facebook

PHP Extension Writing

Sara Golemon
pollita@php.net
Feb 22, 2013

Agenda

1. Extension Basics
2. Module Hooks
3. Objects and Classes
4. Linking External Libraries
5. Where to go next

Extension Basics

Preparing to build extensions

- Distro package: php-devel
- Build PHP from source:
  - ./configure --enable-debug \
    --enable-maintainer-zts
- Debug: Reports memory leaks
- ZTS: Enables thread safety checks
- phpize – Module Makefiles

Hello World

- autoconf fragment: config.m4
- Entry point symbol: ZEND_GET_MODULE()
- Exposes function: hello()

```
dnl config.m4
PHP_ARG_ENABLE(hello,
               whether to enable hello module,
[ --enable-hello   Enable hello support ])

if test “$PHP_HELLO” != “no”; then
  PHP_NEW_EXTENSION(hello,
                    hello.c,
                    $ext_shared)
fi
```

```
/* hello.c */
#include “php.h”
#include “config.h”
#define HELLO_MODULE_VERSION “1.0”
PHP_FUNCTION(hello) {
  php_printf(“Hello World!\n”);
}
zend_function_entry hello_functions[] = {
    PHP_FE(hello,  NULL)
    { NULL, NULL, NULL }
};
zend_module_entry hello_module_entry = {
    STANDARD_MODULE_HEADER,
    ”hello",
    hello_functions,
    NULL, NULL, NULL, NULL, NULL,
    HELLO_MODULE_VERSION,
    STANDARD_MODULE_PROPERTIES
};
#ifdef COMPILE_DL_HELLO
ZEND_GET_MODULE(hello)
#endif
```

Building the Hello extension

- phpize creates ./configure
- ./configure creates Makefile
- make creates modules/hello.so
- Creates modules/hello.so

```
~ $ mkdir hello
~ $ cd hello
hello $ vi config.m4
hello $ vi hello.c
hello $ phpize
PHP Api Version:         20121113
Zend Module Api No:      20121212
Zend Extension Api No:   220121212
hello $ ./configure
…lots of output…
configure: creating ./config.status
config.status: creating config.h
hello $ make
…lots of output…
Build complete.
Don't forget to run 'make test'.
hello $ php -n -dextension_dir=module/ \
            -dextension=hello.so \
            -r 'hello();'
Hello World!
hello $
```

Accepting parameters

- Pull parameters off the stack
- zpp performs type checks and casts to primitive C types
- Automatically throws meaningful error message on FAILURE

```
ZEND_BEGIN_ARG_INFO(hello_arginfo, 0)
  ZEND_ARG_INFO(0, name)
ZEND_END_ARG_INFO();

/* proto void hello(string $name) */
PHP_FUNCTION(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);
}

zend_function_entry hello_functions[] = {
    PHP_FE(hello,  hello_arginfo)
    { NULL, NULL, NULL }
};
```

Accepting More parameters

- Type specifiers match arg types
- All specifiers after pipe optional
  Unmodified if not passed
- Altering data can modify passed variables ***even if not passed by ref***
  - "/" to separate if you expect to modify

Modifier	Meaning
"|"	All subsequent params are optional
"!"	Do not modify var if NULL was passed
"/"	Separate value from caller

Spec.	Argument pass type	Purpose
"b"	zend_bool*	Boolean
"l"/"L"	long*	Integer (Rollover/Trunc)
"d"	double*	Floating Point
"s"	char **, int*	String
"p"	char **, int*	String (w/ NULL checks)
"a"	zval**	Array
"A"	zval**	Array or Object
"h"	HashTable**	Array
"H"	HashTable**	Array or Object Props
"r"	zval**	Resource
"o"	zval**	Object
"O"	zval**,zend_class_entry*	Object of specific Class
"C"	zend_class_entry**	Class (by name)
"f"	zend_fcall_info*, zend_fcall_info_cache*	Callback function
"z"/"Z"	zval** / zval***	Any variable type
"*"/"+"	zval****, int*	Var args

Accepting Optional parameters

```
/* proto void hello(string $name[, string $salutation = “Ms.”]) */
PHP_FUNCTION(hello) {
  char *name, *salutation = “Ms.”;
  int name_len, sal_len = strlen(salutation);

  if (zend_parse_parameters(ZEND_NUM_ARGS() TSRMLS_CC, “s|s”,
                            &name, &name_len, &salutation, &sal_len) == FAILURE) {
    return;
  }

  php_printf(“Hello, %s %s\n”, salutation, name);
}

/*
hello(“Bob”, “Mr.”);
hello(“Alice”, “Dr.”);
hello(“Eve”);

 */
```

Accepting Optional parameters

```
/* proto void hello(string $name[, string $salutation = “Ms.”]) */
PHP_FUNCTION(hello) {
  char *name, *salutation = “Ms.”;
  int name_len, sal_len = strlen(salutation);

  if (zend_parse_parameters(ZEND_NUM_ARGS() TSRMLS_CC, “s|s!”,
                            &name, &name_len, &salutation, &sal_len) == FAILURE) {
    return;
  }

  php_printf(“Hello, %s %s\n”, salutation, name);
}

/*
hello(“Bob”, “Mr.”);
hello(“Alice”, “Dr.”);
hello(“Eve”);
hello(“Janet”, NULL);
 */
```

Accepting Optional parameters

```
/* proto void hello_shout(string $name) */
PHP_FUNCTION(hello_shout) {
  char *name;
  int name_len;

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

  php_strtoupper(name, name_len);
  php_printf(“HELLO, “);
  PHPWRITE(name, name_len);
  php_printf(“\n”);
}
```

Accepting Optional parameters

```
/* proto void hello_opt(string $name[, Array $options = NULL[, $goodbye = false]]) */
PHP_FUNCTION(hello_opt) {
  zend_bool goodbye = 0, shout = 0;
  char *name;
  int name_len;
  zval *zoptions = NULL, **zshout;

  if (zend_parse_parameters(ZEND_NUM_ARGS() TSRMLS_CC, "s/|a!b",
                            &name, &name_len, &zoptions, &goodbye) == FAILURE) {
    return;
  }
  if (zoptions &&
      (zend_hash_find(Z_ARRVAL_P(zarray), "shout", sizeof("shout"), (void**)&zshout)
                                                                    == SUCCESS) &&
      zend_is_true(*zshout)) {
    php_strtoupper(name, name_len);
    shout = 1;
  }
  php_printf("%s %s\n",
             shout ? (goodbye?"GOODBYE":"HELLO") : (goodbye?"Goodbye":"Hello"),
             name);
}
```

PHP Variables

- All types represented by zval
- A given value may be referenced many times
- References may be full: $a =& $b;
- Or copy-on-write: $a = $b;

```
struct {
  union {
    long lval; /* ints, bools, resources */
    double dval;
    struct {
      char *val;
      int len;
    } str;
    HashTable *ht; /* array type */
    zend_object_value obj;
  } value;

  zend_uint refcount__gc;
  zend_uchar type; /* IS_NULL, IS_BOOL, … */
  zend_uchar is_ref__gc;
} zval;
```

Zval Accessors (Reading)

- Readability
- Consistency between versions
- Z_*(v) act on zval
- Z_*_P(pv) act on zval*
  - HASH_OF(pv) also acts on a zval*
- Z_*_PP(ppv) act on zval**
- See Also: Zend/zend_operators.h

Type	Main Macro	Z_TYPE(v)
		IS_NULL
zend_uchar	Z_BVAL(v)	IS_BOOL
long	Z_LVAL(v)	IS_LONG
double	Z_DVAL(v)	IS_DOUBLE
char*	Z_STRVAL(v)	IS_STRING
int	Z_STRLEN(v)	IS_STRING
long	Z_RESVAL(v)	IS_RESOURCE
HashTable*	Z_ARRVAL(v)	IS_ARRAY
HashTable*	HASH_OF(pv)	IS_ARRAY or IS_OBJECT
zend_class_entry*	Z_OBJCE(v)	IS_OBJECT
Many many others...		IS_OBJECT

Type	Main Macro	Meaning
zend_uchar	Z_ISREF(v)	Full-Ref / Copy-on-write
zend_uint	Z_REFCOUNT(v)	Number of references

Accepting Overloaded Parameter

```
/* proto void hello_thing(mixed $thing) */
PHP_FUNCTION(hello_thing) {
  zval *thing;

  if (zend_parse_parameters(ZEND_NUM_ARGS() TSRMLS_CC, "z", &thing) == FAILURE) {
    return;
  }

  switch (Z_TYPE_P(thing)) {
    case IS_NULL:     php_printf("Hello nothing.\n"); break;
    case IS_BOOL:     php_printf("Hello %s\n", Z_BVAL_P(thing) ? "truth" : "lies"); break;
    case IS_LONG:     php_printf("Hello number %ld\n", Z_LVAL_P(thing)); break;
    case IS_DOUBLE:   php_printf("Hello number %f\n", Z_DVAL_P(thing)); break;
    case IS_STRING:   php_printf("Hello '%s'\n", Z_STRVAL_P(thing)); break;
    case IS_ARRAY:    php_printf("Hello array of %d elements\n",
                              zend_hash_num_elements(Z_ARRVAL_P(thing))); break;
    case IS_RESOURCE: php_printf("Hello resource #%ld\n", Z_RESVAL_P(thing)); break;
    case IS_OBJECT:   php_printf("Hello object!\n"); break;
    default:
      php_printf("I don't know what you are…\n");
  }
}
```

Accepting Variable parameters

```
/* proto void hello_many(bool hi, …) */
PHP_FUNCTION(hello_many) {
  zend_bool hi;
  zval ***names;
  int names_count, i;

  if (zend_parse_parameters(ZEND_NUM_ARGS() TSRMLS_CC, “b+”,
                            &hi, &names, &names_count) == FAILURE) {
    return;
  }

  for(i = 0; i < names_count; i++) {
    zval copyval;
    ZVAL_ZVAL(&copyval, *names[i], 1, 0);   // Duplicate stack param
    convert_to_string(&copyval);            // Convert to a string
    php_printf(“%s, %s\n”,  hi ? “Hello” : “Goodbye”, Z_STRVAL(copyval));
    zval_dtor(&copyval);                    // Get rid of temporary copy
  }
  efree(names);                  // Allocated by varargs + (or *) zpp specifier
}
```

PHP is “Shared Noting”

- Per-request memory pool: emalloc()
- Automatically freed at request end

  --enable-debug to be notified of leaks

- Issues E_ERROR on OOM condition
- Does not handle proper cleanup
- See Also: Zend/zend_alloc.h

`void *emalloc(size_t bytes)`
`void efree(void *ptr);`
	`char *str =`
	`emalloc(strlen(“Hi!”)+1);`
	`memcpy(str, “Hi!”,`
	`strlen(“Hi!”)+1);`
	`efree(str);`
		Basic allocate/free analogues of malloc() and free().

`char *estrdup(const char *str);`
`char *estrndup(const char *str,`
`size_t len);`
	`char *str = estrdup(“Hi!”);`
	`php_printf(“%s world!”, str);`
	`efree(str);`
		strdup()/strndup() analogues. Suitable for passing into ZVAL_STRING() as well.

`void *safe_emalloc(size_t nmemb,`
`size_t size,`
`size_t offset);`
	`void *ptr = safe_emalloc(len,`
	`sizeof(UChar),`
	`sizeof(UChar));`
		Performs ((nmemb*size)+offset) with additional boundary and overflow checks.

Zval Accessors (Writing)

- ZVAL_*(pv, V) act on zval*
- RETVAL_*(V) act on function scope zval*: return_value
- RETURN_*(V) set return_value and return from function

Main Macro	Meaning
Z_SET_ISREF(v)	Make full-reference
Z_UNSET_ISREF(v)	Make copy-on-write ref
Z_SET_ISREF_TO(v,ir)	Set/Clear isref
Z_SET_REFCOUNT(v,rc)	Set refcount to rc
INIT_PZVAL(pzv)	IsRef==0, Refcount==1

Macro / Function	Description
ZVAL_NULL(pv)	Sets type to IS_NULL
ZVAL_BOOL(pv, bval)	IS_BOOL, true/false
ZVAL_TRUE(pz)	True
ZVAL_FALSE(pz)	False
ZVAL_LONG(pv, lval)	IS_LONG, long value
ZVAL_DOUBLE(pv, dval)	IS_DOUBLE, double value
ZVAL_STRING(pv, str, cpy)	IS_STRING, null terminated cpy==0 - take existing emalloc cpy==1 - estrdup(str)
ZVAL_STRINGL(pv, str, len, cpy)	Like above, but length specified
ZVAL_RESOURCE(pv, resval)	IS_RESOURCE, long value
array_init(pv)	IS_ARRAY, initially empty
add_index_T(pv, idx, V)	Add values to an array, where T is: null, long, bool, double, string, stringl, or zval. V is appropriate value type.
add_next_index_T(pv, V)	
add_assoc_T(pv, key, V)	

Returning values

- Every function has local zval *return_value
- RETURN_*() macros populate and return immediately
- RETVAL_*() to only populate without returning
  - RETVAL_*() just wraps ZVAL_*() with “return_value” as zval to be populated

```
/* proto void hello_add(int $a, int $b) */
PHP_FUNCTION(hello_add) {
  long a, b;

  if (zend_parse_parameters(ZEND_NUM_ARGS()
           TSRMLS_CC, “ll”, &a, &b)
           == FAILURE) {
    return;
  }

  RETURN_LONG(a + b);

  /* or: RETVAL_LONG(a + b);
         return; */

  /* or: ZVAL_LONG(return_value, a + b);
         return; */
}
```

Returning Strings

- char* emalloc'd and freeable
- Take ownership of new string
- Automatically duplicate string

```
/* proto string hello_title(string $name) */
PHP_FUNCTION(hello_title) {
  char *name, *title;
  int name_len;

  if (zend_parse_parameters(ZEND_NUM_ARGS()
           TSRMLS_CC, "s", &name, &name_len)
           == FAILURE) {
    return;
  }
  /* Crashes:
     RETURN_STRINGL(name, name_len, 0); */

   title = emalloc(name_len + 6);
   memcpy(title, "Boss ", strlen("Boss ");
   memcpy(title + strlen("Boss "),
                 name, name_len + 1);

  /* Leaks:
     RETURN_STRING(title, 1); */
  RETURN_STRING(title, 0);
}
```

Returning Strings

- spprintf() emallocs space for snprintf() output.
- Take ownership of new string
- Automatically duplicate string

```
/* proto string hello_title(string $name) */
PHP_FUNCTION(hello_title) {
  char *name, *title;
  int name_len, title_len;

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



  title_len = spprintf(&title, 0,
                       “Boss %s”, name);




  RETURN_STRINGL(title, title_len, 0);
}
```

Returning Arrays (Zend/zend_API.h)

```
/* proto Array hello_enum_greetings() */
PHP_FUNCTION(hello_enum_greetings) {
  zval *spanish_greetings;
  array_init(return_value);    // Initialize rv as empty array

  add_index_string(return_value, 0,   “Hello”, 1);     // $ret[0]   = “Hello”;
  add_next_index_string(return_value, “Hola”, 1);      // $ret[]    = “Hola”;
  add_assoc_string(return_value, “2”, “Bonjour”, 1);   // $ret[“2”] = “Bonjour”;

  add_next_index_null(return_value);
  add_next_index_bool(return_value, 1);
  add_next_index_long(return_value, 42);
  add_next_index_double(return_value, 3.1415926535);
  add_next_index_stringl(return_value, “Konichiwa”, strlen(“Konichiwa”), 1);

  MAKE_STD_ZVAL(spanish_greetings);
  array_init(spanish_greetings);                                  // $sg = array();
  add_next_index_string(spanish_greetings, “Hola”, 1);        // $sg[] = “Hola”;
  add_next_index_string(spanish_greetings, “Saludos”, 1);     // $sg[] = “Saludos”;
  add_next_index_string(spanish_greetings, “¿Qué Tal?”, 1);   // $sg[] = “¿Qué Tal?”;
  add_assoc_zval(return_value, “spanish”, spanish_greetings); // $ret[“spanish”] = $sg;
}
```

Examining Arrays (Zend/zend_hash.h)

```
/* proto void hello_array(Array $arr) */
PHP_FUNCTION(hello_array) {
  HashTable *arr;
  zval **ppzval;
  char *key = “123”;

  if (zend_parse_parameters(ZEND_NUM_ARGS() TSRMLS_CC, “h”, &arr) == FAILURE) {
    return;
  }

  if (zend_hash_exists(arr, “key”, sizeof(“key”))) {
    php_printf(“$arr[‘key’] is set\n”);
  }
  if (zend_hash_find(arr, “key”, sizeof(“key”), (void**)&ppzval) == SUCCESS) {
    php_printf(“$arr[‘key’] is %s\n”,
               (Z_TYPE_PP(ppzval) == IS_STRING) ? Z_STRVAL_PP(ppzval) : “not a string”);
  }
  if (zend_symtable_exists(arr, key, strlen(key) + 1)) {
    php_printf(“$arr[‘%s’] is set\n”, key);
  }
  // zend_hash_index_exists(arr, 123) zend_hash_index_find(arr, 123, (void**)&ppzval)
}
```

Module Hooks

zend_module_entry

- Module Init/Shutdown
  - Executed when module is loaded/released
  - Registration/Cleanup of non-functions
    - Constants
    - INI Settings
    - Resources
    - Classes
    - Zend callbacks

```
zend_module_entry hello_module_entry = {
    STANDARD_MODULE_HEADER,


    "hello", /* extension name */
    hello_functions, /* global functions */
    NULL, /* Module Init */
    NULL, /* Module Shutdown */
    NULL, /* Request Init */
    NULL, /* Request Shutdown */
    NULL, /* Module Info */
    HELLO_MODULE_VERSION, /* Informational */



    STANDARD_MODULE_PROPERTIES
};
```

zend_module_entry

- Request Init/Shutdown
  - Executed for EVERY page request
  - Pre-processing of request data
    - Filter ext process user input
  - Post-request cleanup
    - Session ext auto-save

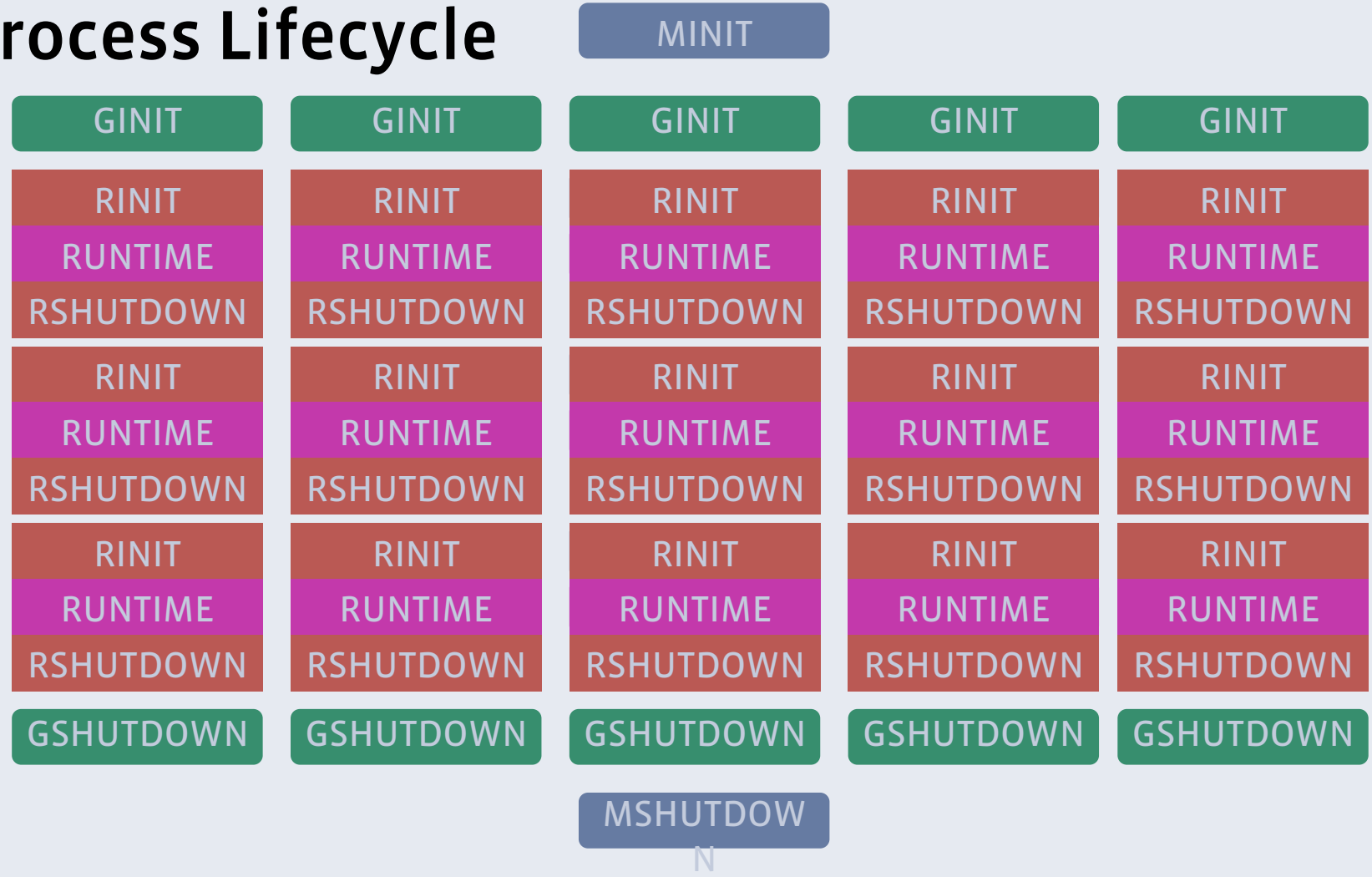
```
zend_module_entry hello_module_entry = {
    STANDARD_MODULE_HEADER,


    "hello", /* extension name */
    hello_functions, /* global functions */
    NULL, /* Module Init */
    NULL, /* Module Shutdown */
    NULL, /* Request Init */
    NULL, /* Request Shutdown */
    NULL, /* Module Info */
    HELLO_MODULE_VERSION, /* Informational */



    STANDARD_MODULE_PROPERTIES
};
```

zend_module_entry (extended)

- INI declaration in module entry
- Named module dependencies
- Per-thread storage pool decl

```
zend_module_entry hello_module_entry = {
    STANDARD_MODULE_HEADER_EX,
    NULL, /* ini entries */
    NULL, /* dependencies */
    "hello", /* extension name */
    hello_functions, /* global functions */
    NULL, /* Module Init */
    NULL, /* Module Shutdown */
    NULL, /* Request Init */
    NULL, /* Request Shutdown */
    NULL, /* Module Info */
    HELLO_MODULE_VERSION, /* Informational */
    0, NULL, /* Thread Globals Size/Var */
    NULL, /* Thread Init */
    NULL, /* Thread Shutdown */
    NULL, /* Post Request Shutdown */
    STANDARD_MODULE_PROPERTIES_EX
};
```

Process Lifecycle



Thread Globals

- Each thread has it's own globals
- GINIT/GSHUTDOWN fired on thread Start/End
- *G() macro for easy access

```
ZEND_BEGIN_MODULE_GLOBALS(hello)
    void *data;
    long counter;
    zend_bool polite;
ZEND_END_MODULE_GLOBALS(hello)
#ifdef ZTS
# define HELLOG(v) TSRMG(hello_globals_id, \
                       zend_hello_globals*, v)
#else
# define HELLOG(v) (hello_globals.v)
#endif
```

```
ZEND_DECLARE_MODULE_GLOBALS(hello)
PHP_FUNCTION(hello_thread) {
  php_printf("Hello #%ld\n",
             ++HELLOG(counter));
}
PHP_GINIT_FUNCTION(hello) {
    hello_globals->data = libdata_init();
    hello_globals->counter = 0;
    hello_globals->polite = 1;
}
zend_module_entry hello_module_entry = {
    STANDARD_MODULE_HEADER,
    "hello", /* extension name */
    hello_functions, /* global functions */
    NULL, NULL, NULL, NULL, NULL
    HELLO_MODULE_VERSION,
    ZEND_MODULE_GLOBALS(hello),
    PHP_GINIT(hello),
    NULL, /* Globals Shutdown */
    NULL, /* Request Post-Shutdown */
    STANDARD_MODULE_PROPERTIES_EX
};
```

INI Settings (Unbound)

- Simple to set up and use
- Require array lookup each time
- Slow for non-string types
  Conversion done on each use
- No set-time validation

```
#include “php_ini.h”
PHP_INI_BEGIN()
  PHP_INI_ENTRY(“hello.polite”, “1”,
                PHP_INI_SYSTEM|PHP_INI_PERDIR,
                NULL)

PHP_INI_END()
PHP_FUNCTION(hello_ini) {
  if (INI_BOOL(“hello.polite”)) {
    php_print(“Good day to you!\n”);
  } else {
    php_printf(“What do you want?\n”);
  }
}
PHP_MINIT_FUNCTION(hello) {
  REGISTER_INI_ENTRIES();
  return SUCCESS;
}
PHP_MSHUTDOWN_FUNCTION(hello) {
  UNREGISTER_INI_ENTRIES();
  return SUCCESS;
}
```

INI Settings (Bound)

- Validation/Conversion at set time
- Quick lookup/use at runtime
- Basic set-time validation
- Can define custom set callbacks

```
#include "php_ini.h"
PHP_INI_BEGIN()
  STD_PHP_INI_BOOLEAN("hello.polite", "1",
                PHP_INI_ALL, OnUpdateBool,
                polite, zend_hello_globals,
                hello_globals);
PHP_INI_END()
PHP_FUNCTION(hello_ini) {
  if (HELLOG(polite)) {
    php_print("Good day to you!\n");
  } else {
    php_printf("What do you want?\n");
  }
}
PHP_MINIT_FUNCTION(hello) {
  REGISTER_INI_ENTRIES();
  return SUCCESS;
}
PHP_MSHUTDOWN_FUNCTION(hello) {
  UNREGISTER_INI_ENTRIES();
  return SUCCESS;
}
```

Declaring Constants

- Global Namespace
  - REGISTER_LONG_CONSTANT
  - REGISTER_DOUBLE_CONSTANT
  - REGISTER_STRING_CONSTANT
  - REGISTER_STRINGL_CONSTANT
- Namespaced Constants
  - REGISTER_NS_LONG_CONSTANT
  - REGISTER_NS_DOUBLE_CONSTANT
  - REGISTER_NS_STRING_CONSTANT
  - REGISTER_NS_STRINGL_CONSTANT

```
PHP_MINIT_FUNCTION(hello) {
    REGISTER_LONG_CONSTANT(“HELLO_LIFE”,
                           42,
                           CONST_PERSISTENT |
                           CONST_CS);
    REGISTER_NS_DOUBLE_CONSTANT(“Hello”, “PI”,
                           3.1415926535,
                           CONST_PERSISTENT);
    return SUCCESS;
}
zend_module_entry hello_module_entry = {
    STANDARD_MODULE_HEADER,
    ”hello", /* extension name */
    hello_functions, /* global functions */
    PHP_MINIT(hello),
    NULL, /* Module Shutdown */
    NULL, /* Request Init */
    NULL, /* Request Shutdown */
    NULL, /* Module Info */
    HELLO_MODULE_VERSION, /* Informational */
    STANDARD_MODULE_PROPERTIES
};
```

Objects and Classes

Declaring Classes

```
zend_class_entry *php_hello_ce;

PHP_MINIT_FUNCTION(hello) {
  zend_class_entry ce;

  INIT_CLASS_ENTRY(ce, “Hello”, NULL);
  php_hello_ce     = zend_register_internal_class(&ce TSRMLS_CC);

  return SUCCESS;
}

/*
Class Hello {
}
*/
```

Declaring Classes (In Namespaces)

```
Zend_class_entry *php_hello_one_ce;

PHP_MINIT_FUNCTION(hello) {
  zend_class_entry ce;

  INIT_CLASS_ENTRY(ce, ZEND_NS_NAME(“Hello”, “One”), NULL);
  php_hello_one_ce = zend_register_internal_class(&ce TSRMLS_CC);

  return SUCCESS;
}

/*
Namespace Hello {
  Class One {
  }
}
*/
```

Declaring Methods

```
zend_class_entry *php_hello_one_ce;

PHP_METHOD(Hello, foo) {
  RETURN_STRING(“bar”, 1)
}

zend_function_entry hello_class_methods[] = {

  PHP_ME(Hello, foo, NULL /* arginfo */, ZEND_ACC_PUBLIC)



  { NULL, NULL, NULL }
};
PHP_MINIT_FUNCTION(hello) {
  zend_class_entry ce;

  INIT_CLASS_ENTRY(ce, “Hello”, hello_class_methods);
  php_hello_one_ce = zend_register_internal_class(&ce TSRMLS_CC);

  return SUCCESS;
}
```

Declaring Methods (Constructor)

```
zend_class_entry *php_hello_one_ce;

PHP_METHOD(Hello, __construct) {
  /* Setup */
}

zend_function_entry hello_class_methods[] = {
  PHP_ME(Hello, __construct, NULL /* arginfo */, ZEND_ACC_PUBLIC | ZEND_ACC_CTOR)
  PHP_ME(Hello, foo,         NULL /* arginfo */, ZEND_ACC_PUBLIC)



  { NULL, NULL, NULL }
};
PHP_MINIT_FUNCTION(hello) {
  zend_class_entry ce;

  INIT_CLASS_ENTRY(ce, “Hello”, hello_class_methods);
  php_hello_one_ce = zend_register_internal_class(&ce TSRMLS_CC);

  return SUCCESS;
}
```

Declaring Methods (Destructor)

```
zend_class_entry *php_hello_one_ce;

PHP_METHOD(Hello, __destruct) {
  /* Cleanup */
}

zend_function_entry hello_class_methods[] = {
  PHP_ME(Hello, __construct, NULL /* arginfo */, ZEND_ACC_PUBLIC | ZEND_ACC_CTOR)
  PHP_ME(Hello, foo,         NULL /* arginfo */, ZEND_ACC_PUBLIC)
  PHP_ME(Hello, __get,       NULL /* arginfo */, ZEND_ACC_PUBLIC)
  PHP_ME(Hello, sFoo,        NULL /* arginfo */, ZEND_ACC_PUBLIC | ZEND_ACC_STATIC)
  PHP_ME(Hello, __destruct,  NULL /* arginfo */, ZEND_ACC_PUBLIC | ZEND_ACC_DTOR)
  { NULL, NULL, NULL }
};
PHP_MINIT_FUNCTION(hello) {
  zend_class_entry ce;

  INIT_CLASS_ENTRY(ce, “Hello”, hello_class_methods);
  php_hello_one_ce = zend_register_internal_class(&ce TSRMLS_CC);

  return SUCCESS;
}
```

Declaring Properties & Constants

```
zend_class_entry *php_hello_one_ce;

PHP_MINIT_FUNCTION(hello) {
  zend_class_entry ce;

  INIT_CLASS_ENTRY(ce, “Hello”, hello_class_methods);
  php_hello_one_ce = zend_register_internal_class(&ce TSRMLS_CC);

  zend_declare_property_null(php_hello_one_ce, “nulp”, strlen(“nulp”),
                             ZEND_ACC_PUBLIC TSRMLS_CC);
  zend_declare_property_long(php_hello_one_ce, “life”, strlen(“life”), 42,
                             ZEND_ACC_PRIVATE | ZEND_ACC_STATIC TSRMLS_CC);
  zend_declare_property_string(php_hello_one_ce, “name”, strlen(“name”), “bob”,
                             ZEND_ACC_PROTECTED TSRMLS_CC);

  zend_declare_class_constant_double(php_hello_one_ce,
                                     “PI”, strlen(“PI”), 3.1415926535 TSRMLS_CC);

  return SUCCESS;
}
```

Fetching and Updating properties

```
zend_class_entry *php_hello_one_ce;

PHP_METHOD(One, getName) {
  zval *obj = getThis();
  zval *name = zend_read_property(php_hello_one_ce, obj,
                                  “name”, strlen(“name”), 0 TSRMLS_CC);

  ZVAL_ZVAL(return_value, name, 1, 0);
}

PHP_METHOD(One, setName) {
  zval *obj = getThis();
  char *name;
  int name_len;

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

  zend_update_property_stringl(php_hello_one_ce, obj, “name”, strlen(“name”),
                               name, name_len TSRMLS_CC);
}
```

Custom PHP Objects

- Arbitrary object storage
- Handle all object actions and refence counting
- More than “arrays with funcs”

```
typedef struct _php_hello_object {
  /* “Inherit” normal object structure */
  zend_object obj;


  /* Additional fields local to us */
  char *name;
  char *salutation;
} php_hello_object;
```

Custom Classes (Creation/Destruction)

```
void hello_object_destroy(php_hello_object *objval TSRMLS_DC) {
  if (objval->name) efree(objval->name);
  if (objval->salutation) efree(objval->salutation);
  zend_object_std_dtor(&(objval->obj) TSRMLS_CC);
  efree(objval);
}
zend_object_value hello_object_create(zend_class_entry *ce TSRMLS_DC) {
  zend_object_value ret;
  php_hello_object *objval = ecalloc(1, sizeof(php_hello_object));
  objval->obj.ce = ce;
  object_properties_init(&(objval->obj), ce);
  ret.handle = zend_objects_store_put(objval, NULL,
                  (zend_objects_free_object_storage_t)hello_object_destroy, NULL TSRMLS_CC);
  ret.handlers = zend_get_std_object_handlers();
  return ret;
}
PHP_MINIT_FUNCTION(hello) {
  zend_class_entry ce;
  INIT_CLASS_ENTRY(ce, “Hello”, hello_class_methods);
  php_hello_one_ce = zend_register_internal_class(&ce TSRMLS_CC);
  php_hello_one_ce->create_object = hello_object_create;
  return SUCCESS;
}
```

Custom Classes (ObjVal Lookup)

```
/* proto void One::__construct(string $name[, string $salutation = “Ms.”]) */
PHP_METHOD(One, __construct) {
  php_hello_object *objval =(php_hello_object*)zend_objects_get_address(getThis() TSRMLS_CC);
  char *name, *salutation = “Ms.”;
  int name_len, sal_len = strlen(salutation);

  if (zend_parse_parameters(ZEND_NUM_ARGS() TSRMLS_CC, “s|s”,
                            &name, &name_len, &salutation, &sal_len) == FAILURE) {
    return;
  }
  objval->name = estrndup(name, name_len);
  objval->salutation = estrndup(salutation, sal_len);
}
PHP_METHOD(One, setName) {
  php_hello_object *objval =(php_hello_object*)zend_objects_get_address(getThis() TSRMLS_CC);
  char *name;
  int name_len;
  if (zend_parse_parameters(ZEND_NUM_ARGS() TSRMLS_CC, “s”, &name, &name_len) == FAILURE) {
    return;
  }
  if (objval->name) efree(objval->name);
  objval->name = estrndup(name, name_len);
}
```

Custom Classes

```
zend_class_entry *php_hello_one_ce;
zend_object_handlers php_hello_one_handlers;

zend_object_value hello_object_create(zend_class_entry *ce TSRMLS_DC) {
  /* yada yada yada… */
  ret.handlers = &php_hello_one_handlers;
  return ret;
}

PHP_MINIT_FUNCTION(hello) {
  zend_class_entry ce;

  INIT_CLASS_ENTRY(ce, “Hello”, hello_class_methods);
  php_hello_one_ce = zend_register_internal_class(&ce TSRMLS_CC);
  php_hello_one_ce->create_object = hello_object_create;

  memcpy(&php_hello_one_handlers,
         zend_get_std_object_handlers(), sizeof(zend_object_handlers));
  php_hello_one_handlers.read_property  = hello_object_read_prop;
  php_hello_one_handlers.read_dimension = hello_object_read_dim;
  return SUCCESS;
}
```

Linking External Libraries

Back to config.m4

- --with versus –enable: convention
- --with-howdy=[DIR] $PHP_HOWDY
- Use header location as hint

```
dnl config.m4
PHP_ARG_WITH(hello,
               whether to enable hello module,
[ --with-hello   Enable hello support ])

if test “$PHP_HELLO” != “no”; then
  PHP_NEW_EXTENSION(hello,
                    hello.c,
                    $ext_shared)
  for i in /usr/local /usr $PHP_HELLO; do
    if test –f $i/include/howdy.h; then
      HOWDY_DIR=$i
    fi
  done
  if –z $HOWDY_DIR; then
    AC_MSG_ERROR(Cannot find libhowdy)
  fi
  PHP_ADD_LIBPATH($HOWDY_DIR/lib,
                  HELLO_SHARED_LIBADD)
  PHP_ADD_INCLUDE($HOWDY_DIR/include)
  PHP_ADD_LIBRARY(howdy,, HELLO_SHARED_LIBADD)
fi
```

Testing for functionality

- PHP_CHECK_LIBRARY() tests for symbols in .so
- AC_TRY_RUN() will build and run test program

```
AC_MSG_CHECKING([for howdy_init2() in
                    libhowdy])
PHP_CHECK_LIBRARY(howdy, howdy_init2, [
  dnl What to do if we have it
  AC_DEFINE(HAVE_HOWDY_INIT2, 1,)
  AC_MSG_RESULT(found)
],[
  dnl What to do if we don't
  AC_MSG_ERROR([Requires libhowdy
               version >= 2.0.0])
],[
  -L$HOWDY_DIR/lib
])
```

Testing for functionality

- PHP_CHECK_LIBRARY() tests for symbols in .so
- AC_TRY_RUN() will build and run test program

```
  AC_MSG_CHECKING([for typedefs in howdy.h])
  AC_TRY_RUN([
#include <howdy.h>

int main(int argc, char *argv[]) {
  howdy_value val;
  return 0;
}
  ],[
    dnl Build succeeded and exitcode == 0
    AC_MSG_RESULT(yes)
    AC_DEFINE(HAVE_HOWDY_TYPEDEF, “1”,)
  ],[
    dnl exitcode != 0 (won’t happen here)
    AC_MSG_RESULT(no)
  ],[
    dnl Build failed
    AC_MSG_RESULT(no)
  ])
```

Windows is an operating system too

- config.w32, JS based
- Similar to config.m4 for *nix

```
// config.m4
ARG_WITH("hello","enable hello module","no")

if (PHP_HELLO != "no") {
  if (CHECK_LIB("howdy.lib", "howdy",
                              PHP_HELLO) &&
      CHECK_HEADER_ADD_INCLUDE("howdy.h",
            "CFLAGS",
            php_usual_include_suspects)) {
    EXTENSION("hello", "hello.c", true);
  }
}
```

Where to go next

Reading List

- “Extending and Embedding PHP” – Dated, but largely relevant
- http://blog.golemon.com/ - Look in the upper-right corner
- http://github.com/php/php-src - Use the source
- http://svn.php.net/pecl - More useful extensions
  - And the crazy stuff: Runkit, Operator, etc…

PHP Community

- http://php.net/lists
- http://news.php.net
  - php.internals
  - php.pecl.dev
- http://twitter.com/SaraMG