

# PHP 5

## Object Oriented 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

# PHP5 Extensions

- ☑ PHP5 extensions are the same as in PHP4
- ☑ 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:

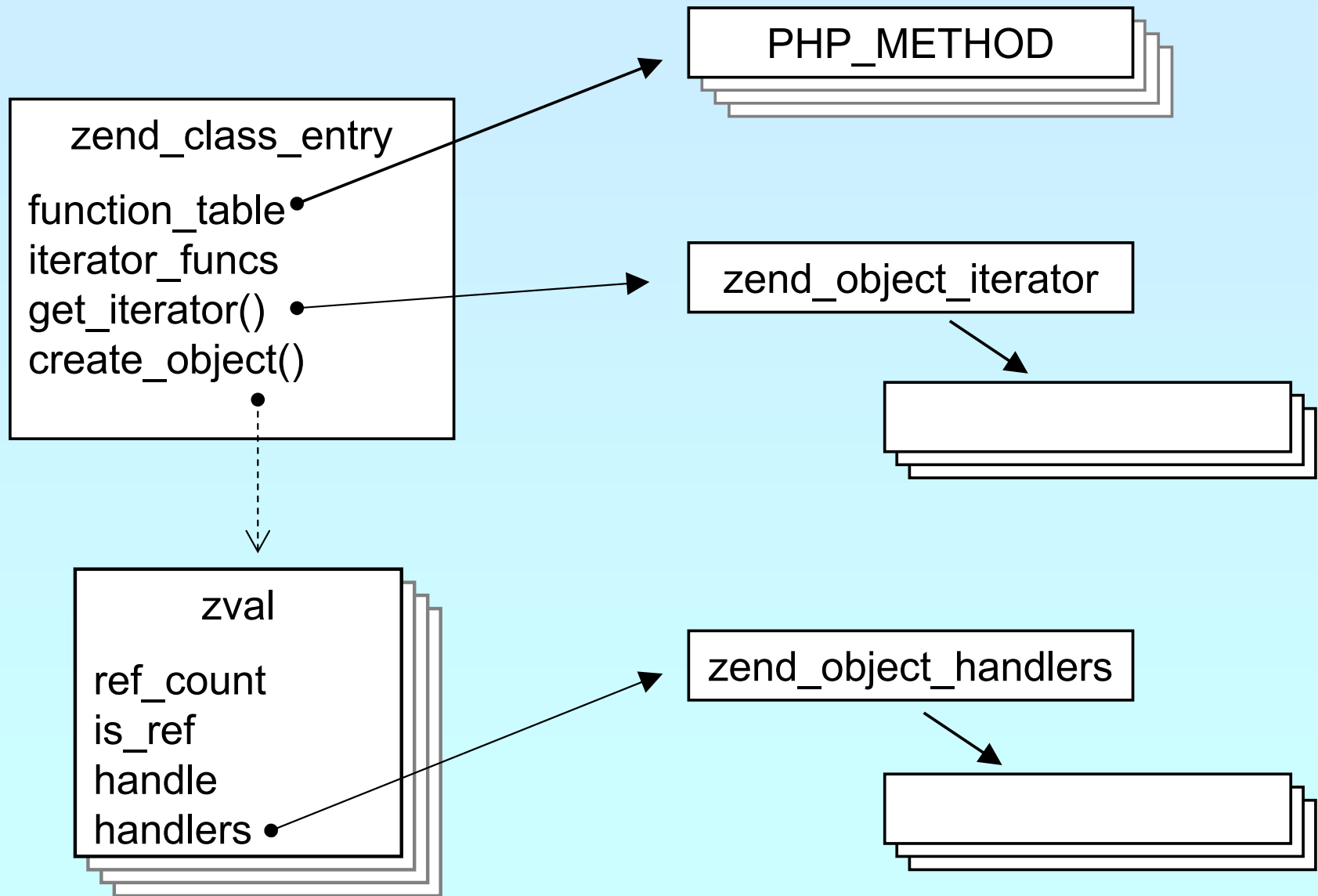
1. \$ cd ..
2. \$ vi ext/util/config.m4
3. \$ ./buildconf
4. \$ ./configure --[with|enable]-util
5. \$ make
6. \$ ./php -f ext/util/util.php
7. \$ vi ext/util/util.c
8. \$ make

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.

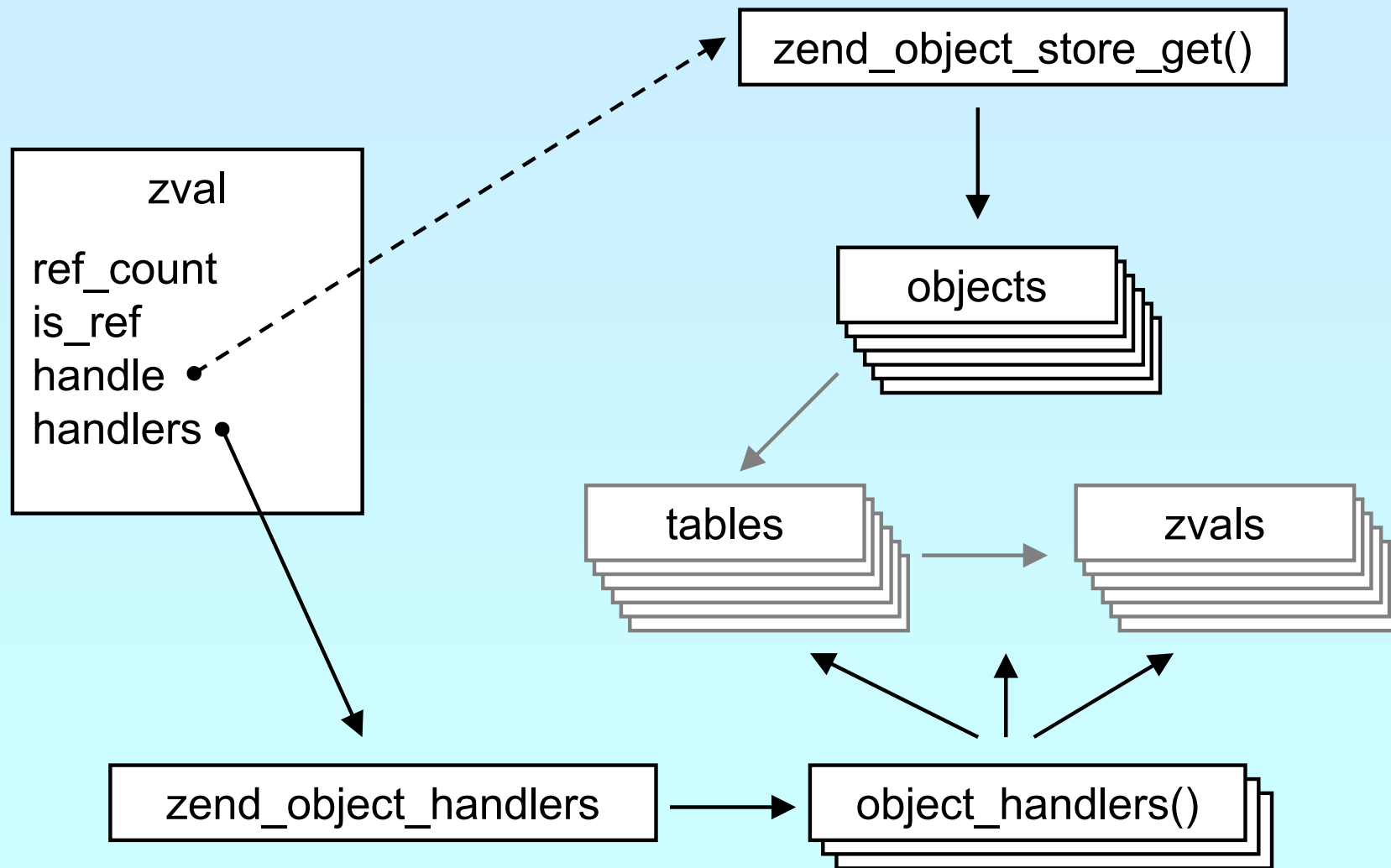
# What is needed?

- ☑ Providing methods
- ☑ Providing a zend\_class\_entry pointer
- ☑ Providing object handlers
- ☑ Registering the class

# General class layout



# General object layout



# Registering

```
/* {{{ PHP_MINIT_FUNCTION(util) */
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;
    zend_class_implements(util_ce_dir TSRMLS_CC, 1, zend_ce_iterator);
    memcpy(&util_dir_handlers, zend_get_std_object_handlers(),
        sizeof(zend_object_handlers));
    util_dir_handlers.clone_obj = util_dir_object_clone;

    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, valid);
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, valid, 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

```
/* 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;
```

# Object creation

```
/* {{{ 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

```
/* {{{ util_dir_object_new */
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

```
/* {{{ util_dir_object_clone */
static zend_object_value
util_dir_object_clone(zval *zobject TSRMLS_DC)
{
    zend_object_value new_obj_val;
    zend_object *old_object;
    zend_object *new_object;
    zend_object_handle handle = Z_OBJ_HANDLE_P(zobject);
    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;

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

    zend_objects_clone_members(new_object, new_obj_val, old_object,
                              handle TSRMLS_CC);

    return new_obj_val;
}
/* }}} */
```

# Object destruction

```
/* {{{ 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

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

# A simple method

```
/* {{{ 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

```
/* {{{ proto void dir::__construct(string path)
   Constructs a new dir iterator from a path. */
PHP_METHOD(dir, __construct)
{
    zval *object = getThis();
    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) == FAILURE) {
        php_set_error_handling(EH_NORMAL, NULL TSRMLS_CC);
        return;
    }

    intern = (util_dir_object*)zend_object_store_get_object(object
        TSRMLS_CC);
    util_dir_open(intern, path TSRMLS_CC);

    php_set_error_handling(EH_NORMAL, NULL TSRMLS_CC);
}
/* }}} */
```



# Iterators

```
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);
static void util_dir_it_invalidate(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,
    util_dir_it_invalidate
};
/* }}} */
```

# Creating the iterator

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

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

# Invalidating the iterator

```
/* {{{ util_dir_it_invalidate */
static void util_dir_it_invalidate(zend_object_iterator *iter TSRMLS_DC)
{
    util_dir_it *iterator = (util_dir_it *)iter;

    if (iterator->current) {
        zval_ptr_dtor(&iterator->current);
        iterator->current = NULL;
    }
}
/* }}} */
```

# Destructing the iterator

```
/* {{{ 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;  
  
    util_dir_it_invalidate(iter TSRMLS_CC);  
    efree(iterator);  
}  
/* }}} */
```

# Iterator valid()



Check whether more data is available

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

# Getting the data

- ✓ Data is read on next() calls
- ✓ A zval\* is stored inside the iterator

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

# Iterator current\_data

```
/* {{{ 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;

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

# Iterator current\_key

```
/* {{{ 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 move\_forward

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

# Iterator rewind

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

# What else ?



Objects can overload several handlers

- Object casting
- Array access
- Property access

# 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_has_dimension_t has_dimension;
    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;
```

# Object casting

```
/* {{{ */
static int
zend_cast_exception(zval *readobj, zval *writeobj, int type, int
should_free TSRMLS_DC)
{
    if (type == IS_STRING) {
        zval fname, *retval;

        ZVAL_STRING(&fname, "__tostring", 0);
        if (call_user_function_ex(NULL, &readobj, &fname,
            &retval, 0, NULL, 0, NULL TSRMLS_CC) == SUCCESS)
        {
            ZVAL_STRING(writeobj, Z_STRVAL_P(retval), 1);
            zval_ptr_dtor(&retval);
            return SUCCESS;
        }
    }
    return FAILURE;
}
/* }}} */
```

# Array overloading

```
static zval *
sxe_dimension_read(zval *object, zval *offset TSRMLS_DC)
{
    convert_to_string_ex(&offset);
    /* MAKE_STD_ZVAL() */
    return sxe_prop_dim_read(object, offset, 0, 1, 0 TSRMLS_CC);
}

static void
sxe_dimension_write(zval *object, zval *offset, zval *value TSRMLS_DC)
{
    convert_to_string_ex(&offset);
    sxe_prop_dim_write(object, offset, value, 0, 1 TSRMLS_CC);
}

static void
sxe_dimension_delete(zval *object, zval *offset TSRMLS_DC)
{
    convert_to_string_ex(&offset);
    sxe_prop_dim_delete(object, offset, 1, 1 TSRMLS_CC);
}
```

# Element overloading

```
static zval *
sxe_element_read(zval *object, zval *offset TSRMLS_DC)
{
    /* MAKE_STD_ZVAL() */
    return sxe_prop_dim_read(object, offset, 0, 1, 0 TSRMLS_CC);
}

static void
sxe_element_write(zval *object, zval *offset, zval *value TSRMLS_DC)
{
    sxe_prop_dim_write(object, offset, value, 0, 1 TSRMLS_CC);
}

static void
sxe_element_delete(zval *object, zval *offset TSRMLS_DC)
{
    sxe_prop_dim_delete(object, offset, 1, 1 TSRMLS_CC);
}

static int
sxe_property_exists(zval *object, zval *member, int check_empty
    TSRMLS_DC)
{
    return 0; /* or 1 */
}
```

# References

- ☑ Source to ext/util  
<http://somabo.de/php/ext/util>
  
- ☑ Documentation and Sources to PHP5  
<http://php.net>
  
- ☑ Describing Zend Engine  
Harald Radi  
Extending PHP  
PHP5's object model  
international php magazine, issue 4.03