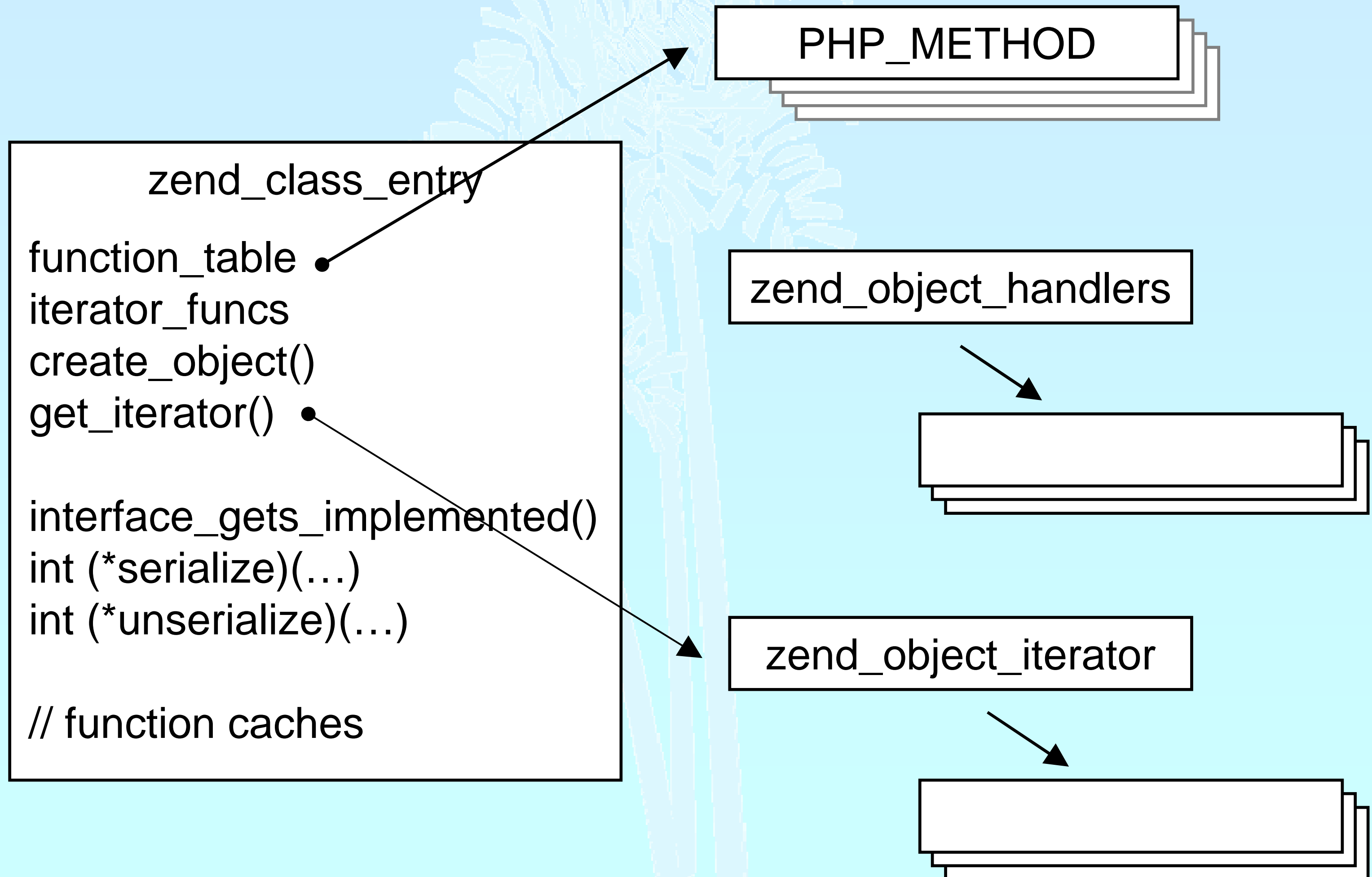
- ☑ PHP 5 extensions are the same as in PHP 4
- ☑ ext_skel generates the basic skeleton

```
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
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.
```

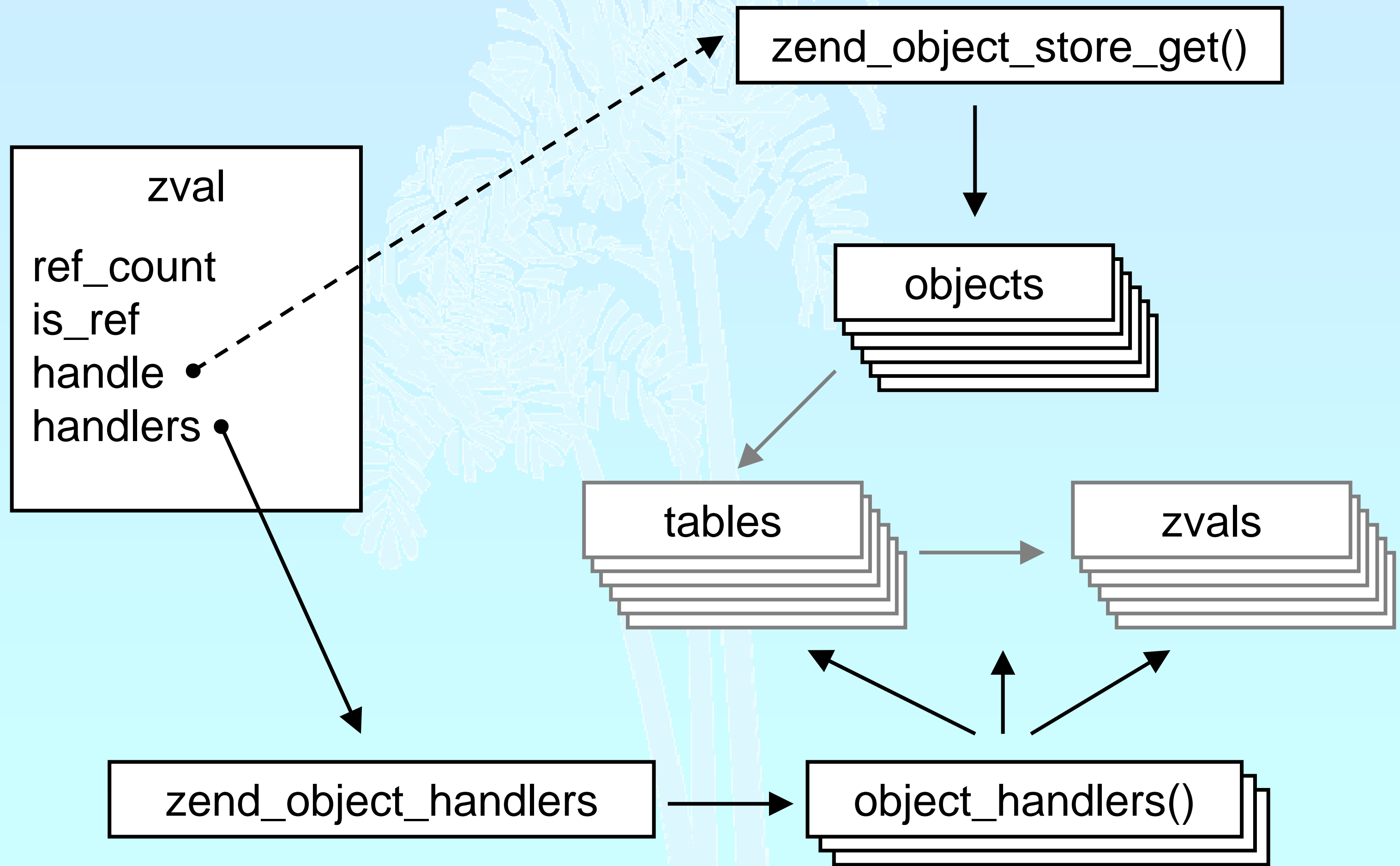
What is needed?

- ☑ 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

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

php | Tropics

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.

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;
```

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

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

php | Tropics

Object destruction

- ☑ Free properties
- ☑ Free all resources and free all allocated memory

```
/* {{{ 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);
} /* }}} */
```

php|Tropics

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

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

php | Tropics

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

php | Tropics

Iterator rewind()

- ☑ Rewind to first element
- ☑ Fetch furst 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

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

What else ?

- ☑ 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_handlers;
```

Don't touch these

Keep or inherit

References

- Source to ext/util
  http://somabo.de/php/ext/util
- This presentation
  http://talks.somabo.de
- Documentation and Sources to PHP5
  http://php.net
- Advanced PHP Programming
  by George Schlossnagle
- Extending and Embedding PHP
  by George Schlossnagle, Wez Furlong
  (not yet published)