Advanced PHP

linux.conf.au

January 22, 2003. Perth

Rasmus Lerdorf

© Copyright 2003 Rasmus Lerdorf

Safe Mode

Safe Mode is an attempt to solve the shared-server security problem. It is architecturally incorrect to try to solve this problem at the PHP level, but since the alternatives at the web server and OS levels aren't very realistic, many people, especially ISP's, use safe mode for now.

The configuration directives that control safe mode are:

```
safe_mode = Off
open_basedir =
safe_mode_exec_dir =
safe_mode_allowed_env_vars = PHP_
safe_mode_protected_env_vars = LD_LIBRARY_PATH
disable_functions =
```

When safe_mode is on, PHP checks to see if the owner of the current script matches the owner of the file to be operated on by a file function.

For example:

```
-rw-rw-r--    1 rasmus   rasmus       33 Jul  1 19:20 script.php
-rw-r--r--    1 root     root       1116 May 26 18:01 /etc/passwd
```

Running this script.php

```
<?php
readfile('/etc/passwd');
?>
```

results in this error when safe mode is enabled:

```
<b>Warning</b>:  SAFE MODE Restriction in effect.  The script whose uid is 500
is not allowed to access /etc/passwd owned by uid 0 in
<b>/docroot/script.php</b> on line <b>2</b>
```

If instead of safe_mode, you set an open_basedir directory then all file operations will be limited to files under the specified directory. For example (Apache httpd.conf example):

```
<Directory /docroot>
php_admin_value open_basedir /docroot
</Directory>
```

If you run the same script.php with this open_basedir setting then this is the result:

```
<b>Warning</b>: open_basedir restriction in effect. File is in wrong directory
in
<b>/docroot/script.php</b> on line <b>2</b>
```

You can also disable individual functions. If we add this to our php.ini file:

```
disable_functions readfile,system
```

Then we get this output:

```
<b>Warning</b>:  readfile() has been disabled for security reasons in
<b>/docroot/script.php</b> on line <b>2</b>
```

Security

Watch for uninitialized variables

```
<?php
    if($user=='rasmus') {
        $ok = true;
    }

    if($ok) {
        echo "$user logged in";
    }
?>
```

Catch these by setting the error_reporting level to E_ALL. The above script would generate this warning (assuming $user is set):

```
<b>Warning</b>:  Undefined variable:  ok in <b>script.php</b> on line <b>6</b>
```

You can of course also turn off register_globals, but that addresses the symptom rather than the problem.

Security

Never trust user data!

```
<?php
    readfile($filename);
?>
```

Turning off register_globals doesn't make this any more secure. The script would instead look like this:

```
<?php
    readfile($HTTP_POST_VARS['filename']);
?>
```

The only way to secure something like this is to be really paranoid about cleaning user input. In this case if you really want the user to be able to specify a filename that gets used in any of PHP's file functions, do something like this:

```
<?php
    $doc_root = $HTTP_SERVER_VARS['DOCUMENT_ROOT'];
    $filename = realpath($filename);
    readfile($doc_root.$filename);
?>
```

You may also want to strip out any path and only take the filename component. An easy way to do that is to use the basename() function. Or perhaps check the extension of the file. You can get the extension using this code:

```
<?php
    $ext = substr($str,strrpos($str,'.'));
?>
```

Security

Again, never trust user data!

```
<?php
    system("ls $dir");
?>
```

In this example you want to make sure that the user can't pass in $dir set to something like: ".;cat /etc/passwd" The remedy is to use escapeshellarg() which places the argument inside single quotes and escapes any single quote characters in the string.

```
<?php
    $dir=escapeshellarg($dir);
    system("ls $dir");
?>
```

Beyond making sure users can't pass in arguments that executes other system calls, make sure that the argument itself is ok and only accesses data you want the users to have access to.

Security

Many users place code in multiple files and include these files:

```
<?php
    require 'functions.inc';
?>
```

Or perhaps

```
<?php
    require 'functions.php';
?>
```

Both of these can be problematic if the included file is accessible somewhere under the DOCUMENT_ROOT directory. The best solution is to place these files outside of the DOCUMENT_ROOT directory where they are not accessible directly. You can add this external directory to your include_path configuration setting.

Another option is to reject any direct requests for these files in your Apache configuration. You can use a rule like this in your "httpd.conf" file:

```
<Files ~ "\.inc$">
    Order allow,deny
    Deny from all
</Files>
```

Security

Take this standard file upload form:

```
<FORM ENCTYPE="multipart/form-data" ACTION="upload.php" METHOD=POST>
<INPUT TYPE="hidden" name="MAX_FILE_SIZE" value="100000">
Send this file: <INPUT NAME="myfile" TYPE="file">
<INPUT TYPE="submit" VALUE="Send File">
</FORM>
```

The correct way to put the uploaded file in the right place:

```
<?php
    / Not under DOCUMENT_ROOT /
    $destination = "/some/path/$myfile_name";

    move_uploaded_file($myfile, $destination);
?>
```

If you are uploading files to be placed somewhere under the DOCUMENT_ROOT then you need to be very paranoid in checking what you are putting there. For example, you wouldn't want to let people upload arbitrary PHP scripts that they can then browse to in order to execute them. Here we get paranoid about checking that only image files can be uploaded. We even look at the contents of the file and ensure that the file extension matches the content.

```
<?php
    $type = $HTTP_POST_FILES['myfile']['type'];
    $file = $HTTP_POST_FILES['myfile']['tmp_name'];
    $name = $HTTP_POST_FILES['myfile']['name'];
    $types = array(0,'.gif','.jpg','.png','.swf');
    list(,,$type) = getimagesize($file);
    if($type) {
        $name = substr($name,0,strrpos($str,'.'));
        $name .= $types[$type];
    }
    move_uploaded_file($myfile, "$DOCUMENT_ROOT/images/$name");
?>
```

HTTP

Client/Server Request/Response

HTTP is a simple client/server protocol with stateless request/response sequences.

The Client HTTP Request

7 possible HTTP 1.1 request types: GET, PUT, POST, DELETE, HEAD, OPTIONS and TRACE. Any number of HTTP headers can accompany a request.

```
GET /filename.php HTTP/1.0
Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, image/png, /
Accept-Charset: iso-8859-1,*,utf-8
Accept-Encoding: gzip
Accept-Language: en
Connection: Keep-Alive
Host: localhost
User-Agent: Mozilla/4.77 [en] (X11; U; Linux 2.4.5-pre4 i686; Nav)
```

The Server HTTP Response

```
HTTP/1.1 200 OK
Date: Mon, 21 May 2001 17:01:51 GMT
Server: Apache/1.3.20-dev (Unix) PHP/4.0.7-dev
Last-Modified: Fri, 26 Jan 2001 06:08:38 GMT
ETag: "503d3-50-3a711466"
Accept-Ranges: bytes
Content-Length: 80
Keep-Alive: timeout=15, max=100
Connection: Keep-Alive
Content-Type: text/html
```

Keep-Alive

When a keep-alive request is granted the established socket is kept open after each keep-alive response. Note that a keep-alive response is only possible when the response includes a content-length header.

```
request 1
request 2
request 3
request 4
```

```
20 bytes
120 bytes
60 bytes
?? bytes
```

You cannot rely on the keep-alive feature for any sort of application-level session state maintenance.

Using Output Buffering to get content-length

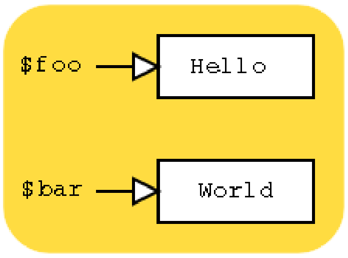
```
<?php
    ob_start();
    echo "Your Data";
    $l = ob_get_length();
    Header("Content-length: $l");
    ob_end_flush();
?>
```

You will have to weigh the trade-off between the extra cpu and memory that output buffering takes against the increased effciency of being able to use keep-alive connections for your dynamic pages.

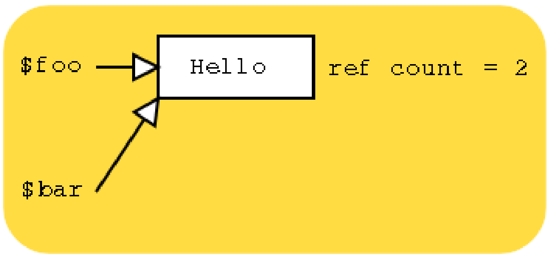
References

References are not pointers!

```
<?php
    $foo = 'Hello';
    $bar = 'World';
?>
```



```
<?php
    $bar = & $foo;
?>
```



Returning References

Passing arguments to a function by reference

```
<?php
function inc(& $b) {
    $b++;
}
$a = 1;
inc($a);
echo $a;
?>
```

Output:

```
2
```

A function may return a reference to data as opposed to a copy

```
<?php
function & get_data() {
    $data = "Hello World";
    return $data;
}
$foo = & get_data();
?>
```

Variable variables

A variable variable looks like this: $$var

So, if $var = 'foo' and $foo = 'bar' then $$var would contain the value 'bar' because $$var can be thought of as $'foo' which is simply $foo which has the value 'bar'.

Variable variables sound like a cryptic a useless concept, but they can be useful sometimes. For example, if we have a configuration file consisting of configuration directives and values in this format:

```
foo=bar
abc=123
```

Then it is very easy to read this file and create corresponding variables:

```
<?php
$fp = fopen('config.txt','r');
while(true) {
    $line = fgets($fp,80);
    if(!feof($fp)) {
        if($line[0]=='#' || strlen($line)<2) continue;
        list($name,$val)=explode('=',$line,2);
        $$name=trim($val);
    } else break;
}
fclose($fp);
?>
```

Along the same lines as variable variables, you can create compound variables and variable functions.

```
<?php
  $str = 'var';
  $var_toaster = "Hello World";
  echo ${$str.'_toaster'};

  $str();  // Calls a function named var()
  ${$str.'_abc'}();  // Calls a function named var_abc()
?>
```

Cookie Expiry

Problem

Short expiry cookies depend on users having their system clocks set correctly.

Solution

Don't depend on the users having their clocks set right. Embed the timeout based on your server's clock in the cookie.

```
<?php
  $value = time()+3600 . ':' . $variable;
  SetCookie('Cookie_Name',$value);
?>
```

Then when you receive the cookie, decode it and determine if it is still valid.

```
<?php
list($ts,$variable) = explode(':',$Cookie_Name,2);
if($ts < time()) {
   ...
} else {
    SetCookie('Cookie_Name','');
}
?>
```

Optimization

Don't use a regex if you don't have to

PHP has a rich set of string manipulation functions - use them!

```
BAD: <?  $new = ereg_replace("-","_",$str); ?>

GOOD:<?  $new = str_replace("-","_",$str);  ?>

BAD: <?  preg_match('/(\..*?)$/',$str,$reg);?>

GOOD:<?  substr($str,strrpos($str,'.'));    ?>
```

Use References if you are passing large data structs around to save memory

There is a tradeoff here. Manipulating references is actually a bit slower than making copies of your data, but with references you will be using less memory. So you need to determine if you are cpu or memory bound to decide whether to go through and look for places to pass references to data instead of copies.

Use Persistent Database connections

Some database are slower than others at establising new connections. The slower it is, the more of an impact using persistent connections will have. But, keep in mind that persistent connections will sit and tie up resources even when not in use. Watch your resource limits as well. For example, by default Apache's

Using MySQL? Check out mysql_unbuffered_query()

Use it exactly like you would mysql_query(). The difference is that instead of waiting for the entire query to finish and storing the result in the client API, an unbuffered query makes results available to you as soon as possible and they are not allocated in the client API. You potentially get access to your data quicker, use a lot less memory, but you can't use mysql_num_rows() on the result resource and it is likely to be slightly slower for small selects.

Hey Einstein!

Don't over-architect things. If your solution seems complex to you, there is probably a simpler and more obvious approach. Take a break from the computer and go out into the big (amazingly realistic) room and think about something else for a bit.

Connection Handling

PHP maintains a connection status bitfield with 3 bits:

- 0 - NORMAL
- 1 - ABORTED
- 2 - TIMEOUT

By default a PHP script is terminated when the connection to the client is broken and the ABORTED bit is turned on. This can be changed using the ignore_user_abort() function. The TIMEOUT bit is set when the script timelimit is exceed. This timelimit can be set using set_time_limit().

```
<?php
set_time_limit(0);
ignore_user_abort(true);
/ code which will always run to completion /
?>
```

You can call connection_status() to check on the status of a connection.

```
<?php
ignore_user_abort(true);
echo "some output";
if(connection_status()==0) {
    // Code that only runs when the connection is still alive
} else {
    // Code that only runs on an abort
}
?>
```

You can also register a function which will be called at the end of the script no matter how the script was terminated.

```
<?php
function foo() {
    if(connection_status() & 1)
        error_log("Connection Aborted",0);
    if(connection_status() & 2)
        error_log("Connection Timed Out",0);
    if(!connection_status())
        error_log("Normal Exit",0);
}
register_shutdown_function('foo');
?>
```

Optimizing MySQL Usage

Check these

- Don't fetch 10,000 rows and only use 5, use a LIMIT clause
- Make sure you have keys on the right columns
- --log-slow-queries[=file_name] can help here
- So can --log-long-format to show non-indexed queries
- Use mysqldumpslow to look at these

A non-indexed query

```
<?
  mysql_connect('localhost');
  mysql_select_db('foo');
  $result = mysql_query("select * from users where ts>0");
?>
```

And an indexed query

```
<?
  $result = mysql_query(
    "select * from users where id='rasmus'");
  echo "<pre>";
  system('mysqldumpslow');
  echo "</pre>\n";
?>
```

Output:

```
Count: 7  Time=0.00s (0s)  Lock=0.00s (0s)  Rows=2.0 (14),
nobody[nobody]@localhost
  select id, email,
  date_format(ts,'S') as d
  from users order by ts

Count: 6  Time=0.00s (0s)  Lock=0.00s (0s)  Rows=2.0 (12),
nobody[nobody]@localhost
  select * from users

Count: 1  Time=0.00s (0s)  Lock=0.00s (0s)  Rows=2.0 (2),
nobody[nobody]@localhost
  select * from comments

Count: 3  Time=0.00s (0s)  Lock=0.00s (0s)  Rows=2.0 (6),
nobody[nobody]@localhost
  select * from users where ts>N

Count: 6  Time=0.00s (0s)  Lock=0.00s (0s)  Rows=2.0 (12),
nobody[nobody]@localhost
  select * from users order by id

Count: 5  Time=0.00s (0s)  Lock=0.00s (0s)  Rows=3.0 (15),
nobody[nobody]@localhost
  select * from comments order by ts desc
```

UDF

User Defined Functions

You can execute PHP code as a user defined function inside MySQL.

Loading the PHP UDF library

```
CREATE FUNCTION php RETURNS STRING SONAME 'myphp.so';
```

Hello World

```
mysql> SELECT php('hello, world!');
+----------------------------------+
| php('hello, world!')             |
+----------------------------------+
| hello, world!                    |
+----------------------------------+
1 row in set (0.36 sec)

mysql> SELECT php('strftime("%c")');
+--------------------------+
| php('strftime("%c")')    |
+--------------------------+
| Thu Oct 24 18:12:04 2002 |
+--------------------------+
1 row in set (0.36 sec)
```

Applying PHP on a selected column

```
mysql> CREATE TABLE test (s varchar(255));
mysql> insert into test values ('iguana'),('turtle'),('aardvark');
mysql> select s,php('strlen($argv[1])',s) FROM test;
+----------+--------------------------+
| s        | php('strlen($argv[1])',s) |
+----------+--------------------------+
| iguana   | 6                        |
| turtle   | 6                        |
| aardvark | 8                        |
+----------+--------------------------+
3 rows in set (1.07 sec)
```

Oh my!

```
mysql> create table code (name varchar(255), code mediumblob);
mysql> insert into code values ('capitalize','strtoupper($argv[1])');
mysql> select php(code,s) FROM test,code WHERE code.name = 'capitalize';
+-------------+
| php(code,s) |
+-------------+
| IGUANA      |
| TURTLE      |
| AARDVARK    |
+-------------+
```

Installing Related Libraries

Before installing or building PHP you will need to install the various libraries you are planning on having PHP use. We will be using the pdflib library later on in this course. Here is how you would download, compile and install this library.

pdflib

```
% wget http://www.pdflib.com/pdflib/download/pdflib-4.0.3.tar.gz
% tar zxvf pdflib-4.0.3.tar.gz
% cd pdflib-4.0.3
% ./configure  --enable-php=no --without-java --without-tcl
% make
% make install
```

But, for the most part you should be able to download the appropriate -devel package from whataver Linux distribution you are on.

Building PHP from source

Grab a tarball (.tar.gz file) from php.net/downloads.php or a CVS snapshot from snaps.php.net.

```
% tar zxvf php-4.3.0.tar.gz
% cd php-4.3.0
% ./configure --help


`configure' configures this package to adapt to many kinds of systems.

Usage: ./configure [OPTION]... [VAR=VALUE]...

To assign environment variables (e.g., CC, CFLAGS...), specify them as
VAR=VALUE.  See below for descriptions of some of the useful variables.

Defaults for the options are specified in brackets.

Configuration:
  -h, --help              display this help and exit
      --help=short        display options specific to this package
      --help=recursive    display the short help of all the included packages
  -V, --version           display version information and exit
  -q, --quiet, --silent   do not print `checking...' messages
      --cache-file=FILE   cache test results in FILE [disabled]
  -C, --config-cache      alias for `--cache-file=config.cache'
  -n, --no-create         do not create output files
      --srcdir=DIR        find the sources in DIR [configure dir or `..']

Installation directories:
  --prefix=PREFIX         install architecture-independent files in PREFIX
                          [/usr/local]
  --exec-prefix=EPREFIX   install architecture-dependent files in EPREFIX
                          [PREFIX]

By default, `make install' will install all the files in
`/usr/local/bin', `/usr/local/lib' etc.  You can specify
an installation prefix other than `/usr/local' using `--prefix',
for instance `--prefix=$HOME'.

For better control, use the options below.

Fine tuning of the installation directories:
  --bindir=DIR           user executables [EPREFIX/bin]
  --sbindir=DIR          system admin executables [EPREFIX/sbin]
  --libexecdir=DIR       program executables [EPREFIX/libexec]
  --datadir=DIR          read-only architecture-independent data [PREFIX/share]
  --sysconfdir=DIR       read-only single-machine data [PREFIX/etc]
  --sharedstatedir=DIR   modifiable architecture-independent data [PREFIX/com]
  --localstatedir=DIR    modifiable single-machine data [PREFIX/var]
  --libdir=DIR           object code libraries [EPREFIX/lib]
  --includedir=DIR       C header files [PREFIX/include]
  --oldincludedir=DIR    C header files for non-gcc [/usr/include]
  --infodir=DIR          info documentation [PREFIX/info]
  --mandir=DIR           man documentation [PREFIX/man]

System types:
  --build=BUILD     configure for building on BUILD [guessed]
  --host=HOST       build programs to run on HOST [BUILD]

Optional Features:
  --disable-FEATURE       do not include FEATURE (same as --enable-FEATURE=no)
  --enable-FEATURE[=ARG]  include FEATURE [ARG=yes]

SAPI modules:

  --disable-cli           Disable building CLI version of PHP
                           (this forces --without-pear).
  --enable-embed=TYPE     Enable building of embedded SAPI library
                           TYPE is either 'shared' or 'static'. TYPE=shared
  --enable-roxen-zts      Build the Roxen module using Zend Thread Safety.
  --disable-cgi           Disable building CGI version of PHP
  --enable-force-cgi-redirect
                          Enable the security check for internal server
                           redirects.  You should use this if you are
                           running the CGI version with Apache.
  --enable-discard-path   If this is enabled, the PHP CGI binary
                           can safely be placed outside of the
                           web tree and people will not be able
                           to circumvent .htaccess security.
  --enable-fastcgi        If this is enabled, the cgi module will
                           be built with support for fastcgi also.
  --disable-path-info-check  If this is disabled, paths such as
                           /info.php/test?a=b will fail to work.

General settings:

  --enable-debug          Compile with debugging symbols.
  --enable-safe-mode      Enable safe mode by default.
  --enable-sigchild       Enable PHP's own SIGCHLD handler.
  --enable-magic-quotes   Enable magic quotes by default.
  --disable-rpath         Disable passing additional runtime library
                          search paths
  --enable-libgcc         Enable explicitly linking against libgcc
  --disable-short-tags    Disable the short-form start tag by default.
  --enable-dmalloc        Enable dmalloc
  --disable-ipv6          Disable IPv6 support

Extensions:

  --with-EXTENSION=shared[,PATH]

    NOTE: Not all extensions can be build as 'shared'.

    Example: --with-foobar=shared,/usr/local/foobar/

      o Builds the foobar extension as shared extension.
      o foobar package install prefix is /usr/local/foobar/

 --disable-all   Disable all extensions enabled by default.
 --enable-all    Enable all extensions.

  --enable-bcmath         Enable bc style precision math functions.
  --enable-calendar       Enable support for calendar conversion
  --disable-ctype         Disable ctype functions
  --enable-dba            Build DBA with builtin modules
  --enable-dbase          Enable the bundled dbase library
  --enable-dbx            Enable dbx
  --enable-dio            Enable direct I/O support
  --enable-exif           Enable EXIF (metadata from images) support
  --enable-filepro        Enable the bundled read-only filePro support.
  --enable-ftp            Enable FTP support
  --enable-gd-native-ttf  GD: Enable TrueType string function.
  --enable-mbstring       Enable multibyte string support
  --disable-mbregex       Disable multibyte regex support
  --enable-mime-magic     Enable mime_magic support
  --disable-overload      Disable user-space object overloading support.
  --enable-pcntl          Enable experimental pcntl support (CLI/CGI only)
  --disable-posix         Disable POSIX-like functions
  --disable-session       Disable session support
  --enable-shmop          Enable shmop support
  --enable-ucd-snmp-hack  Enable UCD SNMP hack
  --enable-soap           Enable soap support
  --enable-sockets        Enable sockets support
  --enable-sysvmsg        Enable sysvmsg support
  --enable-sysvsem        Enable System V semaphore support.
  --enable-sysvshm        Enable the System V shared memory support.
  --disable-tokenizer     Disable tokenizer support
  --enable-wddx           Enable WDDX support.
  --disable-xml           Disable XML support using bundled expat lib
  --enable-xslt           Enable xslt support.
  --enable-yp             Include YP support.

Other settings:

  --enable-versioning     Export only required symbols.
```

```
                          See INSTALL for more information
  --enable-shared=PKGS    build shared libraries default=yes
  --enable-static=PKGS    build static libraries default=yes
  --enable-fast-install=PKGS  optimize for fast installation default=yes
  --disable-libtool-lock  avoid locking (might break parallel builds)
  --enable-experimental-zts   This will most likely break your build
  --disable-inline-optimization If building zend_execute.lo fails, try
                                this switch.
  --enable-memory-limit   Compile with memory limit support.
  --enable-zend-multibyte Compile with zend multibyte support.

Optional Packages:
  --with-PACKAGE[=ARG]    use PACKAGE [ARG=yes]
  --without-PACKAGE       do not use PACKAGE (same as --with-PACKAGE=no)
  --with-aolserver=DIR    Specify path to the installed AOLserver
  --with-apxs=FILE        Build shared Apache 1.x module. FILE is the optional
                           pathname to the Apache apxs tool; defaults to "apxs".
  --with-apache=DIR       Build Apache 1.x module. DIR is the top-level Apache
                           build directory, defaults to /usr/local/apache.
  --with-mod_charset      Enable transfer tables for mod_charset (Rus Apache).
  --with-apxs2=FILE       EXPERIMENTAL: Build shared Apache 2.0 module. FILE is
the optional
                           pathname to the Apache apxs tool; defaults to "apxs".
  --with-apache-hooks=FILE
                          EXPERIMENTAL: Build shared Apache 1.x module. FILE is
the optional
                           pathname to the Apache apxs tool; defaults to "apxs".
  --with-apache-hooks-static=DIR
                          EXPERIMENTAL: Build Apache 1.x module. DIR is the
top-level Apache
                           build directory, defaults to /usr/local/apache.
  --with-caudium=DIR      Build PHP as a Pike module for use with Caudium
                           DIR is the Caudium server dir, with the default value
                           /usr/local/caudium/server.
  --with-fastcgi=SRCDIR   Build PHP as FastCGI application
  --with-isapi=DIR        Build PHP as an ISAPI module for use with Zeus.
  --with-milter=DIR       Build PHP as Milter application
  --with-nsapi=DIR        Build PHP as NSAPI module for use with iPlanet.
  --with-phttpd=DIR       Build PHP as phttpd module
  --with-pi3web=DIR       Build PHP as Pi3Web module
  --with-roxen=DIR        Build PHP as a Pike module. DIR is the base Roxen
                           directory, normally /usr/local/roxen/server.
  --with-servlet=DIR      Include servlet support. DIR is the base install
                           directory for the JSDK.  This SAPI prereqs the
                           java extension must be built as a shared dl.
  --with-thttpd=SRCDIR    Build PHP as thttpd module
  --with-tux=MODULEDIR    Build PHP as a TUX module (Linux only)
  --with-webjames=SRCDIR  Build PHP as a WebJames module (RISC OS only)
  --with-layout=TYPE      Sets how installed files will be laid out.  Type is
                           one of "PHP" (default) or "GNU"
  --with-config-file-path=PATH
                          Sets the path in which to look for php.ini,
                           defaults to PREFIX/lib
  --with-config-file-scan-dir=PATH
  --with-pear=DIR         Install PEAR in DIR (default PREFIX/lib/php)
  --without-pear          Do not install PEAR
  --with-exec-dir=DIR     Only allow executables in DIR when in safe mode
                           defaults to /usr/local/php/bin
  --with-openssl=DIR      Include OpenSSL support (requires OpenSSL >= 0.9.5)
  --with-zlib=DIR         Include ZLIB support (requires zlib >= 1.0.9).
  --with-zlib-dir=<DIR>   Define the location of zlib install directory
  --with-bz2=DIR          Include BZip2 support
  --with-cpdflib=DIR      Include cpdflib support (requires cpdflib >= 2).
  --with-jpeg-dir=DIR     jpeg dir for cpdflib 2.x
  --with-tiff-dir=DIR     tiff dir for cpdflib 2.x
  --with-crack=DIR        Include crack support.
  --with-curl=DIR         Include CURL support
  --with-curlwrappers     Use CURL for url streams
  --with-cyrus            Include Cyrus IMAP support
  --with-db               Include old xDBM support (deprecated use --with-dba)
  --with-gdbm=DIR         Include GDBM support
  --with-ndbm=DIR         Include NDBM support
  --with-db4=DIR          Include Berkeley DB4 support
  --with-db3=DIR          Include Berkeley DB3 support
  --with-db2=DIR          Include Berkeley DB2 support
  --with-dbm=DIR          Include DBM support
  --with-cdb=DIR          Include CDB support
  --with-flatfile         Include FlatFile support

  --with-dom=DIR          Include DOM support (requires libxml >= 2.4.14).
                           DIR is the libxml install directory.
  --with-zlib-dir=DIR     DOMXML: Set the path to libz install prefix.
  --with-dom-xslt=DIR     DOMXML: Include DOM XSLT support (requires libxslt >=
1.0.18).
                           DIR is the libxslt install directory.
  --with-dom-exslt=DIR    DOMXML: Include DOM EXSLT support (requires libxslt >=
1.0.18).
                           DIR is the libexslt install directory.
  --with-fam              Include fam support
  --with-fbsql=DIR        Include FrontBase support. DIR is the FrontBase base
directory.
  --with-fdftk=DIR        Include FDF support.
  --with-fribidi=DIR      Include FriBidi support (requires FriBidi >= 0.10.4).
  --with-gd=DIR           Include GD support where DIR is GD install prefix.
                          If DIR is not set, the bundled GD library will be
used.
  --with-jpeg-dir=DIR     GD: Set the path to libjpeg install prefix.
  --with-png-dir=DIR      GD: Set the path to libpng install prefix.
  --with-zlib-dir=DIR     GD: Set the path to libz install prefix.
  --with-xpm-dir=DIR      GD: Set the path to libXpm install prefix.
  --with-ttf=DIR          GD: Include FreeType 1.x support
  --with-freetype-dir=DIR GD: Set the path to FreeType 2 install prefix.
  --with-t1lib=DIR        GD: Include T1lib support.
  --with-gettext=DIR      Include GNU gettext support.
  --with-gmp              Include GNU MP support
  --with-hwapi=DIR        Include official Hyperwave API support
  --with-hyperwave        Include Hyperwave support
  --with-iconv=DIR        Include iconv support
  --with-imap=DIR         Include IMAP support. DIR is the c-client install
prefix.
  --with-kerberos=DIR     IMAP: Include Kerberos support. DIR is the Kerberos
install dir.
  --with-imap-ssl=DIR     IMAP: Include SSL support. DIR is the OpenSSL install
dir.
  --with-informix=DIR     Include Informix support.  DIR is the Informix base
                           install directory, defaults to
${INFORMIXDIR:-nothing}.
  --with-ingres=DIR       Include Ingres II support. DIR is the Ingres
                           base directory (default $II_SYSTEM/II/ingres)
  --with-interbase=DIR    Include InterBase support.  DIR is the InterBase base
                           install directory, defaults to /usr/interbase
  --with-ircg             Include IRCG support
  --with-ircg-config=PATH IRCG: Path to the ircg-config script
  --with-java=DIR         Include Java support. DIR is the JDK base install
directory.
                           This extension is always built as shared.
  --with-ldap=DIR         Include LDAP support.
  --with-mcal=DIR         Include MCAL support.
  --with-mcrypt=DIR       Include mcrypt support.
  --with-mcve=DIR         Include MCVE support
  --with-mhash=DIR        Include mhash support.
  --with-ming=DIR         Include MING support
  --with-mnogosearch=DIR
                          Include mnoGoSearch support.  DIR is the mnoGoSearch
                           base install directory, defaults to
/usr/local/mnogosearch.
  --with-msession=DIR     Include msession support
  --with-msql=DIR         Include mSQL support.  DIR is the mSQL base
                           install directory, defaults to /usr/local/Hughes.
  --with-mysql=DIR        Include MySQL support. DIR is the MySQL base
directory.
                           If unspecified, the bundled MySQL library will be
used.
  --with-mysql-sock=DIR   MySQL: Location of the MySQL unix socket pointer.
                           If unspecified, the default locations are searched.
  --with-zlib-dir=DIR     MySQL: Set the path to libz install prefix.
  --with-ncurses=DIR      Include ncurses support (CLI/CGI only).
  --with-oci8=DIR         Include Oracle-oci8 support. Default DIR is
ORACLE_HOME.
```

```
  --with-adabas=DIR       Include Adabas D support.  DIR is the Adabas base
                           install directory, defaults to /usr/local.
  --with-sapdb=DIR        Include SAP DB support.  DIR is SAP DB base
                           install directory, defaults to /usr/local.
  --with-solid=DIR        Include Solid support.  DIR is the Solid base
                           install directory, defaults to /usr/local/solid
  --with-ibm-db2=DIR      Include IBM DB2 support.  DIR is the DB2 base
                           install directory, defaults to /home/db2inst1/sqllib
  --with-empress=DIR      Include Empress support.  DIR is the Empress base
                           install directory, defaults to \$EMPRESSPATH.
                           From PHP4, this option only supports Empress Version
                           8.60 and above
  --with-empress-bcs=DIR
                          Include Empress Local Access support.  DIR is the
                           Empress base install directory, defaults to
                           \$EMPRESSPATH.  From PHP4, this option only supports
                           Empress Version 8.60 and above.
  --with-birdstep=DIR     Include Birdstep support.  DIR is the Birdstep base
                           install directory, defaults to /usr/local/birdstep.
  --with-custom-odbc=DIR  Include a user defined ODBC support.
                           The DIR is ODBC install base directory,
                           which defaults to /usr/local.
                           Make sure to define CUSTOM_ODBC_LIBS and
                           have some odbc.h in your include dirs.
                           E.g., you should define following for
                           Sybase SQL Anywhere 5.5.00 on QNX, prior to
                           run configure script:
                              CPPFLAGS=\"-DODBC_QNX -DSQLANY_BUG\"
                              LDFLAGS=-lunix
                              CUSTOM_ODBC_LIBS=\"-ldblib -lodbc\".
  --with-iodbc=DIR        Include iODBC support.  DIR is the iODBC base
                           install directory, defaults to /usr/local.
  --with-esoob=DIR        Include Easysoft OOB support. DIR is the OOB base
                           install directory,
                           defaults to /usr/local/easysoft/oob/client.
  --with-unixODBC=DIR     Include unixODBC support.  DIR is the unixODBC base
                           install directory, defaults to /usr/local.
  --with-openlink=DIR     Include OpenLink ODBC support.  DIR is the
                           OpenLink base install directory, defaults to
                           /usr/local.  This is the same as iODBC.
  --with-dbmaker=DIR      Include DBMaker support.  DIR is the DBMaker base
                           install directory, defaults to where the latest
                           version of DBMaker is installed (such as
                           /home/dbmaker/3.6).
  --with-oracle=DIR       Include Oracle-oci7 support.  Default DIR is
                           ORACLE_HOME.
  --with-ovrimos=DIR      Include Ovrimos SQL Server support. DIR is the
                           Ovrimos libsqlcli install directory.
  --without-pcre-regex    Do not include Perl Compatible Regular Expressions
                           support. Use --with-pcre-regex=DIR to specify DIR
                           where PCRE's include and library files are located,
                           if not using bundled library.
  --with-pdflib=DIR       Include PDFlib support.
  --with-jpeg-dir=DIR     PDFLIB: define libjpeg install directory.
                                     (OPTIONAL for PDFlib v4)
  --with-png-dir=DIR      PDFLIB: define libpng install directory.
                                     (OPTIONAL for PDFlib v4)
  --with-zlib-dir=DIR     PDFLIB: define libz install directory.
                                     (OPTIONAL for PDFlib v4)
  --with-tiff-dir=DIR     PDFLIB: define libtiff install directory.
                                     (OPTIONAL for PDFlib v4)
  --with-pfpro=DIR        Include Verisign Payflow Pro support.
  --with-pgsql=DIR        Include PostgreSQL support.  DIR is the PostgreSQL
                           base install directory, defaults to /usr/local/pgsql.
  --with-pspell=DIR       Include PSPELL support.
                           This replaces the old ASPELL extension.
  --with-qtdom            Include QtDOM support (requires Qt >= 2.2.0).
  --with-libedit=DIR      Include libedit readline replacement (CLI/CGI only).
  --with-readline=DIR     Include readline support (CLI/CGI only).
  --with-recode=DIR       Include recode support.
  --with-mm=DIR           Include mm support for session storage
  --with-snmp=DIR         Include SNMP support.
  --with-regex=TYPE       regex library type: system, apache, php. Default: php
                           WARNING: Do NOT use unless you know what you are
doing!
```

```
  --with-swf=DIR          Include swf support
  --with-sybase=DIR       Include Sybase-DB support.  DIR is the Sybase home
                           directory, defaults to /home/sybase.
  --with-sybase-ct=DIR    Include Sybase-CT support.  DIR is the Sybase home
                           directory. Defaults to /home/sybase.
  --with-expat-dir=DIR    XML: external libexpat install dir
  --with-xmlrpc=DIR       Include XMLRPC-EPI support.
  --with-expat-dir=DIR    XMLRPC-EPI: libexpat dir for XMLRPC-EPI.
  --with-iconv-dir=DIR    XMLRPC-EPI: iconv dir for XMLRPC-EPI.
  --with-xslt-sablot=DIR  XSLT: Enable the sablotron backend.
  --with-expat-dir=DIR    XSLT: libexpat dir for Sablotron.
  --with-iconv-dir=DIR    XSLT: iconv dir for Sablotron.
  --with-sablot-js=DIR    XSLT: enable JavaScript support for Sablotron.
  --with-yaz=DIR          Include YAZ support (ANSI/NISO Z39.50).
                           DIR is the YAZ bin install directory.
  --with-zip=DIR          Include ZIP support (requires zziplib >= 0.10.6).
  --with-gnu-ld           assume the C compiler uses GNU ld default=no
  --with-pic              try to use only PIC/non-PIC objects default=use both
  --with-tsrm-pth=pth-config    Use GNU Pth.
  --with-tsrm-st
  --with-tsrm-pthreads    Use POSIX threads (default)

Some influential environment variables:
  CC          C compiler command
  CFLAGS      C compiler flags
  LDFLAGS     linker flags, e.g. -L<lib dir> if you have libraries in a
              nonstandard directory <lib dir>
  CPPFLAGS    C/C++ preprocessor flags, e.g. -I<include dir> if you have
              headers in a nonstandard directory <include dir>
  CPP         C preprocessor
  CXX         C++ compiler command
  CXXFLAGS    C++ compiler flags
  CXXCPP      C++ preprocessor

Use these variables to override the choices made by `configure' or to help
it to find libraries and programs with nonstandard names/locations.
```

Building PHP from source

If you just do a ./configure && make && make install with no arguments you will get the command-line php interpreter installed in /usr/local/bin

Normally you will want to build the DSO (Dynamic Shared Object) Apache Module version of PHP and you will probably want to link against your own MySQL client libraries:

```
% ./configure --with-apxs=/usr/sbin/apxs --with-mysql=/usr
% make
% make install
```

Typical configure flags

```
./configure --with-pdflib=/usr/local \
            --with-gd \
            --with-freetype-dir=/usr \
            --enable-gd-native-ttf \
            --enable-gd-imgstrttf \
            --with-jpeg-dir=/usr \
            --with-png-dir=/usr \
            --with-xpm-dir=/usr/X11R6 \
            --enable-exif \
            --with-config-file-path=/etc \
            --with-mysql=/usr \
            --enable-inline-optimization \
            --with-zlib \
            --with-zlib-dir=/usr \
            --with-expat=/usr \
            --enable-xslt \
            --with-xslt-sablot=/usr/local \
            --with-imap \
            --with-gettext \
            --with-kerberos \
            --with-imap-ssl \
            --with-mcrypt=/usr/local \
            --with-ldap \
            --with-pspell \
            --with-iconv \
            --with-ming \
            --enable-sockets \
            --with-openssl \
            --with-apxs
```

Building PHP from source

PHP as a static Apache module

PHP can also be built as a static module directly into your Apache httpd process. On some systems this will be fractionally faster, but it makes it a bit less convenient to upgrade PHP as you have to rebuild Apache as well. The procedure for doing this is:

```
% ./configure --with-apache=/usr/local/src/apache_1.3.22 \
              --with-mysql=/usr
% make
% make install
% cd /usr/local/src/apache_1.3.22
% ./configure --activate-module=src/modules/php4/libphp4.a
% make
% make install
```

The php.ini file

[php.ini]

```
[PHP]

; Enable the PHP scripting language engine under Apache.
engine = On

; Allow the <? tag.  Otherwise, only <?php and <script> tags are recognized.
; NOTE: Using short tags should be avoided when developing applications or
; libraries that are meant for redistribution, or deployment on PHP
; servers which are not under your control, because short tags may not
; be supported on the target server. For portable, redistributable code,
; be sure not to use short tags.
short_open_tag = On

; Allow ASP-style <% %> tags.
asp_tags = Off

; The number of significant digits displayed in floating point numbers.
precision    =  12

; Enforce year 2000 compliance (will cause problems with non-compliant browsers)
y2k_compliance = On

; Output buffering allows you to send header lines (including cookies) even
; after you send body content, at the price of slowing PHP's output layer a
; bit.  You can enable output buffering during runtime by calling the output
; buffering functions.  You can also enable output buffering for all files by
; setting this directive to On.  If you wish to limit the size of the buffer
; to a certain size - you can use a maximum number of bytes instead of 'On', as
; a value for this directive (e.g., output_buffering=4096).
output_buffering = Off

; You can redirect all of the output of your scripts to a function.  For
; example, if you set output_handler to "mb_output_handler", character
; encoding will be transparently converted to the specified encoding.
; Setting any output handler automatically turns on output buffering.
; Note: People who wrote portable scripts should not depend on this ini
;       directive. Instead, explicitly set the output handler using ob_start().
;       Using this ini directive may cause problems unless you know what script
;       is doing.
; Note: You cannot use both "mb_output_handler" with "ob_iconv_handler"
;       and you cannot use both "ob_gzhandler" and "zlib.output_compression".
;output_handler =

; Transparent output compression using the zlib library
; Valid values for this option are 'off', 'on', or a specific buffer size
; to be used for compression (default is 4KB)
; Note: Resulting chunk size may vary due to nature of compression. PHP
;       outputs chunks that are few hundreds bytes each as a result of
;       compression. If you prefer a larger chunk size for better
;       performance, enable output_buffering in addition.
; Note: You need to use zlib.output_handler instead of the standard
;       output_handler, or otherwise the output will be corrupted.
zlib.output_compression = Off

; You cannot specify additional output handlers if zlib.output_compression
; is activated here. This setting does the same as output_handler but in
; a different order.
;zlib.output_handler =

; Implicit flush tells PHP to tell the output layer to flush itself
; automatically after every output block.  This is equivalent to calling the
; PHP function flush() after each and every call to print() or echo() and each
; and every HTML block.  Turning this option on has serious performance
; implications and is generally recommended for debugging purposes only.
implicit_flush = Off

; The unserialize callback function will called (with the undefind class'
; name as parameter), if the unserializer finds an undefined class
; which should be instanciated.
; A warning appears if the specified function is not defined, or if the
; function doesn't include/implement the missing class.
; So only set this entry, if you really want to implement such a
; callback-function.
unserialize_callback_func=

; Whether to enable the ability to force arguments to be passed by reference
; at function call time.  This method is deprecated and is likely to be
; unsupported in future versions of PHP/Zend.  The encouraged method of
; specifying which arguments should be passed by reference is in the function
; declaration.  You're encouraged to try and turn this option Off and make
; sure your scripts work properly with it in order to ensure they will work
; with future versions of the language (you will receive a warning each time
; you use this feature, and the argument will be passed by value instead of by
; reference).
allow_call_time_pass_reference = On

; Safe Mode
;
safe_mode = Off

; By default, Safe Mode does a UID compare check when
; opening files. If you want to relax this to a GID compare,
; then turn on safe_mode_gid.
safe_mode_gid = Off

; When safe_mode is on, UID/GID checks are bypassed when
; including files from this directory and its subdirectories.
; (directory must also be in include_path or full path must
; be used when including)
safe_mode_include_dir =

; When safe_mode is on, only executables located in the safe_mode_exec_dir
; will be allowed to be executed via the exec family of functions.
safe_mode_exec_dir =

; Setting certain environment variables may be a potential security breach.
; This directive contains a comma-delimited list of prefixes.  In Safe Mode,
; the user may only alter environment variables whose names begin with the
; prefixes supplied here.  By default, users will only be able to set
; environment variables that begin with PHP_ (e.g. PHP_FOO=BAR).
;
; Note:  If this directive is empty, PHP will let the user modify ANY
; environment variable!
safe_mode_allowed_env_vars = PHP_

; This directive contains a comma-delimited list of environment variables that
; the end user won't be able to change using putenv().  These variables will be
; protected even if safe_mode_allowed_env_vars is set to allow to change them.
safe_mode_protected_env_vars = LD_LIBRARY_PATH

; open_basedir, if set, limits all file operations to the defined directory
; and below.  This directive makes most sense if used in a per-directory
; or per-virtualhost web server configuration file. This directive is
; NOT affected by whether Safe Mode is turned On or Off.
;open_basedir =

; This directive allows you to disable certain functions for security reasons.
; It receives a comma-delimited list of function names. This directive is
; NOT affected by whether Safe Mode is turned On or Off.
disable_functions =

; Colors for Syntax Highlighting mode.  Anything that's acceptable in
; <font color="??????"> would work.
;highlight.string  = #DD0000
;highlight.comment = #FF9900
;highlight.keyword = #007700
;highlight.bg      = #FFFFFF
;highlight.default = #0000BB
;highlight.html    = #000000


;
```

```
; Misc
;
; Decides whether PHP may expose the fact that it is installed on the server
; (e.g. by adding its signature to the Web server header).  It is no security
; threat in any way, but it makes it possible to determine whether you use PHP
; on your server or not.
expose_php = On


;;;;;;;;;;;;;;;;;;;
; Resource Limits ;
;;;;;;;;;;;;;;;;;;;

max_execution_time = 30 ;Maximum execution time of each script, in seconds
max_input_time = 60 ;Maximum amount of time each script may spend parsing
request data
memory_limit = 8M   ;Maximum amount of memory a script may consume (8MB)

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; Error handling and logging ;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

; error_reporting is a bit-field.  Or each number up to get desired error
; reporting level
; E_ALL             - All errors and warnings
; E_ERROR           - fatal run-time errors
; E_WARNING         - run-time warnings (non-fatal errors)
; E_PARSE           - compile-time parse errors
; E_NOTICE          - run-time notices (these are warnings which often result
;                     from a bug in your code, but it's possible that it was
;                     intentional (e.g., using an uninitialized variable and
;                     relying on the fact it's automatically initialized to an
;                     empty string)
; E_CORE_ERROR      - fatal errors that occur during PHP's initial startup
; E_CORE_WARNING    - warnings (non-fatal errors) that occur during PHP's
;                     initial startup
; E_COMPILE_ERROR   - fatal compile-time errors
; E_COMPILE_WARNING - compile-time warnings (non-fatal errors)
; E_USER_ERROR      - user-generated error message
; E_USER_WARNING    - user-generated warning message
; E_USER_NOTICE     - user-generated notice message
;
; Examples:
;
;   - Show all errors, except for notices
;
;error_reporting = E_ALL & ~E_NOTICE
;
;   - Show only errors
;
;error_reporting = E_COMPILE_ERROR|E_ERROR|E_CORE_ERROR
;
;   - Show all errors except for notices
;
error_reporting  =  E_ALL & ~E_NOTICE

; Print out errors (as a part of the output).  For production web sites,
; you're strongly encouraged to turn this feature off, and use error logging
; instead (see below).  Keeping display_errors enabled on a production web site
; may reveal security information to end users, such as file paths on your Web
; server, your database schema or other information.
display_errors = On

; Even when display_errors is on, errors that occur during PHP's startup
; sequence are not displayed.  It's strongly recommended to keep
; display_startup_errors off, except for when debugging.
display_startup_errors = Off

; Log errors into a log file (server-specific log, stderr, or error_log (below))
; As stated above, you're strongly advised to use error logging in place of
; error displaying on production web sites.
log_errors = Off

; Set maximum length of log_errors. In error_log information about the source is
; added. The default is 1024 and 0 allows to not apply any maximum length at
all.
log_errors_max_len = 1024

; Do not log repeated messages. Repeated errors must occur in same file on same
; line until ignore_repeated_source is set true.
ignore_repeated_errors = Off

; Ignore source of message when ignoring repeated messages. When this setting
; is On you will not log errors with repeated messages from different files or
; sourcelines.
ignore_repeated_source = Off

; If this parameter is set to Off, then memory leaks will not be shown (on
; stdout or in the log). This has only effect in a debug compile, and if
; error reporting includes E_WARNING in the allowed list
report_memleaks = On

; Store the last error/warning message in $php_errormsg (boolean).
track_errors = Off

; Disable the inclusion of HTML tags in error messages.
;html_errors = Off

; If html_errors is set On PHP produces clickable error messages that direct
; to a page describing the error or function causing the error in detail.
; You can download a copy of the PHP manual from http://www.php.net/docs.php
; and change docref_root to the base URL of your local copy including the
; leading '/'. You must also specify the file extension being used including
; the dot.
;docref_root = /phpmanual/
;docref_ext = .html

; String to output before an error message.
;error_prepend_string = "<font color=ff0000>"

; String to output after an error message.
;error_append_string = "</font>"

; Log errors to specified file.
;error_log = filename

; Log errors to syslog (Event Log on NT, not valid in Windows 95).
;error_log = syslog


;;;;;;;;;;;;;;;;;
; Data Handling ;
;;;;;;;;;;;;;;;;;
;
; Note - track_vars is ALWAYS enabled as of PHP 4.0.3

; The separator used in PHP generated URLs to separate arguments.
; Default is "&".
;arg_separator.output = "&amp;"

; List of separator(s) used by PHP to parse input URLs into variables.
; Default is "&".
; NOTE: Every character in this directive is considered as separator!
;arg_separator.input = ";&"

; This directive describes the order in which PHP registers GET, POST, Cookie,
; Environment and Built-in variables (G, P, C, E & S respectively, often
; referred to as EGPCS or GPC).  Registration is done from left to right, newer
; values override older values.
variables_order = "EGPCS"

; Whether or not to register the EGPCS variables as global variables.  You may
; want to turn this off if you don't want to clutter your scripts' global scope
; with user data.  This makes most sense when coupled with track_vars - in which
; case you can access all of the GPC variables through the $HTTP_*_VARS[],
; variables.
;
; You should do your best to write your scripts so that they do not require
; register_globals to be on;  Using form variables as globals can easily lead
; to possible security problems, if the code is not very well thought of.
register_globals = Off

; This directive tells PHP whether to declare the argv&argc variables (that
; would contain the GET information).  If you don't use these variables, you
; should turn it off for increased performance.
register_argc_argv = On

; Maximum size of POST data that PHP will accept.
post_max_size = 8M

; This directive is deprecated.  Use variables_order instead.
gpc_order = "GPC"

; Magic quotes
;

; Magic quotes for incoming GET/POST/Cookie data.
magic_quotes_gpc = On

; Magic quotes for runtime-generated data, e.g. data from SQL, from exec(), etc.
magic_quotes_runtime = Off

; Use Sybase-style magic quotes (escape ' with '' instead of \').
magic_quotes_sybase = Off

; Automatically add files before or after any PHP document.
auto_prepend_file =
auto_append_file =

; As of 4.0b4, PHP always outputs a character encoding by default in
; the Content-type: header.  To disable sending of the charset, simply
; set it to be empty.
;
; PHP's built-in default is text/html
default_mimetype = "text/html"
;default_charset = "iso-8859-1"

; Always populate the $HTTP_RAW_POST_DATA variable.
;always_populate_raw_post_data = On

; Allow handling of WebDAV http requests within PHP scripts (eg.
; PROPFIND, PROPPATCH, MOVE, COPY, etc..)
; If you want to get the post data of those requests, you have to
; set always_populate_raw_post_data as well.
;allow_webdav_methods = On

;;;;;;;;;;;;;;;;;;;;;;;;;
; Paths and Directories ;
;;;;;;;;;;;;;;;;;;;;;;;;;

; UNIX: "/path1:/path2"
;include_path = ".:/php/includes"
;
; Windows: "\path1;\path2"
;include_path = ".;c:\php\includes"

; The root of the PHP pages, used only if nonempty.
; if PHP was not compiled with FORCE_REDIRECT, you SHOULD set doc_root
; if you are running php as a CGI under any web server (other than IIS)
; see documentation for security issues.  The alternate is to use the
; cgi.force_redirect configuration below
doc_root =

; The directory under which PHP opens the script using /~username used only
; if nonempty.
user_dir =

; Directory in which the loadable extensions (modules) reside.
extension_dir = ./

; Whether or not to enable the dl() function.  The dl() function does NOT work
; properly in multithreaded servers, such as IIS or Zeus, and is automatically
; disabled on them.
enable_dl = On

; cgi.force_redirect is necessary to provide security running PHP as a CGI under
; most web servers.  Left undefined, PHP turns this on by default.  You can
; turn it off here AT YOUR OWN RISK
; *You CAN safely turn this off for IIS, in fact, you MUST.*
; cgi.force_redirect = 1

; if cgi.force_redirect is turned on, and you are not running under Apache or
Netscape
; (iPlanet) web servers, you MAY need to set an environment variable name that
PHP
; will look for to know it is OK to continue execution.  Setting this variable
MAY
; cause security issues, KNOW WHAT YOU ARE DOING FIRST.
; cgi.redirect_status_env = ;

; FastCGI under IIS (on WINNT based OS) supports the ability to impersonate
; security tokens of the calling client.  This allows IIS to define the
; security context that the request runs under.  mod_fastcgi under Apache
; does not currently support this feature (03/17/2002)
; Set to 1 if running under IIS.  Default is zero.
; fastcgi.impersonate = 1;

; cgi.rfc2616_headers configuration option tells PHP what type of headers to
; use when sending HTTP response code. If it's set 0 PHP sends Status: header
that
; is supported by Apache. When this option is set to 1 PHP will send
; RFC2616 compliant header.
; Set to 1 if running under IIS.  Default is zero.
;cgi.rfc2616_headers = 0


;;;;;;;;;;;;;;;;
; File Uploads ;
;;;;;;;;;;;;;;;;

; Whether to allow HTTP file uploads.
file_uploads = On

; Temporary directory for HTTP uploaded files (will use system default if not
; specified).
;upload_tmp_dir =

; Maximum allowed size for uploaded files.
upload_max_filesize = 2M


;;;;;;;;;;;;;;;;;;
; Fopen wrappers ;
;;;;;;;;;;;;;;;;;;

; Whether to allow the treatment of URLs (like http:// or ftp://) as files.
allow_url_fopen = On

; Define the anonymous ftp password (your email address)
;from="john@doe.com"

; Define the User-Agent string
; user_agent="PHP"

; Default timeout for socket based streams (seconds)
default_socket_timeout = 60

; If your scripts have to deal with files from Macintosh systems,
; or you are running on a Mac and need to deal with files from
; unix or win32 systems, setting this flag will cause PHP to
; automatically detect the EOL character in those files so that
; fgets() and file() will work regardless of the source of the file.
; auto_detect_line_endings = Off


;;;;;;;;;;;;;;;;;;;;;;
```

```
; Dynamic Extensions ;
;;;;;;;;;;;;;;;;;;;;;;
;
; If you wish to have an extension loaded automatically, use the following
; syntax:
;
;   extension=modulename.extension
;
; For example, on Windows:
;
;   extension=msql.dll
;
; ... or under UNIX:
;
;   extension=msql.so
;
; Note that it should be the name of the module only; no directory information
; needs to go here.  Specify the location of the extension with the
; extension_dir directive above.


;Windows Extensions
;Note that MySQL and ODBC support is now built in, so no dll is needed for it.
;
;extension=php_bz2.dll
;extension=php_ctype.dll
;extension=php_cpdf.dll
;extension=php_crack.dll
;extension=php_curl.dll
;extension=php_cybercash.dll
;extension=php_db.dll
;extension=php_dba.dll
;extension=php_dbase.dll
;extension=php_dbx.dll
;extension=php_domxml.dll
;extension=php_dotnet.dll
;extension=php_exif.dll
;extension=php_fbsql.dll
;extension=php_fdf.dll
;extension=php_filepro.dll
;extension=php_gd.dll
;extension=php_gettext.dll
;extension=php_hyperwave.dll
;extension=php_iconv.dll
;extension=php_ifx.dll
;extension=php_iisfunc.dll
;extension=php_imap.dll
;extension=php_ingres.dll
;extension=php_interbase.dll
;extension=php_java.dll
;extension=php_ldap.dll
;extension=php_mbstring.dll
;extension=php_mcrypt.dll
;extension=php_mhash.dll
;extension=php_ming.dll
;extension=php_mssql.dll
;extension=php_oci8.dll
;extension=php_openssl.dll
;extension=php_oracle.dll
;extension=php_pdf.dll
;extension=php_pgsql.dll
;extension=php_printer.dll
;extension=php_shmop.dll
;extension=php_snmp.dll
;extension=php_sockets.dll
;extension=php_sybase_ct.dll
;extension=php_tokenizer.dll
;extension=php_w32api.dll
;extension=php_xslt.dll
;extension=php_yaz.dll
;extension=php_zlib.dll


;;;;;;;;;;;;;;;;;;;
```

```
; Module Settings ;
;;;;;;;;;;;;;;;;;;;

[Syslog]
; Whether or not to define the various syslog variables (e.g. $LOG_PID,
; $LOG_CRON, etc.).  Turning it off is a good idea performance-wise.  In
; runtime, you can define these variables by calling define_syslog_variables().
define_syslog_variables  = Off

[mail function]
; For Win32 only.
SMTP = localhost

; For Win32 only.
sendmail_from = me@localhost.com

; For Unix only.  You may supply arguments as well (default: "sendmail -t -i").
;sendmail_path =

[Java]
;java.class.path = .\php_java.jar
;java.home = c:\jdk
;java.library = c:\jdk\jre\bin\hotspot\jvm.dll
;java.library.path = .\

[SQL]
sql.safe_mode = Off

[ODBC]
;odbc.default_db    =  Not yet implemented
;odbc.default_user  =  Not yet implemented
;odbc.default_pw    =  Not yet implemented

; Allow or prevent persistent links.
odbc.allow_persistent = On

; Check that a connection is still valid before reuse.
odbc.check_persistent = On

; Maximum number of persistent links.  -1 means no limit.
odbc.max_persistent = -1

; Maximum number of links (persistent + non-persistent).  -1 means no limit.
odbc.max_links = -1

; Handling of LONG fields.  Returns number of bytes to variables.  0 means
; passthru.
odbc.defaultlrl = 4096

; Handling of binary data.  0 means passthru, 1 return as is, 2 convert to char.
; See the documentation on odbc_binmode and odbc_longreadlen for an explanation
; of uodbc.defaultlrl and uodbc.defaultbinmode
odbc.defaultbinmode = 1

[MySQL]
; Allow or prevent persistent links.
mysql.allow_persistent = On

; Maximum number of persistent links.  -1 means no limit.
mysql.max_persistent = -1

; Maximum number of links (persistent + non-persistent).  -1 means no limit.
mysql.max_links = -1

; Default port number for mysql_connect().  If unset, mysql_connect() will use
; the $MYSQL_TCP_PORT or the mysql-tcp entry in /etc/services or the
; compile-time value defined MYSQL_PORT (in that order).  Win32 will only look
; at MYSQL_PORT.
mysql.default_port =

; Default socket name for local MySQL connects.  If empty, uses the built-in
; MySQL defaults.
mysql.default_socket =

; Default host for mysql_connect() (doesn't apply in safe mode).
```

```
mysql.default_host =

; Default user for mysql_connect() (doesn't apply in safe mode).
mysql.default_user =

; Default password for mysql_connect() (doesn't apply in safe mode).
; Note that this is generally a bad idea to store passwords in this file.
; Any user with PHP access can run 'echo get_cfg_var("mysql.default_password")
; and reveal this password!  And of course, any users with read access to this
; file will be able to reveal the password as well.
mysql.default_password =

; Maximum time (in secondes) for connect timeout. -1 means no limimt
mysql.connect_timeout = -1

; Trace mode. When trace_mode is active (=On), warnings for table/index scans
and
; SQL-Erros will be displayed.
mysql.trace_mode = Off

[PostgresSQL]
; Allow or prevent persistent links.
pgsql.allow_persistent = On

; Detect broken persistent links always with pg_pconnect(). Need a little
overhead.
pgsql.auto_reset_persistent = Off

; Maximum number of persistent links.  -1 means no limit.
pgsql.max_persistent = -1

; Maximum number of links (persistent+non persistent).  -1 means no limit.
pgsql.max_links = -1

; Ignore PostgreSQL backends Notice message or not.
pgsql.ignore_notice = 0

; Log PostgreSQL backends Noitce message or not.
; Unless pgsql.ignore_notice=0, module cannot log notice message.
pgsql.log_notice = 0

[dbx]
; returned column names can be converted for compatibility reasons
; possible values for dbx.colnames_case are
; "unchanged" (default, if not set)
; "lowercase"
; "uppercase"
; the recommended default is either upper- or lowercase, but
; unchanged is currently set for backwards compatibility
dbx.colnames_case = "unchanged"

[bcmath]
; Number of decimal digits for all bcmath functions.
bcmath.scale = 0

[browscap]
;browscap = extra/browscap.ini

[Session]
; Handler used to store/retrieve data.
session.save_handler = files

; Argument passed to save_handler.  In the case of files, this is the path
; where data files are stored. Note: Windows users have to change this
; variable in order to use PHP's session functions.
; As of PHP 4.0.1, you can define the path as:
;     session.save_path = "N;/path"
; where N is an integer.  Instead of storing all the session files in
; /path, what this will do is use subdirectories N-levels deep, and
; store the session data in those directories.  This is useful if you
; or your OS have problems with lots of files in one directory, and is
; a more efficient layout for servers that handle lots of sessions.
; NOTE 1: PHP will not create this directory structure automatically.
;         You can use the script in the ext/session dir for that purpose.
; NOTE 2: See the section on garbage collection below if you choose to
;         use subdirectories for session storage
session.save_path = /tmp

; Whether to use cookies.
session.use_cookies = 1

; This option enables administrators to make their users invulnerable to
; attacks which involve passing session ids in URLs; defaults to 0.
; session.use_only_cookies = 1

; Name of the session (used as cookie name).
session.name = PHPSESSID

; Initialize session on request startup.
session.auto_start = 0

; Lifetime in seconds of cookie or, if 0, until browser is restarted.
session.cookie_lifetime = 0

; The path for which the cookie is valid.
session.cookie_path = /

; The domain for which the cookie is valid.
session.cookie_domain =

; Handler used to serialize data.  php is the standard serializer of PHP.
session.serialize_handler = php

; Define the probability that the 'garbage collection' process is started
; on every session initialization.
; The probability is calculated by using gc_probability/gc_dividend,
; e.g. 1/100 means 1%.

session.gc_probability = 1
session.gc_dividend    = 100

; After this number of seconds, stored data will be seen as 'garbage' and
; cleaned up by the garbage collection process.
; WARNING: Your filesystem must store access times.  Windows FAT does
;          not.  So, see session_set_save_handler() and write your own
;          session handler with a different mechanism for cleaning up sessions.
session.gc_maxlifetime = 1440

; NOTE: If you are using the subdirectory option for storing session files
;       (see session.save_path above), then garbage collection does not
;       happen automatically.  You will need to do your own garbage
;       collection through a shell script, cron entry, or some other method.
;       For example, the following script would is the equivalent of
;       setting session.gc_maxlifetime to 1440 (1440 seconds = 24 minutes):
;          cd /path/to/sessions; find -cmin +24 | xargs rm

; PHP 4.2 and less have an undocumented feature/bug that allows you to
; to initialize a session variable in the global scope, albeit register_globals
; is disabled.  PHP 4.3 and later will warn you, if this feature is used.
; You can disable the feature and the warning seperately. At this time,
; the warning is only displayed, if bug_compat_42 is enabled.

session.bug_compat_42 = 1
session.bug_compat_warn = 1

; Check HTTP Referer to invalidate externally stored URLs containing ids.
; HTTP_REFERER has to contain this substring for the session to be
; considered as valid.
session.referer_check =

; How many bytes to read from the file.
session.entropy_length = 0

; Specified here to create the session id.
session.entropy_file =

;session.entropy_length = 16

;session.entropy_file = /dev/urandom
```

```
; Set to {nocache,private,public,} to determine HTTP caching aspects
; or leave this empty to avoid sending anti-caching headers.
session.cache_limiter = nocache

; Document expires after n minutes.
session.cache_expire = 180

; trans sid support is disabled by default.
; Use of trans sid may risk your users security.
; Use this option with caution.
; - User may send URL contains active session ID
;   to other person via. email/irc/etc.
; - URL that contains active session ID may be stored
;   in publically accessible computer.
; - User may access your site with the same session ID
;   always using URL stored in browser's history or bookmarks.
session.use_trans_sid = 0

; The URL rewriter will look for URLs in a defined set of HTML tags.
; form/fieldset are special; if you include them here, the rewriter will
; add a hidden <input> field with the info which is otherwise appended
; to URLs.  If you want XHTML conformity, remove the form entry.
; Note that all valid entries require a "=", even if no value follows.
url_rewriter.tags = "a=href,area=href,frame=src,input=src,form=,fieldset="

[Assertion]
; Assert(expr); active by default.
;assert.active = On

; Issue a PHP warning for each failed assertion.
;assert.warning = On

; Don't bail out by default.
;assert.bail = Off

; User-function to be called if an assertion fails.
;assert.callback = 0

; Eval the expression with current error_reporting().  Set to true if you want
; error_reporting(0) around the eval().
;assert.quiet_eval = 0

[Sockets]
; Use the system read() function instead of the php_read() wrapper.
sockets.use_system_read = On

[mbstring]
; language for internal character representation.
;mbstring.language = Japanese

; internal/script encoding.
; Some encoding cannot work as internal encoding.
; (e.g. SJIS, BIG5, ISO-2022-*)
;mbstring.internal_encoding = EUC-JP

; http input encoding.
;mbstring.http_input = auto

; http output encoding. mb_output_handler must be
; registered as output buffer to function
;mbstring.http_output = SJIS

; enable automatic encoding translation accoding to
; mbstring.internal_encoding setting. Input chars are
; converted to internal encoding by setting this to On.
; Note: Do not use automatic encoding translation for
;       portable libs/applications.
;mbstring.encoding_translation = Off

; automatic encoding detection order.
; auto means
;mbstring.detect_order = auto

; substitute_character used when character cannot be converted
; one from another
;mbstring.substitute_character = none;

; overload(replace) single byte functions by mbstring functions.
; mail(), ereg(), etc are overloaded by mb_send_mail(), mb_ereg(),
; etc. Possible values are 0,1,2,4 or combination of them.
; For example, 7 for overload everything.
; 0: No overload
; 1: Overload mail() function
; 2: Overload str*() functions
; 4: Overload ereg*() functions
;mbstring.func_overload = 0

[exif]
; Exif UNICODE user comments are handled as UCS-2BE/UCS-2LE and JIS as JIS.
; With mbstring support this will automatically be converted into the encoding
; given by corresponding encode setting. When empty mbstring.internal_encoding
; is used. For the decode settings you can distinguish between motorola and
; intel byte order. A decode setting cannot be empty.
;exif.encode_unicode = ISO-8859-15
;exif.decode_unicode_motorola = UCS-2BE
;exif.decode_unicode_intel    = UCS-2LE
;exif.encode_jis =
;exif.decode_jis_motorola = JIS
;exif.decode_jis_intel    = JIS

; Local Variables:
; tab-width: 4
; End:
```

Apache ErrorHandler

An interesting Apache configuration trick is that you can tell Apache to pass requests that generate a given error to a PHP script.

httpd.conf

```
ErrorHandler 404 /error.php
```

Later on we will use this to do some very interesting things.

Apache Forcetype

Another Apache configuration trick is to use the ForceType directive to force specific urls to be treated as PHP scripts:

httpd.conf

```
<Location /products>
  ForceType application/x-httpd-php
</Location>
```

Again, we will put this to use later on.

Template Systems

Why?

- Separate content and design
- Separate designers from programmers

Basics

- Some sort of special tags for variables: {variable}, <variable>, etc
- Most have support for blocks (looping for table rows)
- Some have support for conditional sections

TemplatePower

http://templatepower.codocad.com

- Nested dynamic blocks
- Block/file includes
- Ignore tag
- Save/load parsed templates
- Database template support

Code

```
<?php
include 'tplutil.inc';
include 'class.TemplatePower.inc.php';

$tpl = new TemplatePower('presentations/slides/intro/templatepower.tpl');
$tpl->prepare();

$tpl->assign("TITLE", 'My Title');
$tpl->assign("USER", get_user());

foreach (get_users() as $user) {
  $tpl->newBlock('USERS');
  $tpl->assign('USER', $user);
}

$tpl->printToScreen();
```

Template

```
<h1>{TITLE}</h1>
<form><select>
<option>{USER}</option>
<!-- START BLOCK : USERS -->
<option>{USER}</option>
<!-- END BLOCK : USERS -->
</select></form>
```

Output

Smarty

http://smarty.php.net/

- Probably the most complex and complete system
- Loops
- Conditional sections
- Variable modifiers
- Plugins
- Built-in functions
- Configuration files
- More... (See Andrei and Sterling's tutorial.)

Code

```
<?php
include 'tplutil.inc';
require 'Smarty.class.php';

$tpl = new Smarty();
$tpl->assign('TITLE', 'my title');
$tpl->assign('USER', get_user());
$tpl->assign('USERS', get_users());

$tpl->display('smarty.tpl');
```

Template

```
<h1>{$TITLE|capitalize}</h1>
<form><select>
{html_options values=$USERS output=$USERS selected=$USER}
</select></form>
```

PHP

http://www.php.net/

- Full progamming language
- Include files
- Loops and conditional statements
- User-defined functions
- Etc.

Code

```
<?php
include 'tplutil.inc';
$title = 'My Title';
$user = get_user();
$users = get_users();
include 'php.tpl';
```

Template

```
<h1><?=$title?></h1>
<form><select>
<?foreach ($users as $option) {?>
<option <?=($option == $user) ? 'selected' : ''?>><?=$option?></option>
<?}?>
</select></form>
```

Output

debug_backtrace

debug_backtrace() is a new function in PHP 4.3

Custom error handler

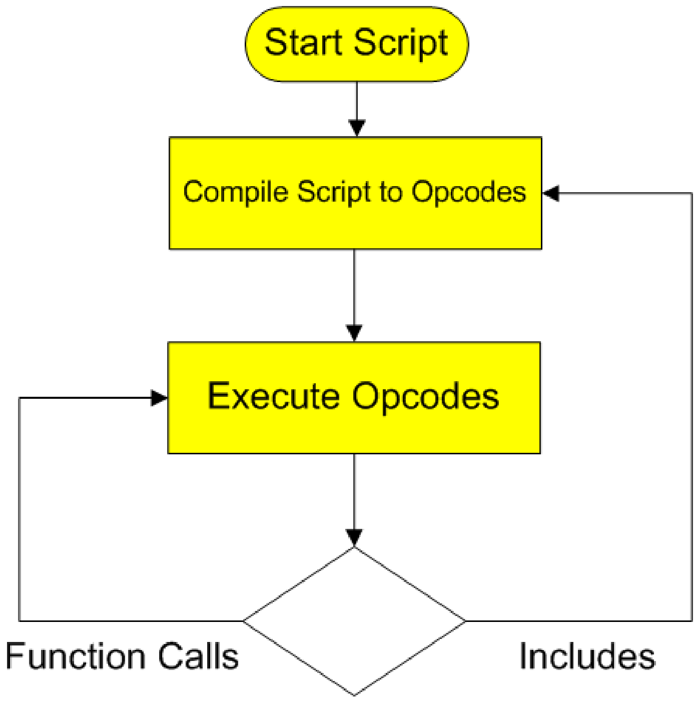
```
<?
  function short($str) {
    if(strstr($str,'/'))
      return substr(strrchr($str,'/'),1);
    else return $str;
  }
  function myErrorHandler($errno,$errstr,$errfile,$errline)
  {
    echo "$errno: $errstr in ".short($errfile)." at line $errline<br />\n";
    echo "Backtrace<br />\n";
    $trace = debug_backtrace();
    foreach($trace as $ent) {
      if(isset($ent['file'])) $ent['file'].':';
      if(isset($ent['function'])) {
        echo $ent['function'].'(';
        if(isset($ent['args'])) {
          $args='';
          foreach($ent['args'] as $arg) { $args.=$arg.','; }
          echo rtrim(short($args),',');
        }
        echo ') ';
      }
      if(isset($ent['line'])) echo 'at line '.$ent['line'].' ';
      if(isset($ent['file'])) echo 'in '.short($ent['file']);
      echo "<br />\n";
    }
  }

  set_error_handler('myErrorHandler');
  include 'file2.php';
  test2(1,0);
?>
```

Custom error handler

```
<?
    function test1($b,$a) {
        $a/$b;
    }

    function test2($a,$b) {
        test1($b,$a);
    }
?>
```

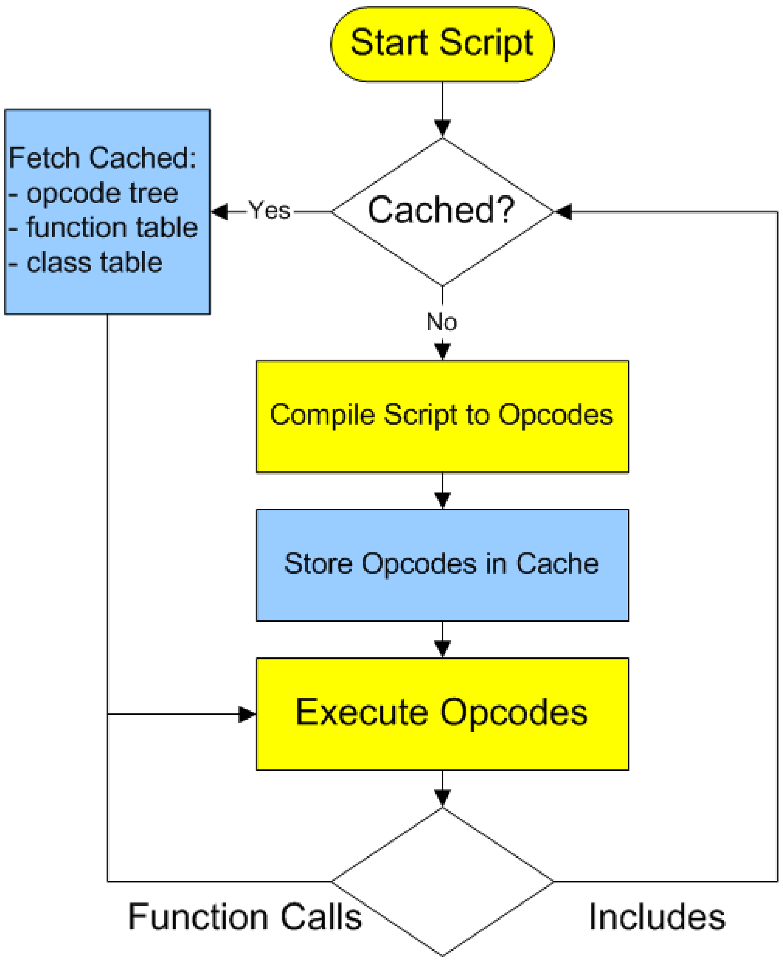
PHP Opcode Caches

Standard PHP



PHP Opcode Caches

PHP with an Opcode Cache



PHP Opcode Caches

There are a number of them out there.

- APC - open source
- IonCube Accelerator - free, but closed source
- Zend Cache - commercial

Installation

Typically very trivial. For IonCube, for example, in your php.ini add:

```
zend_extension = "/usr/local/lib/php/php_accelerator_1.3.3r2.so"
```

So why isn't this just built into standard PHP?

This gets asked often, and although not a good answer, the answer is that since it wasn't in PHP from the start a number of competing plugin cache systems were developed and choosing one over another at this point would effectively chop these projects, both commercial and free, off at their knees. This way they can compete and innovate against each other at the cost of duplicated effort.

Profiling PHP

Why Profile?

Because your assumptions of how things work behind the scenes are not always correct. By profiling your code you can identify where the bottlenecks are quantitatively.

How?

PEAR/Pecl to the rescue!

```
www:~> pear install apd
downloading apd-0.4p1.tgz ...
...done: 39,605 bytes
16 source files, building
running: phpize
PHP Api Version        : 20020918
Zend Module Api No     : 20020429
Zend Extension Api No  : 20021010
building in /var/tmp/pear-build-root/apd-0.4p1
running: /tmp/tmprFlAqf/apd-0.4p1/configure
running: make
apd.so copied to /tmp/tmprFlAqf/apd-0.4p1/apd.so
install ok: apd 0.4p1
```

Woohoo!

```
www:~> pear info apd
About apd-0.4p1
===============
+-----------------+---------------------------------------------------+
| Package         | apd                                               |
| Summary         | A full-featured engine-level profiler/debugger   |
| Description     | APD is a full-featured profiler/debugger that is |
|                 | loaded as a zend_extension. It aims to be an     |
|                 | analog of C's gprof or Perl's Devel::DProf.      |
| Maintainers     | George Schlossnagle <george@omniti.com> (lead)   |
| Version         | 0.4p1                                             |
| Release Date    | 2002-11-25                                        |
| Release License | PHP License                                       |
| Release State   | stable                                            |
| Release Notes   | Fix for pre-4.3 versions of php                   |
| Last Modified   | 2002-12-02                                        |
+-----------------+---------------------------------------------------+
```

```
www:~> pear config-show
Configuration:
==============
+----------------------+-----------------+-------------------------------------+
| PEAR executables     | bin_dir         | /usr/local/bin                      |
| directory            |                 |                                     |
| PEAR documentation   | doc_dir         | /usr/local/lib/php/docs             |
| directory            |                 |                                     |
| PHP extension        | ext_dir         | /usr/local/lib/php/extensions/no-de |
| directory            |                 | bug-non-zts-20020429                |
| PEAR directory       | php_dir         | /usr/local/lib/php                  |
| PEAR Installer cache | cache_dir       | /tmp/pear/cache                     |
| directory            |                 |                                     |
| PEAR data directory  | data_dir        | /usr/local/lib/php/data             |
| PEAR test directory  | test_dir        | /usr/local/lib/php/tests            |
| Cache TimeToLive     | cache_ttl       | <not set>                           |
| Preferred Package    | preferred_state | stable                              |
| State                |                 |                                     |
| Unix file mask       | umask           | 18                                  |
| Debug Log Level      | verbose         | 1                                   |
| HTTP Proxy Server    | http_proxy      | <not set>                           |
| Address              |                 |                                     |
| PEAR server          | master_server   | pear.php.net                        |
| PEAR password (for   | password        | <not set>                           |
| maintainers)          |                  |                                      |
| PEAR username (for    | username         | <not set>                            |
| maintainers)          |                  |                                      |
+-----------------------+------------------+--------------------------------------+
```

Profiling PHP

```
www:~> cd /usr/local/lib/php
www:/usr/local/lib/php> ln -s extensions/no-debug-non-zts-20020429/apd.so apd.so
```

Then in your php.ini file:

```
zend_extension = "/usr/local/lib/php/apd.so"
apd.dumpdir = /tmp
```

It isn't completely transparent. You need to tell the profiler when to start profiling. At the top of a script you want to profile, add this call:

```
<?php
apd_set_pprof_trace();
?>
```

The use the command-line tool called pprofp:

```
wwww: ~> pprofp
pprofp <flags> <trace file>
    Sort options
    -a          Sort by alphabetic names of subroutines.
    -l          Sort by number of calls to subroutines
    -m          Sort by memory used in a function call.
    -r          Sort by real time spent in subroutines.
    -R          Sort by real time spent in subroutines (inclusive of child
calls).
    -s          Sort by system time spent in subroutines.
    -S          Sort by system time spent in subroutines (inclusive of child
calls).
    -u          Sort by user time spent in subroutines.
    -U          Sort by user time spent in subroutines (inclusive of child
calls).
    -v          Sort by average amount of time spent in subroutines.
    -z          Sort by user+system time spent in subroutines. (default)

    Display options
    -c          Display Real time elapsed alongside call tree.
    -i          Suppress reporting for php builtin functions
    -O <cnt>    Specifies maximum number of subroutines to display. (default 15)
    -t          Display compressed call tree.
    -T          Display uncompressed call tree.


www: ~> ls -latr /tmp/pprofp.*
-rw-r--r--    1 nobody   nobody      16692 Dec  3 01:19 /tmp/pprof.04545

www: ~> pprofp -z /tmp/pprof.04545


Trace for /home/rasmus/phpweb/index.php
Total Elapsed Time =    0.69
Total System Time  =    0.01
Total User Time    =    0.08


         Real         User        System             secs/    cumm
%Time (excl/cumm)  (excl/cumm)  (excl/cumm) Calls    call    s/call  Memory
Usage Name
--------------------------------------------------------------------------------------
 33.3  0.11  0.13   0.02  0.03   0.01  0.01     7   0.0043    0.0057
298336 require_once
 22.2  0.02  0.02   0.02  0.02   0.00  0.00   183   0.0001    0.0001
-33944 feof
 11.1  0.01  0.01   0.01  0.01   0.00  0.00     3   0.0033    0.0033
-14808 define
 11.1  0.04  0.04   0.01  0.01   0.00  0.00   182   0.0001    0.0001
112040 fgetcsv
 11.1  0.25  0.25   0.01  0.01   0.00  0.00     6   0.0017    0.0017
3768 getimagesize
```

```
 11.1  0.01  0.01   0.01  0.01   0.00  0.00    55   0.0002    0.0002
2568 sprintf
  0.0  0.00  0.00   0.00  0.00   0.00  0.00     7   0.0000    0.0000
-136 printf
  0.0  0.00  0.00   0.00  0.00   0.00  0.00     1   0.0000    0.0000
136 htmlspecialchars
  0.0  0.00  0.00   0.00  0.00   0.00  0.00     1   0.0000    0.0000
-16 mirror_provider_url
  0.0  0.00  0.00   0.00  0.00   0.00  0.00     7   0.0000    0.0000
112 spacer
  0.0  0.00  0.00   0.00  0.00   0.00  0.00    10   0.0000    0.0000
-552 delim
  0.0  0.00  0.00   0.00  0.00   0.00  0.00     1   0.0000    0.0000
112 mirror_provider
  0.0  0.00  0.00   0.00  0.00   0.00  0.00    20   0.0000    0.0000
-624 print_link
  0.0  0.00  0.00   0.00  0.00   0.00  0.00     1   0.0000    0.0000
24 have_stats
  0.0  0.00  0.00   0.00  0.00   0.00  0.00     1   0.0000    0.0000
-72 make_submit
  0.0  0.00  0.00   0.00  0.00   0.00  0.00     2   0.0000    0.0000
112 strrchr
  0.0  0.08  0.08   0.00  0.00   0.00  0.00     2   0.0000    0.0000
168 filesize
  0.0  0.00  0.00   0.00  0.00   0.00  0.00     1   0.0000    0.0000
-16 commonfooter
  0.0  0.00  0.11   0.00  0.00   0.00  0.00     2   0.0000    0.0000
0 download_link
  0.0  0.00  0.25   0.00  0.01   0.00  0.00     6   0.0000    0.0017
208 make_image
```

$PATH_INFO

$PATH_INFO is your friend when it comes to creating clean URLS. Take for example this URL:

```
http://www.company.com/products/routers
```

If the Apache configuration contains this block:

```
<Location "/products">
  ForceType application/x-httpd-php
</Location>
```

Then all you have to do is create a PHP script in your DOCUMENT_ROOT named 'products' and you can use the $PATH_INFO variable which will contain the string, '/routers', to make a DB query.

ErrorDocument

Apache's ErrorDocument directive can come in handy. For example, this line in your Apache configuration file:

```
ErrorDocument 404 /error.php
```

Can be used to redirect all 404 errors to a PHP script. The following server variables are of interest:

- $REDIRECT_ERROR_NOTES - File does not exist: /docroot/bogus
- $REDIRECT_REQUEST_METHOD - GET
- $REDIRECT_STATUS - 404
- $REDIRECT_URL - /docroot/bogus

Don't forget to send a 404 status if you choose not to redirect to a real page.

```
<? Header('HTTP/1.0 404 Not Found'); ?>
```

Interesting uses

- Search for closest matching valid URL and redirect
- Use attempted url text as a DB keyword lookup
- Funky caching

Efficient image caching

First, set up an ErrorDocument 404 handler for your images directory.

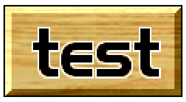
```
<Directory /home/doc_root/images>
    ErrorDocument 404 /images/generate.php
</Directory>
```

Then generate.php looks like this:

```
<?php
$filename = basename($_SERVER['REDIRECT_URL']);
if(preg_match('/^([^]?)([^]?)([^_]?)\.(.?)$/',$filename, $reg)) {
    $type = $reg[1];
    $text = $reg[2];
    $rgb = $reg[3];
    $ext = $reg[4];
}

if(strlen($rgb)==6) {
    $r = hexdec(substr($rgb,0,2));
    $g = hexdec(substr($rgb,2,2));
    $b = hexdec(substr($rgb,4,2));
} else $r = $g = $b = 0;

    switch(strtolower($ext)) {
        case 'jpg':
            Header("Content-Type: image/jpg");
            break;
        case 'png':
        case 'gif': / We don't do gif - send a png instead /
            Header("Content-Type: image/png");
            break;
        default:
            break;
    }

    switch($type) {
        case 'solid':
            $im = imagecreatetruecolor(80,80);
            $bg = imagecolorallocate($im, $r, $g, $b);
            imagefilledrectangle($im,0,0,80,80,$bg);
            break;
        case 'button':
            $si = 32; $font = "php";
            $im = imagecreatefrompng('blank_wood.png');
            $tsize = imagettfbbox($si,0,$font,$text);
            $dx = abs($tsize[2]-$tsize[0]);
            $dy = abs($tsize[5]-$tsize[3]);
            $x = ( imagesx($im) - $dx ) / 2;
            $y = ( imagesy($im) - $dy ) / 2 + $dy;
            $white = ImageColorAllocate($im,255,255,255);
            $black = ImageColorAllocate($im,$r,$g, $b);
            ImageTTFText($im, $si, 0, $x, $y, $white, $font, $text);
            ImageTTFText($im, $si, 0, $x+2, $y, $white, $font, $text);
            ImageTTFText($im, $si, 0, $x, $y+2, $white, $font, $text);
            ImageTTFText($im, $si, 0, $x+2, $y+2, $white, $font, $text);
            ImageTTFText($im, $si, 0, $x+1, $y+1, $black, $font, $text);
        break;
    }
    Header("HTTP/1.1 200 OK");
    $dest_file = dirname($_SERVER['SCRIPT_FILENAME']).'/'.$filename;
    switch(strtolower($ext)) {
        case 'png':
        case 'gif':
            @ImagePNG($im,$dest_file);
            ImagePNG($im);
            break;
        case 'jpg':
            @ImageJPEG($im,$dest_file);
            ImageJPEG($im);
            break;
```

```
          }
    ?>
```

The URL, http://localhost/images/button_test_000000.png produces this image:



Funky Caching

An interesting way to handle caching is to have all 404's redirected to a PHP script.

```
ErrorDocument 404 /generate.php
```

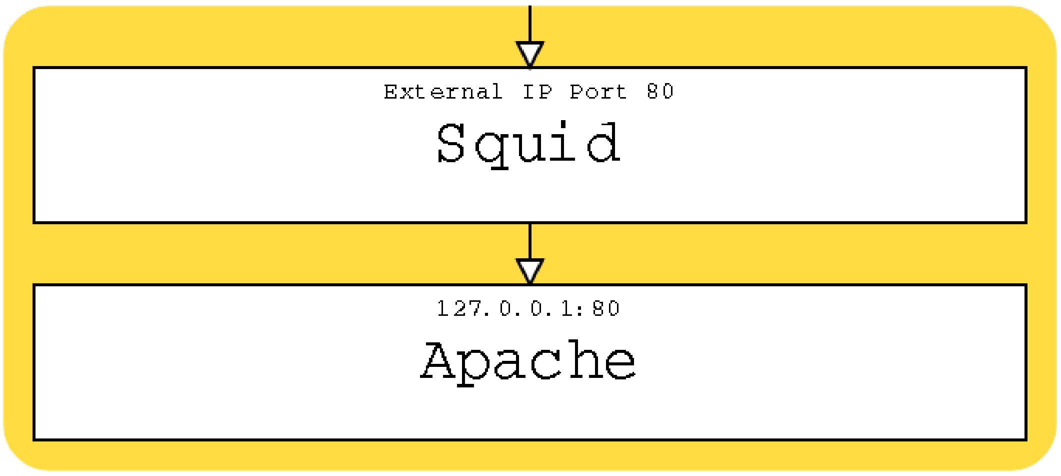
Then in your generate.php script use the contents of $REDIRECT_URI to determine which URL the person was trying to get to. In your database you would then have fields linking content to the URL they affect and from that you should be able to generate the page. Then in your generate.php script do something like:

```
<?php
    $s = $REDIRECT_URI;
    $d = $DOCUMENT_ROOT;
    // determine requested uri
    $uri = substr($s, strpos($s,$d) + strlen($d) + 1);
    ob_start();  // Start buffering output
    // ... code to fetch and output content from DB ...
    $data = ob_get_contents();
    $fp = fopen("$DOCUMENT_ROOT/$uri",'w');
    fputs($fp,$data);
    fclose($fp);
    ob_end_flush(); // Flush and turn off buffering
?>
```

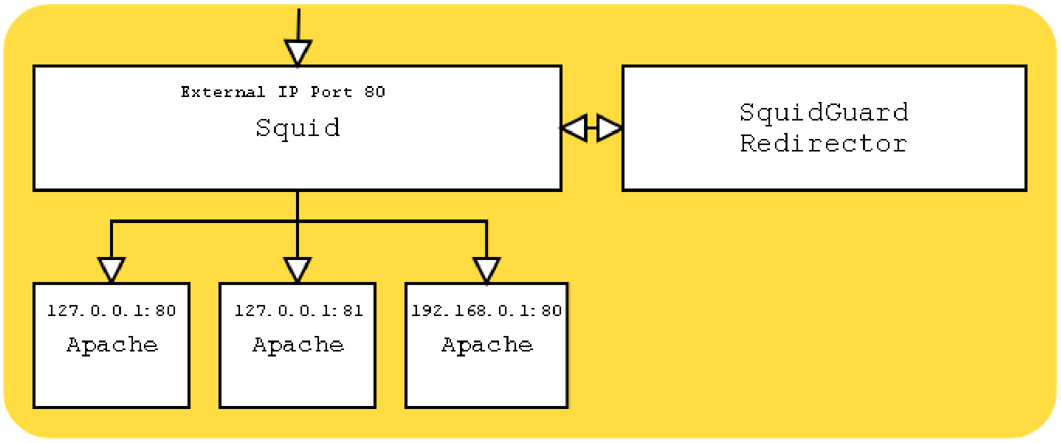
So, the way it works, when a request comes in for a page that doesn't exist, generate.php checks the database and determines if it should actually exist and if so it will create it and respond with this generated data. The next request for that same URL will get the generated page directly. So in order to refresh your cache you simply have to delete the files.

Squid

For really busy sites, a reverse proxy like Squid is magical! Either run it as a single-server accelerator:



Or as a front-end cache to a number of local or remote servers:



Note:

Watch out for any use of $REMOTE_ADDR in your PHP scripts. Use $HTTP_X_FORWARDED_FOR instead.

Squid Configuration

Make it listen to port 80 on our external interface:

```
http_port 198.186.203.51:80
```

If we don't do cgi-bin stuff, comment these out:

```
#acl QUERY urlpath_regex cgi-bin
#no_cache deny QUERY
```

If we have plenty of RAM, bump this up a bit:

```
cache_mem 16MB
maximum_object_size 14096 KB
```

Specify where to store cached files (size in Megs, level 1 subdirs, level 2 subdirs)

```
cache_dir ufs /local/squid/cache 500 16 256
```

Get rid of the big store.log file:

```
cache_store_log none
```

Set our SNMP public community string:

```
acl snmppublic snmp_community public
```

Get rid of "allow all" and use list of hosts we are blocking (1 ip per line):

```
#http_access allow all
acl forbidden src "/local/squid/etc/forbidden"
http_access allow !forbidden
```

Set user/group squid should run as:

```
cache_effective_user squid
cache_effective_group daemon
```

Single-server reverse proxy setup (set up Apache to listen to port 80 on the loopback):

```
httpd_accel_host 127.0.0.1
httpd_accel_port 80
httpd_accel_single_host on
httpd_accel_uses_host_header on
```

Only allow localhost access through snmp:

```
snmp_access allow snmppublic localhost
```

Load Balancing with Squid

There are two primary ways to configure Squid to be a load balancer.

Private /etc/hosts

The easiest is to simply list multiple ips for the httpd_accel_host in your /etc/hosts file and Squid will automatically round-robin across the backend servers.

Use cache_peer

This is more complex. Squid has advanced support for communicating with other caches. It just so happens that this communication happens over HTTP so you can set Squid up to treat the web servers you wish to load balance as peer caches. The configuration would look something like this:

```
httpd_accel_host www.visible-domain.com
httpd_accel_uses_host_header on
never_direct allow all
cache_peer server1 parent 80 0 no-query round-robin
cache_peer server2 parent 80 0 no-query round-robin
cache_peer server3 parent 80 0 no-query round-robin
```

no-query in the above tells Squid not to try to send ICP (Internet Cache Protocol) requests to the Apache servers. You could turn on the echo port on each server and redirect these ICP requests to there which would make this system automatically detect a server that is down.

Avoiding Redirectors

It is generally a good idea to avoid external redirectors. A lot of things can be done directly in Squid's config file. For example:

```
acl domain1 dstdomain www.domain1.com
acl domain2 dstdomain www.domain2.com
acl domain3 dstdomain www.domain3.com
cache_peer_access server1 allow domain1
cache_peer_access server2 allow domain2
cache_peer_access server3 allow domain3
cache_peer_access server1 deny all
cache_peer_access server2 deny all
cache_peer_access server3 deny all
```

This would configure Squid to send requests for pages on certain domains to certain backend servers. These backend servers could actually be aliases to different ips on the same server if you wanted to run multiple Apache instances on different ports on the same box.

Load Balancing

The Easy (And Expensive) Solution

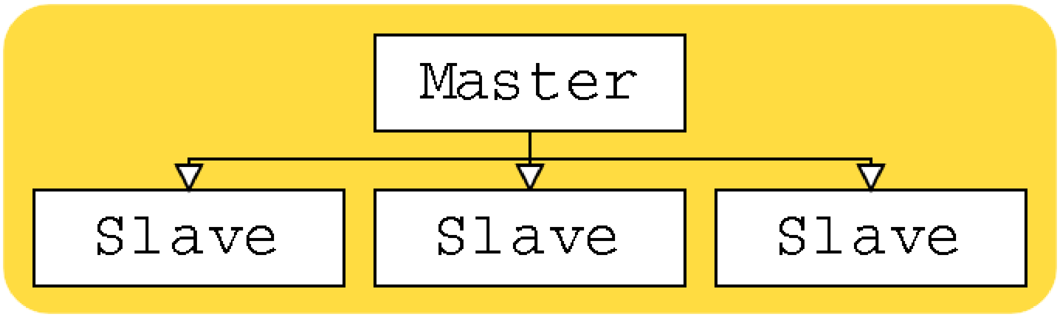
You can buy dedicated boxes that do all sorts of advanced load balancing for you. Some popular ones are:

- Cisco CSS
- Foundry ServerIron
- Intel Netstructure
- F5 BIG-IP

If you are on Linux you could also try LVS. See www.LinuxVirtualServer.org.

MySQL Replication

As of version 3.23.15 (try to use 3.23.29 or later), MySQL supports one-way replication. Since most web applications usually have more reads than writes, an architecture which distributes reads across multiple servers can be very beneficial.



In typical MySQL fashion, setting up replication is trivial. On your master server add this to your "my.cnf" file:

```
[mysqld]
log-bin
server-id=1
```

And add a replication user id for slaves to log in as:

```
GRANT FILE ON . TO repl@"%" IDENTIFIED BY 'foobar';
```

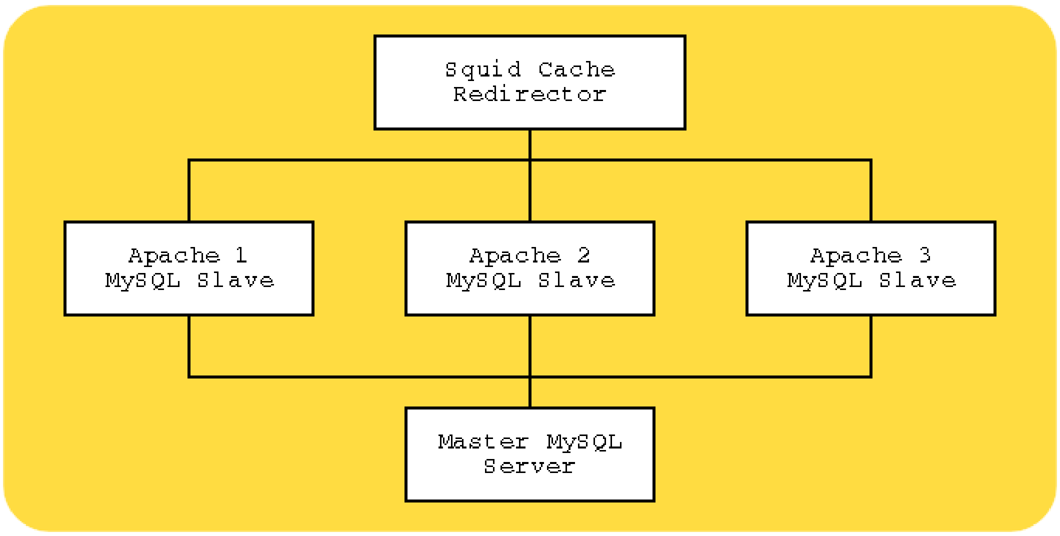
If you are using MySQL 4.0.2 or later, replace FILE with REPLICATION SLAVE in the above. Then on your slave servers:

```
[mysqld]
set-variable = max_connections=200
log-bin
master-host=192.168.0.1
master-user=repl
master-password=foobar
master-port=3306
server-id=2
```

Make sure each slave has its own unique server-id. And since these will be read-only slaves, you can start them with these options to speed them up a bit:

```
--skip-bdb --low-priority-updates
--delay-key-write-for-all-tables
```

Stop your master server. Copy the table files to each of your slave servers. Restart the master, then start all the slaves. And you are done. Combining MySQL replication with a Squid reverse cache and redirector and you might have an architecture like this:

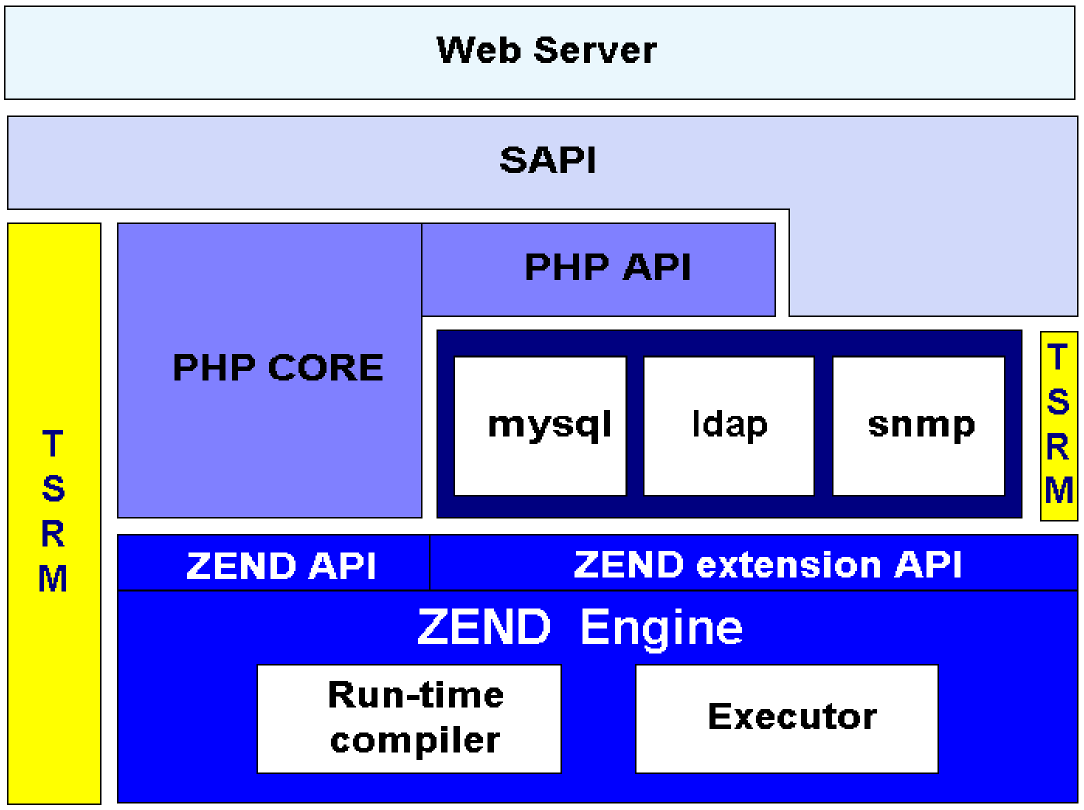


You would then write your application to send all database writes to the master server and all reads to the local slave. It is also possible to set up two-way replication, but you would need to supply your own application-level logic to maintain atomicity of distributed writes. And you lose a lot of the advantages of this architecture if you do this as the writes would have to go to all the slaves anyway.

Why hack PHP?

- Fix a bug
- Create a PHP API for a C/C++ library
- Optimization (implementing performance-critical logic in C)

Architecture Diagram



PHP Tree Layout

```
% cvs -d:pserver:cvsread@cvs.php.net:/repository co php4
```

Contents of php4/

- Zend/ - the Zend Engine
- TSRM/ - the thread-safe resource manager
- main/ - the basics
- ext/ - Extensions
- ext/standard/ - the standard functions
- ext/mysql/
- ext/imap/
- ext/hyperwave/
- ...
- sapi/ - the Server API
- sapi/apache/
- sapi/cgi/
- sapi/isapi/
- ...

Getting Started

- Grab a copy of the current php4 CVS tree
- Read README.EXT_SKEL
- Read README.SELF-CONTAINED-EXTENSIONS
- Read README.PARAMETER_PARSING_API
- Look at the existing extensions for code examples
- Follow this example

Grabbing a copy of the CVS tree - Anonymous CVS instructions

```
% cvs -d:pserver:cvsread@cvs.php.net:/repository login
Password: phpfi
% cvs -d:pserver:cvsread@cvs.php.net:/repository co php4
% cd php4
% cvs -d:pserver:cvsread@cvs.zend.com:/repository login
Password: Zend
% cvs -d:pserver:cvsread@cvs.zend.com:/repository co Zend
% cvs -d:pserver:cvsread@cvs.zend.com:/repository co TSRM
```

Before you dive in...

You need to make sure your environment is ready. Install bison, flex, autoconf, libtool and of course gcc and make.

Next, install the command-line version of PHP

```
% cd php4
% ./configure --with-mysql=/usr --with-pgsql --with-zlib \
  --enable-inline-optimization --with-config-file-path=/etc
% make
% make install
```

Test It

```
% /usr/local/bin/php -v
4.3.0-dev
```

Plan your Extension

Try to plan your extension before you start writing your code. A good way to do this is to write a sample PHP script that illustrates how your extension is going to be accessed from a PHP script.

XMMS Sample Extension

```
<?php
xmms_start();
if(isset($url)) {
  xmms_url($url);
  $len = xmms_playlist_length();
  xmms_play($len-1);
} else {
  xmms_play(5);
}
$song = xmms_current();
$list = xmms_playlist(range($song['pos']-5, $song['pos']+5));
foreach($list as $pos=>$plsong) {
  echo "$pos: ".$plsong['title']."<br>\n";
}
sleep(10);
xmms_stop();
?>
```

Define the functions

From your plan, generate a function definition file

xmms.def

```
int    xmms_start(void)        Start up XMMS if it is not already running
bool   xmms_is_running(void)   Check if XMMS is running
int    xmms_playlist_length(void) Get the number of songs in the playlist
int    xmms_playlist_pos(void) Get current playlist position
array  xmms_playlist([mixed pos]) Get full or partial playlist info
void   xmms_play([int pos])    Play current or provided song position
void   xmms_stop(void)         Stop playing current song
void   xmms_pause(void)        Pause current song (toggle)
void   xmms_quit(void)         Force XMMS to terminate
int    xmms_version(void)      Get the XMMS version
void   xmms_next(void)         Go to next song in playlist
void   xmms_prev(void)         Go to previous song in playlist
string xmms_status(void)       Returns current XMMS status
bool   xmms_is_repeat(void)    Get repeat setting
int    xmms_repeat([bool setting]) Toggle/Set repeat setting
bool   xmms_is_shuffle(void)   Get shuffle setting
int    xmms_shuffle([bool setting]) Toggle/Set shuffle setting
void   xmms_add(mixed songs)   Add entries to playlist
void   xmms_delete(int pos)    Delete entry from playlist
void   xmms_clear(void)        Clear playlist
array  xmms_current(void)      Get current song info
void   xmms_volume([int level [, int right_level]]) Set volume
void   xmms_url(string url)    Play from url
void   xmms_skin([string skin_file]) Set XMMS skin
float  xmms_time([float time_pos]) Get/Set output time position
void   xmms_main_win(bool setting) Hide/Unhide main xmms window
void   xmms_pl_win(bool setting) Hide/Unhide playlist xmms window
void   xmms_eq_win(bool setting) Hide/Unhide equalizer xmms window
object xmms([bool start])      XMMS class constructor
```

A skeleton extension

Run the ext_skel script which creates all the required skeleton files.

```
% cd php4/ext
% ./ext_skel --extname=xmms --proto=xmms.def
Creating directory xmms
Creating basic files: config.m4 Makefile.in .cvsignore xmms.c
                      php_xmms.h CREDITS EXPERIMENTAL
                      tests/001.phpt xmms.php [done].
% ls xmms
config.m4    CREDITS       EXPERIMENTAL    xmms.c
xmms.php     Makefile.in   php_xmms.h      tests/
```

Standalone or Embedded?

At this point you have a choice on how to proceed. You can either develop your extension under the main PHP framework where you run the top-level buildconf script to include your extension in PHP's main configuration script, or you can develop your extension in a completely separate directory where you run the phpize script to populate your extension directory with all the required files.

Embedded

```
% cd xmms
% vi config.m4
% cd ../..
% ./buildconf
% ./configure --[with|enable]-xmms
% make
```

Standalone

```
% mv xmms /tmp
% cd /tmp/xmms
% vi config.m4
% phpize
% ./configure --[with|enable]-xmms
% make
```

Compiling

Compiling the test skeleton extension

```
% cd /tmp/xmms
% vi config.m4
```

The config.m4 file

The config.m4 file adds any extension-specific configuration options. If your extension relies on external libraries and header files, you need to define those in this file and uncomment the --with-xmms section. If your extension does not rely on any external libraries, uncomment the --enable-xmms section.

Configure, build and install

```
% ./configure
% make
% cp modules/xmms.so /usr/local/lib/php
```

Testing your extension

The ext_skel script created a sample xmms.php script. A quick way to check if your extension is working is to edit this script and add dl('xmms.so') to the top. Then run it through your command-line PHP parser:

```
% php -q xmms.php
Congratulations!
You have successfully modified ext/xmms/config.m4.
Module xmms is now compiled into PHP.
```

Structure

The structure of a typical extension contains the following components:

includes

```
#ifdef HAVE_CONFIG_H
#include "config.h"
#endif

#include "php.h"
#include "php_ini.h"
#include "ext/standard/info.h"
#include "php_xmms.h"
```

function_entry

```
function_entry xmms_functions[] = {
    PHP_FE(confirm_xmms_compiled,    NULL)
    {NULL, NULL, NULL}
};
```

module_entry

```
zend_module_entry xmms_module_entry = {
    "xmms",
    xmms_functions,
    PHP_MINIT(xmms),
    PHP_MSHUTDOWN(xmms),
    PHP_RINIT(xmms),
    PHP_RSHUTDOWN(xmms),
    PHP_MINFO(xmms),
    STANDARD_MODULE_PROPERTIES
};
```

PHP_INI_BEGIN

```
PHP_INI_BEGIN()
    STD_PHP_INI_ENTRY("xmms.value",      "42",
                      PHP_INI_ALL, OnUpdateInt, global_value,
              zend_xmms_globals, xmms_globals)
    STD_PHP_INI_ENTRY("xmms.string", "xmmsbar",
                      PHP_INI_ALL, OnUpdateString, global_string,
              zend_xmms_globals, xmms_globals)
PHP_INI_END()
```

PHP_MINIT_FUNCTION

```
PHP_MINIT_FUNCTION(xmms)
{
    REGISTER_INI_ENTRIES();
    return SUCCESS;
}
```

PHP_MSHUTDOWN_FUNCTION

```
PHP_MSHUTDOWN_FUNCTION(xmms)
{
    UNREGISTER_INI_ENTRIES();
    return SUCCESS;
}
```

PHP_RINIT_FUNCTION

```
PHP_RINIT_FUNCTION(xmms)
```

```
{
    return SUCCESS;
}
```

PHP_RSHUTDOWN_FUNCTION

```
PHP_RSHUTDOWN_FUNCTION(xmms)
{
    return SUCCESS;
}
```

PHP_MINFO_FUNCTION

```
PHP_MINFO_FUNCTION(xmms)
{
    php_info_print_table_start();
    php_info_print_table_header(2, "xmms support", "enabled");
    php_info_print_table_end();

    DISPLAY_INI_ENTRIES();
}
```

functions

```
PHP_FUNCTION(confirm_xmms_compiled)
{
    char *arg = NULL;
    int arg_len, len;
    char string[256];

    if (zend_parse_parameters(ZEND_NUM_ARGS() TSRMLS_CC,
                              "s", &arg, &arg_len) == FAILURE) {
        return;
    }

    len = sprintf(string, "Congratulations!
You have successfully modified ext/%.78s/config.m4.
Module %.78s is now compiled into PHP.", "xmms", Z_STRVAL_PP(arg));
    RETURN_STRINGL(string, len, 1);
}
```

the pval/zval datatype

Data in PHP is stored in the pval or zval datatype. Two names for the same thing. They are defined like this:

```
typedef union _zvalue_value {
    long lval;          / long value /
    double dval;        / double value /
    struct {
        char *val;
        int len;
    } str;
    HashTable ht;      / hash table value */
    zend_object obj;
} zvalue_value;

struct _zval_struct {
    zvalue_value value; / value /
    zend_uchar type;    / active type /
    zend_uchar is_ref;
    zend_ushort refcount;
};

typedef struct _zval_struct zval;
typedef zval pval;
```

Convenience Macros

```
#define Z_LVAL(zval)        (zval).value.lval
#define Z_BVAL(zval)        ((zend_bool)(zval).value.lval)
#define Z_DVAL(zval)        (zval).value.dval
#define Z_STRVAL(zval)      (zval).value.str.val
#define Z_STRLEN(zval)      (zval).value.str.len
#define Z_ARRVAL(zval)      (zval).value.ht
#define Z_OBJ(zval)         &(zval).value.obj
#define Z_OBJPROP(zval)     (zval).value.obj.properties
#define Z_OBJCE(zval)       (zval).value.obj.ce
#define Z_RESVAL(zval)      (zval).value.lval

#define Z_LVAL_P(zval_p)        Z_LVAL(*zval_p)
#define Z_BVAL_P(zval_p)        Z_BVAL(*zval_p)
#define Z_DVAL_P(zval_p)        Z_DVAL(*zval_p)
#define Z_STRVAL_P(zval_p)      Z_STRVAL(*zval_p)
#define Z_STRLEN_P(zval_p)      Z_STRLEN(*zval_p)
#define Z_ARRVAL_P(zval_p)      Z_ARRVAL(*zval_p)
#define Z_OBJ_P(zval_p)         Z_OBJ(*zval_p)
#define Z_OBJPROP_P(zval_p)     Z_OBJPROP(*zval_p)
#define Z_OBJCE_P(zval_p)       Z_OBJCE(*zval_p)
#define Z_RESVAL_P(zval_p)      Z_RESVAL(*zval_p)

#define Z_LVAL_PP(zval_pp)      Z_LVAL(**zval_pp)
#define Z_BVAL_PP(zval_pp)      Z_BVAL(**zval_pp)
#define Z_DVAL_PP(zval_pp)      Z_DVAL(**zval_pp)
#define Z_STRVAL_PP(zval_pp)    Z_STRVAL(**zval_pp)
#define Z_STRLEN_PP(zval_pp)    Z_STRLEN(**zval_pp)
#define Z_ARRVAL_PP(zval_pp)    Z_ARRVAL(**zval_pp)
#define Z_OBJ_PP(zval_pp)       Z_OBJ(**zval_pp)
#define Z_OBJPROP_PP(zval_pp)   Z_OBJPROP(**zval_pp)
#define Z_OBJCE_PP(zval_pp)     Z_OBJCE(**zval_pp)
#define Z_RESVAL_PP(zval_pp)    Z_RESVAL(**zval_pp)','100%')
```

Parsing Parameters

In its simplest form you can call ZEND_NUM_ARGS() to check the number of parameters passed to your function.

```
PHP_FUNCTION(count_args)
{
    RETURN_LONG(ZEND_NUM_ARGS());
}
```

You can use the WRONG_PARAM_COUNT macro to output a generic wrong parameter count warning.

```
PHP_FUNCTION(my_func)
{
    if(ZEND_NUM_ARGS() > 1) {
        WRONG_PARAM_COUNT;
    }
    RETURN_TRUE;
}
```

Real Parameter Parsing

To do real parameter handling, use the zend_parse_parameters() function.

```
PHP_FUNCTION(return_arg)
{
    int argc = ZEND_NUM_ARGS();
    long arg;

    if (zend_parse_parameters(argc TSRMLS_CC,
                              "l", &arg) == FAILURE)
        return;

    RETURN_LONG(arg);
}
```

You can make it an optional argument by putting a "|" in the argument format string. Like this:

```
PHP_FUNCTION(return_arg)
{
    int argc = ZEND_NUM_ARGS();
    long arg;

    if (zend_parse_parameters(argc TSRMLS_CC,
                              "|l", &arg) == FAILURE)
        return;

    if(argc) RETURN_LONG(arg)
    else RETURN_FALSE
}
```

Here are all the possible modifiers in the argument format string:

```
'l'  long (integer)
'd'  double (floating point)
's'  string
'b'  boolean
'r'  resource
'a'  array
'o'  any object
'O'  specific object
'z'  mixed
```

So, for example, a function that takes a string, an integer, and array and an optional resource would look like this:

```
PHP_FUNCTION(complex_func)
{
    int argc = ZEND_NUM_ARGS();
    char *str = NULL;
    int str_len;
    int res_id = -1;
    long arg;
    zval *arr = NULL;
    zval *res = NULL;

    if (zend_parse_parameters(argc TSRMLS_CC,
                              "sla|r", &str, &str_len,
                  &arg, &arr, &res) == FAILURE)
        return;

    if (res) {
        ZEND_FETCH_RESOURCE(???, ???, res, res_id,
                            "???", ???_rsrc_id);
    }

    RETURN_FALSE;
}
```

Array Parameters

If your function takes an array parameter. You can check it and access it like this:

```
PHP_FUNCTION(my_arr)
{
    int argc = ZEND_NUM_ARGS();
    zval *tmp, arr = NULL;

    if (zend_parse_parameters(argc TSRMLS_CC,
                              "a", &arr) == FAILURE)
        return;

    zend_hash_internal_pointer_reset(Z_ARRVAL_P(arr));
    while (zend_hash_get_current_data(Z_ARRVAL_P(arr),
                                      (void **)&tmp) == SUCCESS) {
        convert_to_long_ex(tmp);
        php_printf("%ld\n",Z_LVAL_PP(tmp));
        zend_hash_move_forward(Z_ARRVAL_P(arr));
    }
    RETURN_TRUE;
}
```

Here the call to zend_parse_parameters() does the type check. If the user passes a non-array to this function s/he will get a message that looks like this:

```
foo.php(4) : Warning - my_arr() expects argument 1 to be array, integer given
```

If you wanted to write a function that could take either an array of integers or perhaps a single integer and have your function figure it out automatically you do do it like this:

```
PHP_FUNCTION(my_arr)
{
    int argc = ZEND_NUM_ARGS();
    zval *tmp, arg = NULL;

    if (zend_parse_parameters(argc TSRMLS_CC,
                              "z", &arg) == FAILURE)
        return;

    switch(Z_TYPE_P(arg)) {
    case IS_ARRAY:
        zend_hash_internal_pointer_reset(Z_ARRVAL_P(arg));
        while (zend_hash_get_current_data(Z_ARRVAL_P(arg),
                                          (void **)&tmp) == SUCCESS) {
            convert_to_long_ex(tmp);
            php_printf("%ld\n",Z_LVAL_PP(tmp));
            zend_hash_move_forward(Z_ARRVAL_P(arg));
        }
        break;
    case IS_STRING:
    case IS_LONG:
    case IS_DOUBLE:
        convert_to_long_ex(&arg);
        php_printf("%ld\n",Z_LVAL_P(arg));
        break;
    }
    RETURN_TRUE;
}
```

The possible types are:

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

```
#define IS_CONSTANT_ARRAY   9
```

Returning Values

The simple types are returned using the following convenience macros:

```
#define RETVAL_RESOURCE(l)          ZVAL_RESOURCE(return_value, l)
#define RETVAL_BOOL(b)              ZVAL_BOOL(return_value, b)
#define RETVAL_NULL()               ZVAL_NULL(return_value)
#define RETVAL_LONG(l)              ZVAL_LONG(return_value, l)
#define RETVAL_DOUBLE(d)            ZVAL_DOUBLE(return_value, d)
#define RETVAL_STRING(s, dup)       ZVAL_STRING(return_value, s, dup)
#define RETVAL_STRINGL(s, l, dup)   ZVAL_STRINGL(return_value, s, l, dup)
#define RETVAL_EMPTY_STRING()       ZVAL_EMPTY_STRING(return_value)
#define RETVAL_FALSE                ZVAL_BOOL(return_value, 0)
#define RETVAL_TRUE                 ZVAL_BOOL(return_value, 1)

#define RETURN_RESOURCE(l)          { RETVAL_RESOURCE(l); return; }
#define RETURN_BOOL(b)              { RETVAL_BOOL(b); return; }
#define RETURN_NULL()               { RETVAL_NULL(); return;}
#define RETURN_LONG(l)              { RETVAL_LONG(l); return; }
#define RETURN_DOUBLE(d)            { RETVAL_DOUBLE(d); return; }
#define RETURN_STRING(s, dup)       { RETVAL_STRING(s, dup); return; }
#define RETURN_STRINGL(s, l, dup)   { RETVAL_STRINGL(s, l, dup); return; }
#define RETURN_EMPTY_STRING()       { RETVAL_EMPTY_STRING(); return; }
#define RETURN_FALSE                { RETVAL_FALSE; return; }
#define RETURN_TRUE                 { RETVAL_TRUE; return; }
```

As you can see, returning a value from a function involves setting the special return_value zval to the return value and then just returning from the function call. At the C level, all functions return void.

Simple sum() function:

```
PHP_FUNCTION(my_sum)
{
    int argc = ZEND_NUM_ARGS();
    long arg1;
    long arg2;

    if (zend_parse_parameters(argc TSRMLS_CC,
                              "ll", &arg1, &arg2) == FAILURE)
        return;

    RETURN_LONG(arg1+arg2);
}
```

Here we uppercase every other character of a string.

```
PHP_FUNCTION(my_upper)
{
    char *str = NULL;
    int argc = ZEND_NUM_ARGS();
    int str_len;
    char *p;
    int i;

    if (zend_parse_parameters(argc TSRMLS_CC,
                              "s", &str, &str_len) == FAILURE)
        return;

    for(p=str, i=0; *p; p++,i++) {
        if(i%2) p = toupper(p);
    }
    RETURN_STRINGL(str,str_len,1);
}
```

Returning an Array

Returning an array is a little bit more complex. Here we take a string and a count and return an array with the specified number of elements containing the given string.

```
PHP_FUNCTION(my_array)
{
    char *str = NULL;
    int argc = ZEND_NUM_ARGS();
    int str_len;
    long count;
    int i;

    if (zend_parse_parameters(argc TSRMLS_CC,
                              "sl", &str, &str_len, &count) == FAILURE)
        return;

    array_init(return_value);
    for(i=0; i<count; i++) {
        add_index_stringl(return_value, i, str, str_len, 1);
    }
}
```

So, to return an array you first call array_init() on the return_value zval and then populate the array by calling the various add_* functions. The functions available to populate an array are:

```
add_index_long(zval *arg, uint idx, long n)
add_index_null(zval *arg, uint idx)
add_index_bool(zval *arg, uint idx, int b)
add_index_resource(zval *arg, uint idx, int r)
add_index_double(zval *arg, uint idx, double d)
add_index_string(zval arg, uint idx, char str, int dup)
add_index_stringl(zval arg, uint idx, char str, uint length, int dup)
add_index_zval(zval arg, uint index, zval value)
```

Recall that PHP supports the non-indexed array syntax. eg.

```
<? $a[] = 1; ?>
```

This same concept is available in the extension API using these functions:

```
add_next_index_long(zval *arg, long n)
add_next_index_null(zval *arg)
add_next_index_bool(zval *arg, int b)
add_next_index_resource(zval *arg, int r)
add_next_index_double(zval *arg, double d)
add_next_index_string(zval arg, char str, int dup)
add_next_index_stringl(zval arg, char str, uint length, int dup)
add_next_index_zval(zval arg, zval value)
```

You can also create an array with non-numeric indices. Otherwise known as an associative array using these functions:

```
add_assoc_long(zval arg, char key, long n)
add_assoc_null(zval arg, char key)
add_assoc_bool(zval arg, char key, int b)
add_assoc_resource(zval arg, char key, int r)
add_assoc_double(zval arg, char key, double d)
add_assoc_string(zval arg, char key, char *str, int dup)
add_assoc_stringl(zval arg, char key, char *str, uint length, int dup)
add_assoc_zval(zval arg, char key, zval *value)
```

To return a 2-dimensional array, you simply make new arrays and add them as elements of the top array. Like this:

```
PHP_FUNCTION(my_array)
{
    char *str = NULL;
    int argc = ZEND_NUM_ARGS();
    int str_len;
    long count;
    int i;
    pval *tmp_arr;

    if (zend_parse_parameters(argc TSRMLS_CC,
                          "sl", &str, &str_len,
              &count) == FAILURE)
        return;

    array_init(return_value);
    for(i=0; i<count; i++) {
        MAKE_STD_ZVAL(tmp_arr);
        array_init(tmp_arr);
        add_assoc_long_ex(tmp_arr,
                      str, str_len,
              count+i);
        add_next_index_zval(return_value, tmp_arr);
    }
}
```

Note the use of add_assoc_long_ex() in the above example. If we know the length of the key, then this is slightly faster than calling add_assoc_long(). The macro definition shows why:

```
int add_assoc_long_ex(zval arg, char key, uint key_len, long n);
#define add_assoc_long(__arg, __key, __n) \\
        add_assoc_long_ex(__arg, __key, strlen(__key)+1, __n)
```

Object Factory Approach

This gets quite a bit more complex than returning simple types, or even arrays. The first thing to do is to declare an extension-wide global class ptr along with the methods available in this object you are creating.

```
static zend_class_entry *my_class_entry_ptr;
static zend_function_entry php_my_class_functions[] = {
    PHP_FE(add,  NULL)
    PHP_FALIAS(del,           my_del,     NULL)
    PHP_FALIAS(list,          my_list,    NULL)
    {NULL, NULL, NULL}
};
```

Next, in our module_init function we initialize and register our class definition.

```
PHP_MINIT_FUNCTION(test) {
    zend_class_entry my_class_entry;

    INIT_CLASS_ENTRY(my_class_entry, "my_class", php_my_class_functions);
    my_class_entry_ptr = zend_register_internal_class(&my_class_entry
TSRMLS_CC);
    return SUCCESS;
}
```

And then we create our object factory function which will return an instantiated object.

```
PHP_FUNCTION(my_object) {
{
    char *fn = NULL;
    int argc = ZEND_NUM_ARGS();
    int fn_len;
    long length;
    double price;
    zend_bool setting;

    if (zend_parse_parameters(argc TSRMLS_CC,
                              "sbld", &fn, &fn_len,
                &setting, &length, &price) == FAILURE)
        return;

    object_init_ex(return_value, my_class_entry_ptr);

    add_property_stringl(return_value,
                         "filename", fn, fn_len, 1);
    add_property_bool   (return_value,
                         "toggle", setting ? 0 : 1);
    add_property_long   (return_value,
                         "length", length);
    add_property_double (return_value,
                         "price", price);
}
```

From user space it would be called like this:

```
<?php
  $obj = my_object();
  $obj->add(...);
?>
```

If you want to have the more traditional class instantiation syntax and want to have a constructor run for it. eg.

```
<?php
  $obj = new my_class();
?>
```

Then create a constructor function:

```
PHP_FUNCTION(my_class) {
    char *fn = NULL;
    int argc = ZEND_NUM_ARGS();
    int fn_len;
    long length;
    double price;
    zend_bool setting;

    if (zend_parse_parameters(argc TSRMLS_CC,
                              "sbld", &fn, &fn_len,
              &setting, &length, &price) == FAILURE)
        return;

    add_property_stringl(this_ptr, "filename", fn, fn_len, 1);
    add_property_bool(this_ptr, "toggle", setting?0:1);
    add_property_long(this_ptr, "length", length);
    add_property_double(this_ptr, "price", price);
}
```

Note the use of this_ptr here.

If you want to access properties from within one of the methods, you can do this:

```
pval **tmp;
if(zend_hash_find(HASH_OF(this_ptr),
                  "my_property", 11, (void **)&tmp) == SUCCESS) {
    convert_to_string_ex(tmp);
    php_printf("my_property is set to %s\n", Z_STRVAL_PP(tmp));
}
```

Global Variables

If you need extension-wide global variables, you can't just use standard C-style static variables because PHP needs to be thread-safe. To achieve this, globals are placed in a struct and passed around. To add global variables to your extension, first define them in your header file:

```
ZEND_BEGIN_MODULE_GLOBALS(test)
    int   some_integer;
    char *some_string;
ZEND_END_MODULE_GLOBALS(test)

/ shortcut macros /
#ifdef ZTS
# define TEST_G(v) TSRMG(test_globals_id, zend_test_globals *, v)
#else
# define TEST_G(v) (test_globals.v)
#endif
```

Then in your .c file:

```
ZEND_DECLARE_MODULE_GLOBALS(foo)

static void php_foo_init_globals(zend_foo_globals *foo_globals)
{
    foo_globals->some_integer = 0;
    foo_globals->some_string = NULL;
}
```

And in your MINIT function:

```
ZEND_INIT_MODULE_GLOBALS(foo, php_foo_init_globals, NULL);
```

Now, anywhere you need to get/set one of these variables, use TEST_G(some_integer). This will only work if the test_globals struct is available in the function, of course. All the standard PHP functions will automatically have the globals struct available, but if you write your own utility functions you either have to call TSRMLS_FETCH(); at the top of the function, or better yet, pass in the TSRM struct. Like this:

```
test_utility_function(my_arg TSRMLS_CC);
```

And the function is declared using:

```
static void test_utility_function(int my_arg TSRMLS_DC)
```

SAPI Globals

SAPI is the Server abstraction API and you can access server-related global variables using the SG() macro. Include SAPI.h in your .c file to get the SG macro. You can then access the elements of the sapi_globals_struct which looks like this:

```
typedef struct _sapi_globals_struct {
    void *server_context;
    sapi_request_info request_info;
    sapi_headers_struct sapi_headers;
    int read_post_bytes;
    unsigned char headers_sent;
    struct stat global_stat;
    char *default_mimetype;
    char *default_charset;
    HashTable *rfc1867_uploaded_files;
    long post_max_size;
    int options;
} sapi_globals_struct;
```

To access the default MIME type, you would use:

```
SG(default_mimetype)
```

And to access the request_uri you would use:

```
SG(request_info).request_uri
```

Executor Globals

These are run-time globals. The ones you are likely to be interested in are the symbol_table and active_symbol_table globals. For example, to access the user-space $foo variable from the current global context, you would do this:

```
pval **tmp;
if(zend_hash_find(&EG(symbol_table),
                  "foo", 4, (void **)&tmp) == SUCCESS) {
  RETURN_STRINGL(Z_STRVAL_PP(tmp), Z_STRLEN_PP(tmp), 1);
} else {
  RETURN_FALSE;
}
```

Adding php.ini Entries

Generally the extension-wide global variables are actually configuration options set in the php.ini file, or perhaps in the Apache httpd.conf file. Adding such configuration options for your extension is easy. To add a test.my_ini_setting configuration variable, first follow the instructions for creating a global variable from the last slide. Then in your .c file add:

```
PHP_INI_BEGIN()
    STD_PHP_INI_ENTRY("test.some_integer", "0",
                      PHP_INI_ALL, OnUpdateInt,
              some_integer, zend_test_globals, test_globals)
PHP_INI_END()
```

The arguments to STD_PHP_INI_ENTRY here are:

- entry name
- entry value
- change permissions
- pointer to change notification handler
- corresponding global variable
- globals struct type
- globals struct

You can implement your own change notification handlers using PHP_INI_MH(), but here we will just use the default OnUpdateInt which basically just takes the string in the .ini file and does an atoi() on it. The other built-in handlers are OnUpdateReal(), OnUpdateBool(), OnUpdateString(), and OnUpdateStringUnempty().

After adding your STD_PHP_INI_ENTRY block to your .c file, in your MINIT function after the ZEND_INIT_MODULE_GLOBALS() add this:

```
REGISTER_INI_ENTRIES();
```

And likewise, in your MSHUTDOWN function add:

```
UNREGISTER_INI_ENTRIES();
```

And finally, you can optionally add this to your MINFO function:

```
DISPLAY_INI_ENTRIES();
```

Memory Management

PHP has its own memory management functions which adds a safety net in case you forget to free things. It also helps catch overflows and other memory-related issues. The functions are exactly like their typical C counterparts:

```
emalloc
efree
estrdup
estrndup
ecalloc
erealloc
```

Resources

A generic data container. Used to hold file handles, database connections, or anything else that needs to be initialized and then passed around for other functions to act on. To create a resource in your extension, first declare an extension-wide true global. We don't have to worry about thread-safety here. And then if your resource holds a complex datatype of some sort, typedef it as well at the top of your .c file (on in your header file):

```
static int le_test;

typedef struct _test_le_struct {
    char *name;
    long age;
} test_le_struct;
```

Next we need to create a resource destructor function.

```
static void _php_free_test(zend_rsrc_list_entry *rsrc TSRMLS_DC) {
    test_le_struct test_struct = (test_le_struct )rsrc->ptr;

    efree(test_struct->name);
    efree(test_struct);
}
```

Then in our MINIT function we declare/initialize our resource:

```
le_test = zend_register_list_destructors_ex(_php_free_test,
                                            NULL, "test", module_number);
```

Then we might have a function that looks like this that returns a resource:

```
PHP_FUNCTION(my_init) {
    char *name = NULL;
    int name_len, age;
    test_le_struct *test_struct;

    if (zend_parse_parameters(ZEND_NUM_ARGS() TSRMLS_CC,
                              "sl", &name, &name_len, &age) == FAILURE) {
        return;
    }
    test_struct = emalloc(sizeof(test_le_struct));
    test_struct->name = estrndup(name, name_len);
    test_struct->age = age;
    ZEND_REGISTER_RESOURCE(return_value, test_struct, le_test);
}
```

Followed by other functions that then take a resource as an argument and do something with it:

```
PHP_FUNCTION(my_get)
{
    test_le_struct *test_struct;
    pval *res;

    if (zend_parse_parameters(ZEND_NUM_ARGS() TSRMLS_CC,
                              "r", &res) == FAILURE) {
        return;
    }

    ZEND_FETCH_RESOURCE(test_struct, test_le_struct *,
                        &res, -1, "test", le_test);

    if(!test_struct) RETURN_FALSE;

    array_init(return_value);
    add_assoc_string(return_value, "name", test_struct->name, 1);
    add_assoc_long(return_value, "age", test_struct->age);
}
```

PHP5 TODO

Zend Engine 2

- Important stuff to finish: PPP members/PPP methods, support of overloaded extensions, possibly differentiate between class and namespace as discussed with Stig in Germany.
- Responsibility: Zeev, Andi, Stas
- Timeframe: Couple of months for first part and extensions will be ongoing while PHP is being fixed. Things to leave for later versions: Native aggregation support, accessing static members via object and not class.

Aggregation

- Responsibility: Andi, Stig?
- Timeframe: ?

Redesign of API Versioning

- Responsibility: ?
- Timeframe: 1 month

Thread Safety

- Identify the extensions that are not thread safe by design or due to dependant libraries and identify them as such. If possible try to resolve thread safety issue via code improvements (if php code or patches will be accepted by library maintainers). For situations where thread safety cannot easily be acheived a flag in the extension API is set so PHP can identify non-thread safe extensions. These extensions will not be loaded in a ZTS compiled binary (unless it is cli/cgi).
- Responsibility: Spiderman or someone similar with superhuman powers
- Timeframe: ?

SAPI

- Environment variables defined in the CGI spec need to be verified in each SAPI module that they conform to the CGI spec correctly. If they do not, the SAPI module needs to fix the variable prior to script execution. Having this conformity will aid in having PHP scripts run correctly under different sapi modules.
- Responsibility: Shane Caraveo & each sapi module owner
- Timeframe: ? (but shouldn't be much effort, most modules are probably ok)

Input Filtering

- Implement a SAPI input filter hook that will get called just before registering a variable in the treat_data/post_handler hooks.
- Make sure this is also done in mbstring
- Provide access functions, or perhaps a new $_RAW_GET/POST/Cookie set of superglobals to get at the unfiltered data
- Provide a .ini directive which allows people to set their input filter to one of the built-in strip_tags, htmlspecialchars or whatever other internal function might be useful here.
- (The main benefit of this is to make it easier for people to solve the XSS problem once and for all without having to go through every line of their code and adding input validation/filtering everywhere)
- Responsibility: Rasmus

- Timeframe: Yesterday

RPC Abstraction Layer

- Porting java, com, dotnet, xmlrpc, corba, soap and python, srm (are there more ?) to work with the new oo api and preferably by using ext/rpc.
- Responsibility: Harald
- Time frame: 2 months (but i have to wait for a few engine features first)

OO Extensions

- Each OO extension has to be revised and rewritten to fit into the new OO model. We should decide which extensions are a must to have for the release and which can be ported by the maintainer later as a separate pecl release.
- A list of extensions to be extended that have to be investigated:
- * browscap
- * aggregate
- all sql extensions (*_fetch_object)
- * domxml (seems like christian is rewriting it anyways)
- * ming
- Responsibility: Harald (, extension maintainers)

MySQL Extension

- Complete rewrite, leveraging the new MySQL 4 / MySQL 5 features.
- Responsibility: Georg Richter, Zak Greant
- Timeframe: ?

XML

- Rewrite DOMXML and incorporate all (or most of) W3C-DOM2.
- Use the new ZE2 features (Exceptions, setter/getter).
- Add SAX(2), XML Schema.
- XSLT, HTML, XPath, XPointer, DTD Validation will still be supported, have to find a meaningful API for it.
- Break BC, warn users now.
- Look at the libxml2 patch by lukas schröder and see if we can prevent memory leaks with it (anyway, getting rid of mem-leaks and intelligent memory management is on top prio...)
- In the longer term, domxml (or another name, as with todays features domxml is a little bit misleading) shall be the main xml-class, which covers most of what's needed for decent XML support in PHP ;) But there is certainly place for others like Sablotron etc.
- Responsibility: Christian Stocker
- Timeframe: ?

Test Suite

- Extending the test suite with atleast a test for every function in an extension that doesn't require external resources. Also developing an automated test thing which cvs ups's, compiles and tests the build on a daily base on as much platforms/extensions as possible.
- The test suite will also be extended to support threaded testing and testing for differing sapi modules (via http calls or other methods).
- Reponsibility: Derick (, extension maintainers)
- Timeframe: 3 months

Index